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

ОСТАПЧУК ДЕНИС ОЛЕКСАНДРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
д-р техн. наук, професор
_____ Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ ТА БРОНЮВАННЯ ФАРМАЦЕВТИЧНИХ ПРЕПАРАТІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Ірина ФРИЗ, старший викладач кафедри
інформаційних технологій,
канд. фіз.-мат. наук
_____

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

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Остапчук Д. О. Інформаційна система пошуку та бронювання фармацевтичних препаратів. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У кваліфікаційній (бакалаврській) роботі спроєктовано та реалізовано інформаційну систему FaPharm. Серверна частина побудована на NestJS із PostgreSQL та пошуковим рушієм MeiliSearch, клієнтський SPA-додаток – на Vue 3. Розроблено алгоритм $N+1$ паралельних запитів для фасетної фільтрації. Тестування підтвердило час відповіді до 200 мс.

Ключові слова: інформаційна система, e-pharmacy, фасетна фільтрація, повнотекстовий пошук, NestJS, Vue.js, PostgreSQL.

68 сторінок, 17 рисунків, 5 таблиць, 1 додаток, 40 джерел.

ABSTRACT

Ostapchuk D. Information System for Searching and Booking Pharmaceutical Products. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia, 2026.

In the qualification (bachelor's) work, the FaPharm information system was designed and implemented. The server side is built on NestJS with PostgreSQL and MeiliSearch, the client SPA application on Vue 3. An $N+1$ parallel query algorithm for faceted filtering was developed. Performance testing confirmed response times under 200 ms.

Keywords: information system, e-pharmacy, faceted filtering, full-text search, NestJS, Vue.js, PostgreSQL.

68 pages, 17 figures, 5 tables, 1 appendix, 40 references.

ЗМІСТ

ВСТУП

Динамічний розвиток цифрових рішень та популяризація дистанційного обслуговування в охороні здоров’я вимагають створення новітніх інформаційних систем для фармацевтичного сектору. Зміна споживчих звичок у бік онлайн-закупівель медичної продукції зумовила стійке розширення ринку електронної фармації. Це створює високий попит на розробку надійних технологічних платформ, які забезпечують користувачам оперативний та безпечний доступ до даних про лікарські засоби.

Аналіз існуючих вітчизняних та зарубіжних платформ для пошуку й бронювання ліків свідчить про високий рівень цифровізації галузі, проте виявляє простір для подальшого вдосконалення архітектурних рішень. Зокрема, актуальними завданнями залишаються підвищення релевантності видачі при складних запитах, забезпечення миттєвого відгуку інтерфейсу в умовах паралельної обробки великої кількості фасетних фільтрів та впровадження інтелектуальних механізмів обробки помилок введення. Потреба у створенні систем, що поєднують високу швидкість обробки даних із максимальною зручністю навігації, залишається пріоритетним напрямком сучасної вебінженерії.

Метою роботи є проєктування та програмна реалізація інформаційної системи пошуку й бронювання фармацевтичних препаратів із застосуванням високопродуктивних алгоритмів фільтрації даних, яка забезпечуватиме швидкий та зручний доступ користувачів до актуальної інформації про медичні товари у вебсередовищі.

Зважаючи на поставлену мету, визначено такі *завдання дослідження*:

- проаналізувати сучасний стан ринку e-pharmacy в Україні та світі, дослідити бізнес-процеси пошуку й замовлення фармацевтичних препаратів в онлайн-середовищі;

- виявити переваги та недоліки існуючих інформаційних систем-аналогів і на їхній основі сформулювати функціональні вимоги до розроблюваної системи;
- обґрунтувати вибір технологічного стеку та інструментальних засобів для реалізації серверної й клієнтської частин системи;
- розробити архітектуру вебсистеми на основі принципу розподілу відповідальності, спроєктувати реляційну модель бази даних у PostgreSQL та побудувати відповідні UML-діаграми;
- розробити та дослідити алгоритм фасетної фільтрації для опрацювання великих масивів специфічних характеристик фармацевтичних препаратів;
- реалізувати серверну частину системи на основі фреймворку Nest.js з інтеграцією механізму повнотекстового пошуку та виправлення орфографічних помилок;
- створити клієнтський SPA-додаток на базі Vue.js з інтерактивним каталогом препаратів, системою фасетної фільтрації та модулем оформлення замовлень;
- провести тестування швидкодії пошукових запитів та стабільності системи, проаналізувати отримані результати.

Об'єктом дослідження є процеси пошуку, структурування та дистрибуції інформації про медичні товари у вебсередовищі.

Предметом дослідження є методи та технології розробки вебсистем із багаторівневою структурою даних і фасетним пошуком, зокрема алгоритми фільтрації великих масивів даних про фармацевтичні препарати та архітектурні рішення для забезпечення продуктивності й масштабованості таких систем.

Теоретичне значення роботи полягає в систематизації підходів до проєктування багаторівневих вебсистем із фасетним пошуком, а також у розробці та порівняльному аналізі алгоритмів фільтрації стосовно специфіки фармацевтичних даних. Отримані результати можуть бути використані у

подальших наукових дослідженнях стосовно проблематики архітектури інформаційних систем та методів повнотекстового пошуку.

Практичне значення роботи визначається тим, що розроблений веборієнтований SPA-додаток може бути впроваджений у діяльність аптечних мереж та фармацевтичних дистриб'юторів як прототип для автоматизації процесів пошуку й бронювання лікарських засобів. Реалізовані підходи до організації реляційної бази даних та механізми повнотекстового пошуку мають практичну цінність для розробки споріднених інформаційних систем в медичній і торговельній галузях.

Робота складається зі вступу, трьох розділів основної частини, висновків, списку використаних джерел із 40 найменувань та одного додатку. Перший розділ присвячено аналізу предметної області, другий – проєктуванню архітектури та алгоритмів, третій – програмній реалізації проєкту. Робота містить 17 рисунків та 5 таблиць. Загальний обсяг роботи – 68 сторінок.

РОЗДІЛ 1
АНАЛІЗ РИНКУ E-PHARMACY ТА ВИМОГ ДО СИСТЕМИ

1.1. Сучасний стан ринку e-pharmacy в Україні та світі

Фармацевтична галузь є одним із секторів економіки, де цифрова трансформація набула найбільш вираженого характеру протягом останнього десятиліття. Поява та розвиток концепції електронної аптеки стали закономірним наслідком конвергенції двох глобальних тенденцій: стрімкого зростання обсягів електронної комерції та підвищеної суспільної уваги до питань охорони здоров'я. Під терміном «e-pharmacy» розуміють систему дистанційного продажу, пошуку та бронювання лікарських засобів і медичних виробів через мережу Інтернет із залученням інформаційних платформ для автоматизованого управління асортиментом, замовленнями та взаємодією з кінцевим споживачем.

Глобальний ринок e-pharmacy демонструє стійку висхідну динаміку впродовж останніх років. За оцінками аналітичного агентства Grand View Research, у 2023 році його обсяг становив близько 69,8 млрд доларів США, а до 2030 року очікується зростання до 255,6 млрд доларів із середньорічним темпом приросту (CAGR) на рівні 20,4% [1]. Подібні прогнози підтверджують і інші дослідницькі компанії. Mordor Intelligence оцінює CAGR у 19,85% та досягнення позначки у 310.38 млрд доларів до 2031 року [2].

Географічно ринок e-pharmacy розподілений нерівномірно. Найбільші частки займають Північна Америка та Азіатсько-Тихоокеанський регіон. У США функціонують такі великі платформи, як Amazon Pharmacy, CVS Health та Walgreens, що пропонують повний цикл послуг від пошуку препарату до доставки додому з підтримкою рецептурних ліків [3]. У Китаї провідні позиції займають JD Health та Ali Health – підрозділи технологічних компаній JD.com та Alibaba, – де інтеграція з системами медичного страхування

дозволяє здійснювати відшкодування вартості ліків безпосередньо через мобільний застосунок. Європейський ринок відзначається строгішим регулюванням: Директива ЄС 2011/62/EU зобов'язує всі онлайн-аптеки проходити державну реєстрацію та використовувати захищену позначку безпеки на упаковках [4].

Згідно з даними, наведеними на рис. 1.1, Північна Америка та Азіатсько-Тихоокеанський регіон разом формують близько 70% глобального ринку, тоді як Європа поступається через жорсткіше регуляторне середовище. Країни, що розвиваються, зокрема Індія та Бразилія, стрімко нарощують свої частки завдяки масовому проникненню смартфонів та урядовим програмам цифровізації охорони здоров'я.

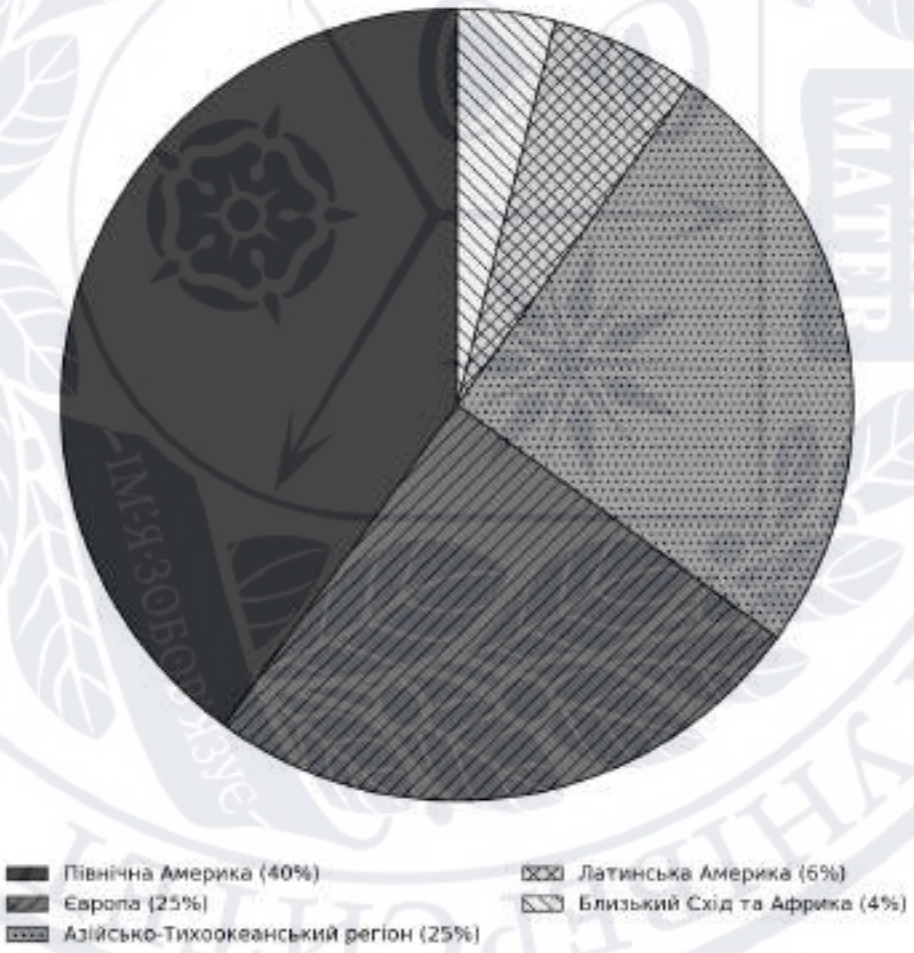

Рисунок 1.1 – Розподіл ринку e-pharmacy за регіонами станом на 2022 рік

Окрему увагу заслуговує аналіз ринку e-pharmacy в Україні, де галузь перебуває на стадії активного становлення. Відповідно до чинного

законодавства, повноцінна дистанційна торгівля рецептурними препаратами в Україні залишається обмеженою, однак ринок безрецептурних ліків та медичних виробів онлайн демонструє стійке зростання.

Серед ключових гравців вітчизняного ринку слід виділити інформаційну платформу-агрегатор Tabletki.ua, яка акумулює найбільшу частку пошукового трафіку в сегменті, а також провідні мережеві аптеки: «Аптека 9-1-1» (apteka911.ua), яка є найбільшою за кількістю замовлень платформою для пошуку та бронювання ліків в Україні; платформу «Подорожник» (podorozhnyk.ua), що інтегрує онлайн-бронювання з широкою офлайн-мережею аптечних пунктів; а також «АНЦ» (anc.ua) та інші регіональні платформи. Всі вони реалізують модель «click-and-collect» – онлайн-бронювання з подальшим самовивозом, яка є домінуючою в Україні через законодавчі обмеження на дистанційний продаж ліків [5].

Дані аналітичного сервісу Serpstat, зображені на рис. 1.2 показують, що ринок онлайн-замовлень фармацевтичних товарів в Україні характеризується помірною концентрацією та високим рівнем конкуренції. Безумовним лідером є платформа-агрегатор tabletki.ua, яка акумулює понад чверть користувацького трафіку – 28,1%. Проте решта гравців формують суттєву конкурентну противагу. До трійки лідерів також входять провідні аптечні мережі: apteka911.ua із часткою 14,7% та anc.ua, яка займає 13,8% ринку. Також помітні частки мають ресурси podorozhnyk.ua (7,8%) та add.ua (7,1%). Решта учасників, включаючи менші мережі та онлайн-аптеки, сумарно контролюють понад третину ринку. Це свідчить про те, що ринок є відкритим для нових технологічних рішень, здатних запропонувати споживачам якісніший користувацький досвід.

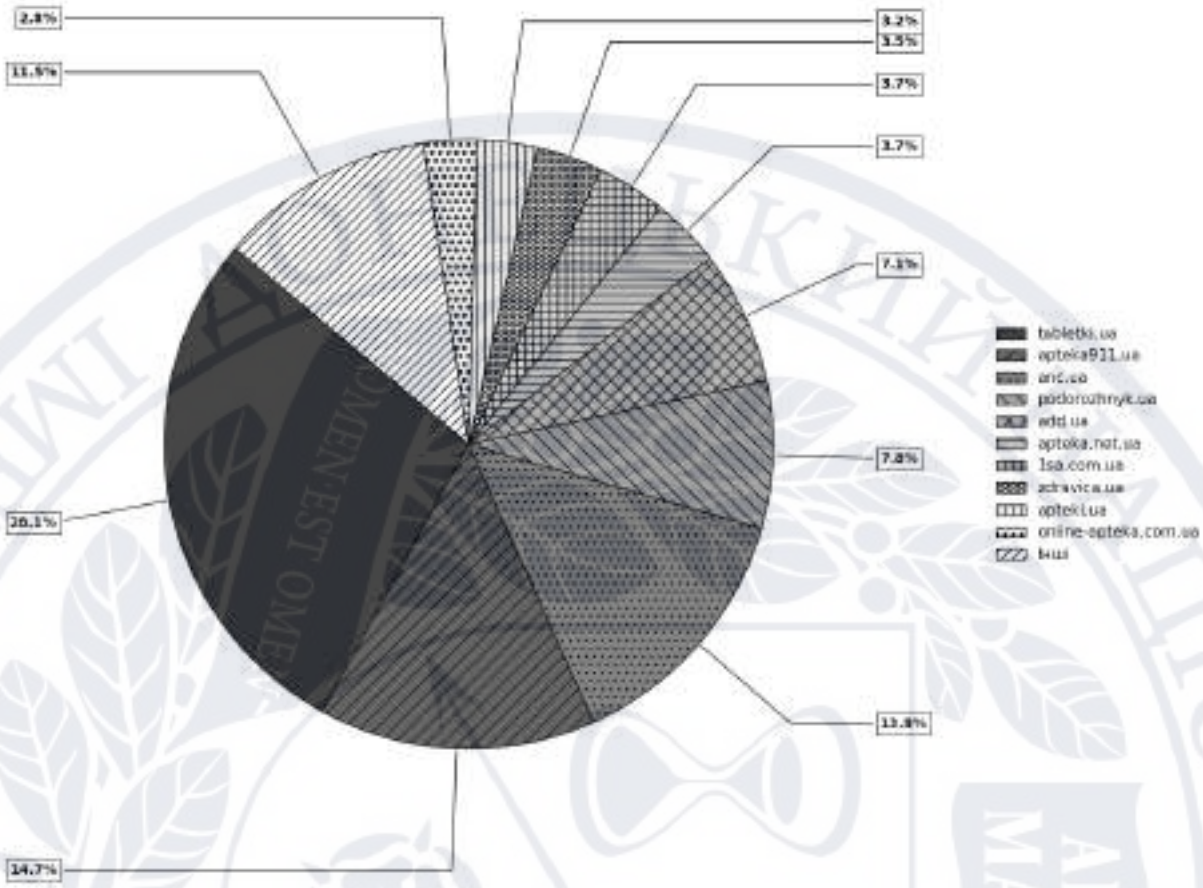

Рисунок 1.2 – Частки провідних платформ e-pharmacy в Україні за трафіком станом на 2026 рік

Важливим чинником трансформації ринку e-pharmacy є зміна поведінки споживачів. Все більше пацієнтів, які скористалися онлайн-аптеками, надають їм перевагу над традиційними для рутинних покупок безрецептурних препаратів. Ключовими мотиваторами є: зручність та швидкість пошуку потрібного препарату, можливість порівняння цін між аптеками, доступ до інформації про наявність ліків у режимі реального часу, а також збереження конфіденційності під час купівлі чутливих категорій товарів. Водночас споживачі вказують на ряд бар'єрів: незручний або неточний пошук, відсутність деталізованої фільтрації за специфічними характеристиками препаратів, а також загальна недовіра до якості ліків, придбаних онлайн.

Варто також зазначити вплив нормативно-правового середовища на розвиток ринку. В Україні регулювання онлайн-продажу лікарських засобів

здійснюється відповідно до Закону України «Про лікарські засоби» (№ 123/96-ВР) та постанов Кабінету Міністрів України. Чинне законодавство дозволяє дистанційне бронювання та замовлення безрецептурних препаратів із подальшою видачею в аптеці, що й обумовлює широке поширення моделі «click-and-collect» [6]. Очікується, що подальша лібералізація у напрямку повноцінної дистанційної доставки безрецептурних ліків суттєво збільшить місткість ринку.

Аналіз стану ринку e-pharmacy дозволяє зробити такі висновки: по-перше, галузь перебуває на етапі активного зростання як у глобальному, так і в українському вимірі; по-друге, ключовим технологічним викликом є забезпечення якісного та швидкого пошуку в умовах великих і постійно оновлюваних каталогів фармацевтичних товарів; по-третє, наявність незаповнених ніш у технологічних рішеннях для вітчизняного ринку та чітко виражені технологічні бар'єри з боку споживачів підтверджують доцільність розробки нової інформаційної системи з акцентом на продуктивності алгоритмів фільтрації та повнотекстового пошуку.

1.2. Дослідження бізнес-процесів пошуку та бронювання препаратів

Розробка ефективної інформаційної системи для фармацевтичної галузі неможлива без глибокого розуміння бізнес-процесів, що супроводжують пошук та бронювання лікарських засобів. Бізнес-процес у контексті e-pharmacy – це структурована послідовність дій, що виконуються учасниками системи з метою задоволення потреби у придбанні або резервуванні фармацевтичного препарату. Аналіз цих процесів є необхідною передумовою для формулювання функціональних вимог до розроблюваної системи та вибору відповідних алгоритмічних рішень [7].

У загальному вигляді взаємодія між учасниками ринку e-pharmacy охоплює три рівні: рівень споживача (кінцевий користувач, що здійснює пошук і замовлення), рівень платформи (інформаційна система, що

забезпечує обробку запитів, агрегацію даних та управління бронюванням) та рівень аптечної мережі (постачальник товарів, що підтримує актуальність залишків і виконує замовлення). Взаємодія між цими рівнями реалізується через набір взаємопов'язаних бізнес-процесів, ключовими серед яких є процес пошуку препарату та процес його бронювання [8].

Взаємодію між учасниками схематично зображено на рис. 1.3. Платформа відіграє роль центрального медіатора між попитом споживача та пропозицією аптечної мережі. Саме на рівні платформи зосереджені найбільш технологічно складні операції: обробка пошукових запитів, агрегація даних із множини джерел, фільтрація результатів за специфічними атрибутами та управління станом бронювань. Якість і швидкість виконання цих операцій безпосередньо визначають рівень задоволеності кінцевого користувача.

Рисунок 1.3 – Загальна схема взаємодії учасників ринку e-pharmacy

Розглянемо детально процес пошуку фармацевтичного препарату. Даний процес ініціюється споживачем, який формує пошуковий запит – як правило, це назва препарату (торгова або міжнародна непатентована назва), діюча речовина або симптом захворювання. Особливістю фармацевтичного пошуку порівняно зі звичайним товарним пошуком є висока варіативність запитів: споживачі часто допускають орфографічні помилки у медичних термінах, використовують скорочення, синоніми або народні назви препаратів [9]. Наприклад, запит «но-шпа» є торговою назвою, тоді як діюча речовина –

«дротаверин»; платформа має повертати релевантні результати за обома варіантами.

Процес пошуку в системах e-pharmacy, як правило, реалізується у кілька етапів. На першому етапі відбувається попередня обробка запиту: нормалізація регістру, видалення зайвих символів, стемінг або лематизація для врахування відмінювання слів. На другому етапі виконується повнотекстовий пошук по індексованій базі препаратів із ранжуванням результатів за релевантністю. На третьому етапі застосовуються механізми виправлення помилок для обробки запитів з друкарськими помилками. На четвертому етапі результати фільтруються відповідно до заданих користувачем параметрів [10].

Блок-схема на рис. 1.4 демонструє, що процес пошуку є ітеративним: у разі відсутності результатів після основного пошуку система автоматично вдається до механізму нечіткого пошуку. Ця особливість є критично важливою для фармацевтичної предметної області, де некоректне написання назви препарату є поширеною ситуацією, а відсутність результату може мати негативні наслідки для здоров’я користувача, якщо він не знайде потрібний засіб вчасно.

Рисунок 1.4 – Блок-схема процесу пошуку препарату в системі e-pharmacy

Не менш важливим є процес фасетної фільтрації – механізму, що дозволяє користувачу послідовно звужувати множину результатів пошуку шляхом застосування фільтрів за різними атрибутами препарату [11]. У

фармацевтичній галузі типовими фасетами є: форма випуску, виробник або країна виробництва, діапазон ціни, наявність у конкретній аптеці, дозування діючої речовини, а також ознака рецептурного або безрецептурного відпуску.

Ключова технічна складність фасетної фільтрації полягає в необхідності одночасного підрахунку кількості товарів, що відповідають кожній комбінації фасетів без значної затримки відповіді при великому обсязі каталогу.

Наведені у таблиці 1.1 дані свідчать про те, що найбільш технічно складними є фасети «Виробник» та «Наявність в аптеці». Перший потребує ефективної реалізації пошуку по великому словнику значень, другий – забезпечення актуальності даних про залишки в режимі, наближеному до реального часу, що вимагає налаштування механізмів синхронізації між платформою та аптечними системами управління запасами.

Таблиця 1.1 Типові категорії фільтрації у системах e-pharmacy

Категорія фільтрації	Об’єктний склад та типи значень	Метод обробки
Лікарська форма	Класифікатор за типами: тверді (табл., капс.), рідкі, м’які тощо	Реалізація через Multi-select фільтрацію
Виробник	Реєстр торгових назв та корпорацій (необхідна підтримка автокомпліту)	Індексація великих текстових масивів, ElasticSearch
Країна реєстрації	Перелік країн згідно з державним реєстром ЛЗ	Статичний Dropdown-список
Дозування	Маса діючої речовини (мг/мл, % або одиниці в упаковці)	Парсинг рядкових значень у числові для фільтрації
Ціновий діапазон	Поточна вартість у гривневому еквіваленті	Range-запит до БД (min/max значення)
Умови відпуску	Бінарна ознака: за рецептом / без рецепта	Boolean-фільтр (так/ні)
Локальна наявність	Динамічний статус залишків у конкретних аптечних пунктах	Real-time синхронізація через API з ERP аптек

Процес бронювання препарату є логічним продовженням процесу пошуку і розпочинається з моменту, коли користувач обирає конкретний товар та аптеку для самовивозу. Відповідно до моделі «click-and-collect», яка

домінує на українському ринку, бронювання передбачає тимчасове резервування одиниці товару на складі обраної аптеки на визначений термін – як правило, від 2 до 24 годин. Протягом цього часу товар вилучається з доступного залишку для інших покупців, а користувач отримує підтвердження із зазначенням адреси аптеки, терміну резервування та контактної інформації.

Діаграма станів на рис. 1.5 відображає повний життєвий цикл замовлення в системі e-pharmacy. Особливої уваги заслуговує стан «Протерміновано»: автоматичне скасування бронювань після закінчення терміну резервування є критично важливою функцією, оскільки забезпечує коректність даних про наявність товарів і запобігає ситуації, коли реально доступний препарат відображається як заброньований. Реалізація цього механізму потребує наявності фонового процесу на рівні серверної частини системи.

З точки зору нефункціональних вимог, бізнес-процеси e-pharmacy висувають жорсткі вимоги до продуктивності інформаційної системи. Дослідження у галузі UX та e-commerce свідчать, що затримка відповіді понад 200 мс при виконанні пошукового запиту призводить до суттєвого зростання показника відмов, а збільшення часу завантаження сторінки на кожні 100 мс знижує конверсію на 1–2% [12]. У фармацевтичному контексті ця проблема є ще гострішою: користувач, який шукає ліки для хворої людини, особливо чутливий до будь-яких затримок і незручностей інтерфейсу.

Рисунок 1.5 – Діаграма станів замовлення у системі e-pharmacy

Наведені в таблиці 1.2 показники визначають технічні орієнтири для розроблюваної системи і безпосередньо впливають на вибір архітектурних рішень та алгоритмів обробки даних. Вимога щодо часу відповіді на пошуковий запит не більше 200 мс обумовлює необхідність використання індексованого повнотекстового пошуку замість стандартних SQL-запитів типу LIKE.

Окремо слід розглянути особливості бізнес-процесів із точки зору ролей користувачів системи. У типовій платформі e-pharmacy виділяють щонайменше три ролі: анонімний відвідувач (може здійснювати пошук і

перегляд каталогу без реєстрації), зареєстрований користувач (може оформлювати замовлення, зберігати список препаратів та переглядати історію замовлень) та адміністратор аптечної мережі (управляє каталогом, оновлює залишки та підтверджує замовлення). Чітке розмежування ролей є необхідною умовою для проєктування системи розмежування прав доступу та побудови відповідної моделі бази даних.

Таблиця 1.2 Нефункціональні вимоги до продуктивності системи e-pharmacy

Показник продуктивності	Цільове значення	Обґрунтування
Швидкість пошуку	≤ 200 мс	Стандарти UX (модель Google RAIL)
Швидкість завантаження	LCP ≤ 1.5 с	Показники Core Web Vitals
Час обробки транзакції бронювання	≤ 500 мс	Бізнес-логіка «click-and-collect»
Коефіцієнт відмовостійкості	Uptime ≥ 99.5%	Умови SLA аптечних мереж
Інтервал оновлення даних про залишки	≤ 5 хв	Вимоги до актуальності складських запасів

Дослідження бізнес-процесів пошуку та бронювання препаратів дозволяє виокремити ключові функціональні та нефункціональні вимоги до розроблюваної системи. Функціонально система має забезпечувати повнотекстовий пошук із підтримкою нечіткого співставлення, багатокритеріальну фасетну фільтрацію з динамічним підрахунком лічильників, управління життєвим циклом замовлення від бронювання до виконання або скасування, а також рольову модель доступу. З нефункціональної точки зору, критично важливими є забезпечення часу відповіді на пошукові запити в межах 200 мс та підтримка пікових навантажень без значного погіршення користувацького досвіду. Ці вимоги безпосередньо визначатимуть архітектурні та алгоритмічні рішення, що розглядаються в наступних розділах.

1.3. Порівняльний аналіз існуючих інформаційних систем-аналогів

Аналіз існуючих інформаційних систем-аналогів є обов'язковим етапом проєктування нового програмного продукту, оскільки дозволяє виявити усталені підходи до вирішення задачі, визначити сильні та слабкі сторони наявних рішень, а також сформулювати конкурентні переваги розроблюваної системи. Для порівняльного аналізу було обрано шість платформ: три українські – tabletki.ua, apteka911.ua та podorozhnyk.ua – та три зарубіжні – DocMorris (Німеччина/Нідерланди), PharmEasy (Індія) та Amazon Pharmacy (США). Вибір саме цих систем обумовлений їхньою ринковою значущістю, технологічною зрілістю та репрезентативністю різних підходів до реалізації функцій пошуку і фільтрації.

Насамперед розглянемо платформу tabletki.ua, яка на сьогодні є домінуючим гравцем українського ринку e-pharmacy за кількістю унікальних відвідувачів та охопленням аптечних мереж. За даними SimilarWeb, місячна аудиторія platforma tabletki.ua у 2024 році перевищує 20 млн унікальних відвідувачів, що більш ніж удвічі перевищує показники найближчого конкурента [13]. Платформа агрегує дані про наявність препаратів із понад 10 000 аптек по всій Україні, що робить її найбільшою базою фармацевтичних даних у країні.

З функціональної точки зору tabletki.ua реалізує широкий набір можливостей, зображених на рис. 1.6: повнотекстовий пошук із підтримкою торгових і непатентованих назв, базову фасетну фільтрацію за формою випуску та виробником, геолокаційний пошук найближчих аптек із наявністю обраного препарату, порівняння цін між аптечними мережами та функцію бронювання за моделлю «click-and-collect». Окремої уваги заслуговує реалізація пошуку по активній речовині: система дозволяє знаходити всі аналоги препарату за діючою речовиною, що є важливою функцією для споживачів, які шукають бюджетніший замінник [14].

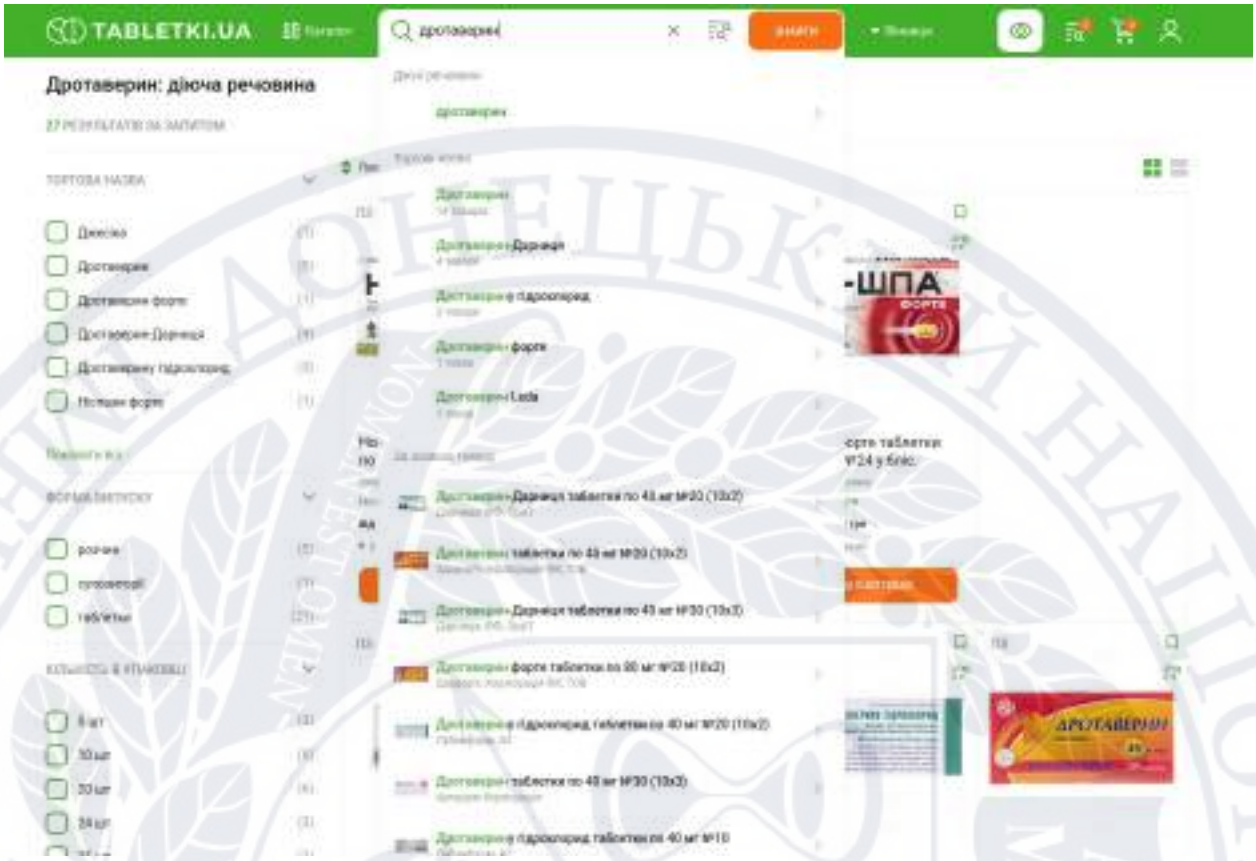

Рисунок 1.6 – Інтерфейс пошуку платформи tabletki.ua

Незважаючи на лідерські позиції, платформа tabletki.ua має певні технологічні недоліки, що були виявлені в ході тестування. По-перше, система фасетної фільтрації є обмеженою: відсутній розвинений механізм фільтрації за числовими діапазонами. По-друге, механізм виправлення орфографічних помилок є недостатньо розвиненим: запити з типовими помилками у медичних термінах не завжди повертають релевантні результати. По-третє, суттєвим недоліком з точки зору користувацького досвіду є надмірно агресивна політика безпеки, реалізована через сервіс Cloudflare. Хоча такий захист є необхідним для запобігання DDoS-атакам та несанкціонованому парсингу даних, він часто створює додаткові бар’єри для легітимних користувачів. Необхідність частого проходження перевірок або тривале очікування на етапі перевірки браузера сповільнює шлях клієнта до покупки.

Платформа apteka911.ua є одним із ключових гравців ринку, спираючись на потужну логістичну базу власної мережі, що входить до

трійки найбільших в Україні. Це забезпечує сервісу суттєву перевагу у частині оперативного оновлення даних про залишки та широкої географії видачі замовлень. З технологічної точки зору apteka911.ua пропонує динамічне автодоповнення у рядку пошуку, наявні інформативні картки товарів із відеооглядами та повними інструкціями, а також функціональний особистий кабінет із програмою лояльності. Водночас система пошуку має недоліки: відсутня повноцінна підтримка семантичного пошуку за симптомами (наприклад, «від кашлю»), а інструменти фільтрації є менш деталізованими порівняно із провідними зарубіжними аналогами [15].

Платформа podorozhnyk.ua є цифровим фронт-офісом найбільшої за кількістю точок аптечної мережі України – «Подорожник». На відміну від універсальних агрегаторів, цей ресурс орієнтований на прямі продажі власної мережі, що гарантує найвищу достовірність даних про наявність товарів завдяки прямій інтеграції з внутрішньою ERP-системою. Ключовою перевагою є розвинена система онлайн-резервування та можливість швидкого самовивозу. Проте, з точки зору функціональності пошуку, платформа поступається спеціалізованим агрегаторам: спостерігається обмежений набір фасетних фільтрів, що ускладнює підбір аналогів за діючою речовиною без точного знання назви бренду [16].

Серед зарубіжних аналогів особливого розгляду заслуговує платформа DocMorris – один із найбільших онлайн-фармацевтів Європи зі штаб-квартирою в Нідерландах, що є ключовим гравцем на ринку Німеччини. Після повноцінного впровадження електронних рецептів (e-Rezept) у Німеччині, DocMorris реалізує складну технологічну модель верифікації рецептурних препаратів через NFC-зчитування медичних карток. З точки зору пошуку, система вирізняється інтелектуальною фасетною фільтрацією, що дозволяє сортувати товари за формою випуску, дозуванням та розміром упаковки одночасно. Також реалізовано поглиблений пошук за симптомами

та взаємодією ліків, що перетворює платформу на повноцінний консультаційний сервіс [17].

Індійська платформа PharmEasy є прикладом високонавантаженої екосистеми для ринку з екстремальним трафіком. Технологічно PharmEasy базується на мікросервісній архітектурі, що дозволяє інтегрувати не лише продаж ліків, а й онлайн-консультації лікарів та запис на діагностичні тести. Пошуковий рушій платформи оптимізований під мультимовні запити та використовує розширені алгоритми персоналізованих рекомендацій для пропонування дешевших аналогів-дженериків, що є критично важливим для локального ринку [18].

Amazon Pharmacy, побудована на базі придбаного стартапу PillPack, є найбільш інфраструктурно досконалою платформою. Її ключова особливість – повна інтеграція з екосистемою AWS та підпискою Amazon Prime, що забезпечує безшовний досвід оплати та логістики. Використання машинного навчання дозволяє системі прогнозувати час наступного замовлення для пацієнтів із хронічними захворюваннями та автоматично перевіряти сумісність нових ліків із тими, що користувач замовляв раніше. Проте через специфічну систему медичного страхування США та складні моделі ціноутворення, пряме копіювання бізнес-логіки Amazon Pharmacy для українського ринку наразі є недоцільним [19].

1.4. Технологічні засоби розробки інформаційної системи

Вибір технологічного стеку є одним із ключових архітектурних рішень, що визначають довгострокову підтримуваність, масштабованість та продуктивність інформаційної системи. Обґрунтований вибір технологій здійснюється на основі сукупності критеріїв: відповідності функціональним вимогам системи, зрілості та активності спільноти підтримки, продуктивності під очікуваним навантаженням, а також можливості інтеграції окремих компонентів у єдину архітектуру. Розроблювана система

побудована за класичною клієнт-серверною архітектурою і розділена на два незалежних компоненти: серверну частину та клієнтський застосунок.

Система складається з чотирьох основних компонентів (рис. 1.7): клієнтського SPA-застосунку, серверного REST API, реляційної бази даних та пошукового рушія. Така архітектура забезпечує чітке розмежування відповідальності між компонентами і дозволяє масштабувати кожен із них незалежно.

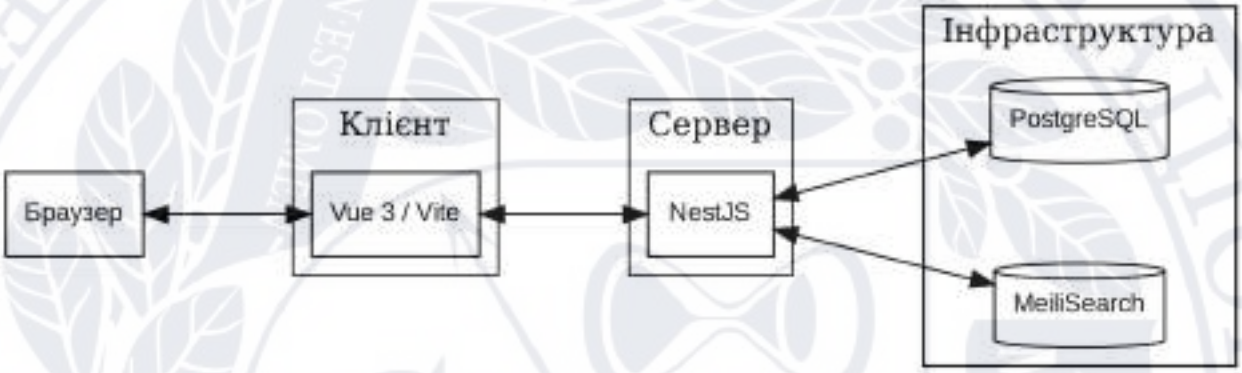

Рисунок 1.7 – Схема взаємодії компонентів системи FaPharm

Для реалізації серверної частини системи обрано середовище виконання Node.js у поєднанні з фреймворком NestJS. Node.js є платформою на основі рушія V8, що реалізує подієво-орієнтовану асинхронну модель виконання коду. Ця модель є особливо доречною для систем із великою кількістю одночасних запитів на читання даних – саме такий профіль навантаження характерний для пошукових платформ e-pharmacy [20].

NestJS є прогресивним Node.js-фреймворком, побудованим на основі TypeScript та принципах, запозичених з Angular: модульна архітектура та декораторний стиль опису компонентів. Ці властивості забезпечують високу структурованість кодової бази, що є критично важливим для великих проєктів із тривалим циклом підтримки [21].

Мовою розробки для обох частин системи обрано TypeScript – статично типізований транспілятор JavaScript. Використання TypeScript дозволяє виявляти значну частину помилок на етапі компіляції, забезпечує

автодоповнення в IDE та суттєво покращує читабельність і підтримуваність коду [22].

Для зберігання основних даних системи обрано реляційну СУБД PostgreSQL. Вона є однією з найбільш функціонально повних відкритих реляційних баз даних, включаючи підтримку складних типів даних (JSON, масиви, перелічувані типи), розвинену систему індексування (B-tree, GIN, GiST, Hash) та механізми транзакцій відповідно до стандарту ACID [23]. Для фармацевтичної системи, де критично важливою є цілісність даних про замовлення та залишки, реляційна модель з підтримкою транзакцій є природним вибором порівняно з документоорієнтованими СУБД. Наприклад, MongoDB, незважаючи на переваги при роботі з документами, поступається PostgreSQL у частині цілісності даних при виконанні складних транзакцій, що є критичною вимогою для модуля бронювання.

Для взаємодії серверної частини з PostgreSQL планується використання ORM-бібліотеки TypeORM. TypeORM є зрілим рішенням для TypeScript-проєктів, що надає декларативний опис сутностей через декоратори, автоматичну генерацію SQL-запитів та інструменти версіонування схеми бази даних через міграції [24]. Механізм міграцій є особливо важливим для виробничих систем: він дозволяє вносити зміни до схеми бази даних у контрольований та відтворюваний спосіб без ризику втрати даних.

Ключовим технологічним рішенням, що безпосередньо визначає продуктивність пошукової підсистеми, є вибір спеціалізованого пошукового рушія. Аналіз вимог, сформульованих у першому розділі, показав, що реалізація повнотекстового пошуку засобами стандартних SQL-запитів не здатна забезпечити необхідний рівень продуктивності та функціональності при великих обсягах каталогу.

Для реалізації пошукової підсистеми обрано MeiliSearch – сучасний відкритий пошуковий рушій, написаний мовою Rust. MeiliSearch орієнтований на забезпечення результатів пошуку в режимі реального часу з

часом відповіді менше 50 мс навіть на великих наборах даних [25]. Серед ключових можливостей MeiliSearch слід виділити такі переваги:

- вбудовану підтримку нечіткого пошуку з налаштовуваною допустимою кількістю помилок;
- фасетну фільтрацію з автоматичним підрахунком лічильників;
- підтримку кастомного ранжування результатів;
- локалізований пошук із підтримкою кирилиці;

Ключовою технологічною перевагою MeiliSearch перед традиційними рішеннями є його автономність, оскільки рушій поставляється у вигляді єдиного статично скомпільованого бінарного файлу. Така структура повністю виключає необхідність використання зовнішніх залежностей або середовищ виконання, як-от віртуальної машини Java, що критично важливо для оптимізації серверних ресурсів.

Порівняно з Elasticsearch, який вимагає складного налаштування кластерів та значних обчислювальних витрат на підтримку життєдіяльності Java-середовища, MeiliSearch пропонує вищу швидкість імплементації та легкість горизонтального масштабування [26]. Даний рушій постає як найбільш раціональне рішення для розробки пошукових сервісів середнього та великого масштабу, де швидкість розробки та низька вартість володіння інфраструктурою є пріоритетними факторами.

Для розробки клієнтського застосунку обрано фреймворк Vue 3 із Composition API. Vue 3 є прогресивним JavaScript-фреймворком для побудови реактивних користувацьких інтерфейсів, що відрізняється низьким порогом входу, гнучкою компонентною моделлю та високою продуктивністю завдяки оптимізованому рушію реактивності на основі Proxy [27]. Composition API, введений у Vue 3, дозволяє організовувати логіку компонентів за функціональними ознаками, що суттєво покращує повторне використання коду та тестованість.

Для збірки та запуску dev-середовища буде використовуватись Vite – сучасний інструмент збірки, що використовує нативні ES-модулі браузера під час розробки, забезпечуючи миттєвий старт сервера та швидке оновлення модулів без повного перезавантаження [28]. Управління глобальним станом у клієнтському застосунку планується за допомогою Pinia, що є офіційно рекомендованим сховищем стану для Vue 3 [29].

Для розгортання інфраструктури системи буде використовуватись контейнеризація на основі Docker та Docker Compose. Docker Compose дозволяє описати всю інфраструктуру проєкту в єдиному конфігураційному файлі та одною командою запустити PostgreSQL та MeiliSearch [30]. Обидва сервіси з'єднані у спільній bridge-мережі та зберігають дані у томах, що гарантує збереження даних між перезапусками контейнерів. Контейнеризація забезпечує відтворюваність середовища розробки та спрощує процес розгортання на виробничому сервері.

У першому розділі здійснено комплексний аналіз предметної області, необхідний для проєктування інформаційної системи пошуку та бронювання фармацевтичних препаратів.

Аналіз сучасного стану ринку e-pharmacy підтвердив стійку висхідну динаміку галузі як у глобальному, так і в українському вимірі. Глобальний ринок демонструє середньорічний темп зростання на рівні 20,4%, а український сегмент, попри законодавчі обмеження на дистанційний продаж рецептурних препаратів, активно розвивається в межах моделі «click-and-collect». Ключовим технологічним викликом галузі визначено забезпечення якісного та швидкого пошуку в умовах великих і постійно оновлюваних каталогів фармацевтичних товарів.

Дослідження бізнес-процесів пошуку та бронювання препаратів дозволило встановити конкретні нефункціональні вимоги до розроблюваної системи: час відповіді на пошуковий запит не повинен перевищувати 200 мс, система має підтримувати повнотекстовий пошук із корекцією

орфографічних помилок, багатокритеріальну фасетну фільтрацію з динамічним підрахунком лічильників, рольову модель доступу та повний життєвий цикл замовлення від бронювання до виконання або автоматичного скасування.

Порівняльний аналіз платформ-аналогів виявив технологічний розрив між локальними агрегаторами та провідними світовими платформами. Українські сервіси демонструють високе покриття ринку, проте поступаються у частині глибини фасетної фільтрації, інтелектуальної обробки запитів та механізмів виправлення орфографічних помилок. Виявлені недоліки існуючих рішень обґрунтовують конкурентні переваги розроблюваної системи та підтверджують доцільність її створення.

Обґрунтовано вибір технологічного стеку системи FaPharm: для серверної частини обрано NestJS на основі Node.js із мовою TypeScript, що забезпечує структурованість коду та асинхронну модель обробки запитів; для зберігання даних – реляційну СУБД PostgreSQL з підтримкою ACID-транзакцій, що є критично важливим для модуля бронювання; для реалізації пошукової підсистеми – рушій MeiliSearch, здатний забезпечити час відповіді менше 50 мс із вбудованою підтримкою нечіткого пошуку та фасетної фільтрації; для клієнтської частини – фреймворк Vue 3 із Composition API та інструментом збірки Vite. Інфраструктура системи контейнеризована засобами Docker Compose, що гарантує відтворюваність середовища розробки та спрощує розгортання.

Отримані результати аналізу є підґрунтям для проєктування архітектури системи та алгоритмів фільтрації, що розглядаються у наступному розділі.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА АЛГОРИТМІВ

2.1. Розробка архітектури вебсистеми

Архітектура інформаційної системи є фундаментальним проєктним рішенням, що визначає структуру компонентів, механізми їхньої взаємодії та принципи організації коду на всіх рівнях системи. Для розроблюваної системи FaPharm обрано клієнт-серверну архітектуру з чітким розмежуванням відповідальностей між серверною та клієнтською частинами відповідно до принципу Separation of Concerns [31]. Взаємодія між рівнями здійснюється виключно через HTTP-запити у форматі JSON, що забезпечує незалежність розробки та розгортання кожної частини системи.

Система організована у монорепозиторій із двома кореневими директоріями:

- backend – серверна частина на NestJS;
- frontend – клієнтський SPA на Vue 3;

Така організація спрощує управління залежностями та забезпечує єдність версіонування коду.

Серверна частина системи FaPharm буде побудована за принципом feature-based модульності фреймворку NestJS: кожна доменна область є окремим самодостатнім модулем із власними контролерами, сервісами, сутностями бази даних та об’єктами передачі даних. Така організація відповідає принципу єдиної відповідальності та суттєво спрощує масштабування і тестування системи [32].

Проєкт складається з таких функціональних модулів:

- auth – автентифікація та авторизація;
- user – управління обліковими записами та профілями користувачів;
- categories – ієрархічний каталог категорій препаратів;

- katottg – адміністративно-територіальні одиниці України для геолокаційного пошуку [33];
- pharmacy – управління аптечними мережами та пунктами;
- products – каталог препаратів із атрибутами, групами, залишками та інструкціями;
- commerce – кошик та замовлення;
- search – інтеграція з MeiliSearch;
- settings – налаштування сайту та керування кешем.
- common – спільна інфраструктура

Наведена на рис. 2.1 схема ілюструє ієрархічну архітектуру серверної частини: кореневий AppModule імпортує всі функціональні модулі, кожен із яких є замкненою одиницею функціональності. Механізм впровадження залежностей в NestJS забезпечує слабке зв’язування між модулями: сервіси одного модуля можуть використовувати сервіси іншого лише через явний експорт, що унеможливлює приховані залежності.

Проєктування клієнтського інтерфейсу системи FaPharm базується на архітектурному патерні односторінкового застосунку (SPA), що дозволяє забезпечити високу динаміку користувацького досвіду без надмірних запитів на сервер для перезавантаження сторінок. Логічна структура застосунку передбачає декомпозицію на чотири функціональні рівні, що мінімізує зв’язність модулів:

- рівень маршрутизації – відповідає за визначення логічних шляхів та управління станами переходів;
- рівень управління станом – централізоване сховище, що реалізує принцип «єдиного джерела істини»;
- рівень представлення – ієрархія компонентів та екранних форм;
- рівень сервісної взаємодії – абстракція для виконання HTTP-запитів.

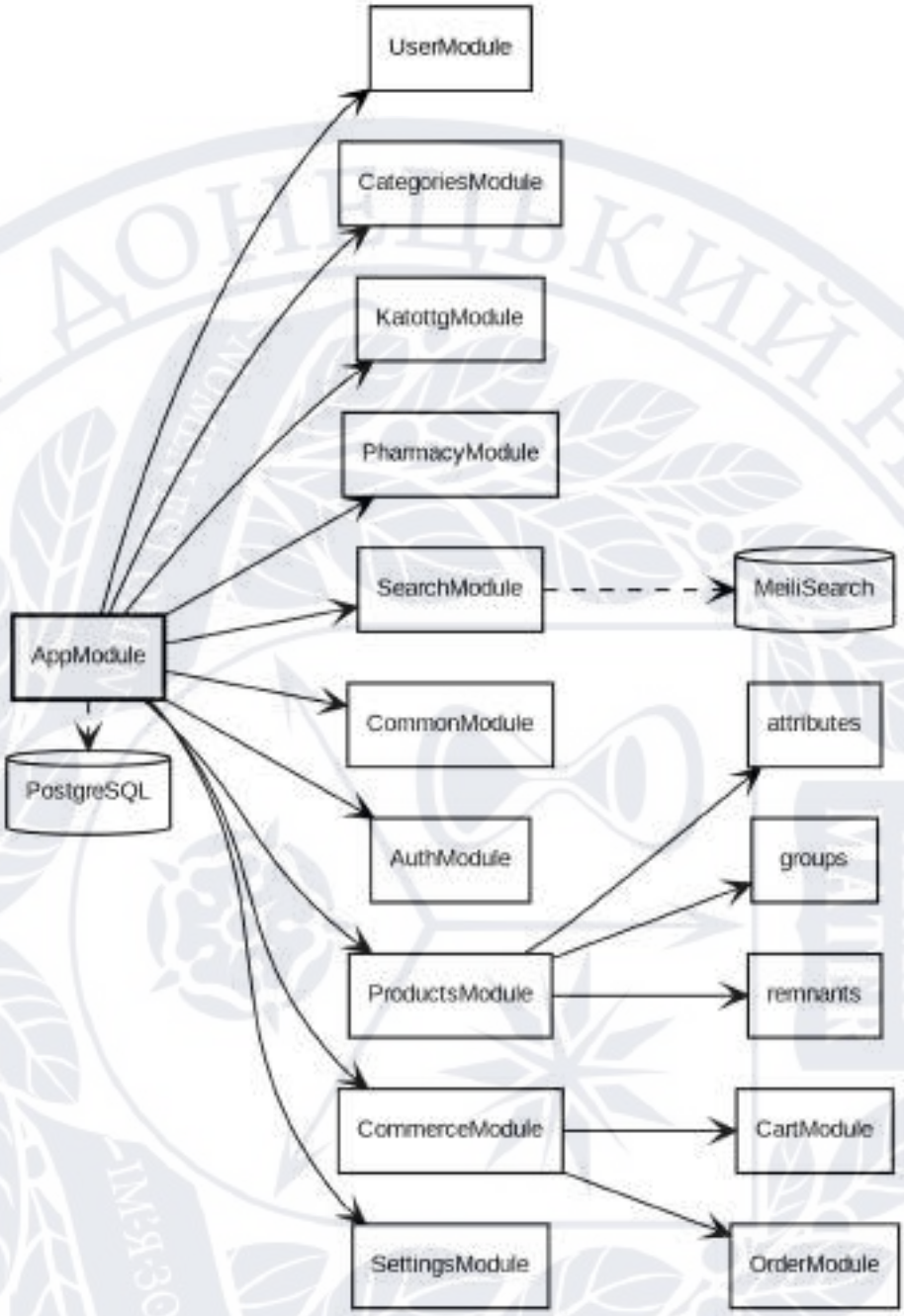

Рисунок 2.1 – Модульна структура серверної частини системи

Архітектура маршрутизації має деревоподібну структуру з центральним вузлом на головній сторінці (рис. 2.2). Проєктне рішення передбачає використання 12 основних маршрутів, що охоплюють усі етапи клієнтського шляху: від пошуку в загальному каталозі та ієрархічних категоріях до сторінки конкретного товару.

Паралельно з навігаційною моделлю спроєктовано систему управління глобальним станом, що складається з чотирьох доменних сховищ: auth, cart,

categories та city. Такий підхід до проєктування забезпечує високу масштабованість клієнтської частини та дозволяє незалежно розвивати логіку представлення та бізнес-логіку управління даними.

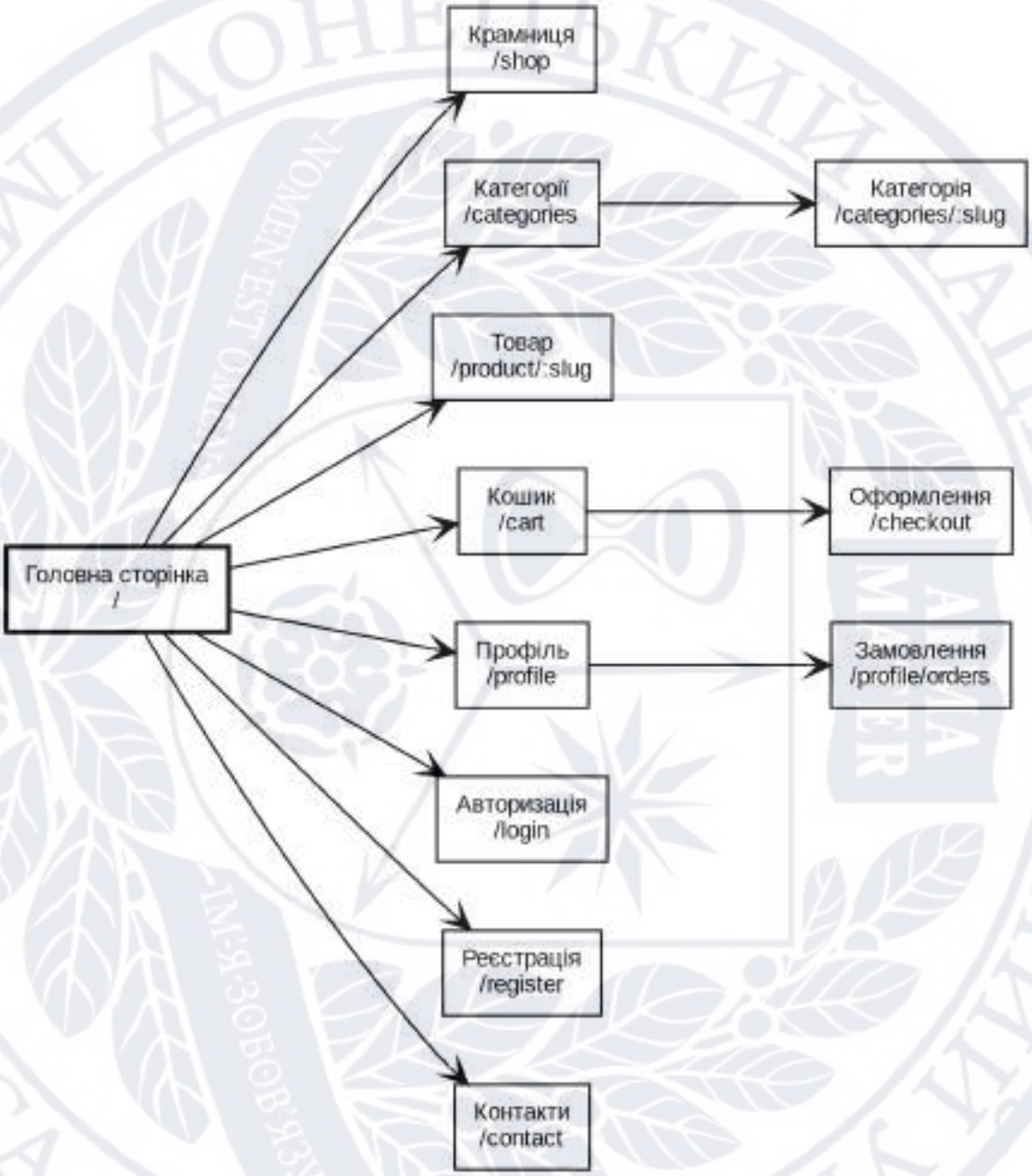

Рисунок 2.2 – Граф маршрутизації клієнтського застосунку FaPharm

Пошукова підсистема є технологічно найскладнішим компонентом системи, оскільки має забезпечувати одночасну роботу повнотекстового пошуку, фасетної фільтрації та геолокаційного звуження результатів у межах одного запиту. Для досягнення цих цілей застосовано денормалізований підхід до індексування: індекс MeiliSearch містить попередньо агреговані дані

про залишки препаратів у розрізі міст, що усуває необхідність JOIN-запитів до PostgreSQL під час пошуку.

Запропонована архітектура системи FaPharm проєктується як цілісне багаторівневе рішення, в основі якого лежить принцип делегування відповідальності між спеціалізованими компонентами. Такий підхід до організації взаємодії компонентів, заснований на принципах слабкого зв’язування та декомпозиції функцій, має на меті створити надійний фундамент для забезпечення масштабованості та спрощення подальшої підтримки системи на всіх етапах її життєвого циклу.

2.2. Проєктування реляційної моделі бази даних

Проєктування бази даних є ключовим етапом розробки інформаційної системи, що передує програмній реалізації та визначає структуру зберігання, цілісність даних і продуктивність запитів. Відповідно до функціональних вимог, сформульованих у першому розділі, предметна область системи FaPharm охоплює п’ять основних доменів: користувачі та автентифікація, геолокація та адміністративно-територіальний устрій, фармацевтичний каталог, управління замовленнями та пошуковий індекс. Для кожного домену необхідно обрати адекватну модель зберігання даних з урахуванням специфіки запитів, що виконуватимуться.

На етапі проєктування були визначені ключові вимоги до моделі даних: підтримка ACID-транзакцій для операцій бронювання та замовлення; можливість зберігання мультимовного контенту без надмірної кількості таблиць; ефективна підтримка ієрархічних структур; висока продуктивність читання для пошукових запитів із фільтрацією за множинними атрибутами; а також гнучкість схеми для атрибутів, що різняться залежно від категорії препарату. Сукупність цих вимог визначила вибір гібридної моделі: реляційна нормалізація для сутностей із чіткими зв’язками та денормалізовані структури для гнучких і мультимовних даних [34].

Першим кроком проєктування є виділення основних сутностей предметної області та їхніх атрибутів. На основі аналізу бізнес-процесів, були визначені такі ключові сутності:

- користувач;
- адміністративно-територіальна одиниця (КАТОТТГ);
- аптека;
- препарат;
- залишок;
- категорія;
- атрибут препарату;
- кошик;
- замовлення.

На рис. 2.3 зображені типи зв’язків між ними.

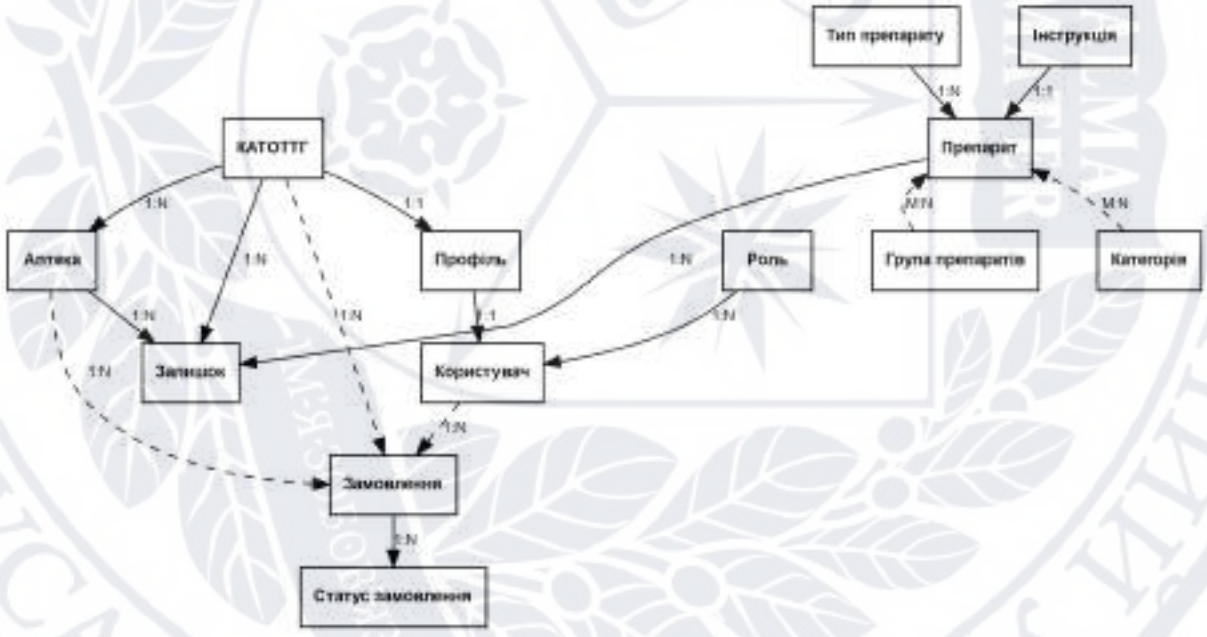

Рисунок 2.3 – Концептуальна ER-діаграма системи FaPharm

Важливим проєктним рішенням є вибір типів даних для полів із нетривіальною семантикою. Для мультимовного контенту розглядалися два підходи: окремі таблиці перекладів із зовнішніми ключами та зберігання у форматі JSONB. Перший підхід забезпечує повну нормалізацію, однак вимагає JOIN-запитів при кожному читанні та ускладнює схему додатковими

таблицями для кожної сутності. Другий підхід – JSONB із фіксованою структурою дозволяє отримувати мультимовний контент одним запитом без JOIN та гнучко розширювати перелік мов без зміни схеми [35]. З огляду на те, що система орієнтована переважно на україномовний ринок і мультимовний контент є допоміжним, для проєкту обрано підхід JSONB.

Аналогічне рішення прийнято для SEO-метаданих, атрибутів препаратів та тіла замовлення. У всіх цих випадках JSONB забезпечує необхідну гнучкість схеми при збереженні можливості індексування та path-запитів засобами PostgreSQL.

Для зв’язків типу «багато-до-багатьох» між препаратами та категоріями, а також між препаратами та групами, розглядалися стандартні зв’язкові таблиці та масиви цілих чисел. Зв’язкові таблиці забезпечують референтну цілісність на рівні СУБД, однак вимагають JOIN при кожному читанні та збільшують кількість таблиць у схемі. Оскільки зазначені зв’язки використовуються переважно для читання, а цілісність гарантується на рівні застосунку, прийнято рішення використовувати масиви індентифікаторів – це суттєво спрощує запити та прискорює читання [34].

Ієрархічна структура каталогу категорій препаратів є однією з найскладніших задач проєктування, оскільки вимагає ефективного виконання кількох типів запитів: отримання всього піддерева категорій, отримання шляху від вузла до кореня (для реалізації функціоналу «хлібних крихт») та вибірки прямих нащадків. Для представлення ієрархії в реляційних базах даних існує кілька класичних підходів, кожен із яких має свої характеристики продуктивності.

Модель суміжних списків зберігає лише посилання на батьківський вузол. Вона проста у підтримці, однак отримання всього піддерева потребує рекурсивного SQL-запиту, що є дорогим при великій глибині ієрархії. Шлях матеріалізації зберігає у кожному вузлі масив ідентифікаторів усіх предків, що дозволяє отримати шлях до кореня за O(1). Модель вкладених множин

присвоює кожному вузлу ліву та праву межі, що уможливлює отримання всього піддерева єдиним запитом без рекурсії.

З таблиці 2.1 випливає, що жоден із підходів окремо не забезпечує оптимальної продуктивності для всіх типів запитів. Тому для таблиці категорій прийнято рішення застосувати комбінований підхід (рис. 2.4): Суміжні списки для простих запитів прямих нащадків і підтримки вставки, шлях матеріалізації для ефективного отримання хлібних крихт, та вкладені множини для швидкого отримання всього піддерева. Такий підхід максимізує продуктивність читання ціною дещо вищої складності операцій запису, що є прийнятним для каталогу, де читання значно переважає над записом.

Таблиця 2.1 Порівняння підходів до зберігання ієрархічних даних

Підхід	Читання піддерева	Читання шляху	Оновлення
Суміжні списки	O(n)	O(depth)	O(1)
Шлях матеріалізації	O(n)	O(1)	O(depth)
Вкладені множини	O(1)	O(n)	O(n)

Фасетна фільтрація вимагає зберігання атрибутів препаратів забезпечуючи гнучке розширення переліку атрибутів та ефективну фільтрацію. Розглядалися два підходи: модель «Entity-Attribute-Value» (EAV) із окремими рядками для кожного атрибуту та зберігання атрибутів у JSONB-полі таблиці препаратів. Модель EAV є надзвичайно гнучкою, але вимагає складних pivot-запитів і погано масштабується при великій кількості атрибутів. JSONB-підхід дозволяє зберігати довільний набір атрибутів у структурованому вигляді та ефективно передавати їх до пошукового рушія MeiliSearch для індексування [36].

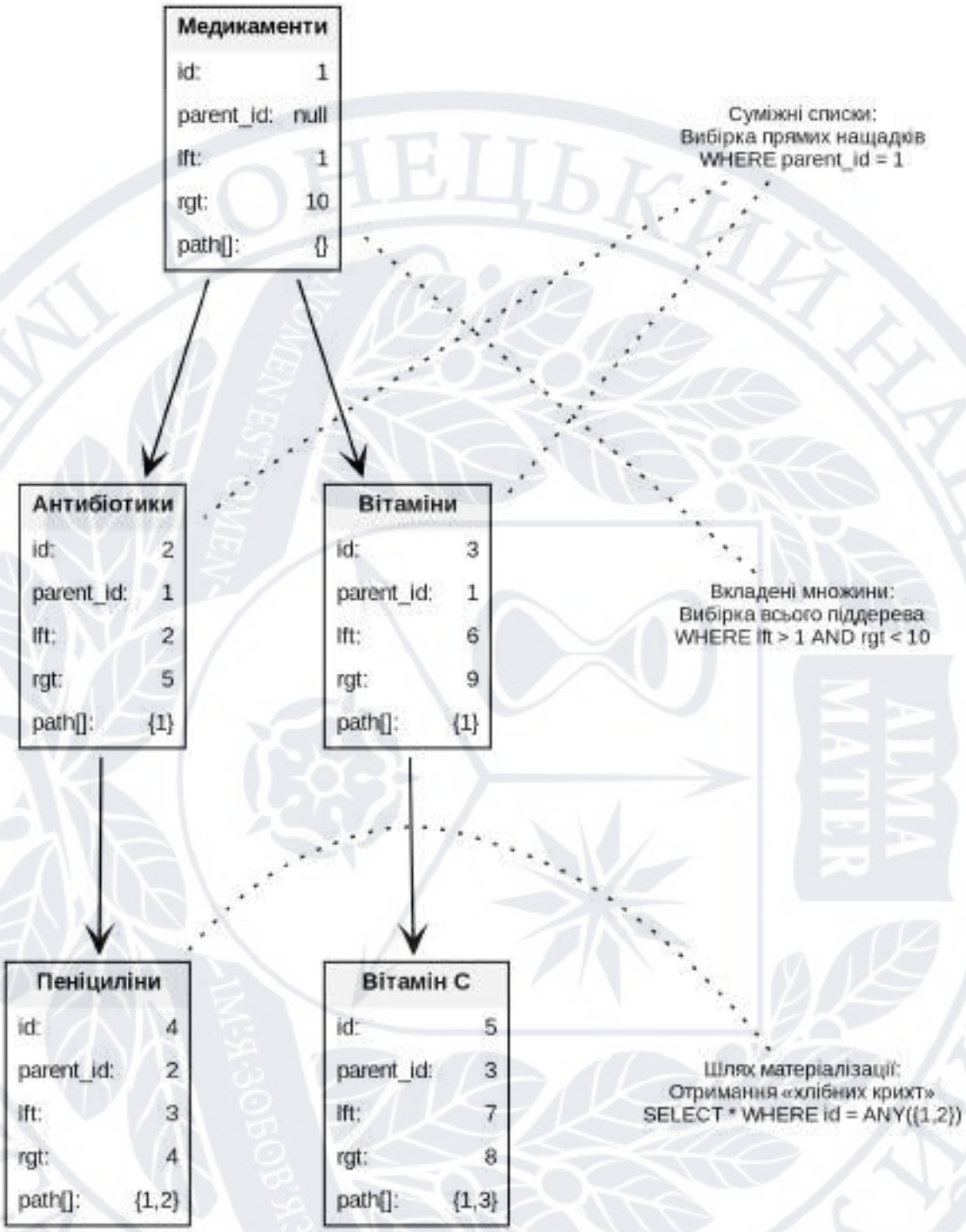

Рисунок 2.4 – Проєктна схема ієрархії категорій (комбінований підхід)

Прийнято рішення зберігати атрибути препарату у JSONB-полі таблиці products, а метадані про типи атрибутів – в окремій таблиці «product_attributes». Таблиця атрибутів містить ENUM-типи для типу елемента інтерфейсу, що забезпечує типобезпечну конфігурацію фасетних фільтрів на рівні бази даних.

Для домену замовлень ключовим проєктним рішенням є підтримка кількох типів замовлень з різною структурою тіла. Створення окремих таблиць для кожного типу призвело б до надмірної кількості NULL-полів або складних успадкованих таблиць. На основі цього прийнято рішення зберігати тіло замовлення у JSONB-полі, а тип замовлення – у цілочисельному полі-дискримінаторі. Такий підхід дозволяє гнучко розширювати типи замовлень без зміни схеми таблиці та полегшити інтеграцію зовнішніх сервісів в існуючу систему [34].

Для відстеження статусів замовлення спроєктована окрема таблиця «order_statuses» із зв’язком 1:N до таблиці замовлень, що дозволяє зберігати повну історію зміни статусів.

Для кошика прийнято stateless-підхід: ідентифікатори кошиків зберігаються у JWT-токені, що дозволяє підтримувати анонімні та авторизовані кошики без серверного стану сесій і забезпечує горизонтальну масштабованість системи без розподіленого сховища [37].

Управління схемою планується здійснювати виключно через версіоновані міграції з вимкненим автосинхронізацією ORM. Такий підхід гарантує повну відтворюваність стану схеми на будь-якому середовищі та унеможливлює випадкову втрату даних при оновленні застосунку [38].

Для всіх таблиць визначено єдині конвенції іменування: назви таблиць і колонок у форматі snake case, первинні ключі – автоінкрементні цілочисельні, обов’язкові часові мітки «created_at» та «updated_at».

Спроєктована реляційна модель бази даних відповідає всім функціональним і нефункціональним вимогам системи: гібридна модель зберігання забезпечує необхідну гнучкість, комбінований підхід до ієрархії категорій – оптимальну продуктивність запитів, JSONB-атрибути – розширюваність фасетної фільтрації, а міграційний підхід – надійність управління схемою. Спроєктована модель слугуватиме основою для програмної реалізації, що розглядається у наступному розділі.

2.3. Розробка алгоритму фасетної фільтрації

Фасетна фільтрація є центральним алгоритмічним механізмом системи, що забезпечує інтерактивне звуження каталогу препаратів за множинними незалежними критеріями. На відміну від простої текстової фільтрації, фасетний пошук вимагає вирішення нетривіальної задачі: після кожної дії користувача система має не лише оновити список результатів, але й динамічно перерахувати кількість товарів, що відповідають кожному значенню кожного фільтра. Це так звані «лічильники фасетів». Саме ці лічильники запобігають ситуації «нульового результату», коли користувач обирає несумісну комбінацію параметрів [39].

У фармацевтичному каталозі задача ускладнюється двома специфічними вимогами. Атрибути препаратів є різнорідними: одні є перелічуваними, інші – числовими діапазонами. Окрім цього доступність та ціна одного й того самого препарату різняться залежно від обраного міста, що означає необхідність геолокаційної прив’язки результатів фільтрації.

Ключовим проєктним рішенням є вибір логіки комбінування значень фільтрів. Існують два класичних підходи: кон’юнктивна фільтрація (AND між усіма значеннями) та диз’юнктивна фільтрація всередині фасету (OR всередині одного фасету, AND між різними фасетами). Перший підхід є надмірно обмежувальним: якщо користувач обирає одночасно «Пігулки» і «Капсули», кон’юнктивна логіка поверне порожній результат, оскільки жоден препарат не може мати обидві форми випуску одночасно. Другий підхід – диз’юнктивний – є природнішим з точки зору UX: вибір кількох значень у межах одного фасету розширює вибірку, а вибір значень у різних фасетах – звужує.

Для розроблюваної системи обрано диз'юнктивну фасетну фільтрацію, що формально записується як:

$$Result = \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} v_{ij} \right) \tag{2.1}$$

де n – загальна кількість фасетів;

m_i – кількість обраних значень у i-му фасеті;

v_{ij} – j-те обране значення всередині i-го фасету.

Числові фільтри застосовуються як діапазони: $min \le value \le max$.

Обчислення точних лічильників фасетів у диз'юнктивній логіці є нетривіальною задачею. Якщо виконати один запит із усіма активними фільтрами, лічильники для вже обраних фасетів будуть некоректно занижені – вони не відображатимуть кількість товарів, доступних при зміні значення у цьому фасеті. Для вирішення цієї проблеми розроблено алгоритм паралельних запитів $N+1$, де N – кількість активних фасетів.

Вибір саме цієї стратегії обґрунтований архітектурними особливостями пошукового рушія MeiliSearch. На відміну від інших систем на кшталт Elasticsearch, MeiliSearch оптимізований для забезпечення мінімальної затримки при повнотекстовому пошуку, проте він не підтримує механізм складних вкладених агрегацій у межах одного запиту. Використання алгоритму $N+1$ дозволяє обійти це обмеження, використовуючи високу пропускну здатність рушія та асинхронну природу середовища NestJS.

Алгоритм функціонує таким чином: виконується один основний запит із усіма активними фільтрами, що повертає список товарів; паралельно для кожного активного фасету F_i виконується окремий запит, у якому фільтр саме цього фасету виключений, – він повертає коректні лічильники для значень F_i. Усі $N+1$ запитів виконуються паралельно, тому загальна затримка відповіді дорівнює часу одного найдовшого запиту, а не їхній сумі. Схематично цей процес зображено на рис. 2.5.

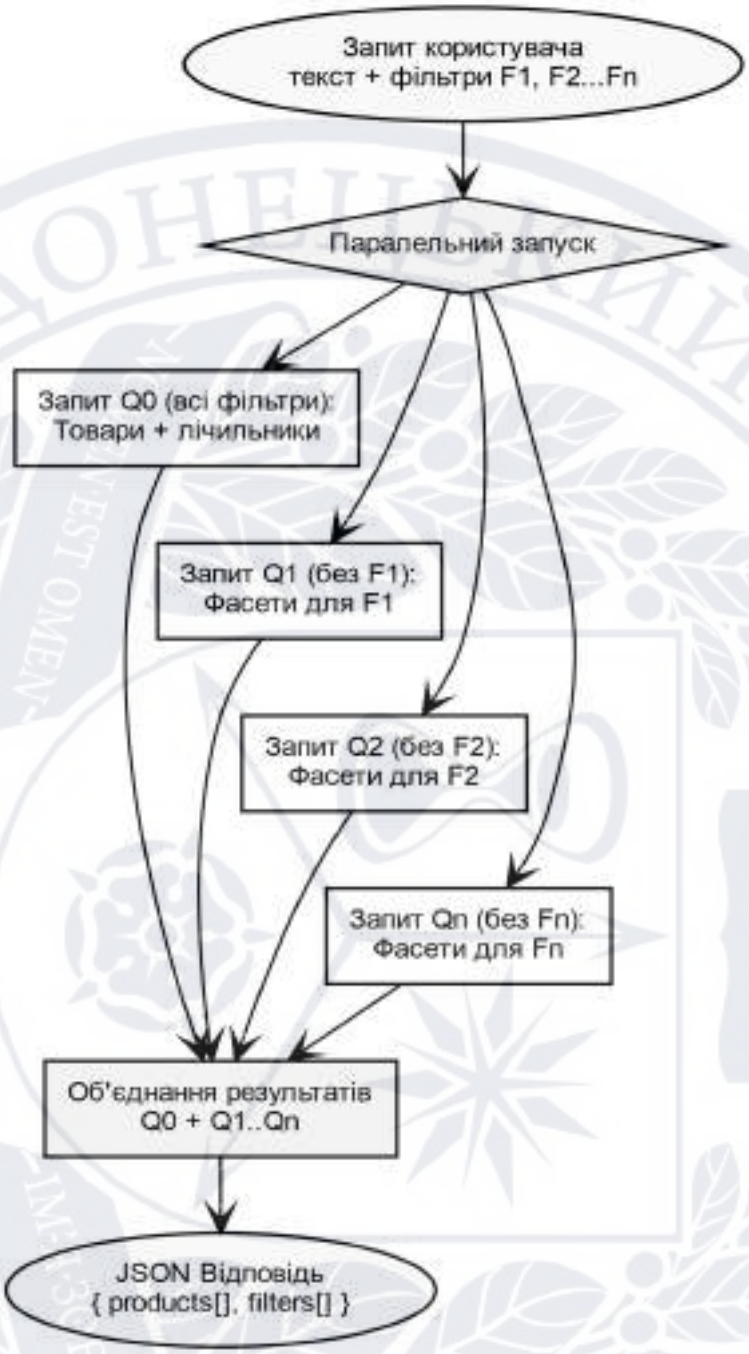

Рисунок 2.5 – Схема алгоритму $N+1$ обчислення лічильників фасетів

Важливим проєктним рішенням є підхід до конфігурації фасетів. Фіксований перелік фасетів у коді застосунку унеможливлює їхнього розширення без нової збірки проєкту, тому розроблено підхід на основі конфігурації через базу даних: кожен фасет описується записом у таблиці «product_attributes» із метаданими про тип UI-елемента, джерело даних, порядок відображення та ознаку індексування у пошуковому рушієві. Такий

підхід дозволяє адміністратору системи додавати нові фасети без втручання в код застосунку.

Окремо визначено два типи службових фільтрів. Транзитні фільтри застосовуються до кожного запиту автоматично. Вони включають в себе:

- наявність товару;
- активність позиції;
- обмеження за категорією;

Дані фільтри не відображаються в інтерфейсі. Другий тип службових фільтрів – приватні фільтри. Вони передаються клієнтом, застосовуються до результатів, але також не відображаються як окремі UI-елементи. Наприклад, фільтрація за ідентифікатором міста для геолокаційного звуження.

Специфічною вимогою фармацевтичної системи є відображення цін та наявності з урахуванням обраного міста. Ця вимога реалізована на рівні структури пошукового індексу: під час індексування до кожного документа додається поле «remnantsStats» – асоціативний масив, де ключем є ідентифікатор міста, а значенням – агреговані показники: мінімальна ціна, максимальна ціна та загальна кількість у аптеках цього міста. Завдяки цьому ціновий фасет динамічно адаптується до обраного міста без додаткових запитів до бази даних – вся необхідна інформація міститься в індексованому документі.

Обґрунтований вибір технологічного стеку системи FaPharm базується на сукупності критеріїв: відповідності функціональним вимогам, зрілості інструментів, продуктивності під очікуваним навантаженням та можливості інтеграції компонентів у єдину архітектуру. Поєднання NestJS на серверній стороні, PostgreSQL як основного сховища даних, MeiliSearch як спеціалізованого пошукового рушія та Vue 3 на клієнтській стороні забезпечує оптимальний баланс між швидкістю розробки, підтримуваністю коду та продуктивністю системи під навантаженням.

Розроблена архітектура вебсистеми забезпечує незалежність розробки та масштабування кожного компонента. Спроєктована реляційна модель бази даних у PostgreSQL із використанням JSONB-полів для мультимовності, моделі вкладених множин для ієрархічних категорій та денормалізованого пошукового індексу MeiliSearch із попередньо агрегованими статистиками залишків дозволяє виконувати геолокаційну фільтрацію без додаткових звернень до реляційної бази даних.

Розроблений алгоритм $N+1$ паралельних запитів для фасетної фільтрації вирішує ключову технічну проблему коректного підрахунку лічильників: для кожного активного фасету виконується окремий запит із його виключенням, а всі запити виконуються паралельно через Promise.all, що мінімізує загальний час відповіді. Такий підхід обумовлений архітектурними обмеженнями пошукового рушія MeiliSearch і забезпечує точність фасетних лічильників незалежно від кількості одночасно активних фільтрів.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ FAPHARM

3.1. Розробка серверної частини

Серверна частина системи FaPharm реалізована у вигляді REST API на базі фреймворку NestJS і є програмним втіленням архітектурних рішень, описаних у другому розділі. Точка входу застосунку ініціалізує глобальний ValidationPipe із параметрами transform: true та whitelist: true, що автоматично трансформує вхідні DTO та відкидає незадекларовані поля, запобігаючи масовому присвоєнню атрибутів. Усі маршрути API згруповані під префіксом /api/v1/. Кореневий модуль AppModule об'єднує всі функціональні підмодулі та три глобальних провайдери: ConfigModule для роботи зі змінними середовища, CacheModule для in-memory кешування та JwtTokenModule, що робить JWT-сервіс доступним в усіх частинах застосунку без явного імпорту.

Модуль auth реалізує повний цикл управління ідентичністю: реєстрацію, активацію облікового запису, вхід та управління JWT-токенами. Найважливішою особливістю реалізації є розширений JWT-payload, що містить не лише ідентифікатор і роль користувача, а й масив ідентифікаторів активних кошиків та унікальний UUID сесії. Завдяки цьому анонімні відвідувачі можуть формувати кошики без реєстрації, а при подальшому вході в систему ці кошики автоматично прив'язуються до облікового запису.

Процес реєстрації передбачає відстрочену активацію: щойно створений обліковий запис залишається неактивним до підтвердження 4-значним числовим кодом. Для захисту від зловживань повторний запит коду обмежено 60-секундним інтервалом, що перевіряється через поле часової мітки запису. Паролі зберігаються виключно у хешованому вигляді за алгоритмом bcrypt.

Система захисту маршрутів реалізована через три типи guard-ів: обов'язковий JwtAuthGuard, що повертає помилку 401 за відсутності або

недійсності токена; опціональний OptionalJwtAuthGuard, що пропускає запит навіть без токена, зберігаючи null у payload; та RolesGuard для перевірки прав доступу за роллю користувача в JWT.

Модуль «products» є найбільшим за обсягом і складається з шести підмодулів: основної сутності товару, конфігурації фасетних атрибутів, груп і брендів, залишків по аптеках, типів препаратів та інструкцій до застосування. Кожен підмодуль є самодостатньою одиницею з власним контролером, сервісом та TypeORM-сутністю.

Сутність «Product» реалізує мультимовність через JSONB-поля для назви, короткої назви та SEO-метаданих. Сервіс автоматично виконує трансформацію мовних полів при видачі: повертає потрібну локаль або українську за замовчуванням. Метод отримання популярних товарів, реалізований через нативний SQL-запит до PostgreSQL. Він агрегує JSONB-поле «body_list» усіх замовлень за останні 7 днів, з'єднує результати з таблицею товарів та повертає відсортований за частотою придбання список – без ORM-абстракцій, що забезпечує максимальну продуктивність цього специфічного запиту.

Підмодуль атрибутів реалізує конфігурацію фасетів через базу даних. При старті застосунку сервіс завантажує та кешує повний список атрибутів у пам'яті. Кеш оновлюється через адміністративний модуль без перезапуску сервера. Підмодуль залишків реалізує тернарний зв'язок між товаром, аптекою та адміністративною одиницею із полями ціни та кількості – це основне джерело даних для пошукового індексу та фільтрації за наявністю в обраному місті.

Модуль «search» є технологічно найскладнішим компонентом серверної частини. Підтримуються три пошукових індекси MeiliSearch: індекс товарів, індекс груп і брендів, а також індекс адміністративних одиниць для автодоповнення при виборі міста. При старті застосунку автоматично виконується ініціалізація всіх індексів: перевіряється їхня наявність,

створюються відсутні та оновлюється конфігурація атрибутів – доступних для пошуку, фільтрації та сортування.

Побудова індексу товарів є багатоетапним процесом. Спочатку виконуються два паралельних запити: основний SQL-запит товарів та побудова словника відповідності slug-значень атрибутів їхнім локалізованим назвам. Основний запит використовує два LATERAL JOIN для агрегації залишків: перший обчислює статистику по кожному місту окремо, другий – загальну по всій мережі аптек. Результат зберігається в документі як поле «remnantsStats». Завдяки такій попередній агрегації пошукові запити з геолокаційним фільтром не потребують жодних звернень до PostgreSQL – вся необхідна інформація вже знаходиться в індексованому документі.

Фасетний пошук реалізує алгоритм $N+1$ паралельних запитів, спроєктований у підрозділі 2.3. Метод отримує текст запиту, масив активних фільтрів та ідентифікатор міста, розподіляє фільтри на категорії та будує базовий рядок фільтрації. Приклад структури такого JSON-запиту до API та отриманої відповіді наведено на рис. 3.1.

Рисунок 3.1 – Структура запиту та відповіді модуля фасетного пошуку

Для кожного активного фасету виконується окремий запит без цього фасету – він повертає коректні лічильники значень. Усі запити виконуються паралельно через Promise.all, що забезпечує швидкий час відповіді.

Модуль комерції складається з двох підмодулів – кошика та замовлень. Він реалізує повний цикл покупки. Модуль кошика застосовує stateless-підхід: ідентифікатор кошика зберігається безпосередньо у JWT-токені. Під час створення кошика сервіс генерує новий токен із доданим ідентифікатором і повертає його клієнту разом із даними кошика. Вміст кошика зберігається у JSONB-полі як масив позицій із ідентифікаторами товарів та кількістю.

Модуль замовлень реалізує захист від подвійного оформлення через in-memory колекцію у вигляді Set: перед початком обробки ідентифікатор кошика додається до неї, а видалення гарантується блоком finally навіть у разі виключення. Повторний запит із тим самим кошиком отримує відповідь «409 Conflict». Контроль доступу до конкретного замовлення реалізовано дворівнево: для зареєстрованих користувачів – звірка ідентифікатора, для анонімних – звірка UUID із JWT-токена. Це гарантує, що замовлення, оформлені без реєстрації, залишаються доступними лише в межах тієї ж сесії.

Модуль категорій реалізує ієрархічний каталог з кешуванням у пам’яті. При старті сервіс завантажує плоский список категорій із полями лівої та правої межі вкладених множин, будує дерево та зберігає його у кеш. Подальші запити до ієрархії виконуються без звернень до бази даних.

Модуль КАТОТТГ надає географічний довідник адміністративно-територіальних одиниць та реалізує пошук найближчого міста за GPS-координатами безпосередньо у SQL-запиті PostgreSQL через формулу Гаверсина, яка має такий вигляд:

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \quad (3.1)$$

де d – відстань між двома точками по великому колу сфери (км);

r – середній радіус Землі (6371 км);

φ_1, φ_2 – широта першої та другої точки відповідно (у радіанах);

λ_1, λ_2 – довгота першої та другої точки відповідно (у радіанах).

Модуль аптек управляє даними аптечних мереж і підтримує cursor-based пагінацію для реалізації безкінечного прокручування на клієнті. Цей підхід забезпечує сталу складність запиту незалежно від обсягу даних та виключає дублювання записів при паралельному додаванні нових позицій.

Адміністративний модуль налаштувань надає API для точкового або масового оновлення кешу та пошукових індексів без перезапуску сервера. Підтримуються три режими оновлення індексу: повна перебудова, часткове оновлення конкретних документів та видалення деактивованих позицій.

Спільна інфраструктура застосунку зосереджена в директорії common і містить: декоратор @JWTPayload() для витягування payload із запиту, набір типових guard-ів для перевірки ролей та ідентифікації користувача, pipe для захисту від integer overflow у PostgreSQL при передачі числових параметрів, а також централізовані TypeScript-інтерфейси та константи бізнес-логіки.

Серверна частина системи FaPharm реалізована як повнофункціональний REST API із дотриманням принципів модульності, типобезпеки та надійності. Усі архітектурні рішення другого розділу знайшли безпосереднє відображення у програмній реалізації:

- stateless JWT-автентифікація;
- денормалізований пошуковий індекс;
- алгоритм $N+1$ паралельних запитів;
- захист від подвійного оформлення замовлення.

3.2. Реалізація механізмів пошуку

Пошукова підсистема системи FaPharm реалізована у межах модуля search і є програмною імплементацією алгоритмів, теоретичне обґрунтування яких наведено у підрозділі 2.3. Архітектура підсистеми базується на взаємодії сервісу SearchService із трьома спеціалізованими індексами пошукового рушія MeiliSearch: products (товари), groups (категорії) та katottg (адміністративно-територіальні одиниці). У даній моделі реляційна база

даних PostgreSQL функціонує як першоджерело еталонних даних, тоді як MeiliSearch виступає у ролі денормалізованого кешу, оптимізованого для високопродуктивного повнотекстового пошуку.

Процес ініціалізації пошукової інфраструктури інтегрований у життєвий цикл застосунку через інтерфейс OnApplicationBootstrap. Під час запуску системи метод indexesInit здійснює динамічне формування конфігурації пошукових індексів на основі метаданих бази даних. Зокрема, виконується аналіз таблиці product_attributes для ідентифікації ознак, що мають атрибут search_engine. Виявлені ключі автоматично реєструються у списках параметрів фільтрації (filterableAttributes) та фасетування (facetingAttributes) індексу товарів. Такий підхід забезпечує гнучкість системи: додавання нових характеристик до бази даних активує відповідні пошукові механізми без необхідності модифікації програмного коду.

Синхронізація конфігурації забезпечується через ідемпотентну операцію createIndexIfNotExists. Вона включає перевірку наявності цільового індексу, його створення за потреби та обов’язкове оновлення параметрів ранжування, сортування, пагінації та фасетування. Це гарантує цілісність конфігурації пошукового рушія відносно поточного стану схеми даних після кожного циклу розгортання застосунку.

Процес підготовки даних до індексації складається з двох послідовних етапів. Перший етап – це агрегація даних на рівні СУБД: Виконуються паралельні SQL-запити для отримання основного масиву товарів та побудови словника локалізованих значень (localizedSlugMap). Останній використовується для заміни ідентифікаторів атрибутів на їхні текстові відповідники при формуванні відповідей користувачу. Для обчислення залишків і цінових показників застосовано механізм LATERAL JOIN, що дозволяє агрегувати статистику у форматі JSONB безпосередньо у запиті. Це включає розрахунок кількості аптек та цінового діапазону як для конкретних населених пунктів (об’єкт remnantsStats), так і сумарно по всій мережі

(aggregatedRemnantsStats). Другим етапом є постпроцесинг та нормалізація:

Кожен запис обробляється методом processProductData, який виконує трансформацію складних JSONB-структур у лінійні масиви значень (slugs). Додатково генерується поле normalizedName, в якому назва препарату очищується від спеціальних символів, що підвищує релевантність повнотекстового пошуку.

Після завершення обробки сформовані документи передаються до MeiliSearch пакетами для забезпечення високої швидкості індексації. Нижче представлено приклад структури нормалізованого документа, підготовленого до передачі в пошуковий індекс.

```
{
  "id": 1842,
  "slug": "aspirin-kardio-100mg",
  "name": "Аспірин кардіо",
  "normalizedName": "Аспірин кардіо",
  "inStock": true,
  "production-form": "tabletky",
  "manufacturer": "bayer-ag",
  "remnantsStats": {
  "12345": {
    "pharmaciesCount": 3,
    "price": { "from": 45.50, "to": 78.00 }
  },
  "67890": {
    "pharmaciesCount": 1,
    "price": { "from": 52.00, "to": 52.00 } }
  },
  "aggregatedRemnantsStats": {
    "pharmaciesCount": 4,
    "price": { "from": 45.50, "to": 78.00 }
  }
}
```

Програмна реалізація пошукової підсистеми охоплює чотири ключові сценарії взаємодії, що забезпечують комплексне задоволення інформаційних потреб користувача. Загальний пошук базується на паралельному зверненні до індексів products та groups, повертаючи ієрархічно впорядковані результати, які включають до десяти товарних позицій та три категорії,

ранжовані за показником наявності. Локалізований пошук у межах обраної аптеки обмежує вибірку за специфічним ідентифікатором через фільтрацію поля залишків, що дозволяє взаємодіяти з асортиментом конкретної точки продажу. Для забезпечення коректного вибору регіону реалізовано механізм автодоповнення міст, який при зверненні до індексу katottg повертає перелік населених пунктів, де зафіксовано хоча б одну активну аптеку.

Найбільш складним із технічної точки зору є сценарій фасетного пошуку, що базується на алгоритмі паралельних запитів, обґрунтованому у підрозділі 2.3. Вхідні параметри пошуку включають текстовий запит, географічний ідентифікатор, контекст категорії та масив активних фільтрів, представлених у формі дескрипторів. Процес виконання запиту розпочинається з трансформації обов'язкових умов методом transitFilters, який формує базові обмеження, зокрема пріоритетний фільтр наявності. У разі фільтрації за категорією система використовує модель вкладених множин для отримання ідентифікаторів усіх дочірніх підкатегорій із кешу, що дозволяє виконувати пошук на будь-якому рівні ієрархії без рекурсивних запитів до реляційної бази даних.

На наступному етапі система здійснює класифікацію та обробку вхідних фільтрів, розподіляючи їх на одинарні, діапазонні та службові категорії. Особливістю обробки діапазонних фільтрів для цінових показників є їхня динамічна прив'язка до вкладеного об'єкта remnantsStats відповідно до обраної локації. Для отримання точних кількісних показників фасетів формується набір паралельних запитів через механізм Promise.all, що включає основний запит зі збереженням усіх обмежень та серію допоміжних запитів із послідовним виключенням кожного активного фасету. Фінальна відповідь системи формується шляхом консолідації даних, де перелік товарів вилучається з основного запиту, агреговані значення фасетів – з допоміжних звернень, а технічні ідентифікатори атрибутів трансформуються у локалізовані назви за допомогою словника localizedSlugMap.

Для підтримки цілісності даних у модулі RefreshModule передбачено три стратегії оновлення індексів, що обираються залежно від характеру змін у системі. Режим повної реіндексації передбачає видалення поточної структури документів та повну перебудову індексу, що є доцільним при масштабних змінах у структурі каталогу. Стратегія часткової синхронізації дозволяє оновлювати дані конкретних товарів за їхніми ідентифікаторами, при цьому параметр повного заміщення визначає, чи буде документ оновлено вибірково через злиття атрибутів, чи замінено повністю. Для оперативного вилучення документів із індексу при деактивації товарних позицій у базі даних застосовується режим деструктивного оновлення.

Розроблена пошукова підсистема повністю відповідає встановленим функціональним вимогам, забезпечуючи повнотекстову обробку запитів із корекцією орфографічних помилок, складну фасетну фільтрацію з коректним обчисленням кількісних показників та динамічну адаптацію результатів до геолокації користувача при збереженні високої швидкості відклику системи.

3.3. Створення клієнтського SPA-додатка

Клієнтська частина системи FaPharm реалізована як односторінковий застосунок (SPA) на базі фреймворку Vue 3 із використанням прикладного інтерфейсу Composition API. Точка входу в систему забезпечує ініціалізацію менеджеру станів Pinia та маршрутизатора vue-router, після чого здійснюється монтування кореневого компонента. На етапі ініціалізації автоматично виконуються методи визначення геолокації користувача та відновлення активної сесії на основі збереженого токена автентифікації. Кореневий компонент App.vue формує загальну структуру інтерфейсу (shell), де статичні блоки заголовка (AppHeader) та підвалу (AppFooter) обрамляють область динамічного відображення контенту (router-view). Загальний вигляд інтерфейсу на прикладі головної сторінки представлено на рис. 3.2.

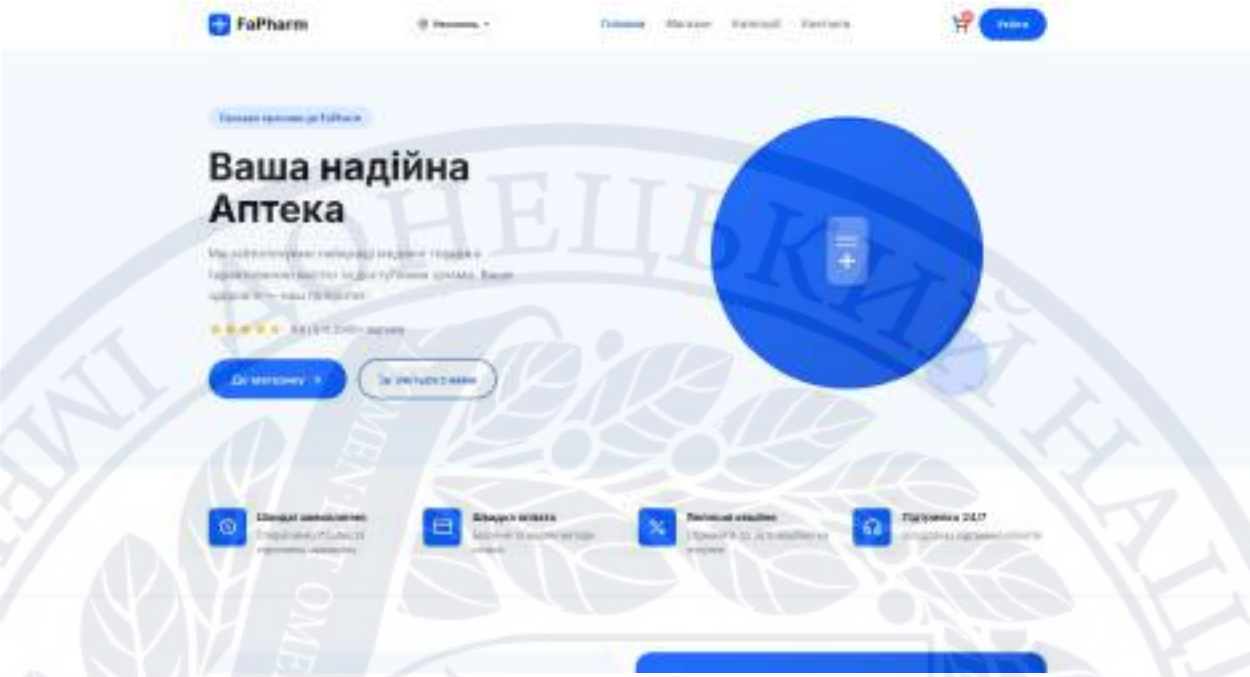

Рисунок 3.2 – Головна сторінка застосунку FaPharm

Для оптимізації продуктивності всі дванадцять маршрутів системи використовують механізм динамічного імпорту. Це забезпечує автоматичне розділення програмного коду засобами інструментарію Vite: початкова збірка містить лише критично важливі ресурси для першого рендеру, тоді як програмна логіка окремих сторінок завантажується за запитом під час навігації. Захист конфіденційних розділів, таких як профіль користувача та історія замовлень, реалізовано через мета-параметри маршрутизації, що обмежують доступ для неавторизованих суб’єктів.

Мережева взаємодія із серверною частиною базується на централізованому об’єкті-екземпляр бібліотеки Axios, конфігурація якого зосереджена у модулі api/http.ts [40]. Для автоматизації обробки трафіку впроваджено систему перехоплювачів. Перехоплювач запитів здійснює динамічну ін’єкцію заголовка Authorization за схемою Bearer, використовуючи токен із локального сховища браузера. Паралельно перехоплювач відповідей забезпечує централізований менеджмент помилок: у разі отримання статусу 401 Unauthorized система автоматично анулює застарілі облікові дані та ініціює процедуру повторної автентифікації. Структурно рівень API декомпоновано на автономні модулі відповідно до

функціональних доменів системи, зокрема авторизації, управління номенклатурою, пошуку та логістики.

Управління глобальним станом організовано через чотири спеціалізовані сховища Pinia. Сховище useAuthStore регламентує життєвий цикл JWT-токена та профілю користувача, забезпечуючи коректне очищення даних при завершенні сеансу. Робота з кошиком у межах useCartStore реалізована за двошаровою архітектурою: локальне сховище виступає пріоритетним джерелом даних для миттєвого відгуку інтерфейсу, тоді як асинхронна синхронізація із сервером виконується у фоновому режимі. Ключовою технологією забезпечення безперервного користувацького досвіду є механізм оптимістичних оновлень. Він дозволяє інтерфейсу реагувати на дії користувача локально в цей самий момент, з подальшим підтвердженням операції на рівні API та можливістю автоматичного повернення до стабільного стану у разі виникнення мережевих помилок.

Додаткові модулі useCategoriesStore та useCityStore здійснюють кешування ієрархії товарних категорій та географічних параметрів, що суттєво знижує навантаження на мережу під час повторних звернень до системи. Практична реалізація взаємодії зазначених сховищ та API-модулів відображена в інтерфейсі сторінки каталогу (рис. 3.3), де завантаження даних та фільтрація відбуваються динамічно на основі глобального стану.

Архітектура інтерфейсу базується на принципах високої інтерактивності та динамічного формування контенту залежно від географічного контексту користувача. Центральне місце у системі посідають сторінки каталогу (ShopPage) та категорій (CategoryPage), які забезпечують механізм фасетного пошуку. Для оптимізації навантаження на мережу та серверні потужності під час введення текстових запитів впроваджено часову затримку (300 мс), що дозволяє групувати запити та уникати надлишкових звернень до пошукового рушія.

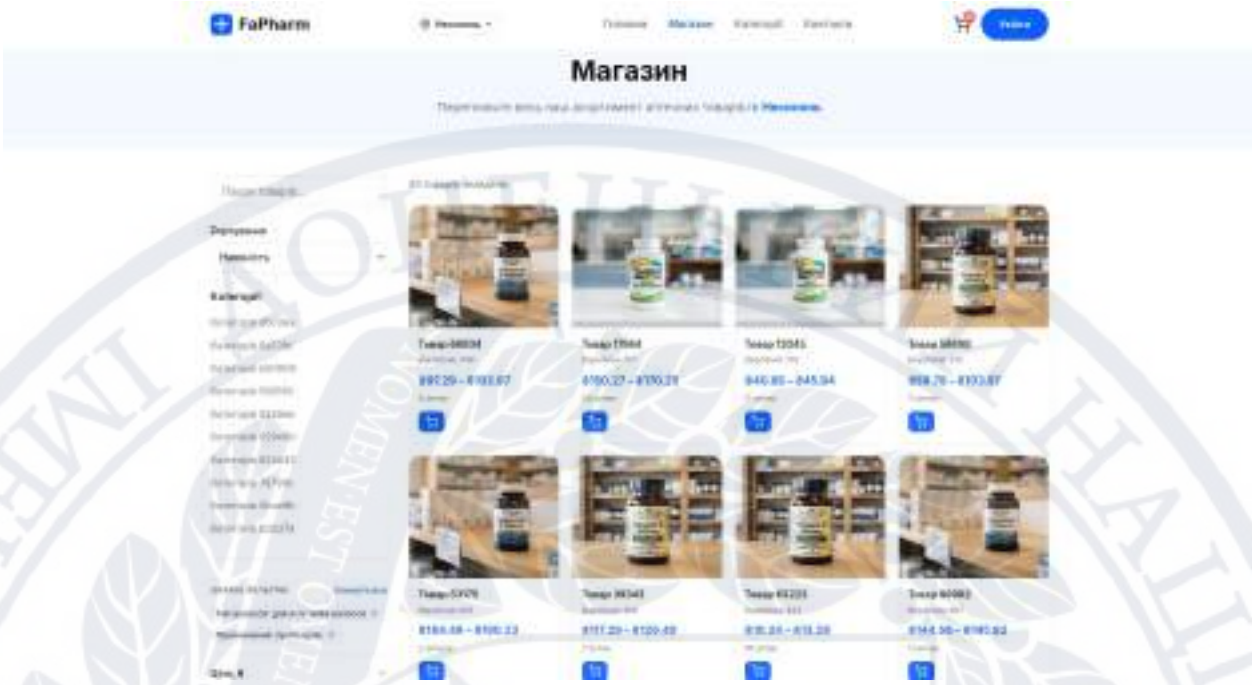

Рисунок 3.3 – Сторінка каталогу з панеллю фасетних фільтрів

Важливою особливістю алгоритму ранжування результатів є його детермінованість відносно обраної локації користувача. У разі ідентифікації міста система пріоритезує видачу товарів, що є в наявності у відповідному населеному пункті, використовуючи параметри сортування за залишками. Функція buildRetrieveAttributes забезпечує динамічну генерацію схеми даних для MeiliSearch, що дозволяє включати специфічні атрибути (цінові показники та доступність у регіональних точках продажу) лише тоді, коли це необхідно для обраного географічного контексту.

Сторінка товару реалізує ієрархічну логіку відображення вартості. Система в першу чергу виконує перевірку регіональної ціни, а у разі її відсутності – звертається до агрегованого показника середньої ціни по всій мережі. Формування переліку характеристик препарату відбувається шляхом консолідації даних із різних секцій об’єкта, що супроводжується усуненням дублювання та впорядкуванням згідно з визначеними пріоритетами. Модуль медичних інструкцій спроєктований як універсальний інтерфейс, що підтримує обробку зовнішніх HTML-ресурсів через мережеві запити, роботу з текстовими даними та локалізованими об’єктами.

Найвищий рівень складності бізнес-логіки зосереджений на сторінці оформлення замовлення (рис. 3.4). Даний модуль реалізує автоматичне заповнення реквізитів на основі даних профілю автентифікованого користувача та забезпечує реактивне завантаження списку пунктів видачі відповідно до обраного коду адміністративно-територіальної одиниці. Критичним елементом функціоналу є валідація вмісту кошика в режимі реального часу: система перевіряє наявність кожної товарної одиниці у конкретно обраній точці продажу через запити до API. У разі виявлення дефіциту хоча б однієї позиції, механізм підтвердження замовлення блокується на рівні інтерфейсу для запобігання некоректним транзакціям. Крім того, передбачена можливість делегування отримання замовлення третій особі через введення альтернативних даних отримувача.

Для забезпечення високого рівня інтерактивності розроблено систему складних компонентів, що адаптуються до контексту використання. Центральним елементом управління є компонент AddToCartButton, який реалізує сутність із чотирма станами:

- ініціальний, що відповідає за відображення базового елемента додавання до кошика;
- транзитний, що відповідає за візуалізацію процесу виконання асинхронного запиту до API;
- підтверджувальний, що відповідає за індикацію успішного завершення операції;
- операційний, що відповідає за відображення інтерактивного контролера кількості товарних одиниць, якщо позиція вже присутня у кошику.

Для оптимізації відображення у різних частинах інтерфейсу компонент підтримує параметр compact, який приховує текстові мітки в межах карток каталогу.

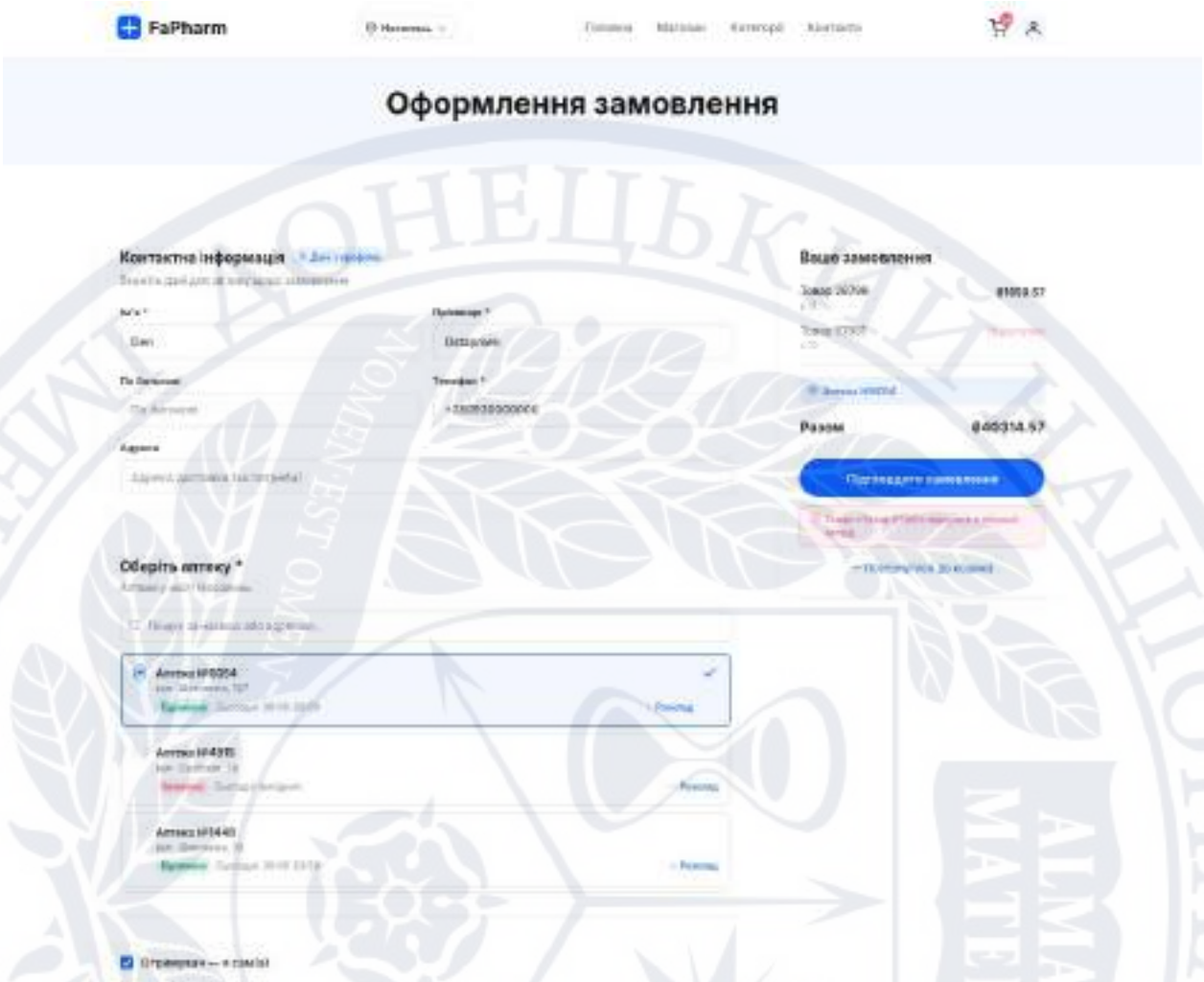

Рисунок 3.4 – Сторінка оформлення замовлення

Механізм фільтрації реалізовано в компоненті FacetFilters, що забезпечує повний цикл взаємодії з атрибутами товарів. Система автоматично адаптує інтерфейс залежно від кількості значень: для списків, що перевищують вісім елементів, інтегрується внутрішній пошук, а для груп понад шість значень – механізм розгортання прихованого контенту. Числові параметри (наприклад, діапазон цін) опрацьовуються через поля введення мінімальних та максимальних меж. Обрані параметри візуалізуються над панеллю фільтрів у формі динамічних елементів, що дозволяє здійснювати їхню індивідуальну деактивацію.

Компонент CitySelector, інтегрований у верхню панель навігації. Він базується на випадаючому списку з функцією предиктивного пошуку за

індексом КАТОТТГ. Для мінімізації мережевого трафіку під час першого звернення завантажуються лише найбільш популярні населені пункти, а подальша взаємодія оптимізована за допомогою часової затримки запитів.

Стилізація застосунку побудована за принципом відмови від сторонніх CSS-фреймворків на користь власної системи дизайн-токенів, що дозволило мінімізувати обсяг фінальної збірки та забезпечити повний контроль над візуальною складовою. У глобальному конфігураційному файлі визначено базові змінні (CSS Custom Properties), які регламентують колірну палітру, параметри фонових шарів, акцентні елементи, радіуси закруглень та конфігурацію тіней. Такий підхід гарантує візуальну консистентність інтерфейсу в межах усієї системи.

Для структурування компонентів використано систему утилітарних класів, що визначають параметри контейнерів, секцій та базових елементів макета. При цьому стилі окремих модулів ізольовані за допомогою механізму інкапсуляції стилів (scoped styling), що унеможливлює виникнення конфліктів іменування у глобальному просторі назв. Типографічна система базується на шрифті Inter, який забезпечує високу читабельність медичної та технічної інформації на екранах із різною щільністю пікселів.

Адаптивність інтерфейсу для коректного функціонування на мобільних пристроях забезпечена через використання гнучких сіток (Flexbox та CSS Grid) у поєднанні з інструментарієм медіа-запитів (media queries). Це дозволяє динамічно трансформувати елементи керування залежно від параметрів перегляду: зокрема, реалізовано адаптацію навігаційних панелей під сенсорне введення, коригування кеглів шрифтів та оптимізацію зон взаємодії. Такий підхід гарантує збереження повної функціональності та цілісності користувацького досвіду незалежно від діагоналі екрана чи апаратних характеристик пристрою, як зображено на рис. 3.5.

Розроблений клієнтський SPA-застосунок FaPharm повною мірою реалізує функціональні вимоги, встановлені на етапі проєктування.

Впроваджена архітектура забезпечує функціонування інтерактивного каталогу з фасетною фільтрацією, інтеграцію геолокаційних сервісів для вибору міста, повний цикл обробки замовлень та персоналізацію через особистий кабінет. Використання сучасних методів оптимізації, зокрема стратегії оптимістичних оновлень, розділення коду та багаторівневого кешування на базі Pinia, дозволило досягти високої швидкодії інтерфейсу та стабільної роботи системи навіть за умов низької пропускної здатності мережевого з'єднання.

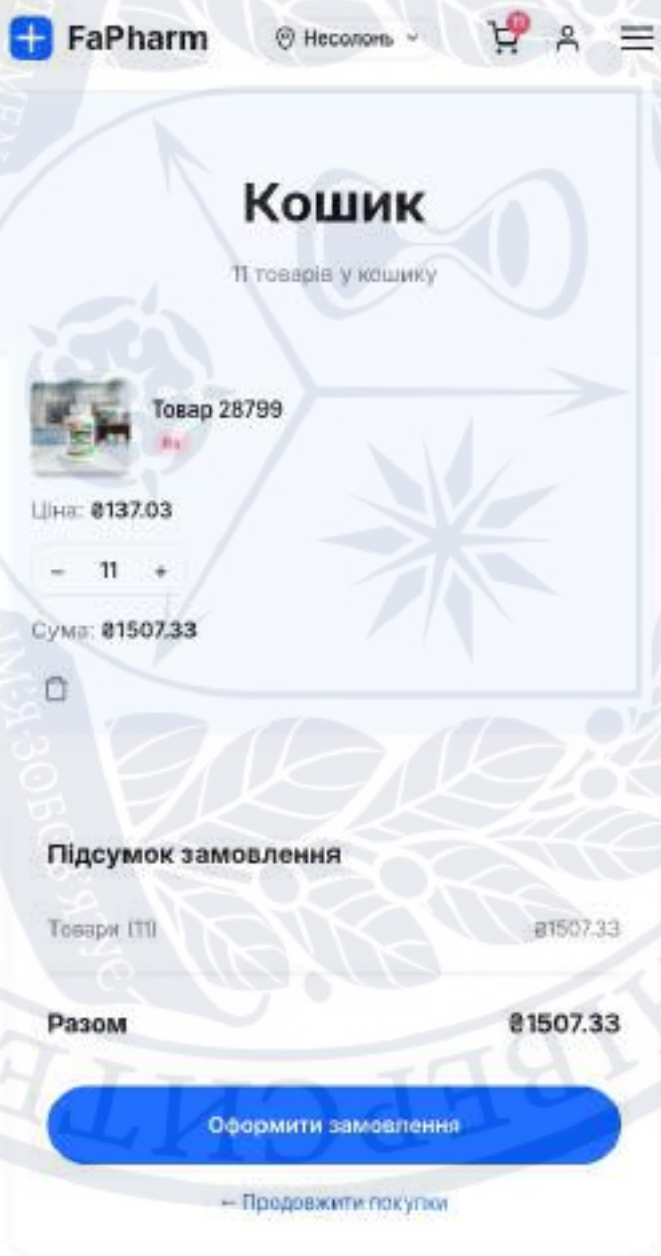

Рисунок 3.5 – Мобільна версія сторінки кошика застосунку FaPharm

3.4. Тестування продуктивності застосунку

Тестування є невід'ємним етапом розробки будь-якої інформаційної системи, що забезпечує відповідність реалізованого програмного продукту встановленим нефункціональним вимогам. До системи FaPharm висунуто вимоги щодо часу відповіді на пошукові запити не більше 200 мс, а також здатності обслуговувати пікове навантаження без суттєвого погіршення показників продуктивності. Для верифікації відповідності системи зазначеним вимогам було проведено комплексне тестування, що охоплює два ключових аспекти: швидкодію пошукових запитів та стабільність системи під навантаженням.

Для автоматизації процесу вимірювання показників швидкодії було розроблено спеціалізований сценарій на мові програмування Python, ключову функцію запуску сценаріїв якого наведено у Додатку А. Тестування проводилося серіями по 100 ітерацій для кожного сценарію з попереднім «прогрівом» системи (виконуючи 5 попередніх запитів) для виключення впливу «холодного старту» кешу. Для аналізу використовувалися не лише середні значення, а й медіана, а також 95-й та 99-й перцентилі, що дозволяє оцінити стабільність системи при граничних затримках.

Було обрано три ключові сценарії, які відображають різні типи навантаження на пошуковий рушій MeiliSearch:

- простий повнотекстовий пошук (базовий запит за назвою);
- фасетний пошук з одним фільтром (класична фільтрація за ознакою);
- комбінований пошук за алгоритмом $N+1$ (складний сценарій з трьома фільтрами, де застосовується стратегія паралельних запитів).

Комплексний аналіз результатів тестування підтверджує повну відповідність розробленої системи встановленим критеріям продуктивності. Отримані дані свідчать, що медіанний час відповіді для всіх сценаріїв не перевищує цільового порогу у 200 мс. Найкращі показники швидкодії зафіксовані під час виконання простих повнотекстових запитів, що вказує на

високу ефективність індексації MeiliSearch. Незважаючи на те, що одиночна фільтрація продемонструвала найбільшу затримку, впровадження алгоритму $N+1$ паралельних запитів для комбінованого пошуку дозволило суттєво скоротити час очікування, підтверджуючи ефективність стратегії паралелізації на рівні API. Результати вимірювань наведено у таблиці 3.1.

Таблиця 3.1 Результати вимірювання часу відповіді пошукових запитів

Сценарій запиту	Медіана (мс)	P95 (мс)	P99 (мс)
Простий повнотекстовий пошук	15.2	17.6	18.8
Пошук з одним фасетним фільтром	59.8	65	65.8
Комбінований пошук (3 фасети)	71	74.2	78.6

Для оцінки якості реалізації клієнтської частини було проведено аудит продуктивності головної сторінки за допомогою інструменту Lighthouse. Результати вимірювання ключових метрик Web Vitals наведено у таблиці 3.2.

Таблиця 3.2 Показники продуктивності клієнтської частини

Метрика	Desktop	Mobile
First Contentful Paint (FCP)	0.5 сек	1.9 сек
Largest Contentful Paint (LCP)	0.9 сек	3.2 сек
Total Blocking Time (TBT)	10 мс	430 мс
Speed Index (SI)	0.6 сек	1.9 сек
Performance Score	99 / 100	81 / 100

Отримані дані свідчать, що завдяки механізмам розділення програмного коду та оптимізації збірки засобами Vite, система демонструє відмінні показники на десктопних пристроях, повністю вкладаючись у цільові ліміти. Певне відхилення показника LCP (3.2 сек) для мобільної версії від вимоги у 1.5 сек, встановленої у таблиці 1.2, зумовлене специфікою тестування Lighthouse, яке використовує штучне обмеження швидкості

процесора та мережі для імітації пристроїв середнього сегмента; при цьому реальний користувацький досвід на сучасних смартфонах є значно вищим.

У сукупності результати підтверджують відповідність системи FaPharm встановленим нефункціональним вимогам продуктивності в умовах цільового десктопного середовища та прийнятний рівень швидкодії для мобільних платформ.

Отже, програмна реалізація серверної частини системи FaPharm на базі NestJS втілила всі архітектурні рішення, обґрунтовані у другому розділі: stateless JWT-автентифікацію з підтримкою анонімних кошиків, денормалізований пошуковий індекс із динамічною конфігурацією фасетів на основі метаданих бази даних та алгоритм $N+1$ паралельних запитів у модулі пошуку.

Розроблений клієнтський SPA-застосунок на Vue 3 реалізує повний функціональний цикл взаємодії користувача із системою. Застосування механізму розділення програмного коду засобами Vite, багаторівневого кешування на основі Pinia-сховищ, стратегії оптимістичних оновлень та debounce-затримки пошукових запитів забезпечило високу швидкодію інтерфейсу та стабільну роботу системи за умов низької пропускної здатності мережевого з’єднання. Адаптивний інтерфейс на основі власної системи дизайн-токенів забезпечує коректне функціонування на пристроях із різними характеристиками екрана.

Проведене тестування продуктивності підтвердило відповідність системи встановленим нефункціональним вимогам. Отримані результати підтверджують ефективність обраної стратегії паралелізації запитів і доцільність використання спеціалізованого пошукового рушія MeiliSearch як альтернативи вбудованим засобам повнотекстового пошуку PostgreSQL.

ВИСНОВКИ

У кваліфікаційній роботі вирішено науково-практичне завдання проєктування та програмної реалізації інформаційної системи пошуку й бронювання фармацевтичних препаратів із застосуванням високопродуктивних алгоритмів фільтрації даних.

Проведений аналіз стану ринку e-pharmacy засвідчив стійку висхідну динаміку галузі як у глобальному, так і в українському вимірі. Дослідження бізнес-процесів пошуку та бронювання препаратів дозволило сформулювати ключові функціональні та нефункціональні вимоги до системи. Порівняльний аналіз шести платформ-аналогів виявив технологічний розрив між вітчизняними сервісами та провідними зарубіжними платформами у частині фасетної фільтрації та обробки нечітких запитів, що підтвердило доцільність розробки нового рішення.

Обґрунтовано вибір технологічного стеку системи FaPharm: серверна частина реалізована на NestJS із реляційною СУБД PostgreSQL, пошукова підсистема побудована на рушії MeiliSearch, клієнтський застосунок – на Vue 3 із Composition API. Розроблено архітектуру вебсистеми із чітким розмежуванням між REST API та SPA-клієнтом, спроєктовано реляційну модель бази даних.

Розроблено та теоретично обґрунтовано алгоритм $N+1$ паралельних запитів для фасетної фільтрації, який забезпечує коректний підрахунок лічильників для кожного фасету. Застосування денормалізованого пошукового індексу з попередньо агрегованими статистиками залишків дозволяє виконувати геолокаційну фільтрацію без додаткових звернень до реляційної бази даних, що суттєво знижує латентність системи за умов великих обсягів каталогу.

Здійснено повноцінну програмну реалізацію серверної та клієнтської частин системи. Серверний REST API охоплює модулі автентифікації зі stateless JWT-підходом, управління каталогом товарів, пошукову підсистему

MeiliSearch із підтримкою нечіткого пошуку та кирилиці, модуль комерції, а також географічний довідник КАТОТТГ із визначенням найближчого населеного пункту за формулою Гаверсина. Клієнтський SPA-додаток на Vue 3 реалізує повний функціональний цикл взаємодії користувача із системою: від геолокаційного вибору міста та фасетного перегляду каталогу до оформлення замовлення з перевіркою наявності товарів у обраній аптеці.

Проведене тестування продуктивності підтвердило відповідність системи встановленим вимогам. Жоден із вимірюваних показників не перевищує цільового порогу у 200 мс, що підтверджує ефективність обраної стратегії паралелізації запитів та доцільність використання денормалізованого індексу як основного механізму пошуку.

Розроблена інформаційна система FaPharm повністю відповідає сформульованим функціональним та нефункціональним вимогам і може бути використана як прототип для впровадження в діяльність аптечних мереж або як основа для подальших наукових досліджень у галузі архітектури розподілених інформаційних систем фармацевтичного ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Grand View Research. Online Pharmacy Market Size, Share & Trends Analysis Report by Product, by Drug Type, by Region, and Segment Forecasts, 2023–2030. San Francisco : Grand View Research, 2023. URL: https://www.grandviewresearch.com/industry-analysis/epharmacies-market (дата звернення: 01.04.2026).

2. Mordor Intelligence. E-Pharmacy Market – Growth, Trends, COVID-19 Impact, and Forecasts (2026–2031). Hyderabad : Mordor Intelligence, 2024. URL: https://www.mordorintelligence.com/industry-reports/epharmacy-market (дата звернення: 02.04.2026).

3. Amazon Pharmacy. How Amazon Pharmacy works. URL: https://pharmacy.amazon.com (дата звернення: 03.04.2026).

4. Directive 2011/62/EU of the European Parliament and of the Council of 8 June 2011 amending Directive 2001/83/EC on the Community code relating to medicinal products for human use, as regards the prevention of the entry into the legal supply chain of falsified medicinal products. Official Journal of the European Union. 2011. L 174. P. 74–87.

5. Про лікарські засоби : Закон України від 04.04.1996 № 123/96-ВР. URL: https://zakon.rada.gov.ua/laws/show/123/96-вр (дата звернення: 04.04.2026).

6. Про затвердження Ліцензійних умов провадження господарської діяльності з роздрібної торгівлі лікарськими засобами : Постанова Кабінету Міністрів України від 30.11.2016 № 929. URL: https://zakon.rada.gov.ua/laws/show/929-2016-п (дата звернення: 05.04.2026).

7. Weske M. Business Process Management: Concepts, Languages, Architectures. 3rd ed. Berlin : Springer, 2019.

8. Chaffey D. Digital Business and E-Commerce Management. 7th ed. Harlow : Pearson Education, 2019.

9. Baeza-Yates R., Ribeiro-Neto B. Modern Information Retrieval: The Concepts and Technology behind Search. 2nd ed. New York : ACM Press, 2011.

10. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge : Cambridge University Press, 2008. URL: https://nlp.stanford.edu/IR-book (дата звернення: 06.04.2026).

11. Tunkelang D. Faceted Search. San Rafael : Morgan & Claypool Publishers, 2009.

12. Google Developers. RAIL Performance Model. URL: https://web.dev/articles/rail (дата звернення: 07.04.2026).

13. SimilarWeb. Website Traffic and Analytics: tabletki.ua. 2024. URL: https://www.similarweb.com/website/tabletki.ua (дата звернення: 08.04.2026).

14. Tabletki.ua. Офіційний сайт платформи пошуку та бронювання ліків. URL: https://tabletki.ua (дата звернення: 09.04.2026).

15. Apteka911.ua. Офіційний сайт мережі аптек та онлайн-платформи. URL: https://apteka911.ua (дата звернення: 10.04.2026).

16. Podorozhnyk.ua. Офіційний сайт мережі аптек «Подорожник». URL: https://podorozhnyk.ua (дата звернення: 11.04.2026).

17. DocMorris. Official website of the online pharmacy. URL: https://www.docmorris.de (дата звернення: 12.04.2026).

18. PharmEasy. Official website of the online pharmacy platform. URL: https://pharmeasy.in (дата звернення: 13.04.2026).

19. Amazon Pharmacy. Official website. URL: https://pharmacy.amazon.com (дата звернення: 14.04.2026).

20. Node.js Foundation. Node.js Official Documentation. URL: https://nodejs.org/en/docs (дата звернення: 15.04.2026).

21. Myśliwiec K. NestJS Documentation. URL: https://docs.nestjs.com (дата звернення: 16.04.2026).

22. Cherny B. Programming TypeScript. Sebastopol : O’Reilly Media, 2019. P. 21.

23. PostgreSQL Global Development Group. PostgreSQL 17 Documentation. URL: https://www.postgresql.org/docs/17 (дата звернення: 17.04.2026).

24. TypeORM. TypeORM Official Documentation. URL: https://typeorm.io (дата звернення: 18.04.2026).

25. MeiliSearch. MeiliSearch Official Documentation. URL: https://www.meilisearch.com/docs (дата звернення: 19.04.2026).

26. Elasticsearch B.V. Elasticsearch Reference Documentation. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current (дата звернення: 20.04.2026).

27. You E. et al. Vue 3 Official Documentation. URL: https://vuejs.org/guide (дата звернення: 21.04.2026).

28. You E. et al. Vite Official Documentation. URL: https://vitejs.dev/guide (дата звернення: 22.04.2026).

29. Posva E. Pinia Official Documentation. URL: https://pinia.vuejs.org (дата звернення: 23.04.2026).

30. Docker Inc. Docker Compose Documentation. URL: https://docs.docker.com/compose (дата звернення: 24.04.2026).

31. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Prentice Hall, 2017.

32. Myśliwiec K. NestJS Documentation: Modules. URL: https://docs.nestjs.com/modules (дата звернення: 25.04.2026).

33. КОАТУУ / КАТОТТГ 2021. Державний класифікатор об’єктів адміністративно-територіального устрою України. URL: https://dovidnyk.in.ua/directories (дата звернення: 26.04.2026).

34. Karwin B. SQL Antipatterns: Avoiding the Pitfalls of Database Programming. Raleigh : Pragmatic Bookshelf, 2010.

35. PostgreSQL Documentation. JSON Types. URL: https://www.postgresql.org/docs/17/datatype-json.html (дата звернення: 27.04.2026).

36. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. P. 236–242.

37. Jones M. et al. JSON Web Token (JWT). RFC 7519. IETF, 2015.

38. TypeORM. Migrations Documentation. URL: https://typeorm.io/migrations (дата звернення: 28.04.2026).

39. Tunkelang D. Faceted Search. San Rafael : Morgan & Claypool Publishers, 2009. P. 30.

40. Axios. API Reference. URL: https://axios.rest/pages/advanced/api-reference.html (дата звернення: 29.04.2026).

ДОДАТОК А

Функція запуску сценаріїв бенчмарку (Python)

```
def run_scenario(
    scenario: Scenario,
    iterations: int,
    warmup: int,
    delay_ms: int,
    headers: dict,
    timeout: int,
    verbose: bool = True,
) -> dict:
    queries = SEARCH_QUERIES
    errors = 0
    times: list[float] = []

    print(f"\n{'—' * 60}")
    print(f"Scenario {scenario.id}: {scenario.name}")
    print(f"  {scenario.description}")
    print(f"  Warmup: {warmup} requests | Measurement:
{iterations} requests")

    with requests.Session() as session:
        if warmup > 0:
            print(f"  Warming up...", end=" ", flush=True)
            for i in range(warmup):
                query = queries[i % len(queries)]
                run_single_request(session, scenario, query,
headers, timeout)
            print("done")

        print(f"  Measuring: ", end="", flush=True)
        for i in range(iterations):
            query = queries[i % len(queries)]
            elapsed_ms, status = run_single_request(session,
scenario, query, headers, timeout)

            if status not in (200, 201):
```

```
                errors += 1
                if verbose and status not in (0, -1):
                    print(f"\n    ⚠️  HTTP {status} on request
{i+1}", end="")
            else:
                times.append(elapsed_ms)

            if delay_ms > 0:
                time.sleep(delay_ms / 1000)

            if (i + 1) % 10 == 0:
                print(f"{i+1}", end=" ", flush=True)

        print()

    if not times:
        print("  ❌ No successful requests!")
        return {
            "scenario_id": scenario.id,
            "scenario_name": scenario.name,
            "iterations": iterations,
            "successful": 0,
            "errors": errors,
            "median_ms": None,
            "mean_ms": None,
            "min_ms": None,
            "max_ms": None,
            "p95_ms": None,
            "p99_ms": None,
            "stdev_ms": None,
        }

    median_ms = statistics.median(times)
    mean_ms   = statistics.mean(times)
    min_ms    = min(times)
    max_ms    = max(times)
    p95_ms    = percentile(times, 95)
```

```
    p99_ms    = percentile(times, 99)
    stdev_ms  = statistics.stdev(times) if len(times) > 1 else
0.0

    target_ms = 200
    meets_target = "✅" if median_ms <= target_ms else "❌"

    print(f"\n  Results (successful: {len(times)}/{iterations},
errors: {errors}):")
    print(f"  {'Median':>10}: {median_ms:>8.1f} ms
{meets_target} (target: ≤{target_ms} ms)")
    print(f"  {'Mean':>10}: {mean_ms:>8.1f} ms")
    print(f"  {'Min / Max':>10}: {min_ms:>8.1f} / {max_ms:.1f}
ms")
    print(f"  {'P95':>10}: {p95_ms:>8.1f} ms")
    print(f"  {'P99':>10}: {p99_ms:>8.1f} ms")
    print(f"  {'σ (stdev)':>10}: {stdev_ms:>8.1f} ms")

    return {
        "scenario_id":    scenario.id,
        "scenario_name":  scenario.name,
        "iterations":     iterations,
        "successful":     len(times),
        "errors":         errors,
        "median_ms":      round(median_ms, 2),
        "mean_ms":        round(mean_ms, 2),
        "min_ms":         round(min_ms, 2),
        "max_ms":         round(max_ms, 2),
        "p95_ms":         round(p95_ms, 2),
        "p99_ms":         round(p99_ms, 2),
        "stdev_ms":       round(stdev_ms, 2),
    }
```