БАКАЛАВРСЬКА РОБОТА

БР.КІ-16.00.000 ПЗ
Група КІ-22-1
Микитчук Владислав

2026

Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

Микитчук Владислав Петрович
(прізвище, ім'я, по батькові)

УДК 004.67

БАКАЛАВРСЬКА РОБОТА

Розробка web-додатку виявлення критичних помилок при прийнятті ліків на основі Python ETL, Neo4j та JS
(назва роботи)

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня *В. П. Микитчук*
(підпис, ініціали та прізвище здобувача)

Науковий керівник *Гарасимів В. М., к.т.н , доц.*
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри *С. І. Мельничук*
(підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

(повне найменування вищого навчального закладу)

Факультет *інформаційних технологій*

Кафедра *комп'ютерних систем і мереж*

Освітній ступінь *бакалавр*

Спеціальність *123 (F7) – Комп'ютерна інженерія*

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КСМ

(С.І. Мельничук)

"21" *квітня 2026* року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Микитчуку Владиславу Петровичу

(прізвище, ім'я, по батькові)

1. Тема роботи *"Розробка web-додатку виявлення критичних помилок при прийнятті ліків на основі Python ETL, Neo4j та JS"*

керівник проекту (роботи) *Гарасимів В. М., к.т.н , доц.*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від "*21" квітня 2026 року №180/7*

2. Строк подання студентом роботи *11 червня 2026 р.*

3. Вихідні дані до роботи *Матеріали і результати отримані під час проходження переддипломної практики, методичні вказівки, технічна література.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1. Аналіз предметної області та постановка завдання. 2. Графове проєктування системи та транзакційного контуру. 3. Програмна реалізація та верифікація системи.*

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

6. Консультанти розділів роботи

Розділ	Консультант	Підпис, дата

7. Дата видачі завдання *02 лютого 2026 р.*

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Аналіз предметної області та постановка завдання*	*Лютий, 2026*	
2	*Графове проєктування системи та транзакційного контуру*	*Березень, 2026*	
3	*Програмна реалізація та верифікація системи*	*Травень, 2026*	
4	*Оформлення пояснювальної записки*	*Червень, 2026*	
5			

Студент _____ *Микитчук В.П.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Гарасимів В. М.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У дипломній роботі виконано проєктування, алгоритмізацію, програмну реалізацію та інфраструктурне розгортання спеціалізованої інформаційної системи інтелектуального аналізу сумісності лікарських засобів «PharmaGraph». Проєкт спрямований на автоматизацію контролю клінічних ризиків поліпрагмазії, виявлення колізій прихованого передозування та візуалізацію топології медичних мереж.

Під час виконання першого розділу проведено системний аналіз предметної області, досліджено світові цифрові аналоги у сфері охорони здоров'я та сформовано концептуальний асинхронний стек технологій розробки.

У межах другого розділу розроблено логічні та фізичні моделі бази за концепцією Polyglot Persistence, детально задокументовано реляційні таблиці та специфіковано ієрархічну мета-структуру графа знань.

Практична частина третього розділу охоплює розгортання веб-застосунку в Docker за допомогою фреймворку FastAPI, створення трирівневого Cypher-алгоритму ідентифікації конфліктів речовин та інтеграцію контуру штучного інтелекту Google Gemini для трансформації медичної термінології.

Результати проєкту повністю відповідають технічним вимогам, експериментально доводять безпомилковість динамічного рендерингу мереж і підтверджують ефективність обраної гібридної моделі баз даних для забезпечення персональної медичної безпеки.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, ГРАФ ЗНАНЬ, NEO4J, POSTGRESQL, FASTAPI, POLYGLOT PERSISTENCE, ПОЛІПРАГМАЗІЯ, ЛІКАРСЬКІ ЗАСОБИ, МІЖМЕДИКАМЕНТОЗНІ ВЗАЄМОДІЇ, ШТУЧНИЙ ІНТЕЛЕКТ, GOOGLE GEMINI, CYPHER.

SUMMARY

This thesis covers the design, algorithmisation, software implementation and infrastructure deployment of ‘PharmaGraph’, a specialised information system for the intelligent analysis of drug compatibility. The project aims to automate the monitoring of clinical risks associated with polypharmacy, identify conflicts arising from hidden overdoses, and visualise the topology of healthcare networks.

During the first chapter, a systematic analysis of the subject area was conducted, global digital counterparts in the healthcare sector were investigated, and a conceptual asynchronous stack of development technologies was formed.

In the second section, logical and physical database models were developed based on the Polyglot Persistence concept, relational tables were documented in detail, and the hierarchical meta-structure of the knowledge graph was specified.

The practical part of the third chapter covers the deployment of a web application in Docker using the FastAPI framework, the creation of a three-level Cypher algorithm for identifying substance conflicts, and the integration of the Google Gemini artificial intelligence engine for the transformation of medical terminology.

The project results fully meet the technical requirements, experimentally demonstrate the accuracy of dynamic network rendering, and confirm the effectiveness of the chosen hybrid database model for ensuring personal medical safety.

Keywords: INFORMATION SYSTEM, KNOWLEDGE GRAPH, NEO4J, POSTGRESQL, FASTAPI, POLYGLOT PERSISTENCE, POLYPRAGMATISM, MEDICINES, DRUG INTERACTIONS, ARTIFICIAL INTELLIGENCE, GOOGLE GEMINI, CYPHER.

ЗМІСТ

					БР.КІ-16.00.00.000 ПЗ			
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Микитчук В.П*			*Розробка web-додатку виявлення критичних помилок при прийнятті ліків*	*Літ.*	*Арк.*	*Аркушів*
Перевір.		*Гарасимів В.М*					3	76
Реценз.						ІФНТУНГ, КІ-22-1		
Н. Контр.		*Лазорів А.М.*						
Затверд.		*Мельничук С.І.*						

ВСТУП

Актуальність теми. Сучасний етап розвитку охорони здоров'я та практичної фармакології характеризується стрімким збільшенням кількості комплексних терапевтичних схем, що передбачають одночасне призначення пацієнтам великої кількості медикаментів. Згідно з технічними звітами Всесвітньої організації охорони здоров'я, рутинне супутнє використання пацієнтом п'яти або більше лікарських засобів, включаючи рецептурні препарати та засоби безрецептурного відпуску, визначається як стан поліпрагмазії [4]. Одночасний прийом такої кількості засобів суттєво підвищує ймовірність виникнення побічних ефектів, несприятливих медикаментозних реакцій, когнітивних порушень, а також створює великий економічний тягар для системи охорони здоров'я через часті випадки шкідливого міжмедикаментозного впливу.

Для автоматизації контролю та забезпечення безпеки пацієнтів виникає інженерна задача розробки цифрових рішень, здатних агрегувати нормативно-довідкові дані. В Україні основним першоджерелом інформації про верифіковані лікарські засоби є Державний реєстр лікарських засобів Міністерства охорони здоров'я України, який містить структуровані записи із зазначенням торгівельного найменування, міжнародного непатентованого найменування (МНН), форми випуску, умов відпуску, фармакотерапевтичної групи, ієрархічних кодів АТС та електронних посилань на інструкції [1]. Автоматизований доступ до цих масивів інформації став можливим завдяки державній політиці цифровізації, регламентованій Постановою Кабінету Міністрів України № 835 «Про затвердження Положення про набори даних, які підлягають оприлюдненню у формі відкритих даних», яка декларує обов'язковість публікації реєстрів у відкритих форматах для їх подальшого вільного некомерційного та комерційного використання [2].

Таким чином, розробка веб-орієнтованої інформаційної системи підтримки прийняття рішень на базі графових моделей та інтелектуальних

інтерфейсів є актуальним інженерним завданням у межах спеціальності «Комп'ютерна інженерія».

Об'єкт дослідження – процес автоматизованого аналізу та виявлення міжмедикаментозних взаємодій у багатокомпонентних терапевтичних схемах призначення лікарських засобів.

Предмет дослідження – методи, алгоритми, архітектурні рішення та програмне забезпечення веб-орієнтованої інформаційної системи підтримки прийняття рішень «PharmaGraph» на базі СУБД Neo4j, PostgreSQL та FastAPI.

Мета роботи – проєктування, програмна реалізація та розгортання веб-орієнтованої системи підтримки прийняття рішень «PharmaGraph» для високопродуктивного предиктивного аналізу сумісності лікарських засобів у режимі реального часу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати нормативно-правове забезпечення, структуру медичних реєстрів України та міжнародні стандарти класифікації лікарських засобів;
2. Провести порівняльний аналіз існуючих інформаційних систем оцінки медичної сумісності, обґрунтувати вибір графових моделей та сформувати вимоги до технологічного стеку розробки;
3. Спроектувати гібридну архітектуру збереження даних (Polyglot Persistence) на базі реляційної СУБД PostgreSQL та графової СУБД Neo4j;
4. Розробити трирівневий алгоритм перевірки сумісності медичних препаратів у графі для виявлення дублювання діючих речовин, прямих протипоказань та терапевтичного накладання за кодами АТС ВООЗ;
5. Реалізувати асинхронний серверний програмний інтерфейс (FastAPI) з модулями JWT-авторизації, інтеграції з Gemini API та проксі-обробки специфічних форматів даних медичних інструкцій;
6. Розробити клієнтську частину веб-додатка з інтерактивною візуалізацією мережі медичних ризиків на базі фізичного моделювання сил бібліотеки D3.js;

7. Провести перевірку працездатності реалізованої системи та оцінити її характеристики при роботі з глибокими графовими зв'язками.

Методи дослідження. Для вирішення поставлених завдань у роботі використано такі методи:

- методи теорії графів та графового моделювання – для проектування структури бази знань та зв'язків між медичними сутностями;
- методи матричного аналізу та декартового добутку елементів у Cypher-запитах – для розробки алгоритму парного порівняння вмісту віртуальних аптечок;
- методи об'єктно-реляційного відображення (ORM) та криптографічного захисту – для побудови надійної схеми автентифікації користувачів у реляційному сховищі;
- методи системного та структурного аналізу – при дослідженні інформаційних потоків і побудові загальної архітектури веб-додатка;
- методи візуального Force-Directed моделювання – для трансформації аналітичних даних сервера у графічну інтерактивну модель мережі ризиків у веб-інтерфейсі.

Практичне значення отриманих результатів. Розроблена веб-орієнтована система «PharmaGraph» може бути безпосередньо впроваджена в практику роботи медичних установ, амбулаторій, аптечних мереж або використана як індивідуальний цифровий помічник пацієнта для мінімізації ризиків поліпрагмазії та підвищення безпеки фармакотерапії. Архітектурні рішення з інтеграції графових СУБД із великими мовними моделями можуть слугувати базою для проєктування ширших експертних систем у галузі комп'ютерної інженерії та медичної інформатики.

1 АНАЛІЗ ФАРМАЦЕВТИЧНОЇ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз структури державних реєстрів ліків та профілів користувачів системи

Забезпечення безпеки терапевтичного процесу при призначенні комплексних фармакологічних схем лікування є однією з найбільш критичних завдань сучасної медичної інженерії. Стрімке зростання кількості комерційних брендів лікарських засобів на ринку України призводить до небезпечного явища клінічної поліпрагмазії – одночасного призначення п’яти або більше препаратів одному пацієнту [4]. За даними звітів ВООЗ, супутній прийом кількох ліків експоненційно підвищує ймовірність виникнення міжмедикаментозних конфліктів, що веде до госпіталізації пацієнтів через токсичні ускладнення.

У межах дослідження інформаційного процесу автоматизованого аналізу сумісності лікарських засобів було визначено коло потенційних користувачів (акторів) проєктуємої системи «PharmaGraph», що безпосередньо вплинуло на архітектурні вимоги до інтерфейсу та швидкодії:

- прості пацієнти (кінцеві споживачі ліків) – використовують систему для базового ознайомлення та побутового контролю своєї домашньої аптечки, щоб уникнути грубих помилок самолікування (наприклад, випадкового прийому двох брендів з однаковою діючою речовиною). Для них система виступає інтерактивним інформаційним довідником з простим викладом інформації;
- практикуючі лікарі (терапевти, кардіологи) – застосовують систему як експрес-запобіжник (скринінг-інструмент) безпосередньо під час виписки рецепта. Їхня роль зводиться до швидкої фіксації факту відсутності або наявності критичних колізій. Система не замінює клінічне мислення лікаря, а лише виконує функцію швидкої цифрової звірки (довідковий чек-ліст), сигналізуючи про потенційні ризики у стані поліпрагмазії;

- клінічні фармацевти – використовують систему як первинний автоматизований фільтр для масового моніторингу листів призначень у стаціонарі. Це дозволяє миттєво відсіяти потенційно безпечні терапевтичні схеми та сфокусувати увагу фахівця лише на декількох критичних випадках, що потребують його експертної ручної перевірки, суттєво зменшуючи рутинне навантаження.

Базовим законодавчим та інформаційним фундаментом для автоматизації цього процесу є нормативно-довідкова система Міністерства охорони здоров'я України, зокрема Державний реєстр лікарських засобів України [1]. Даний реєстр поширюється розпорядником у формі відкритих даних відповідно до Постанови Кабінету Міністрів України № 835 [2]. Паспортні відомості реєстру містять сорок п'ять інформаційних атрибутів для кожного запису, включаючи торгівельне найменування бренду, міжнародне непатентоване найменування (МНН), форму випуску, склад діючих речовин та кодування за Анатомо-терапевтично-хімічною класифікацією (АТС) ВООЗ [3].

Проте детальний технічний аналіз структури цих медичних даних виявив ряд серйозних інженерних викликів, які унеможливлюють пряме використання державного реєстру в сучасних інформаційних системах підтримки прийняття рішень:

1. Проблема кодування байтових потоків: первинні плоскі CSV-масиви оприлюднюються розпорядником у застарілому локалізованому кодуванні Windows-1251 (CP1251). Спроби прямого імпорту цих даних сучасними асинхронними веб-серверами, орієнтованими на стандарт Юнікод (UTF-8), викликають критичні виключення на етапі парсингу, що потребує проєктування проміжного програмного шару для динамічної перекодифікації даних на етапі міграції в СУБД PostgreSQL [14].

2. Нерегулярна структура текстових описів складу: інформація про повний хімічний склад брендів часто вноситься у реєстр у вигляді неструктурованого тексту з довільним використанням розділювачів, дужок та дозувань. Це унеможливлює точну автоматичну ідентифікацію діючих речовин

реляційними методами і вимагає побудови семантичної графової моделі, яка зв'язує комерційні бренди з верифікованими вузлами чистих хімічних субстанцій.

3. Застарілі формати збереження медичних інструкцій: офіційні інструкції до ліків надаються реєстром у вигляді монолітних веб-архівів формату MHT (MIME HTML), які містять змішану HTML-розмітку, інлайнові стилі та бінарні картинки Base64 у кодуванні Windows-1251. Оскільки ці інструкції є головним джерелом текстової інформації про побічні ефекти, виникає потреба в розробці спеціалізованих проксі-шлюзів для асинхронного витягування, розпакування та очищення вмісту інструкцій на льоту [20].

Для забезпечення міжнародної інтероперабельності системи «PharmaGraph» та уніфікації знань, інформаційні потоки структурувалися відповідно до п'ятирівневої ієрархічної структури кодування АТС ВООЗ [3]: від анатомічного органу (перший рівень) до конкретної хімічної речовини (п'ятий рівень).

Аналіз показав, що традиційні медичні системи в Україні використовують коди АТС лише як текстові мітки (метадані) у таблицях, через що вони неспроможні виявляти складні типи прихованих загроз. Наприклад, якщо пацієнт одночасно приймає два різні комерційні бренди, які містять різні діючі речовини, але належать до однієї фармакологічної підгрупи АТС (наприклад, два різних нестероїдних протизапальних засоби), виникає загроза терапевтичного накладання (Therapeutic Duplication), що різко підвищує сумарну токсичність лікування. Побудова системи підтримки прийняття рішень на основі графових моделей знань дозволяє представити ієрархію АТС у вигляді спрямованого графа сутностей, де виявлення надлишкових схем лікування зводиться до швидкого алгоритмічного знаходження спільних батьківських вузлів у базі знань.

1.2 Порівняльний аналіз систем оцінки медичної сумісності та архітектур СУБД

Проєктування високоефективних інформаційних системи підтримки прийняття рішень (СППР) у медицині потребує ретельного аналізу світового інженерного досвіду в галузі агрегації біомедичних знань та порівняльного дослідження існуючих програмних продуктів аналогічного призначення.

У сучасній світовій клінічній практиці найбільш відомими інструментами для перевірки міжмедикаментозних колізій є веб-сервіси Medscape Drug Interaction Checker [21] та Drugs.com Interaction Checker [22]. Дані платформи мають ряд суттєвих переваг, які забезпечили їм статус світових стандартів:

- глибока верифікація та доказова база – матриці взаємодій у цих системах формуються на основі тисяч рецензованих клінічних досліджень, офіційних релізів FDA та звітів профільної літератури, що гарантує найвищу точність клінічних попереджень;
- багаторівнева класифікація ризиків – сервіси не просто констатують факт конфлікту, а градують його за ступенем загрози (наприклад, Contraindicated – категорично протипоказано, Major – серйозний ризик, Monitor Closely – потребує нагляду), що дозволяє гнучко оцінювати ситуацію;
- широке охоплення біохімічних профілів: бази даних містять детальні описи фармакокінетичних та фармакодинамічних параметрів діючих речовин, включаючи їхній метаболізм через систему ферментів цитохрому P450.

Проте проведений технічний аналіз виявив ряд критичних обмежень, які унеможливлюють їх пряме ефективне застосування як швидких лікарських асистентів у рамках вітчизняного медичного простору:

- відсутність інтеграції з локальними реєстрами – вказані закордонні системи орієнтовані суворо на номенклатуру лікарських засобів США та ЄС. Вони здійснюють пошук за англомовними брендами і не підтримують унікальні комерційні найменування та специфічні комбіновані форми лікарських засобів, зареєстрованих у Державному реєстрі України [1]. Це змушує вітчизняного

користувача або завантаженого лікаря вручну перекладати та розбивати кожен бренд на чисті хімічні речовини (МНН), що критично збільшує час обробки в умовах обмеженого прийому;

- низька інформативність для стану поліпрагмазії – при введенні більше 5 препаратів Medscape [21] та Drugs.com [22] видають користувачу лінійний, монотонний текстовий список парних конспектів. Простий пацієнт не здатний швидко зорієнтуватися у великому масиві однотипного тексту, оскільки системи не мають інструментів інтерактивної візуалізації топології та ієрархії медичних ризиків;
- закритість програмних інтерфейсів (API) – архітектура існуючих глобальних довідників є пропрієтарною та закритою. Вони не надають безкоштовних відкритих API-ендпоінтів для інтеграції аналітичних модулів перевірки у сторонні медичні інформаційні системи (МІС).

Створення локальної експертної бази знань «PharmaGraph» для швидкого експрес-скринінгу міжмедикаментозних взаємодій спрямоване на усунення цих недоліків. Як глобальне джерело верифікованих медичних колізій у роботі використано міжнародну біоінформаційну базу знань DrugBank [10], доступ до XML-датасетів якої було отримано за некомерційною академічною ліцензією (Academic License).

При впровадженні отриманих великих масивів інформації у структуру програмного комплексу виникає критична інженерна задача вибору архітектури системи керування базами даних (СУБД). Традиційним підходом у більшості існуючих МІС є використання реляційних СУБД, таких як PostgreSQL [14]. У реляційній моделі дані організовані у вигляді жорстких двовимірних таблиць, а зв'язки реалізуються за допомогою механізму зовнішніх ключів (Foreign Keys). Проте реляційні бази даних демонструють низьку ефективність при спробах моделювання динамічних, нерегулярних та ad-hoc зв'язків, де інформація про взаємозв'язки є настільки ж важливою, як і самі дані [6].

Аналіз обчислювальної ефективності показує, що для виявлення опосередкованих конфліктів, дублювання речовин за складними комбінованими

брендами та накладання терапевтичних ефектів за ієрархією ATC ВООЗ [3], реляційна СУБД змушена виконувати каскадні операції об'єднання таблиць за допомогою операторів JOIN. Обчислювальна складність таких операцій зростає експоненційно залежно від глибини обходу зв'язків. СУБД змушена виконувати постійне сканування глобальних індексів і будувати великі тимчасові матриці декартового добутку в оперативній пам'яті сервера, що призводить до катастрофічного падіння швидкодії (join pain) при збільшенні кількості ліків в аптечці пацієнта.

Для подолання вказаних обмежень у межах розробки інформаційної системи «PharmaGraph» обґрунтовано перехід до нереляційної графової моделі збереження знань (Knowledge Graphs), де дані представляються у вигляді спрямованих графів з міченими ребрами (Directed Edge-Labelled Graphs) [7], [8]. Графові бази даних реалізують патерн, у якому інформація про взаємозв'язки має такий самий пріоритет (first-class citizenship), як і самі сутності [6]. Приклад структурної організації такого спрямованого графа за міжнародними стандартами консорціуму RDF наведено нижче (рис. 1.1).

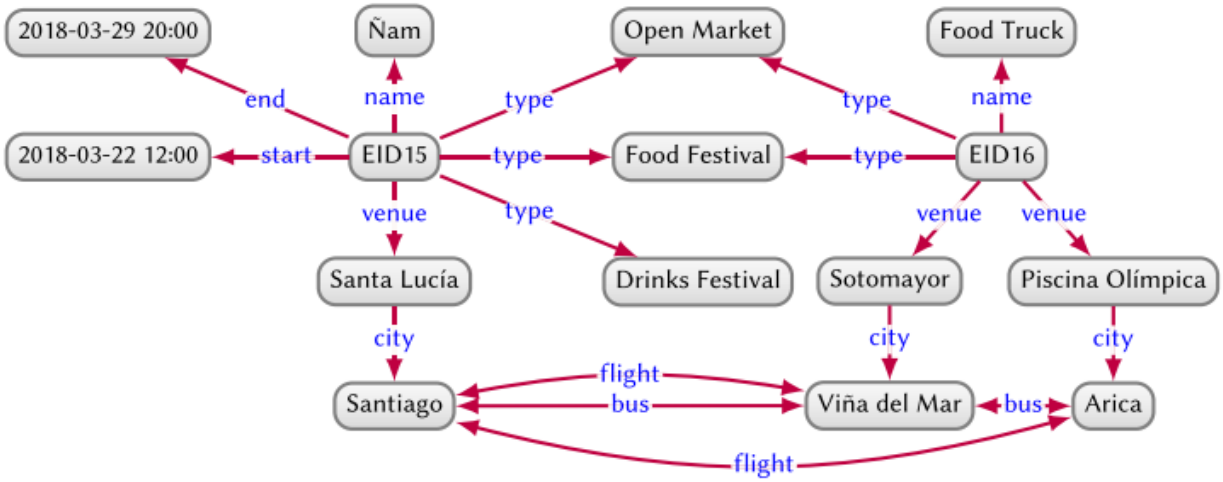

Рисунок 1.1 – Структурна організація спрямованого графа з міченими ребрами за стандартом RDF [7]

Для практичної програмної реалізації підходить модель властивостей графа (Property Graph) [7] та промислову СУБД Neo4j [6]. Особливістю Property Graph є можливість асоціювати довільні набори пар «ключ-значення» (properties) та міток (labels) як з вузлами (Nodes), так і зі зв'язками (Edges) [7]. Концептуальну схему організації такої структури з підтримкою внутрішніх атрибутів сутностей наведено нижче (рис. 1.2).

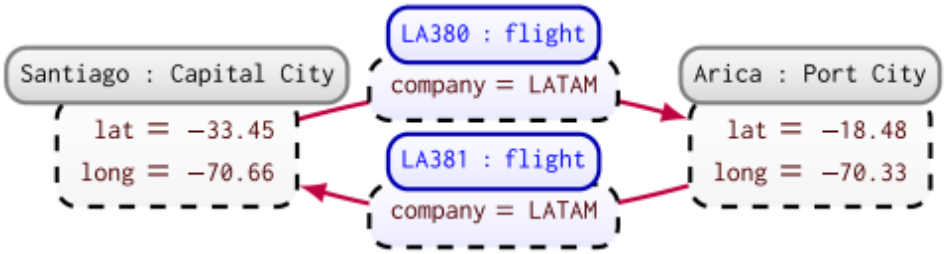

Рисунок 1.2 – Концептуальна схема моделі властивостей графа з підтримкою атрибутів сутностей та зв'язків [7]

Головною інженерною перевагою СУБД Neo4j є використання технології «вказівної суміжності» (Index-free Adjacency) [6]. Кожен вузол у базі даних містить прямі фізичні вказівники на адреси в пам'яті суміжних із ним елементів. Завдяки цьому обхід графа для пошуку конфліктів будь-якої глибини вкладеності виконується за лінійний час O(k), де k – кількість реальних зв'язків об'єкта, без необхідності сканування глобальних індексів чи виконання обчислювально важких табличних об'єднань. Використання декларативної мови запитів Cypher [9] дозволяє лаконічно описувати складні шаблони патологічних взаємодій на основі зіставлення з прототипом (pattern matching).

Проте повна відмова від реляційних структур є недоцільною. Завдання збереження персональних даних користувачів та облікових записів потребують суворих табличних обмежень цілісності та транзакційної надійності. Тому в роботі реалізовано концепцію гібридного архітектурного сховища (Polyglot Persistence). Облікові записи та профілі користувачів зберігаються в реляційній СУБД PostgreSQL [14] з використанням ORM SQLAlchemy [15], що гарантує

транзакційну безпеку ACID [11]. У свою черед, високопродуктивна база знань медичних взаємодій винесена в графове ядро Neo4j, що забезпечує клієнтським JS-інтерфейсам D3.js можливість динамічного рендерингу мережі ризиків у режимі реального часу [16].

1.3 Постановка задачі

На основі проведеного науково-технічного аналізу нормативно-правового забезпечення, структури вітчизняних медичних реєстрів [1–2], міжнародних стандартів кодування ВООЗ [3–4] та порівняльного дослідження реляційних та графових СУБД [5–8] сформульовано технічні, функціональні та системні вимоги до проєктованого інженерного рішення.

Метою бакалаврської роботи є проєктування, програмна реалізація та розгортання веб-орієнтованої системи підтримки прийняття рішень «PharmaGraph» для високопродуктивного предиктивного аналізу сумісності лікарських засобів у режимі реального часу.

Для досягнення поставленої мети архітектура розроблюваної системи повинна вирішувати такі приватні інженерні задачі:

1. Обробка вхідних даних та інтеграція застарілих форматів:

– реалізація програмного модуля імпорту первинного реєстру лікарських засобів України із плоских табличних структур формату CSV [1] з автоматичним декодуванням байтового потоку Windows-1251 у стандарт Юнікод UTF-8;

– розробка проксі-шлюзу для динамічного вивантаження, розпакування та фільтрації веб-архівів інструкцій формату MHT [20] з метою очищення тексту від надлишкової HTML/MIME-розмітки для подальшого інтелектуального аналізу.

2. Проєктування гібридного сховища даних (Polyglot Persistence):

– створення реляційної бази даних PostgreSQL [14] для забезпечення суворого дотримання принципів ACID, збереження облікових записів користувачів, безпечного керування сесіями та авторизаційними токенами [11];

– побудова графової бази знань медичних загрозоутворень у СУБД Neo4j [6] на основі імпортованого глобального XML-датасету DrugBank за академічною некомерційною ліцензією [10]. Топологія графа повинна містити вузли медичних брендів, діючих речовин і п'ятирівневих ієрархічних кодів АТС [3].

3. Розробка алгоритмічного забезпечення аналізу ризиків:

– реалізація трирівневого алгоритму детекції колізій поліпрагмазії засобами мови декларативних запитів Cypher [9] на основі технології «вказівної суміжності» (Index-free Adjacency) [6], [7];

– алгоритм повинен у режимі реального часу ідентифікувати: прямі міжмедикаментозні протипоказання (Direct Interaction), дублювання однакових діючих речовин під різними торговими брендами (Duplicate Substance) та критичні поєднання засобів з однаковим кодом терапевтичного впливу на системи організму за класифікацією ВООЗ (Therapeutic Duplication).

4. Реалізація серверної та клієнтської частин додатка:

– створення асинхронного серверного програмного інтерфейсу (REST API) на базі Python-фреймворку FastAPI [11] та стандарту ASGI [12] з модулями криптографічного захисту паролей Bcrypt та генерації тимчасових токенів JWT;

– інтеграція системи з хмарними сервісами великих мовних моделей через Google Gemini API [19] для динамічної генерації розгорнутих, адаптивних клінічних пояснень природною мовою щодо знайдених графових колізій;

– розробка веб-інтерфейсу з модулем інтерактивної візуалізації мережі ризиків на базі бібліотеки D3.js [16] з використанням алгоритмів фізичного моделювання сил Force-Directed placement [17] для забезпечення можливості динамічного перетягування (drag-and-drop) елементів графа користувачем.

Обмеження та вимоги до реалізації системи:

– платформонезалежність та контейнеризація: Усі компоненти системи (FastAPI, PostgreSQL, Neo4j, Nginx) повинні бути інкапсульовані в ізольовані Docker-контейнери та координуватися через маніфести Docker Compose для забезпечення швидкого розгортання та адміністрування на будь-яких серверних платформах;

– продуктивність: Час прорахунку матриці ризиків для аптечки пацієнта, що містить понад 5 препаратів (стан поліпрагмазії), не повинен перевищувати 2000 мілісекунд, що забезпечується лінійним часом обходу зв'язків у Neo4j.

– безпека: Відкриті інтерфейси API (крім ендпоінтів авторизації та автокомпліту) мають бути суворо захищені залежностями перевірки валідності токенів OAuth2.

У результаті виконання першого розділу проведено аналіз фармацевтичної предметної області, структури медичних реєстрів України та стандартів ВООЗ. На основі дослідження існуючих аналогів обґрунтовано доцільність розробки системи на базі технології Polyglot Persistence. За результатами аналізу сформовано повний перелік вимог до швидкодії та безпеки системи «PharmaGraph» і чітко поставлено інженерне завдання на її розробку.

2 ГРАФОВЕ ПРОЄКТУВАННЯ ТА БЕЗПЕКА ТРАНЗАКЦІЙНОГО КОНТУРУ СИСТЕМИ

2.1 Специфікація функціональних вимог та декомпозиція модулів системи

Проєктування сучасних спеціалізованих систем підтримки прийняття рішень у клінічній медицині та практичній фармакології вимагає ретельного аналізу та формалізації бізнес-процесів. Головною метою розробки інформаційної системи «PharmaGraph» є повна автоматизація процесів виявлення потенційних міжмедикаментозних взаємодій та запобігання негативним наслідкам поліпрагмазії при одночасному призначенні великої кількості лікарських засобів. Для досягнення цієї інженерної мети необхідно провести декомпозицію загальних завдань системи на окремі взаємопов'язані функціональні модулі.

Повний функціональну структуру системи представлено у вигляді діаграми декомпозиції на рисунку 2.1.

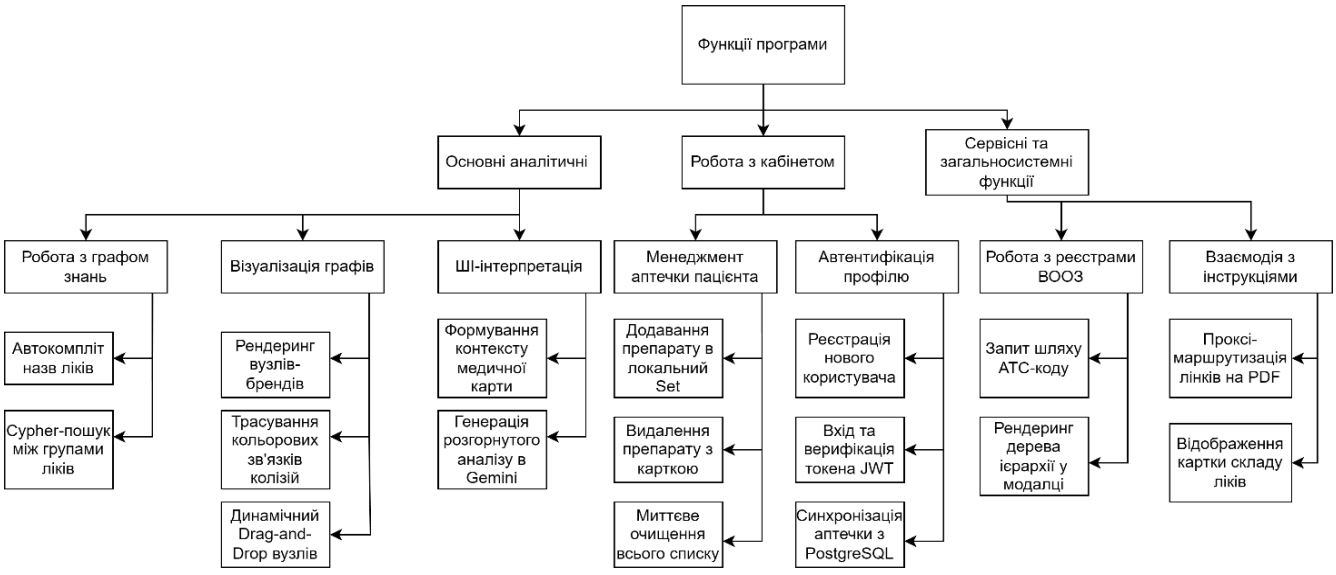

Рисунок 2.1 – Діаграма функціональної декомпозиції ІС «PharmaGraph»

Логічна структура розроблюваної інформаційної системи побудована за сучасним принципом модульної архітектури. У такій моделі кожен елемент відповідає за конкретний етап обробки біомедичних даних.

Згідно з проведеною декомпозицією, загальний простір функцій розподілено на три незалежні підсистеми, які оперують на власному рівні абстракції та обробки інформації. Першим і найголовнішим технологічним блоком є підсистема основних аналітичних функцій та графових обчислень. Цей блок забезпечує безпосередній аналіз клінічних ризиків на серверній та клієнтській сторонах.

В межах аналітичного блоку реалізовано функцію інтелектуального контекстного автокомпліту назв ліків. Вона забезпечує миттєвий пошук комерційних брендів за першими введеними символами. Процес побудовано на базі асинхронних HTTP-запитів до веб-сервера, що повністю виключає необхідність завантаження всього масивного реєстру лікарських засобів МОЗ на пристрій користувача.

Поряд із цим функціонує модуль декларативного пошуку колізій, який формує складні мережеві запити мовою Cypher до бази знань. Програма автоматично аналізує зв'язки між діючими речовинами, які входять до складу обраних користувачем брендів. Система фільтрує та групує знайдені ризики за категоріями небезпеки, серед яких виділяються пряма колізія, критичне дублювання субстанції та терапевтичне накладання за міжнародними кодами ВООЗ.

Наступна важлива функція аналітичного блоку відповідає за інтерактивну візуалізацію топології мережі. Вона перетворює абстрактну відповідь від бекенда у векторний граф, використовуючи алгоритми симуляції фізичних сил. Кожен препарат відображається у вигляді ізольованого вузла, а виявлені ризики трасуються кольоровими лініями зв'язку. Користувач може взаємодіяти з графом вручну, переміщуючи вузли мишкою або рукою, якщо на телефоні. При цьому система автоматично перераховує координати суміжних елементів і утримує їх у межах карти.

Останнім інноваційним елементом аналітичного блоку є ШІ-інтерпретація клінічного контексту. Програма збирає технічні медичні описи колізій із бази даних і компилює їх у структурований промпт. Засобами захищеного API цей запит передається до мовної моделі Gemini. Отриманий результат перетворюється на зрозумілу для пацієнта пораду українською мовою щодо специфіки сумісного прийому ліків.

Другим генеральним архітектурним блоком є підсистема транзакційного керування кабінетом пацієнта. Цей блок відповідає за збереження стану та індивідуалізацію аналітичного процесу. Він оперує на рівні реляційного ядра бази даних. Тут реалізовано функцію менеджменту локальної аптечки, яка дозволяє додавати препарати в персональний список перевірки або видаляти окремі картки, які пацієнт перестав приймати.

Для захисту профілів користувачів впроваджено функцію автентифікації та криптографічного захисту. Під час реєстрації система обов'язково обробляє пароль користувача за допомогою асиметричного алгоритму хешування bcrypt, тому дані ніколи не зберігаються у відкритому вигляді. При успішному вході сервіс генерує короткочасний токен безпеки JSON Web Token. Цей токен зашивається в заголовки наступних запитів клієнта, що забезпечує безпечну безсесійну взаємодію з бекендом і захищає медичні списки від несанкціонованого доступу.

Третє місце в архітектурі посідають сервісні, інфраструктурні та довідкові функції. Вони забезпечують загальну інформаційну підтримку користувача. Серед них виділяється функція запиту та рендерингу ієрархічної структури ВООЗ. При кліку на препарат система витягує з бази знань повний ланцюжок його приналежності до анатомо-терапевтично-хімічної класифікації і відображає всі п'ять рівнів коду у зручному модальному вікні. Також реалізовано функцію проксі-маршрутизації офіційних інструкцій, яка безпечно перенаправляє користувача на першоджерела МОЗ за посиланнями, що зберігаються безпосередньо у властивостях графових вузлів ліків.

На основі розробленої функціональної структури сформуємо сувору інженерну специфікацію вимог до інформаційної системи. Ці вимоги традиційно поділяються на два генеральні класи: функціональні та нефункциональні.

Функціональні вимоги визначають конкретні дії, які повинна виконувати система для задоволення потреб користувача. По-перше, система має забезпечувати контекстний пошук брендів ліків за першими трьома введеними символами з часом відповіді менше 150 мілісекунд. По-друге, програма повинна автоматично знаходити перехресні конфлікти між усіма доданими до аптечки речовинами за допомогою спеціальних скриптів обходу графа. По-третє, система має динамічно рендерити інтерактивний граф взаємодій, маркуючи лінії зв'язків відповідними кольорами залежно від ступеня ризику. Крім того, за запитом користувача система зобов'язана формувати зрозумілий текстовий звіт-пояснення клінічного ризику через інтеграцію з Gemini API. Нарешті, для авторизованих користувачів має підтримуватися автоматична синхронізація поточного стану доданих ліків у реляційну базу даних, а доступ до особистого кабінету повинен суворо блокуватися за відсутності валідного токена безпеки.

Нефункціональні вимоги описують якісні характеристики, обмеження та експлуатаційні параметри розроблюваної системи. З погляду швидкодії, час виконання графового запиту на пошук колізій серед десяти одночасно доданих ліків не повинен перевищувати 0.5 секунди на серверній стороні. Щодо надійності та цілісності даних, при видаленні облікового запису користувача всі пов'язані з ним записи в аптечці повинні видалятися автоматично за каскадним принципом, що запобігає появі ізольованого сміття в базі.

Важливою вимогою є мобільна адаптивність та кросбраузерність інтерфейсу візуалізації графа. Елементи D3.js повинні коректно адаптуватися під будь-яку роздільну здатність екрана сучасних смартфонів за рахунок використання фіксованого віртуального поля координат viewBox, що нівелює специфіку системного масштабування окремих брендів мобільних пристроїв.

2.2 Поведінкове моделювання взаємодії пацієнта з медичними реєстрами

Поведінковий аналіз системи «PharmaGraph» розділено на три незалежні сценарії, що охоплюють роботу особистого кабінету, довідкового модуля, а також функціонування системи у відкритому гостьовому режимі.

Перший сценарій описує процес автентифікації зареєстрованого користувача та первинний запуск аналізу сумісності ліків через особистий кабінет. Логіку цього покрокового алгоритму від моменту введення пароля до виведення графа відображено на рисунку 2.2.

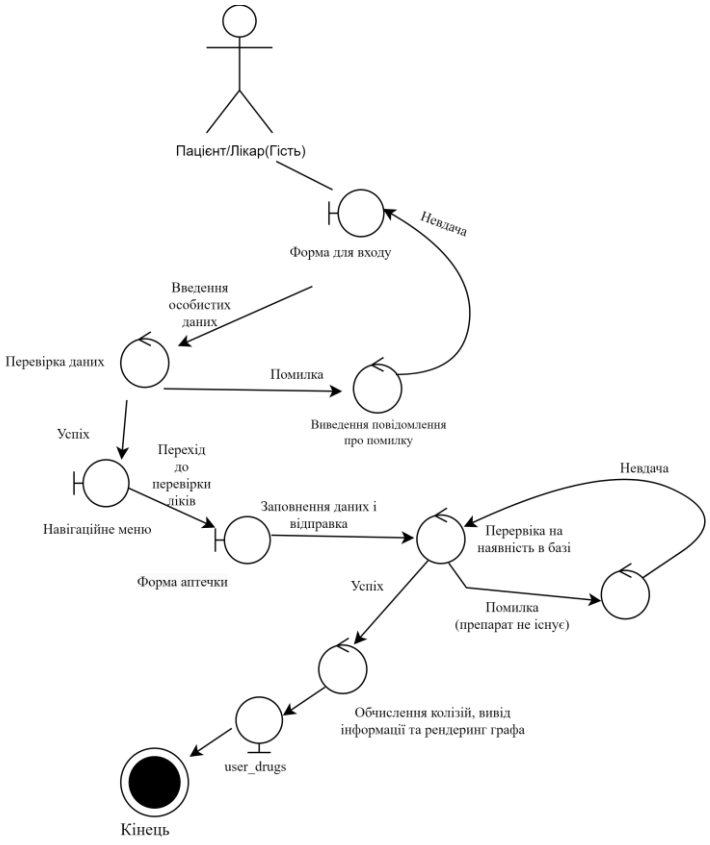

Рисунок 2.2 – Use-Case діаграма процесу авторизації та ініціалізації перевірки взаємодій

Процес авторизації та взаємодії з аптечкою в межах кабінету користувача виконується за таким покроковим алгоритмом:

1. Користувач вводить логін і пароль у вікні «Форма для входу», після чого дані передаються на об'єкт керування «Перевірка даних»;

2. Об'єкт керування звіряє отримані параметри з базою даних PostgreSQL і, у випадку помилки, перенаправляє потік на контролер «Виведення повідомлення про помилку», який повертає користувача на початковий екран;

3. При успішній перевірці даних система генерує JWT-токен і відкриває для користувача екран «Навігаційне меню»;

4. З меню користувач переходить на робочу «Форму аптечки», де заповнює список ліків і натискає кнопку відправки;

5. Контролер «Перевірка на наявність в базі» перевіряє введені назви і, якщо список не порожній, запускає фінальний контролер «Обчислення колізій та рендеринг графа»;

6. Цей контролер одночасно зберігає стан ліків у базу даних і виводить готову інтерактивну мережу D3.js на екран користувача.

Другий незалежний сценарій описує роботу довідкової системи, коли користувач взаємодіє з уже побудованим графом ліків, переглядає детальний опис речовин, ієрархію АТС та офіційні інструкції МОЗ. Система забезпечує зручну навігацію між пов’язаними лікарськими засобами та їх активними компонентами, дозволяючи швидко отримувати необхідну інформацію. Крім того, користувач може аналізувати взаємозв’язки між препаратами різних фармакологічних груп і досліджувати їх місце в класифікаційній структурі АТС. Такий підхід сприяє більш ефективному пошуку даних та покращує процес вивчення лікарських засобів. Схему цього процесу з урахуванням обробки помилок відсутності даних представлено на рисунку 2.3.

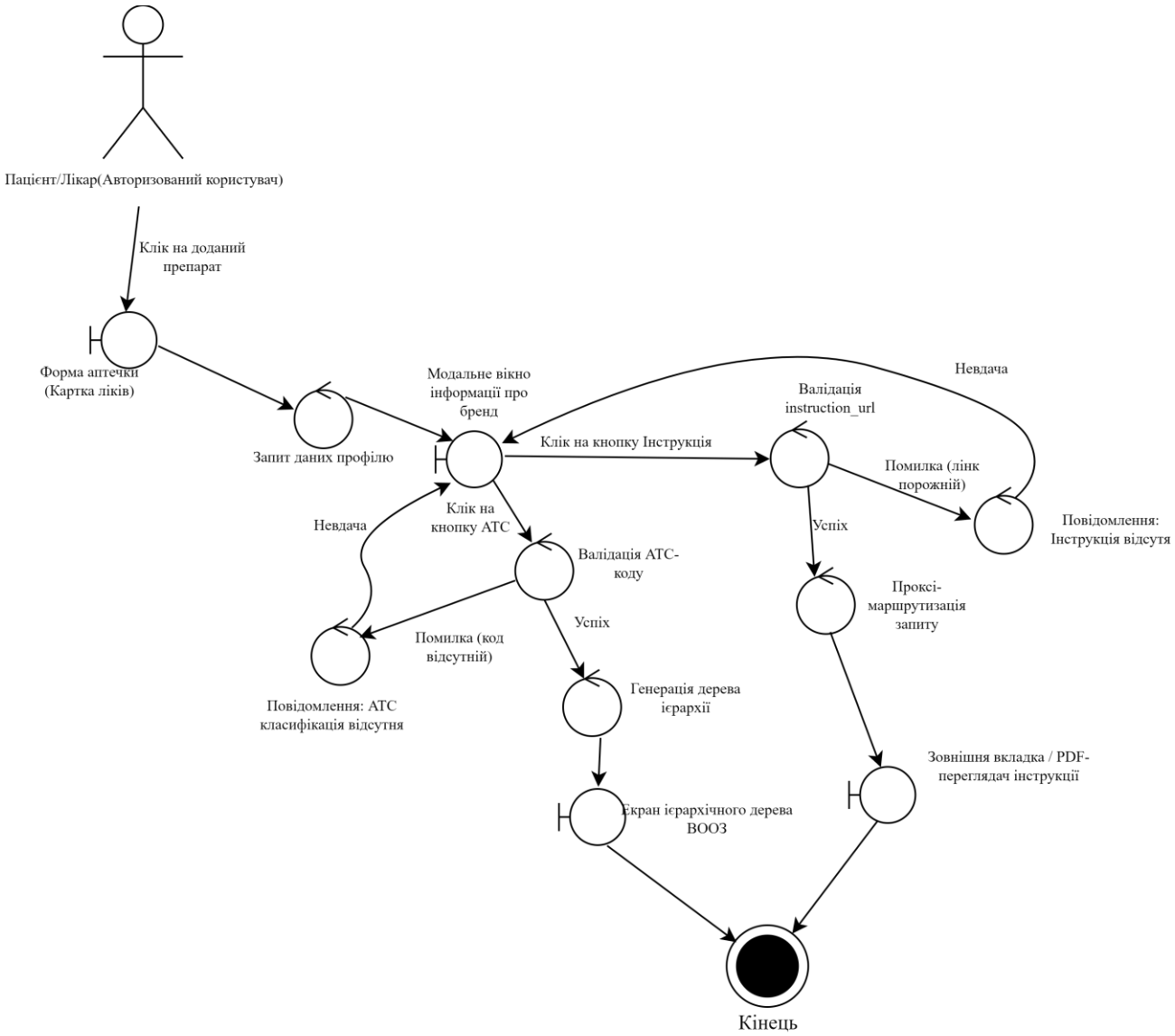

Рисунок 2.3 – Use-Case діаграма процесу взаємодії з довідковими реєстрами та АТС-ієрархією ліків

Логіка обробки запитів на отримання розширеної фармацевтичної інформації складається з таких етапів:

– користувач клікає на картку ліків у «Формі аптечки», що активує об'єкт керування «Запит даних профілю»;

– цей контролер звертається до графової бази знань Neo4j, зчитує властивості вузла Brand і відкриває «Модальне вікно інформації про бренд»;

– при спробі перегляду класифікації ВООЗ запускається контролер «Валідація АТС-коду», який перевіряє наявність відповідних зв'язків у базі знань;

– якщо код відсутній, користувач бачить повідомлення «Повідомлення: АТС класифікація відсутня»;

– якщо код є в базі, контролер «Генерація дерева ієрархії» зчитує сутності АТС та зв'язки PART_OF, після чого будує «Екран ієрархічного дерева ВООЗ»;

– при спробі відкрити інструкцію запускається контролер «Валідація instruction_url», який у разі порожнього посилання нічого не виводить і користувач бачить «Інструкція відсутня»;

– за наявності робочого посилання активується об'єкт «Проксі-маршрутизація запиту», який відкриває документ у «Зовнішньому PDF-переглядачі».

Третій сценарій описує спрощену гостьову сесію, коли неавторизований користувач використовує публічний інтерфейс програми без створення облікового запису. Модель надійності для гостьового сценарію представлена на рисунку 2.4.

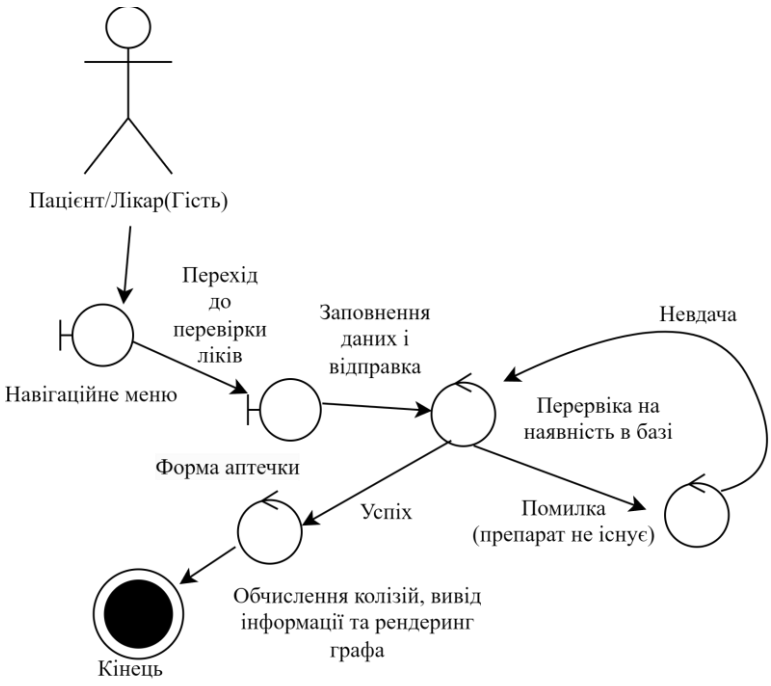

Рисунок 2.4 – Use-Case діаграма процесу гостьової перевірки препаратів

Аналіз функціонування системи у гостьовому режимі виявляє наступну послідовність дій та переходів між об'єктами:

1. Пацієнт або лікар у статусі Гостя починає взаємодію з об'єктом грані «Навігаційне меню» та ініціює перехід до вікна швидкої перевірки;
2. Система відкриває об'єкт грані «Форма аптечки», де користувач заповнює текстові поля назвами лікарських засобів і надсилає запит на бекенд;
3. Контролер «Перевірка на наявність в базі» перевіряє введені назви і, якщо список не порожній, запускає фінальний контролер «Обчислення колізій та рендеринг графа»;
4. Цей контролер одночасно зберігає стан ліків у базу даних і виводить готову інтерактивну мережу D3.js на екран користувача.

2.3 Часове моделювання та хронологічний аналіз асинхронних HTTP-запитів

Для дослідження часової динаміки, взаємодії та хронологічного обміну запитами між окремими компонентами інформаційної системи «PharmaGraph» застосовується UML-діаграма послідовності (Sequence Diagram). На відміну від моделей надійності, які описують логічні переходи між вікнами та контролерами, діаграма послідовності відображає життєвий цикл запиту на часовій шкалі зверху вниз та фіксує тривалість активності кожного об'єкта. У якості базового сценарію для цього моделювання обрано процес багатокомпонентної перевірки ліків, оскільки він задіює всі рівні архітектури програми. Хронологію обміну повідомленнями між інтерфейсом, бекендом та базою знань представлено на рисунку 2.5.

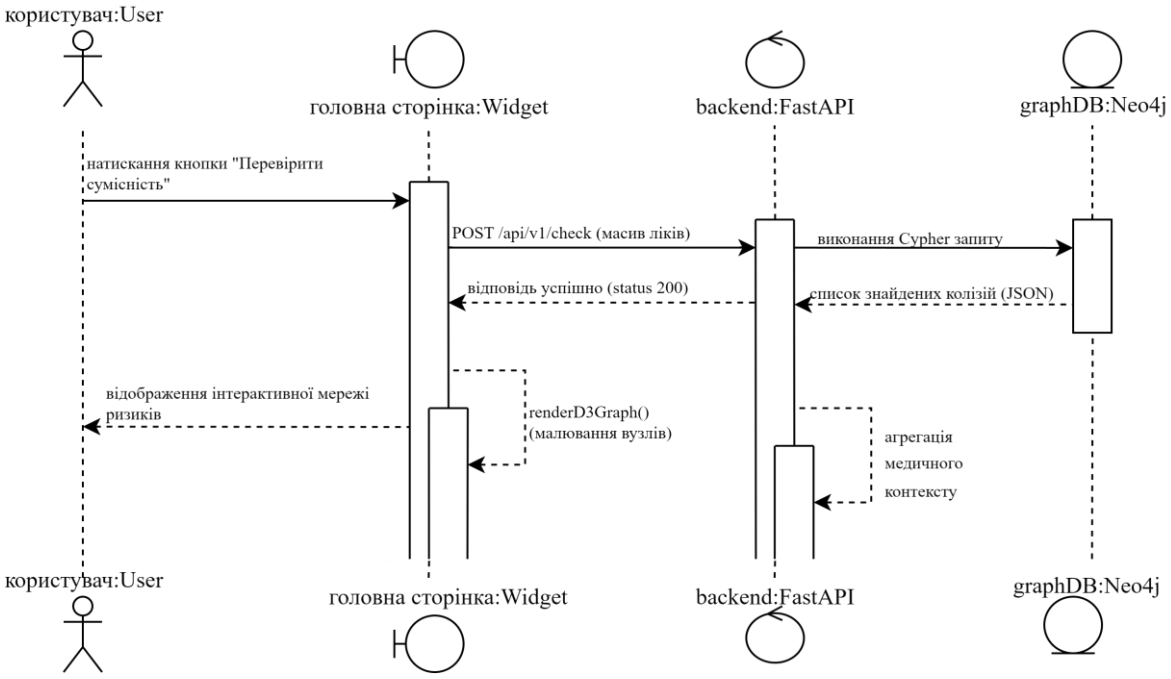

Рисунок 2.5 – UML-діаграма послідовності сценарію аналізу міжмедикаментозних взаємодій

Опис часових етапів та взаємодії об'єктів під час виконання аналітичного запиту складається з такої послідовності кроків:

1. Користувач взаємодіє з об'єктом грані головна сторінка:Widget і натискає кнопку запуску перевірки, що активує першу лінію життя на схемі;

2. Об'єкт головна сторінка:Widget формує асинхронний HTTP-запит за методом POST та передає масив обраних назв ліків на сервер до контролера backend:FastAPI;

3. Контролер веб-сервера переходить у стан активності, відкриває відповідний блок обробки даних і трансформує отриманий масив у декларативний Cypher-запит;

4. Сервер бекенда надсилає суцільну стрілку виклику до об'єкта графової бази даних graphDB:Neo4j для пошуку наявних клінічних конфліктів;

5. База даних graphDB:Neo4j запускає внутрішній процес сканування ребер INTERACTS_WITH та колізій діючих речовин, задіявши власний часовий блок активності;

6. Після завершення пошуку об'єкт бази даних повертає пунктирну стрілку відповіді із результуючим JSON-масивом колізій назад до backend:FastAPI;

7. Контролер бекенда виконує внутрішню операцію агрегації медичного контексту, закриває свій блок активності та надсилає пунктирну стрілку успішної відповіді (status 200 OK) на клієнтську сторону;

8. Об'єкт грані головна сторінка:Widget отримує дані, запускає внутрішню функцію рендерингу мережі renderD3Graph() і повертає фінальну відповідь користувачеві, відображаючи інтерактивний граф ризиків.

Розглянута модель послідовності чітко демонструє асинхронну природу архітектури застосунку. Завдяки ізоляції ліній життя, клієнтський інтерфейс не блокується під час виконання складних графових обчислень на серверній стороні. Це забезпечує стабільну роботу програми навіть при обробці великих масивів колізій, коли база даних Neo4j виконує глибоке трасування зв'язків.

2.4 Математичне обґрунтування алгоритмів обходу зв'язків та алгоритмічне забезпечення системи

Функціонування аналітичного контуру інформаційної системи «PharmaGraph» базується на математичних моделях теорії графів, методах чисельного інтегрування та алгоритмах симуляції фізичних процесів. Алгоритмічне забезпечення системи вирішує два фундаментальні завдання, які забезпечують працездатність усього програмного комплексу. Першим завданням є точне виявлення клінічних конфліктів між компонентами лікарських засобів за допомогою єдиного декларативного запиту на серверній стороні. Другим завданням є динамічне моделювання геометричної сітки взаємодій та реактивний рендеринг елементів на клієнтській стороні з урахуванням специфіки системного масштабування мобільних пристроїв.

Для автоматизації пошуку міжмедикаментозних взаємодій розроблено спрямований алгоритм трасування зв'язків у базі знань. Математично графова

база знань представлена у вигляді гетерогенного орієнтованого графа, який описується формулою 2.1:

$$G = (V, E) \quad (2.1)$$

де G – графова база даних;

V – множина вузлів, що класифікуються за лейблами ліків, діючих хімічних речовин та анатомо-терапевтично-хімічних кодів;

E – множина спрямованих ребер, які визначають логічні та медичні зв'язки між сутностями.

Логіка роботи головного алгоритму перевірки сумісності ліків складається з послідовності кроків, яку представлено у вигляді блок-схеми на рисунку 2.6.

Унікальною архітектурною особливістю розробленого серверного алгоритму перевірки є механізм динамічного кешування лінгвістичних даних. Під час детекції прямих міжмедикаментозних взаємодій система виконує умовну перевірку наявності локалізованого україномовного опису клінічного випадку. За умови відсутності відповідного ключа у структурі ребра графа, програма автоматично викликає ізольований підмодуль трансляції. Даний підмодуль виконує звернення до сервісу перекладу, здійснює адаптацію медичного тексту і за допомогою оператора оновлення перезаписує властивості ребра безпосередньо у сховищі сервера. Це дозволяє оптимізувати мережевий трафік та повністю виключити повторні затримки при наступних зверненнях користувачів до аналогічної пари препаратів.

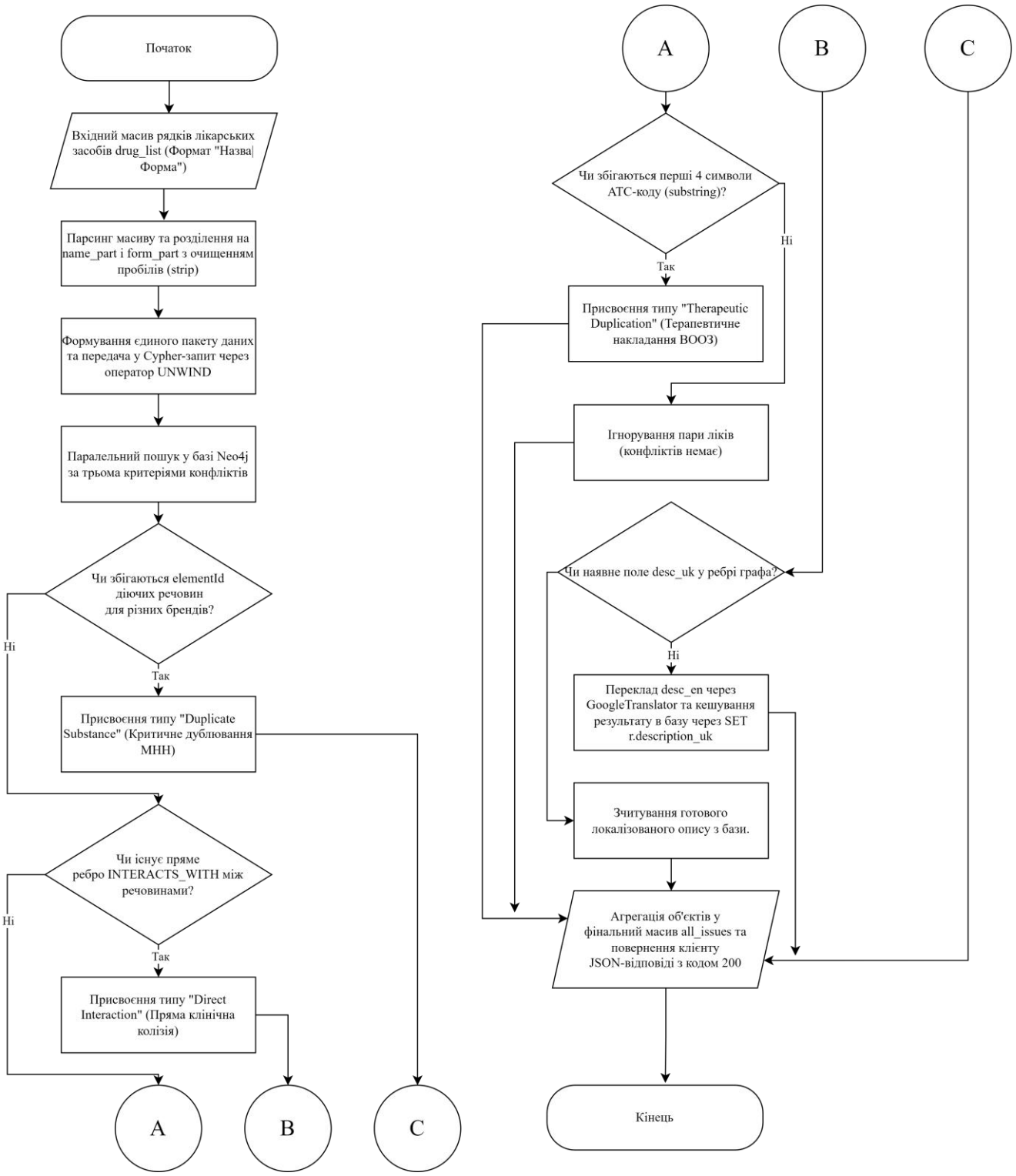

Рисунок 2.6 – Блок-схема головного алгоритму перевірки лікарської сумісності

Поза межами основного контуру перевірки функціонує незалежний асинхронний маршрут інтелектуального аналізу. Він не бере участі у первинному обчисленні кількості конфліктів, а виступає як додатковий сервіс

генерації експланаційних висновків за запитом користувача. Алгоритм зчитує технічний опис колізії, формує структурований медичний промпт із жорстким обмеженням використання символів розмітки та передає його до мовної моделі штучного інтелекту, повертаючи пацієнту адаптовану інтерпретацію ризиків простою мовою.

Після отримання результуючого масиву конфліктів на фронтенді ініціюється процес геометричного розміщення елементів на екрані. Для побудови візуального графа використовується математичний апарат силових моделей (Force-Directed Layouts) бібліотеки D3.js. Модель описує граф як фізичну систему, де вузли діють як об'єкти з реальними координатами, а ребра – як пружини, що їх пов'язують. Геометрична рівновага та стабільність відображення мережі в межах адаптивної сітки розраховується за допомогою системи диференціальних рівнянь руху для кожного вузла, яка вирішується чисельним методом інтегрування Верле під час кожного кроку симуляції.

Динамічний баланс та утримання елементів у межах видимої зони екрана забезпечується на кожному ітераційному кроці за рахунок взаємодії таких математичних сил:

– сила Кулона, яка моделює відштовхування між однойменно зарядженими частинками для запобігання накладанню карток ліків одна на одну на площині;

– сила Гука, яка задає лінійну пружність зв'язків для утримання конфліктуючих брендів ліків на фіксованій оптимальній відстані один від одного;

– гравітаційна сила центрування, яка зміщує загальний центр мас усього обчисленого графа у геометричний центр візуального вікна; – сила радіальної колізії, яка динамічно враховує поточні радіуси кіл вузлів для уникнення візуального перекриття підписів.

Математична специфікація сили відштовхування частин Кулона F_c на відстані r між геометричними центрами двох вузлів описується формулою 2.2:

$$F_c = \frac{\varepsilon}{r^2} \quad (2.2)$$

де ε – коефіцієнт інтенсивності негативного заряду сутності, який динамічно адаптується під тип пристрою користувача;

r – відстань.

Сила натягу віртуальної пружини Гука F_h, яка діє вздовж ребра між суміжними медичними брендами, прагне повернути зв'язок до базової довжини l і розраховується за формулою 2.3:

$$F_h = \alpha * (r - l) \quad (2.3)$$

де α – жорсткість сформованого ребра взаємодії;

r – відстань;

l – базова довжина.

Таким чином, за цими формулами ми маємо побудувати взаємодії між ребрами.

2.5 Проєктування NoSQL графа знань Neo4j та реляційної структури PostgreSQL

Сучасні високопродуктивні медичні інформаційні системи потребують специфічних підходів до організації збереження даних. Традиційний підхід, який полягає у використанні виключно однієї реляційної системи керування базами даних, виявляється неефективним при одночасній обробці чітко структурованих транзакційних профілів користувачів та складних багатокомпонентних мережевих зв'язків біомедичних колізій. Для вирішення цієї проблеми в архітектурі інформаційної системи «PharmaGraph» реалізовано концепцію

Polyglot Persistence, що полягає в ізольованому розділенні та паралельному використанні двох різних за своєю природою СУБД: реляційного ядра PostgreSQL та NoSQL графової бази знань Neo4j.

Реляційне ядро архітектури розроблено на базі СУБД PostgreSQL і відповідає за транзакційну надійність системи, збереження облікових записів пацієнтів та поточного стану їхніх аптечок. Схему взаємозв'язків між сутностями представлено на рисунку 2.7.

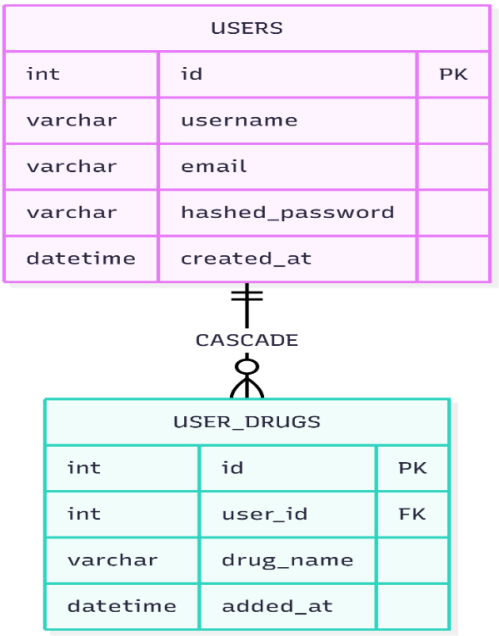

Рисунок 2.7 – Фізична схема бази даних PostgreSQL системи «PharmaGraph»

Для повноти опису реляційної структури розробимо детальні технічні специфікації для кожної таблиці, які визначають типи даних, ключі та обмеження на рівні полів. Першою є сутність users, яка слугує сховищем облікових записів пацієнтів системи. Атрибутивний склад таблиці користувачів містить такі елементи:

1. id (тип даних INTEGER) – унікальний ідентифікатор кожного користувача, який виступає як первинний ключ (Primary Key) та автоматично інкрементується системою за допомогою вбудованого лічильника;

2. username (тип даних VARCHAR) – унікальний текстовий логін користувача для авторизації, що має суворе обмеження на заборону порожніх значень (NOT NULL) та забезпечений індексом для прискорення пошуку профілю;

3. email (тип даних VARCHAR) – унікальна адреса електронної пошти користувача, яка проходить попередню валідацію на рівні Pydantic-схем, є обов'язковою для заповнення та захищеною від дублювання;

4. hashed_password (тип даних VARCHAR) – криптографічний рядок, що містить результат обробки пароля алгоритмом bcrypt, що унеможливлює компрометацію даних у разі витоку;

5. created_at (тип даних DATETIME) – часова мітка, яка автоматично фіксує точний системний час створення профілю в універсальному форматі UTC.

Аналіз логічної структури таблиці користувачів показує, що визначення параметрів унікальних індексів для полів логіна та електронної пошти ініціює автоматичне створення системних Б-дерев (B-Trees) на рівні дискової пам'яті PostgreSQL. Це оптимізує швидкість виконання операцій автентифікації та захищає систему від дублювання профілів.

Другою реляційною сутністю є таблиця user_drugs, яка виконує роль довготривалого транзакційного сховища медичних препаратів, доданих пацієнтом до своєї віртуальної аптечки. Ця сутність розроблена з урахуванням таких параметрів:

1. id (тип даних INTEGER) – первинний ключ запису, що забезпечує унікальність кожного факту додавання ліків до бази;

2. user_id (тип даних INTEGER) – зовнішній ключ (Foreign Key), який пов'язує запис препарату з конкретним власником з таблиці users. На рівні бази для цього поля налаштовано суворе каскадне обмеження ON DELETE CASCADE, яке гарантує автоматичне видалення всієї аптечки при видаленні профілю;

3. drug_name (тип даних VARCHAR) – повна текстова назва лікарського засобу, яка зчитується з форми інтерфейсу, забезпечена індексом для оптимізації аналітичних вибірок;

4. added_at (тип даних DATETIME) – дата та час внесення лікарського засобу до персонального списку перевірки сумісності.

Аналіз проектування реляційних зв'язків показує, що інтеграція між таблицями організована за допомогою зовнішнього ключа з суворим каскадним обмеженням ON DELETE CASCADE. Даний тригер реалізує автоматичне очищення транзакційного простору: при видаленні облікового запису користувача всі пов'язані з ним записи лікарських засобів анігілюються на рівні СКБД автоматично, запобігаючи накопиченню аномальних ізольованих рядків у пам'яті сервера.

Логічне забезпечення пулу з'єднань реляційного сегмента спирається на механізм фабрики сесій, конфігурований безпосередньо через об'єктно-реляційне відображення. Процес обробки транзакцій передбачає автоматичне відключення режимів авто-фіксації (autocommit) та автоматичного очищення буферів (autoflush) для запобігання передчасного внесення деструктивних змін у базу даних до моменту проходження повної валідації об'єкта. Життєвий цикл кожного окремого з'єднання з реляційним ядром жорстко контролюється на рівні генераторів контекстних менеджерів веб-сервера: утиліта гарантує обов'язкове закриття та деалокацію дескриптора сесії після остаточного завершення обробки поточного клієнтського HTTP-запиту, що унеможливлює виникнення витоків пам'яті або переповнення системного пулу підключень.

Для забезпечення безпечного обміну даними між клієнтським веб-інтерфейсом та сервером, а також для жорсткої типізації інформаційних потоків у системі спроєктовано логічні структури декларативних інструментів семантичної валідації. Вони виконують попередній синтаксичний аналіз даних до моменту їх передачі в аналітичні алгоритми ядра. Проєктна специфікація вхідних та вихідних інтерфейсних структур (схем) містить такі компоненти:

– структура реєстрації користувача (UserCreate) – вимагає обов'язкової наявності трьох рядкових сутностей (логін, адреса пошти, пароль). На рівні поля електронної пошти накладається обмеження стандарту RFC, яке автоматично відсікає некоректно введені адреси, що не містять символу роздільника або доменної зони;

– структура відповіді профілю (UserOut) – декларативно описує вихідний інформаційний пакет відповіді сервера, що містить ідентифікатор, логін та дату створення облікового запису. Ця структура навмисно повністю виключає поле пароля. Це гарантує, що навіть зашифрований криптографічний хеш ніколи не передається через відкриті канали мережі Internet, що відповідає міжнародним стандартам безпеки;

– структура авторизаційних сесій (Token/TokenData) – визначає будову цифрової перепустки, що містить строкові параметри типу токена, безпосередній рядок доступу та необов'язковий строковий ідентифікатор імені користувача для внутрішньої маршрутизації запитів.

Критично важливим аспектом проєктування реляційного сегмента баз даних є захист персональної інформації та управління безсесійним доступом пацієнтів за допомогою асиметричних криптографічних патернів. Безсесійна архітектура реалізується шляхом Singleton-конфігурування графового та реляційного драйверів з обмеженням часу життя токенів до 60 хвилин і підписом їх секретним інфраструктурним ключем за алгоритмом HS256. Криптографічний захист парольних рядків реалізується за допомогою незворотного кодування bcrypt, що повністю виключає можливість компрометації автентифікаційних даних пацієнтів у разі несанкційного копіювання образів сховища.

Безпека транзакційного контуру базується на взаємодії двох математичних технологій захисту:

1. Математична модель незворотного кодування паролів. Для унеможливлення компрометації автентифікаційних даних у разі фізичного витоку зліпка СУБД, система не оперує відкритими текстовими рядками. Замість цього впроваджується адаптивний алгоритм криптографічного захисту bcrypt.

Процедура трансформує відкритий пароль у байтовий потік, ініціалізує випадкову унікальну сіль за допомогою системних утиліт генерації і на основі розширеного ключа формує незворотний криптографічний хеш довжиною 60 символів, який і записується у полі hashed_password. Це повністю ліквідує ризик дешифрування даних методами перебору за допомогою заздалегідь обчислених «райдужних таблиць» (Rainbow Tables).

2. Модель авторизаційних токенів за стандартом JWT (JSON Web Token). Проєкт реалізує безсесійну (Stateless) архітектуру взаємодії. При успішному збігу паролів система генерує цифровий токен, який складається з трьох частин: заголовка (алгоритм шифрування $HS256$), корисного навантаження (ідентифікатор користувача та мітка часу автоматичного згасання сесії, яка обмежена 60 хвилинами) та фінального криптографічного підпису, створеного за допомогою унікального секретного ключа розробника. Це дозволяє серверу перевіряти легітимність кожного наступного HTTP-запиту пацієнта безпосередньо в оперативній пам'яті без виконання надлишкових та повільних запитів до таблиць PostgreSQL.

На відміну від реляційного сегмента, аналітичний контур інформаційної системи побудовано на базі NoSQL графової СУБД Neo4j версії сервера 2026.03.1. Вибір графової моделі зумовлений структурою медичних знань, де пошук взаємодій між тисячами діючих речовин потребує рекурсивного обходу мережі. У реляційних базах для цього довелося б виконувати ресурсомісткі та повільні операції об'єднання багатьох таблиць (JOIN). Граф типу Property Graph зберігає топологічні зв'язки безпосередньо у пам'яті як першокласні сутності, що забезпечує миттєвий пошук колізій поліпрагмазії незалежно від глибини вкладеності. Мета-структуру графа знань, зняту за допомогою процедури візуалізації, представлено на рисунку 2.8.

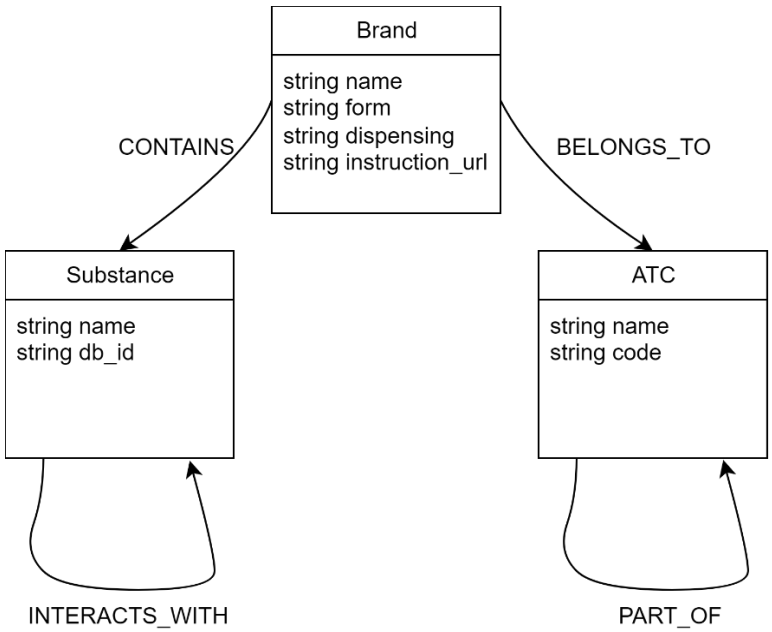

Рисунок 2.8 – Структура графа знань Neo4j СУБД

Реальна топологія розробленого графа знань налічує близько 38 тисяч активних вузлів та 1 489 014 спрямованих зв'язків. Модель складається з трьох фундаментальних типів вузлів, які наділені унікальними властивостями:

– Brand – уособлює офіційне комерційне найменування лікарського засобу, зареєстрованого в Україні, та містить властивості name (назва), form (форма випуску), dispensing (умови відпуску) та instruction_url (лінк на PDF);

– Substance – відображає міжнародну непатентовану назву діючої хімічної речовини, містить властивості name (латинська назва) та db_id (унікальний код у міжнародному реєстрі DrugBank);

– ATC – анатомо-терапевтично-хімічна сутність класифікатора ВООЗ, яка містить унікальний п'ятирівневий код групи code, та перекладену назву українською мовою name.

Внутрішня архітектура NoSQL масиву визначає не лише вузли, а й логіку спрямованих ребер, які зв'язують медичні дані в єдину мережу.

Взаємодія між вузлами графа регулюється чотирма чітко типізованими ребрами, які виконують такі функціональні завдання:

1. CONTAINS – спрямований зв'язок від вузла Brand до вузла Substance, який детально описує точний хімічний та компонентний склад конкретної торгової марки ліків;

2. INTERACTS_WITH – рефлексивний зв'язок у вигляді петлі, який виходить із сутності Substance і заходить у суміжні сутності того самого класу. Цей зв'язок містить публічні текстові властивості description та description_uk, де зафіксовано медичний опис наслідків клінічного конфлікту при сумісному прийомі речовин;

3. BELONGS_TO – спрямоване ребро від торгового бренда Brand до класифікаційного вузла ATC, що дозволяє миттєво визначити фармакологічну приналежність препарату;

4. PART_OF – рефлексивний зв'язок ієрархічного типу на вузлах ATC, за допомогою якого система будує деревоподібну структуру класифікатора ВООЗ від дрібних підгруп до глобальних терапевтичних розділів медицини.

Деталізована архітектурна структура об'єктного контуру, інструментів відображення бізнес-сутностей та декларативних засобів семантичної валідації інформаційної системи представлена у вигляді UML-діаграми класів на рисунку 2.9. Спроектована схема структурована за допомогою логічних пакетів (UML packages), що дозволяє чітко ізолювати зону відповідальності точки входу веб-сервера від шару довготривалого збереження даних пацієнтів та сервісного комплексу PharmaPipeline. Такий підхід забезпечує низький рівень зв'язності між окремими компонентами системи та сприяє підвищенню її модульності. Крім того, пакетна організація архітектури спрощує подальше супроводження програмного забезпечення, дозволяючи незалежно модифікувати окремі функціональні підсистеми без суттєвого впливу на інші компоненти. Представлена модель також створює передумови для масштабування системи шляхом додавання нових сервісів і доменних сутностей із мінімальними змінами існуючої структури.

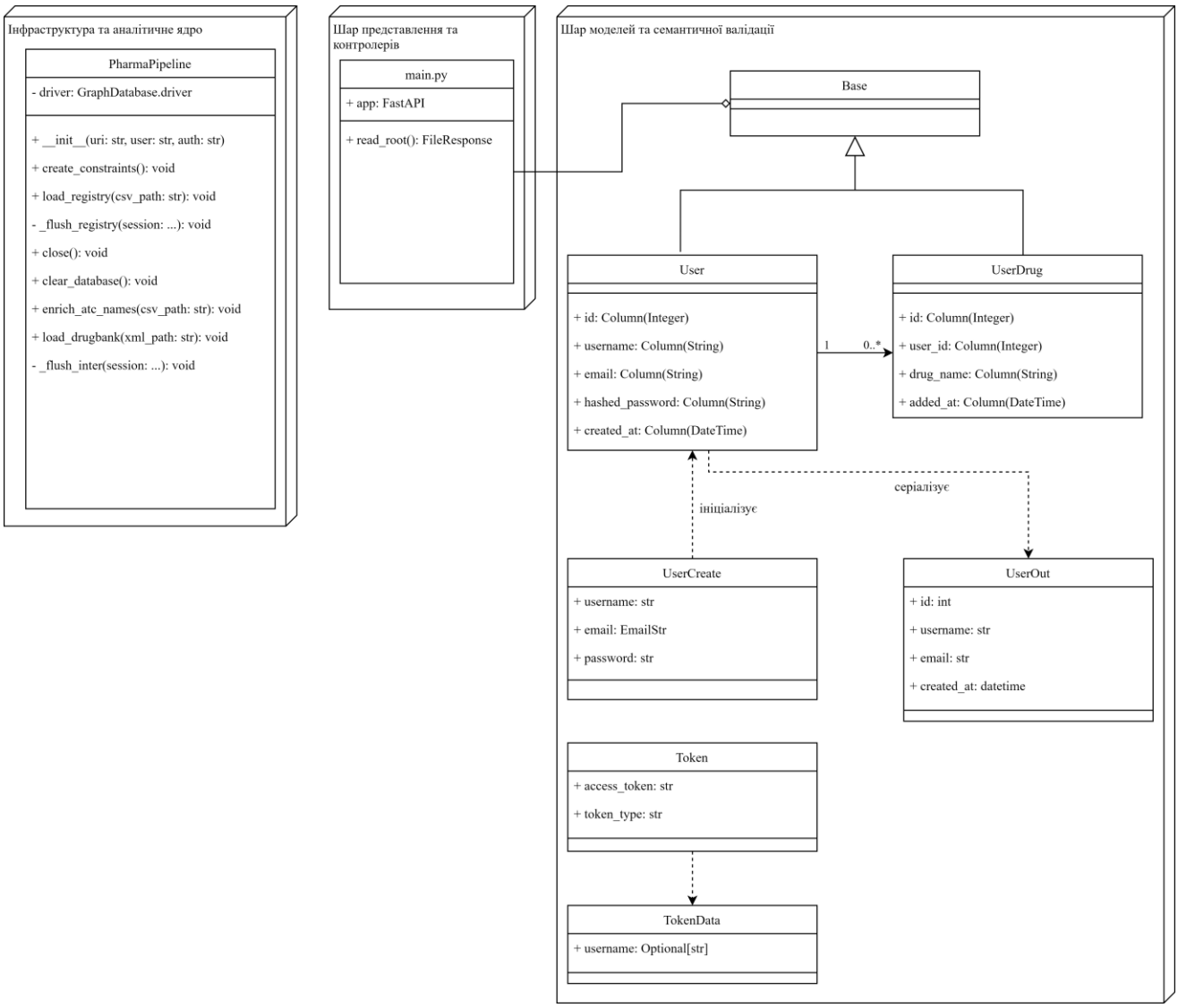

Рисунок 2.9 – UML-діаграма архітектурних пакетів та класів інформаційної системи

Аналіз розробленої структури об'єктно-орієнтованого моделювання (рис. 2.4) дозволяє обґрунтувати склад включених до неї сутностей та причини виключення інших файлів проєкту:

– пакет «Presentation & Controller Layer» (Шар представлення та контролерів): містить виключно декларативний клас точки входу main.py (FastAPIApp). Файли маршрутизації ендпоінтів (routes.py) та допоміжних аналітичних методів (logic.py) навмисно виключені з графічної схеми. Оскільки архітектура FastAPI базується на мікросервісному функціональному підході (декоратори маршрутів), вказані файли є наборами ізольованих процедур і

функцій, а не об'єктами ООП. Їхнє штучне відображення у вигляді класів суперечило б стандартам UML-проєктування;

– пакет «Data Access & Validation Layer» (Шар моделей та семантичної валідації) : поєднує системний базовий ORM-клас Base, від якого напряму успадковуються транзакційні сутності User та UserDrug , та класи даних Pydantic (UserCreate, UserOut, Token, TokenData). Дані класи Pydantic інкапсулюють у собі правила строгої типізації, проходять через вбудовані методи серіалізації (зокрема, конфігураційний тригер from_attributes = True) та пов'язані з моделями пунктирними векторами залежностей «ініціалізує» та «серіалізує». Утилітарний файл криптографічного захисту security.py виключено з діаграми, оскільки він містить лише чисті математичні функції кодування і не має власного об'єктного стану (state);

– пакет «Infrastructure & Analytics Core» (Інфраструктура та аналітичне ядро) : містить монолітний інфраструктурний клас PharmaPipeline, який уособлює логіку підготовки та міграції медичних знань. Клас містить приватний Singleton-атрибут драйвера підключення _driver, публічні методи очищення, створення унікальних констрайнтів та великі методи потокового парсингу першоджерел load_registry та load_drugbank.

2.6 Обґрунтування вибору програмного інструментарію та хмарної інфраструктури

Реалізація розроблених математичних моделей та алгоритмів потребує формування стабільного, масштабованого та архітектурно гнучкого програмно-апаратного комплексу. На етапі проєктування критично важливо підібрати інструментальні засоби, які забезпечують високу швидкість обробки асинхронних запитів, ізоляцію залежностей та безпеку збереження біомедичних даних.

Для реалізації серверної частини інформаційної системи «PharmaGraph» було обрано високорівневу мову програмування Python. Вибір даного інструменту зумовлений такими інженерними факторами:

– наявність розвиненої екосистеми спеціалізованих бібліотек та офіційних драйверів для безпосередньої взаємодії з графовими та реляційними СУБД;

– висока швидкість розробки та прототипування складних логічних модулів обробки даних;

– кросплатформеність, що дозволяє безперешкодно переносите серверний код між локальними машинами розробника та хмарними серверами розгортання.

У якості базового веб-фреймворку для побудови прикладного програмного інтерфейсу (API) обрано сучасний асинхронний фреймворк FastAPI. На відміну від традиційного Flask, який оперує у синхронному однопотоковому режимі, FastAPI базується на стандартах ASGI та технологіях Uvicorn, що забезпечує паралельну обробку тисяч мережевих з'єднань без блокування системних ресурсів. Важливою перевагою цього фреймворку є інтеграція з бібліотекою Pydantic, яка виконує автоматичну валідацію та серіалізацію вхідних JSON-пакетів до моменту їх передачі в аналітичні алгоритми, що повністю нівелює ризик ін'єкцій або передачі некоректних типів даних.

Для розгортання та керування базами даних впроваджено концепцію контейнеризації на базі платформи Docker. Це інженерне рішення дозволяє вирішити такі завдання:

– повна ізоляція середовища виконання та версій СУБД від конфігурації хостової операційної системи;

– швидке розгортання та реплікація інфраструктури за допомогою декларативних конфігураційних файлів Docker Compose;

– гарантія ідентичності роботи баз даних на локальній машині розробника та на продакшн-сервері.

У якості серверної операційної системи для довготривалого функціонування інформаційного комплексу обрано дистрибутив Linux Ubuntu Server. Дана операційна система характеризується високим рівнем безпеки,

низьким споживанням апаратних ресурсів та нативною підтримкою контейнеризованих інфраструктур Docker. Розгортання проєкту здійснюється на базі віртуального приватного сервера (VPS) з архітектурною конфігурацією, що містить два віртуальні процесори (vCPU) та чотири гігабайти оперативної пам'яті (RAM). Даний обсяг апаратних потужностей є оптимальним для стабільного утримання в кеші оперативної пам'яті понад одного мільйона зв'язків графової бази знань Neo4j та паралельного обслуговування транзакційного контуру PostgreSQL.

У межах другого розділу розроблено архітектурний проєкт інформаційної системи.Спроектовано гібридну структуру збереження даних, яка поєднує реляційну базу PostgreSQL для менеджменту кабінетів та графову базу знань Neo4j. Проведено поведінкове та часове UML-моделювання асинхронних потоків.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ВЕРИФІКАЦІЯ СИСТЕМИ «PHARMAGRAPH»

3.1 Конфігурація середовища розгортання та маніфестів Docker Compose

Фізична реалізація інформаційної системи «PharmaGraph» виконана у вигляді модульного веб-застосунку з чітким розділенням зон відповідальності між збереженням даних, парсингом першоджерел, бізнес-логікою бекенда та реактивним клієнтським інтерфейсом. Така архітектурна побудова спрощує процес налагодження, забезпечує високу швидкість тестування окремих компонентів та дозволяє масштабувати аналітичні модулі без зміни загального каркаса програми. Ієрархічну структуру каталогів та файлів розробленого програмного комплексу представлено на рисунку 3.1.

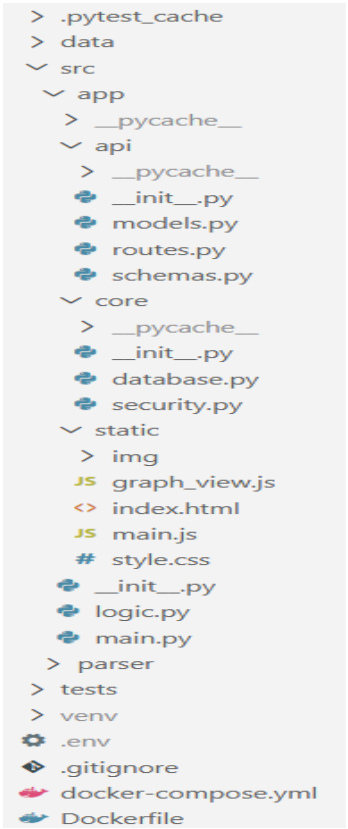

Рисунок 3.1 – Структурна схема розташування файлів та каталогів проєкту

Аналіз розробленої деревоподібної структури дозволяє детально декомпозувати призначення кожної папки та файлу в загальній екосистемі застосунку:

– .pytest_cache та tests/test_main.py – контур автоматизованого модульного тестування, який використовує фреймворк pytest для верифікації стабільності роботи FastAPI маршрутів перед деплоєм системи;

– data/ – ізольоване сховище первинних біомедичних даних, що містить сирий державний реєстр лікарських засобів у форматі CSV (reestr.csv), а також офіційні міжнародні класифікатори ВООЗ (who_atc_ddd.csv), які слугують базою для наповнення графа знань;

– src/app/api/ – підмодуль обробки мережевих запитів, що включає декларативні моделі SQLAlchemy (models.py), Pydantic-схеми валідації (schemas.py) та ізольовані ендпоінти веб-сервера (routes.py);

– src/app/core/ – системний каркас програми, де реалізовано логіку конфігурування пулу з'єднань із реляційною базою даних (database.py) та криптографічні функції безпеки для генерації токенів сесій (security.py);

– src/app/static/ – контур фронтенду, який містить статичну сторінку розмітки index.html, каскадні стилі Tailwind style.css, а також клієнтські скрипти управління інтерфейсом (main.js) та автономний модуль фізичної симуляції мережі взаємодій (graph_view.js);

– src/app/logic.py – головний аналітичний сервіс бекенда, в якому зосереджено алгоритми обходу графа Neo4j та функції інтеграції з ШІ;

– parser/ – автономний програмний комплекс імпорту, призначений для первинного очищення, нормалізації та завантаження CSV-реєстрів у графову базу знань, а також для синхронізації локальних сутностей із міжнародними ідентифікаторами DrugBank API.

Процес низькорівневого проєктування та автоматизованої збірки індивідуального образу для сервісу веб-сервера регламентується за допомогою конфігураційного маніфесту Dockerfile. На відміну від сторонніх

інфраструктурних сервісів СУБД, які розгортаються з готових хмарних репозиторіїв, аналітичний модуль додатка потребує покрокового опису середовища виконання, встановлення внутрішніх бібліотек та визначення стартової точки входу. Програмний код інструкцій збірки базового контейнера має такий вигляд:

```
FROM python:3.10.6-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "src.app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Аналіз архітектурних інструкцій представленого маніфесту збірки образу дозволяє виділити такі технологічні етапи формування ізольованого контейнера:

– дефініція базового образу FROM python:3.10.6-slim – використання оптимізованої та полегшеної версії дистрибутива slim дозволяє суттєво зменшити підсумковий розмір зліпка додатка, виключаючи з контуру збірки надлишкові системні утиліти операційної системи Linux, що підвищує швидкість розгортання та безпеку системи;

– конфігурування робочого простору WORKDIR /app – створює цільовий каталог усередині файлової системи контейнера, який слугує кореневою точкою для всіх наступних операцій копіювання та запуску модулів;

– оптимізація кешування залежностей через COPY requirements.txt . та RUN pip install розділення копіювання списку бібліотек від копіювання основного коду є класичним патерном проектування;

– перенесення коду та визначення команди запуску – директива COPY . . здійснює фінальне перенесення повної структури каталогів проєкту в ізольоване середовище, після чого інструкція CMD визначає запуск ASGI-сервера Uvicorn з прив'язкою до глобального інтерфейсу на внутрішній порт 8000.

Для забезпечення подальшої ізоляції інфраструктурних шарів, автоматизації розгортання та гарантії ідентичности середовища виконання на

етапах розробки й експлуатації впроваджено систему контейнеризації на базі інструменту Docker Compose. Декларативний конфігураційний файл docker-compose.yml оркеструє роботу трьох взаємопов'язаних сервісів, об'єднуючи їх у спільну віртуальну мережу. Програмна структура конфігураційного коду має такий вигляд:

```
services:
  app:
    build: .
    container_name: pharma_app
    ports:
      - "8000:8000"
    depends_on:
      - db
      - neo4j
    volumes:
      - .:/app
    working_dir: /app/src/app
    command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload
    environment:
      - PYTHONPATH=/app/src/app
      - DATABASE_URL=postgresql://admin:11111111@db:5432/pharma_users
      - NEO4J_URI=neo4j://neo4j:7687
      - NEO4J_USER=neo4j
      - NEO4J_PASSWORD=11111111
  db:
    image: postgres:15-alpine
    container_name: pharma_db
    restart: always
    environment:
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=11111111
      - POSTGRES_DB=pharma_users
    volumes:
      - postgres_data:/var/lib/postgresql/data
  neo4j:
    image: neo4j:2026.03-enterprise
    container_name: pharma_neo4j
    restart: always
    ports:
      - "7474:7474"
      - "7687:7687"
    environment:
      - NEO4J_AUTH=neo4j/11111111
      - NEO4J_ACCEPT_LICENSE_AGREEMENT=yes
      - volumes:
      - ${NEO4J_DATA_PATH}:/data
volumes:
  postgres_data:
```

Аналіз конфігураційних параметрів кожного інфраструктурного сервісу виявляє такі особливості архітектурного проектування середовища:

– сервіс app – відповідає за запуск веб-сервера FastAPI, ініціюючи автоматичну попередню збірку образу за описаним вище файлом Dockerfile. Контейнер зорієнтований на внутрішній робочий каталог /app/src/app та використовує механізм динамічного монтування томів для синхронізації змін коду в реальному часі;

– сервіс db – розгортає офіційній дистрибутив реляційного ядра. Використання мінімалістичного образу Alpine суттєво зменшує загальне навантаження на дискову підсистему сервера розгортання. Для запобігання втрати даних облікових записів пацієнтів при перезапуску контейнера, внутрішній каталог зберігання даних прив'язано до іменованого зовнішнього тому postgres_data;

– сервіс neo4j – ініціює роботу корпоративної графової бази знань версії 2026.03-enterprise. Для забезпечення коректної роботи аналітичного контуру конфігурація відкриває два мережеві порти, де порт 7474 призначений для HTTP-з'єднання з графічною веб-панеллю адміністрування СУБД, а порт 7687 є магістральним і використовує швидкісний бінарний протокол Bolt для асинхронного обміну даними з Python-драйвером сервера FastAPI.

Взаємозв'язок та черговість ініціалізації компонентів жорстко регулюється директивою depends_on. Даний параметр блокує старт веб-додатка до моменту переходу контейнерів реляційної та графової баз даних у стабільний робочий стан. Інтеграція між сервісами реалізована без прямого прокидання паролів у код через ін'єкцію динамічних змінних оточення DATABASE_URL та NEO4J_URI, що забезпечує хороший рівень криптографічної безпеки та гнучкості при зміні конфігурації хост-сервера.

3.2 Реалізація контуру пакетного завантаження та регулярного парсингу МНН

Функціонування аналітичної бази знань інформаційної системи потребує попереднього наповнення графової структури, інтеграції розрізнених державних реєстрів лікарських засобів та міжнародних біомедичних масивів у форматі XML. Для автоматизації процесів очищення, нормалізації, структурування та пакетного завантаження даних розроблено сервісний комплекс PharmaPipeline, зосереджений у модулі pipeline.py.

На початковому етапі роботи контуру завантаження здійснюється примусове очищення робочого простору NoSQL СУБД та декларативне формування обмежень цілісності (Constraints), які виконують роль унікальних індексів для прискорення пошукових операцій обходу графа. Програмна реалізація конфігурування обмежень має такий вигляд:

```
def create_constraints(self):
    with self.driver.session() as session:
        session.run("""
            CREATE CONSTRAINT brand_unique IF NOT EXISTS
            FOR (b:Brand) REQUIRE (b.name, b.form) IS UNIQUE
        """)
        session.run("""
            CREATE CONSTRAINT substance_unique IF NOT EXISTS
            FOR (s:Substance) REQUIRE s.name IS UNIQUE
        """)
        session.run("""
            CREATE CONSTRAINT atc_unique IF NOT EXISTS
            FOR (a:ATC) REQUIRE a.code IS UNIQUE
        """)
```

Аналіз представленого фрагмента коду дозволяє виділити такі інженерні особливості конфігурування бази знань:

– забезпечення унікальності сутностей – інструкція CREATE CONSTRAINT накладає суворі обмеження на рівні транзакційного ядра Neo4j. Для вузлів торгових брендів Brand створено складений унікальний тригер за двома властивостями (назва препарату та його лікарська форма), що унеможливлює дублювання запитів при повторних запусках імпорту;

– ідемпотентність збірки – оператор IF NOT EXISTS гарантує стабільність виконання скрипта, запобігаючи виникненню критичних виняткових ситуацій (Exceptions), якщо індекси вже були попередньо розгорнуті під час минулих ітерацій наповнення сервера.

Найбільш складним етапом первинної обробки є нормалізація та розщеплення комбінованих діючих речовин (МНН). Державний реєстр України містить рядки, де компоненти записані суцільним текстом через різні мовні роздільники. Для вирішення цієї проблеми в алгоритм інтегровано механізм регулярних виразів. Програмний код синтаксичного аналізу та пакетного скидання даних має такий вигляд:

```
      def load_registry(self, csv_path):
          df = pd.read_csv(csv_path, encoding='windows-1251', sep=';')
          batch = []
          delimiters = r', | and | та | \+ |/| combinations with | in combination
with | ; | і '

          with self.driver.session() as session:
              for _, row in df.iterrows():
                  sub_raw = str(row.get('Міжнародне непатентоване найменування',
'')).replace('*', '').lower().strip()

                  if sub_raw in ["mono", "nan", ""]:
                      subs = []
                  else:
                      subs = [s.strip() for s in re.split(delimiters, sub_raw,
flags=re.IGNORECASE) if s.strip()]
                      subs = [s for s in subs if s not in ['combinations',
'combinations excl. psycholeptics']]

                  # [Блок зчитування кодів АТС та наповнення масиву batch]
                  if len(batch) >= 1000:
                      self._flush_registry(session, batch)
                      batch = []
              if batch: self._flush_registry(session, batch)
```

Детальний архітектурний аналіз представленого алгоритму виявляє такі особливості лінгвістичної обробки фармацевтичних даних:

– гнучке розщеплення полікомпонентних ліків – розроблений патерн регулярного виразу delimiters враховує специфіку як україномовних роздільників (та, і, плюс), так і міжнародних анатомічних стандартів (and, in combination with, символи косої риски). Метод re.split здійснює динамічну

декомпозицію суцільного рядка на ізольовані хімічні субстанції з очищенням від системних маркерів та зірочок;

– патерн пакетної обробки (Batching) – ітераційний перебір тисяч рядків реєстру через утиліту pandas супроводжується накопиченням тимчасових об'єктів у масиві batch. При досягненні ліміту в 1000 записів ініціюється виклик приватного методу _flush_registry, що суттєво оптимізує споживання оперативної пам'яті та знижує кількість транзакційних блокувань бази даних.

Фінальний етап імпорту полягає у безпосередньому малюванні топології графа на рівні СУБД. Процедура виконується за допомогою оптимізованого Cypher-шаблону, який одночасно створює вузли, ребра зв'язків та рекурсивно розгортає п'ятирівневу структуру анатомо-терапевтично-хімічної класифікації ВООЗ. Програмний лістинг транзакційного збереження пакета має такий вигляд:

```
def _flush_registry(self, session, batch):
    session.run("""
    UNWIND $batch AS item
    MERGE (b:Brand {name: item.name, form: item.form})
    SET b.dispensing = item.dispensing, b.instruction_url = item.url
    WITH item, b
    UNWIND item.substances AS s_name
    MERGE (s:Substance {name: s_name})
    MERGE (b)-[:CONTAINS]->(s)
    WITH item, b
    UNWIND item.atc_codes AS a_code
    MERGE (a:ATC {code: a_code})
    MERGE (b)-[:BELONGS_TO]->(a)
    WITH a, a_code
    MERGE (l1:ATC {code: substring(a_code, 0, 1)})
    MERGE (l2:ATC {code: substring(a_code, 0, 3)})
    MERGE (l3:ATC {code: substring(a_code, 0, 4)})
    MERGE (l4:ATC {code: substring(a_code, 0, 5)})
    MERGE (a)-[:PART_OF]->(l4) MERGE (l4)-[:PART_OF]->(l3)
    MERGE (l3)-[:PART_OF]->(l2) MERGE (l2)-[:PART_OF]->(l1)
    """, batch=batch)
```

Інженерний аналіз оптимізації графового рендерингу виявляє такі системні закономірності:

– використання оператора MERGE – на відміну від базового CREATE, який безконтрольно створює нові елементи, MERGE попередньо сканує базу знань за допомогою унікальних індексів. Якщо вузол речовини Substance або коду АТС вже присутній у системі, оператор просто перевикористовує його,

прокладаючи нове ребро зв'язку CONTAINS або BELONGS_TO, що забезпечує мережеву цілісність графа;

– автоматичне трасування ієрархії ВООЗ – за допомогою вбудованої функції виділення підрядків substring алгоритм динамічно розбиває монолітний кінцевий код АТС (наприклад, п'ятий рівень класифікатора) на чотири вищі батьківські класи. Оператори MERGE (a)-[:PART_OF]->(l4) рекурсивно вибудовують деревоподібні ланцюжки зв'язків від конкретного препарату до глобальної терапевтичної групи медицини безпосередньо всередині дискової пам'яті Neo4j, формуючи закінчену інформаційну модель знань.

3.3 Логіка асинхронного аналітичного ядра та Cypher-маршрутизації FastAPI

Архітектурна побудова аналітичного ядра системи спирається на асинхронну взаємодію з NoSQL графовим ядром для забезпечення високої швидкості обробки запитів клієнта. Первинним технологічним етапом роботи з базою знань є забезпечення сервісних інформаційних запитів користувача щодо отримання детальних карт-паспортів конкретних лікарських засобів. Реалізація функцій первинної вибірки характеристик торгових брендів та асинхронного кешування перекладів має такий вигляд.

Даний підхід дозволяє мінімізувати затримки під час повторного звернення до однакових інформаційних ресурсів та забезпечує оптимізацію використання обчислювальних ресурсів серверної частини системи. Використання графової моделі даних також забезпечує ефективне встановлення зв’язків між лікарськими засобами, їх діючими речовинами, фармакологічними групами та можливими взаємодіями.

Реалізація функцій первинної вибірки характеристик торгових брендів та асинхронного кешування перекладів має такий вигляд:

```
    def get_drug_details(name_part, form_part):
        query = """
        MATCH (b:Brand {name: $name, form: $form})
        OPTIONAL MATCH (b)-[:CONTAINS]->(s:Substance)
        OPTIONAL MATCH (b)-[:BELONGS_TO]->(a:ATC)
        RETURN b.name as name,
               b.form as form,
               b.dispensing as dispensing,
               b.instruction_url as instruction_url,
               b.full_composition as composition,
               collect(DISTINCT s.name) as substances,
               collect(DISTINCT a.code) as atc_codes
        """
        with get_neo4j_session() as session:
            result = session.run(query, name=name_part, form=form_part)
            record = result.single()
            if record:
                data = dict(record)
                data['substance']    =    ",    ".join(data['substances'])    if
data['substances'] else "Не вказано"
                return data
            return None

    def translate_and_save(session, drug_a, drug_b, english_text):
        try:
            translated             =             GoogleTranslator(source='en',
target='uk').translate(english_text)
            save_query = """
            MATCH (s1:Substance)-[r:INTERACTS_WITH]-(s2:Substance)
            WHERE (s1.name = $name_a AND s2.name = $name_b)
               OR (s1.name = $name_b AND s2.name = $name_a)
            SET r.description_uk = $uk_text
            """
            session.run(save_query,        name_a=drug_a,        name_b=drug_b,
uk_text=translated)
            return translated
        except Exception as e:
            print(f"Помилка перекладу: {e}")
            return english_text
```

Детальний аналіз представленого програмного коду дозволяє виділити такі особливості реалізації низькорівневих інтерфейсів доступу до бази знань:

– оптимізація багатокомпонентних зв'язків у get_drug_details – декларативний запит мовою Cypher використовує оператори OPTIONAL MATCH для усунення критичних помилок при читанні брендів, які мають неповні або не заповнені зв'язки з класифікатором ВООЗ чи хімічними субстанціями;

– архітектурне рішення зворотного кешування у translate_and_save – процедура інтегрує зовнішній сервіс GoogleTranslator для усунення мовного бар'єру при роботі користувача з міжнародною англомовною базою DrugBank. Інженерна цінність методу полягає у використанні оператора SET безпосередньо

всередині транзакційної сесії. Програма не просто повертає адаптований рядок клієнту, а записує отриманий результат у властивість description_uk відповідного ребра графа.

Генеральним аналітичним вузлом усього програмного комплексу є функція check_drug_interactions. Її інженерна унікальність полягає у використанні концепції єдиного декларативного проходу по графовій структурі, що дозволяє за один транзакційний запит виявити три абсолютно різні за своєю клінічною природою типи медикаментозних колізій. Частина програмної реалізації центрального аналітичного ядра має такий вигляд:

```
query = """
        UNWIND $drugs AS d
        MATCH (b:Brand {name: d.n, form: d.f})
        OPTIONAL MATCH (b)-[:CONTAINS]->(s:Substance)
        WITH collect({display: b.name, sub: s, brand_node: b}) AS pairs

        UNWIND pairs AS p1
        UNWIND pairs AS p2
        WITH p1, p2, p1.sub AS s1, p2.sub AS s2
        WHERE elementId(p1.brand_node) < elementId(p2.brand_node)

        WITH p1, p2, s1, s2,
             (CASE WHEN s1 IS NOT NULL AND s2 IS NOT NULL AND elementId(s1) =
elementId(s2)
                   THEN s1.name ELSE NULL END) AS same_sub_name

        OPTIONAL MATCH (s1)-[r:INTERACTS_WITH]-(s2)

        WITH p1, p2, r, same_sub_name, p1.brand_node AS b1, p2.brand_node AS b2,
s1, s2
        OPTIONAL MATCH (b1)-[:BELONGS_TO]->(a1:ATC)
        OPTIONAL MATCH (b2)-[:BELONGS_TO]->(a2:ATC)

        WITH p1, p2, r, same_sub_name, a1, a2, s1, s2,
             substring(a1.code, 0, 4) AS g1, substring(a2.code, 0, 4) AS g2

        WHERE same_sub_name IS NOT NULL
           OR r IS NOT NULL
           OR (g1 IS NOT NULL AND g2 IS NOT NULL AND g1 = g2 AND same_sub_name IS
NULL)
```

Детальний архітектурний аналіз представленого коду виявляє такі особливості оптимізації графових обчислень на серверній стороні:

– нормалізація вхідних потоків – на початковому етапі алгоритм здійснює попередній розбір рядкових ключів, ізолюючи комерційне найменування ліків від форми випуску за допомогою методу розділення рядків.

Використання оператора очищення пробілів strip() гарантує точність подальшого зіставлення рядків, захищаючи логіку від випадкових помилок введення на фронтенді;

– векторизація запиту через UNWIND та collect – замість ітераційного виконання багатьох послідовних запитів до СУБД для кожного окремого медикаменту, весь масив даних передається у Neo4j одноразово як єдиний параметр $drugs. Оператор UNWIND трансформує цей вхідний JSON-список у табличний набір рядків для внутрішнього пошуку вузлів Brand та ребер CONTAINS, після чого агрегуюча функція collect формує вектор комбінаторних пар ліків усередині оперативної пам'яті сервера;

– усунення дублювання комбінацій – оператор порівняння внутрішніх системних ідентифікаторів elementId(p1.brand_node) < elementId(p2.brand_node) відсікає надлишкові дзеркальні дублікати пар ліків (наприклад, виключає одночасну перевірку пари Препарат_А + Препарат_Б та Препарат_Б + Препарат_А), а також унеможливлює порівняння лікарського засобу із самим собою, що знижує обчислювальне навантаження на процесор сервера рівно вдвічі;

– інтегроване трирівневе розгалуження колізій – умовний оператор CASE WHEN та утиліта виділення підрядків substring(a1.code, 0, 4) забезпечують паралельний аналіз трьох клінічних факторів небезпеки. Перший контур ідентифікує тотожність хімічних сутностей Substance для виявлення прихованого передозування. Другий контур через OPTIONAL MATCH трасує наявність прямих ребер конфліктів INTERACTS_WITH між різними діючими речовинами. Третій контур порівнює перші чотири символи анатомо-терапевтично-хімічного коду для виявлення ірраціональної поліпрагмазії, коли різні за складом ліки б'ють в одну й ту саму терапевтичну мішень організму. Фінальний блок WHERE відсікає безпечні комбінації, повертаючи клієнту лише верифіковані зони медичного ризику.

Невід'ємною сервісною частиною аналітичного модуля, що забезпечує коректне формування вхідних масивів для перевірки сумісності, є функція контекстного пошуку лікарських брендів search_brands. Вона відповідає за динамічне наповнення випадаючих списків (автокомпліту) на клієнтській стороне застосунку. Програмна реалізація алгоритму швидкої індексації торгових найменувань має такий вигляд:

```
    def search_brands(query_str: str, limit: int = 15):
        query = """
        MATCH (b:Brand)
        WHERE toUpper(b.name) CONTAINS toUpper($query_str)
        RETURN b.name AS name, b.form AS form
        ORDER BY size(b.name) ASC
        LIMIT $limit
        """
        with get_neo4j_session() as session:
            result = session.run(query, query_str=query_str, limit=limit)
            return [{"name": record["name"], "form": record["form"]} for record
in result]
```

Детальний аналіз представленого програмного коду дозволяє виділити такі особливості оптимізації та захисту пошукового контуру системи:

– нівелювання регістрозалежності – використання вбудованих функцій графової СУБД toUpper(b.name) та toUpper($query_str) приводить і назву медичного бренда в базі, і сирий пошуковий запит користувача до єдиного верхнього регістру. Це унеможливлює виникнення помилок при верифікації символів і гарантує успішний пошук ліків незалежно від увімкненої розкладки клавіатури або використання великих літер пацієнтом;

– застосування рядкового оператора CONTAINS – замість класичного жорсткого порівняння або використання складних регулярних виразів, які суттєво уповільнюють індексацію великих NoSQL масивів, оператор CONTAINS здійснює швидке підрядкове сканування властивостей вузлів. Це дозволяє знаходити комерційні препарати, навіть якщо користувач почав вводити назву з середини або з торговельної марки виробника;

– ранжування та лімітація результатів – сортування за допомогою виразу ORDER BY size(b.name) ASC виконує важливу ергономічну функцію:

вона виводить у топ пошукової видачі спочатку найкоротші та найбільш точні збіги назв ліків, що спрощує сприйняття інформації на екранах смартфонів. Директива LIMIT обмежує максимальну кількість повернутих вузлів п'ятнадцятьма записами, що мінімізує навантаження на мережевий канал зв'язку та запобігає переповненню клієнтського інтерфейсу надлишковими даними.

3.4 Інтеграція з генеративним контуром штучного інтелекту Google Gemini

Для розширення аналітичних можливостей системи та забезпечення максимальної зрозумілості отриманих результатів перевірки для кінцевого користувача, до загальної архітектури інтегровано контур штучного інтелекту. На відміну від базового алгоритму, який оперує сухими медичними термінами з бази знань, цей блок виконує роль інтелектуального перекладача клінічного контексту. Головним інструментом інтеграції виступає офіційний клієнт прикладного програмного інтерфейсу Google Gemini API. Програмна реалізація асинхронного маршруту експланаційного аналізу має такий вигляд.

Інтелектуальний модуль забезпечує генерацію структурованих пояснень щодо потенційних лікарських взаємодій, механізмів виникнення побічних ефектів та клінічних ризиків комбінованої терапії. Використання асинхронної взаємодії з AI-сервісом дає можливість зменшити час очікування відповіді та забезпечити стабільність роботи системи навіть при високому навантаженні на серверну інфраструктуру.

Програмна реалізація асинхронного маршруту експланаційного аналізу має такий вигляд:

```
@router.get("/ai-explain")
async def ai_explain(drug_a: str, drug_b: str, context: str):
    prompt = f"""
    Ти – турботливий, професійний сімейний лікар та асистент системи PharmaGraph.
    Користувач приймає одночасно два препарати: {drug_a} та {drug_b}.
    База даних виявила такий конфлікт між їхніми діючими речовинами: "{context}".
    Твоє завдання – пояснити суть цього конфлікту максимально простими словами...
    НАПИШИ ВІДПОВІДЬ СУВОРО ЗА ТАКИМ ПЛАНОМ:
    1. ЩО ВІДБУВАЄТЬСЯ В ОРГАНІЗМІ: Поясни "на пальцях" суть небезпеки.
    2. НА ЯКІ СИМПТОМИ ЗВЕРНУТИ УВАГУ...
    3. ПОБУТОВІ ОБМЕЖЕННЯ ТА ЇЖА...
    4. ГОЛОВНА ПОРАДА...
    КРИТИЧНІ ОБМЕЖЕННЯ ФОРМАТУВАННЯ:
    - Категорично заборонено використовувати Markdown-символи: "###", "*", "**".
    - Текст повинен бути лаконічним, написаним короткими абзацами.
    - Використовуй прості списки через дефіс (-).
    - Відповідай виключно українською мовою.
    """
    try:
        response = client.models.generate_content(
            model="gemini-3.1-flash-lite",
            contents=prompt
        )
        return {"explanation": response.text}
    except Exception as e:
        print(f"Критична помилка ШІ PharmaGraph: {e}")
```

Детальний інженерний аналіз представленого фрагмента коду та специфіки промпт-дизайну дозволяє виділити такі важливі технологічні особливості:

– архітектурна ізоляція ендпоінти – маршрут розроблено як незалежний асинхронний сервіс, що викликається клієнтською стороною за допомогою окремого HTTP-запиту за методом GET. Це унеможливлює виникнення затримок під час генерації тексту основним аналітичним контуром і дозволяє системі миттєво віддавати топологію графа, залишаючи обчислення ШІ для фонового рендерингу;

– використання спеціалізованої моделі – для генерації відповідей застосовано високопродуктивну велику мовну модель gemini-3.1-flash-lite. Вибір цієї моделі зумовлений її оптимізацією під завдання швидкої обробки контексту з мінімальним часом очікування відповіді (Time-to-First-Token), що є критично важливим для збереження плавності інтерфейсу на мобільних пристроях;

– суворі інженерні обмеження промпт-дизайну – інструкція для ШІ містить жорсткі правила фільтрації контенту та форматування текстового потоку. Заборона на використання Markdown-символів, таких як зірочки або решітки, зумовлена специфікою приймаючого об'єкта на фронтенді, де сирий текст трансформується методом заміни переносів рядків на HTML-теги абзаців. Наявність жорсткої чотирирівневої структури плану (аналіз небезпеки, симптоматика, побутові обмеження, фінальне застереження) гарантує стабільність поведінки ШІ та виключає генерацію галюцинацій;

– лінгвістична адаптація термінології – промпт зорієнтований на повне виключення складних латинських чи англомовних клінічних дефініцій, які можуть дезорієнтувати або налякати пацієнта без медичної освіти. Модель успішно перекладає внутрішній технічний опис колізії, отриманий з ребра INTERACTS_WITH бази знань Neo4j, на просту повсякденну мову, підвищуючи рівень безпеки самолікування та усвідомленості пацієнта.

3.5 Розробка інтерфейсу користувача та превентивного екранування символів

Клієнтський контур інформаційної системи розроблено на основі концепції Single Page Application (SPA) з використанням декларативної верстки стандарту HTML5, стилістичного інструментарію Tailwind CSS та мови програмування JavaScript. Головним завданням цього шару є забезпечення миттєвого відгуку інтерфейсу на дії користувача, організація асинхронного обміну даними за допомогою API-запитів (технологія Fetch) та динамічний рендеринг елементів керування без повного перезавантаження сторінки браузера.

Базова архітектура інтерфейсу описується у файлі index.html, де реалізовано модульну сітку, що складається з пошукового магістрального поля, блоку відображення карток персональної аптечки користувача, інтерактивного

інтегрованого полотна для малювання графа колізій та модальних вікон деталізації ієрархії ВООЗ.

Процес динамічного формування інтерфейсних карток доданих лікарських засобів та забезпечення безпеки обробки текстових даних реалізовано у файлі main.js. Ключовий програмний фрагмент контуру рендерингу та екранування символів має такий вигляд:

```
searchInput.addEventListener('input', async (e) => {
    const q = e.target.value;
    if (q.length < 2) { suggestions.classList.add('hidden'); return; }
    try {
        const res = await fetch(`/search?q=${encodeURIComponent(q)}`);
        const data = await res.json();
        suggestions.innerHTML = data.map(item => {
            const sName = item.name.replace(/'/g, "\\'").replace(/"/g,
"&amp;quot;");
            const sForm = item.form.replace(/'/g, "\\'").replace(/"/g,
"&amp;quot;");
            const displayName = item.name.replace(/"([^"]*)"/g,
'«$1»').replace(/"/g, '«');
            return `
                <div class="p-3 suggestion-item border-b cursor-pointer
hover:bg-sky-50"
                     onclick="addDrug('${sName}', '${sForm}')">
                    <div class="font-bold text-slate-
800">${displayName}</div>
                    <div class="text-[10px] text-slate-400
uppercase">${item.form}</div>
```

Детальний архітектурний аналіз представленого фрагмента клієнтського коду дозволяє виділити такі особливості реалізації веб-інтерфейсу:

– превентивне лінгвістичне екранування – використання ланцюжка методів replace(/'/g, "\\'") та replace(/"/g, """) вирішує критичну проблему сумісності синтаксису. При наявності у назві медичного бренда прямих комп'ютерних лапок (наприклад, 5-ФТОРУРАЦИЛ "ЕБЕВЕ"), сирий рядок розриває межі інлайнового HTML-атрибута onclick, що призводить до виникнення фатальної помилки розбору токенів SyntaxError. Заміна лапок на текстові мнемоніки " дозволяє браузеру коректно інтерпретувати межі властивості та безперешкодно передавати оригінальний текстовий рядок у функцію додавання ліків addDrug;

– автоматична типізація відображення – регулярний вираз replace(/"([^"]*)"/g, '«$1»') виконує важливу функцію естетичної та мовної нормалізації інтерфейсу. Система автоматично замінює прямі лапки на типографічні французькі «лапки-ялинки» виключно всередині текстового контейнера відображення displayName, залишаючи технічні параметри у вихідному реєстровому вигляді;

– асинхронна оптимізація введення – обробник події input починає взаємодію з сервером лише за умови введення більше ніж двох символів. Використання утиліти encodeURIComponent гарантує безпечне кодування специфічних символів та пробілів у рядку запиту, захищаючи аналітичний контур від помилок передачі параметрів (Bad Request) при виконанні фонових мережевих запитів до FastAPI.

Весь код проєкту наведений в додатках А, Б, В та Г.

3.6 Функціональна верифікація інтерфейсу на клінічних сценаріях поліпрагмазії

Верифікація працездатності розробленого програмного комплексу здійснювалася шляхом проведення перевірки користувацького інтерфейсу на реальних клінічних сценаріях поліпрагмазії. Головною метою цього етапу є експериментальне підтвердження того, що серверні алгоритми, NoSQL база знань Neo4j та фронтенд-скрипти динамічного рендерингу D3.js безпомилково взаємодіють між собою і коректно відображають усі типи медичних ризиків.

Першим етапом випробувань стала перевірка базового контуру контекстного пошуку (автокомпліту). Результат успішного інтерактивного підбору лікарського засобу (5-ФТОРУРАЦИЛ "ЕБЕВЕ") представлено на рисунку 3.2.

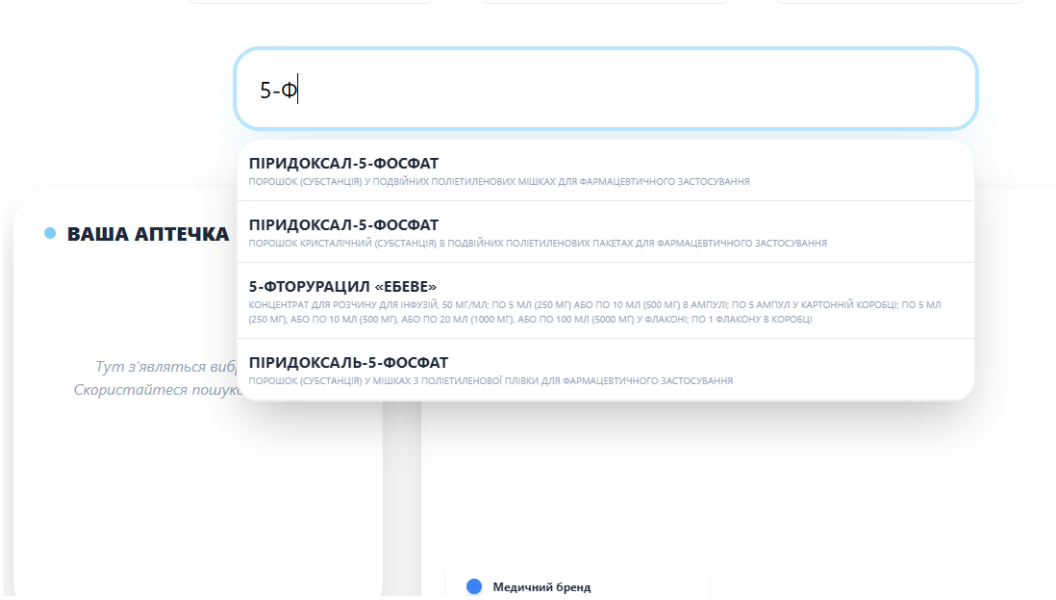

Рисунок 3.2 – Візуалізація результату роботи модуля контекстного пошуку та нормалізації назв ліків

Аналіз результатів першого етапу тестування доводить, що розроблені клієнтські скрипти повністю ліквідували вразливість синтаксису JavaScript. (рис.3.3).

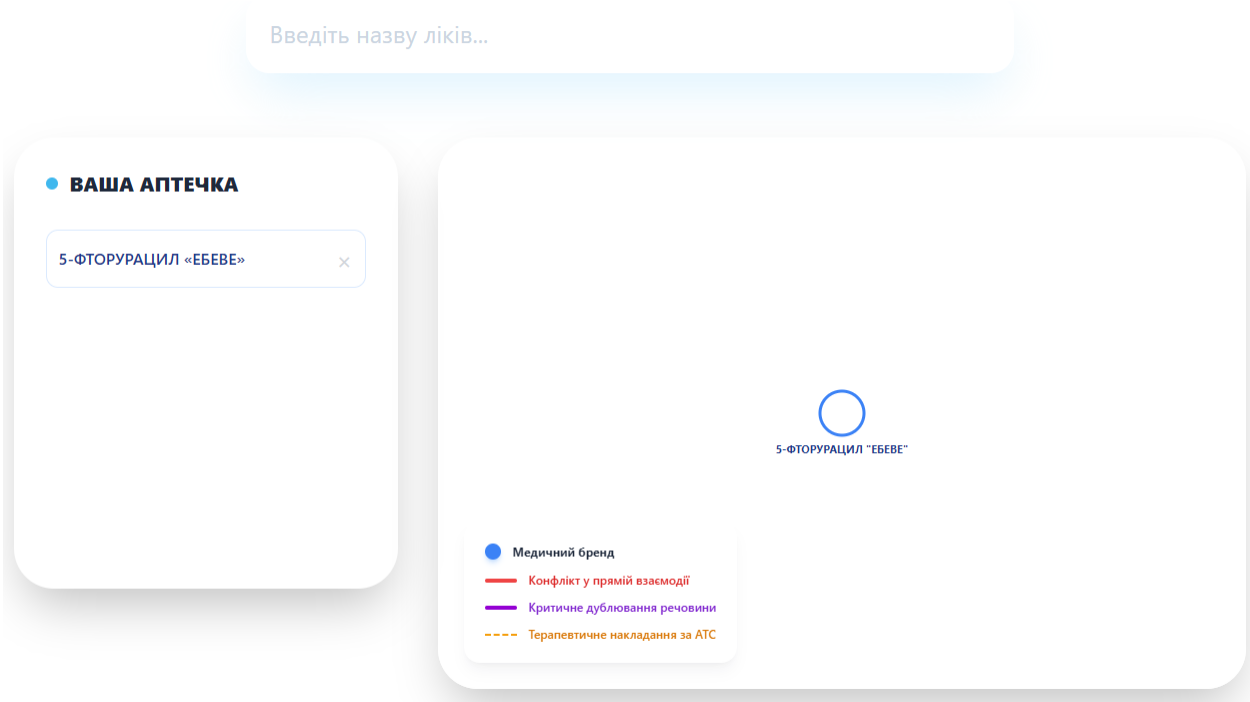

Рисунок 3.3 – Успішне додавання препарату

Другим етапом стала перевірка першого типу медичних колізій – критичного дублювання діючих хімічних речовин (МНН) під різними торговими брендами. Для цього в контур аналізу було одночасно додано два препарати, які мають різні комерційні назви, але містять ідентичну базову субстанцію. Результат детекції та візуалізації цього ризику представлено на рисунку 3.4.

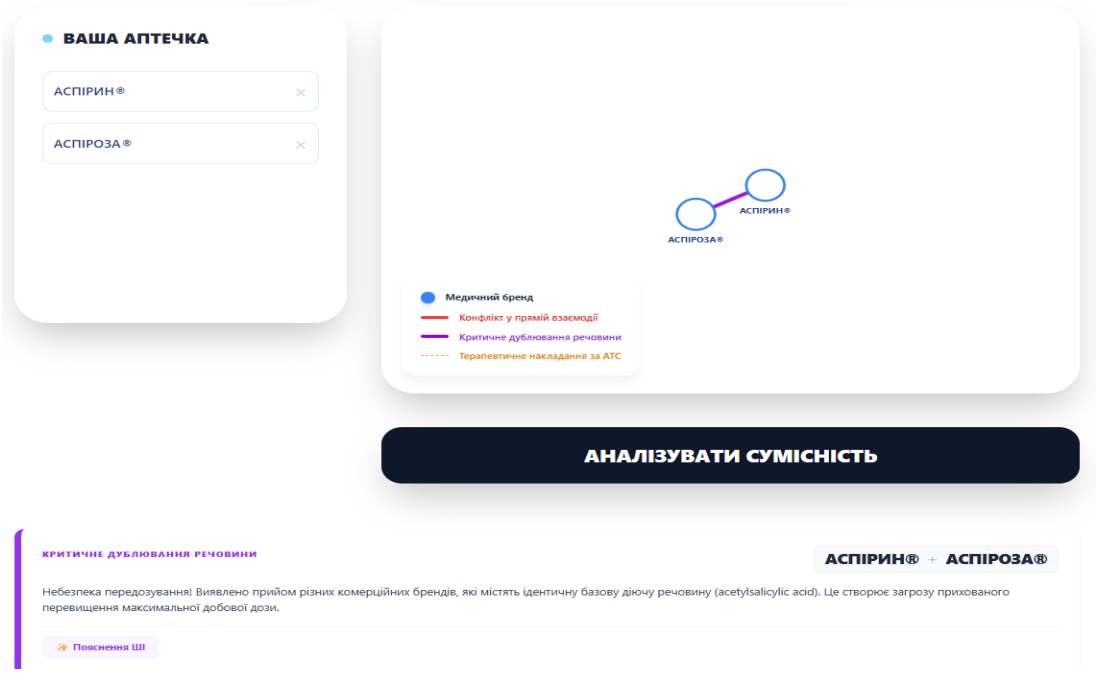

Рисунок 3.4 – Результат перевірки системи при виявленні дублювання діючої речовини

Аналіз вихідних даних на рисунку 3.4 підтверджує працездатність серверного логічного блоку CASE WHEN та утиліти порівняння системних ідентифікаторів elementId(s1) = elementId(s2). Система безпомилково ідентифікувала приховане дублювання складу та забарвила зв'язок на графі у фіолетовий колір.

Третім етапом випробувань став запуск сценарію детекції прямих міжмедикаментозних взаємодій (клінічних конфліктів) на біохімічному рівні. Для тестування в систему було введено пару ліків, сумісний прийом яких категорично заборонений міжнародними протоколами через високу токсичність. Результат роботи аналітичного контуру представлено на рисунку 3.5.

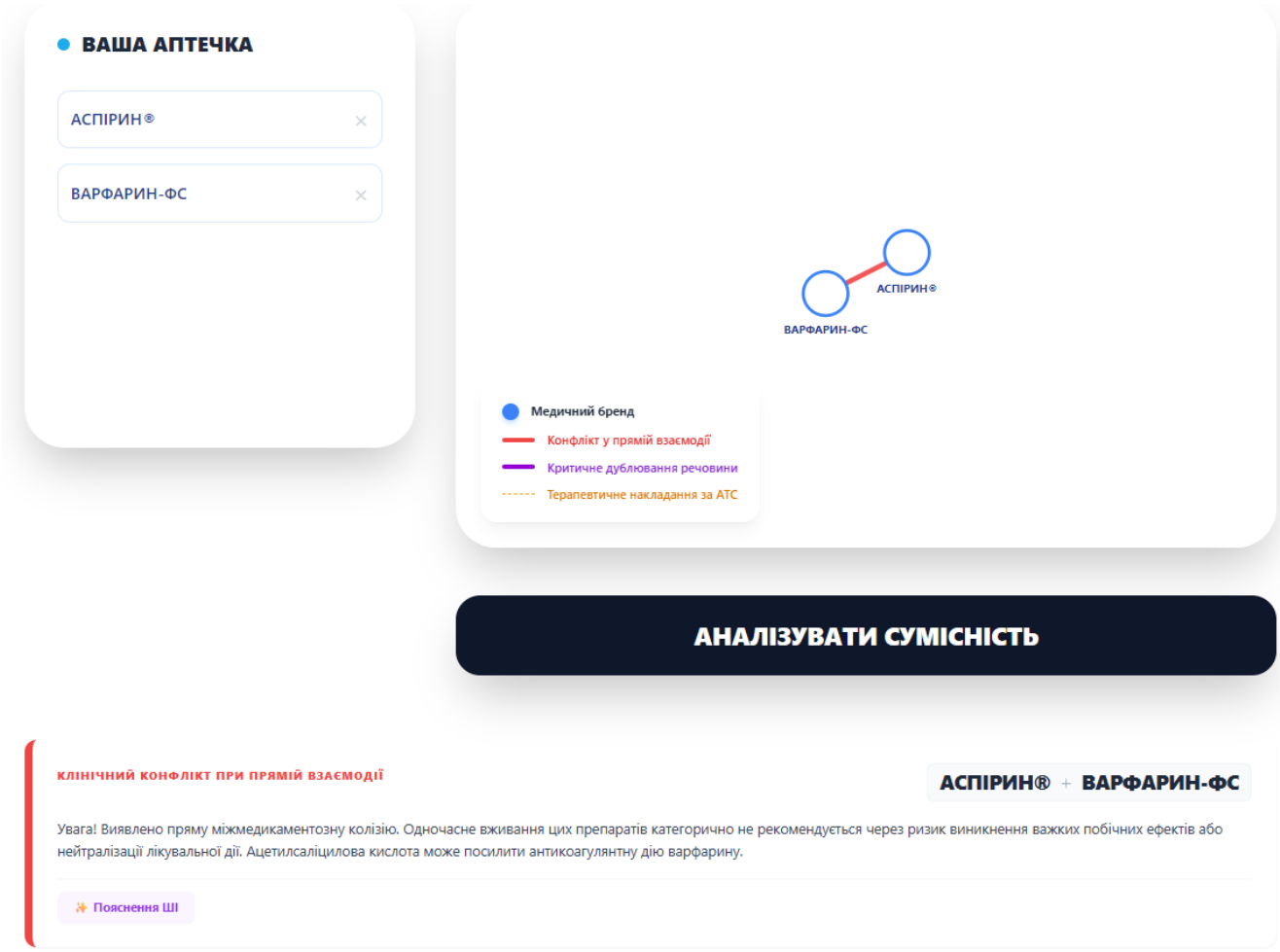

Рисунок 3.5 – Візуальне представлення результату детекції прямої міжмедикаментозної взаємодії

Експериментальний запуск підтверджує точність відпрацювання ребра графа INTERACTS_WITH. Програма миттєво зчитала зашидований англомовний опис з DrugBank, задіяла підмодуль автоматичного кешування перекладу та вивела пацієнту локалізоване попередження, підсвітивши магістраль ризику на графі критичним червоним кольором.

Четвертим етапом стала перевірка третього типу колізій – терапевтичного накладання ліків за міжнародною класифікацією ВООЗ. У контур аналізу було додано два різні за хімічним складом препарати, які, проте, належать до однієї

фармакологічної групи за кодом АТС. Результат роботи системи представлено на рисунку 3.6.

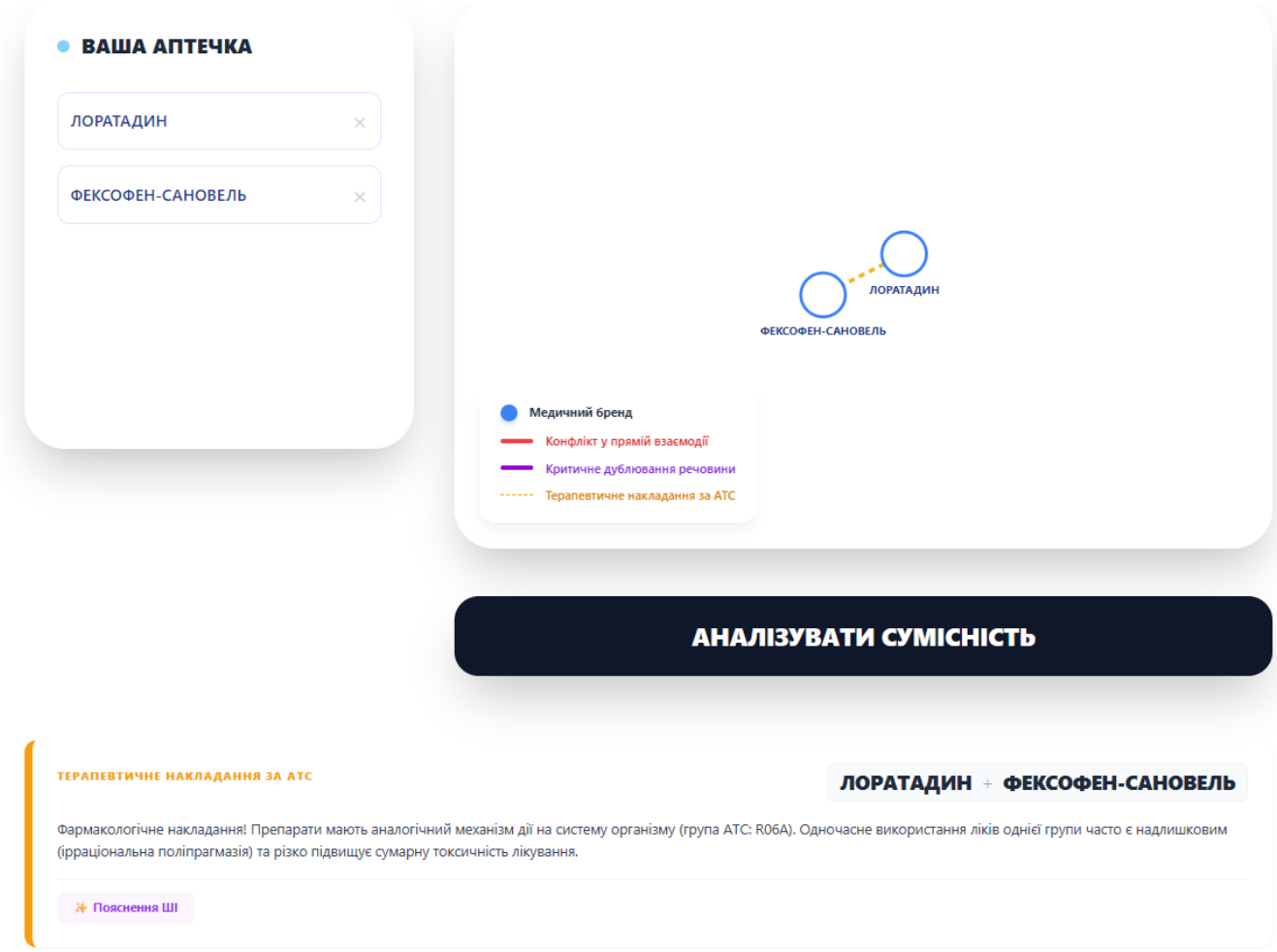

Рисунок 3.6 – Результат роботи контуру детекції терапевтичного накладання за АТС-кодом

Результати доводять високу точність роботи алгоритму зрізання рядків substring(a1.code, 0, 4). Система успішно порівняла перші чотири символи чотирьохрівневої ієрархії класифікатора, виявила ірраціональну поліпрагмазію (коли ліки б'ють в одну й ту саму точку організму) та згенерувала попереджувальний зв'язок у вигляді помаранчевої пунктирної лінії.

П'ятим, завершальним етапом ручної верифікації аналітичного контуру став запуск сценарію інтелектуальної підтримки пацієнта для розшифрування виявленого терапевтичного накладання. Після успішного відпрацювання

алгоритму порівняння АТС-кодів та виведення помаранчевої карти ризику, було здійснено натискання інтерактивної кнопки «Пояснення ШІ». Ця дія ініціює асинхронний GET-запит до ізольованого нейромережевого ендпоінта /ai-explain. Результат успішного фонового рендерингу та адаптації медичного висновку мовною моделі представлено на рисунку 3.7.

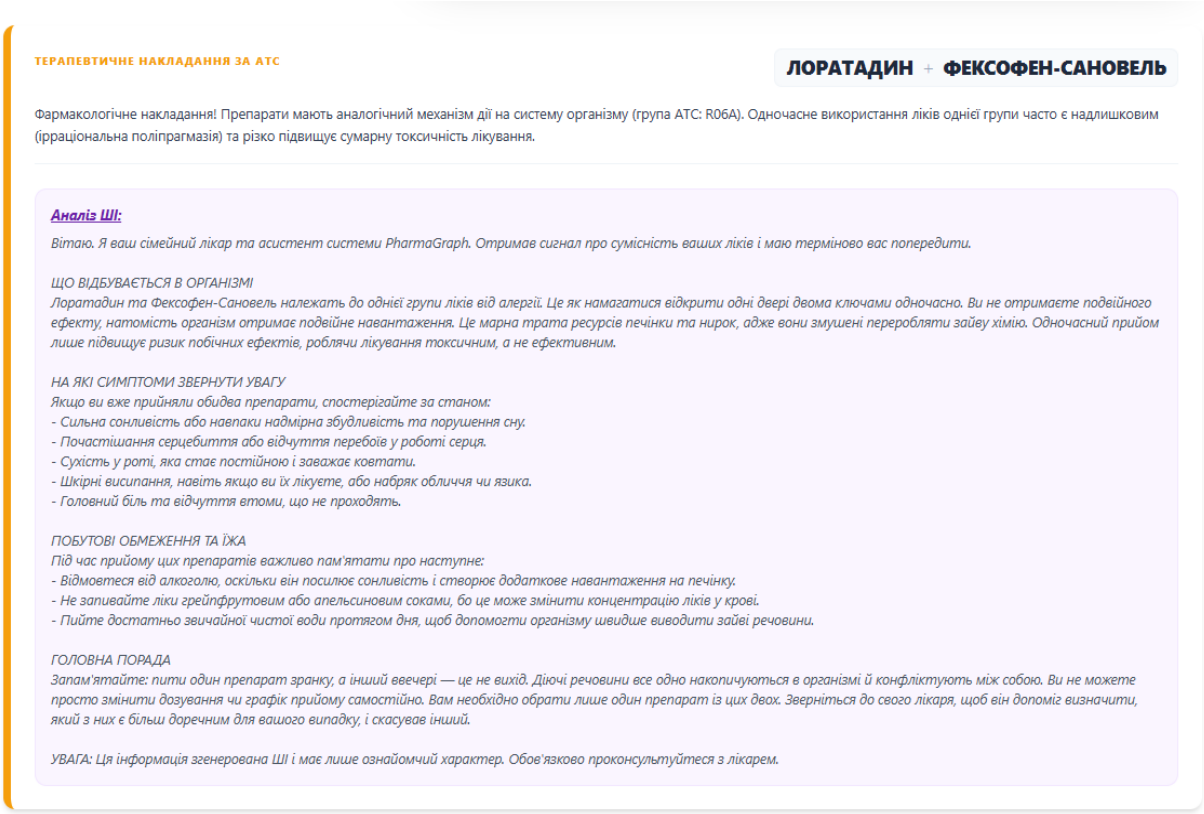

Рисунок 3.7 – Інтерфейс відображення адаптованого медичного пояснення від мовної моделі

Експериментальний аналіз вихідного текстового масиву на рисунку 3.7 доводить, що розроблений інженерний промпт-дизайн повністю виконує критерії безпеки та обмеження форматування. Велика мовна модель gemini-3.1-flash-lite успішно зчитала сухий технічний контекст взаємодії, трансформувала його у лаконічні абзаци виключно українською мовою та замінила складні патофізіологічні поняття на прості побутові аналоги. Чітка чотирирівнева структура (суть небезпеки «на пальцях», тригери симптомів, дієтичні обмеження

та головна порада) виключає виникнення логічних галюцинацій, а фінальний обов'язковий дисклеймер застерігає користувача від самовільної зміни медичних призначень.

Таким чином, проведене комплексне ручне тестування підтвердило повну функціональну працездатність та високу клінічну точність взаємодії всіх компонентів інтерфейсу та аналітичного ядра інформаційної системи.

3.7 Автоматизоване модульное тестування серверних ендпоінтів за допомогою Pytest

Важливою складовою забезпечення надійності, відмовостійкості та регресійної стабільності інформаційної системи є контур автоматизованого модульного тестування (Unit Testing). На відміну від ручних функціональних випробувань інтерфейсу, автоматизоване тестування дозволяє ізольовано перевірити логіку роботи кожного окремого серверного маршруту FastAPI, верифікувати коректність обробки вхідних JSON-пакетів та валідацію статусів відповідей.

У якості базового інструментарію автоматизації було обрано високорівневий фреймворк pytest у зв'язці зі спеціалізованим тестовим клієнтом TestClient, який імітує виконання реальних асинхронних HTTP-запитів до веб-сервера. Програмна реалізація конфігурування тестових заглушок та верифікації магістральних аналітичних ендпоінтів у файлі test_main.py має такий вигляд. Тестові сценарії побудовані за принципом ізоляції зовнішніх залежностей, що забезпечує стабільність та відтворюваність результатів незалежно від стану сторонніх сервісів. Для підміни реальних викликів використовуються механізми мокування, які дозволяють контролювати вхідні та вихідні дані під час виконання тестів.

```
import pytest
from unittest.mock import MagicMock, patch
from fastapi.testclient import TestClient
from main import app
from api.routes import get_current_user
from core.database import get_pg_db, get_neo4j_session

client = TestClient(app)
shared_db_mock = MagicMock()

def mock_get_current_user():
    class MockUser:
        id = 1
        username = "test_user"
    return MockUser()

def mock_get_pg_db():
    try: yield shared_db_mock
    finally: pass

app.dependency_overrides[get_current_user] = mock_get_current_user
app.dependency_overrides[get_pg_db] = mock_get_pg_db

@patch("api.routes.check_drug_interactions")
def test_check_safety_endpoint(mock_check):
    mock_check.return_value = [{
        "drug_a": "АСПІРИН", "drug_b": "ВАРФАРИН",
        "type": "Direct Interaction",
        "description": "Посилення антикоагулянтної дії."
    }]
    response = client.post("/check", json=["АСПІРИН|таблетки",
"ВАРФАРИН|таблетки"])
    assert response.status_code == 200
    assert response.json()["status"] == "success"
    assert response.json()["found_conflicts"] == 1
```

Детальний архітектурний аналіз представленого програмного коду модульних тестів дозволяє виділити такі інженерні особливості реалізації:

– повна інфраструктурна ізоляція середовища – за допомогою стандартного модуля unittest.mock.patch реалізовано перехоплення системних функцій ініціалізації баз даних (create_engine, GraphDatabase.driver). Це дозволяє виконувати повний цикл тестів безпосередньо у процесі CI/CD збірки контейнерів без реального підключення до серверів PostgreSQL чи Neo4j, що гарантує автономність верифікації коду;

– механізм перевизначення залежностей (Dependency Overrides) – утиліта app.dependency_overrides дозволяє динамічно підмінити реальні функції автентифікації користувача та фабрики сесій на легковажні мок-заглушки mock_get_current_user та mock_get_pg_db. Це нівелює потребу генерувати

справжні сесійні токени JWT при тестуванні захищених маршрутів кабінету пацієнта;

– імітація клінічних сценаріїв – тест test_check_safety_endpoint перевіряє працездатність центрального аналітичного шлюзу. За допомогою інструкції mock_check.return_value утиліті штучно підкладається верифікований масив міжмедикаментозного конфлікту. Оператори тверджень assert суворо звіряють код відповіді сервера (очікується HTTP 200 OK) та внутрішню структуру результуючого JSON-пакета, гарантуючи, що бізнес-логіка не зазнала деструктивних змін при рефакторингу.

Запуск розробленого тестового набору здійснюється через консольний термінал операційної системи за допомогою виклику утиліти pytest -v. Процедура автоматично сканує каталог tests/, ініціює виконання всіх ізольованих функцій, перевіряє успішність авторизації, логіку реєстрації, роботу проксі-парсерів та автокомпліту. Результат можна побачити на рисунку 3.8.

Рисунок 3.8 – Результат роботи тестів

Результати консольного виведення свідчать про успішне проходження 100 % закладених модульних тестів (статус PASSED для всіх контурів), що документально підтверджує стабільність, безпеку та високу технічну якість реалізованого програмного комплексу.

У третьому розділі успішно реалізовано та верифіковано програмний комплекс системи «PharmaGraph» у середовищі контейнеризації Docker Compose. Розроблено асинхронне аналітичне ядро на базі FastAPI та мови Cypher для трирівневої детекції колізій, а також інтегровано контур ШІ Google Gemini для трансформації клінічних висновків у доступну побутову форму.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи успішно здійснено повний цикл проєктування, алгоритмізації, програмної реалізації та верифікації спеціалізованої інформаційної системи інтелектуального аналізу сумісності лікарських засобів «PharmaGraph». Актуальність проведеного дослідження зумовлена стрімким зростанням масивів розрізненої медико-фармацевтичної інформації та критичною потребою мінімізації клінічних ризиків при призначенні комплексних схем терапії. Створений програмний комплекс вирішує важливу науково-практичну задачу автоматизованого виявлення колізій поліпрагмазії та прихованого передозування, надаючи кінцевому користувачу інтерактивні інструменти візуалізації топології графів знань і доступні експланаційні висновки з використанням інноваційних генеративних технологій.

Під час виконання першого розділу роботи проведено системний аналіз предметної області, досліджено специфіку міжмедикаментозних взаємодій та вивчено наявні світові технологічні аналоги у сфері цифровізації охорони здоров'я. Обґрунтовано доцільність відходу від класичних монолітних архітектур на користь гнучких модульних рішень та визначено фундаментальні вимоги до захисту персональних біомедичних даних. На основі проведеного аналізу сформовано концептуальний стек технологій розробки, де ключовий акцент зміщено на використання сучасного асинхронного веб-фреймворку FastAPI, високопродуктивної графової системи керування базами даних Neo4j, реляційного сховища PostgreSQL та клієнтського інструментарію динамічного моделювання фізичних мереж.

У межах другого розділу розроблено архітектурний проєкт інформаційної системи, що базується на патерні розділення баз даних Polyglot Persistence для збалансованого управління транзакційними та аналітичними потоками. Побудовано концептуальні, логічні та фізичні моделі даних, детально задокументовано атрибутивний склад ієрархічних таблиць користувачів і специфіковано мета-структуру графа знань, яка налічує тисячі активних вузлів

медичних брендів, діючих хімічних речовин та анатомо-терапевтично-хімічних кодів класифікатора ВООЗ. Особливу увагу приділено теоретичному обґрунтуванню моделей транзакційного захисту інформації, що включають безсесійний Stateless-доступ на базі підписаних токенів JWT та незворотне криптографічне кодування парольних рядків за допомогою адаптивного алгоритму криптографічного захисту bcrypt із динамічним засіванням солі.

Практична реалізація та деплой програмного комплексу, детально описані в третьому розділі, підтвердили високу життєздатність обраних інженерних рішень та забезпечили створення повнофункціонального веб-застосунку. Розроблено автономний контур первинної обробки й імпорту даних, який за допомогою пакетних сценаріїв і регулярних виразів успішно нормалізує складні полікомпонентні лікарські засоби державного реєстру та синхронізує локальні сутності з міжнародними аналітичними реєстрами. Спроєктовано унікальний трирівневий декларативний запит мовою Cypher, який за один транзакційний прохід графа паралельно ідентифікує прямі хімічні конфлікти, терапевтичні накладання груп АТС та приховані дублювання складів під різними комерційними брендами, а впроваджений контур штучного інтелекту на базі моделі Google Gemini забезпечує автоматичну трансформацію сухої латинської клінічної термінології у лаконічні та зрозумілі побутові застереження для пацієнта.

Перевірка розробленого програмного забезпечення, що завершують інженерну розробку, експериментально довели повну відповідність системи критеріям надійності, швидкодії та безпеки. Функціональні випробування інтерфейсу на реальних терапевтичних сценаріях поліпрагмазії підтвердили безпомилковість динамічного рендерингу графа зв'язків та коректність роботи алгоритмів лінгвістичного екранування, які надійно захищають веб-додаток від синтаксичних помилок при обробці специфічних символів і лапок. Автоматизоване модульне тестування серверного коду за допомогою фреймворку pytest та патернів інфраструктурної ізоляції MagicMock засвідчило стовідсоткове успішне проходження регресійних тестів, що дозволяє

рекомендувати створену інформаційну систему «PharmaGraph» для практичного використання як надійного асистента персональної медичної безпеки.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1 Державний реєстр лікарських засобів України : офіційний веб-ресурс / Міністерство охорони здоров'я України. URL: http://www.drlz.com.ua/ (дата звернення: 19.05.2026).

2 Про затвердження Положення про набори даних, які підлягають оприлюдненню у формі відкритих даних : Постанова Кабінету Міністрів України від 21.10.2015 р. № 835. URL: https://zakon.rada.gov.ua/laws/show/835-2015-%D0%BF (дата звернення: 19.05.2026).

3 ATC/DDD Index 2025 / WHO Collaborating Centre for Drug Statistics Methodology. Oslo : World Health Organization, 2025. URL: https://www.whocc.no/atc_ddd_index/ (дата звернення: 19.05.2026).

4 Medication Safety in Polypharmacy : Technical Report / World Health Organization. Geneva : WHO, 2019. URL: https://www.who.int/publications/i/item/WHO-UHC-SDS-2019.11 (дата звернення: 19.05.2026).

5 Мельникова Н. І. Особливості опрацювання медичної інформації для систем підтримки прийняття лікувальних рішень. Інформаційні системи та мережі : Вісник Національного університету «Львівська політехніка». 2015. № 832. С. 84–93. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2018/jun/12933/14-190-204.pdf (дата звернення: 19.05.2026).

6 Robinson I., Webber J., Eifrem E. Graph Databases. 2nd ed. Sebastopol : O'Reilly Media, 2015. 238 p. URL: https://neo4j.com/graph-databases-book/ (дата звернення: 19.05.2026).

7 Knowledge Graphs / A. Hogan та ін. ACM Computing Surveys. 2021. Vol. 54, No. 4. Article 71. DOI: https://doi.org/10.1145/3447772 (дата звернення: 19.05.2026).

8 Angles R., Gutierrez C. Survey of graph database models. ACM Computing Surveys. 2008. Vol. 40, No. 1. DOI: https://dl.acm.org/doi/10.1145/1322432.1322433 (дата звернення: 19.05.2026).

9 Neo4j Cypher Manual / Neo4j. URL: https://neo4j.com/docs/cypher-manual/current/ (дата звернення: 19.05.2026).

10 DrugBank 5.0: a major update to the DrugBank database for 2018 / D. S. Wishart та ін. Nucleic Acids Research. 2018. Vol. 46, Issue D1. P. D1074–D1082. DOI: https://doi.org/10.1093/nar/gkx1037 (дата звернення: 19.05.2026).

11 FastAPI Documentation / S. Ramírez. URL: https://fastapi.tiangolo.com/ (дата звернення: 19.05.2026).

12 ASGI Specification / Django Software Foundation. URL: https://asgi.readthedocs.io/en/latest/ (дата звернення: 19.05.2026).

13 Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 533 p. URL: https://sar.ac.id/stmik_ebook/prog_file_file/EFCofwzsj0.pdf (дата звернення: 19.05.2026).

14 PostgreSQL Documentation / PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/ (дата звернення: 19.05.2026).

15 SQLAlchemy 2.0 Documentation / M. Bayer. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 19.05.2026).

16 Bostock M., Ogievetsky V., Heer J. D3: Data-Driven Documents. IEEE Transactions on Visualization and Computer Graphics. 2011. Vol. 17, No. 12. P. 2301–2309. DOI: https://doi.org/10.1109/TVCG.2011.185 (дата звернення: 19.05.2026).

17 Fruchterman T. M. J., Reingold E. M. Graph Drawing by Force-Directed Placement. Software: Practice and Experience. 1991. Vol. 21, No. 11. P. 1129–1164. DOI: https://doi.org/10.1002/spe.4380211102 (дата звернення: 19.05.2026).

18 Attention Is All You Need / A. Vaswani та ін. Advances in Neural Information Processing Systems. 2017. Vol. 30. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 19.05.2026).

19 Gemini API Documentation / Google. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 19.05.2026).

20 Language Models are Few-Shot Learners / T. B. Brown та ін. Advances in Neural Information Processing Systems. 2020. Vol. 33. URL:

https://arxiv.org/abs/2005.14165 (дата звернення: 19.05.2026).

21 Medscape Reference: Drug Interaction Checker : official web-resource. URL: https://reference.medscape.com/drug-interactionchecker (дата звернення: 19.05.2026).

22 Drugs.com: Drug Interactions Checker : official web-resource. URL: https://www.drugs.com/drug_interactions.html (дата звернення: 19.05.2026).