МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: *Програмний інтерфейс формування каталогу медичних препаратів*

Здобувача групи ІТ.м-24 *Сірик Микола Олександрович*
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Микола СІРИК
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник в.о. завідувача кафедри, ктн., доц. Світлана ВАЩЕНКО _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу магістра студентові
Сірик Микола Олександрович
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи *Програмний інтерфейс формування каталогу медичних препаратів*

затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI
2 Термін здачі студентом кваліфікаційної роботи «_18_» ___грудня___ 2023 р.
3 Вхідні дані до кваліфікаційної роботи *каталог медичних препаратів, технічне завдання на розробку*

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *аналіз предметної області, постановка задачі та методи дослідження, моделювання структури програмного інтерфейсу, конфігурація та налаштування засобів віртуалізації, розробка програмного інтерфейсу інформаційної системи каталогізації медичних препаратів*

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) *актуальність роботи, постановка задачі, огляд і аналіз інформаційних каталогів*, *контекстна діаграма нотації IDEF0, діаграма варіантів використання, діаграма змодельованої бази даних, діаграма моделей системи, демонстрація роботи API-Інтерфейсу*

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	02.10.23 - 16.10.23	Вик.
2	Збір вимог	20.10.23 - 24.10.23	Вик.
3	Вибір інструментів для роботи	25.10.23 - 27.10.23	Вик.
4	Створення WBS та OBS	27.10.23 - 31.10.23	Вик.
5	Складання календарного плану	31.10.23 - 03.11.23	Вик.
6	Визначення ризиків	03.11.23 - 06.11.23	Вик.
7	Дизайн бази даних	06.11.23 - 08.11.23	Вик.
8	Налаштування панелі адміністратора та авторизації	09.11.23 - 14.11.23	Вик.
9	Створення API користувача	15.11.23 - 16.11.23	Вик.
10	Створення API медичних препаратів	17.11.23 - 20.11.23	Вик.
11	Створення API тегів та складників	21.11.23 - 22.11.23	Вик.
12	Створення API зображення медичного препарату	23.11.23 - 24.11.23	Вик.
13	Впровадження фільтрації	27.11.23 - 27.11.23	Вик.
14	Контейнеризація складових програмного інтерфейсу	30.11.23 - 04.12.23	Вик.
15	Тестування інтерфейсу	28.11.23 - 07.12.23	Вик.
16	Підготовка до завантаження	11.12.23 - 12.12.23	Вик.

Магістрант _____ Микола СІРИК

Керівник роботи _____ к.т.н., доц. Світлана ВАЩЕНКО

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Програмний інтерфейс формування каталогу медичних препаратів».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 42 найменувань, додатків. Загальний обсяг роботи − 92 сторінки, у тому числі 72 сторінки основного тексту, 5 сторінок списку використаних джерел, 10 сторінок додатків.

Актуальність роботи полягає в необхідності автоматизації процесів систематизації та класифікації великої кількості медичних препаратів, що надходять на ринок. На сьогоднішній день існують рішення для підтримки процесу каталогізації медичних препаратів, але вони часто обмежені функціональністю та складні у користуванні. Розробка програмного інтерфейсу для формування каталогу медичних препаратів є кроком у напрямку створення зручного й дієвого інструменту для організації даних про медичні засоби та управління ними, з можливістю подальшої інтеграції завдяки відкритим API.

Мета роботи: розробка прикладного програмного інтерфейсу, що дозволить ефективно каталогізувати й класифікувати медичні препарати для комфортного використання фахівцями при розробці систем медичного призначення. Етапи розробки включають аналіз предметної області, проектування та моделювання програмного продукту, розробку бази даних, імплементацію інтерфейсної частини та тестування готового продукту.

Запропонований програмний інтерфейс каталогізації дозволяє спростити роботу розробників по реалізації інформаційних технологій медичного спрямування в частині спрощення організації доступу до інформації. Крім цього, інтерфейс дозволяє додавати теги і фільтри, що забезпечує уніфікацію каталогу і його адаптацію для конкретних завдань.

Ключові слова: API, МЕДИКАМЕНТ, ПРЕПАРАТ, КАТАЛОГ, КІНЦЕВА ТОЧКА, DJANGO, DOCKER.

ЗМІСТ

ВСТУП

Значний поступ у розвитку сучасних технологій надав можливість значно поліпшити управління даними та доступ до них у різних галузях. При розробці інформаційних систем у медичній сфері, де будь-яка інформація щодо препаратів має велике, а іноді навіть критичне значення, використання програмних інтерфейсів для формування каталогу медикаментів стає необхідністю. Це дозволить підвищити ефективність процесів організації й систематизації відомостей про різноманітні медичні засоби для полегшення роботи фахівців у даній сфері та покращення доступу пацієнтів до необхідної інформації.

Актуальність. На ринок перманентно надходить велика кількість медичних препаратів, тому виникає необхідність їх систематизації та класифікації для зручного використання фахівцями. На сьогоднішній день існують деякі рішення для підтримки процесу каталогізації медичних препаратів, але вони часто обмежені функціональністю й складні у користуванні. Розробка програмного інтерфейсу для формування каталогу медичних препаратів є кроком у напрямку створення зручного й дієвого інструменту для організації даних про медичні засоби та управління ними.

Об'єкт дослідження. Процес каталогізації медичних препаратів за допомогою програмного інтерфейсу, його функціональність та здатність оптимізувати доступ до інформації про медичні засоби.

Предмет дослідження. Інформаційна система каталогізації медичних препаратів, яка включає в себе програмний інтерфейс, базу даних та пов'язані з ними компоненти.

Метою даної роботи є розробка прикладного програмного інтерфейсу, що дозволить ефективно каталогізувати й класифікувати медичні препарати для комфортного використання фахівцями при розробці web-орієнтованих систем медичного призначення.

Для досягнення цієї мети поставлені такі завдання:

– аналіз предметної області з метою визначення основних вимог до процесу

каталогізації медичних препаратів;

- проектування та моделювання програмного продукту з використанням сучасних підходів та стандартів;
- розробка бази даних для зберігання та управління інформацією про медичні засоби;
- реалізація інтерфейсної частини та структури програмного продукту з використанням технологій Django, PostgreSQL та Docker;
- тестування готового програмного продукту з метою визначення його ефективності та надійності.

Практичне значення. Виконана робота вирішує актуальну проблему організації інформації про медичні препарати, надаючи фахівцям зручний та доступний інструмент для розробки персоналізованих каталогів медичних засобів.

Розроблений програмний інтерфейс має велике практичне значення. Завдяки розбудованим внутрішнім зв’язкам, що наявні у інтерфейсі, суттєво спрощено організацію доступ до інформації про різноманітні медичні препарати, їх властивості, взаємодію та застосування. Крім того, він дозволяє додати теги і фільтри, що забезпечує уніфікацію каталогу і модифікування його для певних задач. Відкритий API дозволяє у подальшому інтегрувати каталог до інших інформаційних систем, котрі застосовує користувач.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Невпинний розвиток інформаційних технологій та постійне збільшення кількості інформації спричиняє потребу у використанні ефективної каталогізації даних для спрощення їх обробки та полегшення використання людиною.

Фармацевтичний ринок також продовжує зростати, щорічно збільшуючи кількість номенклатури лікарських засобів на десятки одиниць нових патентів щорічно [1]. Фармацевтична галузь України, в свою чергу, зберігає стійку тенденцію до розширення асортименту лікарських засобів [2]. Наприклад, лише кількість номенклатури засобів для лікування захворювань порожнини носа становить більше двох сотень [3], при чому кожен із них може мати різні додаткові інгредієнти (окрім однакової чи схожої діючої речовини). Складність формування каталогів препаратів описано у ґрунтовному дослідженні [4], котре потребувало значних людських ресурсів того часу. Тому через необхідність якісного поточного контролю стану запасів різнопланових номенклатур товарів, розробок, а тим паче їх складників, робота великих підприємств, холдингів і навіть аптек зараз передбачає використання автоматизованих систем обліку медичних препаратів. Завдяки їх використанню досягається економія коштів завдяки значному скороченню часу обробки рецептурної інформації [5].

Особливо це актуально для українського ринку фармакології, адже це полегшуватиме співпрацю українських виробників продукції з виробниками інших країн ЄС. Це є кроком на шляху євроінтеграції та розвитку і зможе підтримати ринок навіть при прогнозованому скороченні кількості фармацевтичних підприємств України [6].

Зараз пілотні проекти автоматизованого контролю і видачі із врахуванням дозувань рецептурних препаратів доводять ефективність застосування систем обліку [7]. Саме системи обліку містять основні дані для формування пакету видачі – запаси, склад і назву препарату.

У статті [8] досліджено ефективність використання електронних інформаційних

систем закупівлі ліків у Індонезії із специфічними рецептурними складниками (наркотичні препарати). А стаття [9] уже на прикладі фармацевтичної компанії у Йорданії показує значну роль автоматизованих систем обліку та видачі у повсякденній роботі корпорації. Для впровадження подібних систем є необхідним застосування каталогізації як на рівні держави, так і на рівні аптек.

Рішення для каталогізації розробляються і змінюють повсякчас. Наприклад, у дослідженні [10] описано використання каталогізації за допомогою Visual Basic, а тепер уже розробляються комплексні рішення електронних госпіталів і аптек на основі штучного інтелекту [11]. Останні поки є досить складними [12], а тому стикаються зі значними витратами на впровадження. Саме тому формування стандартизованої каталогізації на нижніх рівнях пов’язаних систем (як от склад корпусу госпіталю) із використанням додатку з попередньо сформованою структурою зможе здешевити і спростити впровадження. Стаття [13] також вказує, що для впровадження інтелектуальних систем управління у фармацевтиці із використанням машинного навчання, необхідним є наявність великої кількості даних, в якості яких може виступати стандартизований каталог препаратів. Навіть при впровадженні систем машинного розпізнавання препаратів як запропоновано у [14] необхідним є каталогізований результат із ключовими характеристиками.

У фармацевтичній і медичній галузях основною системою автоматизації основних процесів є медична інформаційна система (МІС) [15]. Однією із ключових складових МІС є система управління переліком медичних препаратів або, інакше кажучи, фармацевтичний каталог. Ця складова грає важливу роль у забезпеченні ефективного функціонування медичної установи.

У контексті фармацевтичного каталогу МІС, важливо визначити, як система забезпечує актуальність інформації про медичні препарати. Перевага фармацевтичного каталогу в МІС полягає в тому, що вона дозволяє не лише зберігати дані про лікарські засоби, а й забезпечує їхню постійну актуалізацію та доступність. Однією з ключових властивостей фармацевтичного каталогу є його здатність структурувати інформацію і реагувати на зміни в асортименті медичних препаратів, забезпечуючи можливість поновлення інформації в реальному часі. Це надає

медичним фахівцям надійний інструмент для прийняття обґрунтованих рішень у процесі надання медичної допомоги.

Наявність індивідуалізованого каталогу дозволяє оперувати лише із необхідним або дозволеним переліком препаратів, вносити додаткову інформацію, а також забезпечує безперешкодний доступ до переліку та незалежність від зовнішніх систем чи переліків, приклади яких наведені далі.

Для прикладу, інформаційна система «Медична бібліотека Копендіум» [16] є оцифрованою версією довідника [17], що містить узагальнена та систематизовану інформацію про вітчизняні і зарубіжні лікарські препарати та їхні лікарські форми. Також для кожного препарату тут наведено опис ряду діючих речовин разом із додатковою інформацією (склад, форма випуску, фармакологічна група тощо) [16].

Особливістю каталогу є моніторинг цін і наявності препаратів у різних аптеках України. Крім того, одразу наведено кілька стандартних згрупувань, що дозволяють перейти уже до прямого переліку карток препаратів (рис. 1.1).

Тобто, забезпечена можливість скористатися не лише алфавітним покажчиком із повним переліком препаратів, а також отримати групи записів по виробникам чи за АТС-класифікацією.

Для картки кожного препарату створено окрему веб-сторінку, що містить основну і розширену інформацію та відповідне зображення елемента каталогу (рис. 1.2).

Безперечно, плюсом є універсальність і наукова структурованість каталогу. Проте, такий тип каталогу можна використовувати лише в якості довідника, адже він не забезпечує можливостей носити зміни, додавати теги чи індивідуальну інформацію. Крім того, доступ до каталогу можливий лише у режимі перегляду веб-сайту, адже він не містить відкритих API для можливості інтеграції в персональні системи.

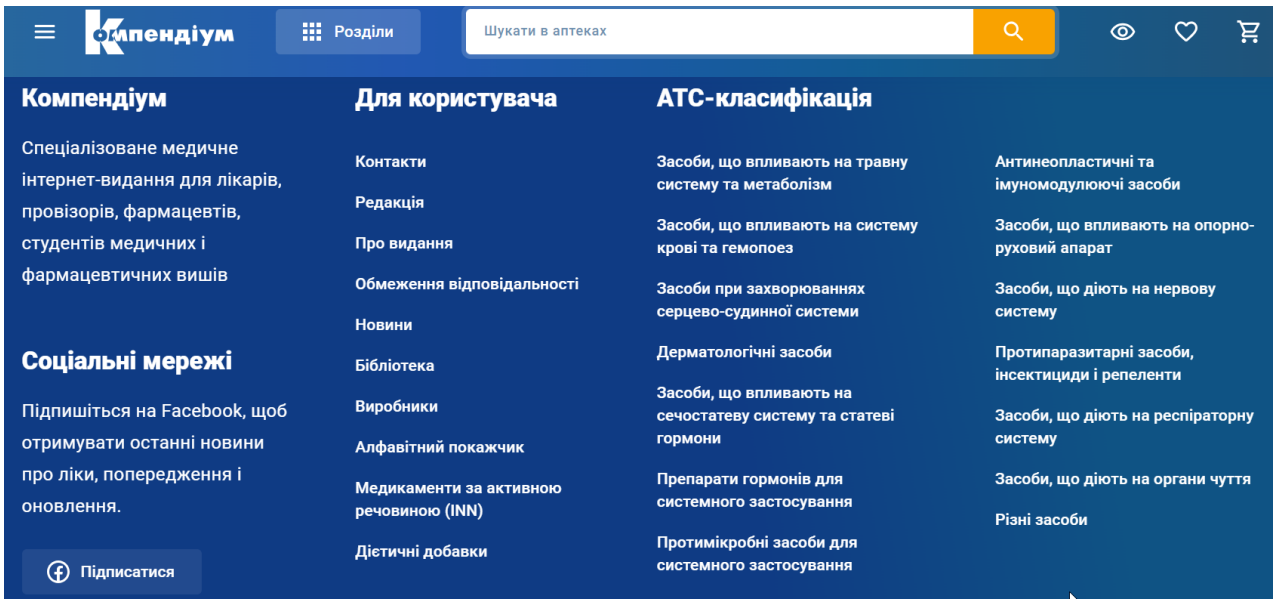


Рисунок 1.1 – Класифікації каталогу Копендіум

Джерело:[16]

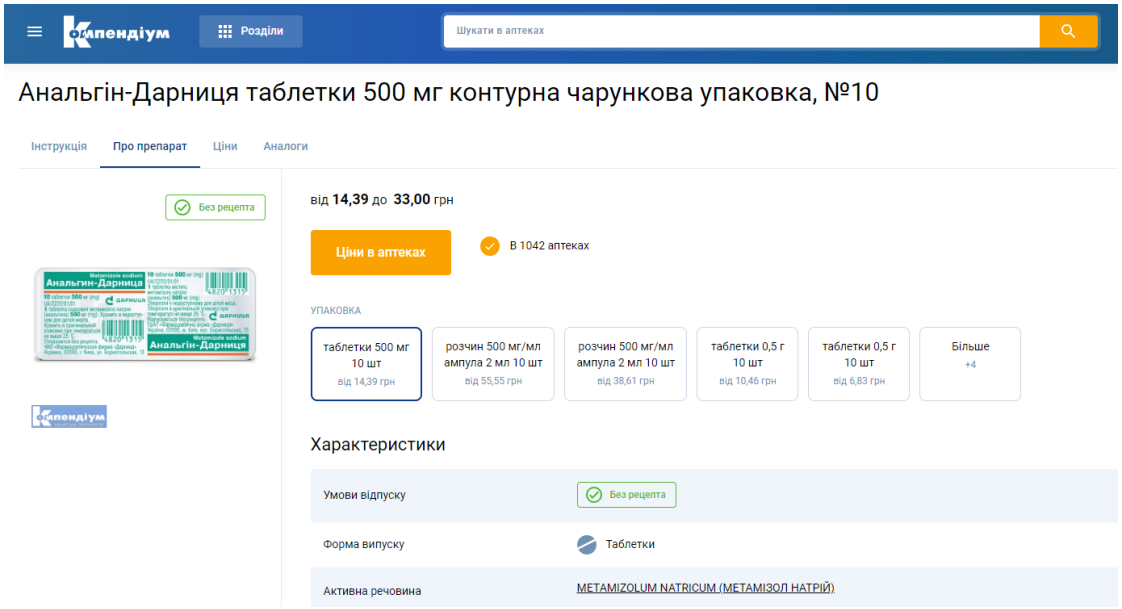


Рисунок 1.2 – Картка елементу каталогу Копендіум

Джерело:[16]

Державний ресурс «Державний реєстр лікарських засобів України» [18] містить повний перелік ліцензованих лікарських препаратів, але дозволяє лише здійснювати розширений пошук по ним (рис. 1.3). Тобто, реєстр не містить ні повного переліку препаратів на окремій сторінці, ні згрупованих за спільними елементами карток елементів каталогу.

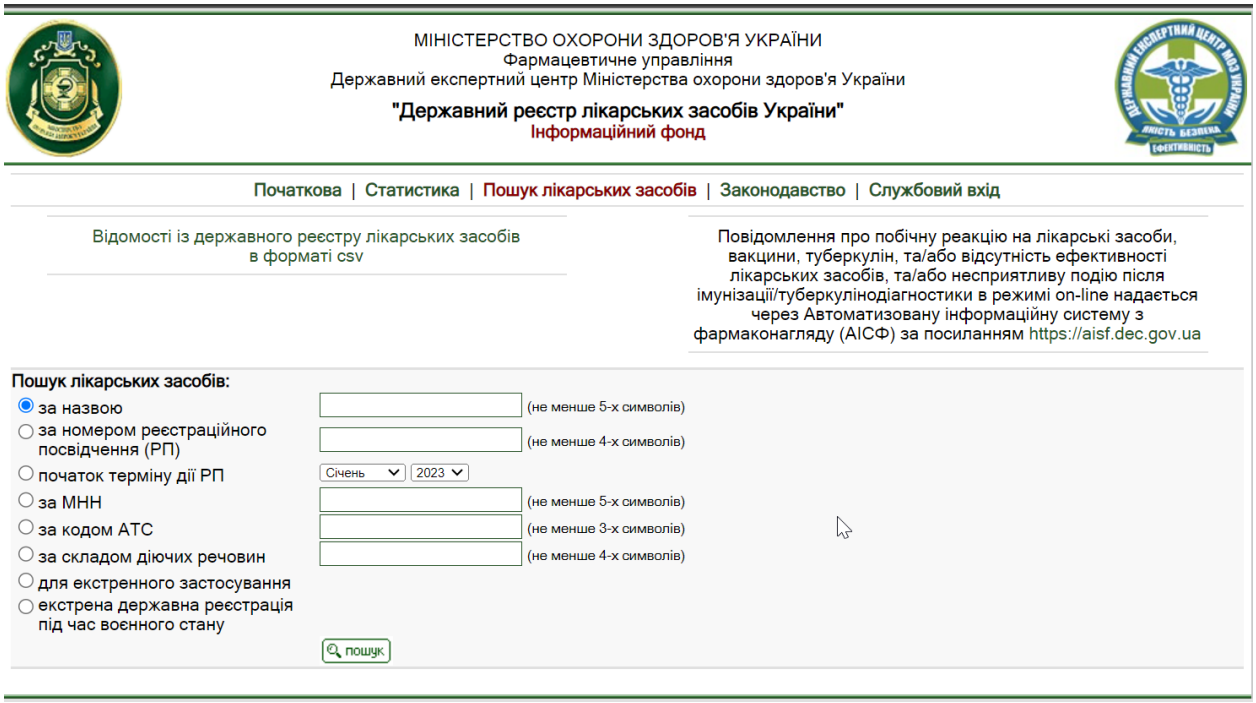


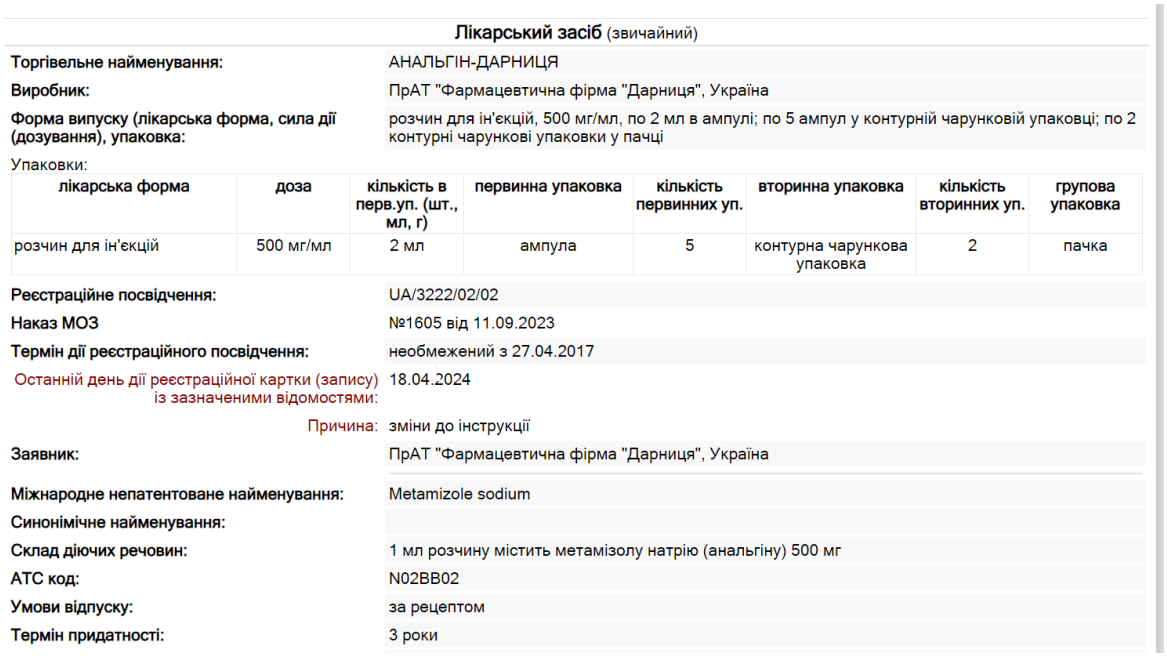
Рисунок 1.3 – Пошук по державному реєстру лікарських засобів

Джерело:[18]

Кожен картка препарату містить розширену інформацію, включно із ліцензійною, проте не містить інструкції (рис. 1.4). Перевагою є можливість експортувати перелік у вигляді csv-файлу, що дозволяє використати цей перелік у якості лише вхідних даних для МІС. Але відсутність API не дозволяє інтегрувати інформацію напряму в комерційні системи.

Враховуючи наявність відкритих каталогів у виглядах веб-сайтів, доцільним є створення системи для можливості переформатувати наявну інформацію, ефективно каталогізувати й класифікувати медичні препарати, а також індивідуально структурувати їх у власній базі даних кожного користувача. За рахунок реалізації доступу до такої системи через документовані API підвищується її доступність і інтеграційна сумісність. Адже система може бути використана в якості бекенд-частини з подальшою інтеграцією у МІС чи додаток в якості основних робочих даних користувача, підприємства чи навіть виробництва, шляхом використання завчасно відомих методів автентифікації та структур запитів і відповідей.

Порівняння розглянутих каталогів наведено у таблиці 1.1.



Лікарський засіб (звичайний)

Торгівельне найменування:	АНАЛЬГІН-ДАРНИЦЯ
Виробник:	ПрАТ "Фармацевтична фірма "Дарниця", Україна
Форма випуску (лікарська форма, сила дії (дозування), упаковка:	розчин для ін'єкцій, 500 мг/мл, по 2 мл в ампулі; по 5 ампул у контурній чарунковій упаковці; по 2 контурні чарункові упаковки у пачці

Упаковки:

лікарська форма	доза	кількість в перв.уп. (шт., мл, г)	первинна упаковка	кількість первинних уп.	вторинна упаковка	кількість вторинних уп.	групова упаковка
розчин для ін'єкцій	500 мг/мл	2 мл	ампула	5	контурна чарункова упаковка	2	пачка

Реєстраційне посвідчення:	UA/3222/02/02
Наказ МОЗ	№1605 від 11.09.2023
Термін дії реєстраційного посвідчення:	необмежений з 27.04.2017
Останній день дії реєстраційної картки (запису) із зазначеними відомостями:	18.04.2024
Причина:	зміни до інструкції
Заявник:	ПрАТ "Фармацевтична фірма "Дарниця", Україна
Міжнародне непатентоване найменування:	Metamizole sodium
Синонімічне найменування:	
Склад діючих речовин:	1 мл розчину містить метамізолу натрію (анальгіну) 500 мг
АТС код:	N02BB02
Умови відпуску:	за рецептом
Термін придатності:	3 роки

Рисунок 1.4. Картка елементу державного реєстру лікарських засобів

Джерело:[18]

Таблиця 1.1 – Порівняння розглянутих проєктів

Критерій	Копендіум	Державний реєстр лікарських засобів України
Інформація про препарат, діючі речовини	Так	Так
Наявність фото препарату	Так	Ні
Групування за критеріями, тегами чи класифікаціями	Так	Ні
Індивідуалізований доступ	Так	Ні
Забезпечення змін і тегів	Ні	Ні
Пошук препаратів	Так	Так
Наявність інструкції	Так	Ні
Отримання повного переліку препаратів	Так	Ні
Доступ через API	Ні	Ні
Відкритість для інтеграції	Ні	Ні
Можливість експорту даних	Ні	Так

Джерело: побудовано автором на основі даних з[16-18]

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою кваліфікаційної роботи є розробка прикладного програмного інтерфейсу каталогізації медичних препаратів.

Для досягнення мети даного проекту необхідно виконати наступні задачі:

- за результатами проведеного в попередньому розділі роботи аналізу сформувати повний перелік функціональних вимог, виконання яких має забезпечуватися програмним інтерфейсом, що розробляється;
- визначити стек технологій, що є найбільш розповсюдженими для виконання вказаного типу систем;
- виконати моделювання та реалізувати структуру системи;
- програмно реалізувати систему каталогізації медичних препаратів;
- здійснити smoke-тестування створеного продукту.

Вказаний програмний інтерфейс дозволить для компаній, дотичних до фармацевтичної галузі, створювати індивідуальні каталоги із персоналізованими тегуваннями та різними типами доступу. Використовуючи вказаний інтерфейс розробники матимуть змогу оперувати власною базою даних продукції, що дозволить зменшити залежність від зовнішніх каталогів, спростити доступність до переліків IT-спеціалістам суміжних систем та підвищити зручність у використанні завдяки можливості змінювати існуючу та додавати необхідну інформацію.

Розроблений інтерфейс має задовольняти такі вимоги до системи.

2. Функції управління:
   - створення картки медичного препарату із ім’ям, основною інформацією (показаннями), діючою речовиною, складом, ціною, формою відпуску та зображенням;
   - оновлення інформації про медичні препарати;
   - можливість додавання тегів до медичного препарату;

- отримання списку препаратів;
- пошук по назві чи діючій речовині;
- видалення препарату із каталогу.

3. Аутентифікація користувачів:
   - можливість реєстрації нових користувачів;
   - автентифікація користувачів за допомогою облікових даних, таких як ім'я користувача та пароль;
   - забезпечити автентифікацію на основі токенів авторизації за для автентифікації наступних запитів до API;
   - реалізація механізмів автентифікації користувачів через для забезпечення доступу до кінцевих точок API.
4. Продуктивність:
   - підтримка одночасних запитів різних користувачів по API без погіршення продуктивності;
   - обробка великого обсягу даних.
5. Безпека:
   - забезпечення використання захищеного протоколу зв'язку HTTPS для ефективного шифрування даних під час їх передачі між клієнтом та сервером;
   - використання сучасні методи автентифікації, зокрема, систему токенів, щоб забезпечити, що лише авторизовані користувачі отримують доступ до захищених ресурсів системи;
   - використання ефективних схем перевірки вхідних даних та санітації, що включають в себе заходи для виявлення та запобігання загрозам безпеки, таким як SQL-ін'єкції.
6. Масштабованість:
   - система повинна забезпечувати можливість легкого масштабування та збільшення об’ємів обробки інформації;
   - API повинно бути легко розширюваним для взаємодії з іншими

медичними системами та інструментами.

7. Обробка помилок та документація:
   - забезпечення генерації змістовних повідомлень та встановлення відповідних кодів статусу HTTP у випадку отримання недійсних запитів або внаслідок системних збоїв;
   - надання докладної та доступної документації до API для спрощення інтеграції та розвитку.

Таким чином майбутня інформаційна система повинна мати правильно спроектовані REST API, адже основною задачею є оптимізація роботи і забезпечення майбутньої інтеграції у інші системи.

Планування робіт про виконанню проекту наведено у додатку А.

2.2 Методи дослідження

Після визначення предметної області роботи і формування базової ідеї реалізації програмного продукту було оглянуто і обрано засоби досягнення поставленої задачі. Для їх пошуку було застосовано кілька методів наукового дослідження [19].

Аналіз літератури. Був використаний для ознайомлення з результатами попередніх досліджень та публікацій у області каталогізації із прив’язкою до медичної галузі. За допомогою нього було визначено та уточнено теоретичні аспекти проекту та враховано сучасні тенденції у IT-галузі.

Теоретичний метод. Цей метод застосовується для огляду і аналізу особливостей та можливостей предметної області. Це дозволяє правильно обрати інструменти та технології для розробки, враховуючи їхні переваги і недоліки при також для побудові шляхів реалізації бізнес-процесів, на основі їх потреб із застосуванням сучасних IT тенденцій.

Системно-функціональний метод. Метод дослідження складових системи, що дозволяє розглянути роботу компонентів окремо, а також її взаємодію в рамках єдиного цілого. Цей метод було використано при формуванні базової архітектури проекту та при проектуванні класів програмного модулю, а також для побудови UML-діаграми взаємодії [20].

Метод моделювання. Включає розробку схем та діаграм, що показують процес роботи над реалізацією програмного продукту та його подальше використання. Використовувався на етапі побудови діаграм IDEF0 [21] та UseCase [22] в контексті реалізації проекту каталогізатора медичних препаратів.

2.3 Технології реалізації

Для реалізації проекту в якості мови програмування було обрано Python через його простоту, читабельність та широкий спектр застосувань, зокрема для веб-розробки та аналізу даних. Основним фреймворком для розробки API буде використано Django і Django Rest Framework (DRF). Django обраний за його зручність у розробці та стійкість, а DRF – через його зрозумілість, адаптивність до швидкої розробки, гнучкості та безпеці. В якості системи управління даними застосовано PostgreSQL завдяки здатності обробляти складні типи даних та забезпечувати надійність та цілісність даних. Для забезпечення ізольованого середовища та легкості розгортання, що полегшує роботу та забезпечує консистентність між середовищами розробки та продакшну проект контейнеризовано за допомогою Docker.

Мова програмування Python

Python [23, 24] була обрана основною мовою для створення REST API. Це інтерпретована мова програмування високого рівня, що зараз все більше користується попитом [25]. Завдяки наявності великої кількості бібліотек та відносній простоті синтаксису Python використовують для різноманітних проектів як у сфері веб-розробки, так і аналізу даних, наукових обчислень і штучного інтелекту [26].

Екосистема фреймворків та інструментів надає розробникам необхідні ресурси для ефективного створення та підтримки API.

До основних переваг Python відносять [27]:

- простота вивчення. Python має зрозумілий і чистий синтаксис, що робить його відносно простим у вивченні.
- велика спільнота. Щоденне зростання кількості застосувань цієї мови програмування, як наслідок, створює активну спільноту користувачів.
- універсальність. Завдяки активній спільноті, Python пропонує розгалужену колекцію бібліотек і фреймворків.

Проте Python також має і недоліки:

- швидкодія. У порівнянні із деякими інтерпретованими та більшістю компільованих мов програмування Python презентує повільнішу роботу.
- глобальне блокування інтерпретатора (GIL) [28]. Через цю особливість у Python може бути обмежена ефективність використання багатопоточності, що потенційно може впливати на продуктивність під час виконання багатопоточних операцій [27].

Фреймворк для імплементації Django

Django [29] - це високорівневий веб-фреймворк, розроблений на мові програмування Python. Відзначається високою продуктивністю та швидкістю розробки і надає інструменти для створення складних веб-додатків з ефективним та безпечним управлінням базами даних.

Основні характеристики та переваги Django [30]:

- Модель-Вид-Контролер (MVC). Модель відповідає за базу даних, Вид - за відображення та взаємодію з користувачем, а Контролер - за обробку логіки додатку.
- ORM (Об'єктно-реляційна модель). Django забезпечує абстракцію бази даних через ORM, чим забезпечує використання за допомогою об'єктів Python замість SQL-запитів.
- Адміністративна панель. Django надає готову адміністративну панель, яка

автоматично створюється на основі моделей додатку.

- Захист від CSRF-атак. Django автоматично включає захист від атаки Cross-Site Request Forgery (CSRF), забезпечуючи безпеку веб-додатків.
- Легкість налаштування URL і API. Маршрутизація URL в Django є чіткою та гнучкою, дозволяючи легко визначати шляхи до різних частин додатку.

Система управління базами даних для реалізації PostgreSQL

PostgreSQL [31] – це об'єктно-реляційна система управління базами даних (ORDBMS), яка володіє великим набором можливостей та високою надійністю. Вона є потужною та відкритою системою управління базами даних, розробленою для забезпечення ефективного зберігання та опрацювання структурованої інформації. Це вільна та спільнотно розроблювана база даних, яка надає розширені можливості для роботи з реляційними даними.

До переваг PostgreSQL можна віднести [32]:

- Розширені функціональні можливості. PostgreSQL має підтримку широкого спектру типів даних, а також підтримує багато розширених функцій, як, наприклад, як повнотекстовий пошук.
- Надійність та стабільність. PostgreSQL відомий своєю стабільністю та надійністю, що робить його популярним вибором для критичних застосунків та великих підприємств.
- Підтримка ACID. PostgreSQL гарантує виконання властивостей ACID (Atomicity, Consistency, Isolation, Durability), що забезпечує цілісність даних та надійність операцій, а також стабільність даних та уникнення втрати інформації.

Стосовно недоліків, які властиві для PostgreSQL, то до них відносять:

- Ресурсозатратність. В порівнянні з деякими іншими базами даних, PostgreSQL може вимагати більше ресурсів, особливо при обробці великих обсягів даних.
- Складність конфігурації. Конфігурація PostgreSQL може бути складною і неправильні налаштування можуть вплинути на продуктивність бази даних.

Віртуалізація та контейнеризація Docker

Docker [33] обраний як частина технологічного стеку з метою полегшення розгортання та управління всім програмним забезпеченням проекту. Цей програмний продукт є потужним інструментом для контейнеризації, який дозволяє упаковувати та розгортати програмне забезпечення та його залежності в ізольованому середовищі. Це полегшує розгортання додатку із забезпеченням узгодженості між окремими сервісами загальної архітектури. Завдяки такому підходу можливо уникнути проблем, пов'язаних з різницею середовищ, та забезпечити консистентність у різних етапах розробки та впровадження.

Переваги використання Docker:

- Легкість розгортання. Контейнери Docker легко створюються та розгортаються, забезпечуючи швидкий процес розробки та випробувань.
- Портативність. Docker контейнери можна запускати на будь-якій операційній системі, що має відповідне програмне забезпечення.
- Масштабованість. Docker спрощує масштабування додатків, дозволяючи розгортати та керувати контейнерами в багатьох екземплярах.

Недоліки використання Docker.

- Великі розміри образів. Docker-образи додатків можуть бути великими, що може впливати на час завантаження та використання ресурсів.
- Використання Docker вимагає розуміння основних концепцій контейнеризації.

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

3.1 Структурно-функціональне моделювання

Щоб отримати глибоке розуміння функціональності системи, визначити ролі різних компонентів та забезпечити базу для подальшого розроблення і впровадження, на етапі моделювання значна роль була відведена застосуванню структурного підходу моделювання систем [35].

Моделювання було реалізовано шляхом використання IDEF0 діаграм, діаграм декомпозиції та UseCase діаграм, що спрямовані на аналіз та розроблення структурних елементів системи та їх взаємодій. Ці інструменти дозволяють розглядати систему на різних рівнях деталізації, ідентифікувати ключові функції та взаємодії, а також розбивати складні системи на менші, керовані блоки. Такий підхід сприяє системному аналізу та проектуванню, роблячи його більш зрозумілим та ефективним.

IDEF0 (Integration Definition for Function Modeling) представляє собою методологію та нотацію, розроблену в межах програми ICAM (Integrated Computer-Aided Manufacturing) для моделювання функцій, які виконує система або організація. Головна мета IDEF0 полягає в створенні зручного інструменту для вираження функціональної інформації системи чи процесу. Методологія спрямована на ієрархічне моделювання, де діаграма може бути побудована на різних рівнях деталізації, представляючи різні рівні уточнення функцій.

Основний акцент у IDEF0 робиться на функціях та їх взаємодіях. Блоки, які представляють функції, вважаються ключовими елементами моделі. Для моделювання взаємодій та потоків даних між функціями використовуються стрілки, які вказують напрямок та об'єм інформації чи управління. Такий підхід дозволяє досліджувати функціональні аспекти (потоки даних, взаємодію між функціями) системи чи процесів у багатьох контекстах. [36]

Для побудови діаграми і аналізу функціональних аспектів програмного інтерфейсу формування каталогу медичних препаратів визначено наступні дані:

1. Вхідні дані: запити користувача та адміністратора, інформація про медичний препарат, інформація про користувача.
2. Управління: API інтерфейс каталогізації, правила реєстрації користувача, інструкції медичних препаратів.
3. Вихідні дані: інформація про препарат, персоналізовані картки за тегами, картки препаратів в каталозі, статус відповіді на запит.
4. Механізми: розробник каталогу та середовище розробки.

На рисунку 3.1 представлена контекстна діаграма IDEF0 програмного інтерфейсу формування каталогу медичних препаратів.

Далі з метою деталізації, визначення взаємозв'язків та уточнення логіки функцій для поліпшення розуміння та моделювання бізнес-процесів було виконано декомпозицію першого рівня в IDEF0. Така декомпозиція являє собою процес розгалуження функцій системи для отримання більш деталізованого опису. Функціональний блок основного процесу, який був виділений на рівні контексту було поділено на менші, що представляють окремі функції.

Для діаграми першого рівня було визначено наступні блоки декомпозиції:

1. Реєстрація користувача
2. Авторизація користувача
3. Додавання інформації про препарат
4. Оновлення інформації про препарат
5. Формування відповіді на запит

Серед уточнених функцій було визначено:

1. Авторизація користувачів можлива лише після реєстрації.
2. Вхідні дані у функціональні блоки взаємодії з інформацією передаються лише після авторизації користувача та отримання токена авторизації для виконання запитів.
3. Відповідь на запит формується після взаємодії із базою даних.

Діаграма декомпозиції IDEF0 першого рівня наведена на рис 3.2

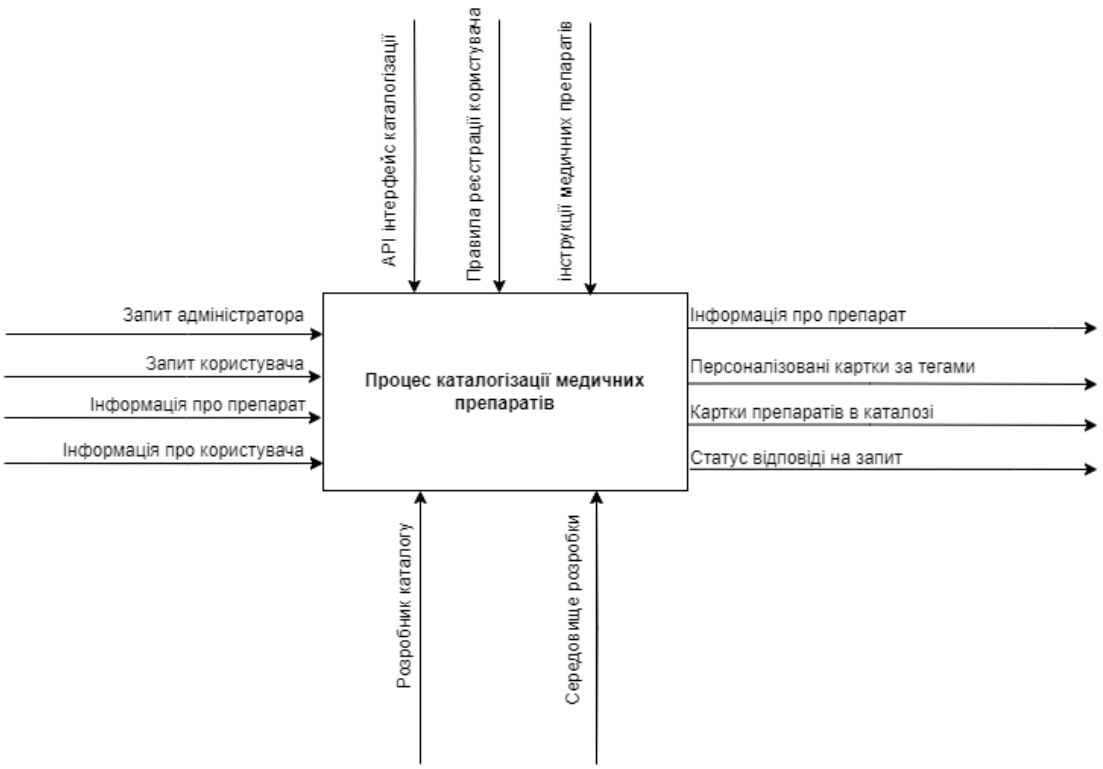


Рисунок 3.1 – Контекстна діаграма нотації IDEF0

Джерело: побудовано автором на основі даних моделювання

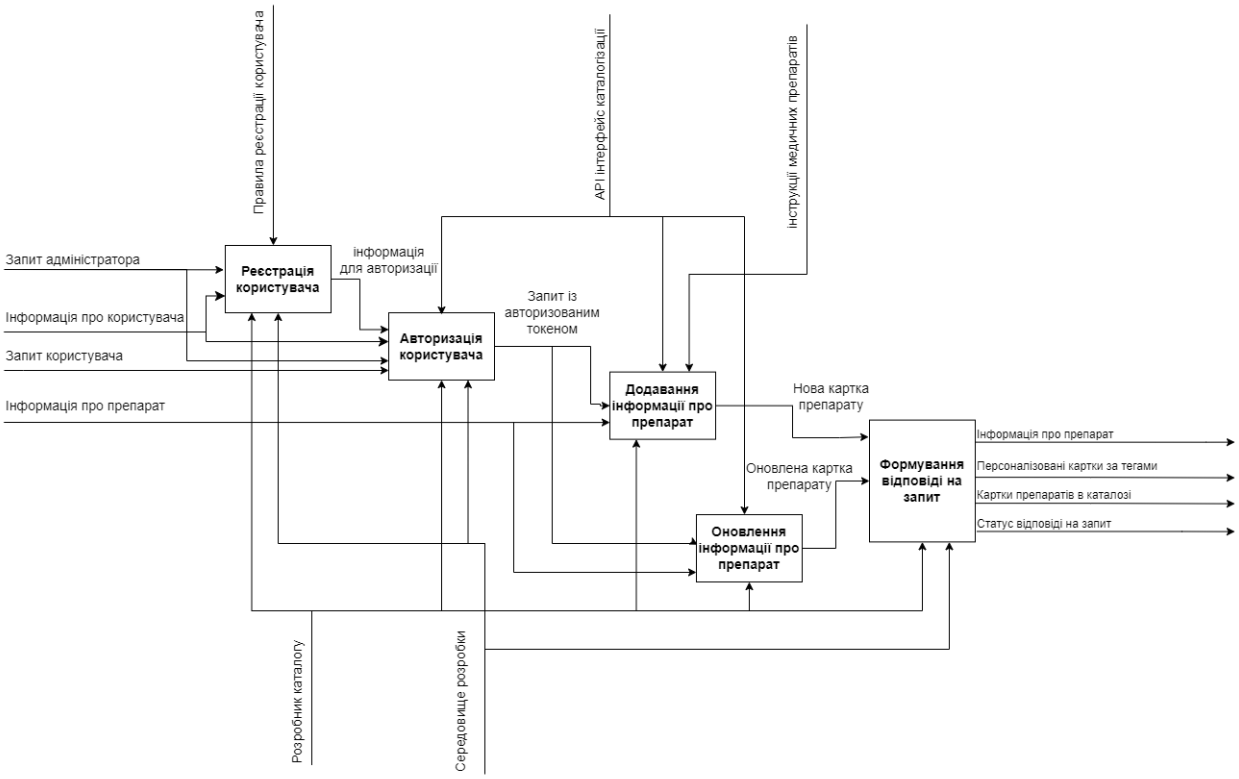


Рисунок 3.2. Діаграма декомпозиції першого рівня

Джерело: побудовано автором на основі даних моделювання

3.2 Діаграма Use Case

Для того, щоб визначити та уточнити вимоги до системи із точки зору використання користувачем, було створено діаграму використання.

Use Case діаграми дозволяють зрозуміти, як функції системи використовуються в конкретних сценаріях. Вони визначають ролі (акторів) та можливості (використання). Актори - це ролі чи об'єкти, які взаємодіють із системою, тоді як використання представляють конкретні функціональні можливості системи. Таким чином вони є інструментом в аналізі та проектуванні програмного забезпечення, який моделює взаємодії між системою та зовнішніми акторами (користувачами або іншими системами) [37, 38].

Для програмного інтерфейсу формування каталогу медичних препаратів було визначено наступні можливості використання:

- Автентифікація
- Видалення користувача
- Видалити препарат
- Видалити/оновити тег/компонент
- Запити API
- Керування користувачами
- Надання прав адміністратора
- Операції з медичними препаратами
- Операції з медичними препаратами
- Отримати інформацію про препарат
- Отримати список препаратів
- Отримати список тегів/компонентів
- Отримати схему API
- Перевірка роботи
- Пошук препарату за тегом/компонентом

- Реєстрація користувача
- Створити тег/компонент
- Створити/оновити препарат

Акторами діаграми Use Case є адміністратор, користувач та база даних.

Діаграма використання для програмного інтерфейсу формування каталогу медичних препаратів наведена на рис 3.3.

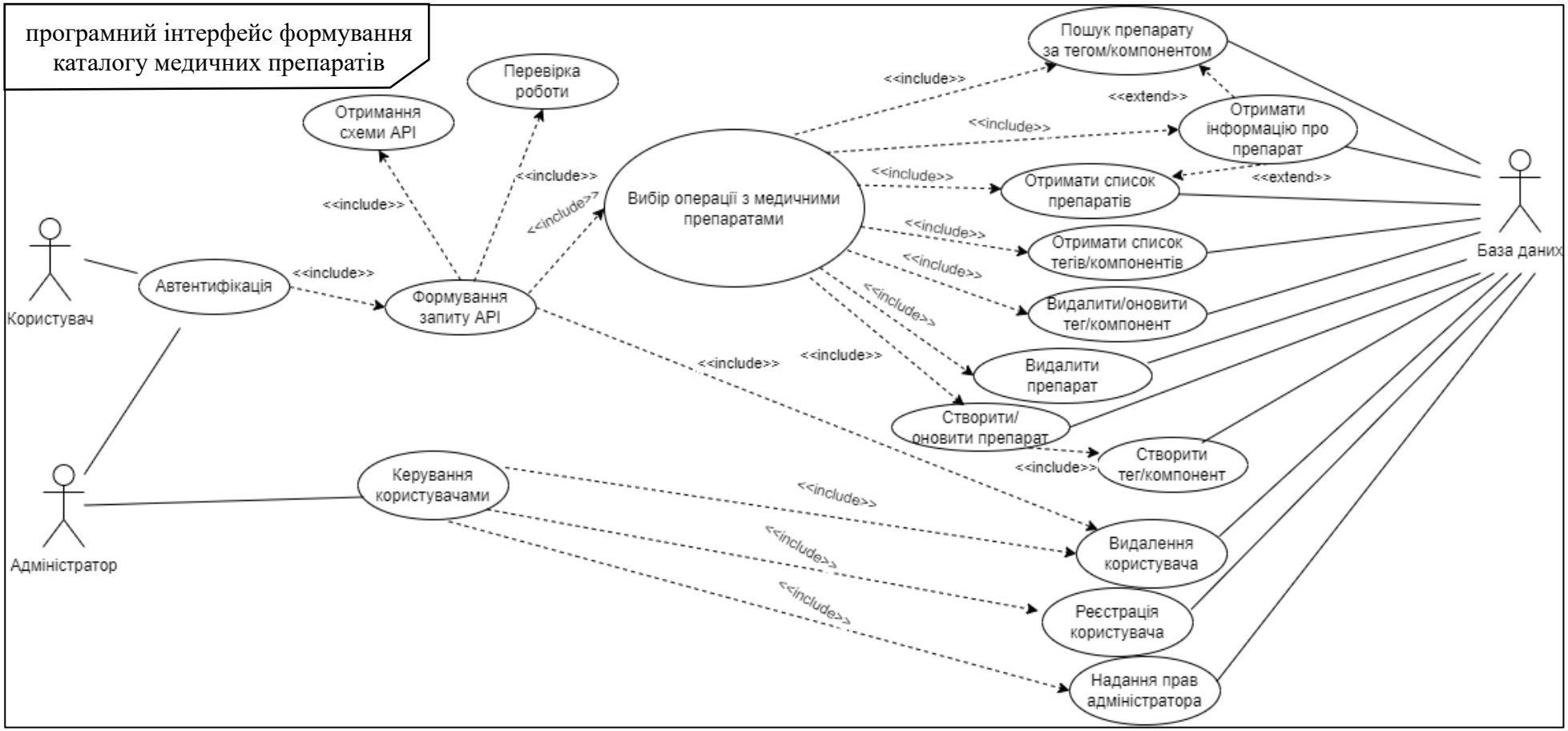


Рисунок 3.3 – Діаграма варіантів використання

Джерело: побудовано автором на основі даних моделювання

3.3 Моделювання бази даних

Моделювання бази даних (БД) є важливим етапом у розробці програмного забезпечення, оскільки воно дозволяє структурувати та організувати дані, які будуть використовуватися в програмному інтерфейсі. Досягається це шляхом попереднього визначення сутностей та ключових відносин між ними для логічної організації, збереження інформації та забезпечення цілісності даних. Крім того, попередньо змодельована БД із визначеними зв'язками дозволяє також уникати дублювання інформації, що може виникнути у процесі написання певних модулів. Наприклад, у різних медичних препаратів досить часто є однакові складники і тому важливо, щоб один і той самий компонент зв'язував препарати за цим параметром, а не був дубльований у БД.

У програмному інтерфейсі формування каталогу медичних препаратів використовується PostgreSQL як система управління базами даних (СУБД), адже PostgreSQL відомий своєю надійністю та підтримкою складних типів даних.

Врахувавши поставлені завдання та типи даних, для програмного інтерфейсу було визначено і у подальшому змодельовано наступні сутності:

- Користувач (CoreUser): Основна сутність, що представляє користувачів системи.
- Група (AuthGroup): Групи користувачів, до яких можуть належати користувачі.
- Дозвіл (AuthPermission): Дозволи, які можуть надаватися користувачам та групам.
- Компонент (Component): Компоненти, які є складовою медичного препарату.
- Тег (Label): Теги (мітки) користувача, якими можуть бути марковані препарати для групування за певними параметрами та пришвидшення пошуку.
- Медичний препарат (Medication): сутність, що містить інформацію про назву, дозу, інструкції, ціну, посилання, теги, компоненти препарату, а також його зображення.

На основі наведеної структури, переліку сутностей та особливостей фреймворку Django, модель бази даних буде складатись з наступних таблиць:

- auth_group: зберігає інформацію про групи користувачів.
- auth_group_permissions: зберігає інформацію про деталі дозволів для груп користувачів.
- auth_permission: зберігає інформацію про деталі дозволів.
- authtoken_token: зберігає інформацію про токени авторизації.
- core_component: зберігає інформацію про компоненти медикаментів.
- core_label: зберігає інформацію про мітки ліків.
- core_medication: зберігає інформацію про медикаменти.
- core_medication_components: Зв'язок між медикаментами та компонентами.
- core_medication_labels: Зв'язок між медикаментами та мітками.
- core_user: зберігає інформацію про користувачів.
- core_user_groups: зберігає зв'язки між користувачами та групами.
- core_user_user_permissions: зберігає зв'язки між користувачами та дозволами.

На ERD-діаграмі бази даних (Рисунок 3.4) візуалізовано зв'язки між сутностями, вказано на відносини та ключі, а також відображено типи даних в таблицях. Таким чином ця діаграма служить зручним інструментом для розуміння структури.

Також на вказаній діаграмі присутні системні таблиці Django:

- django_admin_log: зберігає записи з історії адміністративних дій.
- django_content_type: зберігає інформацію про типи контенту для дозволів.
- django_migrations: зберігає записи міграцій бази даних.
- django_session: зберігає інформацію про сесії

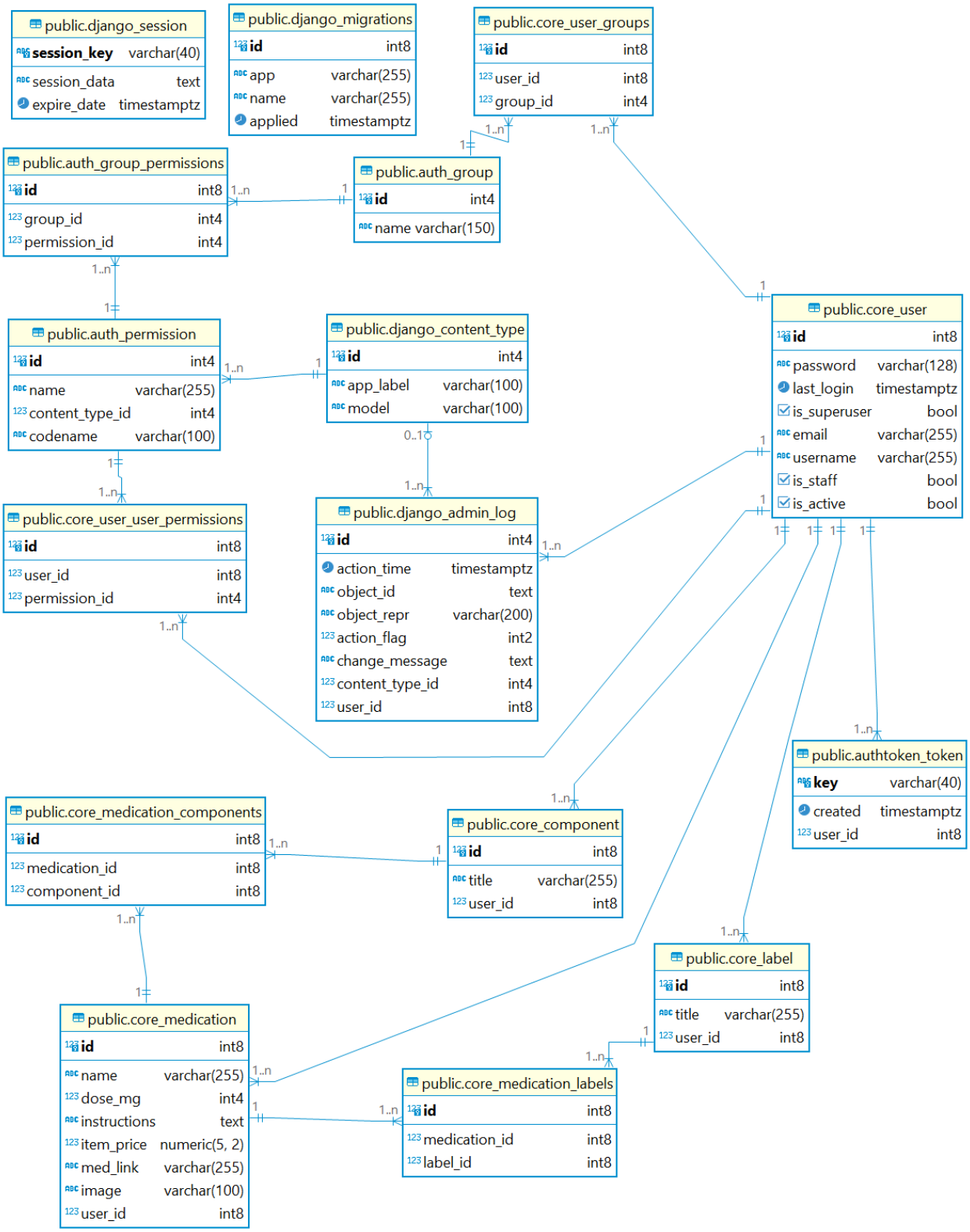


Рисунок 3.4 – ERD діаграма змодельованої БД

Джерело: побудовано автором на основі даних моделювання

Для підвищення ефективності зберігання та доступу до даних пропонується застосовувати такі методи оптимізації.

- Використання цілочисельних типів даних для полів, значення яких можуть бути цілими числами. Наприклад, для полів dose_mg та id варто використати цілочисельні типи даних IntegerField. Це дозволить оптимізувати зберігання та доступ до цих даних, оскільки цілочисельні типи даних займають менше місця в базі даних, ніж текстові типи даних.
- Використання індексів для полів, які часто використовуються для пошуку. Так для полів name та label необхідно забезпечити створення індексів. Індекси дозволяють СУБД швидко знаходити записи в базі даних за значенням цих полів. Це підвищує ефективність пошуку препаратів та тегів.
- Використання FOREIGN KEY для зв'язків між сутностями. FOREIGN KEY дозволяє СУБД зберігати інформацію про зв'язок між сутностями в одній таблиці, що підвищує ефективність зберігання та доступу до даних.

3.4 Моделювання API

Моделювання API - це ключовий етап у розробці, який визначає структуру та поведінку веб-сервісів. Цей розділ описує кінцеві точки API, які дозволяють взаємодіяти з ресурсами системи в програмному інтерфейсі формування каталогу медичних препаратів.

API структуровано за ресурсами, кожен з яких має набір дій для взаємодії з даними. Структура чітко визначає, які дії можна виконувати з кожним ресурсом, а також формат даних, які потрібно використовувати.

Підхід із попередньою розробкою структури має ряд переваг. По-перше, це дозволяє легко зрозуміти, які дані доступні через API, і які дії можна виконувати з цими даними. Це робить API більш зрозумілим для розробників і користувачів.

По-друге, структурування робить продукт більш масштабованим. Додавання нових ресурсів або дій не вимагає суттєвих змін в існуючій структурі. Розробники можуть бути впевнені, що їхні програми будуть продовжувати працювати належним

чином, навіть якщо API буде оновлено, адже схема API є стабільним документом, який не змінюється без необхідності.

Кожна кінцева точка має детальну інформацію про те, що вона робить, які дані вона приймає та повертає, а також будь-які обмеження доступу. Задля підвищення безпеки деякі кінцеві точки вимагають авторизації за допомогою токена, який потрібно передати в заголовку авторизації (Формат токена ключ – значення).

Кінцеві точки, що буде використовувати інтерфейс каталогізації, а також їхні функції наведені в таблиці 3.1.

Таблиця 3.1 – Перелік API програмного інтерфейсу каталогізації

API	Дії
/api/healthy	Перевірка справності API
/api/medication/components	Керування компонентами препарати
/api/medication/labels	Керування мітками препарати
/api/medication/medications	Керування записами про препарати
/api/schema	Отримання схеми API
/api/user	Керування користувачами

Джерело: побудовано автором на основі моделювання

У разі виникнення помилки API повертає відповідь з кодом помилки та описом проблеми. Коди помилок та їхні описи наведені в таблиці 3.2.

Таблиця 3.2 – Перелік кодів помилок

Код помилки	Опис
400	Помилка запиту
401	Неавторизований користувач
403	Недостатньо прав
404	Ресурс не знайдено
500	Внутрішня помилка сервера

Джерело: побудовано автором на основі моделювання

API використовує формат JSON для передачі та отримання даних. Формати даних для кожного ресурсу наведені в таблиці 3.3.

Таблиця 3.3 – Формати даних відповідей кінцевих точок

Ресурс	Формат даних
/api/healthy	{}

/api/medication/components	{
	"id": <ідентифікатор компонента>,
	"title": <назва компонента>,
	"description": <опис компонента>
	}

/api/medication/labels	{
	"id": <ідентифікатор мітки>,
	"title": <назва мітки>,
	"description": <опис мітки>
	}

/api/user	{
	"id": <ідентифікатор користувача>,
	"username": <ім'я користувача>,
	"password": <пароль>,
	"email": < email користувача>
	}

/api/medication/medications	{
	"id": <ідентифікатор запису про медикамент>,
	"name": <назва медикаменту>,
	"components": [<список ідентифікаторів компонентів>],
	"labels": [<список ідентифікаторів міток>],
	"description": <опис медикаменту>,
	"image": <зображення медикаменту>,
	"link": <посилання на медикамент>
	}

/api/schema	JSON схема API

Джерело: побудовано автором на основі моделювання

Наявність такої документованої інформації про форматування даних чіткий та систематизований інтерфейс для взаємодії з інформаційною системою, що спрощує розробку, тестування та впровадження системи.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

Реалізація програмного інтерфейсу формування каталогу медичних препаратів передбачає чотири етапи:

1. Налаштування контейнеризації.
2. Розробка програмного інтерфейсу моделей.
3. Конфігурація і документація API.
4. Покриття програмного інтерфейсу тестами. Тестування.

Розбиття розробки програмного інтерфейсу на етапи дозволяє зосередитися на конкретних завданнях, спрощуючи процес розробки. Кожен етап визначає конкретні завдання, що сприяє структурованості та послідовності робіт. Такий підхід забезпечує ефективну та стабільну розробку програмного інтерфейсу, надаючи можливість контролювати і відстежувати прогрес розробки, сприяє вчасному виявленню можливих проблем та дозволяє здійснювати корективні заходи на ранніх етапах.

4.1 Налаштування контейнеризації

Для ефективної роботи програмного інтерфейсу необхідно налаштувати контейнеризацію. Використання інструментів, таких як Docker, дозволяє створювати ізольовані середовища, що містять усі залежності та конфігурації програмного продукту. Це досягається шляхом забезпечення консистентності середовища між робочою станцією розробника та сервером, що сприяє уникненню можливих конфліктів та забезпечує переносимість між різними операційними системами.

Після встановлення технології віртуалізації Docker, наступний етап передбачає розробку програмного інтерфейсу моделей, включаючи в себе контейнеризацію фреймворків, баз даних та інших програм. Опис цього процесу зосереджений у файлі-сценарії docker-compose.yml, де вказані шляхи до сервісних Dockerfile, шляхи до

директорій директорій, зв'язки між контейнерами та інші необхідні конфігураційні параметри.

Лістинг коду Dockerfile конфігурації серверного оточення наведено на рисунку 4.1.

```
FROM python:3.9-alpine3.13
LABEL maintainer="NickSiryk"

ENV PYTHONUNBUFFERED=1

COPY ./requirements.txt /req/requirements.txt
COPY ./dev.requirements.txt /req/dev.requirements.txt

ARG DEV=false

# Install dependencies and upgrade pip
RUN apk add --update --no-cache bash build-base postgresql-client jpeg-dev \
    && apk add --update --no-cache --virtual .tmp-build-deps \
        build-base postgresql-dev musl-dev zlib zlib-dev linux-headers \
        build-base postgresql-dev musl-dev zlib zlib-dev \
    && pip install --upgrade pip \
    && pip install -r /req/requirements.txt


# If DEV argument is provided, install additional dependencies
RUN if [ $DEV = "true" ]; then apk add --update --no-cache graphviz graphviz-
dev && pip install -r /req/dev.requirements.txt; fi \
    && rm -rf /req \
    && apk del .tmp-build-deps

# Create a non-root user
RUN adduser --disabled-password --no-create-home django-user

# Set up directories for media and static files
RUN mkdir -p /vol/pcs/media && mkdir -p /vol/pcs/static \
    && chown -R django-user:django-user /vol \
    && chmod -R 755 /vol

# Switch to the non-root user and set the working directory
USER django-user
COPY ./app /app/
WORKDIR /app
EXPOSE 8000
```

Рисунок 4.1 – Dockerfile конфігурації

Джерело: розроблено автором (знімок з екрану)

Dockerfile системи починається із налаштування базової системи python:3.9-alpine3.13. Після чого наступним етапом є конфігурація додаткових бібліотек Python, вимоги до яких містяться у файлах requirements.txt, котрі копіюються до віртуального середовища. Далі встановлюються залежності і бібліотеки для роботи із базами даних. На наступних етапах відбувається конфігурація шляхів використання файлів та створюється базовий користувач системи.

Наступним конфігурується файл docker-compose.yml, конкретизує верхньорівневу інфраструктуру системи, описуючи, як контейнери взаємодіють між

собою (у програмному інтерфейсі каталогізації це база даних і сервер). Лістинг коду docker-compose.yml контейнеризації наведено на рисунку 4.2.

```
version: "3.9"

services:

  app:
    build:
      context: .
      args:
        - DEV=true
    ports:
      - "8002:8000"
    volumes:
      - ./app:/app
      - pcs-static-dev:/vol/pcs
    command: >
      sh -c "python manage.py db_ping &&
             python manage.py migrate &&
             python manage.py runserver 0.0.0.0:8000"
    environment:
      - DATABASE_HOST=db
      - DATABASE_NAME=devdb
      - DATABASE_USER=devuser
      - DATABASE_PASSWORD=changeme
      - DEBUG=1
    depends_on:
      - db

  db:
    image: postgres:13-alpine
    volumes:
      - dev-db:/var/lib/postgresql/data
    ports:
      - 5432:5432
    environment:
      - POSTGRES_DB=devdb
      - POSTGRES_USER=devuser
      - POSTGRES_PASSWORD=changeme

volumes:
  dev-db:
  pcs-static-dev:
```

Рисунок 4.2 – docker-compose.yml контейнеризації

Джерело: розроблено автором (знімок з екрану)

У вказаному файлі встановлена залежність між контейнерами серверного оточення, котрий сконфігурований у Dockerfile та базою даних db (postgres:13-alpine). Тут також визначено змінні оточення, які використовуються для налаштування різних параметрів, таких як паролі до бази даних, порти доступу та обміну інформацією та шляхи використання сховищ. Окремо вказано команду запуску системи, що містить три етапи:

1. Очікування запуску бази даних
2. Процес міграції (оновлення схеми бази даних)

3. Безпосередній запуск сервера.

Процес використання Docker у проекті передбачає два основних кроки: спочатку необхідно побудувати контейнеризовану структуру системи за допомогою команди "docker-compose build" (процес наведено на рисунку 4.3). Ця команда відповідає за завантаження необхідного програмного забезпечення, бібліотек та утиліт з визначених Інтернет-репозиторіїв, зазначених у відповідних Dockerfile. Після успішного завершення цього етапу, система готова до подальшого використання.

Рисунок 4.3 – процес побудови контейнеризованої структури системи

Джерело: виконано автором (знімок з екрану)

Кожен наступний запуск системи здійснюється за допомогою команди "docker-compose up". Ця команда піднімає раніше збудовану структуру системи, дозволяючи взаємодіяти з розгорнутою системою без необхідності повторного проходження етапу будівництва системи. Такий підхід полегшує розгортання та тестування, забезпечуючи ефективний цикл розробки.

4.2 Розробка програмного інтерфейсу моделей

При розробці моделей системи для програмного інтерфейсу формування каталогу медичних препаратів було враховано етап моделювання і створено відповідні сутності із вказаними властивостями та полями.

Модель користувача (User) - це основна сутність, що представляє користувачів системи. Містить такі поля:

- id - унікальний ідентифікатор користувача;
- password - пароль користувача;
- last_login - час останнього входу користувача в систему;
- is_superuser - поле, що вказує, чи є користувач адміністратором;
- email - електронна адреса користувача;
- username - логін користувача;
- is_staff - поле, що вказує, чи є користувач співробітником;
- is_active - поле, що вказує, чи є користувач активним.

Реалізація усіх вказаних полів у інтерфейсі Django не є необхідною, адже достатнім зробити наслідування від стандартних класів фреймворку AbstractBaseUser, PermissionsMixin (рис. 4.4).

В результаті це дозволяє створити правила формування відповідної сутності і записів у базі даних.

Крім того, використання UserManager() дозволило додатково задати валідацію та нормалізацію поля email.

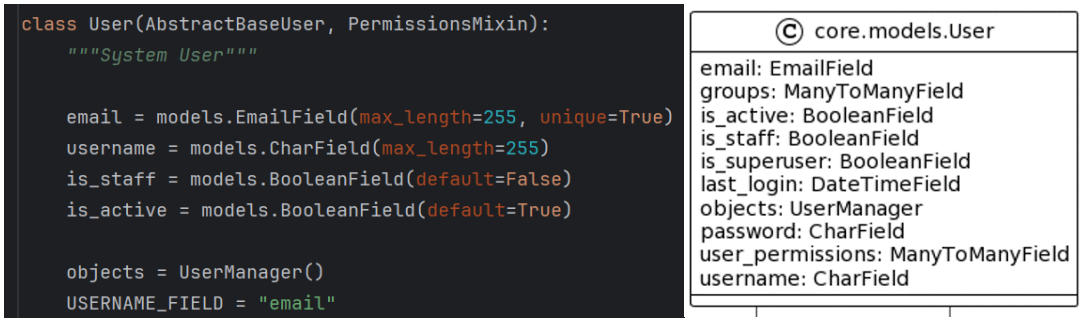


Рисунок 4.4 – Модель користувача в Django і у представленні UML

Джерело: виконано автором (знімок з екрану)

Група (AuthGroup) – це модель групи користувачів, до яких можуть належати користувачі (рис. 4.5). Містить такі поля:

– id - унікальний ідентифікатор групи;

– name - назва групи.

Ця сутність є стандартною сутністю Django і додатково не конфігурується, проте є доступною для використання на панелі адміністратора.

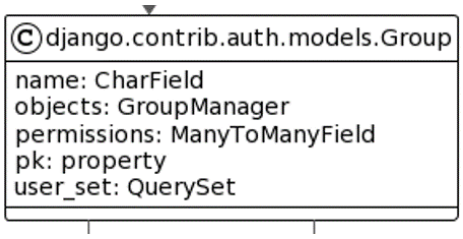


Рисунок 4.5 – Модель групи користувачів представленні UML

Джерело: розроблена інформаційна система

Дозвіл (Permission) – це модель дозволів, які можуть надаватися користувачам та групам. Містить такі поля:

– id - унікальний ідентифікатор дозволу;

– name - назва дозволу;

– content_type - тип об'єкта, на який поширюється дозвіл;

– codename - кодове ім'я дозволу.

Це також одна із стандартних моделей Django (рис. 4.6) і додатково не конфігурується, проте її застосовано для визначення рівнів доступів (адміністратор чи користувач).

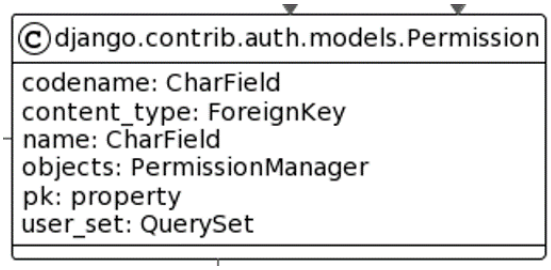


Рисунок 4.6 – Модель дозволів у представленні UML

Джерело: виконано автором (знімок з екрану)

Компонент (Component) - це модель компонентів, які є складовою медичного препарату (рис. 4.7). Містить такі поля:

– id - унікальний ідентифікатор компонента;
– title - назва компонента;
– user - користувач, який створив компонент.

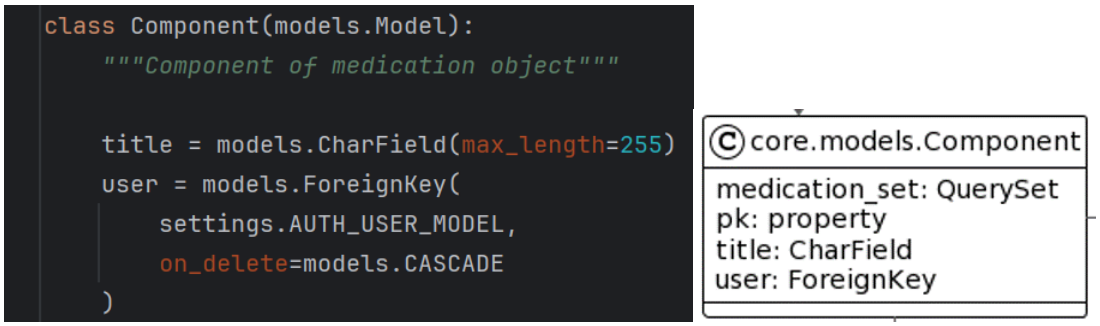


Рисунок 4.7 – Модель компоненту в Django і у представленні UML

Джерело: виконано автором (знімок з екрану)

Тег (Label) – це модель тегів (міток) користувача (рис. 4.8), якими можуть бути марковані препарати для групування за певними параметрами та пришвидшення пошуку. Містить наступні поля:

– id - унікальний ідентифікатор тегу;

– title - назва тегу;
– user - користувач, який створив тег.

Рисунок 4.8 – Модель міток в Django і у представленні UML

Джерело: виконано автором (знімок з екрану)

Медичний препарат (Medication) – основна сутність програмного інтерфейсу, що містить інформацію про препарат, а саме наступні поля:

– id - унікальний ідентифікатор препарату;
– name - назва препарату;
– dose_mg - доза препарату в міліграмах;
– instructions - інструкція по застосуванню препарату;
– item_price - ціна препарату;
– med_link - посилання на сайт виробника препарату;
– image - зображення препарату;
– user - користувач, який створив препарат.

Ця сутність є уже трохи складнішою за попередні, адже містить в якості полів інші сутності. Така структура реалізована в Django шляхом використання стандартних зв'язків ManyToManyField. Реалізація такої структури представлена на рисунку 4.9.

```python
class Medication(models.Model):
    """Medication object"""

    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    components = models.ManyToManyField("Component")
    dose_mg = models.IntegerField()
    labels = models.ManyToManyField("Label")
    instructions = models.TextField(blank=True)
    item_price = models.DecimalField(max_digits=5, decimal_places=2)
    med_link = models.CharField(max_length=255, blank=True)
    image = models.ImageField(
        null=True,
        upload_to=generate_medication_image_file_path
    )
```

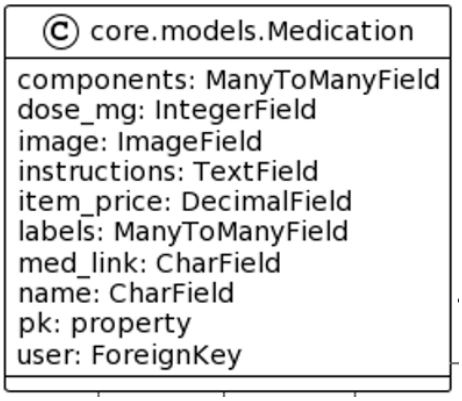


Рисунок 4.9 – Модель препарату в Django і у представленні UML

Джерело: виконано автором (знімок з екрану)

Використання стандартних зв’язків автоматично створює сутності відношення одного до іншого. Так, наприклад модель CoreMedicationComponents з'єднує компоненти з препаратами, в яких вони використовуються. Зв'язок між тегами та препаратами, яким вони належать, забезпечується схожою моделлю CoreMedicationLabels: З'єднує теги з препаратами, яким вони належать.

Загальна діаграма описаних моделей і їх взаємозв’язків наведена на рисунку 4.10.

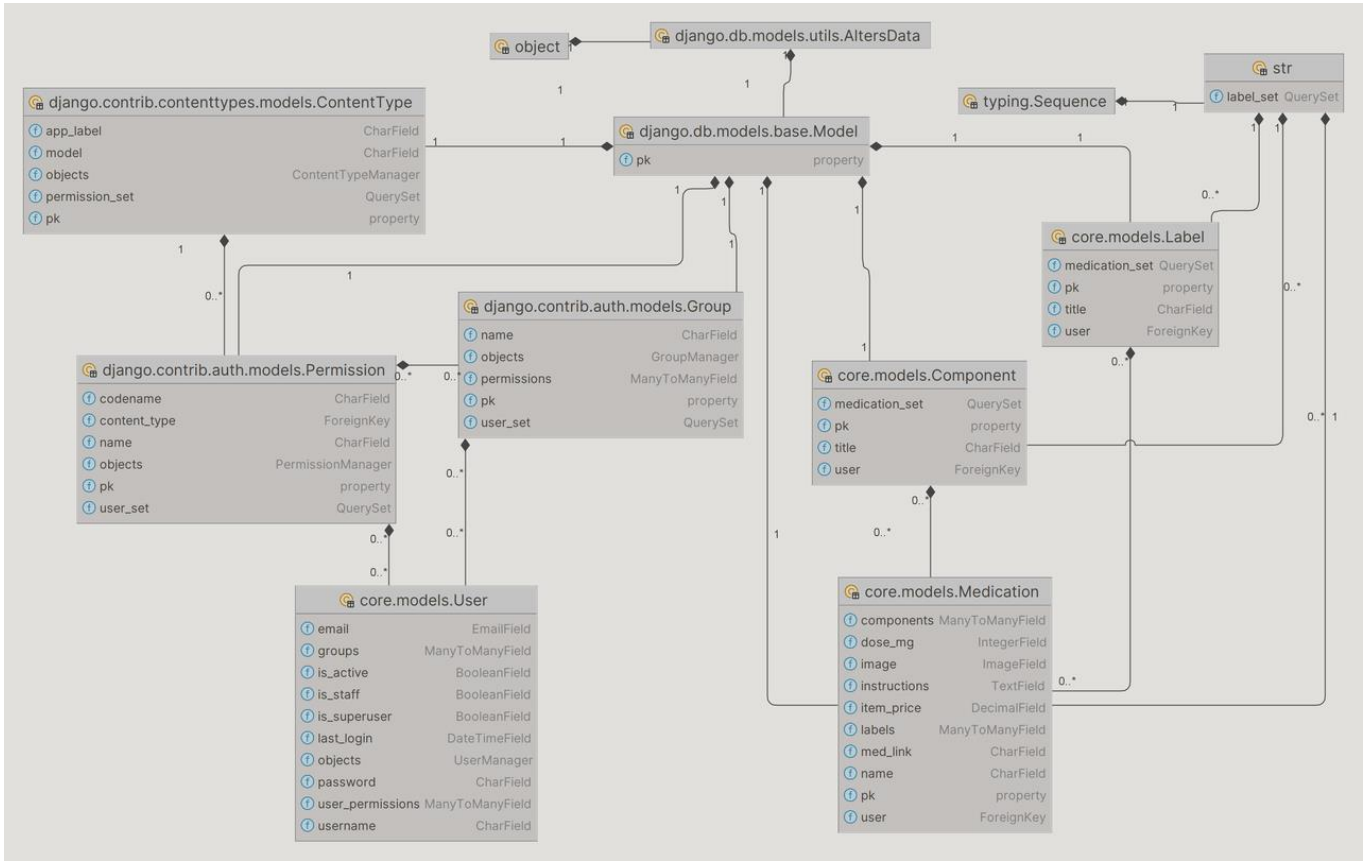


Рисунок 4.10 – Діаграма моделей системи

Джерело: побудовано автором на основі інформаційної системи

4.3 Конфігурація і документація API

Конфігурування API було виконано шляхом використання стандартних для Django зв’язок “адреса”-“метод”. Враховуючи попереднє моделювання, кінцеві точки було розподілено на глобальні і спеціальні. Глобальними було обрано кінцеві точки, що не відносяться до безпосередніх моделей медичних препаратів, тобто ті, котрі надають можливість отримання інформації про сервіс. Таким чином спеціальними є лише кінцеві точки api/user/ та api/medication/, кожна з яких має власні додаткові адреси (рис. 4.11)

```
urlpatterns = [
    path("", admin.site.login),
    path("admin/", admin.site.urls),
    path("api/docs/", SpectacularSwaggerView.as_view(url_name="api-schema"), name="api-docs"),
    path("api/healthy/", core_views.healthy, name="healthy"),
    path("api/medication/", include("medication.urls")),
    path("api/schema/", SpectacularAPIView.as_view(), name="api-schema"),
    path("api/user/", include("user.urls")),
]
```

Рисунок 4.11 – Глобальні кінцеві точки програмного інтерфейсу

Джерело: виконано автором (знімок з екрану)

Розділ "/admin/" у не є кінцевою точкою API, а належить фреймворку Django і представляє адміністративний інтерфейс (адмін-панель), який автоматично створюється для адміністрування моделей вашого додатка (рис. 4.12) . Він дає доступ до адміністративної частини програмного інтерфейсу. В адмін-панелі можна переглядати, редагувати та додавати дані, пов'язані з моделями програмного інтерфейсу, таким чином спрощуючи адміністрування та тестування. Для доступу до нього зазвичай, використовується метод GET, що вимагає автентифікації для отримання доступу. Користувачі повинні ввести свої облікові дані (ім'я користувача та пароль), щоб увійти.

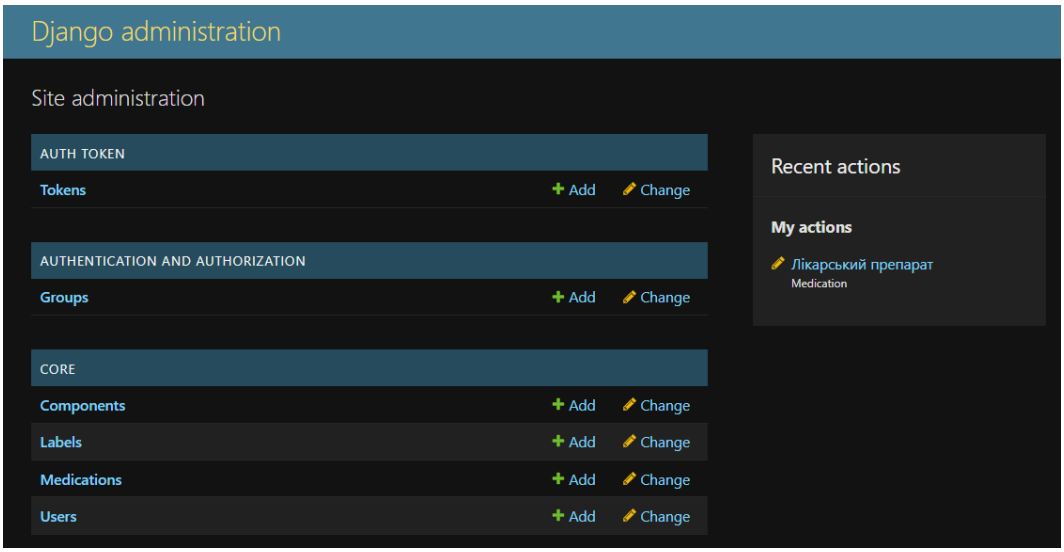


Рисунок 4.12 – Адмін-панель Django

Джерело: побудований програмний інтерфейс (знімок з екрану)

Для перевірки стану сервера перед взаємодією з інформаційною системою слід використовувати точку /api/healthy/. Ця точка доступна за допомогою HTTP-методу GET і не вимагає тіла запиту. Відповідь на успішний запит буде кодом 200 і відсутністю тіла відповіді. Призначена для перевірки доступності системи та її готовності до роботи.

Для отримання схеми API у форматі YAML або JSON використовується точка /api/schema, яка повертає схема REST-framework класу SpectacularAPIView. Ця точка також доступна за допомогою HTTP-методу GET і не вимагає тіла запиту.

Розділ "api/docs/" визначає точку доступу для документації API системи. Ця частина системи призначена для надання зрозумілої та докладної інформації щодо доступних API-точок, їх параметрів, типів даних та інших аспектів використання API. Ця точка також доступна за допомогою HTTP-методу GET і не вимагає тіла запиту. У інтерфейсі браузера вона повертає інтерфейс Swagger (рис. 4.14), що дозволяє ознайомитись із існуючою документацією API, а також виконувати запити різного типу одразу у web-браузері.

Щодо спеціальних кінцевих точок, то вони сконфігуровані у відповідних каталогах програмного інтерфейсу. На рисунках 4.12-4.13 наведено конфігурація спеціальних точок api/user/ та api/medication/.

```
app_name = "user"

urlpatterns = [
    path("create/", views.CreateUserView.as_view(), name="create"),
    path("get-token/", views.CreateTokenView.as_view(), name="get-token"),
    path("info/", views.ManageUserView.as_view(), name="info"),
]
```

Рисунок 4.12 – Спеціальні кінцеві точки користувача програмного інтерфейсу

Джерело: виконано автором (знімок з екрану)

```
router = DefaultRouter()
router.register( prefix: "medications", views.MedicationViewSet)
router.register( prefix: "labels", views.LabelViewSet)
router.register( prefix: "components", views.ComponentViewSet)

app_name = "medication"

urlpatterns = [path("", include(router.urls))]
```

Рисунок 4.13 – Спеціальні кінцеві точки медичного препарату

Джерело: виконано автором (знімок з екрану)

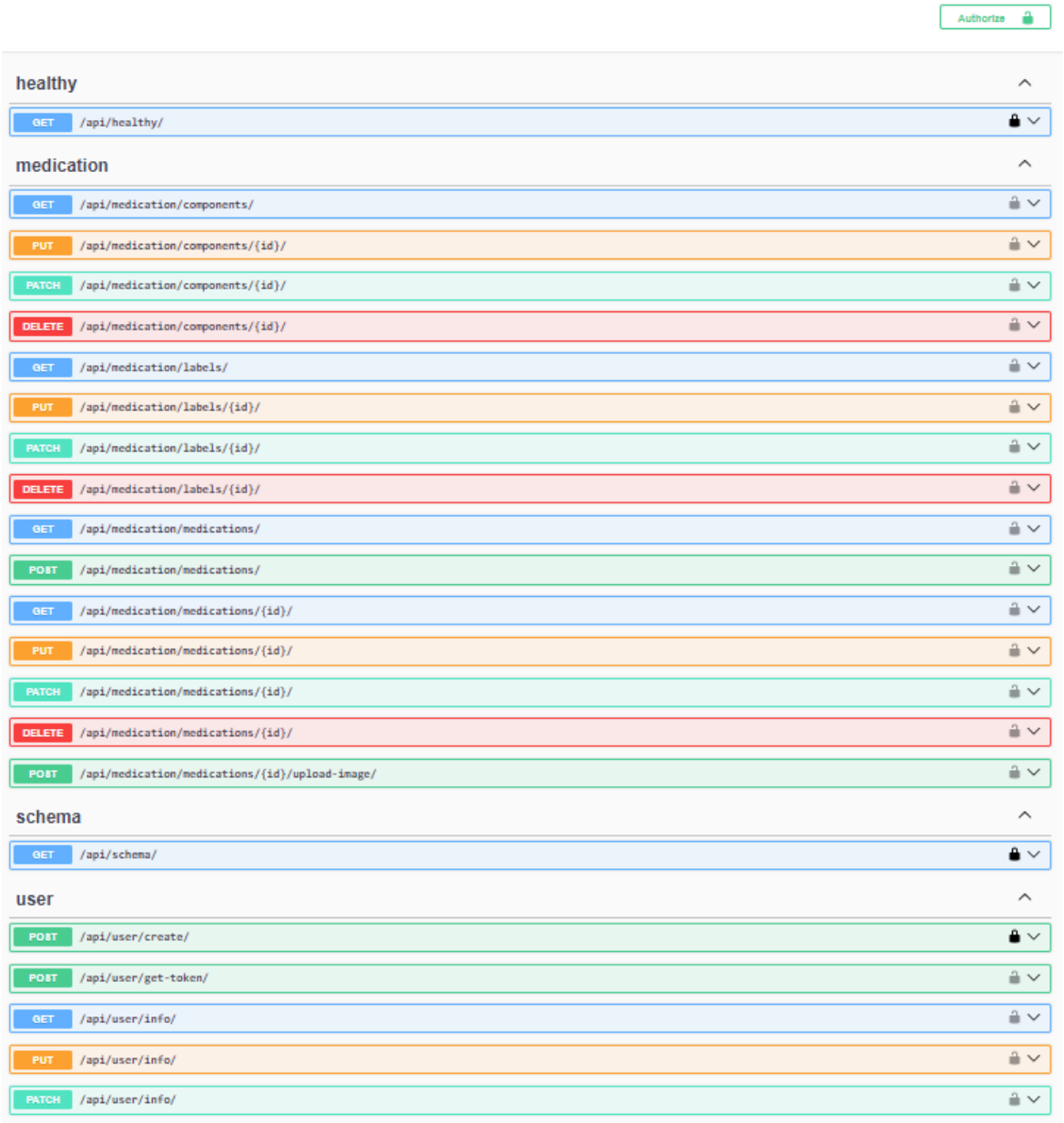


Рисунок 4.14 – Інтерфейс документації API Swagger

Джерело: побудований програмний інтерфейс (знімок з екрану)

Спеціальні кінцеві точки уже потребують авторизації через токен.

Для створення нового користувача слід використовувати точку /api/user/create/. Ця точка доступна за допомогою HTTP-методу POST і приймає дані користувача у форматі JSON, URL-кодування або multipart/form-data:

- email: адреса електронної пошти користувача. Обов'язкове поле.
- password: пароль користувача. Обов'язкове поле.
- username: ім'я користувача. Обов'язкове поле.

Успішний запит поверне код 201 і дані новоствореного користувача у форматі JSON.

Для отримання токену аутентифікації слід використовувати точку /api/user/get-token/. Ця точка доступна за допомогою HTTP-методу POST і приймає дані користувача у форматі JSON, URL-кодування або multipart/form-data:

- email: адреса електронної пошти користувача. Обов'язкове поле.
- password: пароль користувача. Обов'язкове поле.

Успішний запит поверне код 200 і токен аутентифікації у форматі JSON (об'єкт Token з інформацією про новий токен автентифікації). У подальшому цей токен необхідно використовувати для надсилання запитів до інших кінцевих точок для верифікації змін, адже кожна модель містить поле із записом користувача, хто вніс зміни до картки.

Точка /api/user/info/ дозволяє отримати інформацію про поточного користувача або оновити існуючу. За допомогою HTTP-методу GET і не без тіла можна отримати інформацію. Успішний запит поверне код 200 і дані поточного користувача у форматі JSON. А от за допомогою HTTP-методу PUT із тілом запиту, яке має містити нові дані про користувача (аналогічного формату, як і для /api/user/create/) дозволяє оновити інформацію про поточного користувача. HTTP-методу PATCH дозволяє частково оновити інформацію про поточного користувача.

Для отримання інформації про компоненти з бази даних слід використовувати точку /api/medication/components/. Ця точка доступна за допомогою HTTP-методу GET і може приймати параметр assigned_only для фільтрації за призначеними

компонентами та параметр id_or_title для фільтрації за ім’ям чи ідентифікатором (рис. 4.16, 4.17). Відповідь на запит буде масивом об'єктів типу "Component".

Для управління конкретним компонентом в базі даних слід використовувати точку /api/medication/components/{id}/. Ця точка доступна за допомогою методів PUT, PATCH та DELETE і вимагає унікального ідентифікатора компонента в параметрі id або імені для пошуку. PUT вимагає тіла запиту (рис. 4.15) і дозволяє оновити компонент медикаменту, PATCH оновити частково, а DELETE – видалити запис із бази даних.

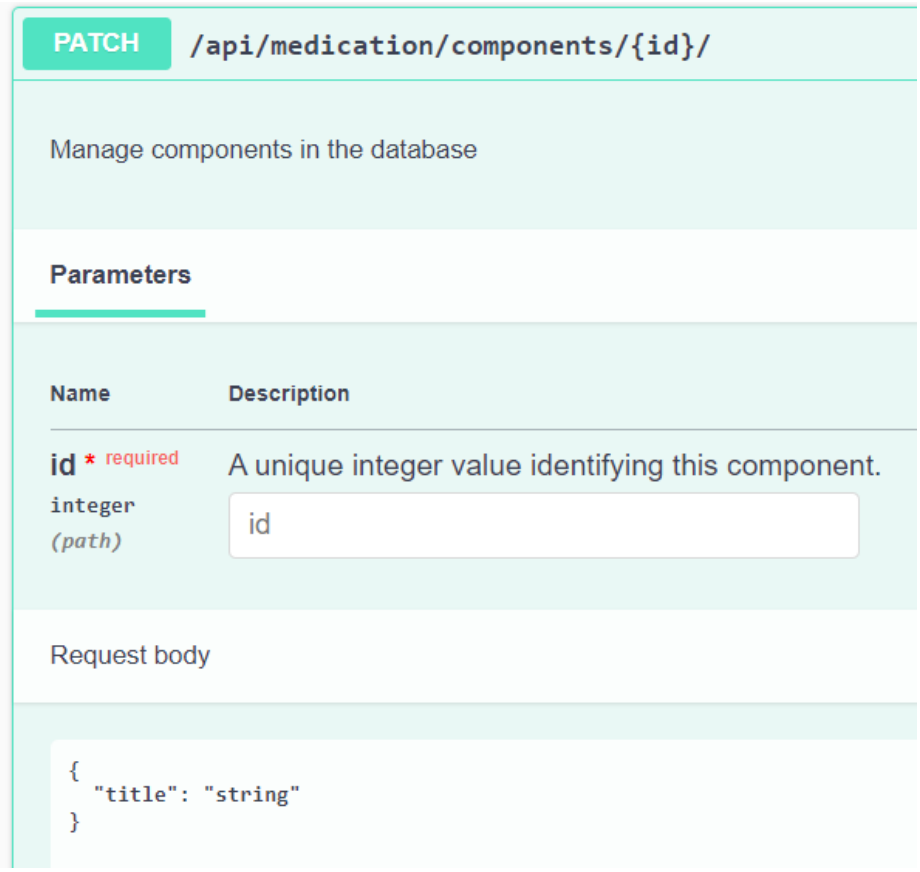


Рисунок 4.15 – Структура тіла запиту для оновлення компонента

Джерело: побудований програмний інтерфейс (знімок з екрану)

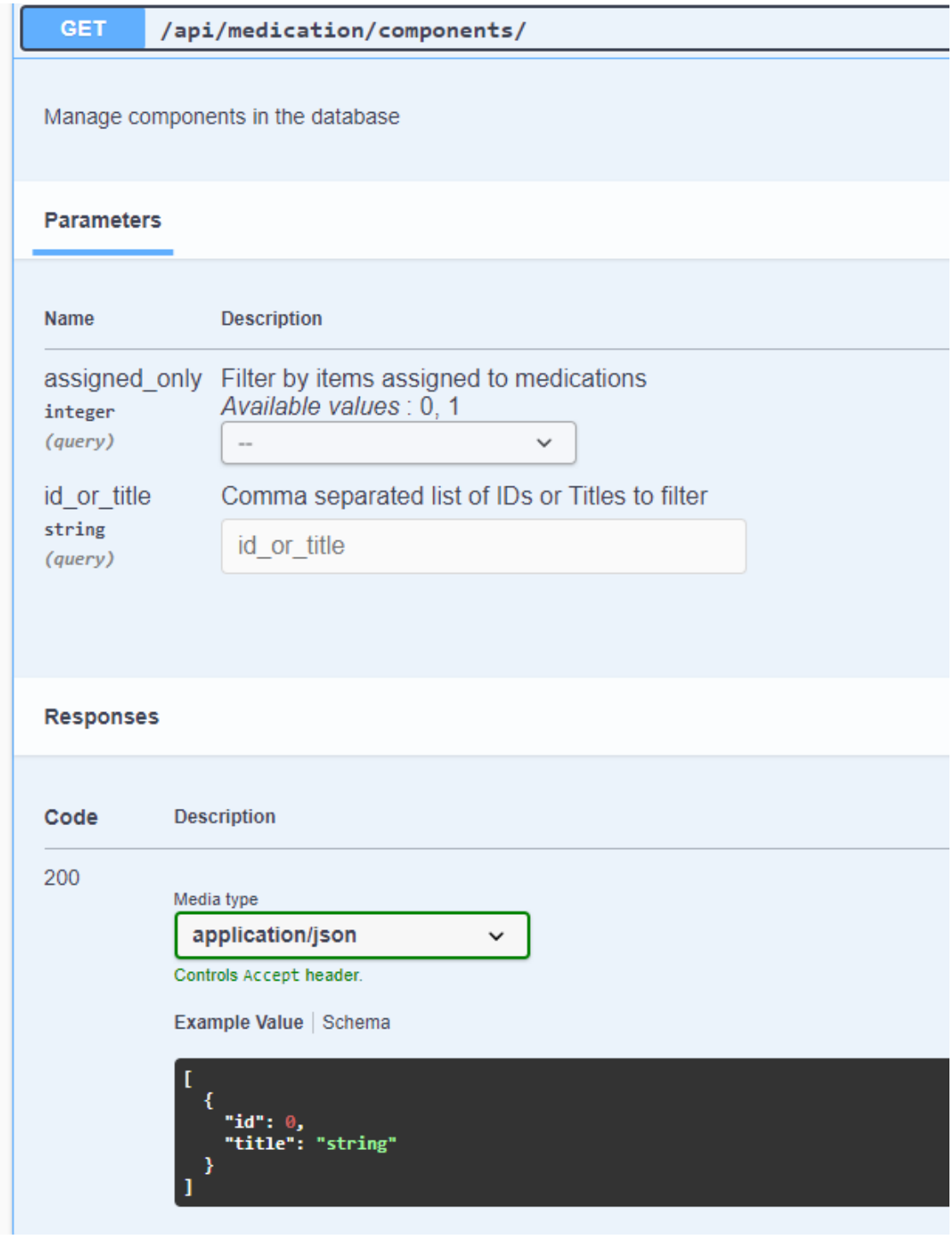


Рисунок 4.16 – Структура тіла запиту для отримання інформації про компонент

Джерело: побудований програмний інтерфейс (знімок з екрану)

Для управління мітками в базі даних слід використовувати точку /api/medication/labels/ сконфігурована аналогічним чином як API компонента і оперує масивом об'єктів типу "Label".

```
    permission_classes = [permissions.IsAuthenticated, IsOwnerOrReadOnly]

    def get_queryset(self):
        """Filter queryset for authenticated user"""
        assigned_only = bool(int(self.request.query_params.get("assigned_only", 0)))
        queryset = self.queryset
        if assigned_only:
            queryset = queryset.filter(medication__isnull=False)

        queryset_id = self.queryset.none()
        if id_or_title := self.request.query_params.get("id_or_title"):
            id_or_title = unquote(id_or_title, encoding="utf-8")
            if ids := _str_list_to_ints(id_or_title):
                queryset_id = queryset.filter(id__in=ids)
            titles = id_or_title.split(",")
            queryset_title = queryset.filter(title__in=titles)

            queryset = queryset_id | queryset_title

        return queryset.order_by("-title").distinct()
```

Рисунок 4.17 – Імплементація отримання інформації про компоненти

Джерело: виконано автором (знімок з екрану)

Для отримання списку лікарських засобів слід використовувати точку /api/medication/medications/, за допомогою HTTP-методу GET. Вона приймає параметри назви чи ідентифікатора компонента або мітки для фільтрації (рис 4.18). Успішний запит поверне код 200 і список лікарських засобів у форматі JSON (рис. 4.19).

```
def get_queryset(self):
    """Retrieve medications for authenticated user"""
    queryset = self.queryset
    queryset_labels = queryset_comp = self.queryset.none()

    if self.request.query_params:
        if labels := self.request.query_params.get("labels"):
            labels = unquote(labels, encoding="utf-8")
            queryset_labels_id = self.queryset.none()
            if labels_ids := _str_list_to_ints(labels):
                queryset_labels_id = queryset.filter(labels__id__in=labels_ids)
            labels_titles = labels.split(",")
            queryset_labels_t = queryset.filter(labels__title__in=labels_titles)
            queryset_labels = queryset_labels_t | queryset_labels_id

        if components := self.request.query_params.get("components"):
            components = unquote(components, encoding="utf-8")
            queryset_comp_id = self.queryset.none()
            if components_ids := _str_list_to_ints(components):
                queryset_comp_id = queryset.filter(components__id__in=components_ids)
            components_titles = components.split(",")
            queryset_comp_t = queryset.filter(components__title__in=components_titles)

            queryset_comp = queryset_comp_t | queryset_comp_id

        queryset = queryset_labels | queryset_comp

    return queryset.order_by("-id").distinct()
```

Рисунок 4.18 – Імплементація отримання інформації про препарат

Джерело: виконано автором (знімок з екрану)

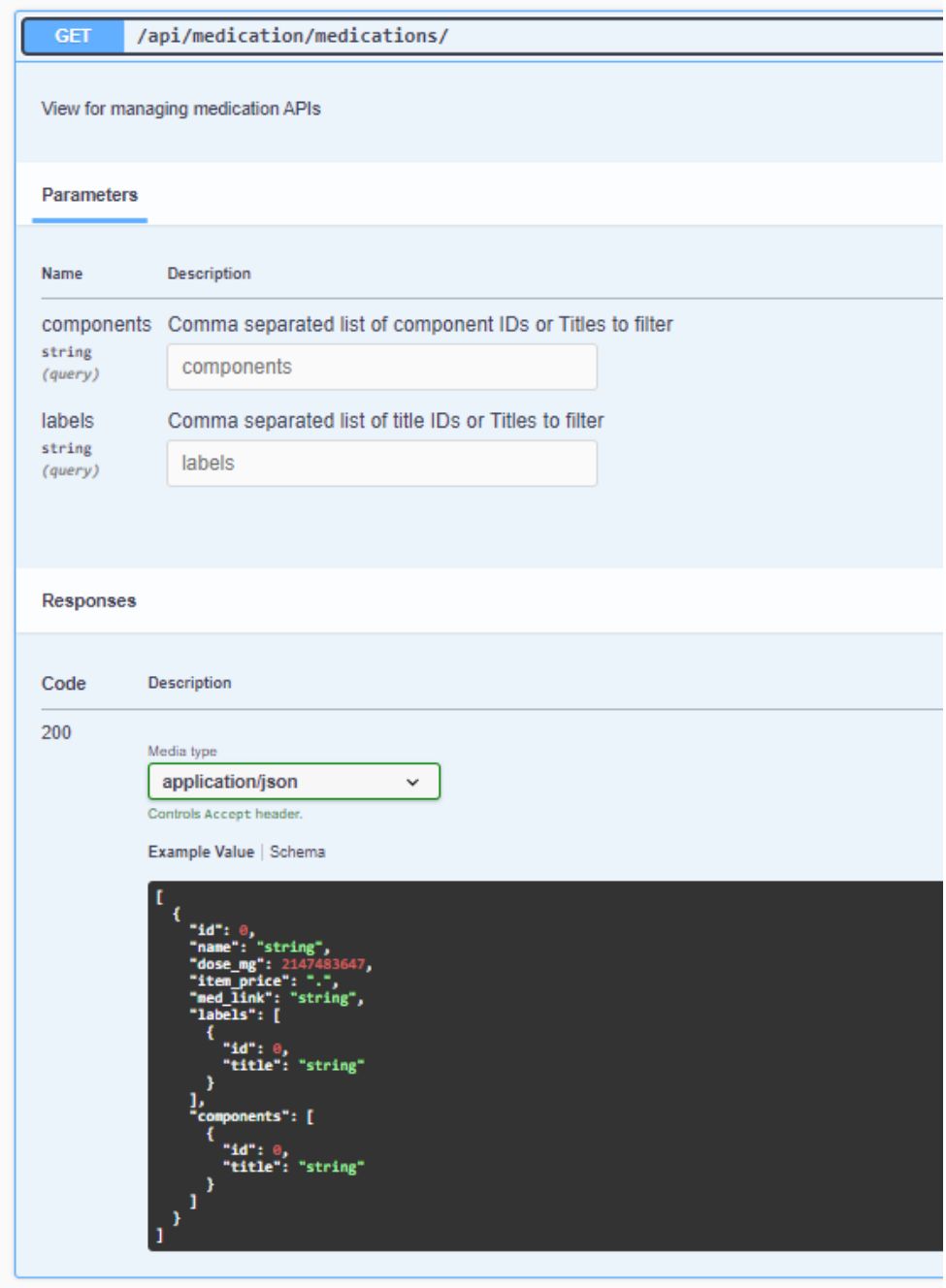


Рисунок 4.19 – Структура запиту для створення медичного препарату

Джерело: побудований програмний інтерфейс (знімок з екрану)

Для створення нового лікарського засобу слід використовувати ту ж саму точку /api/medication/medications/ за допомогою HTTP-методу POST. Вона в такому випадку приймає дані лікарського засобу у форматі JSON, URL-кодування або multipart/form-data (рис. 4.20). Успішний запит поверне код 201 і дані новоствореного лікарського засобу у форматі JSON.

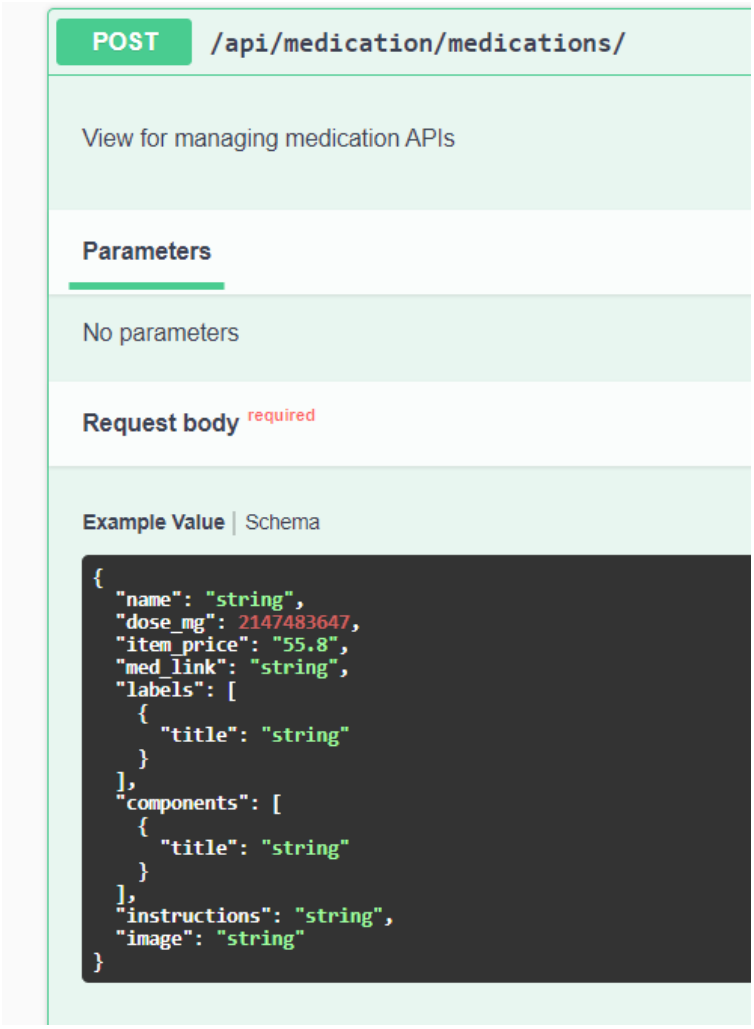


Рисунок 4.20 – Структура тіла запиту для створення медичного препарату

Джерело: побудований програмний інтерфейс (знімок з екрану)

Для управління конкретним медикаментом в базі даних слід використовувати точку /api/medication/medications/{id}/. Ця точка доступна за допомогою методів PUT, PATCH та DELETE і вимагає унікального ідентифікатора медикаменту в параметрі id.

4.4 Покриття програмного інтерфейсу тестами та вивірення коду

Програмний інтерфейс (API) є критичною складовою сучасних програмних рішень, і його надійність та ефективність є ключовими аспектами розробки. Всебічне тестування програмного інтерфейсу є ключовим етапом розробки, спрямованим на виявлення та усунення помилок. Одним з видів тестування є unit-тестування, яке фокусується на тестуванні окремих одиниць програмного коду. Для забезпечення повного покриття такими тестовими випадками всього коду програмного інтерфейсу, кожен модуль містить окремі файли з unit-тестами (рис. 4.21).

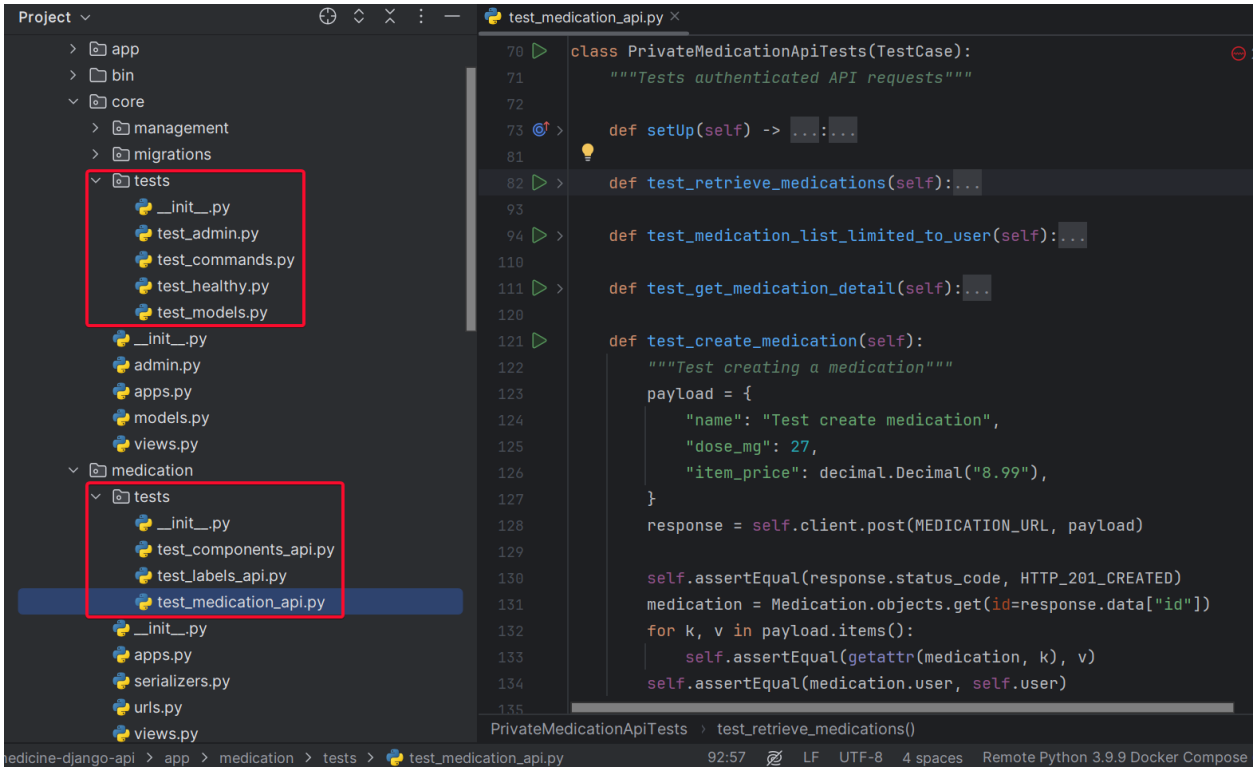


Рисунок 4.21 – Каталоги з unit-тестами та приклад тестування створення картки препарату

Джерело: виконано автором (знімок з екрану)

Покриття програмного інтерфейсу тестами (test coverage) - це міра того, наскільки велика частина програмного інтерфейсу була протестована. Для відслідковування покриття системи тестами використовується спеціальна бібліотека

coverage. Ця бібліотека надає детальні звіти щодо того, які частини коду були або не були покриті тестами, дозволяючи розробникам ефективно визначати області, які вимагають додаткового тестування. Для програмного інтерфейсу налаштоване покриття 100% коду та написаний окремий bash-скрипт для забезпечення читабельного виведення результатів (рис. 4.22, 4.23).

```bash
#!/usr/bin/env bash

set -e

Y='\033[01;33m'
N='\033[0m'

coverage erase

echo -e "\n$Y[1/2] Running UNIT tests\n$N"
coverage run --append manage.py test --timing --failfast --exclude-tag unit --noinput

echo -e "\n$Y[2/2] Generating coverage report\n$N"
coverage report
```

Рисунок 4.22 – bash-скрипт для забезпечення читабельного виведення результатів роботи бібліотеки coverage

Джерело: виконано автором (знімок з екрану)

```
$ docker-compose run --rm app bash -c "./bin/scripts/run_tests.sh"
WARNING: Compose V1 is no longer supported and will be removed from Docker Desktop in an
om/go/compose-v1-eol/
Creating medicine-django-api_app_run ...
Creating medicine-django-api_app_run ... done

[1/2] Running UNIT tests

Found 69 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....................................................................
----------------------------------------------------------------------
Ran 69 tests in 15.690s

OK
Destroying test database for alias 'default'...
Total database setup took 0.870s
  Creating 'default' took 0.869s
Total database teardown took 0.041s
Total run took 17.382s

[2/2] Generating coverage report

Name                          Stmts   Miss Branch BrPart   Cover   Missing
-------------------------------------------------------------------------
core/models.py                   57      0      4      0 100.00%
core/views.py                     5      0      0      0 100.00%
medication/serializers.py        58      0     10      0 100.00%
medication/views.py              79      7     28      3  88.79%   47->56, 50->52, 57-64
user/serializers.py              28      0      4      1  96.88%   24->28
user/views.py                    17      0      0      0 100.00%
-------------------------------------------------------------------------
TOTAL                           244      7     46      4  95.52%
Coverage failure: total of 95.52 is less than fail-under=100.00
2
```

```
$ docker-compose run --rm app bash -c "./bin/scripts/run_tests.sh"
WARNING: Compose V1 is no longer supported and will be removed from Docker De
om/go/compose-v1-eol/
Creating medicine-django-api_app_run ...
Creating medicine-django-api_app_run ... done

[1/2] Running UNIT tests

Found 74 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....................................................................
----------------------------------------------------------------------
Ran 74 tests in 14.888s

OK
Destroying test database for alias 'default'...
Total database setup took 0.924s
  Creating 'default' took 0.924s
Total database teardown took 0.035s
Total run took 16.596s

[2/2] Generating coverage report

Name                          Stmts   Miss Branch BrPart   Cover   Missing
-------------------------------------------------------------------------
core/models.py                   57      0      4      0 100.00%
core/views.py                     5      0      0      0 100.00%
medication/serializers.py        58      0     10      0 100.00%
medication/views.py              79      0     28      0 100.00%
user/serializers.py              28      0      4      0 100.00%
user/views.py                    17      0      0      0 100.00%
-------------------------------------------------------------------------
TOTAL                           244      0     46      0 100.00%
```

Рисунок 4.23 – Звіт бібліотеки coverage до і після перевірки покриття

Джерело: виконано автором (знімок з екрану)

Для розроблюваного програмного продукту було розроблено unit-тести для всіх компонентів програмного інтерфейсу. Результати тестування показали, що програмний інтерфейс повністю покритий тестами.

Вивірення коду - це не менш важливий аспект розробки, спрямований на підтримку читабельності, консистентності та високого стандарту програмного коду, адже використання програмного інтерфейсу каталогізації медичних препаратів передбачає впровадження його у існуючі системи, що вимагає високої якості та зрозумілості коду проекту. У даній роботі код вивірено за допомогою інструменту ruff, який визначено на використання правил трьох важливих інструментів: isort, flake8, та black.

- isort відповідає за автоматичне сортування імпортів в алфавітному порядку, забезпечуючи чистий та організований код.
- flake8 проводить статичний аналіз коду, нагадуючи розробникам про важливі стандарти оформлення, а також виявляючи можливі помилки та неузгодженості.
- black відповідає за автоматичне форматування коду згідно стандарту, що сприяє створенню єдиноформатного та читабельного коду.

Для вивірення програмного інтерфейсу налаштоване покриття 100% коду та написаний окремий bash-скрипт для забезпечення читабельного виведення результатів (рис. 4.24).

Використання ruff перед та після вивірення дозволило продемонструвати поліпшення читабельності та стандарту коду, забезпечуючи більшу зрозумілість та легшу супровідність проекту (рис. 4.25). Це допомогло уникнути потенційних проблем, пов'язаних із змінами у великих командах розробників та полегшити майбутню інтеграцію системи.

```
# Parse command line options
while getopts ":f" opt; do
  case $opt in
    f)
      force_flag=true
      ;;
    \?)
      echo "Invalid option: -$OPTARG" >&2
      exit 1
      ;;
  esac
done

# Shift the options so that $1 now refers to the first non-option argument
shift $((OPTIND-1))

# Check if the -f flag was provided
if [ "$force_flag" = true ]; then
  echo -e "\n$RED Formatting 🔍\n$NC"
  ruff check --isolated --extend-select="I" --line-length=80 --fix --exit-zero
  ruff format
else
  echo -e "\n$YEL Ruffing 🔍\n$NC"
  ruff check --isolated --extend-select="I" --line-length=80 --diff --exit-zero
  ruff check
fi
```

Рисунок 4.24 – bash-скрипт для вивірення коду за допомогою *ruff*

Джерело: виконано автором (знімок з екрану)

```
Ruffing 🔍

--- medication/views.py
+++ medication/views.py
@@ -1,4 +1,5 @@
 from urllib.parse import unquote
+
 from drf_spectacular.plumbing import OpenApiParameter, OpenApiTypes
 from drf_spectacular.utils import extend_schema, extend_schema_view
 from rest_framework import authentication, mixins, permissions, viewsets

Would fix 1 error.
medication/tests/test_medication_api.py:444:101: E501 Line too long (105 > 100)
medication/tests/test_medication_api.py:479:101: E501 Line too long (110 > 100)
Found 2 errors.
ERROR: 1
```

Рисунок 4.25 – Звіт бібліотеки ruff із пропозиціями вивірення

Джерело: виконано автором (знімок з екрану)

4.5 Використання програмного інтерфейсу

Перевірка використання програмного інтерфейсу передбачає мануальне тестування кінцевих точок розгорнутого середовища. Мануальне тестування програмного інтерфейсу (API) включає в себе перевірку функцій, які реалізовані. Воно дозволяє перевірити, чи відповідає API специфікації, а також чи працює він коректно. Перевірка кінцевих точок API передбачає роботу з ними безпосередньо. Для цього використовуються інтерфейс Swagger, який дозволяє надсилати HTTP-запити та отримувати відповіді.

У процесі тестування перевіряється відповідність відповідей API специфікації, а також їх правильність. Наприклад, для кінцевої точки реєстрації користувача перевіряється, чи можна зареєструватися з правильними даними, а також чи не можна зареєструватися з неправильним паролем.

Нижче наведено послідовний опис тестів для різних функціональних частин API, зокрема реєстрації користувача, авторизації, створення медичних препаратів та фільтрації каталогу.

Реєстрація користувача

1. Відкриваємо інтерфейс для перевірки API Swagger.
2. Використовуючи метод POST, відправляємо запит на реєстрацію користувача за адресою /api/user/create/.
3. У тілі запиту вводимо дані нового користувача, такі як email, пароль та ім'я користувача (рис. 4.26).
4. Перевіряємо, що отримали відповідь зі статусом 201 Created, а також відповідні дані про створеного користувача (рис. 4.27).

Отриманий результат:

- Запит на реєстрацію виконаний успішно.
- Створено новий обліковий запис користувача.

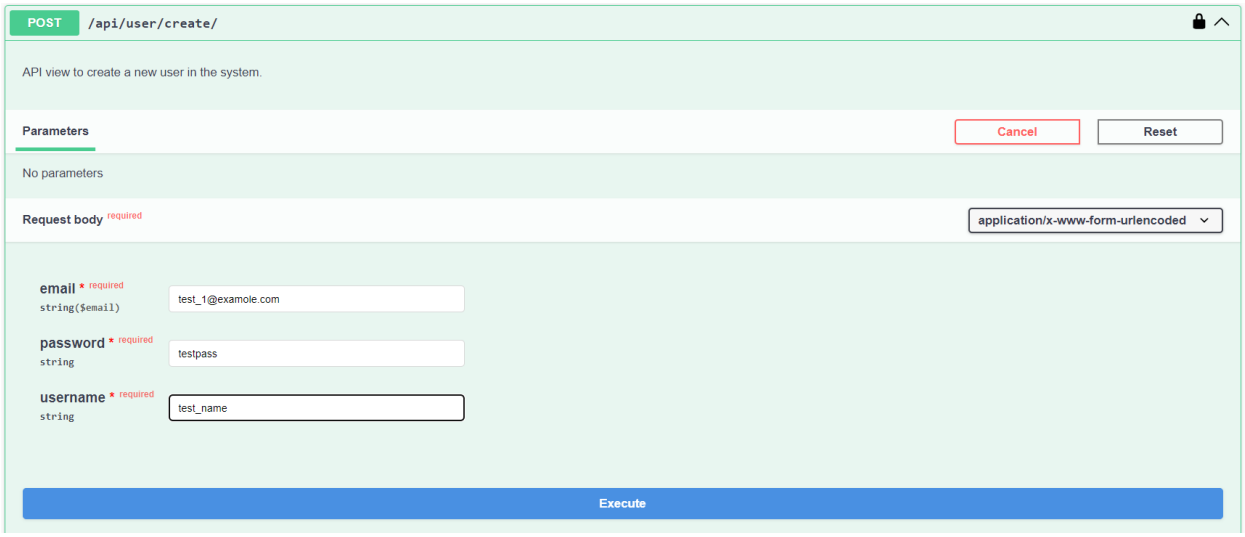
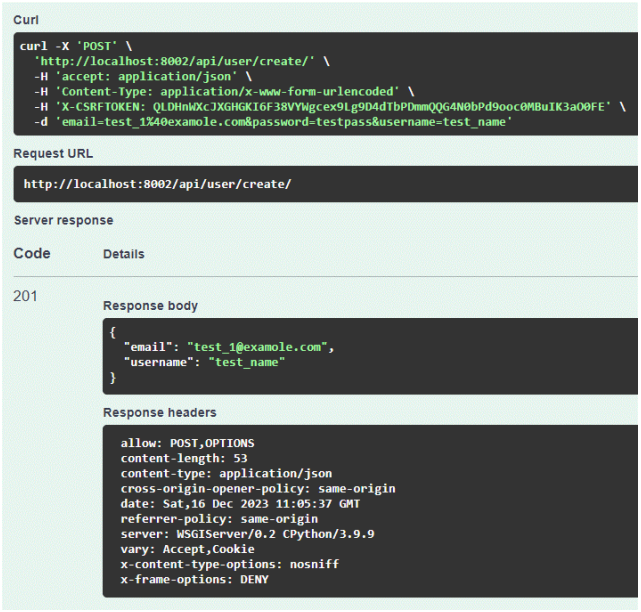


Рисунок 4.26 – Реєстрація нового користувача

Джерело: побудований програмний інтерфейс (знімок з екрану)



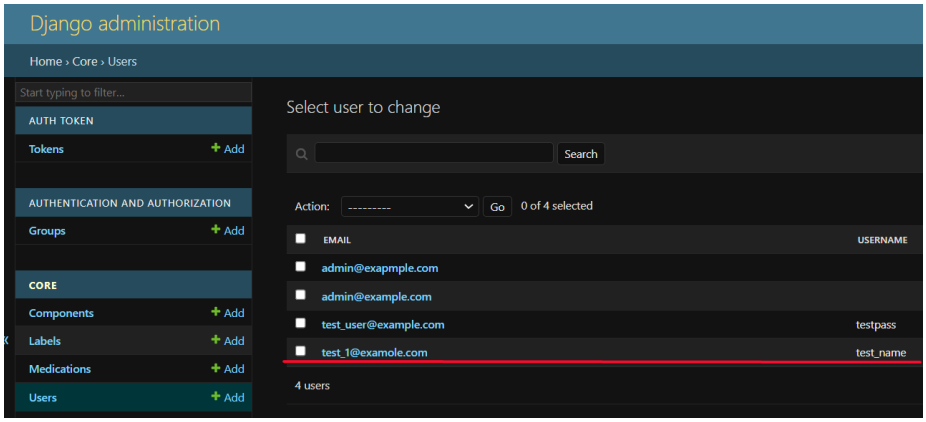


Рисунок 4.27 – Результат реєстрації користувача

Джерело: побудований програмний інтерфейс (знімок з екрану)

Недоступність дій неавторизованим користувачем

1. Відкриваємо інтерфейс для перевірки API Swagger.
2. Використовуючи метод POST, відправляємо запит на отримання списку медичних препаратів за адресою /api/medication/.
3. Перевіряємо, що отримали відповідь зі статусом 401 Unauthorized (рис. 4.28).

Отриманий результат:

– Отримано відповідь зі статусом 401 Unauthorized.

В тілі відповіді може бути повідомлення про те, що необхідна авторизація.

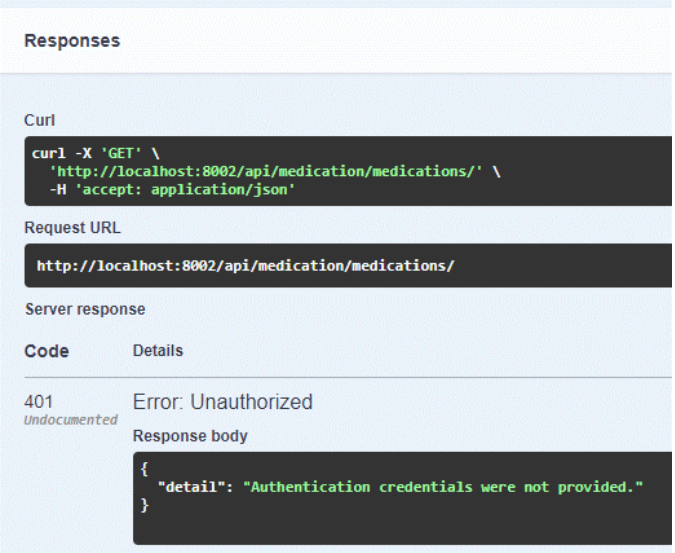


Рисунок 4.28 – Результат виконання запиту без авторизації

Джерело: побудований програмний інтерфейс (знімок з екрану)

Авторизація користувача за допомогою токену авторизації

1. Відкриваємо інтерфейс для перевірки API Swagger.
2. Використовуючи метод POST, відправляємо запит на отримання токену авторизації за адресою /api/user/get-token/.
3. У тілі запиту вводимо дані нового користувача, такі як email та пароль (рис. 4.29).
4. Перевіряємо, що отримали відповідь зі статусом 200 Created, а також токен авторизації (рис. 4.30).
5. Після чого авторизуємось в інтерфейсі (рис. 4.31).

6. Використовуючи метод GET, відправляємо запит на отримання списку медичних препаратів за адресою /api/medication/ (рис. 4.32).

Отриманий результат:

– Запит на отримання токену виконаний успішно.

– Отримано токен авторизації та користувача авторизовано.

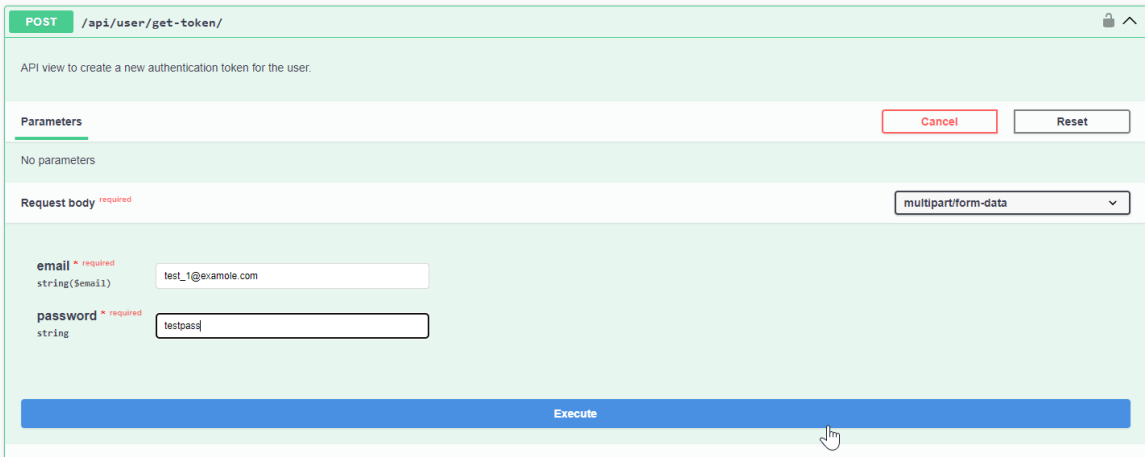


Рисунок 4.29 – Виконання запиту авторизації

Джерело: побудований програмний інтерфейс (знімок з екрану)

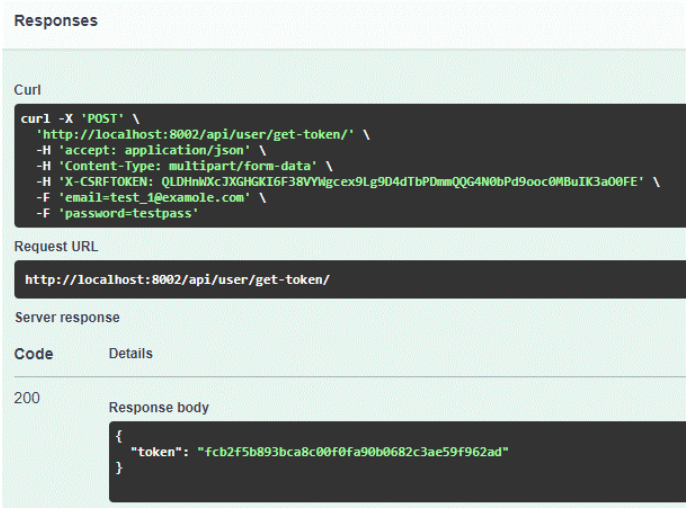


Рисунок 4.30 – Результат виконання запиту авторизації

Джерело: побудований програмний інтерфейс (знімок з екрану)

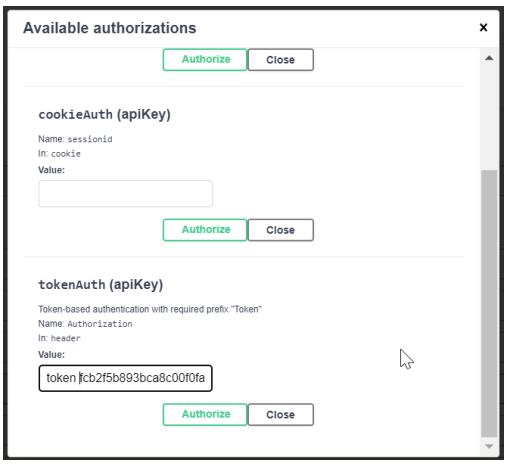


Рисунок 4.31 – Авторизація в інтерфейсі за допомогою токену

Джерело: побудований програмний інтерфейс

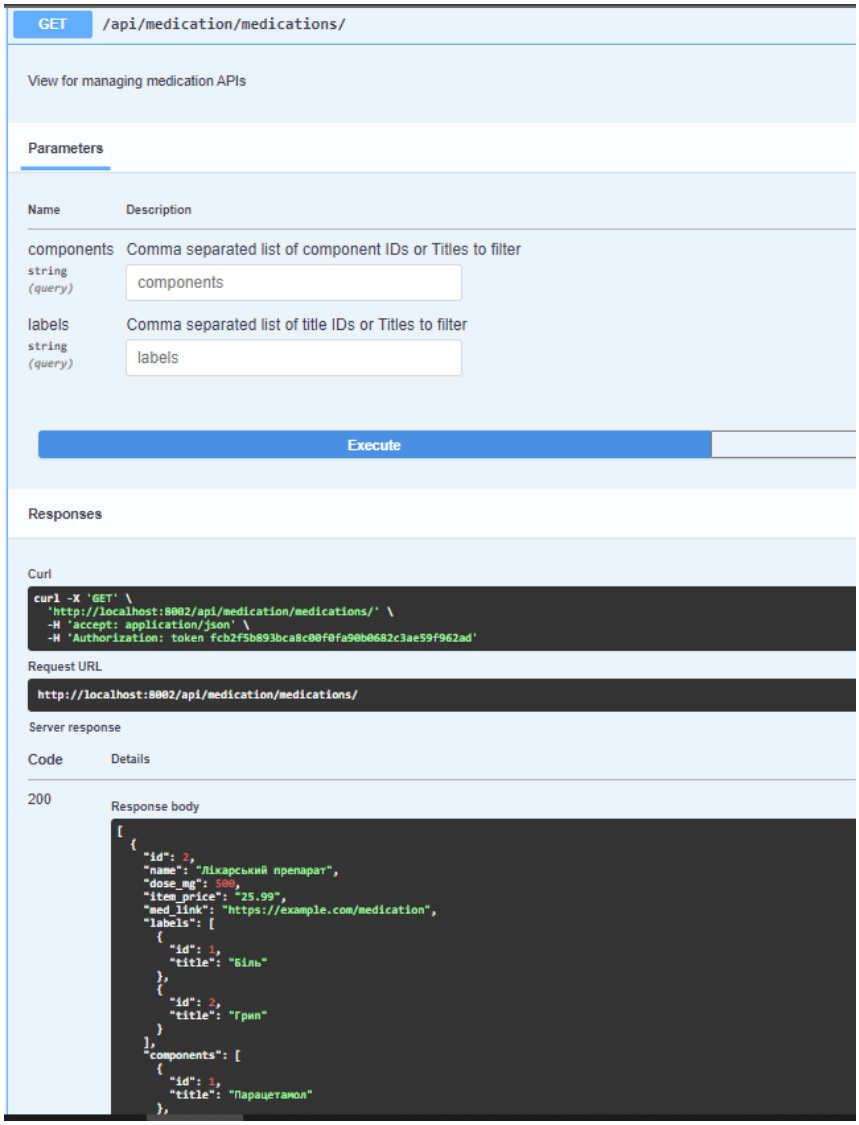


Рисунок 4.32 – Результат виконання запиту авторизованого користувача

Джерело: побудований програмний інтерфейс (знімок з екрану)

Створення медичного препарату

1. Відкриваємо інтерфейс для перевірки API Swagger.
2. Авторизуємось за допомогою токену авторизації.
3. Використовуючи метод POST, відправляємо запит на створення нового медичного препарату за адресою /api/medication/.
4. У тілі запиту вводимо дані для нового медичного препарату, такі як назва, доза, ціна тощо (рис. 4.33).
5. Перевіряємо, що отримали відповідь зі статусом 201 Created, а також відповідні дані про створеного медичного препарату (рис. 4.34).

Отриманий результат:

– Запит на створення медичного препарату виконаний успішно.
– Створено новий медичний препарат у системі.

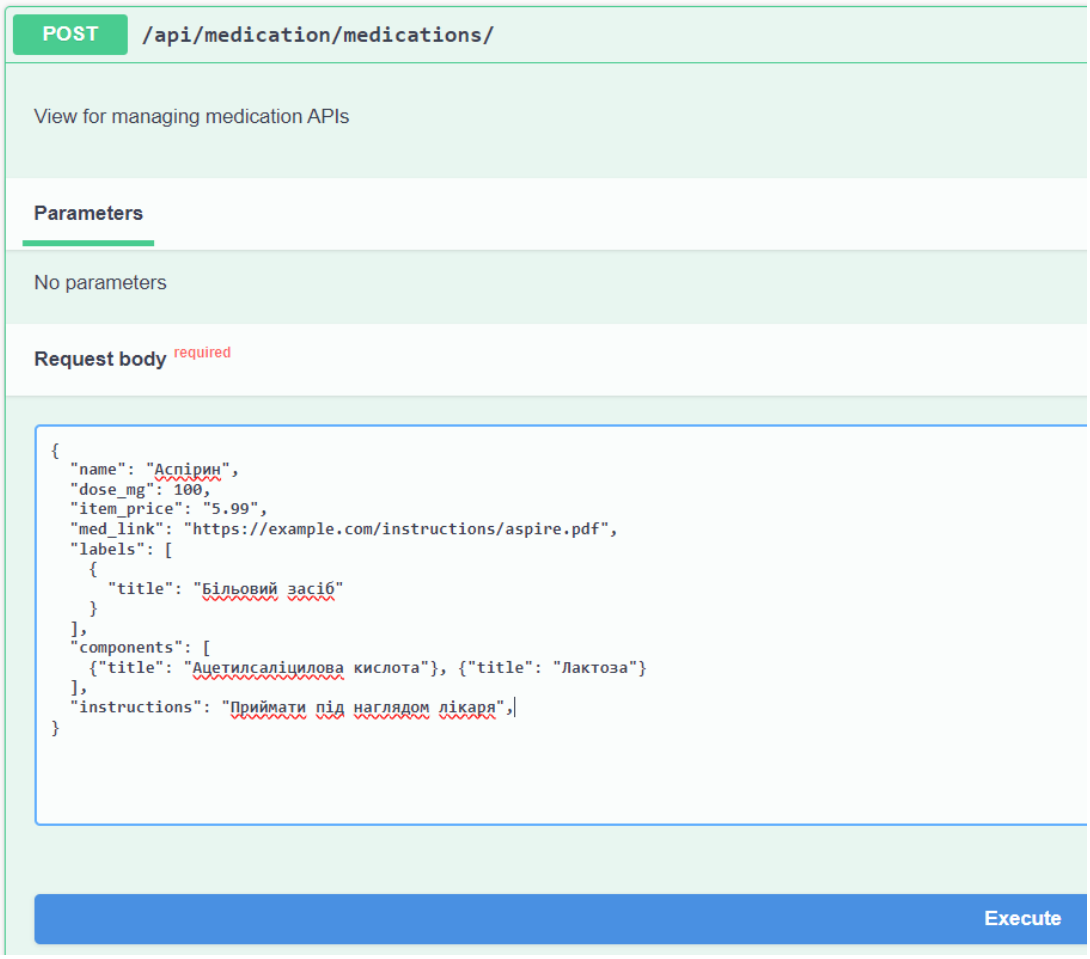


Рисунок 4.33 – Створення медичного препарату

Джерело: побудований програмний інтерфейс (знімок з екрану)

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8002/api/medication/medications/' \
  -H 'accept: application/json' \
  -H 'Authorization: token fcb2f5b893bca8c00f0fa90b0682c3ae59f962ad' \
  -H 'Content-Type: application/json' \
  -H 'X-CSRFTOKEN: pa2N7KzynbpKsxVoyyzFV9m8oKxFP3kbMiAVnaYcuUNQMY2v2TPWXZHqSXr4nUmL' \
  -d '{
  "name": "Аспірин",
  "dose_mg": 100,
  "item_price": "5.99",
  "med_link": "https://example.com/instructions/aspire.pdf",
  "labels": [
    {
      "title": "Більовий засіб"
    }
  ],
  "components": [
    {"title": "Ацетилсаліцилова кислота"}, {"title": "Лактоза"}
  ],
  "instructions": "Приймати під наглядом лікаря"
}'
```

Request URL

```
http://localhost:8002/api/medication/medications/
```

Server response

Code Details

201

Response body


```
{
  "id": 3,
  "name": "Аспірин",
  "dose_mg": 100,
  "item_price": "5.99",
  "med_link": "https://example.com/instructions/aspire.pdf",
  "labels": [
    {
      "id": 3,
      "title": "Більовий засіб"
    }
  ],
  "components": [
    {
      "id": 3,
      "title": "Ацетилсаліцилова кислота"
    },
    {
      "id": 4,
      "title": "Лактоза"
    }
  ],
  "instructions": "Приймати під наглядом лікаря",
  "image": null
}
```

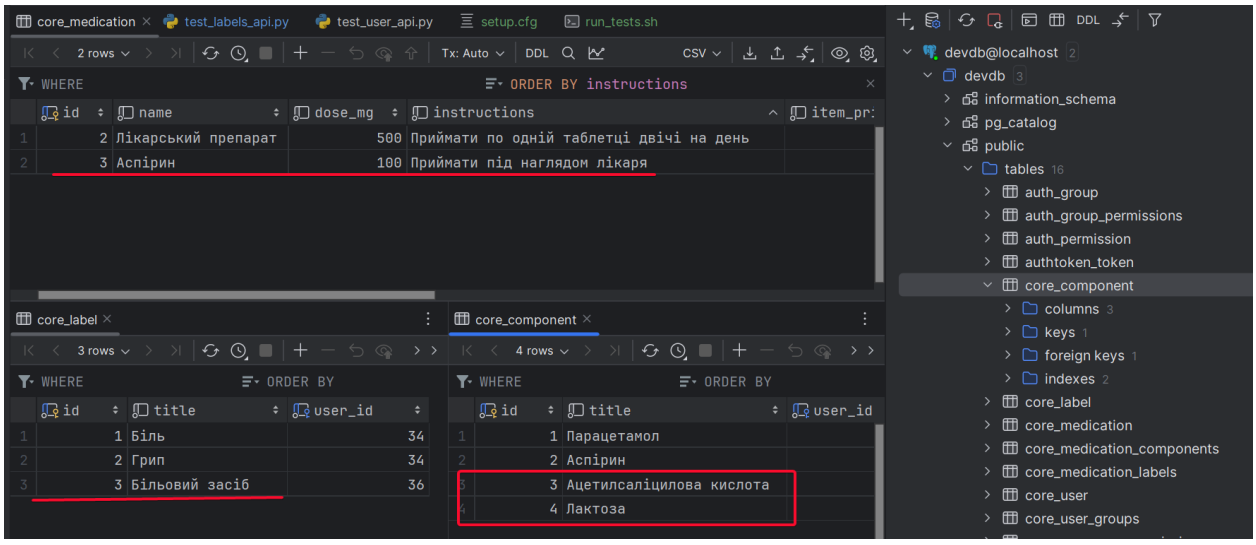



Рисунок 4.34 – Результат створення медичного препарату в інтерфейсі та БД

Джерело: побудований програмний інтерфейс (знімок з екрану)

Внесення змін у записи каталогу

1. Відкриваємо інтерфейс для перевірки API Swagger.
2. Авторизуємось за допомогою токену авторизації.
3. Використовуючи метод PATCH, відправляємо запит на оновлення існуючого медичного препарату за адресою /api/medication/{id}/, де {id} - ідентифікатор медичного препарату.
4. У тілі запиту вводимо дані для оновлення медичного препарату (рис. 4.35).
5. Перевіряємо, що отримали відповідь зі статусом 200, а також відповідні дані оновленого медичного препарату (рис. 4.36).

Отриманий результат:

– Запит на оновлення медичного препарату виконаний успішно.
– Зміни внесено у запис каталогу.

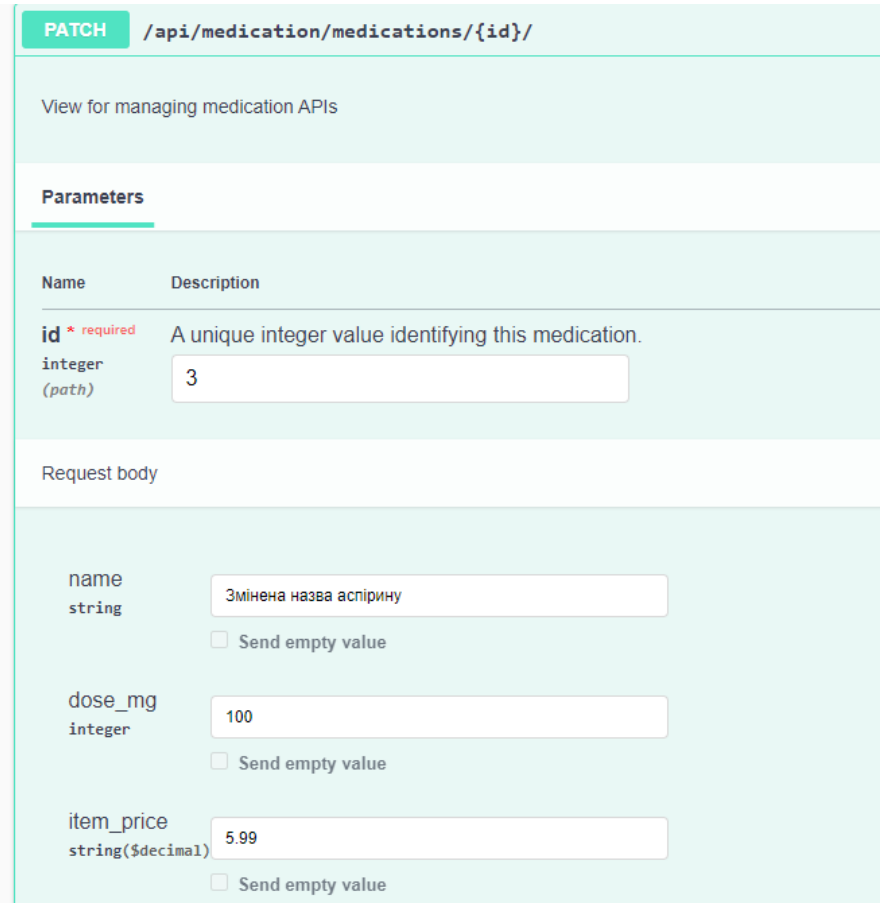


Рисунок 4.35 – Оновлення медичного препарату

Джерело: побудований програмний інтерфейс (знімок з екрану)

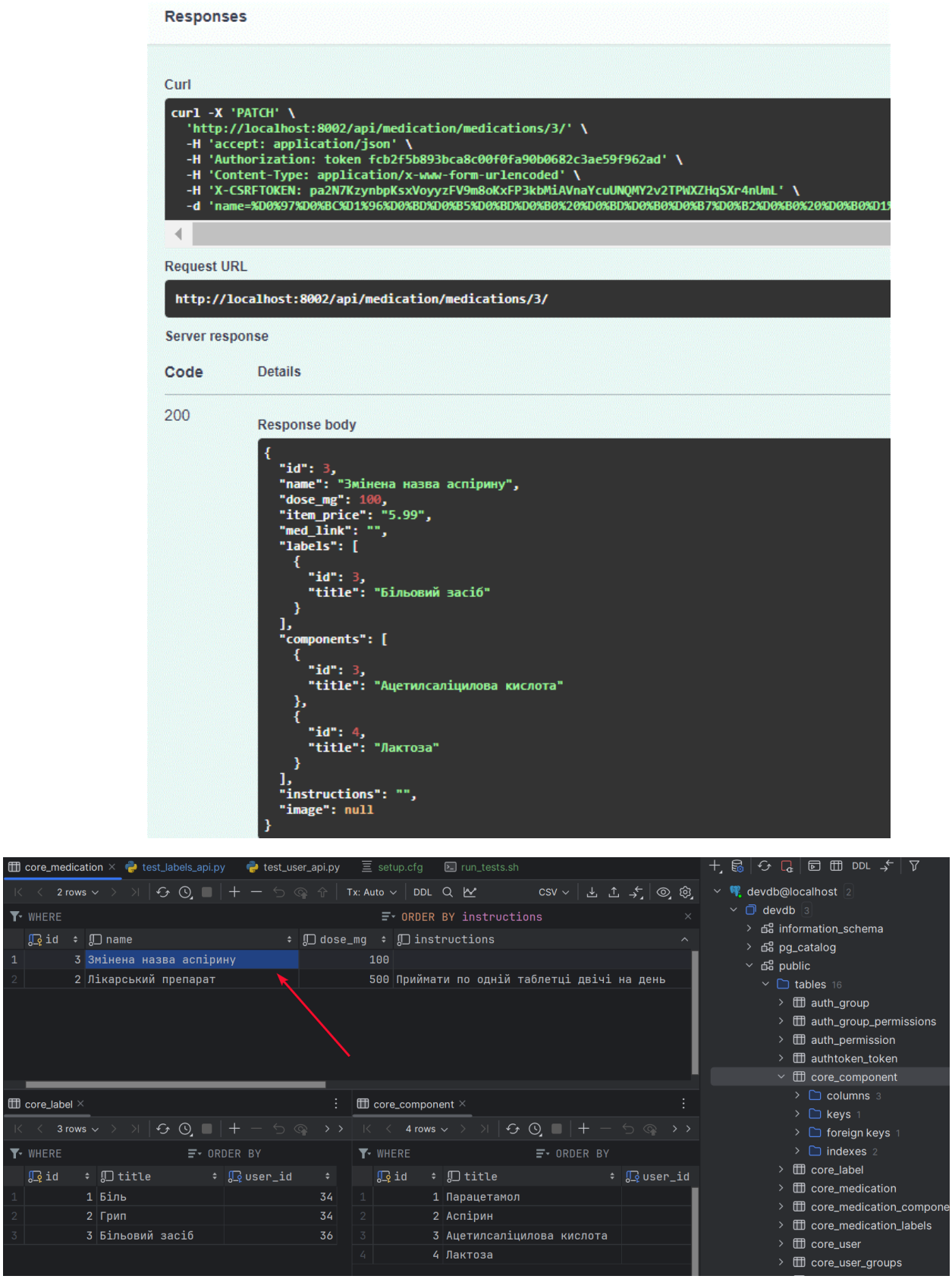


Рисунок 4.36 – Результат Оновлення медичного препарату в інтерфейсі та БД

Джерело: побудований програмний інтерфейс (знімок з екрану)

Додавання зображення препарату

1. Відкриваємо інтерфейс для перевірки API Swagger.
2. Авторизуємось за допомогою токену авторизації.
3. Використовуючи метод POST, відправляємо запит на додавання зображення до медичного препарату за адресою /api/medication/medications/{id}/upload-image/ де {id} - ідентифікатор медичного препарату.
4. У тілі запиту передаємо дані зображення дані для додавання до медичного препарату. Для перевірки скористаємось завантаженням зображення засобами web-браузера (рис. 4.37).
5. Перевіряємо, що отримали відповідь зі статусом 200, а також відповідні дані оновленого медичного препарату (рис. 4.38-4.39).

Отриманий результат:

– Запит на додавання зображення виконаний успішно.
– Зміни внесено у запис каталогу.

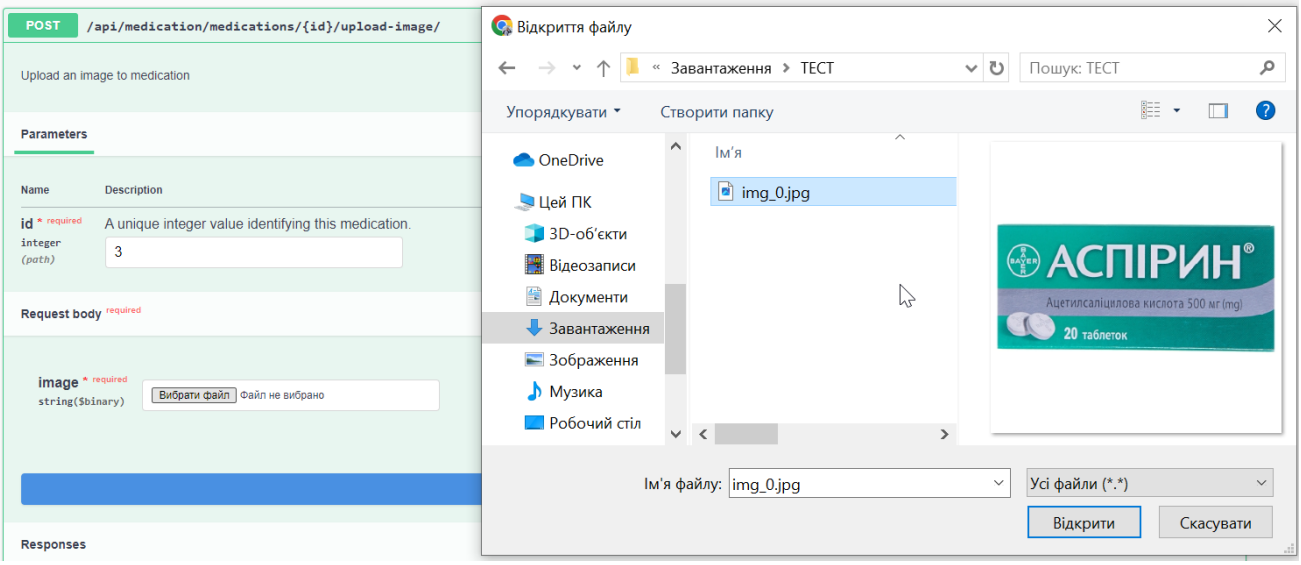


Рисунок 4.37 – Додавання зображення медичного препарату

Джерело: побудований програмний інтерфейс (знімок з екрану)

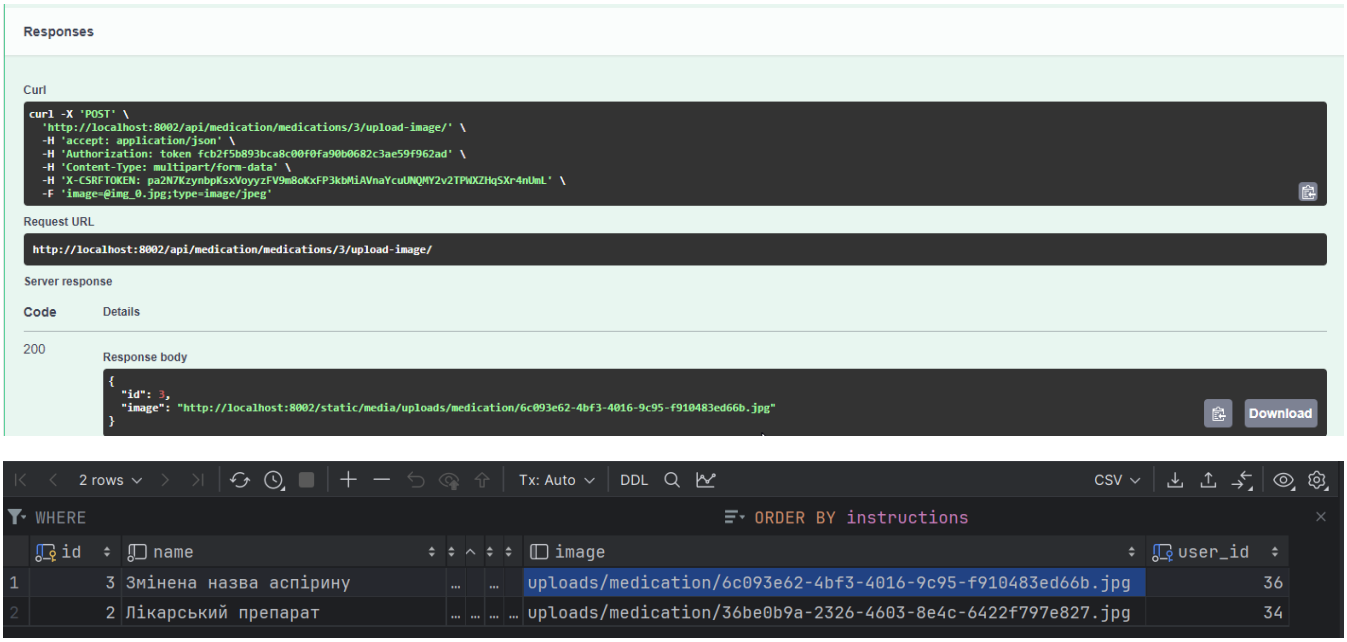


Рисунок 4.38 – Результат Оновлення медичного препарату в інтерфейсі та БД

Джерело: побудований програмний інтерфейс (знімок з екрану)

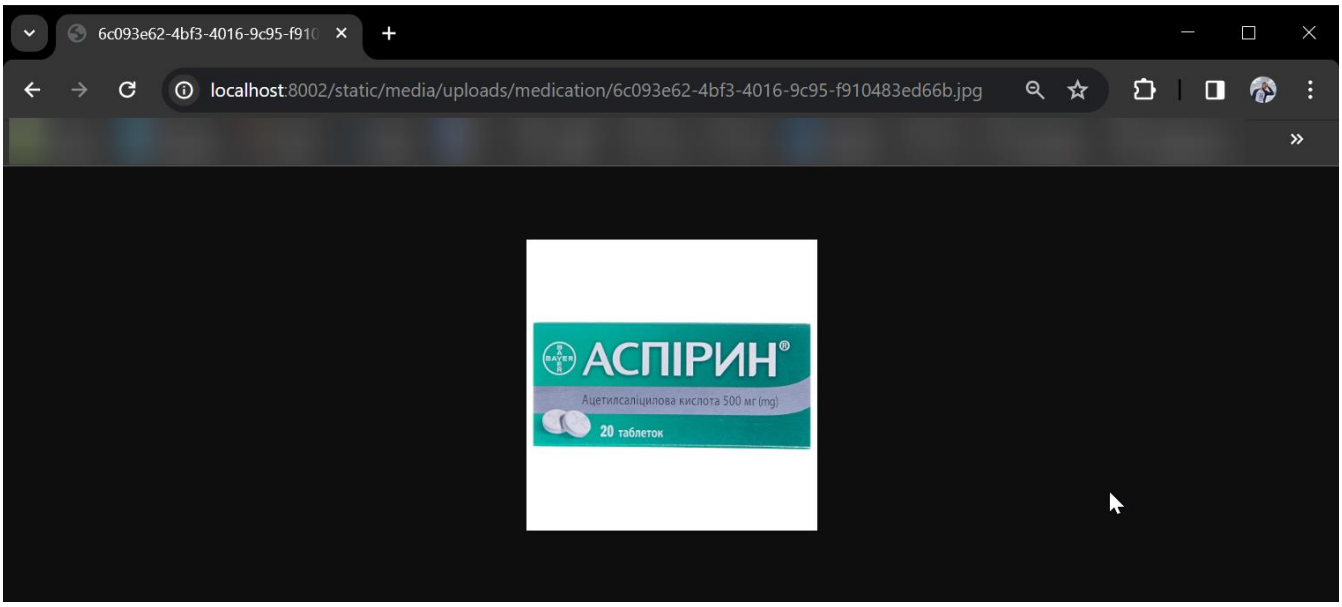


Рисунок 4.39 – перевірка доступу до зображення за створеним UUID

Джерело: побудований програмний інтерфейс (знімок з екрану)

Фільтрація каталогу за тегами

1. Відкриваємо інтерфейс для перевірки API Swagger.
2. Авторизуємось за допомогою токену авторизації.

3. Використовуючи метод Використовуючи метод GET, відправляємо запит на отримання списку медичних препаратів за адресою /api/medication/.
4. Додаємо параметри запиту для фільтрації за тегами (рис. 4.40)., наприклад, api/medication/?labels=%D0%93%D1%80%D0%B8%D0%BF (у вказаному прикладі автоматично енкодовані символи слова "Грип").
5. Перевіряємо, що отримали відповідь зі статусом 200, а також відповідні дані медичного препарату (рис. 4.41).

Отриманий результат:

– Запит на отримання списку медичних препаратів з фільтрацією виконаний успішно.
– Отримано список медичних препаратів, які відповідають заданим тегам.

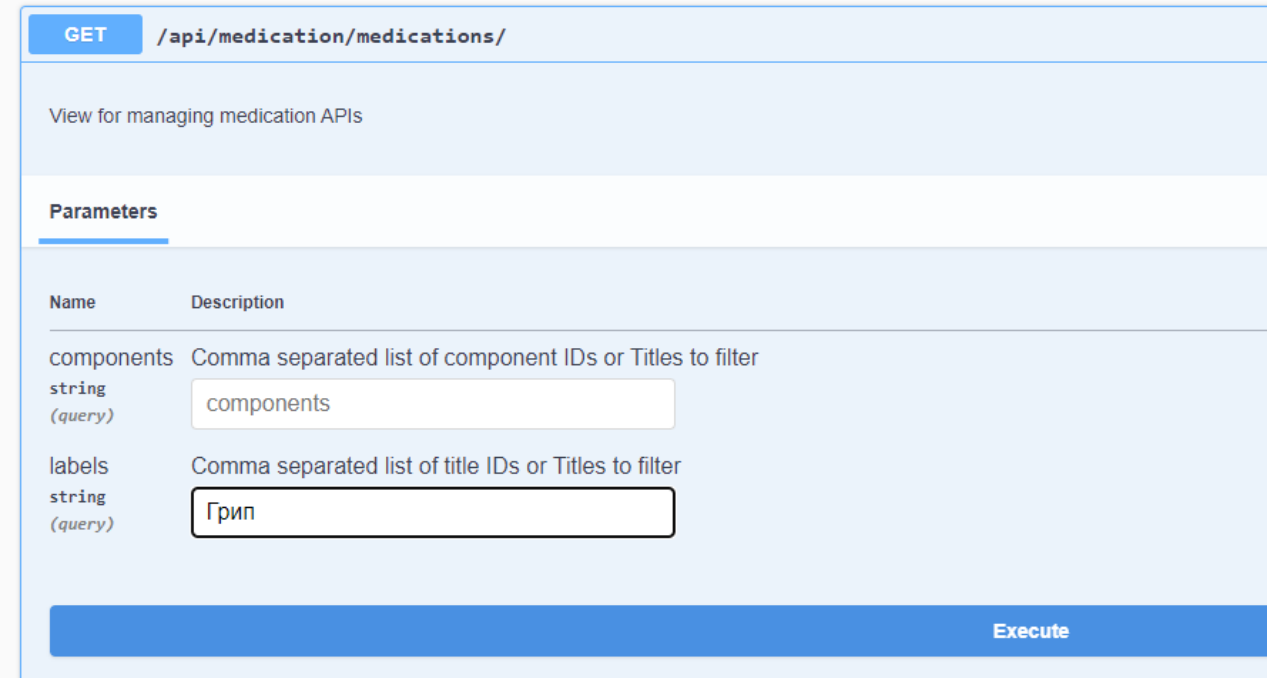


Рисунок 4.39 – Пошук медичного препарату за фільтрами

Джерело: побудований програмний інтерфейс (знімок з екрану)

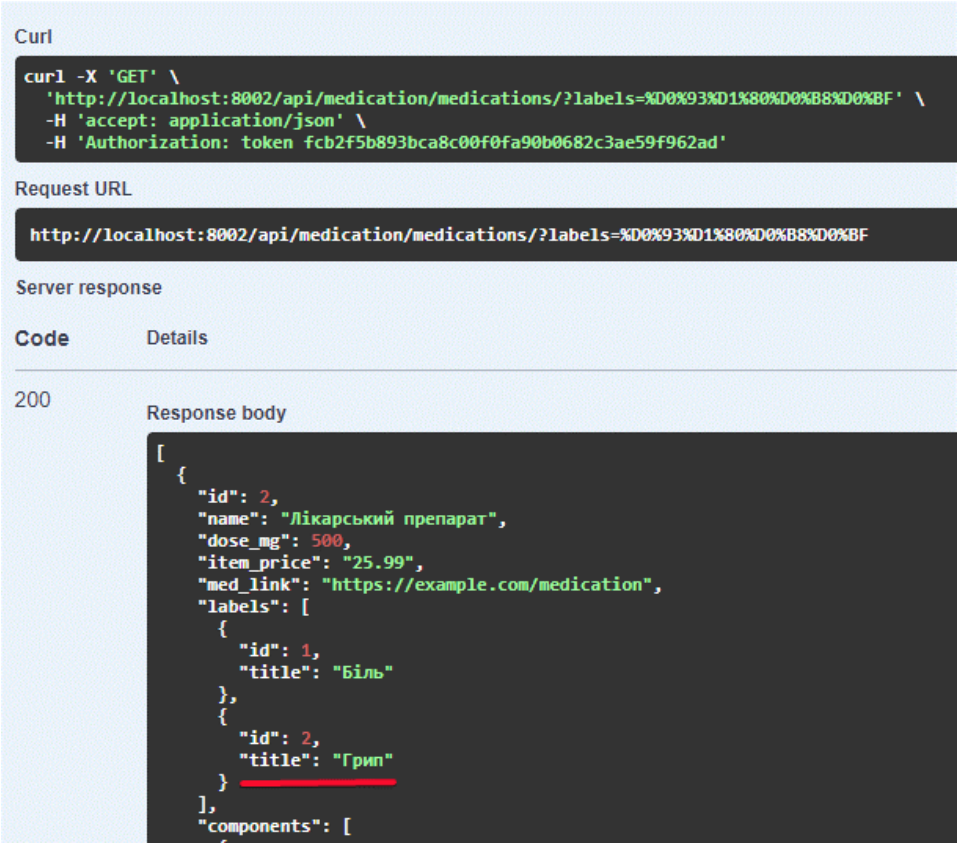


Рисунок 4.40 – Результат Оновлення медичного препарату в інтерфейсі та БД

Джерело: побудований програмний інтерфейс (знімок з екрану)

Мануальне тестування програмного інтерфейсу дозволило впевнитися в правильній роботі ключових функцій. Успішно перевірена можливість реєстрації та авторизації користувачів, коректність створення, редагування та фільтрації медичних препаратів через API. Також, тестове завантаження та обробка зображень пройшли успішно. Крім того, не було виявлено випадків, коли неавторизований користувач міг би викликати кінцеві точки, доступні лише для авторизованих користувачів.

На підставі проведеного мануального тестування можна зробити висновок, що програмний інтерфейс відповідає вимогам, працює стабільно і може бути безпечно впроваджений в експлуатацію.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було виконано проектування і реалізація програмного інтерфейсу формування каталогу медичних препаратів.

Для забезпечення підгрунтя проектування було проведено аналіз публікацій та предметної області, а також виконано огляд та порівняння існуючих каталогізаторів медичних препаратів. Цей аналіз визначив ключові вимоги до нової системи, що були успішно враховані при розробці програмного інтерфейсу.

За допомогою Use Case та IDEF0 діаграм було виконано проектування функціональних можливостей системи. Особлива увага приділялася забезпеченню зручного використання API для користувачів. Виконана програмна реалізація всіх функцій, які визначено в постановці задачі.

Спроектований програмний інтерфейс представляє значущу цінність для розробників інформаційних технологій додатків медичної галузі, оскільки дозволяє спростити процес розробки каталогів та персоналізувати доступ до різноманітної інформації про медичні препарати, їх характеристики, взаємодію та застосування. Відкриті структуровані API дозволяють з незначними витратами часу інтегрувати каталог з даними про препарати в інші інформаційні системи, які використовує користувач.

Мануальне тестування підтвердило якість та надійність програмного інтерфейсу. Усі основні функції були успішно протестовані і працювали коректно.

Таким чином можна впевнено стверджувати, що програмний інтерфейс відповідає вимогам, працює стабільно та надійно. Цей інтерфейс готовий до безпечного впровадження в експлуатацію, надаючи зручний та ефективний інструмент для управління медичними препаратами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Губар М.С. Розкриття понять «аптечна мережа» та «консолідація». Огляд світових фармацевтичних ринків, прогноз темпів росту європейського фармацевтичного ринку // Пріоритетні напрями розвитку науки, LXIV Міжнародна науково-практична інтернет-конференція – м. Вінниця, 5 квітня 2021 року – Ч.1 – с. 112-118.
2. Гайденко А.О., Томашук І.В. Економічна ефективність виробництва фамацевтичних препаратів і матеріалів в Україні // Економіко-правові аспекти господарювання: сучасний стан, ефективність та перспективи: матеріали VIII Міжнародної науково-практичної конференції – Одеса, 23-24 вересня 2022 р – Одеса: 2022 – с. 57-59.
3. N. Y. Stadnytska, Z. D. Parashchyn, I. P. Lobur, і I. V. Fito, АНАЛІЗ ВІТЧИЗНЯНОГО РИНКУ ЛІКАРСЬКИХ ЗАСОБІВ, ЯКІ ЗАСТОСОВУЮТЬСЯ ПРИ ЗАХВОРЮВАННЯХ ПОРОЖНИНИ НОСА – Фармацевтичний часопис. 2, 2021 – с. 36–43.
4. Allen, L.V. The Art, Science, and Technology of Pharmaceutical Compounding, Third Edition – Washington, DC, American Pharmacists Association. XXVIII : 2008 – 556 p.
5. Дегтярьова Л.М. Аналіз використання автоматизованих системи в фармацевтиці / Л.М. Дегтярьова, Є.Ю. Мачула // Тези 71-ої нау-кової конференції професорів, викладачів, наукових працівників, аспірантів та студентів університету (Полтава, 22 квітня – 17 травня 2019 р.). – Полтава : ПолтНТУ, 2019. – Т. 1. – С. 350-351
6. Костюк Г. В., Коваленко А. В. Конкурентоспроможність фармацевтичної промисловості України. Ефективна економіка. 2013. № 11. [Електронний ресурс] – Режим доступу до ресурсу: http://www.economy.nayka.com.ua/?op=1&z=2547

7. O'Kane A, Francis W, Duffy C, et al. Implementing an automated dispensing system for the safe management of controlled drugs. Hospital Pharmacy Europe. 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://hospitalpharmacyeurope.com/news/reviews-research/implementing-an-automated-dispensing-system-for-the-safe-management-of-controlled-drugs/
8. Agustini, K., Hurriyati, R., Gaffar, V., Tiana, B., & Novianti, (2022) W. The effectiveness of electronic purchase on ordering national health insurance drugs at the West Bandung pharmacy of Indonesia. – Journal of Eastern European and Central Asian Research (JEECAR)
9. Al-Qirim, N. The role of information systems in supporting knowledge management: The case of a pharmaceutical company in Jordan. – Information & Management, 43(3), 2006 – p. 366-376
10. Darmawan, E., & Rumambi, (2011) T. Drugs Catalog Application on Abc Pharmacies Using MICROSOFT VISUAL BASIC 6.0.
11. Bu, F., Sun, H., Li, L., Tang, F., Zhang, X., Yan, J., Ye, Z., & Huang, T. (2022). Artificial intelligence-based internet hospital pharmacy services in China: Perspective based on a case study. Frontiers in Pharmacology, 13.
12. Dennehy, D., Griva, A., Pouloudi, N., Dwivedi, Y.K., Mäntymäki, M., & Pappas, I.O. (2022). Artificial Intelligence (AI) and Information Systems: Perspectives to Responsible AI. Information Systems Frontiers, 25, 1-7.
13. Шуть В.С. Інтелектуальні системи управління в фармацевтиці // The 17th International scientific and practical conference "System analysis and intelligent systems for management" (May 02 International Science Group. 2023. 482 p. - 05, 2023) Ankara, Turkey – с. 273-274.
14. Рудніченко М., Синявський О., Петров І. Проект системи автоматизації розпізнавання медичних ліків засобами машиного навчання // «Інформаційні управляючі системи і технології» (ІУСТ-ОДЕСА-2023). Матеріали XI Міжнародної науковопрактичної конференціі, 21 - 23 вересень 2023 р. Одеса / вип. ред. В.В. Вичужанін, 2023 – с. 195-197.

15. Спеціалізоване медичне інтернет-видання для лікарів, провізорів, фармацевтів, студентів медичних і фармацевтичних вишів «Копендіум» [Електронний ресурс] – Режим доступу до ресурсу: https://compendium.com.ua/uk/library/
16. Компендіум 2019 — лікарські препарати / За ред. В.М. Коваленка — К.: МОРІОН, 2019. — 2480 с.
17. Державний реєстр лікарських засобів України [Електронний ресурс] – Режим доступу до ресурсу: http://www.drlz.com.ua/ibp/ddsite.nsf/all/index?opendocument
18. Пілюшенко В.Л., Шкрабак І.В., Славенко Е.І. Наукове дослідження: організація, методологія, інформаційне забезпечення: Навчальний посібник. – Київ: Лібра, 2004. – С. 52-60.
19. Стеченко Д.М., Чмир О.С. Методологія наукових досліджень: Підручник: - К.: Знання, 2005. – С. 91-190
20. Саварин П. Електронний посібник із дисципліни "Комп’ютерні системи та мережі". Лекція 6. Нотація IDEF0 // Луцький національний технічний університет, Факультет цифрових, освітніх та соціальних технологій [Електронний ресурс] – Режим доступу до ресурсу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/%D0%9A%D0%BE%D0%BD%D0%B4%D1%96%D1%83%D1%81%202%20%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%B2%D0%B0/page9.html
21. Основи UML [Електронний ресурс] – Режим доступу до ресурсу: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-basics.html
22. Use Case Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/use-case-diagram/
23. Python [Електронний ресурс] — Режим доступу до ресурсу: https://www.python.org/.
24. IEEE Spectrum опублікувала рейтинг найпопулярніших мов програмування 2023 року [Електронний ресурс] — Режим доступу до ресурсу:

https://speka.media/ieee-spectrum-opublikuvala-reiting-naipopulyarnisix-mov-programuvannya-2023-roku-pn3l49

25. What is Python Used For? 7 Real-Life Python Uses [Електронний ресурс] — Режим доступу до ресурсу: https://www.datacamp.com/blog/what-is-python-used-for
26. Advantages and Disadvantages of Python – Make a Favorable Decision [Електронний ресурс] — Режим доступу до ресурсу: https://squareboat.com/blog/advantages-and-disadvantages-of-python
27. Understanding the Python GIL [Електронний ресурс] — Режим доступу до ресурсу: https://www.dabeaz.com/python/UnderstandingGIL.pdf
28. Django documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.djangoproject.com/
29. 10 Advantages of using Django for web development [Електронний ресурс] — Режим доступу до ресурсу: https://inoxoft.com/blog/10-advantages-of-using-django-for-web-development/
30. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс] — Режим доступу до ресурсу: https://www.postgresql.org/
31. Python [Електронний ресурс] — Режим доступу до ресурсу: https://www.guru99.com/introduction-postgresql.html
32. What is PostgreSQL? Introduction, Advantages & Disadvantages [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/.
33. Docker Containerization and Virtualization [Електронний ресурс] — Режим доступу до ресурсу: https://medium.com/@haticeyildiz/docker-containerization-and-virtualization-benefits-advantages-and-disadvantages-737b31b86213
34. Python and REST APIs: Interacting With Web Services [Електронний ресурс] — Режим доступу до ресурсу: https://realpython.com/api-integration-in-python/
35. G. G. Lendaris, Structural Modeling a Tutorial Guide – IEEE Transactions on Systems, Man, and Cybernetics, Dec. 1980 – vol. 10, no. 12, pp. 807-840

36. Конспект лекцій з дисципліни «Бізнес-процеси в кібербезпеці» для другого (магістерського) рівня вищої освіти спеціальності 125 – Кібербезпека / Укл.: Лебедєва О.Ю. — Одеса, 2021. — 115 с
37. What is Use Case Diagram? [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
38. Use Case Diagram Tutorial (Guide with Examples) [Електронний ресурс] – Режим доступу до ресурсу: https://creately.com/blog/diagrams/use-case-diagram-tutorial/.
39. Swagger. API Development for Everyone [Електронний ресурс] – Режим доступу до ресурсу: https://swagger.io/solutions/api-design/
40. Drf-spectacular documentation [Електронний ресурс] – Режим доступу до ресурсу: https://drf-spectacular.readthedocs.io/en/latest/
41. Visualize databases with DBeaver [Електронний ресурс] – Режим доступу до ресурсу: https://dbeaver.com/2022/06/09/visualize-data-with-dbeaver/
42. Ganttpro - Online Gantt chart maker for project management [Електронний ресурс] – Режим доступу до ресурсу: https://ganttpro.com/en/

ДОДАТОК А

Планування робіт

Щорічно на фармацевтичному ринку реєструються нові препарати, що потрапляють у медичний обіг. Враховуючи те, що препарати взаємодіють із людським організмом, брак інформації щодо них може призвести до значних наслідків. Через це завжди стоїть питання формування каталогів препаратів, їх своєчасного оновлення та можливості адміністрування. Завдяки можливості використання завчасно написаного структурованого каталогізатора медичних препаратів можна скоротити час на розробку систем обліку, додавання нових елементів, структуризації існуючих карток і навіть адміністрування доступів.

Деталізація мети методом SMART. Продуктом дипломного проекту є програмний інтерфейс формування каталогу медичних препаратів. на етапі концептуального проектування – це ключ до успіху всього проекту. Результати деталізації мети проекту методом SMART представлено у таблиці А.1

Таблиця А.1 – Деталізація мети проекту методом SMART

Specific (Конкретна)	Розробити та впровадити програмний інтерфейс формування каталогу медичних препаратів із відкритим API для легкого доступу та пошуку інформації.
Measurable (Вимірювана)	Результатом роботи проекту є розроблений контейнеризований програмний інтерфейс формування і адміністрування каталогу медичних препаратів із документованим.
Achievable (Досяжна)	Для виконання програмного інтерфейсу і отримання позитивного результату роботи необхідні знання мови програмування Python, фреймворку Django, інструменту контейнеризації Docker, системи управління даними застосовано PostgreSQL, та навичок проектування та написання документації до API.

Продовження таблиці А.1

Relevant (Реалістична)	Розроблений програмний інтерфейс дозволить скоротити час на розробку сайтів, систем обліку та адміністративних систем завдяки заздалегідь наявній інфраструктурі та розгорнутій БД універсальної структури із різними рівнями доступу.
Time-framed (Обмежена у часі)	Ціль проєкту має часове обмеження, яке визначено за домовленістю між замовником та виконавцем і становить два місяці. Робота повинна бути завершена у визначені терміни, які були оговорені замовником проєкту та відповідають календарному плану проекту.

Планування змісту робіт. Work Breakdown Structure (WBS) – це графічна ієрархічна структура елементів проекту, яка організована з метою планування та контролю робіт. На найвищому рівні розташований сам проект, вище якого розташований продукт проекту. На другому рівні розташовані основні заходи для досягнення мети проєкту. Декомпозиція робіт проводиться до тих пір, поки вони не стають елементарними, тобто простими і однозначними діями з визначеним результатом. Елементарні роботи мають відповідального виконавця, та для них можна обчислити терміни виконання та витрати праці. WBS служить основою для детальної оцінки термінів, контролю та графіків роботи, а також визначає каркас для подальшого планування та розподілу завдань між командою проєкту.

На рисунку Б.1 представлено WBS з розробки програмного інтерфейсу формування каталогу медичних препаратів.

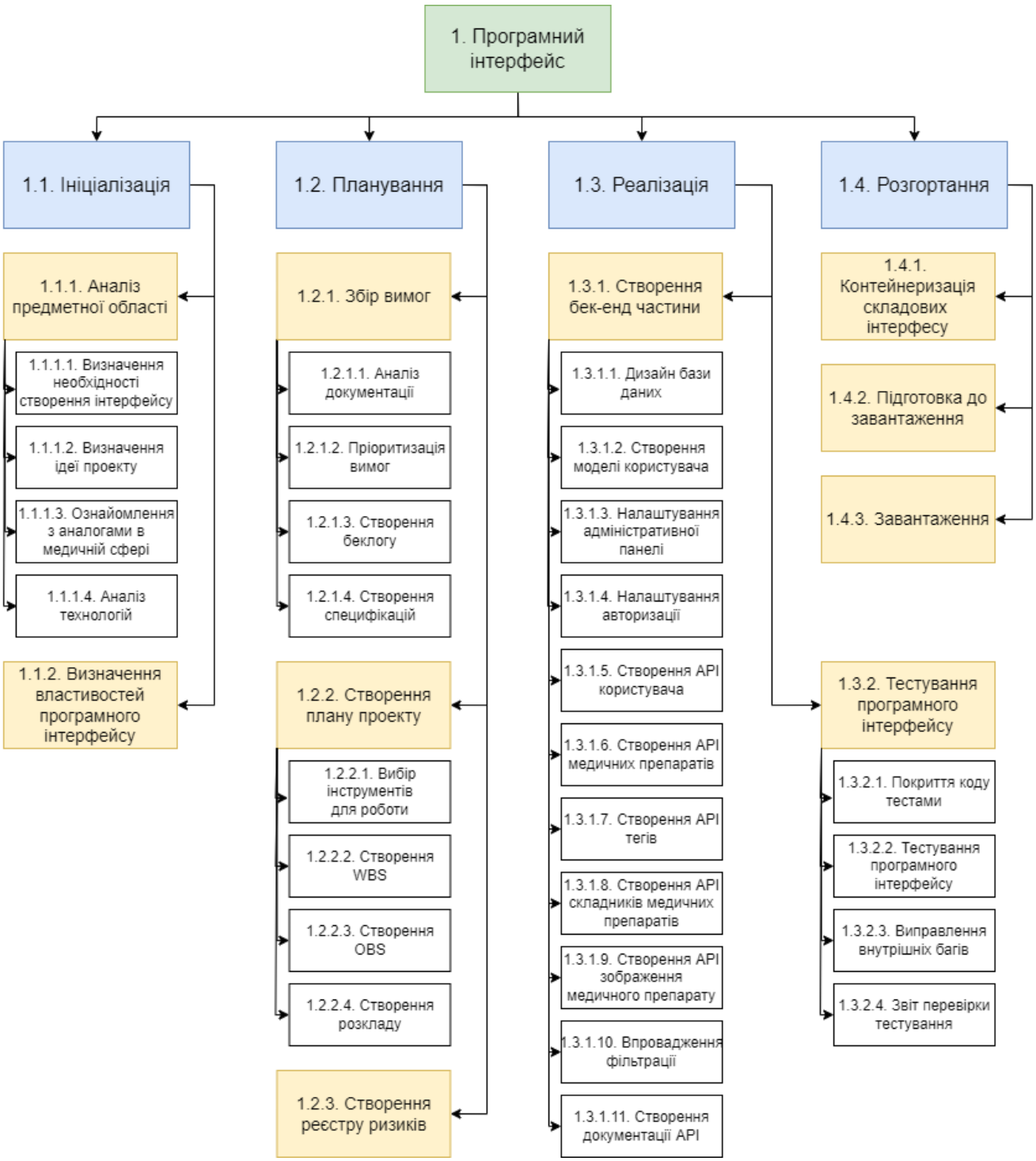


Рисунок Б.1 – WBS-структура робіт проекту

Джерело: побудовано автором на основі планування

Планування структури виконавців. Розробка організаційної структури виконавців або OBS є наступним етапом після декомпозиції процесів. Organizational Breakdown Structure (OBS) або структура розподілу виконавців є графічним відображенням відповідальних осіб, задіяних у розробці проекту та призначених для

виконання елементарних робіт. Організаційна структура виконавців визначається після декомпозиції процесів і визначає організаційні зв'язки та відповідальності. На відміну від організаційної структури проекту, OBS стосується внутрішніх відносин і не включає зв'язки з батьківськими задачами.

У контексті OBS, виконавці виступають у ролі відповідальних осіб, що керують та виконують елементарні роботи, визначені в розробленій WBS. Кожен елементарний проект розглядається як окрема задача, де виконавці взаємодіють із зазначеними ролями та функціями для досягнення цілей проекту. OBS надає чіткий зоровий контур відповідальностей та взаємодій між учасниками проєкту. На рисунку Б.2 представлено організаційну структуру планування робіт проекту. Учасників проекту описано у таблиці А.2.

Таблиця А.2 – Виконавці проекту

Роль	Ім’я	Проектна роль
Розробник	Сірик М.О.	Відповідає за написання програмного коду та реалізацію функціональності проекту.
Проектувальник	Сірик М.О.	Займається проектуванням архітектури системи та визначенням технічних аспектів реалізації.
Тестувальник	Сірик М.О.	Відповідає за визначення та виправлення помилок, а також перевірку відповідності вимогам та специфікаціям.
Керівник проекту	Ващенко С.М.	Забезпечує ефективне керівництво та координацію всіма аспектами проекту.
Менеджер проекту	Сірик М.О.	Відповідає за планування, виконання та контроль всіх етапів проекту.

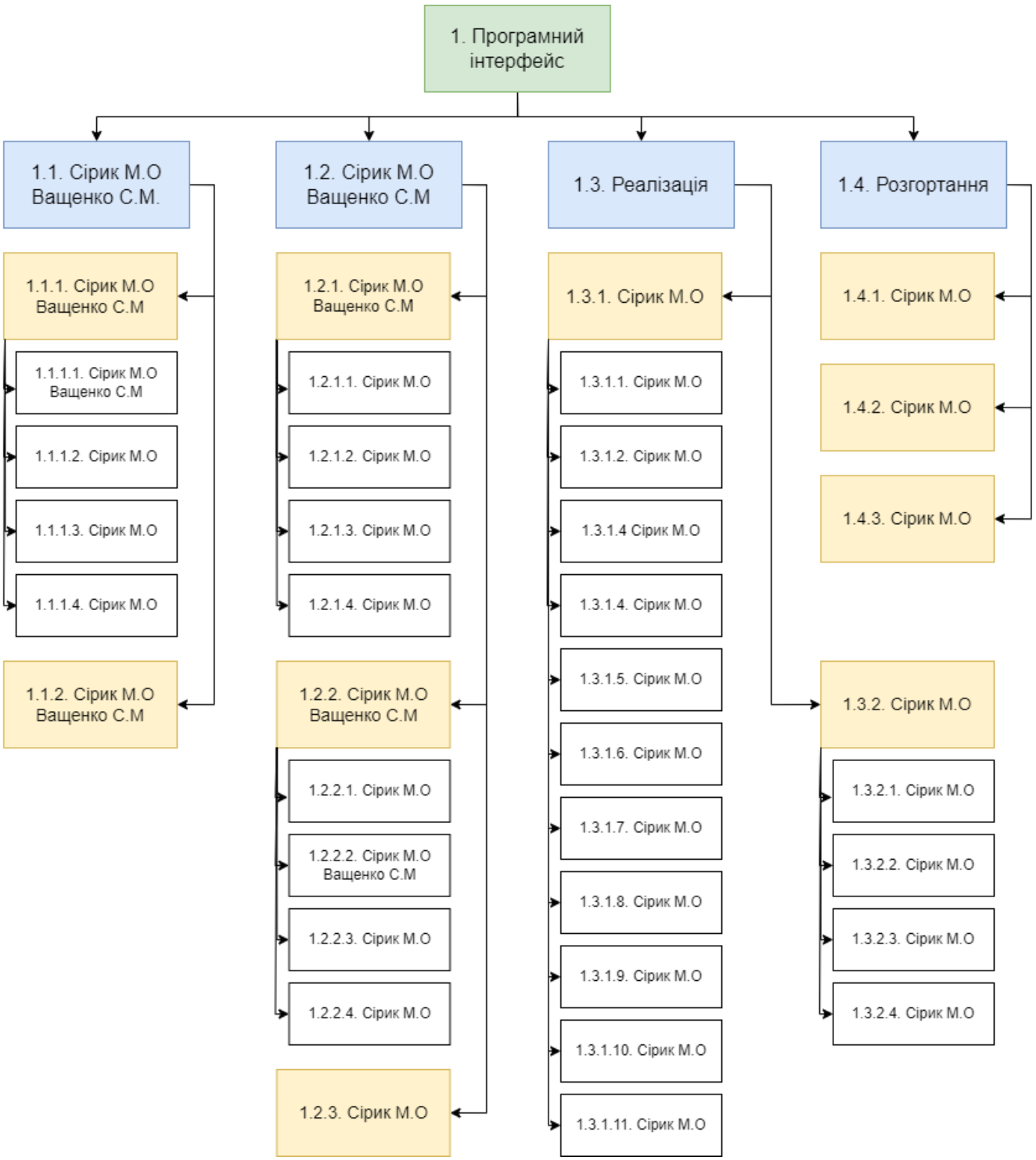


Рисунок А.2 – OBS-структура робіт проекту

Джерело: побудовано автором на основі планування

Діаграма Ганта. Діаграма Ганта є важливим інструментом для побудови календарного плану проекту та відображення графіка виконання робіт з реальним розподілом дат. Цей формат графіка, який часто використовується у всіх галузях,

заснований на гістограмах і дозволяє керівникам проектів відстежувати відсоток завершення кожного завдання. Діаграма Ганта особливо ефективна для проектів з обмеженою кількістю взаємопов'язаних завдань. У даному проекті була розроблена діаграма Ганта, що графічно відображає тривалість кожного процесу, визначеного на етапі формування WBS.

Календарний графік проекту представлено на рисунках А.3 – А.4.

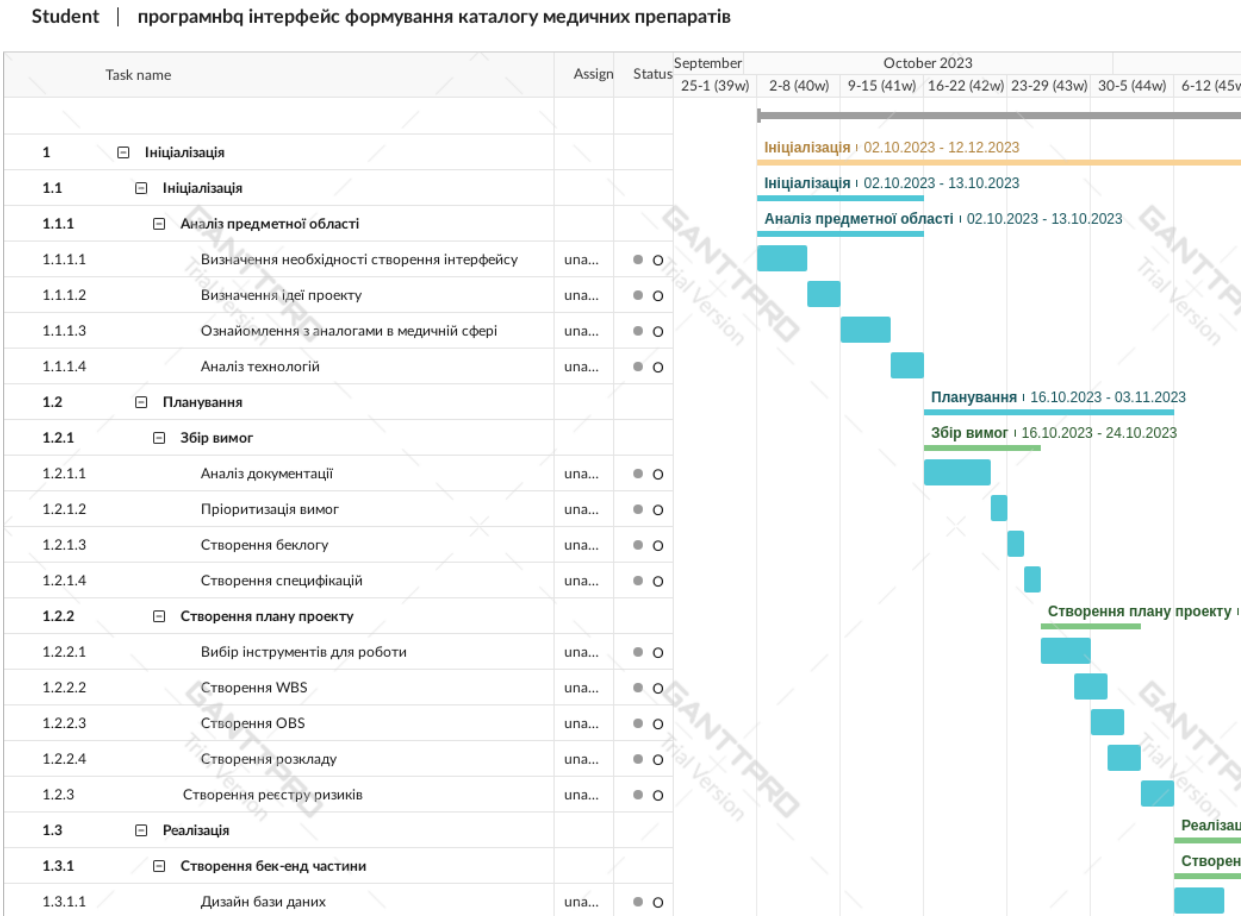


Рисунок А.3 – Календарний графік виконання робіт п. 1.1-1.2

Джерело: побудовано автором на основі даних WBS

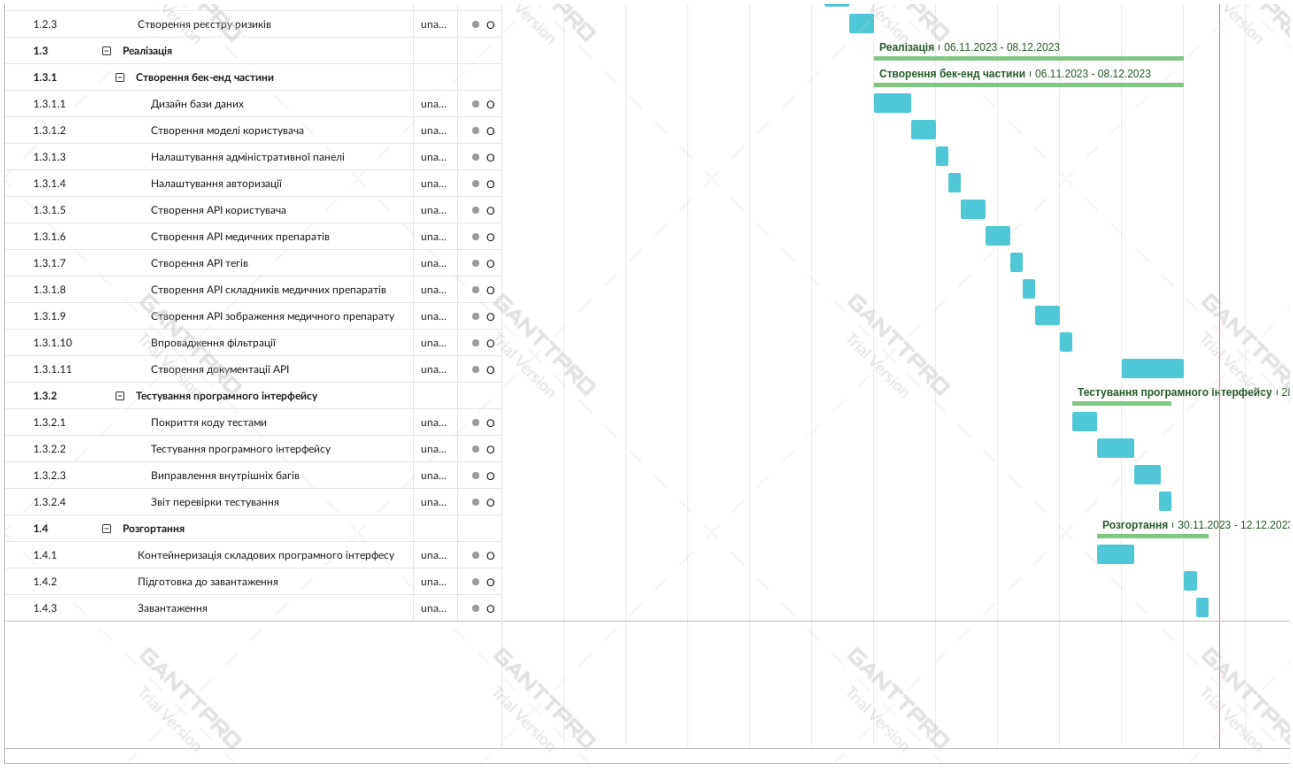


Рисунок А.3 – Календарний графік виконання робіт п. 1.3-1.4

Джерело: побудовано автором на основі даних WBS

Управління ризиками проекту є необхідним етапом у плануванні та виконанні робіт. Це включає якісне та кількісне оцінювання ризиків, спрямоване на ідентифікацію та управління можливими впливами на проект. Шкала оцінки ризиків за ймовірністю та величиною впливу надана у таблиці А.3.

Таблиця А.3 – оцінки ризиків за ймовірністю та величиною впливу.

Оцінка	Ймовірність виникнення	Вплив ризику	Тип ризику
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

Ризик, визначений як ймовірнісна подія, може мати як позитивний, так і негативний вплив на проект. Ризики класифікуються як "відомі" (ідентифіковані та

оцінені) та "невідомі". Ідентифікація ризиків вимагає регулярних зусиль та участі різних учасників, таких як менеджери проекту та користувачі. Класифікація ризиків враховує ймовірність виникнення та величину втрат.

Управління ризиками проекту також передбачає вжиття заходів для усунення або зменшення впливу ризиків. Кількісне оцінювання ризиків виконується відповідно до ступеня важливості ризику.

Таблиця А.4 містить вказане вище оцінювання ризиків, тип, а також шкалу їх класифікації за їхнім впливом на проект та ймовірністю виникнення.

Крім того, у таблиці А.4 описано ризики та стратегії реагування на кожен з них.

Таблиця А.4 – Ризики та стратегії реагування

№	Зона ризику	Опис ризику	Імовірність	Вплив	Оцінка ризику	Типи відповідей	Стратегія реагування на ризики	Резервний план
1	*Зовнішній*	Блекаути	Високий	Високий	Високий	Прийняття	Завчасне забезпечення ноутбуком та блоками живлення, встановлення оптичного інтернету	Купівля незалежних джерел живлення і зв'язку для розробника
2	*Зацікавлені сторони*	Хвороба розробника	Середній	Високий	Високий	Пом'якшення	Перехресне навчання членів команди для виконання важливих ролей.	Заміна учасника проекту
3	*Управління проектом*	Ризик розкладу: брак часу для розробки та тестування певних частин проекту до дати впровадження	Середній	Середній	Середній	Пом'якшення	Детальна консультація з усіма зацікавленими сторонами для чіткого графіку проекту	Виділення додаткового часу розробки для усунення несправностей і налагодження
4	*Вимоги*	Розширення умов проекту	Середній	Середній	Середній	Уникнення	Попередньо домовитися про те, які саме функції мають бути у програмному інтерфейсі. Документування процесів і запитів на зміни. Регулярний перегляд і підтвердження вимог проекту із зацікавленими сторонами.	Планування гнучкого підходу до розробки для врахування змін.

Продовження таблиці А.4.

5	*Технології та дані*	Порушення даних або вразливі місця в безпеці	Високий	Високий	Високий	Пом’якшення	Під час розробки врахування всіх аспектів безпеки. Регулярне проведення оцінки безпеки та тестування на проникнення.	Розробка плану реагування на інцидент безпеки, включно з процедурами сповіщення клієнтів і заходами щодо дотримання нормативних вимог.
6	*Людські ресурси*	Розробнику не вистачає необхідних навичок або знань для вирішення конкретних завдань або технологій.	Середній	Середній	Середній	Пом’якшення	Попереднє визначення навчальних програми або ресурсів для ключових технологій і методологій.	Виділення часу для регулярних тренувань та підвищення кваліфікації. Співпраця із зовнішніми експертами або консультантами для передачі знань.
7	*Людські ресурси*	Вплив розбіжностей, непорозумінь або низького морального духу на продуктивність розробника та прогрес проекту	Низький	Середній	Низький	Пом’якшення	Впровадження відкритого спілкування та створення платформи для зворотного зв’язку.	Визначення причин конфлікту і пошук вирішення непорозумінь.

Продовження таблиці А.4.

8	*Людські ресурси*	Напружені фази проекту або тривалі години роботи призводять до вигоряння та зниження продуктивності.	Середній	Високий	Високий	Пом’якшення	Моніторинг робочого навантаження. Встановлення регулярних перерв.	Використання гнучких методик для планування робочих ритмів, забезпечуючи періоди відпочинку між роботою.
9	*Інше*	Ринкові та користувальницькі ризики: низька кількість користувачів	Середній	Середній	Середній	Пом’якшення	Постійне дослідження ринку та тестування протягом усього періоду розробки.	Планування маркетингової кампанії та кампанії із залучення користувачів, щоб за потреби сприяти сприйняттю.

ДОДАТОК Б

Лістинг основних модулів

app/app/urls.py

```
from django.conf import settings
from django.conf.urls import static
from django.contrib import admin
from django.urls import include, path
from drf_spectacular.views import SpectacularAPIView, SpectacularSwaggerView

from core import views as core_views

urlpatterns = [
    path("", admin.site.login),
    path("admin/", admin.site.urls),
    path("api/docs/", SpectacularSwaggerView.as_view(url_name="api-schema"),
name="api-docs"),
    path("api/healthy/", core_views.healthy, name="healthy"),
    path("api/medication/", include("medication.urls")),
    path("api/schema/", SpectacularAPIView.as_view(), name="api-schema"),
    path("api/user/", include("user.urls")),
]

if settings.DEBUG:
    urlpatterns += static.static(
        settings.MEDIA_URL,
        document_root=settings.MEDIA_ROOT,
    )
```

app/medication/urls.py

```
from django.urls import include, path
from rest_framework.routers import DefaultRouter
from medication import views

router = DefaultRouter()
router.register("medications", views.MedicationViewSet)
router.register("labels", views.LabelViewSet)
router.register("components", views.ComponentViewSet)

app_name = "medication"
urlpatterns = [path("", include(router.urls))]
```

app/user/urls.py

```
from django.urls import path
from user import views

app_name = "user"
urlpatterns = [
    path("create/", views.CreateUserView.as_view(), name="create"),
    path("get-token/", views.CreateTokenView.as_view(), name="get-token"),
    path("info/", views.ManageUserView.as_view(), name="info"),
]
```

app/core/models.py

```
import os
from uuid import uuid4

from django.conf import settings
from django.contrib.auth.models import (
    AbstractBaseUser,
    BaseUserManager,
    PermissionsMixin,
)
from django.db import models


def generate_medication_image_file_path(instance, fn):
    """
    Generate file path for a new medication image.

    Args:
        instance: The instance of the model.
        fn: The original filename.

    Returns:
        str: The generated file path.
    """
    file_extension = os.path.splitext(fn)[1]
    unique_filename = f"{uuid4()}{file_extension}"

    # Construct the file path
    file_path = os.path.join("uploads", "medication", unique_filename)

    return file_path


class UserManager(BaseUserManager):
    """Manager for users in the system"""

    def new_user_create(self, email, password=None, **kwargs):
        """
        Create user, save it in the database, and return.
        """
        user = self.model(email=self.email_normalization(email), **kwargs)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, password):
        """
        Create a new superuser.
        """
        superuser = self.new_user_create(email, password)
        superuser.is_staff = True
        superuser.is_superuser = True
        superuser.save(using=self._db)
        return superuser

    def email_normalization(self, email: str):

        if not email:
            raise ValueError("Email cannot be empty.")
        if "@" not in email:
            raise ValueError("Invalid email format. An email must contain '@'.")
```

```
        return email.strip().lower()


class User(AbstractBaseUser, PermissionsMixin):
    """System User"""

    email = models.EmailField(max_length=255, unique=True)
    username = models.CharField(max_length=255)
    is_staff = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)

    objects = UserManager()
    USERNAME_FIELD = "email"


class Medication(models.Model):
    """Medication object"""

    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    components = models.ManyToManyField("Component")
    dose_mg = models.IntegerField()
    labels = models.ManyToManyField("Label")
    instructions = models.TextField(blank=True)
    item_price = models.DecimalField(max_digits=5, decimal_places=2)
    med_link = models.CharField(max_length=255, blank=True)
    image = models.ImageField(null=True,
upload_to=generate_medication_image_file_path)

    def __str__(self):
        return self.name


class Component(models.Model):
    """Component of medication object"""

    title = models.CharField(max_length=255)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)

    def __str__(self):
        return self.title


class Label(models.Model):
    """Label for medications search and filtering"""

    title = models.CharField(max_length=255)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)

    def __str__(self):
        return self.title
```

app/core/admin.py

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from django.utils.translation import gettext_lazy as _

from core import models


class CustomUserAdmin(UserAdmin):
    """Customize the admin pages"""

    list_display = ["email", "username"]
    ordering = ["id"]

    fieldsets = (
        (None, {"fields": ("email", "password")}),
        (
            _("Permissions"),
            {
                "fields": (
                    "is_active",
                    "is_staff",
                    "is_superuser",
                )
            },
        ),
        (_("Important dates"), {"fields": ("last_login",)}),
    )

    readonly_fields = ["last_login"]

    add_fieldsets = (
        (
            None,
            {
                "classes": ("wide",),
                "fields": (
                    "email",
                    "password1",
                    "password2",
                    "username",
                    "is_active",
                    "is_staff",
                    "is_superuser",
                ),
            },
        ),
    )


# Register custom admin classes for the User model and other related models
admin.site.register(models.User, CustomUserAdmin)
admin.site.register(models.Medication)
admin.site.register(models.Label)
admin.site.register(models.Component)
```

app/medication/views.py

```
from urllib.parse import unquote

from drf_spectacular.plumbing import OpenApiParameter, OpenApiTypes
from drf_spectacular.utils import extend_schema, extend_schema_view
from rest_framework import authentication, mixins, permissions, viewsets
from rest_framework.decorators import action
from rest_framework.response import Response
from rest_framework.status import HTTP_200_OK, HTTP_400_BAD_REQUEST,
HTTP_403_FORBIDDEN
from rest_framework.exceptions import PermissionDenied

from core.models import Component, Label, Medication
from medication import serializers


def _str_list_to_ints(qs):
    """Convert a list of strings to integers"""
    return [int(str_id) for str_id in qs.split(",") if str_id.isdigit()]


class IsOwnerOrReadOnly(permissions.BasePermission):
    message = 'You do not have permission to perform this action.'

    def has_object_permission(self, request, view, obj):
        if request.method in ('GET', 'HEAD', 'OPTIONS'):
            return True

        if obj.user == request.user:
            return True

        raise PermissionDenied(detail=self.message, code=HTTP_403_FORBIDDEN)


@extend_schema_view(
    list=extend_schema(
        parameters=[
            OpenApiParameter(
                "labels",
                OpenApiTypes.STR,
                description="Comma separated list of title IDs or Titles to filter",
            ),
            OpenApiParameter(
                "components",
                OpenApiTypes.STR,
                description="Comma separated list of component IDs or Titles to
filter",
            ),
        ]
    )
)
class MedicationViewSet(viewsets.ModelViewSet):
    """View for managing medication APIs"""

    serializer_class = serializers.MedicationDetailSerializer
    queryset = Medication.objects.all()
    authentication_classes = [authentication.TokenAuthentication]
    permission_classes = [permissions.IsAuthenticated, IsOwnerOrReadOnly]

    def get_queryset(self):
        """Retrieve medications for authenticated user"""
        queryset = self.queryset
```

```
        queryset_labels = queryset_comp = self.queryset.none()

        if self.request.query_params:
            if labels := self.request.query_params.get("labels"):
                labels = unquote(labels, encoding="utf-8")
                queryset_labels_id = self.queryset.none()
                if labels_ids := _str_list_to_ints(labels):
                    queryset_labels_id = queryset.filter(labels__id__in=labels_ids)
                labels_titles = labels.split(",")
                queryset_labels_t = queryset.filter(labels__title__in=labels_titles)
                queryset_labels = queryset_labels_t | queryset_labels_id

            if components := self.request.query_params.get("components"):
                components = unquote(components, encoding="utf-8")
                queryset_comp_id = self.queryset.none()
                if components_ids := _str_list_to_ints(components):
                    queryset_comp_id =
queryset.filter(components__id__in=components_ids)
                components_titles = components.split(",")
                queryset_comp_t =
queryset.filter(components__title__in=components_titles)

                queryset_comp = queryset_comp_t | queryset_comp_id

            queryset = queryset_labels | queryset_comp

        return queryset.order_by("-id").distinct()

    def get_serializer_class(self):
        """Return the serializer class for request"""
        if self.action == "list":
            return serializers.MedicationSerializer
        elif self.action == "upload_image":
            return serializers.MedicationImageSerializer

        return self.serializer_class

    def perform_create(self, serializer):
        """Create a new medication"""
        serializer.save(user=self.request.user)

    @action(methods=["POST"], detail=True, url_path="upload-image")
    def upload_image(self, request, pk=None):
        """Upload an image to medication"""
        medication = self.get_object()
        serializer = self.get_serializer(medication, data=request.data)

        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=HTTP_200_OK)

        return Response(serializer.errors, status=HTTP_400_BAD_REQUEST)


@extend_schema_view(
    list=extend_schema(
        parameters=[
            OpenApiParameter(
                "assigned_only",
                OpenApiTypes.INT,
                enum=[0, 1],
                description="Filter by items assigned to medications",
            ),
```

```
            OpenApiParameter(
                "id_or_title",
                OpenApiTypes.STR,
                description="Comma separated list of IDs or Titles to filter",
            ),
        ]
    )
)
class BaseMedicationAttrViewSet(
    mixins.DestroyModelMixin,
    mixins.UpdateModelMixin,
    mixins.ListModelMixin,
    viewsets.GenericViewSet,
):
    """Base class for managing medication attrs in the database"""

    authentication_classes = [authentication.TokenAuthentication]
    permission_classes = [permissions.IsAuthenticated, IsOwnerOrReadOnly]

    def get_queryset(self):
        """Filter queryset for authenticated user"""
        assigned_only = bool(int(self.request.query_params.get("assigned_only", 0)))
        queryset = self.queryset
        if assigned_only:
            queryset = queryset.filter(medication__isnull=False)

        queryset_id = self.queryset.none()
        if id_or_title := self.request.query_params.get("id_or_title"):
            id_or_title = unquote(id_or_title, encoding="utf-8")
            if ids := _str_list_to_ints(id_or_title):
                queryset_id = queryset.filter(id__in=ids)
            titles = id_or_title.split(",")
            queryset_title = queryset.filter(title__in=titles)

            queryset = queryset_id | queryset_title

        return queryset.order_by("-title").distinct()


class LabelViewSet(BaseMedicationAttrViewSet):
    """Manage labels in the database"""

    serializer_class = serializers.LabelSerializer
    queryset = Label.objects.all()


class ComponentViewSet(BaseMedicationAttrViewSet):
    """Manage components in the database"""

    serializer_class = serializers.ComponentSerializer
    queryset = Component.objects.all()
```

app/medication/serializer.py

```
from rest_framework import serializers

from core.models import Component, Label, Medication


class ComponentSerializer(serializers.ModelSerializer):
    """Serializer for Components"""

    class Meta:
        model = Component
        fields = ["id", "title"]
        read_only_fields = ["id"]


class LabelSerializer(serializers.ModelSerializer):
    """Serializer for Labels"""

    class Meta:
        model = Label
        fields = ["id", "title"]
        read_only_fields = ["id"]


class MedicationSerializer(serializers.ModelSerializer):
    """Serializer for the Medication model."""

    # Nested serializers for related fields
    labels = LabelSerializer(many=True, required=False)
    components = ComponentSerializer(many=True, required=False)

    class Meta:
        model = Medication
        fields = ["id", "name", "dose_mg", "item_price", "med_link", "labels",
"components"]
        read_only_fields = ["id"]

    def _get_or_create_labels(self, labels_data, medication_instance):
        """Handles getting or creating labels and associating them with the
medication."""
        auth_user = self.context["request"].user
        for label_data in labels_data:
            label_obj, _ = Label.objects.get_or_create(user=auth_user, **label_data)
            medication_instance.labels.add(label_obj)

    def _get_or_create_components(self, components_data, medication_instance):
        """Handles getting or creating components and associating them with the
medication."""
        auth_user = self.context["request"].user
        for component_data in components_data:
            component_obj, _ = Component.objects.get_or_create(user=auth_user,
**component_data)
            medication_instance.components.add(component_obj)

    def create(self, validated_data):
        """Creates a new medication instance with associated labels and
components."""
        labels_data = validated_data.pop("labels", [])
        components_data = validated_data.pop("components", [])

        medication_instance = Medication.objects.create(**validated_data)
        self._get_or_create_labels(labels_data, medication_instance)
```

```
        self._get_or_create_components(components_data, medication_instance)

        return medication_instance

    def update(self, instance, validated_data):
        """Updates an existing medication instance with new data."""
        labels_data = validated_data.pop("labels", None)
        components_data = validated_data.pop("components", None)

        if labels_data is not None:
            instance.labels.clear()
            self._get_or_create_labels(labels_data, instance)

        if components_data is not None:
            instance.components.clear()
            self._get_or_create_components(components_data, instance)

        for attr, value in validated_data.items():
            setattr(instance, attr, value)

        instance.save()
        return instance


class MedicationDetailSerializer(MedicationSerializer):
    """Serializer for medication detail view"""

    class Meta(MedicationSerializer.Meta):
        fields = MedicationSerializer.Meta.fields + ["instructions", "image"]


class MedicationImageSerializer(serializers.ModelSerializer):
    """Serializer for uploading images to medications"""

    class Meta:
        model = Medication
        fields = ["id", "image"]
        read_only_fields = ["id"]
        extra_kwargs = {"image": {"required": "True"}}
```