МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЯВГУСІШИН БОГДАН АНАТОЛІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2025 р.

РЕКОМЕНДАЦІЙНА СИСТЕМА ДЛЯ ВИЯВЛЕННЯ НЕСПРАВНОСТЕЙ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Сергій ШТОВБА, професор кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2025

АНОТАЦІЯ

Явгусішин Б.А. Рекомендаційна система для виявлення несправностей у програмному забезпеченні персонального комп'ютера. Спеціальність 122 «Комп’ютерні науки», Освітня програма «Комп’ютерні технології обробки даних». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У дипломній роботі досліджено процес створення та розроблено інтелектуальну рекомендаційну систему для виявлення та діагностики несправностей персональних комп’ютерів. Проаналізовано сучасні підходи до побудови систем діагностики, включно з методами машинного навчання, байєсівськими методами та інтеграцією з великими мовними моделями. Розглянуто архітектуру застосунку, методи збору та оброки інформації, а також організацію баз даних.

Ключові слова: C#, .NET MAUI, Entity Framework Core, PostgreSQL, рекомендаційна система, машинне навчання, діагностика персонального комп’ютера.

70 с., 20 рис., 1 табл., 59 джерел.

ABSTRACT

Yavhusishyn B.A. A Recommender System for Detecting Faults in Personal Computer Software. Specialty 122 «Computer Science», Programme «Computer technologies for data processing». Vasyl Stus Donetsk National University, Vinnytsia, 2025.

The thesis explores the process of creating and developing an intelligent recommendation system for detecting and diagnosing malfunctions of personal computers. Modern approaches to building diagnostic systems are analyzed, including machine learning methods, Bayesian methods, and integration with large language models. The application architecture, methods of collecting and processing information, and database organization are considered.

Keywords: C#, .NET MAUI, Entity Framework Core, PostgreSQL, recommendation system, machine learning, personal computer diagnostics.

70 p., 20 fig., 1 table, 59 sources.

ЗМІСТ

ВСТУП

Актуальність роботи:

З розвитком комп'ютерних технологій та зростанням кількості користувачів персональних комп'ютерів постає проблема ефективної діагностики програмних та апаратних несправностей. Сучасні системи технічної підтримки часто потребують значних часових і людських ресурсів, тоді як автоматизовані рішення здатні зменшити навантаження на фахівців і підвищити швидкість обслуговування.

Особливої актуальності набувають інтелектуальні рекомендаційні системи, здатні аналізувати вхідні дані користувача, визначати ймовірні причини несправностей і пропонувати персоналізовані рішення. Інтеграція таких систем із великими мовними моделями та методами машинного навчання відкриває нові можливості для побудови більш адаптивних і точних інструментів діагностики.

Застосування рекомендаційних систем, байєсівських моделей, нейронних мереж дозволяє автоматизувати процес виявлення несправностей та створити гнучку систему підтримки користувачів. Це особливо важливо для малого бізнесу, навчальних закладів та користувачів, які не мають доступу до повноцінного технічного сервісу та/або достатнього рівня технічних знань.

Мета дослідження: Розробка інтелектуальної рекомендаційної системи для виявлення та діагностики несправностей персональних комп'ютерів із використанням технологій C#, .NET MAUI, ML.NET та інтеграцій з мовними моделями.

Завдання дослідження:

- Проаналізувати сучасні методи діагностики та рекомендаційних систем;
- Розробити архітектуру багато проєктного рішення на основі MAUI;
- Спроєктувати структуру бази даних для зберігання інформації про симптоми, несправності, рекомендації та результати діагностики користувачів;

- Реалізувати базовий прототип системи з інтерактивним опитуванням і формуванням рекомендацій;
- Інтегрувати модуль машинного навчання для підвищення точності діагностики.

Об’єкт дослідження: Процес автоматизованої діагностики програмних несправностей персональних комп’ютерів на основі аналізу наявних симптомів.

Предмет дослідження: Методи побудови та впровадження рекомендаційних систем діагностики, що базуються на принципах аналізу даних, машинного навчання і інтеграції з великими мовними моделями.

Наукова новизна дослідження: У роботі запропоновано модульну архітектуру рекомендаційної системи діагностики несправностей персонального комп’ютера, де у автономні підсистеми виділено адаптивний опитувальник, рівень правил, ймовірнісний модуль, модуль машинного навчання та інтеграція з великими мовними моделями. Такий підхід забезпечує поєднання експертних знань, ймовірнісних оцінок і статистичного аналізу у єдиному діагностичному процесі та дозволяє формувати зрозумілі рекомендації для користувачів, що не мають достатніх технічних знань. У роботі розроблено механізм формування вектора симптомів за допомогою поєднання відповідей користувача та автоматичного аналізу технічних показників, а також використано розділену структуру бази даних, що дозволяє гнучко оновлювати знання та накопичувати реальні діагностичні сценарії.

Практичне значення дослідження: Створення прототипу інтелектуальної рекомендаційної системи, що дозволяє користувачам без спеціальної технічної діагностики підготовки самостійно виявляти та усувати програмні несправності персонального комп’ютера на основі отриманих рекомендацій. Розроблене рішення може бути інтегроване як окремий модуль у сервіси технічної підтримки, освітні заклади, корпоративні інструменти або використовуватися як самостійний застосунок для кінцевих користувачів.

Апробація результатів дослідження: результати досліджень апробовані на IV Міжнародної науково-практичної конференції, м. Вінниця, 05 листопада

2025 р. під назвою «Проєктування рекомендаційної системи для діагностики несправностей персональних комп'ютерів», VI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених, м. Вінниця, 8 грудня 2025 р. під назвою «Інтелектуальна рекомендаційна система діагностики програмних несправностей персональних комп'ютерів: архітектура, алгоритми та реалізація».

Структура кваліфікаційної роботи: Магістерська робота складається із вступу, трьох розділів та висновків та списку використаних джерел.

Робота містить 70 сторінок, 20 рисунків, , таблицю та список літератури з 59 джерел.

РОЗДІЛ 1. ОГЛЯД СУЧАСНОГО СТАНУ ТА АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНОГО СТАНУ ПИТАННЯ

1.1 Проблематика виявлення несправностей у персональних комп'ютерах

Перші комерційні персональні комп'ютери почали поширюватися на початку 1970-х років, а саме: Altair 8800; Apple I; IBM PC.

Хоча дані пристрої мали обмежену продуктивність, але вони заклали основу сучасної комп'ютерної індустрії. Комп'ютери стали масовим інструменти для роботи, навчання та розваг з появою операційних систем для персональних комп'ютерів. Представниками є: MS-DOS; Mac OS; Windows.

Водночас такі фактори як масовість використання персональних комп'ютерів знизило середній рівень технічних знань серед користувачів так і ускладнення апаратного та програмного забезпечення призвело до появи нових типів несправностей, які користувач вже не змозі самостійно вирішити. Наприклад, з розвитком операційних систем, графічних інтерфейсів, драйверів і мережевих технологій з'явилися численні програмні збої, які могли проявлятися у форматі конфліктів між компонентами, нестабільностями роботи комп'ютера і тд. Хоча до цього через високий поріг входу, як технічних знань, так і фінансової доступності, та малу кількість альтернатив у програмному та апаратному забезпечені несправності могли бути виправлені самим користувачем.

Для вирішення несправностей та їх випередженню були створенні діагностичні програми. Одними з перших були створені: Norton Utilities; PC Doctor; QAPlus [1, 2]. Прикладі інтерфейсу програми QAPlus наведено на рисунку 1.1.

Сучасні персональні комп'ютери є складними системами, де апаратна (процесор, оперативну пам'ять, накопичувачі, живлення і тд), операційна та програмна складові тісно інтегровані між собою. Через це несправності можуть виникати з різних причин, часто у комбінації, і інколи лише за певних умов. Наприклад, дефект накопичувача або перегрів можуть виглядати як «зависання»

застосунків, тобто виникли в одному шарі, а проявилися як симптом у зовсім іншому.

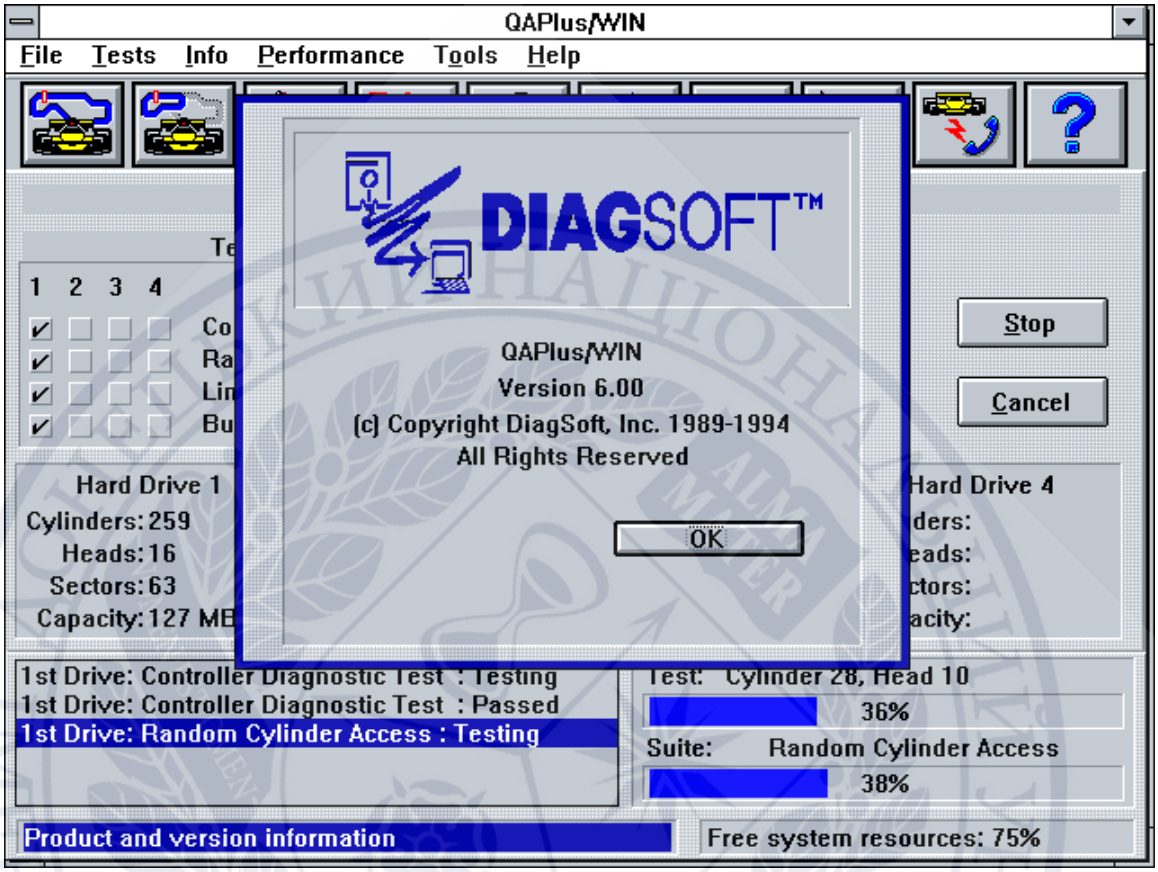

Рисунок 1.1 – Інтерфейс застосунку QAPlus 6.x [2]

Ключова проблема – визначити причину в умовах неоднозначності та обмеженої інформації. Виявлення причин появи несправностей є критично важливим завданням. Основні виклики в діагностиці та ідентифікації джерела проблеми:

- Непередбачуваність та не постійність (інтермітентність). Деякі несправності проявляються лише за певних умов. Наприклад, під певним навантаженням або через певний інтервал часу;
- Можливість множини факторів. Симптоми несправності можуть бути результатом декількох причин одночасно;
- Різноманітність апаратних конфігурацій. Персональні комп’ютери та мають безліч моделей, виробників, варіантів комплектуючих. Те, що для

одного пристрою є симптомом, для іншого може бути нормальним режимом;

- Обмеженість доступної інформації. Деякі компоненти, особливо сторонні драйвери та/або пристрої, можуть не надавати детальних діагностичних даних або журналів. Часто діагностика обмежується лише загальними кодами помилок;
- Шум/помилкові сигнали. Якщо брати за основу системні журнали подій, в яких сучасні операційні системи накопичують велику кількість даних, то доволі часто вони містять тисячі записів, багато з яких є незначущими або «шумом». При невеликих або нечітких симптомах важко знайти релевантні події.

Так як не кожен пересічний користувач персонального комп'ютера має достатні технічні знання, щоб самостійно вирішити несправності, то він може спочатку пошукати рішення в інтернеті на різних сайтах, форумах, звернутися до штучного інтелекту і якщо нічого не допомогло звернутися до сервісного центру.

Це призводить до необхідності створення інтелектуальних систем, які б могли допомогти користувачу і діагностиці без глибоких технічних знань – через покрокові питання, аналіз симптомів та пропозицію можливих рішень. Таким чином, актуальним завданням є розробка рекомендаційної системи діагностики, яка може поєднувати знання експертів, аналітику журналів системи і машинне навчання для підвищення точності визначення несправностей.

Метою такої системи полягає не лише у виявленні поточної несправності, але й у зборі структурованих даних для аналізу типових сценаріїв, що дає змогу передбачати виникнення несправностей у майбутньому [3].

Основні обмеження даної системи:

- Баланс витрату ресурсів на діагностику. При діагностиці можливе суттєве навантаження системи;
- Недостатність технічних знань у користувачів. Може виражатися у хибно наданих відповідях, що може призвести до неправильних рекомендацій;

- Можлива відсутність коректної відповіді на новий вид несправностей.

1.2 Класифікація несправностей

Несправності персональних комп'ютерів умовно поділяють на апаратні, програмні, операційної системи. Такий поділ дає змогу систематизувати причини збоїв і полегшує процес діагностики.

До апаратних відносять усі фізичні проблеми компонентів системи: процесора, оперативної пам'яті, жорсткого диска, системи живлення, материнської плати чи охолодження. Типові приклади апаратних несправностей:

- Перегрів процесора або GPU, спричинений деградацією термопасти або забруднення радіатору та вентилятору пилом;
- Проблеми з оперативною пам'яттю (RAM). Можуть викликати випадкові сині екрани смерті (BSOD). Для діагностики використовують утиліти Windows Memory Diagnostic чи memtest86+;
- Пошкодження жорсткого диску або SSD. Для моніторингу стану накопичувачів застосовують технологію S.M.A.R.T., яка зчитується засобами Windows(vmic diskdrive get status) або утилітами CrystalDiskInfo, smartmontools, тощо;
- Збої блоку живлення (PSU), що можуть бути викликані спонтанними навантаженням електричної напруги.

Апаратні збої найчастіше проявляються у вигляді нестабільної роботи системи, артефактів на екран, шумів або відмови у ввімкненні. Для діагностики таких проблем зазвичай потрібні вимірювальні прилади або тестування компонентів поза системою.

В свою чергу до несправностей операційної системи відносять проблеми ядра, драйверів і служб, які безпосередньо забезпечують роботу апаратного забезпечення. Наприклад:

- Пошкодження системних файлів ядра або драйверів;
- Невдале оновлення операційної системи або драйверів;

- Помилки файлової системи (NTFS, ext4, APFS) через некоректне завершення роботи або збій живлення;
- Порушення реєстру Windows, що впливає на роботу служб.

Для діагностики таких несправностей використовують:

- Windows Event Viewer – перегляд критичних подій системи;
- System File Checker (sfc /scannow) і DISM – для відновлення системних файлів;
- Linux journalctl – перегляд логів ядра та системних служб.

Програмні несправності відносяться до прикладного рівня. Вони зазвичай встановлюються самим користувачем. Типові проблеми:

- Помилки під час виконання;
- Конфлікти між версіями бібліотек;
- Шкідливе програмне забезпечення або сторонні утиліти, що змінюють системні параметри;
- Надмірне споживання ресурсів системи. Наприклад, процесору або відеокарти.

Під час аналізу таких несправностей часто використовуються журнали додатків, якщо наявні або моніторинг ресурсів. Наприклад, Task Manager, команда top або Activity Monitor на девайсах з операційною системою Mac.

Класифікація несправностей є важливою складовою для створення коректної рекомендаційної системи, так як описані характерні симптоми кожного типу несправностей будуть у подальшому використанні під час формування бази знань або навчального набору даних.

1.3 Методи діагностики

Перші системи діагностики виникли разом з появою складних технічних пристроїв, коли виникла потреба визначати причини несправностей без повного розбирання чи ручного тестування. На початкових етапах, діагностика здійснювалася вручну, переважно інженери спиралися на власний досвід, журнали несправностей та прості контрольні тести.

З розвитком обчислювальної техніки в 1960-х – 1970-х роках з’явилися перші автоматизовані діагностичні програми. Вони базувалися на таблицях симптомів і можливих причинах, які заповнювалися експертами. Дані програми були попередниками експертних систем.

Наприклад, DENDRAL, що була розроблена у Стенфордському університеті і є першою в світі програмою експертною системою. Логіка роботи базувалася на підготовлених завчасно даних [4]. DENDRAL була першою системою, що мала окрему базу знань, яку можна було редагувати чи змінювати для нових завдань, але при цьому зберігаючи кодову базу не зміненою [4]. Також була з перших систем, яка була застосована до «реальної проблеми» [4]. Основною ціллю системи була ідентифікація невідомих молекул для допомоги дослідниками органічної хімії.

Також в Стенфордському університеті була розроблена інша система MYCIN. Була створена на початку 1970-х роках для діагностики бактеріальних інфекцій і постановки діагнозу [5]. Дана система використовувала правило «якщо-то», які зберігалися в базі знань. Наприклад, якщо після здачі аналізу крові було виявлено наявність якихось бактерій, то можлива така інфекція. Для введення даних користувач мав використовувати шаблони, так як система не могла «розуміти» природну мову [5].

Що ж до діагностичних програм пов’язаних з несправностями комп’ютерів, то варто згадати DART та SOPHIE. Перша було розроблена IBM і була побудована за допомогою HMYCIN. Експертна система DART була створена для пришвидшення ремонту тих чи інших компонентів системи, де інженер, який немає всіх технічних знань, міг без залучення відповідного експерту знайти осередок несправностей і виправити його, а не тільки - симптоми [6]. Робота над системою SOPHIE почалася на початку 1973 року в університеті Каліфорнія і була закінчена в 1977 році в Массачусетс [7]. SOPHIE використовувалася для локалізації несправностей в аналогових електронних схемах [7]. Дана система мала наступні модельно-орієнтовані техніки для діагностування несправностей:

- Використання моделей поведінки компонентів для проведення логічних висновків при порівнянні даних з реальними спостереженнями;
- Компонент вважається несправним, якщо після його видалення несправність зникає;
- Використовується інформація про типові режими відмов для виключення деталей, які не можуть бути причиною несправності;
- Модельний підхід поєднується з правилами логічної діагностики. Це дозволяє системі переходити від аналітичного аналізу до правил, коли даних недостатньо.

Якщо розглядати діагностику в сфері несправностей персональних комп'ютерів, то можна виділити наступні підходи, що застосовуються на практиці:

- Rule-based підхід. Базується на попередньо визначених експертних правилах, що описують взаємозв'язки між симптомами та можливими причинами несправностей. Кожне правило є формату «якщо-то». Наприклад, якщо комп'ютер перезавантажується без попередження і температура головного процесу перевищує 98 градусів Цельсія, то можлива несправність системи охолодження. Системи з таким підходом мають високу інтерпретованість, є легкі в налаштуванні та підходять для прототипів рекомендаційних систем, але на противагу перевагам їх складно масштабувати, так як для додавання нових правил потребується перевірка створених до цього;
- Ймовірнісні методи. Зокрема, Байєсівські мережі, дозволяють оцінювати вірогідність кожної можливої несправності з урахуванням наявних симптомів. Дана мережа будує залежності між симптомами та несправностями у вигляді графа. З переваг варто виділити ефективність при неповних або неточних даних, що може бути результатом опису симптомів користувачем у форматі довільного тексту. Водночас є два недоліка, а саме складність у правильності опису залежностей між усіма

симптомами і причинами та потреба у початкових ймовірностях подій, що на момент прототипу та початкових етапах розробки відсутні;

- Методи машинного навчання. Застосовуються для автоматизації процесу діагностики шляхом навчання на основі датасету для навчання з симптомами та результатами перевірок і подальшою перевіркою на контрольному датасеті з даними, що були відсутні у датасеті навчання. Типовими представниками алгоритмів є:
    1. Логістична регресія;
    2. Дерева рішень;
    3. Нейронні мережі.

  Основним недоліком, як і ймовірнісних методів, є потреба у достатньому розмірі та якості вибірки для надання точних результатів;

1.4 Рекомендаційні системи

Рекомендаційні системи представляють собою інтелектуальні системи для аналізу інформації та надання користувачу персоналізованих порад або пропозицій [8]. Зазвичай результат системи базується на заготовлених попередніх даних, схожих випадках або на контексті де використовуються.

Однією з перших цифрових реалізацій рекомендаційних систем є Tapestry, що була розроблена у 1992 році в дослідницькому центрі Xerox PARC, нині PARC [9]. Дана система була експериментальною поштовою системою, що використовувала колаборативну фільтрацію, де користувачі за допомогою анотацій або оцінок могли позначити цікаві для них електронні листи і на основі цього система надавала можливість фільтрувати листи.

Надихнувшись системою Tapestry, дослідники з інститутів Массачусетса та Міннесота розробили рекомендаційну систему новин GroupLens, яка прибрала недолік першої, а саме очікування, що користувач може знати велику кількість людей і самостійно обирати фільтр. Дана система брала вподобання користувача, робила на їх основі прогноз, що може сподобатися і пропонувала результати.

Починаючи від перших рекомендаційних систем і закінчуючи сучасними їх представниками можна поділити на кілька класичних підходів:

- Колаборативна фільтрація. Основна ідея у використанні інформації про попередню поведінку користувачів або їхні оцінки, щоб спрогнозувати, які об’єкти можуть зацікавити користувача [10]. Широко використовуються у сфері онлайн торгівлі;
- Контентно-орієнтованість. Формує рекомендації на основі характеристик об’єктів і індивідуальні вподобання користувача. Наприклад, користувачу сподобалась книжка з жанром фантастики, то система запропонує книжки, що також мають жанр фантастика. На противагу колаборативного підходу не потребує великої кількості користувачів чи оцінок від них;
- Рекомендаційні системи, що залежать від знань. Підхід використовується, коли колаборативний та контентно-орієнтований є малоефективними [8]. Наприклад, користувач не дуже часто купує або шукає щось та переважно не залишає оцінок. Рекомендації формуються спираючись на вимоги користувача та базу знань з властивостями об’єктів;
- Гібридний підхід. Здебільшого поєднання вище згаданих підходів з метою усунення їхніх недоліків [10]. Даний підхід дозволяє поєднати сильні сторони кожного підходу, забезпечуючи більш точні результати.

Наразі в сучасному світі рекомендаційні системи стали невід’ємною частиною цифрового простору. Їх використовуються у багатьох сферах:

- Онлайн торгівля: Amazon, eBay, Rozetka;
- Медіа платформи: YouTube, Netflix, Spotify;
- Освітні сервіси: Coursera, Khan Academy.

У контексті виявлення несправностей персональних комп’ютерів рекомендаційна система може виступати як аналітичний модуль, що на основі несправностей, симптомах і користувацьких діях рекомендує найбільш ймовірні рішення для усунення проблеми. На противагу традиційним діагностичним інструментам рекомендаційна система може знаходити закономірності у даних в

журналах подій, зіставляти з випадками несправностей у базі даних і на основі цього формувати потенційні рішення.

1.5 Застосування великих мовних моделей у рекомендаційних системах

У середині XX століття були перші спроби автоматизувати обробку природньої мови, що можна вважати початком розвитку великих мовних моделей (Large Language Models, LLM). На ранніх етапах переважали символічні та правило-орієнтовані підходи, що базувалися на граматиці, лексичних базах і шаблонних відповідях. Відомим прикладом є система ELIZA, яка була розроблена вченим Массачусетського технологічного університету Джозефом Вейценбаумому 1960-х роках [11]. Вона вела діалог із користувачем за допомогою граматичного аналізу з отриманням ключових слів з речень та підстановкою шаблонних фраз. У 1980-1990-х роках перейшли до використання статистичних методів, де головну роль відігравали n-грамні моделі, що оцінювали ймовірність появи слова залежно від попередніх [12].

Прорив у галузі обробки тексту відбувся після появи нейронних мереж і моделей типу: RNN; LSTM; GRU; Transformer [13, 14].

Згадані вище моделі дали змогу враховувати контекст у текстах великого розміру. Ще одним важливим етапом стало використання векторних представлень слів. Наприклад, Word2Vec розроблений інженерами в Google у 2013 році або GloVe розроблений в Стенфордському університеті у 2014 році. Дані вектори дозволяють описати слова у багатовимірному просторі, де слова близькі за змістом розташовуються поруч. У 2017 році було запропонована архітектура Transformer, яка має механізм self-attention, що забезпечує ефективне паралельне навчання й масштабування моделей [15]. Саме на цій архітектурі з’явилися системи, як BERT, GPT-2, T5, RoBERTa, що за започаткували нову еру у світі мовних моделей.

Починаючи з 2020 року, розвиток мовних моделей набув стрімких темпів. Вихід GPT-3 з 175 мільярдами параметрів показав, що модель може виконувати

складні мовні завдання без додаткового навчання завдяки режимів без (zero-shot) або з дуже малою (few-shot) кількістю прикладів. Завдяки підходу навчання з використанням зворотного зв'язку від користувача, у 2022 році з'явився ChatGPT, що зробило використання великих мовних моделей доступним для широкого загалу користувачів.

Наразі мовні моделі на кшталт GPT-4, Claude, Gemini, можуть не тільки працювати з текстом, а й аналізувати зображення та аудіо, створювати зображення по запиту, робити запиту в інтернет та отримувати інформацію з електронних джерел і тд [16, 17]. Даний функціонал демонструє універсальність використання великих мовних моделей у широкому спектрі завдань. У випадку даної кваліфікаційної роботи у побудові інтелектуальних рекомендаційних систем.

Великі мовні моделі мають значну перевагу над традиційними системами рекомендацій. На відміну класичним алгоритмам, які працюють виключно зі структурованими даними та вимагають чітко сформульованих запитів для коректних результатів, мовні моделі ефективно обробляють неструктуровану текстову інформацію, яка може бути отримана від користувача у форматі діалогу з моделлю.

Якщо брати до уваги контекст рекомендаційної системи для виявлення несправностей в персональному комп'ютері, то мовна модель може виступати не лише як інтерфейс для отримання додаткових даних, а й як аналітичний компонент, що бере участь у прийнятті рішення. Також вона може об'єднати результати роботи інших модулів системи, узагальнивши їх і надавши кінцеву рекомендацію в зрозумілому текстовому форматі. Іншими прикладними використаннями мовних моделей в рекомендаційній системі є:

- Аналіз симптомів несправностей описаних користувачем у довільній формі;
- Інтерпретація журналів подій, де виділяються потенційно ключові записи відносно несправності;

- Роз’яснення результатів потенційного рішення несправності на рівні спілкування зрозумілому для користувача. Як простою мовою, якщо у користувача мало технічних знань, так і з додатковими деталями для більш досвідчених.

Попри численні переваги, мовні моделі мають й недоліки. З них можна виділити:

- Нестабільність відповідей. Відповіді можуть змінюватися залежно від формулювання запиту або контексту;
- Відсутність гарантії достовірності відповідей або так зване явище «галюцинацій». Коли в моделі відсутня конкретна відповідь, то замість того, щоб видати результат, що він не знає, видає помилковий результат. Причиною є те, що він «вважає», що краще видати хоч якусь відповідь, ніж нічого [18];
- Значні обчислювальні витрати. Для використання персональних або локальних моделей потрібні значні ресурси обчислювальних машин, а якщо використовувати сторонні сервіси, то зазвичай користувач має оплати використання у форматі підписки на якийсь період часу для необмеженого доступу або використовувати безкоштовно, але з обмеженням на кількість запитів і можливо на якість відповідей.

1.6 Аналіз існуючих рішень

У галузі діагностики персональних комп’ютерів для виявлення несправностей існує низка програмних продуктів, які збирають технічні показники, аналіз стану системи та формування рекомендацій для користувача. Найбільш відомими серед них є AIDA64, HWiNFO, TechSuite, PC-Doctor Service Center та CrystalDiskInfo. Кожне з цих рішень має власну спеціалізацію та функціональні можливості.

AIDA64 є одним з найпоширеніших інструментів для комплексного аналізу апаратного забезпечення. Програма надає детальну інформацію про процесор, пам’ять, відеокарту, материнську плату та програмне забезпечення.

Однією з ключових переваг є можливість стрес тестування системи, що дозволяє виявити перегрівання, нестабільність або апаратні дефекти під навантаженням [19]. Хоча AIDA64 забезпечує глибоку технічну діагностику і надає можливість моніторингу стану системи, застосунок орієнтований насамперед на користувачів, які володіють достатнім рівнем технічних знань. HWiNFO є універсальним рішенням для аналізу та моніторингу обладнання, що підтримує широкий спектр операційних систем [20]. Застосунок надає детальний перегляд характеристик комп'ютера. Основною перевагою є фокус на оперативному відстеженні роботи компонентів та мінімальному навантаженні на ресурси. TechSuite є багато функціональним інструментом, що дозволяє автоматизувати типові операції з персональними комп'ютерами [21]. Наприклад, діагностика апаратного забезпечення, усунення шкідливого програмного забезпечення, тощо. Орієнтований на людей з відносно високим рівнем технічних знань. PC-Doctor Service Center це професійний комплект діагностичних інструментів для апаратного забезпечення, який підтримує Windows, macOS, Android. Він включає сотні тестів, можливість завантаження з USB, створення локалізованих звітів і тд [22]. CrystalDiskInfo безкоштовна утиліта для моніторинку стану жорстких дисків, твердотільних накопичувачів та деяких зовнішніх дисків через систему S.M.A.R.T [23].

Розглянуті рішення демонструють широкий спектр можливостей у сфері аналізу й діагностики персональних комп'ютерів. Усі рішення мають спільну рису, а саме орієнтованість на користувачів із достатнім рівнем технічних знань, які здатні самостійно інтерпретувати технічні показники.

Висновок до розділу 1

У першому розділі проаналізовано сучасні підходи до діагностики несправностей персональних комп'ютерів. Розглянуто класифікацію несправностей, основні методи виявлення несправностей, роль експертних і рекомендаційних систем у процесі діагностики та існуючі рішення. Окрему увагу приділено можливостям застосування великих мовних моделей для підвищення

ефективності та адаптивності рекомендаційних рішень. Сформовано теоретичну основу для подальшої розробки рекомендаційної системи для виявлення несправностей у персональних комп’ютерах.

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОЄКТУВАННЯ АРХІТЕКТУРИ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

2.1 Постановка задачі та вимоги

Перед проєктуванням програмного забезпечення необхідно чітко сформулювати вимоги до майбутньої системи беручи за основу проведені у попередньому розділі аналізу предметної області. Це дозволить визначити конкретні задачі, функціональні можливості, технічні обмеження та критерії якості розроблюваного застосунку.

Кінцевий результат роботи має бути представлений у вигляді MVP програми для платформи Windows, але з можливістю розширення на інші платформи. Дана програма позиціонується як допоміжний інструмент для користувача, що не має достатнього рівня технічних знань для самостійного усунення несправностей, і водночас як аналітична платформа для дослідження частоти, типів і тенденцій несправностей. У перспективі результат роботи може бути розширений і слугувати основою для побудови інтелектуальних сервісів технічної підтримки.

Програма має виконувати наступні поставлені задачі:

- Здійснювати інтерактивну діагностику. Реалізація у форматі опитувальника користувача з адаптивною логіку запитань, тобто вибір наступного запитання залежить від попередніх відповідей;
- Фіксувати результати діагностики. Після роботи програми мають бути наявні логи;
- Автоматично збирати системну інформацію:
    1. Перевіряти стан оновлень;
    2. Визначати швидкість інтернет з’єднання;
    3. Зчитувати технічні характеристики пристрою;
    4. На основі зібраних характеристик сповіщати користувача про потужність його системи в порівняння.

- Формування результатів інтерактивної і автоматичної діагностики у форматі звіту. Якщо наявні несправності, то звіт має містити висновок про їх ймовірні причини та структуровані рекомендації по їх вирішенню;
- Підтримувати можливість розширення без потреби зміни ключового функціоналу. Наприклад, додавання питань і відповідей в інтерактивну діагностику, можливість додавання користувацькі shell скрипти або інші, які підтримуються оперативною системою;
- Забезпечення анонімізації даних. Збережені дані не мають містити персоналізовані дані користувача.

Беручи за основу поставлені задачі можна виділити функціональні та не функціональні вимоги.

Функціональні вимоги:

- Реєстрація користувачів і ведення історії діагностичних сесій. Дані про користувача містять лише ідентифікатор і роль;
- Проведення інтерактивного опитування. Додатково мати можливість пропускати запитання та/або уточнення відповідей;
- Формування гіпотез про можливі причини несправності за допомогою використання:
    1. Rule-based правил;
    2. Ймовірнісних методів;
    3. Машинного навчання;
    4. Великих мовних моделей.
- Автоматичний збір технічних показників системи:
    1. Характеристики процесору, графічного ядра, оперативної пам’яті;
    2. Стан операційної системи та чи наявні оновлення;
    3. Тест швидкості інтернету;
    4. Формування локального профілю продуктивності для порівняння з іншими системами.
- Збереження всіх сесій у базі даних. Система має фіксувати дії користувача під час діагностики і записувати їх в базу даних. Наприклад, які запитання

були поставлені користувачі і які були надані відповіді, результати діагностики та дані, які аналізувалися для надання рішення. Ці дані можуть бути використані для аналізу типових несправностей, навчання моделей машинного аналізу та оцінки ефективності системи;

- Механізм для збирання журналів подій;
- Можливість надання зворотного зв'язку. Наприклад, «чи допомогла у вирішенні несправності надана рекомендація?» з відповідями «Так», «Ні», «Не повністю».

Нефункціональні вимоги:

- Кросплатформеність. Застосунок повинен працювати на різних платформах, а саме: Windows, macOS, Linux. І потенційно на Android, iOS;
- Продуктивність. Застосунок має коректно працювати навіть на відносно слабких системах та використовувати малу кількість ресурсів;
- Масштабованість. Можливість розширення функціоналу додатковими сервісами діагностики, джерел даних і тд;
- Безпека. Збережені дані та дані, що передаються на стороні ресурси не мають містити персональних даних користувача;
- Можливість розширення UI, а саме можливість заміни візуального відображення інтерфейсу та оформлення програми;
- Відмовостійкість. За відсутності інтернету додаток має працювати коректно, але з обмеженими результатами;
- Логування. Автоматичне створення записів про несправності, відповіді користувача та надані рекомендації.

2.2 Вибір технологій реалізації

Враховуючи поставлену задачу та вимоги, які передбачають розробку інтерфейсу користувача, обробку даних, збереження історії сесій, інтеграцію машинного навчання та взаємодію з великими мовними моделями, для реалізації даних можливостей необхідна гнучка, розширювана та кросплатформна технологічна основа.

У межах даної кваліфікаційною роботи було обрано такий стек технологій: C#, .NET 8, .NET MAUI, Entity Framework Core та ML.NET. Таке поєднання дозволяє створити єдину екосистему, у якій клієнтський інтерфейс, бізнес-логіка, аналітика та робота з базою даних інтегруються без потреби у використанні сторонніх бібліотек чи скриптів для поєднання з іншими мовами програмування.

Головним архітектором мови програмування C# є Андерс Гейлсберг, що також є творцем Turbo Pascal та головний архітектор Delphi [24]. З моменту створення C# стала основною мовою в екосистемі .NET, що дає можливість створювати кросплатформні застосунки, тобто можуть мати одну кодову базу і при цьому буде один застосунок, якщо не брати до уваги специфічних реалізацій викликів притаманних кожній платформі.

Мова C# було обрана як основний інструмент реалізації, оскільки вона поєднує строгу типізацію та високу продуктивність. Також C# реалізовує об’єктно-орієнтовану парадигму, що базується на чотирьох ключових принципів: інкапсуляція, успадкування, поліморфізм та абстракція [25-28]. Окрім цього мова поєднує ще елементи функціонального програмування: підтримка лямбда-виразів, делегатів та змінні, що не мають свого стану. Це дозволяє створювати більше декларативні, чисті та готові до тестування структури даних, що є важливим для аналітичних систем і модулів машинного навчання. Окрім вище згаданих переваг варто зазначити те, що код C# виконується у спільному середовищі виконання, який забезпечує автоматичне керування пам’яттю, збір сміття і обробку винятків.

Серед ключових причин вибору даної мови програмування:

- Єдність середовища розробки;
- Строга типізація;
- Підтримка асинхронного програмування;
- Сумісність із кросплатформним середовищем MAUI;
- Кросплатформеність;
- Велика кількість бібліотек.

Основою застосунку буде .NET MAUI, яка є сучасною технологією від Microsoft, що дозволяє створювати нативні інтерфейси для кількох операційних систем на основі спільного коду C# [25, 26]. MAUI являє собою еволюційне покращення фреймворку Xamarin, що був найпопулярнішим для використання у створені мобільних додатків для iOS та Android. Через зростаючу популярність таких фреймворків як Flutter та React і зростання обмежень у розвитку самого фреймворку було створено екосистему .NET MAUI [29]. З основних переваг можна виділити покращення елементів керування інтерфейсом з високою гнучкістю у створені власних елементів схожих до нативних на платформах [30]. Також в реальному часі при зміні коду оновлюється інтерфейс застосунку, що дуже важливо під час розробки бачити як найшвидше результат впливу змін в коді на роботу кінцевого продукту [31]. І звичайно довго строкова підтримка фреймворку, що гарантує стабільність у розробці.

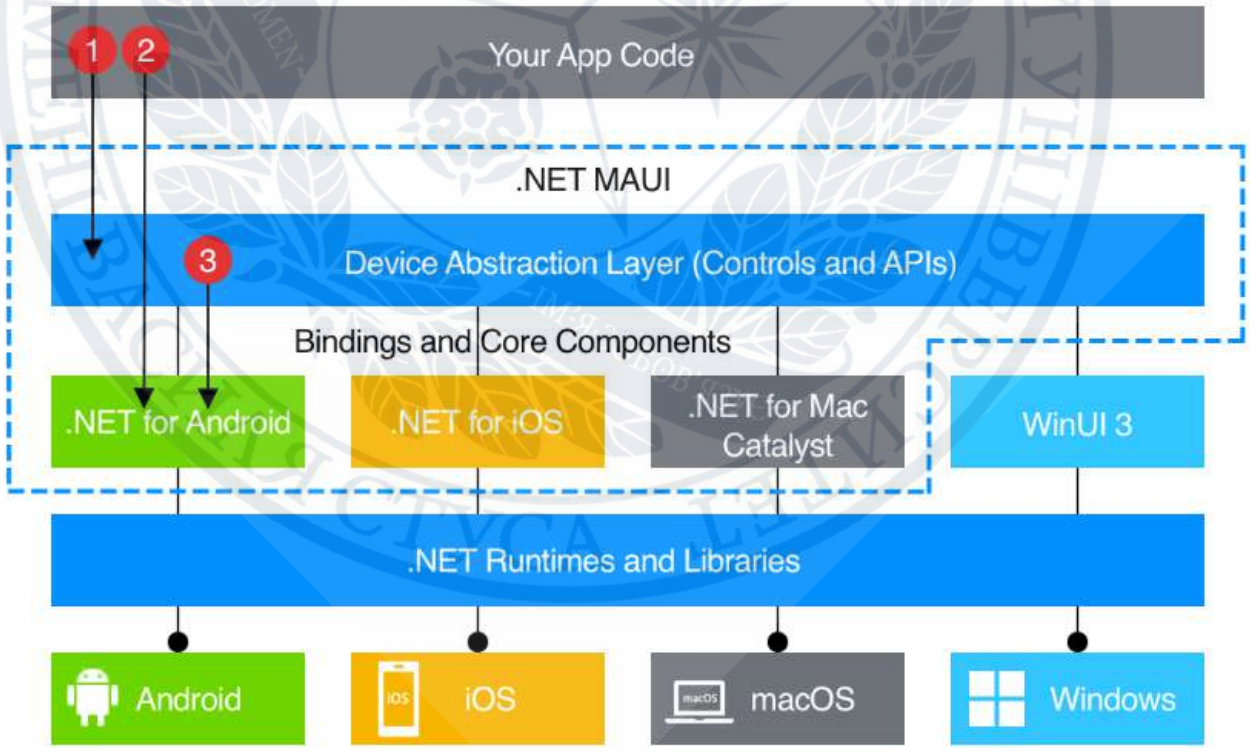

Рисунок 2.1 – Загальний огляд архітектури .NET MAUI [31]

Для взаємодії з базою даних обрано Entity Framework Core – сучасну кросплатформну версію ORM-технологію, що дозволяє працювати з базами

даних через об’єкту модель без ручного написання SQL-запитів. Entity Framework Core є переосмисленою й переписаною версією класичного Entity Framework 6. На відміну від старих версій Entity Framework Core спочатку створювалася як кросплатформна.

З основних переваг даного фреймворку можна виділити:

- Обгортка ObjectContext у вигляді DbContext. Дана обгортка містить набір API викликів, які відзначаються простотою та зручністю у використанні [32]. Також з DbContext має перевагу у підходах до створення бази даних, так як окрім Database First та Model First, містить підхід Code First на противагу ObjectContext;
- Більшу кількість підтримуваних баз даних. Наприклад, Jet (Microsoft Access), Azure Cosmos DB та в пам’яті для тестування;
- POCO-класи замість EntityObject. Оновлений фреймворк більше не використовує для сутностей базовий клас EntityObject. Тепер класи сутностей завжди є об’єктами POCO [33]. Термін POCO є абревіатурою Plain Old C# Object або Plain Old Class Object для уникнення прив’язки до конкретних мов або технологій. Використовується для опису класів, які не успадковуються від спеціальних базових класів і не використовуються особливі типи властивостей [34].

Варто зазначити, що не весь функціонал Entity Framework 6 буде імплементований в Entity Framework Core, так як деякі функціональні частини залежали від Entity Data Model та/або є складними в реалізації з мінімальною потрібністю, але Entity Framework Core пропонує багато нововведень в порівнянні з попередником [35].

Для реалізації аналітики буде використано фреймворк ML.NET, що являє собою машинне навчання. Основною перевагою є відсутність потреби у використані інших мов програмування чи бібліотек, як TensorFlow або Python, хоча при потребі можна доповнити функціонал їх використанням [36, 37]. Окрім того фреймворк є високо продуктивним і точним. Навчена модель на наборі даних Amazon обсягом 9 Гб, ML.NET мав точність 95%, в той час як інші

популярні фреймворки машинного навчання мали помилки через відсутність достатньої кількості пам’яті. І навіть при зменшеному розмірі даних того ж датасету ML.NET продемонстрував високу швидкість опрацювання[36].

Для тренування власної моделі фреймворк надає три підходи:

- Model Builder. Візуальний інструмент, який інтегрований у редактор Visual Studio, є простим та має зрозумілий інтерфейс. Він надає можливість покроково налаштувати процес навчання моделі. Після завершення навчання інструмент дає можливість створити автоматично згенерований код з обраними налаштуваннями для тренування і зберігає модель у форматі zip-архіву, яку можна потім використовувати у проєкті;
- Код. Передбачає безпосереднє створення й налаштування обробки даних у коді. Даний підхід є максимально гнучким, так як візуальний інструмент не може відобразити всі можливі налаштування для тренування, які надає API ML.NET;
- Командний рядок. Фреймворк дозволяє використати командний рядок для створення, тренування й експортування моделі без запуску графічного редактора. Однією з переваг є можливість використання, коли немає доступу до графічного редактору. Наприклад, процес налаштування відбувається на сервері або у хмарному сервісі. Також даний підхід даний можливість для автоматизації процесу навчання і інтеграції з CI/CD.

Також однією з головних перевага даного фреймворку є наявність бібліотеки AutoML, яка дозволяє автоматизувати процес вибору оптимальної моделі та її гіперпараметрів [38]. Бібліотека проводить низку експериментів в яких порівнюються різні алгоритми і конфігурації даних. Потім з отриманих результатів обирає найвищий варіант. Використання AutoML є зручним варіантом для використання у навчальних або дослідницьких проєктах, де треба отримати якісну модель без глибоких знань у галузі машинного навчання. Дана бібліотека підтримується як в у візуальному інструменті Model Builder та і через ML.NET API у коді.

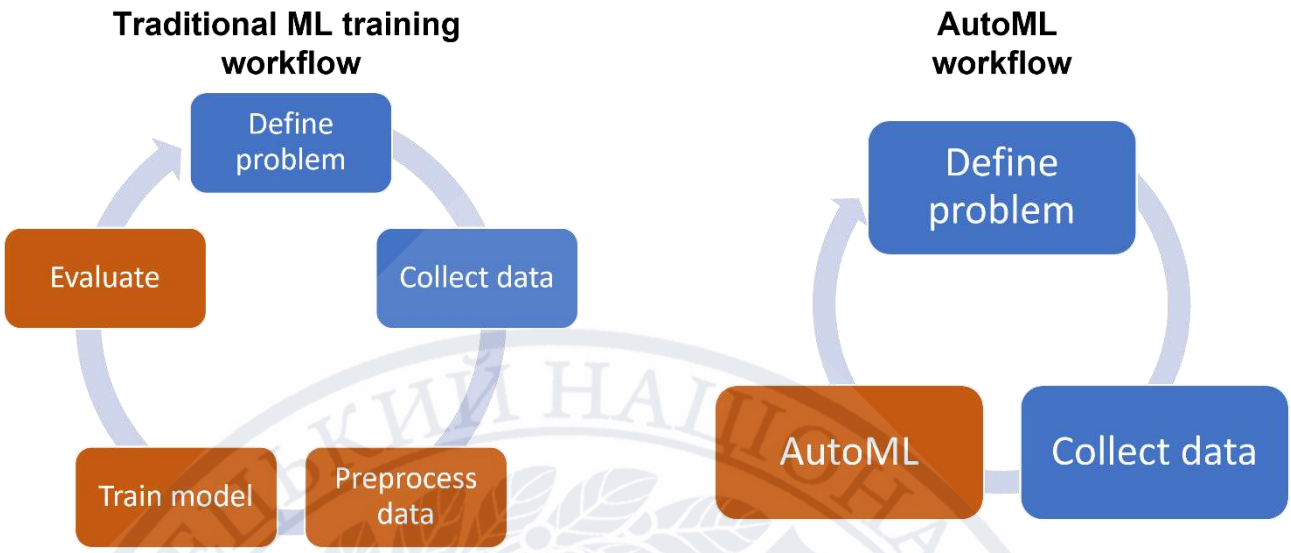

Рисунок 2.2 – Порівняння традиційного машинного навчання моделей та навчання з AutoML [39]

2.3 Архітектура додатку

Під час розробки рекомендаційної системи важливо спроєктувати архітектуру, яка забезпечуватиме не лише працездатність на початкових етапах, але й матиме можливість для подальшого розширення, підтримки та модифікацій. У переважній більшості невеликих або експериментальних проєктів, зокрема на етапі мінімально життєздатного продукту (Minimum viable product, MVP) часто обирають монолітну архітектуру, яка передбачає, що всі компоненти розміщені в одному логічному модулі. Такий підхід є цілком виправданим на ранніх етапах, оскільки забезпечує високу швидкість розробки, мінімальні витрати на налаштування структури та швидке отримання працездатного проєкту.

Однак мета не лише у тому, щоб програма виконувала діагностику, а щоб її структура була логічною, зрозумілою та зручною для супроводу. Якісно спроєктована архітектура дозволяє швидко знаходити та усувати помилки, розширювати функціонал і підтримувати існуючий. Під час розробки будь-якої

системи необхідно розраховувати не лише на індивідуальне використання коду, а й на те, що в майбутньому проєкт може підтримуватися чи розвиватися іншими людьми. Саме тому важливо одразу закласти структуру з чітким розділенням обов’язків, що забезпечить читабельність, тестованість і стабільність для довготривалої підтримки.

Оскільки рекомендаційна система включає різні етапи роботи з інформацією, а саме збирання, аналіз, видачу рекомендацій та інтеграцію з зовнішніми сервісами, і базуючись на твердженнях викладених вище було обрано багаторівневу архітектуру [40]. Такий підхід дозволяє чітко відокремити логіку відображення інтерфейсів від бізнес логіки діагностування, отримання даних з баз від їх обробки. В результаті кожен рівень системи виконує власну функцію і може вдосконалюватися та/або замінюватися незалежно від інших.

Рисунок 2.3 – Схематична архітектура рекомендаційної системи

Архітектура рекомендаційної системи передбачає поділ на:

- Рівень представлення.

  Відповідатиме за відображення інтерфейсів користувачеві, а саме відображення стану діагностичної сесії, взаємодію з опитувальником та передачу користувацької взаємодії на рівень бізнес логіки. На даному рівні буде відсутня будь-яка логіка обчислень і т. ін.;

- Рівень бізнес логіки.

  Відповідатиме за виконання всієї логіки рекомендаційної системи, а саме за формування вектору симптомів на основі відповідей користувача, визначатимуться наступні кроки діагностики та здійснюватиметься виконання відповідних діагностичних модулів. Крім того буде реалізовано інтеграцію із сторонніми сервісами;

- Рівень даних.

  Відповідатиме за взаємодію з базою/базами даних. Буде реалізовано підключення, отримання та запис даних. Відокремлення взаємодій з базою/базами даних дає можливість масштабування без впливу на бізнес логіку.

Узагальнена схема архітектури, що наведено на рисунку 2.3, демонструє структуру системи та показує як організовано поділ чітко розділених модулів на архітектурні рівні.

2.4 Архітектурні та проєктувальні патерни

Для реалізації рекомендаційної системи планується використання низки архітектурних та проєктувальних патернів, які забезпечать чітке розділення відповідальностей, зменшення зв’язності між компонентами та можливість подальшого масштабування системи. Використання патернів дозволить сформувати структуровану, гнучку та легко підтримувану архітектуру, що є ключовим аспектом для багатошарового застосунку. Нижче наведено патерни, які будуть використані під час розробки рекомендаційної системи:

- MVVM. Model-View-ViewModel. Є основним у побудові взаємодії між бізнес логікою та інтерфейсом користувача. Даний патерн є частиною сімейства архітектурних шаблонів MV*. Наприклад, Model-View-Controller (MVC) , Model-View-Presenter (MVP), і тд. Вони забезпечують чітке розділення зон відповідальності логіки застосунку та його візуальної частини [41-43]. Кожна частина в назві патерну відповідає окремому рівню відповідальності:
    1. Model. Представляє структуровані дані отримані від бізнес логіки або інших компонентів;
    2. View. Відповідає за візуальне відображення інтерфейсу та виклик подій, які виконуються під час взаємодії з користувачем;
    3. ViewModel. Містить логіку керування станом інтерфейсу та реагуванням на події від View.

В реалізації використовується для розділення рівнів бізнес логіки та представлення. ViewModel отримуватимуть дані з сервісів та представлятимуть їх у форматі зручному для користувачів.

Рисунок 2.4 – Граф залежностей патерну MVVM [44]

- Strategy. Є ключовим поведінковим шаблоном проєктування, який використовується для організації різних варіантів виконання певної операції або алгоритму. Даний патерн дозволяє інкапсулювати логіку у вигляді окремих класів та перемикатися між ними як під час роботи програми, так і в залежності від кінцевої платформи користувача. Реалізовується зазвичай за допомогою одного спільного інтерфейсу, який визначає загальний контракт для всіх можливих імплементацій. Клас або сервіс, який користується стратегією, працює лише через інтерфейс і не залежить від конкретної реалізації алгоритму. У контексті

рекомендаційної системи даний патерн використовуватиметься для реалізації різних підходів діагностики та технічної діагностики, яка обиратиметься залежно від платформи;

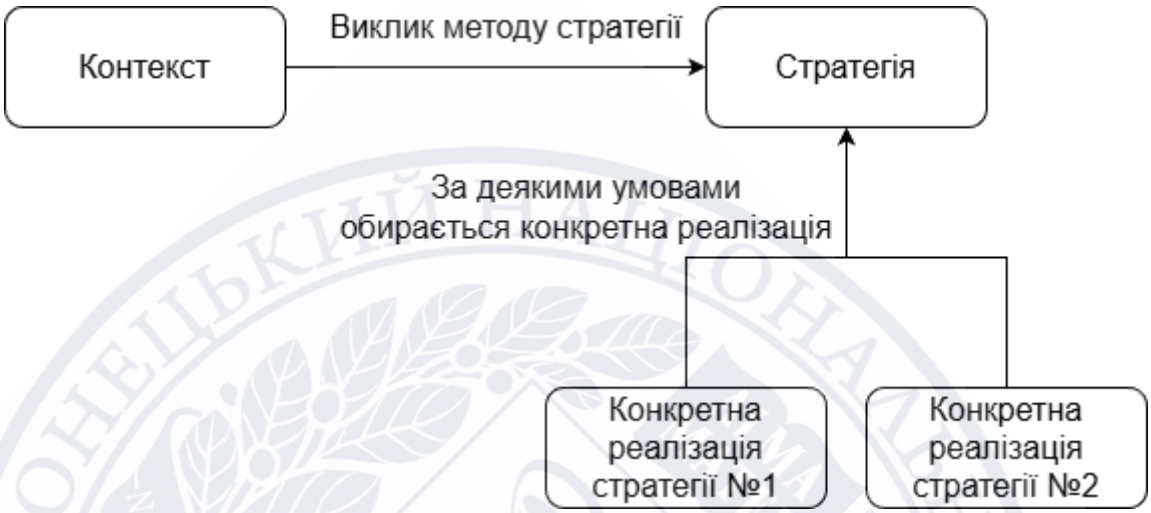

Рисунок 2.5 – Схема роботи патерну Strategy [45]

- DI. Dependency Injection. Важливий архітектурний підхід, що є одним з фундаментальних принципів для побудови слабкозв'язних систем. Патерн базується на ідеї інверсії залежностей, тобто замість того, щоб класи самостійно створювали необхідні їм об'єкти, ці об'єкти передаються їм ззовні. Досягається така умова за допомогою відокремленого процесу, що відповідає за створення об'єктів, а класи отримають залежності переважно через конструктор або в інших реалізаціях патерну через метод, властивість або атрибут. В обраному фреймворку .NET MAUI наявна імплементація даного патерну у форматі вбудованого контейнеру, який дозволяє реєструвати сервіси, класи і інші компоненти з різними областями життєвого циклу, що визначають як довго існуватиме об'єкт і в яких умовах він буде повторно використаний. Основні варіанти життєвого циклу:
    1. Singleton. Створює об'єкт один раз на весь період роботи застосунку;
    2. Scoped. Створюється один раз для певного логічного контексту виконання. Наприклад, для однієї сторінки або операції;

3. Transient. Створюється щоразу при запиті на отримання об’єкту.

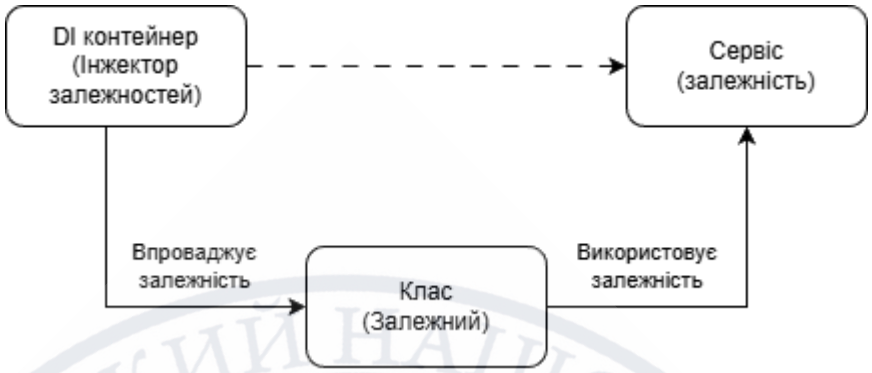

Рисунок 2.6 – Схема роботи патерну впровадження залежностей [46]

- Adapter. Структурний патерн проєктування, що використовується для узгодження взаємодії між двома компонентами, які не мають сумісних інтерфейсів. Для реалізації створюється проміжний клас, який перетворює один формат даних отриманий, наприклад, від стороннього сервісу. Основна мета полягає у зменшені потреби модифікації бізнес логіки та залежності від сторонніх сервісів. В реалізації патерн використовуватиметься для інтеграції системи з великими мовними моделями.

2.5 Порівняльний аналіз підходів до діагностики

У попередньому розділі було розглянуто основні підходи до виявлення несправностей, що включають у себе rule-based моделі, ймовірнісні методи, зокрема Байєсівський метод, машинне навчання, а також можливість використання великих мовних моделей. Кожен із цих підходів має власні переваги, обмеження та сфери застосування. Для побудови ефективної рекомендаційної системи важливо здійснити порівняльний аналіз, який надасть цілісне уявлення про сильні сторони кожного підходу та дозволить визначити доцільність їх застосування в одній рекомендаційній системі. Нижче наведено таблицю 2.1, що узагальнює основні властивості кожного з розглянутих підходів та демонструє їх ключові відмінності.

Таблиця 2.1 – Схематична структура рекомендаційної системи.

Критерій	Rule-based	Ймовірнісний	Машинне навчання	Великі мовні моделі
Потреба у даних	Низька, достатньо експертних правил	Помірна, потрібні апріорні та умовні ймовірності	Висока, потрібен великий датасет	Низька для використання, висока для навчання моделей
Інтерпретованість	Дуже висока, чіткі правила	Висока, ймовірності	Низька-середня (залежить від моделі)	Середня, залежить від запитів
Стійкість до шуму	Низька	Висока	Середня	Середня
Гнучкість та масштабованість	Низька, важко підтримувати великі набори правил	Середня	Висока	Висока
Точність	Висока для типових випадків	Висока за повних ймовірностей	Дуже висока при хорошому датасеті	Нестабільна, можливі «галюцинації»
Необхідність участі експертів	Дуже висока	Висока	Середня	Низька
Затрати на реалізацію	Низькі	Середні	Високі	Середні-високі

Зважаючи на вимоги точності, пояснюваності та можливості подальшого розвитку, доцільним є використання гібридного підходу, який поєднає сильні сторони кожного підходу:

- Rule-based підхід. Забезпечує швидке отримання результату завдяки чітко визначеним правилам і високій інтерпретованості;
- Ймовірнісний підхід. Має стійкість до неповних та суперечливих даних, дозволяє перевизначати ймовірності кожної потенційної несправності;
- Машинне навчання. Може надавати результати з високою точність за умови наявності достатнього обсягу історичних даних у датасеті та здатне виявляти приховані закономірності, недоступні традиційним підходам;
- Великі мовні моделі. Забезпечує гнучкість у використанні для пересічного користувача, оскільки здатні працювати з природною мовою, формувати пояснення та адаптувати рекомендації в контексті конкретної несправності.

Порівняльний аналіз підходів діагностики показує, що жоден з підхідів не є універсальним рішенням, оскільки вони по різному поводяться за умов нестачі даних, змінності симптомів або появи нових типів несправностей. Використання лише одного підходу призведе до обмеженої точності. тому у їх комбінація дозволить створити систему, яка поєднає переваги і компенсуватиме недоліки.

2.6 Алгоритм діагностики

Процес діагностики несправностей у рекомендаційній системі складається з кількох послідовних етапів та базується на комбінування трьох підходів: правил, ймовірнісних оцінок та методів машинного навчання. Така діагностика дозволяє не лише врахувати експертні знання, але й адаптуватися до нових даних. Схематична структура рекомендаційної системи з використанням трьох етапів під час діагностики наведено на рисунку 2.7.

Процес діагностики несправностей у рекомендаційній системі, як було зазначено у вимогах, буде містити інтерактивне опитування користувача. Дане опитування буде мати динамічний формат, де послідовність та вибір запитань

буде залежати від обраних відповідей. Кожна відповідь визначатиме наступне питання, створюючи адаптивне дерево рішень, що дозволятиме зібрати потенційно релевантні симптоми щодо несправності.

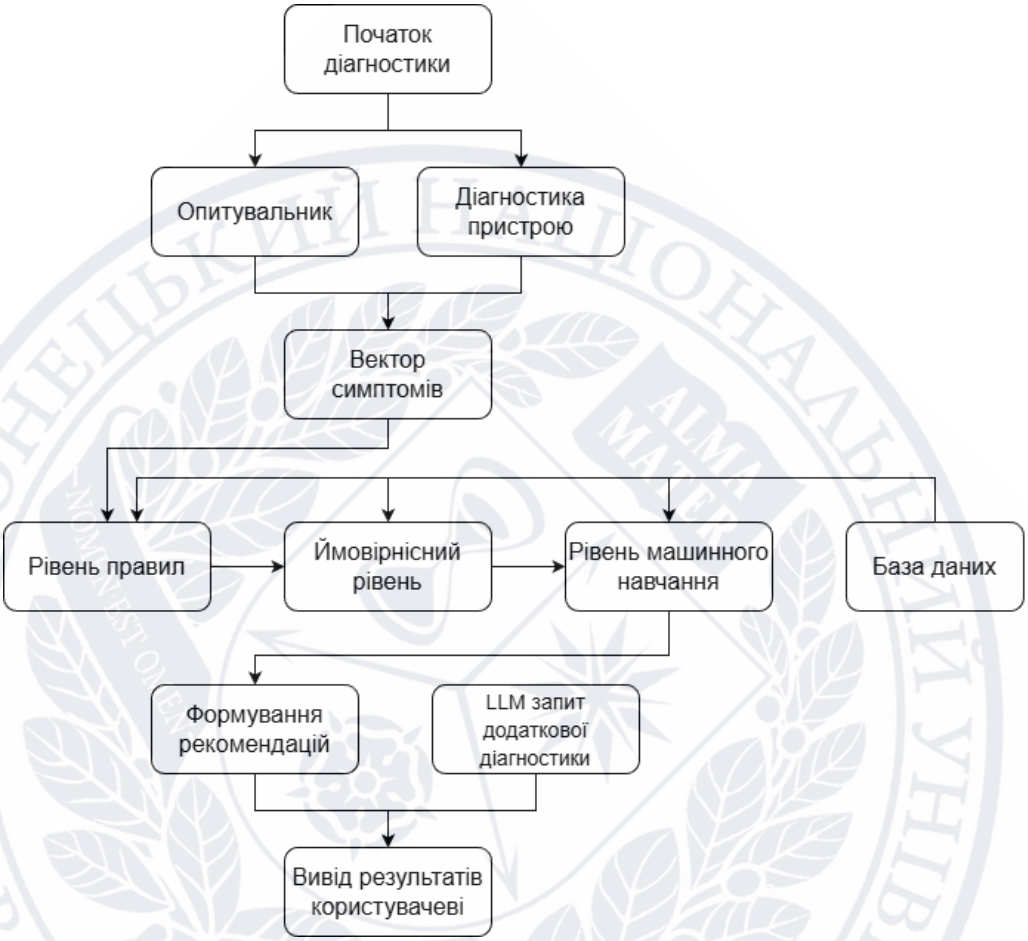

Рисунок 2.7 – Схематична структура рекомендаційної системи зі всіма етапами діагностики

Паралельно система здійснюватиме автоматичний збір технічних показників і журналів подій. До технічних показників належать температура процесора, навантаження на систему, стан дискової підсистеми і тд. Крім цього здійснюється аналіз журналів подій, що дасть змогу виявити типові збої в роботі драйверів, помилки застосунків або системні повідомлення про критичні події. Ці дані не зберігатимуться у вигляді сирих величин, а будуть інтерпретовані у формат логічних симптомів. Наприклад, CPU_OVERHEAT, що позначатиме перегрів центрального процесора. Порогові значення для формування таких

симптомів визначатимуться на основі технічній документації. Наприклад, документації виробників обладнання та/або постачальників.

Після завершення збору всіх даних формується єдиний вектор симптомів, що поєднує три джерела: відповіді користувача, технічні показники та журнали подій. Даний вектор передається в аналітичну підсистему, де проходитиме послідовну обробку трьома рівнями алгоритму діагностики:

1. Рівень правил. На цьому етапі застосовуватимуться експертні правила, які описуватимуть залежності формату «Якщо симптоми X, Y, Z, то можлива несправність є A». Кожне правило складатиметься з одного або декількох симптомів і визначатиме не тільки потенційну несправність, а й первинний рівень впевненості у правильності знайденої несправності. Даний рівень формуватиме початковий список кандидатів, які могли спричини наявні проблеми системи. Цей етап діагностики буде виконувати функцію первинної фільтрації та буде основним джерелом можливих діагнозів;
2. Байєсівський рівень. На основі списку кандидатів отриманих від попереднього рівня буде робитися розрахунок апостеріорної оцінки, що буде враховувати ймовірності кожної несправності зі списку кандидатів та умовні ймовірності наявних симптомів. Апріорні та умовні ймовірності зберігатимуться в експертній базі знань. Їх значення будуть визначатися на основі статистики, експертної оцінки або даних реальних діагностичних сесій. Даний рівень не розширюватиме список можливих несправностей, а лише уточнюватиме ймовірності вже знайдених на минулому рівні кандидатів;
3. Рівень машинного навчання. Фінальний етап використовуватиме модель навчену на історичних даних користувачів. Модель буде отримувати вектор ознак, результати рівня правил та Байєсівського рівня. На основі отриманих даних буде розраховувати остаточний прогноз. Поєднання статистичних та експертних методів дозволяє системі адаптуватися до нових типів симптомів та з часом покращувати точність результатів.

Результати діагностики міститимуть перелік найімовірніших несправностей разом з набором рекомендацій щодо їх усунення. Спочатку результати зберігатимуться локально на пристрої користувача, а після отримання зворотного зв’язку щодо корисності рекомендацій передаватимуться на центральний сервер. Це дозволить поступово уточнювати експертні ймовірності, оновлювати модель машинного навчання та підвищувати точність системи загалом.

2.7 Архітектура та організація даних рекомендаційної системи

На основі прописаних вимог до рекомендаційної системи можна виділити потребу у двох логічно незалежних базах даних:

- Користувацька база – зберігатиме інформацію про сесії користувачів, результати діагностики та інші дані, що формуються під час проходження опитування;
- Експертна база знань – міститиме типові несправності, можливі симптоми, взаємозв’язки симптомів та несправностей та рекомендовані рішення, тобто довідникову інформацію, необхідну для діагностики та надання рекомендацій.

Таке розділення відповідає принципам сегрегації відповідальностей і дозволяє зручно керувати динамічними та статичними даними, а саме користувацькими та експертними відповідно. Також забезпечується безпечне оновлення експертних даних без ризику втрати історичних даних, можливість навчання машинних моделей на накопичених сесіях, а також можливість окремого розгортання експертної бази знань.

Користувацька база даних може бути реалізована як на стороні клієнта, так і на стороні центрального серверу. В першого варіанту основною перевагою є повна анонімність користувача, так як дані будуть зберігатися локально і унеможливлюватиме витік будь-яких даних. В той же час другий варіант надає можливість віддалено перенавчати машинні моделі для покращення результатів роботи на основі вже реальних даних та буде можливість аналізу ефективності

наданих рекомендацій експертної бази знань. Так як обидва підходи мають сильні переваги і передбачається, що інформація не буде містити користувацьких персональних даних, можна об’єднати локальну базу для користувача та базу в яку будуть приходити записи від всіх користувачів, щоб надалі надавати кращі рекомендації.

В свою чергу експертна база знань також має три варіанти реалізації, а саме: інтеграція в сам додаток, реалізація на стороні центрального серверу та змішаний. Перший варіант має перевагу тільки, якщо у користувача немає доступу до мережі інтернету, так як вона при такому підході буде знаходитися локально, то користувач зможе скористатися рекомендаційною системою. Але в той же час при збільшенні бази знань буде зростати розмір додатку, що обмежить коло потенційних користувачів у майбутньому. Рішення з реалізацією на стороні центрального серверу надає користувачам можливість отримання найактуальніших рекомендацій, зменшення потреби у частому оновленні застосунку для отримання нових рекомендацій та централізований контроль якості рекомендацій. Змішаний варіант передбачає часткове інтегрування бази в застосунок, який буде використовуватися для надання рекомендацій найпоширеніших несправностей за відсутності підключення до мережі інтернет, а коли підключення наявне у користувача та є стабільним буде відправлятися запит на сервер з актуальною експертною базою знань.

На основі вимог до побудови рекомендаційної системи та аналізу наявних типів баз даних було обрано реляційну модель збереження даних. Такий вибір зумовлений необхідністю підтримки складних зв’язків між сутностями, забезпечення цілісності даних та можливості виконання складних аналітичних запитів.

Проєктування реляційної бази даних для рекомендаційної системи діагностики несправностей персонального комп’ютера вимагає особливої уваги до нормалізації. Однією з основних причин є те, що система буде працювати з великою кількістю різних сутностей даних. Наприклад, діагностичними сесіями, симптомами, запитаннями, відповідями до них, правилами, ймовірностями та

рекомендаціями. Кожна з сутностей бере участь у формуванні вектору симптомів та в подальшому в діагностуванні у трьохрівневому алгоритмі. Тому важливо, щоб збережені дані не містили надлишковості, суперечностей або неоднозначних залежностей, які можуть призводити до помилок діагностики.

Нормалізація баз даних є послідовним процесом перетворення структур таблиць таким чином, щоб зменшити дублювання, уникнути логічних помилок та забезпечити цілісність даних [47]. Під час проєктування баз даних рекомендаційної системи нормалізація виконуватиметься до третьої нормальної форми, що відповідає загальноприйнятим практикам побудови баз даних та забезпечує захист від переважної частини типових аномалій.

Перша нормальна форма визначає дві ключові вимоги:

- Усі значення повинні бути атомарними. Має на увазі, що значення містять одну логічну неподільну одиницю інформації;
- Кожен набір пов’язаних даних має бути виокремлений у окрему таблицю;
- Кожен набір пов’язаних даних має містить первинний ключ.

У контексті рекомендаційної системи це дозволяє гарантувати, що всі елементи будуть чітко визначеними та однорідними [47]. В свою чергу, недотримання першої нормальної форми призводить до неможливості працювати алгоритму діагностики, проблеми при міграціях та зміні структури даних.

Друга нормальна форма застосовується до таблиць, де первинний ключ складається з декількох неподільних одиниць інформації. Дана форма вимагає, щоб кожне неключове поле даних залежало від усього ключа, не від його частини. Дозволяє уникнути ситуацій, наприклад, коли зміна одного елемента спричиняє потрбеу оновлювати десятки або сотні рядків. У контексті рекомендаційної системи дозволяє гарантувати, що описові або класифікаційні властивості сутностей даних не дублюються.

Третя нормальна форма забезпечує усунення транзитивних залежностей. Дана форма вимагає, щоб жоден неключове поле даних не залежало від іншого неключового поля даних. Наприклад, якщо певна інформація може бути

виведена на основі іншої характеристики сутності даних, яка не є ключовою, це створює ризик дублювання або появи суперечливих даних. Дотримання третьої нормальної форми у діагностичній системі дозволяє запобігти накопиченню зайвої інформації, дублюванню неключових полів даних.

Нормалізація до третьої форми дозволяє уникнути шести основних аномалій, що виникають у неправильно структурованих базах даних:

- Аномалія вставлення. Виникає, коли при додаванні нового запису потрібно вводити додаткові дані, які логічно не стосуються об’єкта або ще недоступні на момент вставки. Така поведінка означає, що структура таблиці побудована таким чином. Що окремі сутності не можуть існувати самостійно, хоча з логічної точки зору мають бути незалежні;
- Аномалія оновлення. Виникає, коли одна й та сама логічна інформація зберігається у декількох рядках. У такій ситуації будь-які зміни потрібно виконувати в кожному рядку, де повторюються дані. Якщо оновлення не буде проведено повністю або послідовно, у базі з’являться суперечливі значення. Це призводить до втрати цілісності даних, логічних конфліктів під час аналітичних запитів і потенційних помилок у роботі бази даних, яка покладається на узгодженість інформації;
- Аномалія часткової залежності. Є наслідком порушення другої нормальної форми. Виникає у таблицях зі складеним первинним ключем. Призводить до ситуації, коли частина інформації дублюється у багатьох рядках. У результаті структура стає надлишковою, менш керованою і призводить до появи інших аномалій. Наприклад, аномалії оновлення та аномалії вставлення;
- Аномалія транзитивної залежності. Виникає при порушені третьої нормальної форми, тобто коли неключові поле даних залежить не лише від первинного ключа. Призводить до надмірної пов’язаності полів даних і до потенційної неконсистентності даних. Будь-яка зміна одного з членів транзитивних залежностей вимагає оновлення інших пов’язаних полів, інакше база даних міститиме суперечливі значення;

- Аномалія надлишковості. Виникає при дублюванні даних без логічної необхідності. Може бути результатом неправильного структурування сутностей даних, поєднанням кількох логічних об’єктів у одну таблицю або повторенням однакових полів даних у багатьох рядках. Надлишковість призводить до збільшення розміру бази даних, ускладнення підтримки інформації та до зниження продуктивності запитів. Крім того, робить базу даних вразливою до аномалій оновлення та видалення.

2.8 Принципи збереження та анонімізації даних

Забезпечення безпечного збереження та конфіденційності інформації користувача є одним з ключових аспектів проєктування рекомендаційної системи діагностики несправностей. Архітектура системи спроєктована таким чином, щоб гарантувати мінімальне збирання інформації, відсутність персональних ідентифікаторів у базах даних, а також можливість подальшого аналітичного використання даних у знеособленій формі.

В системі буде використовуватися дві бази даних: користувацька та експертна. Кожна буде мати локальну та віддалену версію, тільки віддалена користувацька буде містити записи від всіх користувачів. Локальні зберігання гарантують повну конфіденційність, тоді як віддалені будуть містити дані, що перед відправкою будуть проходити етап анонімізації, тобто вилучення або заміну всіх полів з даними, які можуть безпосередньо або опосередковано ідентифікувати користувача. Наприклад, може бути використано псевдоанонімізацію на основі односторонніх хешів, що дозволяє зберегти унікальність запитів без можливості відновлення первинних значень або щоб розмежовувати користувачів і мати анонімність, можна використовувати цілочислове значення usedId, який буде унікальним і буде видаватися користувачу перед відправкою запиту.

Для гарантування безпеки передбачається реалізація низки технічних та організаційних механізмів:

- Шифроване передавання даних між клієнтським додатком і сервером за допомогою протоколу HTTPS/SSL;
- Локальне шифрування бази даних;
- Розмежування рівнів доступу. Аналітичні модулі працюватимуть лише з агрегованими даними, не маючи доступу до повного вмісту збережених сесій;
- Тимчасове зберігання. Якщо користувач не надав згоду на відправку анонімізованих даних, то зібрані інформація буде використана лише для поточної діагностики без зберігання локально та відправки на сервер.

2.9 Потенційні ризики та обмеження системи

Під час проєктування та реалізації рекомендаційної системи важливо врахувати низку технічних, методологічні та експлуатаційні ризики, які можуть впливати на точність діагностики, стабільність роботи та якість взаємодії з користувачем. Оскільки система поєднуватиме декілька різних підходів у реалізації алгоритму діагностики, кожен з них має власні обмеження. Знання про ці обмеження є критично важливим для подальшої модернізації системи, забезпечення надійності та мінімізації можливих помилок під час визначення потенційних несправностей.

Одним із ключових джерел ризиків є можливість роботи з неповними або суперечливими вхідними даними. Частина технічних показників може бути недоступною через обмеження операційної системи, відсутність необхідних драйверів, недостатній рівень прав користувача або нестабільність обладнання. До цього додається ймовірність того, що на початкових етапах система не матиме достатньої кількості історичних діагностичних прикладів, необхідних для коректного налаштування ймовірнісного рівня та рівня машинного навчання в алгоритмі діагностики. Важливим чинником є також велика різноманітність апаратних конфігурацій користувачів, що може призводити до відхилень у поведінці системи, які важко передбачити заздалегідь.

Окрему групу ризиків становлять ті, що пов’язані з взаємодією користувача з системою. Частина діагностичних даних формується на основі опитувальника, тому неправильне трактування запитання, помилковий вибір варіанта або випадкове надання неточних даних можуть призвести до некоректного формування вектору симптомів. У свою чергу це впливає на точність рівня правил та ймовірнісного рівня. Крім того, користувач може неправильно інтерпретувати надані рекомендації або обрати нерелевантний спосіб їх виконання, що може призвести до хибного уявлення про результативність порад. Такі ситуації можуть створювати викривлення в користувацькому сприйнятті точності системи, хоча алгоритм може працювати коректно.

Певні обмеження виникають і під час використання великих мовних моделей. Незважаючи на їхню здатність формувати пояснення та інтерпретувати симптоми природною мовою, зберігається ризик генерації некоректної або вигаданої інформації, що відоме як «галюцинації». Відповіді великих мовних моделей можуть бути чутливими до формулювання запиту, що спричиняє непередбачувані результати. Додатковим аспектом є необхідність забезпечення конфіденційності даних, так як для отримання окремої діагностики треба передати діагностичну інформацію у сторонній сервіс.

Не менш важливими є ризики, пов’язані з використанням машинного навчання. Ефективність моделей значною мірою залежить від обсягу та якості доступних даних, а на ранніх етапах розвитку системи їх може бути недостатньо. У таких випадках модель може бути надто чутливою до шуму або демонструвати низьку здатність до узагальнення. З часом можливе явище деградації моделі, коли змінюються характерні симптоми, сценарії використання або з’являються нові типи несправностей , що потребує регулярного перенавчання [48].

Варто враховувати й ризики, пов’язані зі збереженням та обробкою даних. Пошкодження локальних файлів, збій у файловій системі або некоректне завершення роботи додатка можуть призвести до втрати частини інформації. Також коли дані передаватимуться та зберігатимуться на віддаленому сервер,

виникає потреба у забезпеченні надійного захисту та анонімності під час комунікації мережевими каналами. Також важливо враховувати можливі перебої у доступі до серверної інфраструктури, оскільки це може обмежувати роботу аналітичних модулів системи.

Висновок до розділу 2

В другому розділі визначено вимоги до рекомендаційної системи, проведено порівняльний аналіз підходів до діагностики та обґрунтовано вибір технологій. Сформовано загальну структуру застосунку, описано моделі даних і спроєктовано роботу діагностичних модулів. Також визначено організацію баз даних, принципи анонімізації даних та потенційні ризики і обмеженнями системи, що створює основу для реалізації рекомендаційної системи у вигляді застосунку.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Створення проєкту

На початку реалізації застосунку було створено нове рішення на основі обраної технології .NET MAUI. Фреймворк .NET надає декілька різних шаблонів для створення MAUI-проєктів, кожен із яких має свою структуру та призначення. Це надає можливість обрати найбільш відповідний варіант відповідно до архітектурних вимог і складності майбутнього застосунку. На момент створення було доступно такі шаблони:

- Однопроєктний застосунок (Single project app). Створює рішення з одним проєктом, де міститься вся логіка, інтерфейси та платформно специфічні частини. Підходить для невеликих застосунків або прототипів;
- Blazor застосунок. Шаблон для створення гібридного застосунку, який поєднує можливості .NET MAUI та веб-технології Blazor. В таких проєктах інтерфейс будується за допомогою компонентів Razor, що дозволяє використовувати HTML, CSS та мову програмування C# одночасно;
- Багатопроєктний застосунок (Multi-project app). Створює розширену структуру, що складається з кількох проєктів розділених за принципом відповідальності. На відміну від однопроєктного шаблон код використовується модульний підхід, який дозволяє ізолювати різні частини системи. Уся основна логіка та користувацький інтерфейс знаходяться у проєкті бібліотеці, а платформні проєкти відповідають лише за реалізацію застосунку та специфічні можливості платформ.

Для реалізації рекомендаційної системи було обрано багатопроєктний шаблон, оскільки він найкраще відповідає вимогам архітектурного розділення, надає можливість легкої підтримки застосунку для розробника або команди розробників та дозволяє в майбутньому розширити застосунок з мінімально життєздатного продукту до повноцінного готового для використання користувачами.

Початкова структура, сформована шаблоном, та завчасно створенні проєкти за архітектурними вимогами містить такі проєкти:

- Бібліотека MAUI. Основний проєкт з загальним кодом застосунку;
- Проєкти по платформам. WinUI (Windows), Droid (Android), iOS, Mac;
- Бібліотека спільного коду (Common Library). Містить код, який зв'язує основну логіку додатку з фреймворком .NET MAUI. Також містить файли з розширенням xaml, що відповідають за побудову UI і є аналогом html файлів з веб програмування, та основні налаштування відносно побудови і стилів застосунку;
- Шар даних (Data Layer). Містить класи, які є відповідальними за надання доступу до даних та роботу з джерелами зберігання;
- Шар сутностей (Entity Layer). Містить класи, які описують структуру даних та моделі застосунку;
- Шар ViewModel (ViewModel Layer). Шар представлення логіки та даних для інтерфейсу користувача.
- Проєкт з машинним навчанням (ML Layer). У проєкті ізолюється фреймворк ML.NET від діагностичної логіки.
- Проєкт адаптер великих мовних моделей (LLM Layer). В даному проєкті ізолюється використання конкретного сервісу великої мовної моделі.

На рисунку 3.1 наведено структуру залежностей вищезгаданих проєктів застосунку. Початок кожного проєкту містить назву «SIDES», що є абревіатурою назви системи Software Issue Detection Expert System. Дана структура проєктів була побудована, спираючись на курс від Microsoft Visual Studio, у якому демонструють принципи побудови багатошарових застосунків та розподілення відповідальностей у рішенні [49]. Матеріали курсу, що складається з 18 епізодів, було використано як орієнтир для формування архітектури та логічної організації проєктів у багатопроєктному MAUI застосунку. У межах обраної архітектури основна частина бізнес логіки знаходитися в окремих проєктах, що є бібліотеками класів, які реалізують шари Data, Entity, ViewModel. Такий підхід дозволяє мінімізувати кількість залежностей між бізнес логікою та фреймворком

відображення інтерфейсу, оскільки UI-частина ізолюється в окремому проєкті. Також така архітектура надає суттєву гнучкість у випадку необхідності заміни фреймворку, що відповідає за інтерфейс та взаємодію з користувачем, так як базовий функціонал застосунку залишиться придатним до повторного використання. Таким чином, бізнес логіка не прив'язується до конкретного UI фреймворку, а застосунок отримує кращу масштабованість та можливість еволюції в майбутньому.

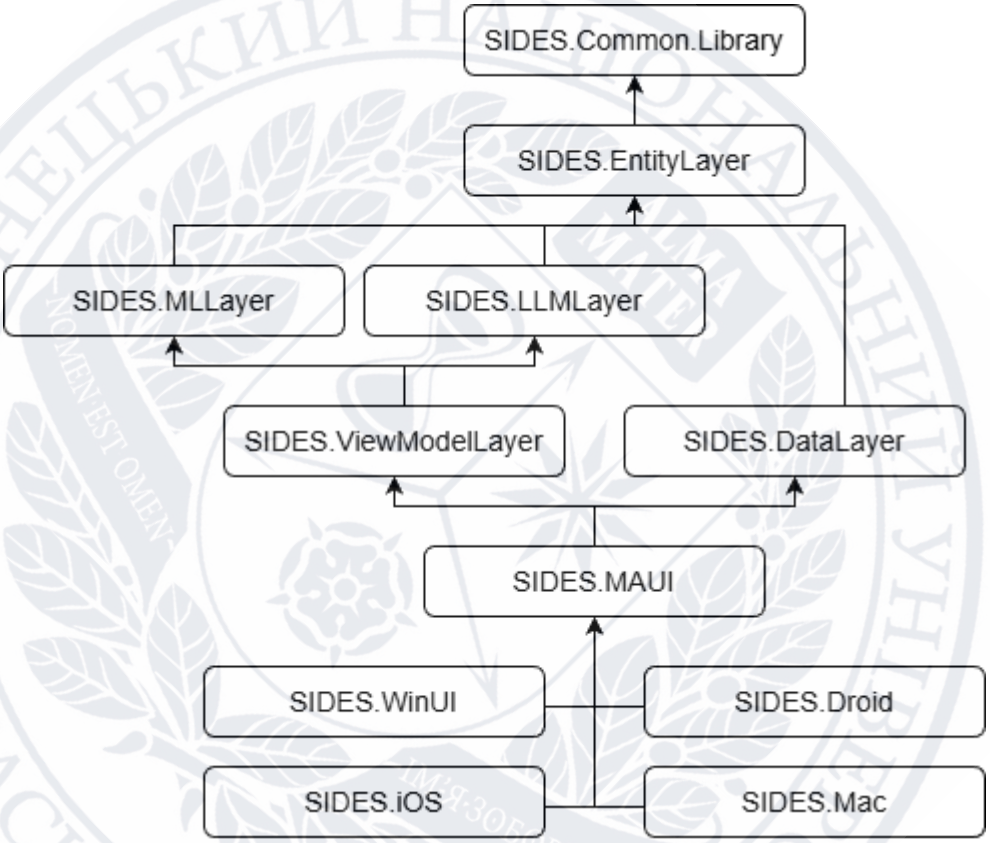

Рисунок 3.1 – Структура залежності проєктів у застосунку

3.2 Реалізація інтерфейсу рекомендаційної системи

Інтерфейс рекомендаційної системи реалізовано на основі фреймворку .NET MAUI із застосуванням архітектурного патерну MVVM. Усі екрани побудовані з використанням XAML розмітки, тоді як логіка взаємодії реалізована у відповідних View та ViewModel класах, що реєструються у контейнері залежносте, що забезпечує автоматичне створення та надання необхідних залежностей.

Для навігації між екранами використовується MAUI Shell, де переходи до сторінок здійснюється за допомогою іменованих маршрутів, що задаються рядковими ключами та пов’язаними з ними класами сторінок. Реєстрація маршрутів здійснюється в файлі AppShell.xaml.cs.

На основі вимог, визначних під час проєктування системи, було побудовано інтерфейс користувача, який охоплює всі основні сценарії взаємодії з рекомендаційною системою. Під час формування UI враховано послідовність проходження діагностики та логічні переходи між сторінками. Кожна сторінка відповідає окремому етапу роботи системи та має чітко визначену зону відповідальності.

На рисунку 3.2 наведено загальну схему зв’язків і переходів між сторінками застосунку. Основні сторінки, що були реалізовані:

- Головний сторінка. Стартова точка взаємодії з кнопкою запуску діагностики;
- Модуль питань. Сторінка, яка використовується для відображення запитання з варіантами відповідей;
- Сторінка діагностики. Інформує про перебіг аналізу системи;
- Сторінка результатів. Дана сторінка містить три інші сторінки. Одна відповідає за виведення підсумкової інформації діагностики, а саме потенційні несправності та рекомендації. Друга та третя з’являється у користувача, якщо він погоджується на використання великих мовних моделей. Друга виводить перетворені рекомендацій діагностичної сесії на зрозумілі текстові інструкції з поясненням, а третя - результати діагностики від мовної моделі;
- Сторінка налаштувань. Дозволяє користувачеві керувати дозволами та параметрами роботи рекомендаційної системи;
- Сторінка з історією минулих діагностичних сесій. Містить перелік попередніх сесій і надає користувачеві можливість переглянути результати повторно;
- Сторінка оцінювання рекомендацій. Оцінка якості наданих рекомендацій.

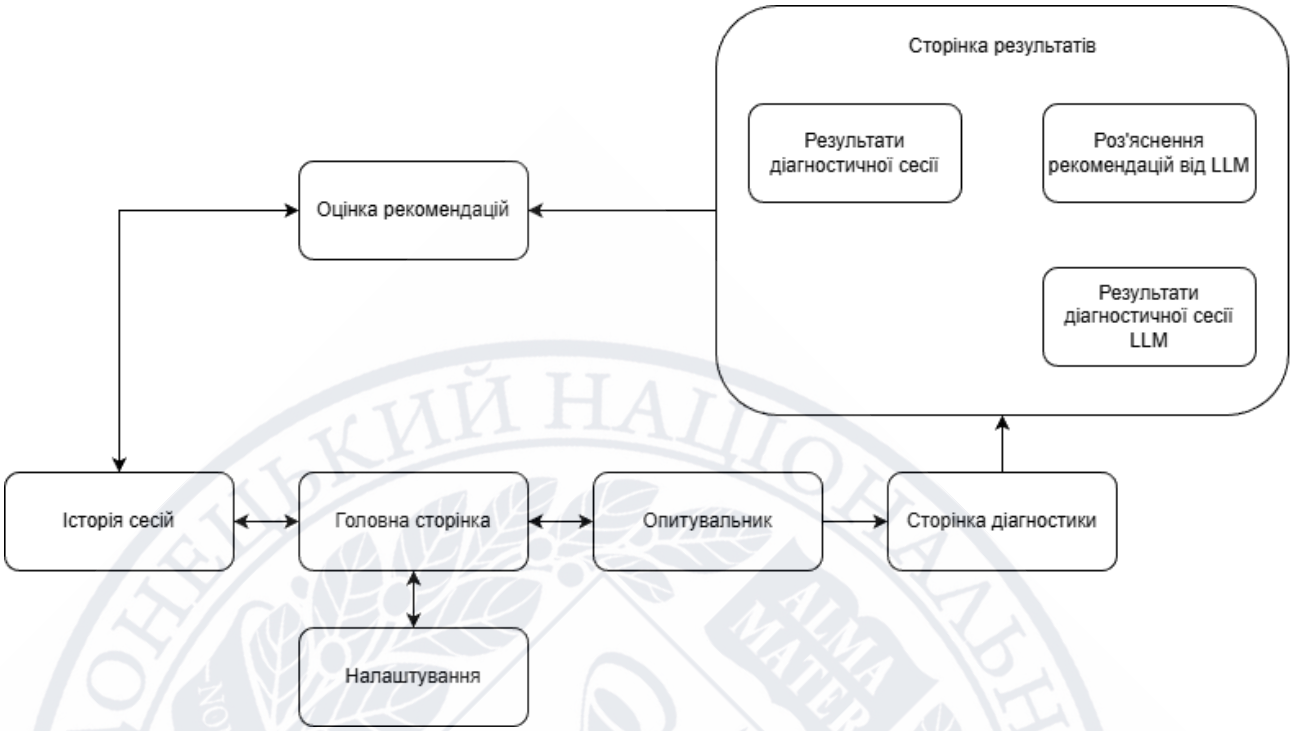

Рисунок 3.2 – Схема зв’язків і переходів сторінок у застосунку

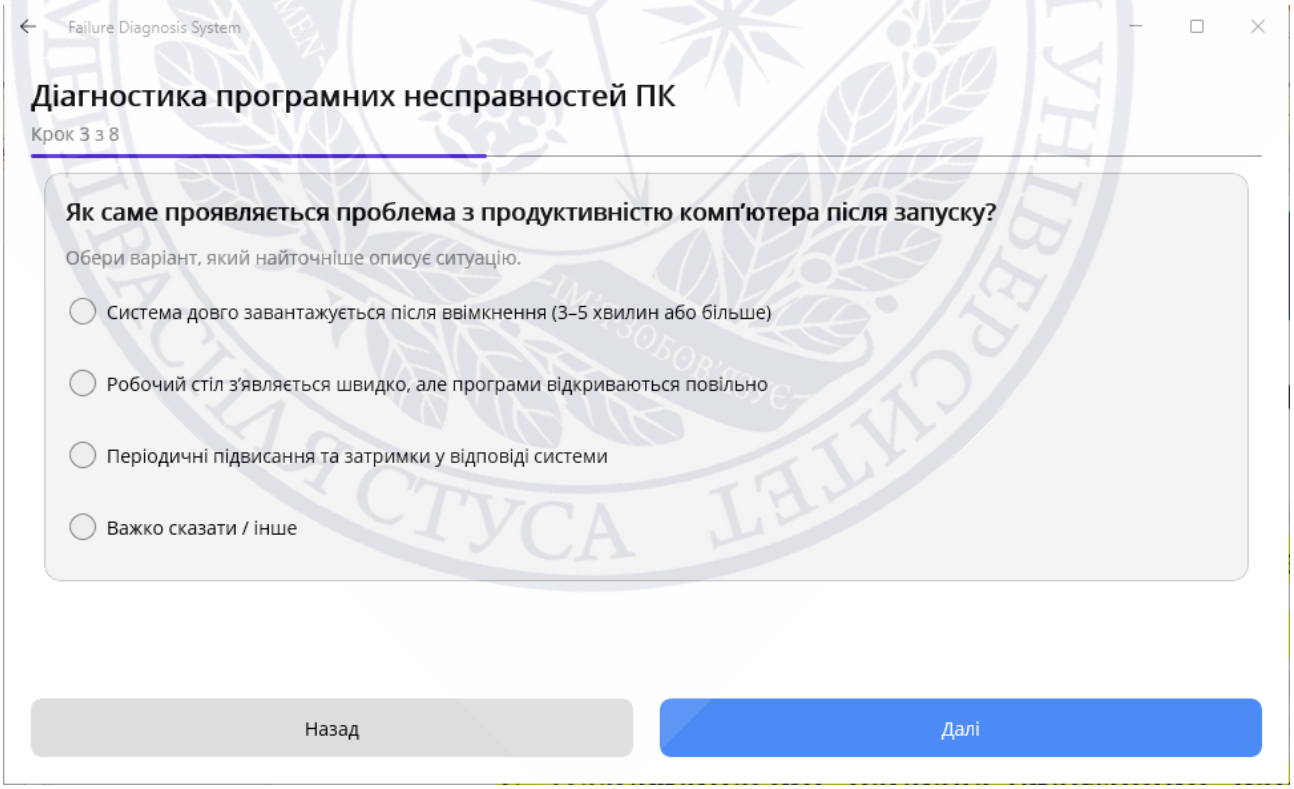

Рисунок 3.3 – Приклад запитання на сторінці опитувальника

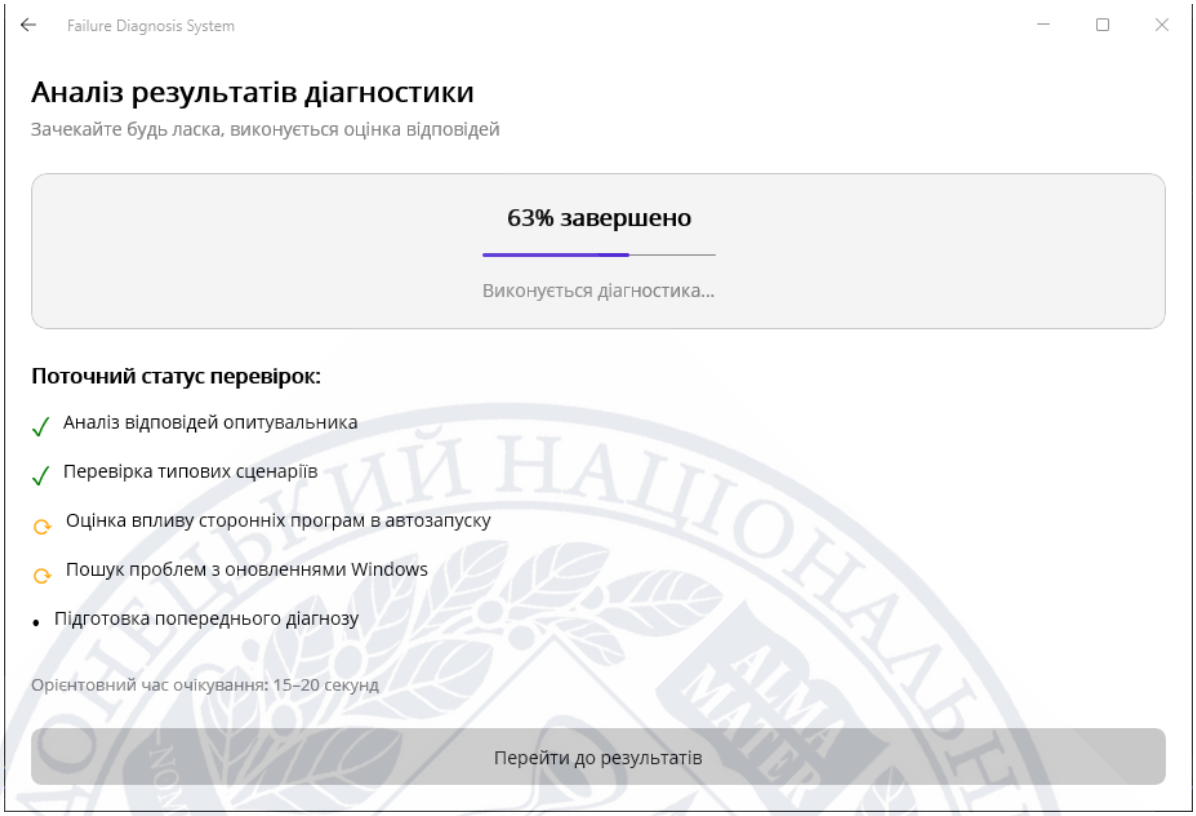

Рисунок 3.4 – Приклад сторінки діагностики

3.3 Збереження даних у PostgreSQL через EF Core

Для зберігання даних рекомендаційної системи було використано реляційну базу даних PostgreSQL, розгорнуту у контейнеризованому середовищі Docker Desktop. Такий підхід дозволяє швидко отримати робочу інфраструктуру, що не залежить від локальних налаштувань операційної системи та може бути відтворена на будь-якому комп’ютері [50, 51]. Дані переваги від контейнеризації спрощують розробку так, як кожен учасник проєкту використовуватиме однакові налаштування та середовище локальної бази даних, що унеможливлює появу конфліктів компонентів, помилки конфігурації та проблеми через різницю у середовища запуску бази даних.

База даних запускалася за допомогою файлу docker-compose.yml, який містить конфігурацію двох сервісів: сервера PostgreSQL та інструмента адміністрування pgAdmin 4. Після запуску контейнерів PostgreSQL працює як повноцінний ізольований сервер, доступний через локальний порт. Всі дані бази зберігаються у спеціально визначеному томі Docker, завдяки чому дані не

втрачається після перезапуску контейнера [51]. Нижче наведений код файлу docker-compose.yml, який використовувався для підняття контейнерів в середовищі Docker Desktop:

```
services:
  database:
    image: postgres:15
    container_name: my_postgres
    restart: always
    ports:
      - "5555:5432"
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
      POSTGRES_HOST_AUTH_METHOD: trust
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - pgnet

  pgadmin:
    image: dpage/pgadmin4
    container_name: my_pgadmin
    restart: always
    ports:
      - "5050:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@example.com
      PGADMIN_DEFAULT_PASSWORD: admin
    networks:
      - pgnet

volumes:
  postgres_data:

networks:
  pgnet:
```

Середовище Docker дозволяє уникнути мануального встановлення PostgreSQL так, як контейнер містить попередньо налаштований екземпляр сервера, який можна відразу використовувати для підключення із застосунку за допомогою EF Core. Єдине, що необхідно вказати у застосунку, це рядок підключення до локального контейнера.

Другим компонентом конфігурації є pgAdmin 4, який також запускається в контейнері. Це графічна панель керування PostgreSQL через веб-браузер. Він надає зручний інтерфейс для створення таблиць, перегляду схеми/схем бази, аналізу зв'язків між таблицями, редагування даних і виконання SQL запитів [52]. Під час розробки дозволяє легко підключитися до контейнера PostgreSQL і перевірити, як EF Core створює таблиці або виконує міграції [52, 53].

PostgreSQL було обрано як основну систему управління базами даних завдяки її стабільності, підтримці транзакцій, а також можливості зручно працювати із структурованими даними [54-57]. Також одним з важливих факторів вибору PostgreSQL є відповідність вимогам сучасних серверних застосунків, що працюють з великою кількістю запитів на зчитування та запис. Дана база даних підтримує ACID властивості, що гарантують збереження цілісності даних під час одночасного доступу декількох користувачів або при виникненні збоїв у роботі [54-57]. В порівнянні з іншими типами реляційних баз даних PostgreSQL пропонує ширший набір інструментів для побудови складних моделей даних та керуванням реляційними зв'язками. Доступність різних типів індексів, що використовують додаткові механізми для оптимізації, забезпечують високу продуктивність при виконанні запитів навіть за умови зростання обсягів бази даних. Дана перевага є однією з критично важливих для подальшого розширення експертної та користувацької бази даних.

Підключення до бази даних зі сторони застосунку реалізовано за допомогою Entity Framework Core [58]. Для створення таблиць в базі даних використовується підхід Code First. Даний підхід реалізовується у вигляді C# класів, що описують сутності доменної області. EF Core автоматично створює відповідні таблиці на основі цих моделей [59]. Даний підхід забезпечує повну відповідність між моделями застосунку та збереженими даними, що дозволяє швидко змінювати структуру та уникати неузгодженостей між кодом застосунку та базою даних. Зміни в структурі моделей відстежуються через механізм міграцій, які дозволяють генерувати та оновлювати схему PostgreSQL без необхідності мануального написання SQL запитів. Також міграції дозволяють

коректно розгортати базу даних як під час розробки, так і в інших середовищах, гарантуючи узгодженість структури бази даних на всіх етапах життєвого циклу додатку. EF Core також забезпечує виконання CRUD операцій через контекст даних, що дозволяє працювати з таблицями бази на рівні об’єктів і спрощує взаємодію з базою даних. Водночас маємо можливість аналізувати згенеровану структуру та результат виконання запитів за допомогою pgAdmin.

Після розгортання середовища збереження даних у PostgreSQL було реалізовано дві схеми, які відповідають базам даних описаних у другому розділі роботи. Реалізація через схеми дозволяє логічно відокремити та забезпечити майже незалежний розвиток обох баз. На рисунках 3.5, 3.6 та 3.7 наведено ER (Entity–relationship) діаграми користувацької та експертної баз даних, які відображають структуру таблиць, їхні зв’язки та ключові поля. Таблиці сформовано згідно з вимогами нормалізації, які були описані у другому розділі.

Використання схем є оптимальним рішенням контексті реалізації мінімально життєздатного продукту. Схема у PostgreSQL дозволяє чітко розмежувати логічні частини, але при цьому зберегти спільне з’єднання та загальні параметри конфігурації. Це суттєво спрощує адміністрування, резервне копіювання та розгортання середовща, оскільки на даному етапі розробки відсутня потреба у підключенні до двох незалежних баз даних. Також така організація є продуктивною, оскільки всі дані знаходяться в одній фізичній базі, але ізольовані на рівні схем.

У застосунку було створено два незалежних контексти Entity Framework Core, а саме ExpertDbContext та UserDbContext. Перший відповідає за експертну базу даних, а другий – користувацьку базу даних. Обидва контекста використовують спільний рядок підключення до PostgreSQL, але призначаються різні схеми через HasDefaultSchema(). Поділ на два контексти дозволяє незалежно моделювати структуру обох баз даних так, як кожен контекст має власний набір міграцій.

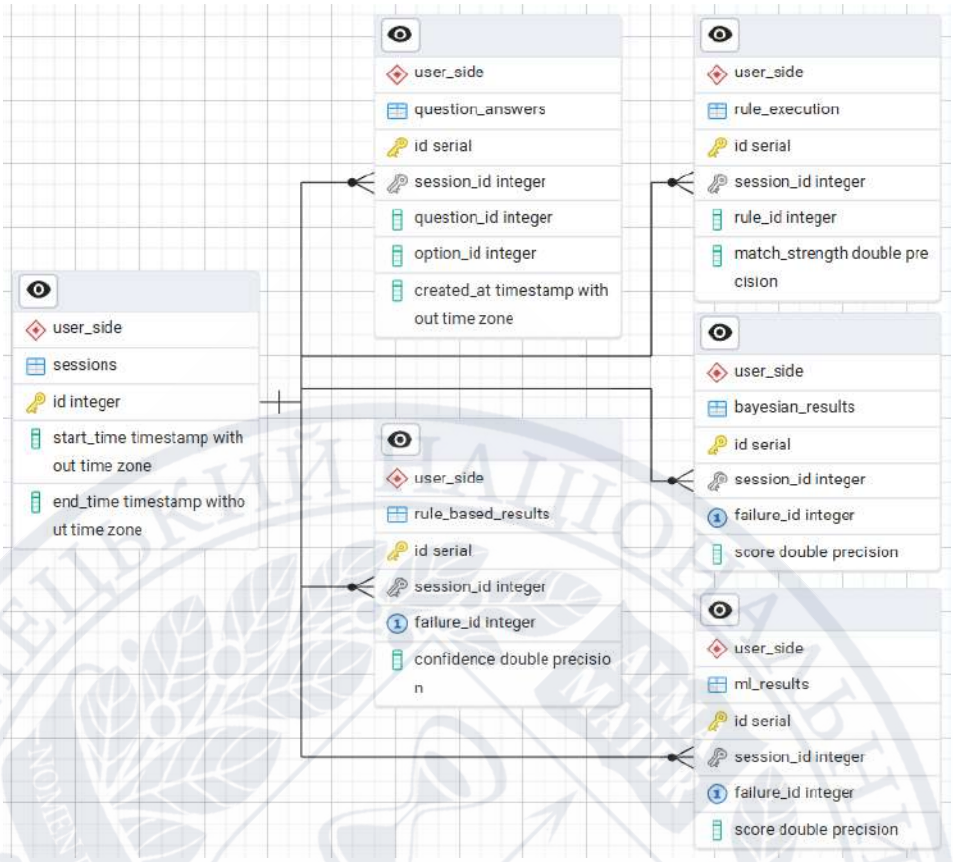

Рисунок 3.5 – Структура користувацької бази даних

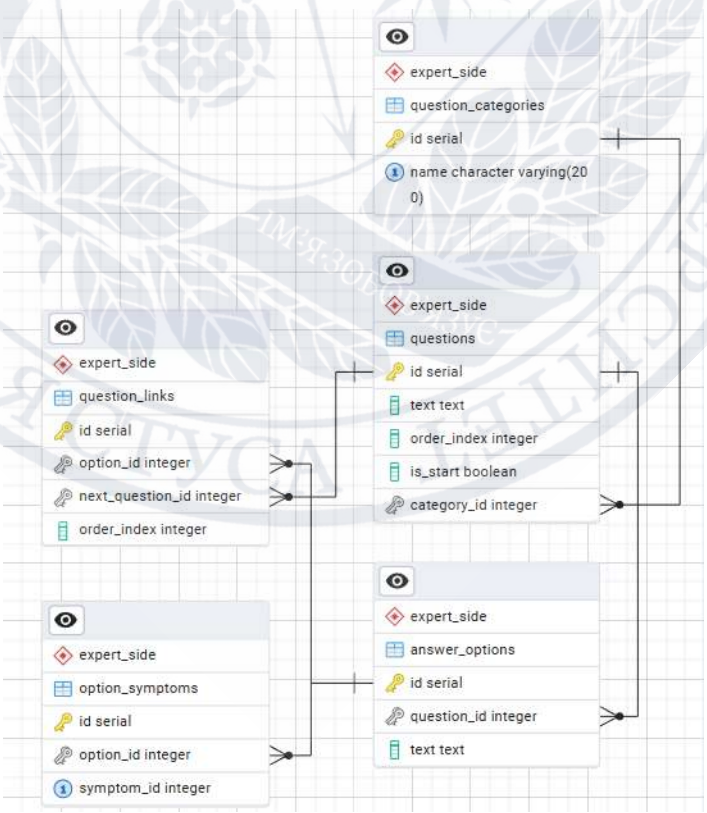

Рисунок 3.6 – Структура експертної бази даних. Таблиці відносно запитань

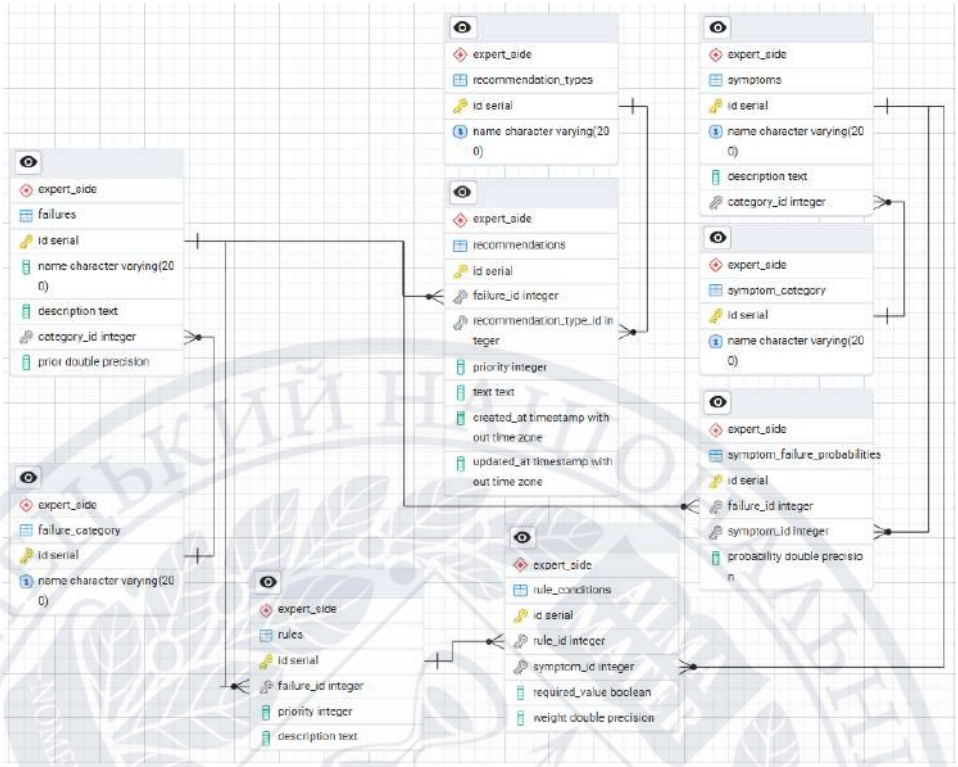

Рисунок 3.7 – Структура експертної бази даних. Таблиці несправностей, симптомів, рекомендацій та таблиці для діагностики

3.4 Отримання вхідних даних і формування вектора симптомів несправності

До того як використовувати трьохрівневу аналітичну підсистему рекомендаційна система повинна сформувати набір вхідних даних з якими працюватимуть діагностичні алгоритми. У реалізації проєкту це означає побудову вектора симптомів, що містить кількість симптомів N, які можна отримати за допомогою унікального ключа SYMPTOM_ID з таблиці SYMPTOPMS. Таблиця SYMPTOPMS містить перелік можливих симптомів та кожен симптом має назву, опис та категорію. Ключовим полем є SYMPTOM_ID, яке надалі використовується у всіх діагностичних рівнях.

Формування вектора симптомів реалізовано на основі декількох джерел, які були описані в другому розділі, а саме інтерактивного опитувальника та системних модулів збору і аналізу технічної інформації. Блок взаємодії з користувачем реалізований за допомогою таблиць: QUESTIONS,

QUESTIONS_CATEGORIES, ANSWER_OPTIONS, QUESTION_LINKS та OPTION_SYMPTOMS. Дані таблиці містять інформацію по кожному сформованому запитанню, потенційних відповідей до запитання, прив’язку відповідей до симптомів з таблиці SYMPTOPMS за допомогою таблиці OPTION_SYMPTOMS та прив’язку відповідей до наступних можливих уточнюючих запитань. У рамках реалізації другого джерела вхідних даних збір та аналіз технічних даних робиться під кожну платформу окремо, але основний підхід формування даних залишається незмінним. Зібрані технічні дані проходять через аналітичний блок та перетворюють їх у симптоми з таблиці SYMPTOPMS. Наприклад, при виявленні високої температури центрального процесору, система записує симптом CPU_OVERHEAT. У аналітичному блоці порогові значення та умови трансформування технічних значень не задаються довільно, а визначаються на основі офіційної документації виробників, стандартів моніторингу і інших офіційних джерел. Використання аналітичного блоку дозволяє перетворювати неструктуровані системні дані у нормалізований набір симптомів.

Після обробки вхідних даних з різних джерел формується остаточний вектор симптомів SYMPTOM_ID, який виступає базовим набором ознак для подальшої діагностики. Усі операції прив’язані до конкретної діагностичної сесії, ідентифікатор якої зберігається у таблицю SESSIONS, що дозволяє потім аналізувати ефективність діагностики. Надалі ідентифікатор використовується для зберігання отриманого вхідного вектору симптомів.

3.5 Реалізація рівня правил

Рівень правил є першим етапом алгоритму діагностики, що відповідає за формування переліку потенційних несправностей. Реалізація цього рівня базується на експертних правилах, які зберігаються у схемі EXPERT_SIDE та описуються у таблицях RULES, RULE_CONDITIONS, які в свою чергу зв’язані з таблицями SYMPTOMS та FAILURES.

Кожне правило представлено записом у таблиці RULES. Правило має прив’язку до конкретної несправності, а саме поля FAILURE_ID з таблиці FAILURES, та може містити довільну кількість умов, які описуються записами в таблиці RULE_CONDITIONS. Умови правил містять наступні поля:

- SYMPTOM_ID – ідентифікатор симптому, який має бути присутнім або відсутнім в залежності від значення REQUIRED_VALUE;
- REQUIRED_VALUE – булевий показник, що вказує на те чи повинен бути присутній симптом;
- WEIGHT – вага умови, що дозволяє регулювати вклад кожного симптому у фінальну оцінку правила.

Завдяки цьому механізму правила мають характер логічних виразів типу:

Якщо (Симптом А присутній) і (Симптом В присутній) і (Симптом С відсутній), то це пов’язано з несправністю Х.

Таким чином, система надає можливість описувати як прості, так і складні залежності між множиною симптомів та відповідною несправністю, що підвищує точні рівня правил.

Після формування вектора симптомів система починає послідовну перевірку всіх правил. Для кожного правила обчислюється ступінь відповідності умовам, тобто відсоток виконання умов правила. Обрахунок показника робиться за наступним алгоритмом:

1. Для поточного правила отримуються всі пов’язані умови;
2. Для кожної умови перевіряється присутні відповідного симптому SYMPTOM_ID у вхідному векторі симптомів та чи збігається значення REQUIRED_VALUE;
3. Якщо умова виконується, додається її вага WEIGHT;
4. Робиться обрахунок коефіцієнта відповідності правила до вхідного вектором за формулою:

$$matched\_strength = \frac{\sum_{i=1}^{n} w_i^{matched}}{\sum_{i=1}^{n} w_i} \quad (3.1)$$

де $w_i^{matched}$ – вага i-ї умови правила у якого збігається показник REQUIRED_VALUE;

w_i – вага i-ї умови правила;

n – загальна кількість умов правила;

$matched\_strength$ - коефіцієнт відповідності правила.

Далі зі всіх правил, що мають значення коефіцієнта відповідності більше 0, виконується запис даних у таблицю RULE_EXECUTION для можливості відтворення результатів діагностики рівня правил та аналітики впливу правил на результати діагностики. Записи мають наступну структуру:

- SESSION_ID – ідентифікатор діагностичної сесії з таблиці SESSIONS;
- RULE_ID – ідентифікатор виконуваного правила, що знаходиться в таблиці RULES;
- MATCH_STRENGTH – коефіцієнт відповідності наявних симптомів до правила.

Коли всі правила виконанні, система переходить до агрегування результатів. Для кожного правила, яке має позитивну відповідність, визначається несправність FAILURE_ID, яка пов’язана з цим правилом. Далі робиться запис визначених несправностей у таблицю RULE_BASED_RESULTS. Записи в даній таблиці мають наступну структуру:

- SESSION_ID – ідентифікатор діагностичної сесії з таблиці SESSIONS;
- FAILURE_ID – ідентифікатор несправності з таблиці FAILURES;
- CONFIDENCE – показник ймовірності несправності. Розрахунок показника виконується за формулою:

$$confidence(F) = \frac{\sum_{i=1}^{n}(match\_strength_i * priority_i)}{\sum_{i=1}^{n} priority_i} \quad (3.2)$$

де F – несправність;

$match\_strength_i$ – коефіцієнт відповідності i -го правила;

$priority_i$ – пріоритет i-го правила;

n – загальна кількість правил пов'язаних з несправністю F;

$confidence$ – показник ймовірності несправності.

Варто зазначити, що під час розрахунку показника ймовірності несправності, додатково у таблицю RULE_EXECUTION роблять збереження правил, які мали відповідність до несправності рівною 0. Це робиться для забезпечення відтворюваності діагностики навіть при зміні правил та їх умов у експертній базі даних.

3.6 Реалізація Байєсівського рівня

Байєсівський рівень є другим етапом діагностики, що виконує уточнення оцінок, отриманих на рівні правил, та дозволяє визначити наскільки кожна з відібраних несправностей узгоджується з вхідним вектором симптомів. Якщо рівень правил працює з фіксованими логічними виразами, то Байєсівський рівень використовує статистичний підхід, що враховує як ймовірність появи симптомів при певній несправності, так і загальну частоту виникнення цієї несправності в реальних умовах.

Для реалізації ймовірнісного підходу експертна база знань містить два типи параметрів. Перший це апріорна ймовірність несправності, що безпосередньо зберігається у колонці PRIOR таблиці FAILURES. Це значення відображає загальну тенденцію до виникнення певної несправності й задається спочатку експертно, а надалі змінюється з урахуванням статистики збережених діагностичних сесій користувачів. Значення не нормалізується при записі, так як вона виконується на фінальному етапі розрахунків.

Другий тип даних це умовні ймовірності симптомів відносно несправностей, що описані у таблиці SYMPTOM_FAILURE_PROBABILITIES. Запис складаються з ідентифікаторів симптомів, ідентифікаторів несправностей та значення ймовірності. Кожен запис описує ймовірність проявлення певного

симптому при певній несправності. Ці значення так само напочатку визначаються експертами і надалі оновлюються з урахування діагностичних сесій користувачів. Разом з апріорними оцінками вони формують повний набір даних, необхідний Байєсівському рівню.

Під час обчислення система розглядає лише ті симптоми, які пов'язані з конкретною несправністю та мають записи у таблиці SYMPTOM_FAILURE_PROBABILITIES. Таким чином для кожної несправності визначається множина лише релевантних симптомів, щоб інші симптоми не впливали на значення ймовірності. Релевантні симптоми поділяються на активні та неактивні. Активними вважаються ті, які є у вхідному векторі симптомів, а неактивними ті, що описанні в таблиці SYMPTOM_FAILURE_PROBABILITIES, але не були виявлені під час діагностування. Обчислення виконується для несправностей, які були визначені рівнем правил як потенційні, за спрощеною формулою, яка побудована на релевантній множині симптомів:

$$Posterior(F_i) = P(F_i) * \prod_{s_j \in S} P(s_j|F_i) * \prod_{s_k \in S^-} P(1 - (s_k|F_i)) \qquad (3.3)$$

де F_i – i -та несправність, для якої виконується обчислення;

$P(F_i)$ – апріорна ймовірність несправності F_i;

S – множина релевантних із наявних симптомів несправності F_i;

S^- – множина релевантних із відсутніх симптомів несправності F_i;

$P(s_j|F_i)$ – умовна ймовірність, того що симптом s_j з'являється у випадку несправності F_i;

$P(1 - (s_k|F_i))$ – ймовірність того, що симптом s_k, який типовий для несправності, не проявився у поточній діагностичній сесії;

$Posterior(F_i)$ – зважена не нормалізована оцінка, яка відображає узгодженість симптомів релевантних несправності F_i.

Оскільки Байєсівський рівень працює як продовження рівня правил, отримане значення комбінується з початковою оцінкою, отриманою на попередньому етапі:

$$Final(F_i) = Posterior(F_i) * RB(F_i) \quad (3.4)$$

де F_i – i -та несправність, для якої обраховується підсумкова оцінка;

$Posterior(F_i)$ – постеріорна ймовірність несправності F_i;

$RB(F_i)$ – показник ймовірності несправності F_i, яка отримана з результатів рівні правил;

$Final(F_i)$ – підсумкова оцінка несправності F_i.

Після цього значення нормалізуються, утворюючи підсумковий набір ймовірностей, які відображають узгодженість кожної несправності з наявними симптомами. Результати розрахунків зберігаються у таблиці BAYESIAN_RESULTS. Записи в таблиці містять ідентифікатор діагностичної сесії, ідентифікатор несправності та нормалізовану ймовірність відносно отриманих під час діагностики симптомів.

3.7 Інтеграція з ML.NET

Машинне навчання є завершальними рівнем діагностики та використовується для уточнення результатів, отриманих на попередніх етапах. На відміну від попередніх двох рівнів, які працюють в основному використовують експертні знання для формування результатів діагностики, рівень машинного навчання використовує накопичену історичну інформацію про попередні діагностичні сесії. Основною метою цього етапу діагностики є виявлення закономірностей, які можуть бути непомітними у формальних правилах або у ймовірнісні моделі, та підвищення точності діагнозу завдяки статистичному узагальненню.

Навчання моделі відбувається централізовано на сервері, де зберігається історія всіх завершених діагностичних сесій користувачів. На сервері з цих сесій формується навчальний набір: вхідний вектор симптомів, результати рівня правил та Байєсівського рівня і підтверджений користувачем фінальний діагноз. Після оновлення машинна модель експортується у вигляді файлу та доставляється клієнтському застосунку разом із черговим оновленням. Оскільки всі дані, необхідні для формування ознак вже містяться базі, система не дублює їх у вигляді окремих агрегованих таблиці.

Під час діагностичної сесії рівень машинного навчання отримує на вхід такий самий набір даних, що і навчальний, за виключенням підтвердженого користувачем фінального діагнозу. Далі машинна модель повертає прогнозовані ймовірності кожної потенційної несправності. Рівень машинного навчання працює як додатковий механізм уточнення, використовуючи накопичену статистику з попередніх випадків.

Результати роботи машинного рівня зберігаються у таблиці ML_RESULTS. Записи в таблиці містять наступні дані: ідентифікатор діагностичної сесії, ідентифікатор несправності та ймовірність, що несправність є коректною.

3.8 Адаптер для інтеграції з LLM

Для розширення функціональності системи та підвищення якості рекомендацій після завершення основних етапів діагностики було створено адаптер для інтеграції з великими мовними моделями. Адаптер працює як окремий модуль, який не впливає на логіку прийняття рішень основними рівнями діагностики. Головною метою адаптера є перетворення структурованих рекомендацій діагностичної сесії на зрозумілі текстові інструкції з поясненням.

Окрім цього адаптер також виконує окремий аналіз на основі отриманих даних. Велика мовна модель формує власне припущення про можливу причину несправності, використовуючи той вхідний вектор симптомів, технічних метрик та відповідей, що основний алгоритм діагностики.

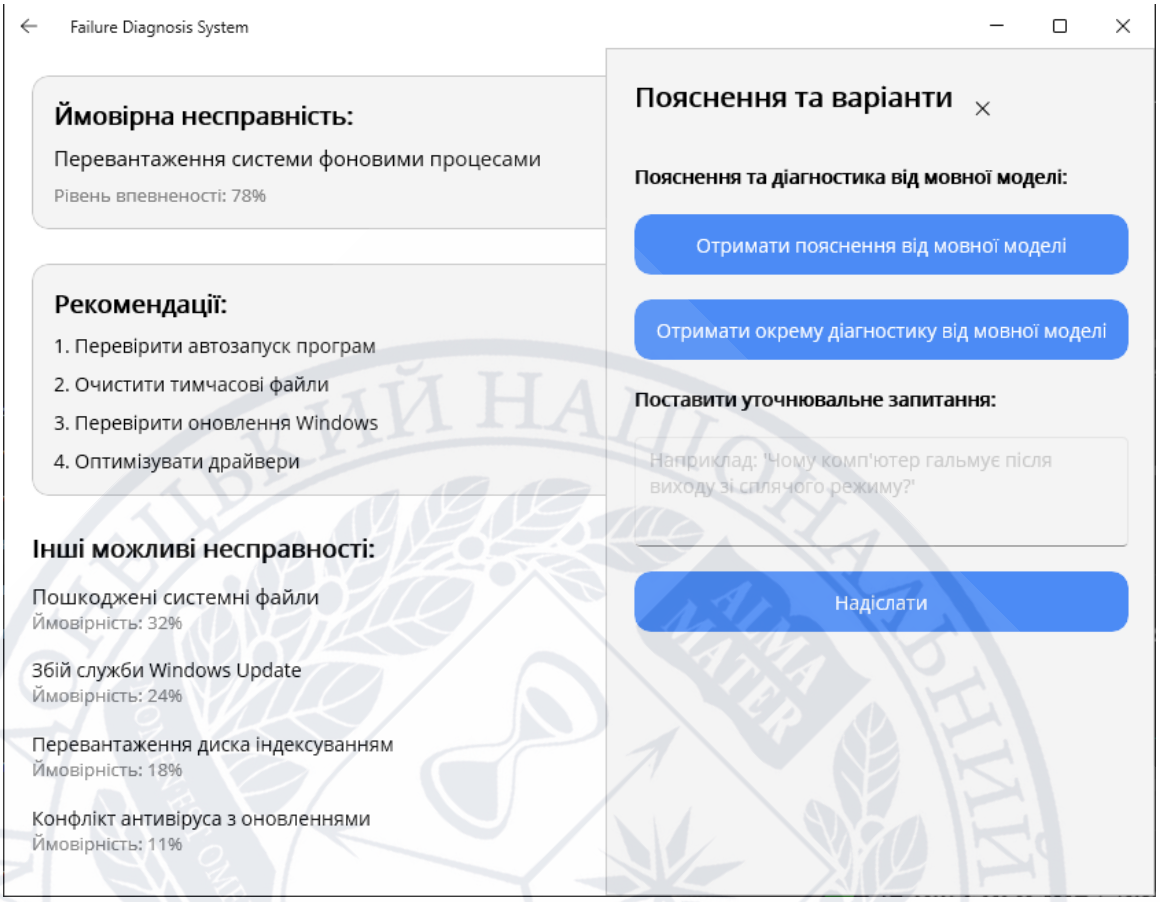

Рисунок 3.8 – Інтерфейс результатів діагностики, де в правій частині інтерфейс адаптера мовних моделей

В реалізації рекомендаційної системи адаптер функціонує лише як допоміжний модуль, тобто він не змінює результати діагностики та не втручається в логіку надання рішення. Даний функціонал є відокремленим на екрані результатів і тільки виконується зі згоди користувача. На рисунку 3.8 показано інтерфейс адаптера на екрані результатів.

3.9 Інтеграція з іншою рекомендаційною системою

Також у межах дослідження було розглянуто можливість інтеграції діагностичного алгоритму застосунку з додатковою рекомендаційною системою. Основна мета додатковою системи є поглиблений аналіз користувацької взаємодії. Було розроблено архітектуру, що обидві системи в одному застосунку.

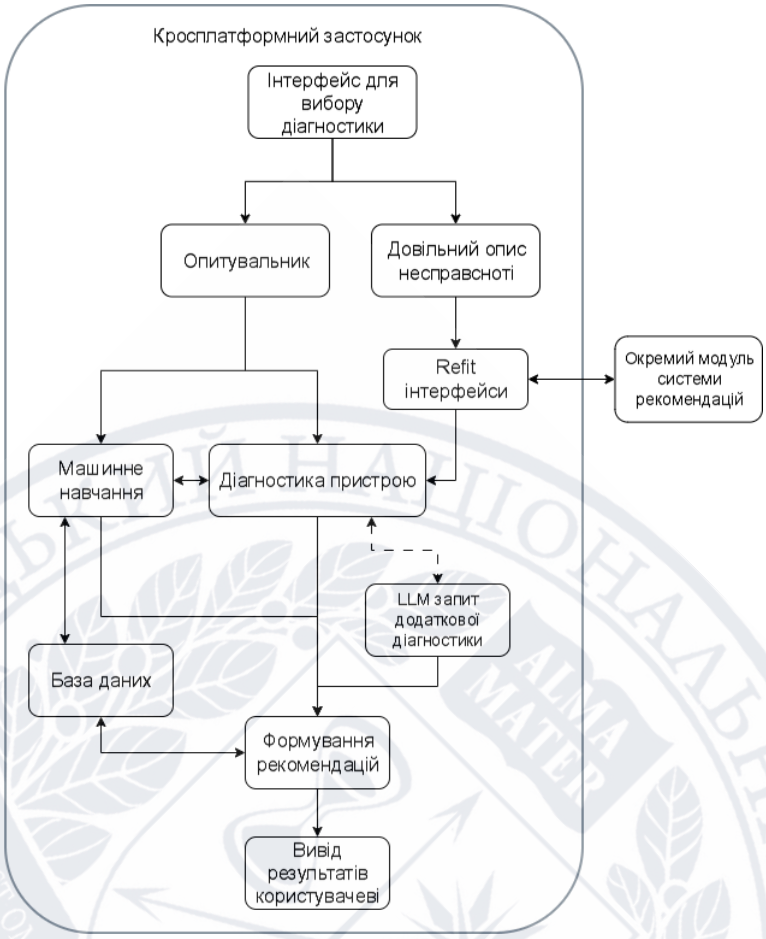

Рисунок 3.9 – Взаємодії кросплатформного застосунку з модулем аналізу користувацької взаємодії

Розроблена концепція передбачає використання двох автономних підсистем. Взаємодія між ними здійснюється через стандартизований REST API. Основний застосунок передає до сервісу узагальнений результат діагностики, а аналітичний сервіс використовуючи доступ до єдиної бази знань, здійснює вибір релевантних рекомендацій. Після чого повертає структуровану відповідь у форматі, придатному для відображення в інтерфейсі застосунку. На рисунку 3.9 наведено схему взаємодії кросплатформного застосунку з модулем аналізу користувацької взаємодії.

У перспективі інтеграція дозволяє формувати комплексну інтелектуальну систему, здатну враховувати технічні показники та поведінкові особливості користувача.

3.10 Приклад діагностичної сесії

Для демонстрації роботи рекомендаційної системи розглянемо приклад проходження діагностичної сесії користувачем. Наведений приклад охоплює всі основні етапи від запуску діагностики до отримання підсумкових рекомендацій.

Після відкриття застосунку користувач переходить на головну сторінку, де має можливість розпочати діагностику натиском на відповідну кнопку.

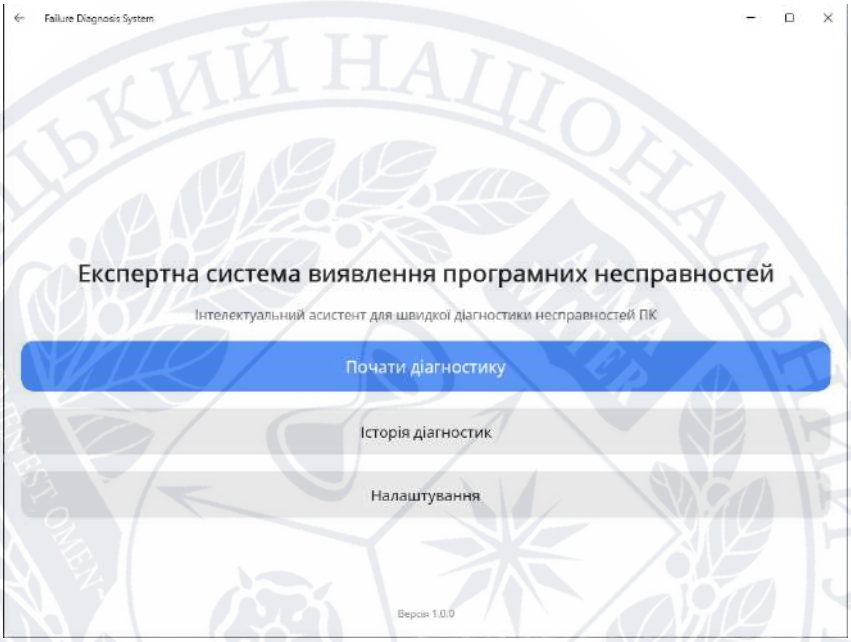

Рисунок 3.10 – Інтерфейс головного екрану

Після запуску діагностики система послідовно задає запитання, що беруться з експертної бази знань. З обраних користувачем відповідей формується вектор симптомів та вибір наступних запитань. Паралельно з проходженням опитувальна виконується технічна діагностика персонального комп’ютера. Якщо користувач надав відповіді на всі запитання раніше ніж завершиться технічна діагностика та виконання основного алгоритму діагностики, то йому відображається інтерфейс поточним станом виконання етапів основного алгоритму.

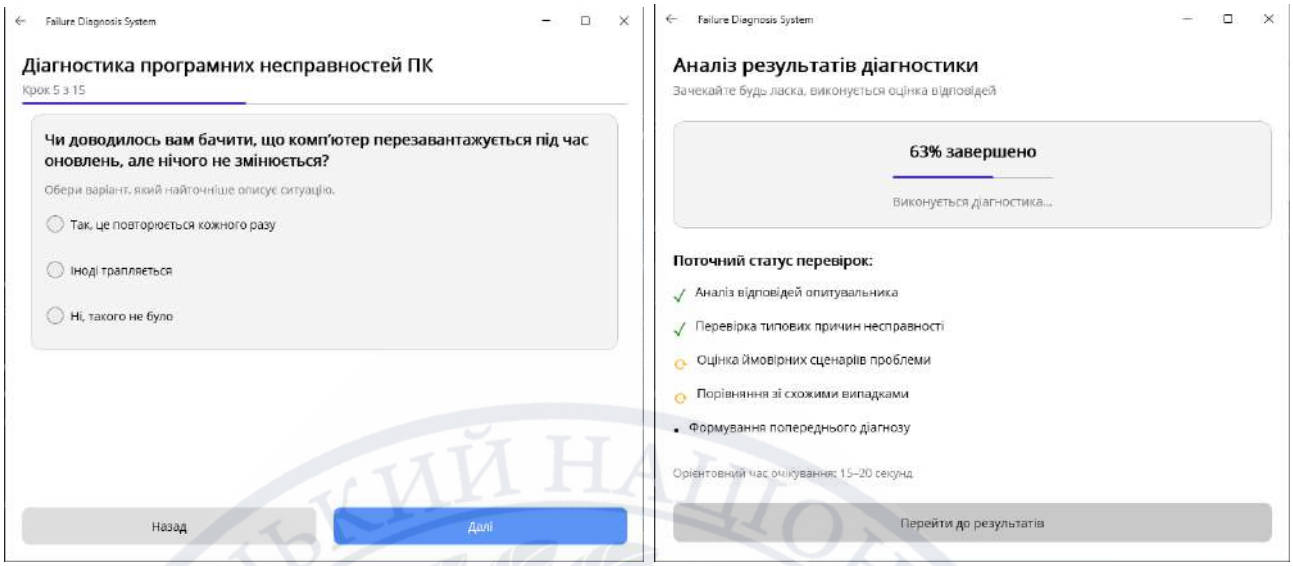

Рисунок 3.11 – Інтерфейси опитувальника та поточного стану діагностики

Після завершення усіх етапів діагностики користувачеві показується інтерфейс з результатами, де користувач може переглянути потенційні несправності, рекомендації щодо їх усунення. За бажанням користувач може отримати розширені пояснення та окрему діагностику сформовану великою мовною моделлю. Після ознайомлення з результатами користувачеві надається можливість оцінити якість отриманих рекомендацій.

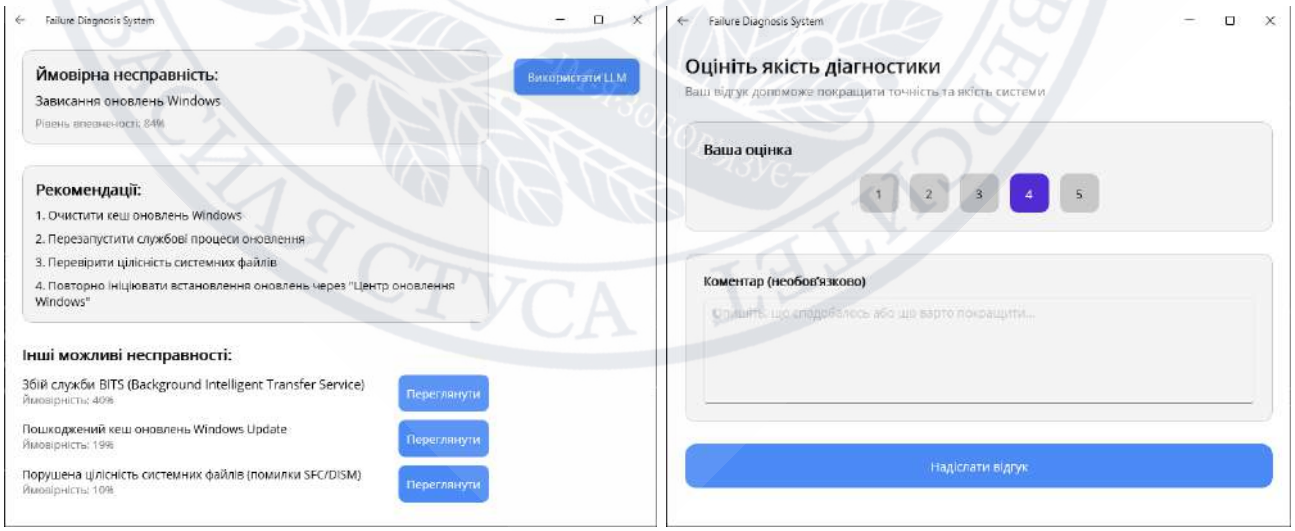

Рисунок 3.12 – Інтерфейси з результатами діагностики та інтерфейс для оцінки якості рекомендацій

3.11 Подальші напрями розвитку рекомендаційної системи

Наявна реалізація рекомендаційної системи може бути розширена та вдосконалена за кількома напрямами, що дозволить підвищити її точність, масштабованість, гнучкість та зручність використання. Подальший розвиток має охоплювати як технічні, так і аналітичні аспекти, забезпечуючи поступове перетворення системи на повноцінний інтелектуальний інструмент діагностики несправностей.

Одним із перспективних напрямів є застосування кластеризації типових проблем на основі накопичених сесій. Аналіз частоти симптомів, подібності сценаріїв та виявлення повторювальних патернів дасть можливість знаходити нові групи несправностей або покращувати інформацію щодо існуючих. Це сприятиме покращенню роботи ймовірнісного рівня та допоможе системі адаптуватися до появи нових несправностей.

Окрему увагу може бути приділено персоналізації діагностики, тобто система буде адаптувати рекомендації під конкретного користувача, історію попередніх звернень. Такий підхід надасть можливість для надання більш точних, індивідуалізованих порад, що підвищить практичну корисність системи.

Перспективним напрямком розвитку також є використання локальних мовних моделей, здатних працювати без підключення до сторонніх сервісів. Це дозволить забезпечити вищий рівень приватності даних, прискорить взаємодію та зменшить залежність від зовнішніх API.

Для розширення аудиторії та покращення доступності система може отримати веб-версію, яка надасть можливість користуватися нею з будь-якого пристрою без потреби встановлення програмного забезпечення.

Ще одним перспективним напрямком розвитку є інтеграція методів навчання з підкріпленням. За допомогою яких система зможе вдосконалювати свої рекомендації на основі зворотного зв’язку від користувачів. Поступове накопичення інформації про корисність запропонованих порад дозволить системі навчатися на реальних взаємодіях, що дозволить підвищити точність і релевантність діагностики в майбутньому.

Подальший розвиток системи може охоплювати широкий спектр напрямів. Реалізація навіть частини зазначених можливостей має посприяти зростанню функціональності, гнучкості та надійності рекомендаційної системи.

Висновок до розділу 3

В третьому розділі реалізовано програмну частину рекомендаційної системи відповідно до визначених вимог і проєктних рішень. Створено багатопроєктну архітектуру MAUI-застосунку з використанням архітектурних патернів. Реалізовано інтерфейс користувача, механізм збору інформації та алгоритм діагностики. Налаштовано роботу з PostgreSQL за допомогою EF Core. Крім того розглянуто можливість інтеграції з зовнішньою рекомендаційною системою, надано приклад діагностичної сесії та розглянуто подальші напрями розвитку рекомендаційної системи.

ВИСНОВКИ

У роботі зроблено огляд сучасних підходів до діагностики несправностей персональних комп’ютерів. Розглянуто класифікацію типових несправностей, принципи роботи експертних та рекомендаційних систем, а також можливості застосування машинного навчання і великих мовних моделей для підвищення ефективності діагностики та надання рекомендацій.

На основі аналізу предметної області було визначено вимоги до рекомендаційної системи та було обрано доцільний стек технологій. Розроблено архітектуру багатопроєктного застосунку, спроєктовано структуру бази даних, а також описано логіку алгоритму діагностики несправностей, надання рекомендацій та анонімізації даних.

Реалізовано мінімально життєздатний продукт у вигляді застосунку, створеного з використанням фреймворку .NET MAUI. Створено інтерфейс користувача, що включає систему опитування, екрани результатів та інші елементи взаємодії. Також реалізовано модулі збору інформації, трьохрівневий алгоритм діагностики та надання рекомендацій. Налаштовано роботу бази даних PostgreSQL за допомогою Entity Framework Core. Також інтегровано адаптер для інтеграції з великими мовними моделями, що дозволяє виконувати додаткову діагностику на основі зібраних даних та перетворювати структуровані рекомендації системи на зрозумілі текстові інструкції з поясненням.

У процесі виконання кваліфікаційної роботи всі поставлені завдання дослідження були виконані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Sad Tragic Death of Norton Utilities. URL: https://newsome.org/2006/01/13/sad-tragic-death-of-norton-utilities/ (дата звернення: 08.10.2025)
2. WinWorld. QAPlus 6.x. URL: https://winworldpc.com/product/qaplus/6x (дата звернення: 08.10.2025)
3. Said, Nawel & Mansouri, Majdi & Hmouz, Rami & Khedher, Atef. Deep Learning Techniques for Fault Diagnosis in Interconnected Systems: A Comprehensive Review and Future Directions. Applied Sciences. URL: https://www.researchgate.net/publication/392340662_Deep_Learning_Techniques_for_Fault_Diagnosis_in_Interconnected_Systems_A_Comprehensive_Review_and_Future_Directions (дата звернення: 08.10.2025)
4. Robert K. Lindsay, Bruce G. Buchanan, Edward A. Feigenbaum, Joshua Lederberg. DENDRAL: a case study of the first expert system for scientific hypothesis formation. URL: https://web.mit.edu/6.034/www/6.s966/dendral-history.pdf (дата звернення: 08.10.2025)
5. ПРИКЛАДИ ВІДОМИХ ЕКСПЕРТНИХ СИСТЕМ. URL: https://stud.com.ua/158235/informatika/prikladi_vidomih_ekspertnih_sistem (дата звернення: 08.10.2025)
6. James S. Bennett, Clifford R. Hollander. DART: An Expert System for Computer Fault Diagnosis. URL: https://www.ijcai.org/Proceedings/81-2/Papers/050.pdf (дата звернення: 08.10.2025)
7. Kleer, Johan & Brown, John. Model-based diagnosis in SOPHIE III. URL: https://www.researchgate.net/publication/251183238_Model-based_diagnosis_in_SOPHIE_III (дата звернення: 08.10.2025)
8. Нескородєва Т.В., Федоров Є.Є., Січко Т.В., Нескородєва А.Р.. Експертні та рекомендаційні системи, 2022, 207 с.
9. David Goldberg, David Nichols, Brian Oki, Douglas Terry. Using Collaborative Filtering to Weave an Information Tapestry. URL:

https://bitsavers.org/pdf/xerox/parc/techReports/CSL-92-10_Using_Collaborative_Filtering_to_Weave_an_Information_Tapestry.pdf (дата звернення: 08.10.2025)

10. Dietmar Jannach, Markus Zanker, Alexander Felfernig, Gerhard Friedrich. Recommender Systems. An Introduction. URL: https://pzs.dstu.dp.ua/DataMining/recom/bibl/1jannach_dietmar_zanker_markus_felfernig_alexander_friedrich.pdf (дата звернення: 08.10.2025)
11. A brief overview of large language models. URL: https://fsulib.com/a-brief-overview-of-large-language-models/ (дата звернення: 09.10.2025)
12. Zichong Wang, Zhibo Chu, Thang Viet Doan, Shiwen Ni, Min Yang, Wenbin Zhang. History, Development, and Principles of Large Language Models-An Introductory Survey. URL: https://arxiv.org/abs/2402.06853 (дата звернення: 09.10.2025)
13. Michael Brenndoerfer. Recurrent Neural Networks - Machines That Remember. URL: https://mbrenndoerfer.com/writing/history-rnn-recurrent-neural-networks (дата звернення: 09.10.2025)
14. Sebastian Raschka. Build a Large Language Model (From Scratch). Manning, 2024. 368 с.
15. Aeree Cho, Grace C. Kim, Alexander Karpekov, Alec Helbling, Zijie J. Wang, Seongmin Lee, Benjamin Hoover, Duen Horng Chau. Transformer Explainer: Interactive Learning of Text-Generative Models. URL: https://arxiv.org/abs/2408.04619 (дата звернення: 09.10.2025)
16. Дослідницька група. GPT-4 Technical Report. URL: https://arxiv.org/abs/2303.08774 (дата звернення: 09.10.2025)
17. Gemini Team, Google. Gemini: A Family of Highly Capable Multimodal Models. URL: https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf (дата звернення: 10.10.2025)
18. Дослідницька група. Why language models hallucinate. URL: https://openai.com/index/why-language-models-hallucinate/ (дата звернення: 10.10.2025)

19. AIDA64. AIDA64 Extreme. URL: https://www.aida64.com/products/aida64-extreme?language_content_entity=en (дата звернення: 13.10.2025)
20. HWiNFO. About HWiNFO. URL: https://www.hwinfo.com/about-software/ (дата звернення: 13.10.2025)
21. What are the Best Automated Windows Repair Tools?. URL: https://www.technibble.com/what-are-the-best-automated-windows-repair-tools/ (дата звернення: 13.10.2025)
22. PC-Doctor. PC-Doctor Service Center. URL: https://www.pcdservicecenter.com/ (дата звернення: 13.10.2025)
23. CrystalDiskInfo. Key Features. URL: https://crystalmark.info/en/software/crystaldiskinfo/crystaldiskinfo-key-features/ (дата звернення: 13.10.2025)
24. Marcelo Guerra Hahn. Learn C# with Visual Studio 2022: Comprehensive guide to C# fundamentals, Core .NET concepts, advanced features, and building with Visual Studio 2022, BPB Publications, 2025. 368 с.
25. Joseph Albanari, Eric Johannsen. C# 12 in a Nutshell: The Definitive Reference, O'Reilly Media, 2023. 1083 с.
26. John Sharp. Microsoft Visual C# Step by Step (Developer Reference), Microsoft Press, 2022. 832 с.
27. Marcin Jamro. C# Data Structures and Algorithms: Harness the power of C# to build a diverse range of efficient applications, Packt Publishing, 2024. 372 с.
28. Mike McGrath. C# Programming in easy steps, In Easy Steps Limited, 2022. 192 с.
29. Myroslav Budzanivskyi. Xamarin vs .NET MAUI: What You Need to Know in 2025. URL: https://www.codebridge.tech/articles/xamarin-vs-netmaui (дата звернення: 15.10.2025)
30. .NET MAUI vs Xamarin: Which is Best for your Project?. URL: https://eluminoustechnologies.com/blog/net-maui-vs-xamarin/ (дата звернення: 15.10.2025)

31. What is .NET MAUI? URL: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-9.0 (дата звернення: 15.10.2025)
32. Jignesh Trivedi. ObjectContext VS DBContext. URL: https://www.c-sharpcorner.com/UploadFile/ff2f08/objectcontext-vs-dbcontext/ (дата звернення: 16.10.2025)
33. Holger Schwichtenberg. Modern Data Access with Entity Framework Core. URL: https://dl.ebooksworld.ir/motoman/Apress.Modern.Data.Access.with.Entity.Framework.Core.www.EBooksWorld.ir.pdf (дата звернення: 16.10.2025)
34. David Mohundro. 'POCO' definition. URL: https://stackoverflow.com/questions/250001/poco-definition (дата звернення: 16.10.2025)
35. Compare EF Core & EF6. URL: https://learn.microsoft.com/en-us/ef/efcore-and-ef6/ (дата звернення: 16.10.2025)
36. ML.NET. An open source and cross-platform machine learning framework. URL: https://dotnet.microsoft.com/en-us/apps/ai/ml-dotnet#extensibility (дата звернення: 16.10.2025)
37. Jarred Capellman. Hands-On Machine Learning with ML.NET: Getting started with Microsoft ML.NET to implement popular machine learning algorithms in C#, Packt Publishing, 2020. 451 с.
38. Matt Eland. Why use ML.NET? URL: https://mattonml.net/post/why_mlnet/ (дата звернення: 16.10.2025)
39. What is Automated Machine Learning (AutoML)? URL: https://learn.microsoft.com/en-us/dotnet/machine-learning/automated-machine-learning-mlnet (дата звернення: 16.10.2025)
40. Microsoft Learn. Common web application architectures. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення: 16.10.2025)
41. Matt Goldman. .NET MAUI in Action, Manning, 2023. 899 с.

42. Michael C., Daniel H., Johan K. .NET MAUI Projects - Third Edition, Packt Publishing, 2024. 630 с.
43. Jesse L., Rodrigo J. .NET MAUI for C# Developers, Packt Publishing, 2023. 296 с.
44. Microsoft Learn. Model-View-ViewModel (MVVM). URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (дата звернення: 18.10.2025)
45. IONOS editorial team. Strategy Design Pattern. URL: https://www.ionos.co.uk/digitalguide/websites/web-development/strategy-pattern/ (дата звернення: 18.10.2025)
46. Dependency Injection(DI) Design Pattern. URL: https://www.geeksforgeeks.org/system-design/dependency-injectiondi-design-pattern/ (дата звернення: 19.10.2025)
47. Microsoft Learn. Description of the database normalization basics. URL: https://learn.microsoft.com/en-us/troubleshoot/microsoft-365-apps/access/database-normalization-description (дата звернення: 19.10.2025)
48. Evidently AI Team. What is concept drift in ML, and how to detect and address it. URL:https://www.evidentlyai.com/ml-in-production/concept-drift (дата звернення: 19.10.2025)
49. Microsoft Visual Studio. Building Apps with XAML and .NET MAUI - Visual Studio Toolbox. URL: https://youtube.com/playlist?list=PLReL099Y5nRdDJre4TGscXx3EzV74O04X&si=x0K5dvUxVw7h7ek1 (дата звернення: 18.10.2025)
50. Sean P. Kane, Karl Matthias. Docker: Up & Running: Shipping Reliable Containers in Production 3rd Edition. O'Reilly Media, 2023. 416 с.
51. Docker. What is Docker?. URL: https://docs.docker.com/get-started/docker-overview/ (дата звернення: 19.10.2025)
52. pgAdmin. What is pgAdmin 4?. URL: https://www.pgadmin.org/faq/#1 (дата звернення: 19.10.2025)

53. Richard Johnson. Practical pgAdmin Deployment and Administration: Definitive Reference for Developers and Engineers, HiTeX Press, 2025. 274 с.
54. Regina O. Obe, Leo S. Hsu. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database 3rd Edition, O'Reilly Media, 2017. 314 с.
55. C. J. Date. Database Design and Relational Theory: Normal Forms and All That Jazz. Apress, 2019. 470 с.
56. Jan L. Harrington. Relational Database Design and Implementation, 4th Edition, Morgan Kaufmann, 2016. 712 с.
57. Toby J. Teorey, Sam S. Lightstone, Tom Nadeau, H.V. Jagadish. Database Modeling and Design, 4th Edition, Morgan Kaufmann, 2010. 296 с.
58. Jon P Smith. Entity Framework Core in Action, Second Edition, Manning Publications, 2021. 624 с.
59. Microsoft Learn. Code First to a New Database. URL: https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database (дата звернення: 19.10.2025)