Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Інформаційна система для обміну даними мережі аптек
Назва теми

КвРІСТ 200185.20.01.01 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *126 «Інформаційні системи та технології»*
Шифр, назва

Освітня програма *«Інформаційні системи та технології»*
Назва

Виконав: студент IV курсу, група ІСТ-20-1 _____ В.В. Гусаченко
Підпис Ініціали, прізвище

Керівник _____ І.О. Засорнова
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ І.О. Засорнова
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«14» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень Бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 126 Інформаційні системи та технології

Освітня програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О. Говорущенко

" 10 " 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Гусаченко Вікторії Володимирівній

Прізвище, ім`я, по батькові студента

1. Тема проекту (роботи) Інформаційнасистема для обміну даними мережі аптек

Керівник проекту (роботи) Засорнова І.О., к.т.н., доцент

Прізвище, ім`я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Теоретичні основи досліджуваної проблеми

Проєктування інструментальних засобів створення і використання інформаційних систем та технологій

Реалізація та тестування інструментальних засобів створення і використання інформаційних систем та технологій

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

Блок-схема принципу роботи ІС

UML діаграми взаємодії користувача та адміністратора ІС

Структура бази даних інформацційної системи обміну даними мережі аптек

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І.О., доцент кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання « 10 » 01 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – дослідження предметної області та огляд існуючих інформаційних систем для розв'язання завдання	01.03.2024	виконано
4	Робота над розділом 2 – вибір компонентів для розробки інформаційнох системи для обміну даними мережі аптек	01.04.2024	виконано
5	Робота над розділом 3 – реалізація інформаційної системи для обміну даними мережі аптек	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	30.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ В.В. Гусаченко
Підпис Ініціали, прізвище

Керівник роботи _____ І.О. Засорнова
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кількість листів	№ екз	Примітка
			Текстові документи			
1		КвРІСТ 200185.20.01.01 ПЗ	Пояснювальна записка	57		
			Графічні матеріали			
2		КвРІСТ 200185.20.01.01 Е8	Блок-схема принципу роботи ІС	1		
3		КвРІСТ 200185.20.01.01 Е8	UML-діаграма взаємодії користувача та адміністратора ІС	1		
4		КвРІСТ 200185.20.01.01 Е8	Структура бази даних інформаційної системи обміну даними мережі аптек	1		

КвРІСТ 200185.20.01.01 ВП

Зм	Арк	№ докум	Підпис	Дата	Відомість проєкту	Літера	Аркуш	Аркушів
Розробив(ла)		Гусаченко				У	1	1
Перевір		Засорінова				ХНУ, ІСТ-20-1		
Н. контр.		Засорінова						
Затв.		Говорущенко		24.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Інформаційна система «Обмін даними мережі аптек»».

Автор роботи: Гусаченко Вікторія Володимирівна.

Керівник роботи: Засорнова Ірина Олександрівна

Пояснювальна записка: 57 с., 10 рис., 1 табл., 3 дод., 54 джерел.

Графічна частина: 3 креслення.

ІНФОРМАЦІЙНА СИСТЕМА, API, JSON, REST, HTTP, HTTPS, БАЗА ДАНИХ.

Метою роботи є забезпечення ефективного і безпечного обміну даними між аптечними мережами.

Об'єктом дослідження є процес роботи інформаційної системи для обміну даними мережі аптек.

Предметом дослідження є система для обміну даними мережі аптек з використанням технологій ASP.NET Web API, Entity Framework і PostgreSQL.

Практичне значення інформаційної системи для обміну даними мережі аптек є значне покращення управління запасами лікарських засобів і зменшує час на виконання бізнес-процесів, що сприяє ефективнішому обслуговуванню клієнтів та оптимізації операцій у аптечних мережах.

ГВГ

Підпис студента

30.05.24

Дата

ЗМІСТ

Зм.	Арк.	№докум.	Підпис	Дата	КвРІСТ.200185.20.01.01 ПЗ		
Виконав		Гусаченко В.В.			Інформаційна система для обміну даними мережі аптек Пояснювальна записка	Літера	Аркуш 2	Аркушів 69
Перевір.		Засорінова І.О.				У		
Н.контр.		Засорінова І.О.				ХНУ ІСТ-20-1		
Затвер.		Говорущенко Т.О.		24.0				

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application Programming Interface.

ІС – інформаційна система.

JSON – JavaScript Object Notation.

HTTP – HyperText Transfer Protocol.

HTPPS – HyperText Transfer Protocol Secure.

REST – Representational State Transfer.

TCP – Transmission Control Protocol.

UDP – User Datagram Protocol.

CQRS – Command and Query Responsibility Segregation.

DTO – Data Transfer Object.

ВСТУП

Сучасні інформаційні технології постійно еволюціонують, розширюючи свої можливості та сфери застосування. Однією з найбільш актуальних сфер є автоматизація бізнес-процесів у галузі охорони здоров'я, зокрема у фармацевтичному секторі. Збільшення обсягів даних, що циркулюють у мережах аптек, потребує ефективних рішень для управління та обміну інформацією між різними структурними підрозділами. Основні тенденції розвитку інформаційних систем у цій галузі включають інтеграцію різних програмних продуктів, підвищення рівня безпеки даних, а також удосконалення методів аналізу та обробки інформації.

Проблема, на розв'язання якої спрямована ця кваліфікаційна робота, полягає у створенні ефективної інформаційної системи для обміну даними між аптеками в межах однієї мережі. Така система повинна забезпечувати швидкий і безпечний доступ до необхідної інформації, сприяти автоматизації процесів замовлення, обліку та розподілу лікарських засобів, а також підвищувати загальну ефективність роботи аптечної мережі.

У даній кваліфікаційній роботі об'єктом є інформаційна система для обміну даними мережі аптек. Це означає що об'єктом дослідження є реалізація конкретної інформаційної системи з використанням технологій ASP.NET Web API, Entity Framework, PostgreSQL для серверної частини, а також HTML, CSS, JavaScript для клієнтської частини.

Актуальність теми даної кваліфікаційної роботи визначається необхідністю підвищення ефективності та безпеки обміну даними між аптеками в межах однієї мережі. Це дозволить не лише покращити якість обслуговування клієнтів, але й оптимізувати внутрішні бізнес-процеси, що в свою чергу підвищить конкурентоспроможність аптечної мережі на ринку. Комерційна привабливість результатів роботи полягає у можливості впровадження розробленої системи у різних аптечних мережах, що потенційно може збільшити їх прибутковість.

Метою цієї кваліфікаційної роботи є забезпечення ефективного і безпечного обміну даними між аптечними мережами. Це включає автоматизацію бізнес-процесів, підвищення точності та швидкості обробки даних, зниження ймовірності помилок і забезпечення високого рівня захисту інформації. Така система дозволить оптимізувати управління запасами, покращити взаємодію між різними аптеками в мережі та підвищити якість обслуговування клієнтів.

Для досягнення цієї мети необхідно виконати наступні завдання:

- провести огляд сучасних інформаційних систем у фармацевтичному секторі;
- визначити ключові вимоги до розроблюваної системи;
- розробити архітектуру інформаційної системи;
- реалізувати програмне забезпечення для обміну даними;
- провести тестування та оцінку ефективності розробленої системи.

Розроблена інформаційна система призначена для застосування у мережах аптек, що мають декілька філій і потребують централізованого управління та обміну даними. Вона дозволить автоматизувати процеси замовлення, обліку та розподілу лікарських засобів, підвищити точність і швидкість обміну інформацією, а також забезпечити високий рівень захисту даних.

1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

В ході даної роботи розглянуто методи передачі даних на прикладі мережі аптек, але спочатку потрібно описати проблематику. Перед тим як з’явиться змога запропонувати надійний варіант для вирішення проблеми, потрібно більш детально розібратися у самих проблемах, для того щоб більш глибше розібратись у підходах для їх вирішення. Аналізуючи дану систему є можливість виділити основні проблеми даної інформаційної системи:

- розподіленість та розбіжність інформаційної системи;
- безпека даних;
- стандартизація даних і протоколи обміну;
- інтеграція системи;
- ефективність та швидкодія обміну даними.

Розподіленість та розбіжність інформаційної системи. Тільки в одному місті кількість фізичних точок обслуговування може бути досить великою, що говорить про масштаби країни. У кожній фізичній точці може бути своя внутрішня система обліку препаратів та обслуговування, тому, однією із основних цілей буде налаштування універсальної системи для максимально зручної та гладкої інтеграції з різними системами різних мереж аптек. Для цього потрібно використовувати загальноприйняті правила розробки API з використанням стандартних та популярних протоколів. Це буде розглянуто трохи пізніше в цьому пункті. Також потрібно використовувати загальноприйняті правила для встановлення безпекових норм та обмежень в системі.

Безпека даних при розробці саме даної ІС являється досить важливим етапом, тому що порушення правил безпеки медичних даних пацієнтів карається законом відповідно до 132 та 145 статей Кримінального Кодексу

України [1]. Запровадження безпекових норм та правил, в першу чергу, є зобов’язанням перед законом.

Відповідно до вимог безпекових стандартів, ключовим елементом є застосування відповідних протоколів безпеки та механізмів передачі даних. Протокол HTTPS [2] відіграє критичну роль у забезпеченні конфіденційності та цілісності інформації, зашифровуючи дані під час їх передачі через мережу. У даному розділі детально розглянуті загальні протоколи і правила обробки даних. На основі міжнародних стандартів і рекомендацій будуть розглянуті кращі практики і методи інтеграції систем, спрямовані на забезпечення ефективної та безпечної роботи інформаційних систем. Одним із важливих аспектів стандартизації є використання API та методів передачі даних у форматі JSON. JSON є популярним вибором для обміну даними через мережу завдяки своїй простоті та універсальності. Хоча він не є обов'язковою частиною REST архітектури, він широко використовується у сучасних API системах. Однак, варто зазначити виняток у вигляді технології RPC та її новітньої версії gRPC [3]. gRPC використовує протокол HTTP/2, який забезпечує передачу інформації у вигляді байтових потоків замість традиційних JSON пакетів, що є типовим для протоколів HTTP та HTTPS [4]. Це забезпечує високу швидкість передачі даних та ефективність мережевого взаємодії у розподілених системах. Враховуючи різноманітність протоколів та форматів передачі даних, важливо вибрати той підхід, який найкращим чином відповідає вимогам конкретного проекту з урахуванням його безпекових і функціональних вимог

Стандартизація даних також являє собою правила щодо розробки та підтримки системи автоматизації та синхронізації даних. Тут можна оглянути дивитись на REST. REST – це загальний набір правил щодо розробки стандартизованих API систем, який включає в себе ряд правил яких важливо дотримуватись. REST системи зараз являються досить популярними, відповідно, і загально стандартизованими. Для інтеграції різних систем також використовуються правила REST для забезпечення автоматизованого

програмного забезпечення, яке не потребує зовнішнього втручання. REST [5] також пропонує вести спеціальну документацію щодо програмного забезпечення для полегшення вивчення вашого API іншими спеціалістами.

Для передачі інформації також використовуються спеціальні протоколи, тому потрібно розглянути, що таке протокол. Протокол передачі інформації – це лише набір мережевих правил для передачі інформації [19]. Цими правилами описується відповідний пакет інформації, для того щоб отримувач розумів як його правильно обробляти. Протокол HTTP являється протоколом верхнього рівня які відомий дуже багатьом людям за рахунок його постійного перебування в URL стрічці. Протокол HTTPS являється надбудовою над протоколом HTTP, який додає спеціальні криптографічні шифрування для забезпечення шифрування інформації, яка передається по мережі. Без цього шифрування дані являються повністю відкритими і їх може побачити будь який зловмисник, який захоче перехопити вашу інформацію. Існують також інші протоколи передачі інформації більш нижчого рівня, а ніж протоколи HTTP та HTTPS. Такими протоколами являються TCP та UDP [6]. Наразі зараз роздивлятись глибокі концепції протоколів, але звернемо увагу на їх принцип роботи та основну різницю між ними.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

Інформаційні системи обміну даними мережі аптек на сьогоднішній день виявляються не лише необхідним елементом для їх ефективного функціонування, але й ключовим інструментом для досягнення конкурентних переваг. Завдяки цим системам, аптеки мають можливість автоматизувати ряд процесів, що сприяє підвищенню продуктивності та оптимізації бізнес-процесів. Важливо зазначити, що переваги існуючих рішень у цій галузі надають аптекам можливість не лише підтримувати високий рівень обслуговування клієнтів, але й раціонально управляти ресурсами та аналізувати дані для прийняття обґрунтованих управлінських рішень.

Інформаційні системи для обміну даними в мережі аптек відіграють важливу роль у забезпеченні ефективного управління, оптимізації запасів та покращенні обслуговування клієнтів. Однак, перед вибором конкретного рішення варто ретельно проаналізувати його переваги та недоліки. Для початку, однією з основних переваг існуючих рішень є автоматизація процесів управління складом та обліку товарів. Системи дозволяють в реальному часі відстежувати рух товарів між аптеками, що сприяє покращенню контролю за запасами та уникненню нестач.

Інформаційні системи грають ключову роль у забезпеченні ефективного обміну даними між аптеками, що в мережі. Це не лише сприяє оперативній реакції на зміни в попиті та оптимізації запасів, але й значно полегшує ведення обліку продажів. В результаті цього значно поліпшується можливість аналізу ефективності кожної окремої аптеки та всієї мережі в цілому. Однак важливо враховувати, що розробка і впровадження таких систем потребують комплексного підходу та врахування високих стандартів безпеки даних для забезпечення надійності та захисту конфіденційної інформації.

Невід'ємною перевагою інформаційних систем є також підвищення рівня обслуговування клієнтів. Завдяки автоматизованій системі обліку, персонал може швидко та точно визначити наявність товару та надати клієнту необхідну інформацію. Це сприяє підвищенню задоволеності клієнтів та збільшенню їх лояльності до мережі аптек. Проте, наряду з перевагами, існують і недоліки інформаційних систем для обміну даними в мережі аптек. Один з таких недоліків полягає у високих витратах на впровадження та підтримку системи. Вартість розробки програмного забезпечення, навчання персоналу та обслуговування системи може бути значною, особливо для невеликих мереж аптек.

Крім того, інтеграція інформаційної системи з іншими вже існуючими програмами та обладнанням може виявитися складною. Наприклад, важкість синхронізації з обліковими системами або POS-терміналами [11] може

призвести до проблем у веденні обліку продажів та контролю за залишками товарів. Також, інформаційні системи можуть бути вразливі до кібератак та порушень безпеки даних. Збереження конфіденційної інформації про клієнтів та фінансові дані потребує відповідних заходів захисту, а недостатня безпека може призвести до серйозних проблем і втрати довіри з боку клієнтів.

Отже, при виборі інформаційної системи для обміну даними в мережі аптек важливо розглянути різноманітні аспекти. Серед них функціональні можливості системи, її здатність відповідати потребам у оперативному обміні даними та ефективному обліку продажів. Також важливо врахувати масштабованість системи і її здатність до інтеграції з іншими інформаційними системами. Забезпечення високого рівня безпеки даних, особливо у відношенні конфіденційної інформації і персональних даних клієнтів, є критично важливим аспектом. Оцінка вартості впровадження, підтримки та оновлення системи, а також наявність якісної технічної підтримки і сервісу також впливають на вибір оптимального рішення. Врахування користувацького досвіду, зокрема зручності інтерфейсу системи, що сприяє підвищенню продуктивності працівників, завершує обґрунтований аналіз, спрямований на досягнення ефективного управління та високого рівня обслуговування клієнтів в мережі аптек.

1.3 Методологічні підходи до вирішення задачі за темою дослідження

На даному етапі потрібно визначитись з вимогами щодо самої системи. Найбільш ефективним варіантом буде використання стеку .NET [12] через його гнучкість та можливість розгортання на різних платформах, а також багатий стек фреймворків та бібліотек, можливість інтеграції з багатьма сервісами та наявність дуже низки готових рішень. Для побудови веб системи у вигляді веб-API мій вибір однозначно схиляється в сторону фреймвфорку asp.net core 8 web api [7]. На це є низка причин, які розглянуто далі в розділі, але спочатку

потрібно розглянути технічне завдання та розподілити варіанти реалізації для приблизної оцінки розробки системи. Але потрібно поняття клієнта та серверу.

Клієнт – це програма яка користується певними ресурсами для взаємодії з кінцевим користувачем, тобто являється інтерфейсом для взаємодії кінцевого користувача з системою обробки даних [13]. Сервер – це місце в інфраструктурі програмного забезпечення або програмної системи яка відповідає за обробку даних, їх транспортування та зв’язок з місцем їх зберігання, а також надсилання відповідних результатів клієнту у відповідь на запит про обробку або отримання інформації [14]. З сервером є можливість взаємодіяти за допомогою стандартного протоколу передачі даних HTTP.

Протокол HTTP має набір методів, яка вказують сервер як правильно реагувати на отримане повідомлення. Основними методами HTTP запиту є:

- GET;
- POST;
- PUT;
- PATCH;
- DELETE.

Розберемо кожен з них по черзі. Перший метод HTTP запиту це GET [33]. Цей тип запиту вказує серверу на те, що клієнт хоче отримати якусь інформацію, тому цей метод має відповідну назву. Основною його відмінністю є відсутність тіла запиту, яке використовується для передачі інформації між клієнтом та сервером, тому що GET запит передає інформацію через URL рядок. Ця інформація називається параметрами. Параметри GET запиту використовуються для уточнення інформації, яку хоче отримати клієнт від серверу, що дозволяє серверу робити вибірку даних та різного роду маніпуляції перед тим як віддати кінцевий результат клієнту. Прикладом використання методу GET є будь який запит, який кінцевий користувач бачить при переході на сайт, при взаємодії з мобільним або настільним додатком, якщо вони

реалізуються HTTP запити на сервер. Дані, які передаються користувачу при GET запиті являються результатом роботи серверу.

POST метод використовується для передачі інформації на сервер та являється одним з основних методів. Ці два методи, GET та POST, можуть повністю задовільнити потреби спілкування клієнта та сервера, оскільки вони реалізують отримання інформації від сервера та передачу інформації на сервер. POST метод відрізняється від GET тип, що він має тіло запиту, в якому передається інформація, в основному, в форматі JSON. Також, POST метод не входить в журнал обліку запитів на відмінну від GET запиту, відповідно, інформація про нього ніде не зберігається, що забезпечує конфіденціальність даних при їх передачі. Прикладом для використання POST запиту є будь яка інформація, яка відправляється на сервер шляхом взаємодії з кінцевим користувачем. Найяскравіший приклад цього це система реєстрації та авторизації. Після заповнення полів відповідними даними клієнт робить відправку даних на сервер використовуючи POST запит, попередньо шифруючи ваші дані для їх захисту та збереження конфіденціальності [15].

Метод PUT являється модернізацією методу POST який використовує можливості GET запиту [8]. Тобто, можемо передати частину інформації через тіло запиту, а іншу частину через URL рядок. Зазвичай, це метод використовується для оновлення інформації. В тілі запиту вказується інформація, яку потрібно оновити на сервері, а в URL рядок поміщається ідентифікатор, кому ці дані належать. Сервер ідентифікує сутність, якій належать ці дані та виконую відповідно обробку, оновлення та збереження, а потім повертає результат роботи який вказує на успішність або помилку з відповідним повідомленням.

PATHC являється схожим з методом PUT [9], але використовується для часткового оновлення інформації, про що каже його назва. Цей тип запиту працює трошки з іншим форматом даних, а саме з JsonPatchDocument, який представляє собою документ з вказівками щодо маніпуляції даними [16].

Структура документу складається з вказанням поля об’єкту, який потрібно оновити, та відповідну команду для нього. Ось основний список команд, які використовує JsonPatchDocument для маніпуляції даними:

- Add;
- Remove;
- Replace;
- Copy;
- Move;
- Test.

Під час входів виконання кваліфікаційної роботи будуть розглянуті не всі ці команди, але потрібно зазначити, що використання PATCH запиту використовується для часткового оновлення об’єктів з великим набором значень, коли буде потрібно оновити не весь об’єкт, але лише деякі його значення.

Для видалення використовується DELETE запит [20]. Він дуже схожий з методом GET, оскільки він передає лише ідентифікатор в URL рядку для ідентифікації ресурсу, який потрібно видалити, але не повертає нічого.

Усі вище вказані методи являються досить ефективними для опису ефективного API ресурсу з використанням стандартних правил взаємодії та загальноприйнятому протоколу передачі інформації HTTP.

Тепер, коли розібрано основні поняття при роботі з протоколом передачі даних HTTP та його методами передачі інформації, опишемо основні кінцеві точки на сервері для взаємодії з клієнтом. Основна кінцева точка, над якою має бути проведена найбільша робота – це метод для отримання інформації з серверу. Цей метод має повністю відповідати правилам RESTful API ресурсу та, за можливості, бути само документованим [17].

Друга основна кінцева точка – це синхроназація між клієнтом та сервером [18]. Клієнт забов’язаний регулярно оновлювати дані щодо актуальності

фізичних ресурсів, які вимагає програмна система. Це може бути не одна кінцева точка, для прикладу, коли може оновлюватись інформація від клієнту:

- покупка товару на фізичній точці;
- списання товару зі складу;
- списання товару з фізичної точки;
- завезення товару на склад;
- бронювання товару на складі;
- бронювання товару на фізичній точці;
- перевезення товару з складу на фізичну точку;
- перевезення товару з фізичної точки на склад;
- перевезення товару між фізичними точками;
- перевезення товару між складами.

Усі ці фактори повинні враховуватися в системі, але це не обов'язково повинні бути окремі кінцеві точки на сервері. Тут можливо ввести систему клієнтів та їх ідентифікації, але кожна подія повинна бути зафіксована та задокументована. Відповідно до цього, кожний клієнт повинен інформувати систему про рух товару в середині системи для забезпечення актуальності даних.

Вище описано два ключових моменти, які потребують детального вивчення та якісної роботи над ними. Про це описано в наступних розділах. Наразі залишається відкритим питання тестування під час розробки. Для тестування буде використовуватись стандартний інструмент для здійснення HTTP запитів та, відповідно, тестування серверу – Postman [10]. Postman – це безкоштовний програмний продукт, який симулює запити формату HTTP на сервер. Він являється досить гнучким та дозволяє зберігати вже описані запити для повторного тестування кінцевих точок серверу.

1.4 Постановка задачі

Створення інформаційної системи для обміну даними мережі аптек є важливим завданням, спрямованим на поліпшення ефективності управління та оптимізацію бізнес-процесів в цій галузі. Першою задачею в цьому процесі є аналіз потреб та вимог користувачів системи. Це включає вивчення поточних методів обміну даними між аптеками, їхніх проблем та недоліків, а також визначення функціональних вимог до нової системи.

На другому етапі слід розробити концепцію системи, визначивши її основні компоненти, архітектуру та логіку роботи. Особлива увага повинна бути приділена забезпеченню безпеки даних та захисту від несанкціонованого доступу, оскільки в інформаційних системах аптек зберігаються конфіденційні медичні дані пацієнтів.

Третій етап полягає у реалізації розробленої концепції шляхом програмування та налаштування необхідного програмного забезпечення. Важливо забезпечити сумісність системи з існуючими програмами та технологічними рішеннями, що використовуються в аптечній галузі.

Четвертий етап передбачає тестування розробленої системи з метою виявлення помилок та недоліків, а також перевірки її працездатності в реальних умовах. Для цього можуть бути організовані пілотні проекти в обраних аптеках з наступним аналізом отриманих результатів.

На заключному етапі система повинна бути впроваджена в експлуатацію та підтримуватися в актуальному стані. Це включає навчання персоналу, забезпечення технічної підтримки та постійне вдосконалення системи з урахуванням змін в умовах ринку та технологічних інновацій. Регулярні оновлення та модернізації допоможуть забезпечити високу ефективність та конкурентоспроможність інформаційної системи в майбутньому.

1.5 Висновок

У даному розділі здійснено теоретичний аналіз створення та функціонування інформаційної системи обміну даними в мережі аптек. Виділено ключові компоненти, що впливають на ефективність та надійність таких систем, а також можливі проблеми та виклики при їх впровадженні. Основними аспектами є архітектура систем, технології передачі даних, безпека інформації та інтеграція програмних продуктів. Особливу увагу приділено надійності та безперебійності функціонування системи, що критично для підтримання високого рівня обслуговування клієнтів.

Впровадження інформаційної системи обміну даними дозволяє підвищити ефективність управління запасами, покращити взаємодію між відділами та точками продажу, а також забезпечити швидке обслуговування клієнтів. Це вимагає не лише технічних рішень, але й організаційної підготовки.

Таким чином, теоретичний аналіз дозволяє стверджувати, що сучасна інформаційна система обміну даними є необхідною умовою для ефективного функціонування мережі аптек в умовах зростаючої конкуренції та вимог до якості обслуговування. Впровадження таких систем сприятиме підвищенню конкурентоспроможності та сталого розвитку мережі аптек.

2 ПРОЄКТУВАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ СТВОРЕННЯ І ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

2.1 Доступ та обробка даних

Зберігання даних є критично важливим аспектом будь-якої інформаційної системи. Бази даних виконують роль основних компонентів, які забезпечують збереження, доступ та обробку інформації. У цій кваліфікаційній роботі розглядається процес вибору, проєктування та реалізації бази даних для інформаційної системи, зокрема використання PostgreSQL та підходу Code First [22].

Сховище даних виконує роль основного місця збереження інформації. Це може бути будь-який сервер або джерело, яке має функції запису інформації. Файли різних форматів можуть слугувати джерелами зберігання даних [23]. Технічно, інформацію можливо зберігати у текстових, бінарних та інших форматах. Наприклад, фотографія зберігає інформацію у вигляді певного набору байтів, які перетворюються в колір при відтворенні файлу певною програмою.

Будь-який файл є набором даних, який виконує свою цільову функцію при використанні відповідною програмою, яка коректно інтерпретує інформацію у цих файлах. Програма може використовувати будь-яке джерело для збереження даних, але необхідно враховувати вимоги до збереження даних для вибору оптимального місця їх розміщення. Основні вимоги до систем зберігання даних включають:

- структурованість даних;
- швидкість читання даних;
- швидкість запису даних;
- швидкість обробки запитів;
- зручність роботи людиною.

Саме таким джерелом зберігання даних є структура під назвою база даних, яка задовольняє весь указаний список вимог щодо розміщення та зберігання даних.

На момент написання кваліфікаційної роботи, у світі є великий вибір баз даних, які можна використовувати для виконання даної кваліфікаційної роботи [25]. Найбільш популярними є наступні наведені в таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика популярних баз даних.

База даних	Опис	Основні переваги
1	2	3
Microsoft SQL Server	Microsoft SQL Server — це реляційна система управління базами даних (СУБД), розроблена компанією Microsoft. Вона забезпечує високу продуктивність, масштабованість та надійність.	Інтеграція з Microsoft – проста інтеграція з продуктами Microsoft, такими як Azure, Visual Studio, Power BI. Безпека – шифрування даних, управління ідентифікацією та контроль доступу. Аналітика в реальному часі – можливості аналізу даних у реальному часі. Підтримка великих даних – обробка великих обсягів даних та високий рівень навантаження.
PostgreSQL	PostgreSQL — це реляційна СУБД з відкритим вихідним кодом, яка відома своєю надійністю, функціональністю та відповідністю стандартам SQL.	Безкоштовний, відкритий вихідний код з широкою функціональністю для підтримки розширених типів даних, процедур, індексів, користувацьких функцій, розширюваністю через плагіни та модулі і сумісністю з багатьма мовами програмування, включаючи C#, Java, Python, Ruby.

Кінець таблиці 2.1

1	2	3
Oracle	Oracle Database — це одна з найпотужніших і найпопулярніших комерційних реляційних СУБД, відома своєю високою продуктивністю, надійністю та безпекою.	Висока продуктивність – оптимізовані алгоритми обробки даних, масштабування. Безпека – шифрування даних, управління ідентифікацією користувачів, контроль доступу та аудит. Інтеграція та сумісність – легка інтеграція з продуктами Oracle, підтримка роботи з великими обсягами даних. Підтримка аналітики та Big Data – інструменти для аналізу та обробки великих даних.
MongoDB	MongoDB — це документо-орієнтована NoSQL база даних, яка добре підходить для зберігання неструктурованих даних.	Гнучкість моделі даних – зберігання даних у форматі BSON (бінарний JSON). Швидкість і масштабованість – висока продуктивність та горизонтальне масштабування. Простота використання – швидке налаштування та інтеграція. Підтримка реплікації та шардингу – висока доступність та стійкість до відмов завдяки реплікації та розподіленню навантаження.

Для виконання кваліфікаційної роботи буде використано базу даних PostgreSql [26], оскількість вона являється безкоштовною, не має обмеження щодо об’єму даних, а також сумісна з більшістю мов програмування, зокрема, з C# та платформою .NET.

Під час розробки інформаційної системи буде використовуватися підхід для проєктування та відтворення бази даних Code First. Цей підхід використовується, коли необхідно створювати та змінювати базу даних у залежності від змін в коді. Іншими словами, база даних повністю підпорядковується коду програми, а не навпаки.

Даний рівень має назву «Data Layer». Для реалізації цього рівня в програмі буде використовуватися фреймворк Entity Framework Core, який дозволяє зручно працювати з базами даних у середовищі .NET [27-29]. Для зміни структури бази даних та відповідних таблиць будуть використовуватись міграції [30].

Наступний етап проєктування бази даних – це аналіз програми, визначення відповідних вимог та створення моделі даних, необхідних для побудови схеми даних. Для цього потрібно проаналізувати сферу діяльності, щоб визначити набори даних, які будуть зберігатись у базі. До таких наборів даних належать:

- постачальники;
- поставки товарів;
- типи та категорії товарів;
- товари;
- аптеки;
- користувачі;
- клієнти;
- покупки.

Схема даних, побудована в результаті етапу проєктування, зображена на рисунку 2.1.

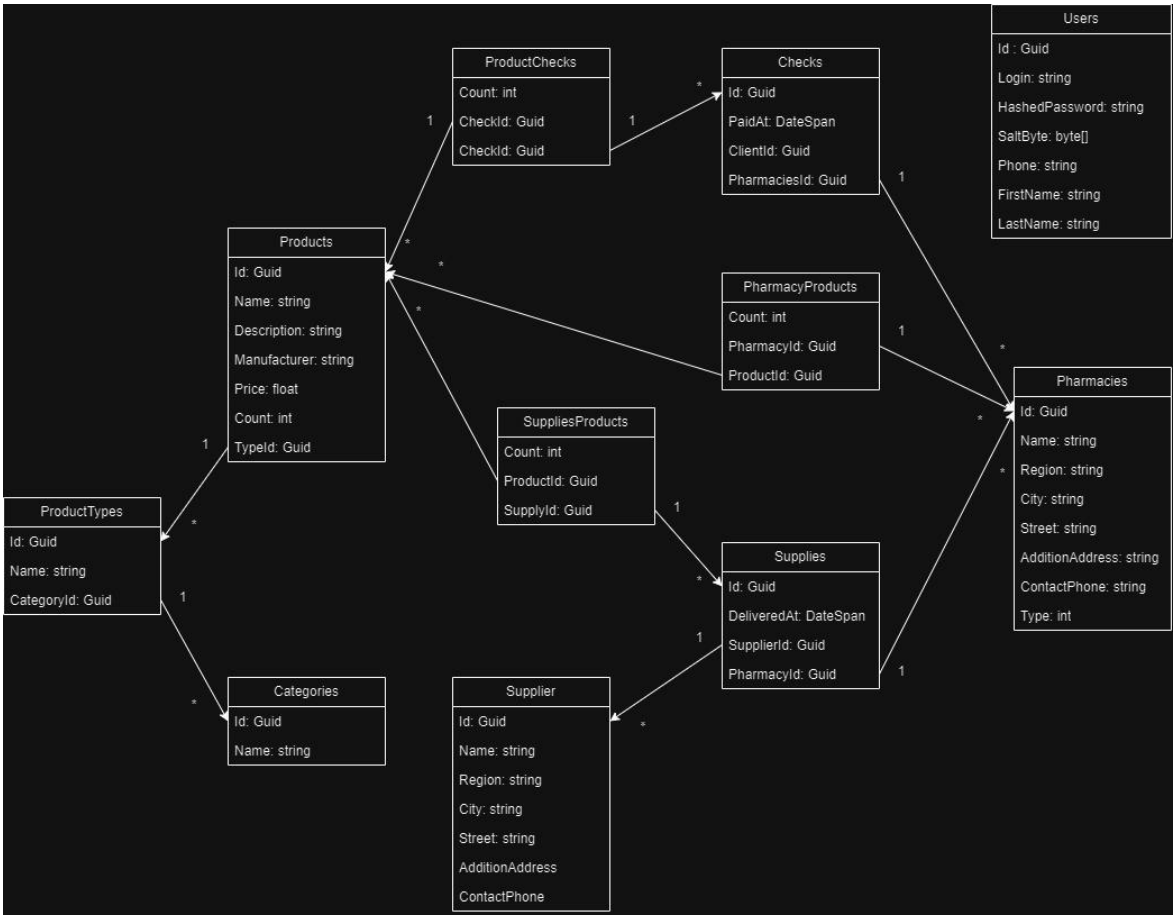


Рисунок 2.1 – Спроєктована схема даних

Рівень доступу до даних розглядається не на етапі проєктування, а на етапі реалізації. Доступ до даних виконується за допомогою фреймворку Entity Framework Core та репозиторіїв. Для забезпечення оптимізації доступу до даних ми відходимо від концепції GenericRepository для відокремлення методів доступу до даних від запитів на отримання та зміну даних [31, 32].

2.2 Аналіз та проєктування серверної частини інформаційної системи

Серверна частина будь-якої інформаційної системи поділяється на три основних рівні. Графічно ця структура умовно зображена на рисунку 2.2.

Рисунок 2.2 – Структура серверної частини

Верхній рівень серверної частини представляють методи зовнішнього впливу, які надають можливість кінцевому користувачу взаємодіяти з системою. За допомогою фреймворку ASP.NET та подібних технологій, цю роль відіграють контролери та методи контролерів, які ще називаються кінцевими точками серверу (endpoint) [33]. Ці кінцеві точки приймають запити від клієнтів, обробляють їх та повертають відповідь, реалізуючи логіку, необхідну для функціонування клієнтського інтерфейсу.

Методи обробки даних представляються внутрішнім функціоналом серверу. Цей рівень може використовувати сторонні засоби, такі як інші API, сервіси або бібліотеки. Стандартним підходом реалізації таких методів є сервісний підхід. У цьому підході кожний сервіс відповідає за одну або більше таблиць у базі даних або цільових наборів даних [34]. Прикладом такого сервісу може бути сервіс для виконання авторизації або реєстрації користувачів. Цей сервіс працює з авторизаційними даними користувача та взаємодіє з таблицею користувачів у базі даних, але не змінює інші дані.

Методи доступу до даних реалізуються через репозиторії та сервіси нижнього рівня. Ці методи можуть працювати із запитами до бази даних, вибіркою або іншими операціями, які повинні бути ізольовані від іншої частини програми. Зазвичай, такі методи винесені в окремі бібліотеки системи та можуть залежати від інших методів для роботи з типовими даними.

Проєктування серверної частини інформаційної системи починається з вибору методу доступу до даних. У цій реалізації не буде використовуватися підхід GenericRepository, який зазвичай застосовується як стандартна практика у подібних системах [35]. Натомість було вибрано підхід з використанням окремих репозиторіїв для забезпечення оптимізації запитів до бази даних та відокремлення рівня доступу до даних у вигляді прямих запитів з використанням відповідного фреймворку.

Для рівня обробки даних буде використано шаблон реалізації патерну Медіатор. Він забезпечить верхнім рівням системи спрощену роботу із методами обробки даних та загальну знижену складність конфігурації проєкту. Зменшення складності конфігурації проєкту також позитивно впливає на продуктивність під час запуску та обробки перших запитів до системи, оскільки на запити виділяється оперативна пам'ять та кешуються відповідні дані. Для реалізації патерну Медіатор буде використано бібліотеку MediatR, логіка роботи якої зображена на рисунку 2.3.

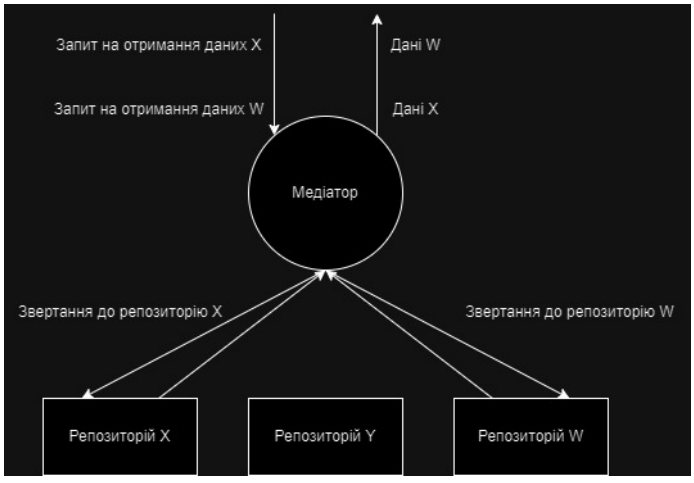


Рисунок 2.3 – Схема роботи медіатора

Ця схема ілюструє процес обробки різних запитів, які проходять через інтерфейс медіатора. Основною перевагою роботи з медіатором є єдина точка входу, через яку переправляються всі запити. Інтерфейс медіатора використовує внутрішню маршрутизацію для виклику необхідного методу, який приймає на вхід конкретний запит. Зазвичай, використання медіатора комбінується з підходом CQRS (Command Query Responsibility Segregation). Цей підхід передбачає використання спеціальних об’єктів DTO (Data Transfer Objects) для передачі даних до відповідного обробника запиту [36-38].

Переваги використання медіатора:

- єдина точка входу для запитів – спрощує архітектуру системи та забезпечує централізовану обробку запитів;
- чітке розмежування логіки – відокремлює логіку обробки запитів від контролерів, що підвищує підтримуваність коду;
- гнучкість та масштабованість – легко додавати нові типи запитів та обробників без значних змін у загальній структурі системи;
- полегшене тестування – забезпечує можливість окремого тестування обробників запитів, що спрощує процес написання тестів та підвищує надійність системи.

Впровадження цього підходу дозволяє створити добре структуровану, масштабовану та легку у підтримці серверну частину інформаційної системи.

2.3 Проєктування користувацього інтерфейсу та методів зв’язку з сервером

Розробка плану тестування та валідації системи є критично важливою частиною процесу розробки інформаційної системи обміну даними мережі аптек. Цей план визначає методики, стратегії та процедури, які будуть використані для перевірки функціональності, надійності та відповідності системи вимогам.

Початковим кроком у розробці плану тестування є аналіз вимог до системи, який визначає очікувані результати та функціональні можливості системи. На основі цього аналізу формулюються тести на функціональність, що перевіряють, чи відповідає система вимогам користувача та бізнес-вимогам. Крім того, розробляються тести на витривалість, які перевіряють, як система працює під навантаженням та при тривалому використанні.

План тестування також включає в себе стратегії тестування безпеки, які перевіряють вразливості системи та забезпечують захист від потенційних загроз. Окрему увагу приділяють тестуванню сумісності, щоб переконатися, що система працює на різних платформах та середовищах. Важливою складовою плану тестування є валідація, яка включає упевнення, що система відповідає стандартам якості та вимогам користувача. Це здійснюється через ретельне порівняння результатів тестів з вимогами та аналіз результатів для виявлення потенційних проблем. Завершальним етапом розробки плану тестування є складання документації, яка включає опис методик тестування, розклад тестів, структуру тестових випадків та очікувані результати. Ця документація є важливим ресурсом для тестерів та інших учасників проекту та забезпечує консистентність та ефективність процесу тестування.

Після розробки плану тестування і валідації системи, необхідно забезпечити його виконання та проведення тестів відповідно до визначених процедур. Для цього створюються тестові набори, які включають набір тестових сценаріїв та вхідних даних для проведення тестів. Кожен тестовий сценарій має чіткий опис тестової дії, очікуваного результату та процедури для виконання тесту. Під час виконання тестів збираються результати, які аналізуються та документуються. Виявлені дефекти та недоліки фіксуються у системі управління помилками (наприклад, системі відстеження помилок) та передаються розробникам для виправлення. Після виправлення дефектів та недоліків тести повторюються для перевірки ефективності виправлень.

Важливою частиною процесу тестування є також оцінка покриття тестами, яка визначає, які частини системи були протестовані, а які залишились непротестованими. Це допомагає визначити, наскільки вичерпно було проведено тестування та які частини системи можуть вимагати додаткового тестування. Загальний результат процесу тестування є підтвердження відповідності системи вимогам, а також ідентифікація та виправлення дефектів та недоліків. Це забезпечує впевненість у функціональності та надійності системи перед її впровадженням у виробниче середовище. Оцінка витрат на розробку та впровадження системи є ключовим етапом у процесі реалізації будь-якого проєкту інформаційної системи. Ця оцінка включає в себе ретельний аналіз ресурсів, необхідних для розробки та впровадження системи, таких як програмне забезпечення, обладнання, ресурси персоналу, а також витрати на навчання та підтримку системи. Наприклад, витрати на програмне забезпечення можуть включати в себе вартість ліцензій на необхідне програмне забезпечення або витрати на розробку власного програмного забезпечення. Обладнання може охоплювати витрати на сервери, комп'ютери, мережеве обладнання та інше необхідне обладнання для роботи системи.

Крім того, оцінка витрат також передбачає врахування витрат на ресурси персоналу, таких як зарплати, відрахування та інші витрати, пов'язані з роботою персоналу, який буде займатися розробкою та впровадженням системи. Навчання персоналу також може бути значною витратою, оскільки необхідно забезпечити, щоб персонал мав необхідні знання та навички для ефективного використання системи. На окремий розділ витрат також варто виокремити витрати на підтримку системи, які можуть включати в себе витрати на технічну підтримку, оновлення та усунення несправностей.

Паралельно з оцінкою витрат проводиться аналіз потенційних економічних вигод від використання системи. Цей аналіз оцінює ефективність вкладених коштів, прибутковість інвестицій та можливість отримання конкурентних переваг від використання інформаційної системи. Наприклад, впровадження ефективної

системи обміну даними мережі аптек може призвести до зменшення витрат на управління запасами, оптимізації логістики та підвищення обсягів продажів через покращення обслуговування клієнтів. Такі економічні вигоди можуть переважити витрати на розробку та впровадження системи, забезпечуючи позитивний вплив на бізнес і підвищення його конкурентоспроможності на ринку.

Забезпечення обґрунтування та оцінки ризиків є важливим етапом у керуванні будь-яким проєктом, включаючи розробку інформаційних систем. Ризики можуть виникнути на будь-якому етапі проєкту і можуть вплинути на його успішне завершення, вчасність та витрати. Тому важливо виявити, оцінити та керувати ризиками відповідно до їх потенційного впливу на проєкт. Першим кроком у процесі забезпечення обґрунтування та оцінки ризиків є ідентифікація потенційних загроз. Це включає аналіз можливих проблем, небезпек та невдач, які можуть виникнути під час розробки та впровадження системи обміну даними мережі аптек. Наприклад, це може бути нестабільність програмного забезпечення, технічні проблеми з інтеграцією з іншими системами аптек чи забезпеченням безпеки даних.

Після ідентифікації ризиків проводиться їх оцінка, включаючи визначення ймовірності виникнення та впливу на проєкт. Це дозволяє визначити, які ризики потребують найбільшої уваги та ресурсів для їх мінімізації чи управління. Після цього розробляються стратегії мінімізації, управління та мітігації ризиків. Це включає в себе розробку планів дій для запобігання або зменшення впливу ризикових подій на проєкт. Наприклад, це може бути створення альтернативних планів дій у разі виникнення проблем, залучення додаткових ресурсів для реалізації певних заходів з безпеки, або переговори з потенційними постачальниками щодо умов співпраці. Ретельний аналіз ризиків та впровадження відповідних стратегій допомагає забезпечити надійність та стабільність системи під час її реалізації та експлуатації. Це дозволяє зберегти якість проєкту, виконання вчасно та в межах бюджету, а також підвищує ймовірність досягнення його цілей.

2.4 Висновок

Розробка інформаційної системи для обміну даними в мережі аптек є важливим етапом в удосконаленні їхнього управління та обслуговування. Ця система не лише автоматизує всі основні бізнес-процеси, а й робить їх більш ефективними та швидкими, забезпечуючи одночасно точне та надійне оновлення даних. Основні компоненти розроблюваної системи включають рівень контролерів для керування процесами в системі, сервісний рівень для надання різноманітних послуг та рівень доступу до даних, що забезпечує ефективне управління інформацією та її захист. Використання PostgreSQL як основної бази даних гарантує високу надійність, стабільність та можливість масштабування, що особливо важливо в умовах зростаючого обсягу інформації в мережі аптек. Технології ASP.NET Core і Entity Framework Core в свою чергу дозволяють значно підвищити продуктивність системи, забезпечити її ефективність та гнучкість в розвитку. Впровадження такої інформаційної системи суттєво покращує процеси обміну даними між аптеками, сприяє підвищенню рівня обслуговування клієнтів та оптимізації управління запасами. Автоматизація ключових процесів дозволяє знизити витрати на адміністрування і підтримку системи, забезпечує високу точність та надійність інформації, що є критично важливим для успішної роботи аптечної мережі в сучасному бізнес-середовищі.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ СТВОРЕННЯ І ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

3.1 Робота з базою даних

Реалізація серверної частини має включати в себе реалізацію різних програмних компонентів для обробки даних, до яких відноситься реалізація CRUD операцій для різних даних [21]. Уся серверна частина розділена на різні рівні, серед яких виділяють два, інколи більше, рівнів, які використовуються для роботи з базою даних. У проєкті, який буде реалізовано в ході виконання кваліфікаційної роботи, буде використавно реалізацію з двома рівнями доступу до даних.

Проєкт буде носити назву PharmaCheck що наближено точно наближає його до дійсного призначення та цілі використання. У даному проєкті міститимуться дві бібліотеки із відповідними назвами PharmaCheck.Database та PharmaCheck.EntityFramework. Розробка рівня доступу до даних розпочинається з проєктування бази даних з використанням підходу Code First. Підхід Code First передбачає проєктування бази даних за допомогою програмного коду на основі якого буде побудована база даних та відповідні зв’язки між таблицями. Для цього використовується фреймворк Entity Framework Core [24].

Проєктування бази даних за допомогою фреймворку Entity Framework передбачає використання цільового класу DbContext для зв’язку, налаштування на конфігурації бази даних. Підключення до бази даних використовується за допомогою так званого рядку підключення. Для збереження рядка підключення в режимі розробки використовуються статичні файли конфігурації у форматі .json. Фреймворк asp.net надає стандартний файл конфігурації «З коробки» який має назву appsettings.json [39]. На лістингу 3.1 представлена структура даного файлу конфігурації.

Лістинг 3.1 – Файл конфігурації

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=127.0.0.1;Host=localhost;Database=PharmacyDb;Port=5
432;username=postgres;password=admin;Include Error Detail =
true;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "JwtOptions": {
    "SecurityKey":
"Knhd&rX]ESzI_B=q4rwq6M#9#&pj{HQTRp*Ou!!13#>m_}5qqD7.vSLFg+
9*a)!",
    "ExpirationTime": 86400
  }
}
```

Даний файл має формат JSON об’єкту з різними секціями. Цільовою секцією для встановлення підключення до бази даних являється секція ConnectionStrings та підксекція DefaultConnection. Значення цієї секції зберігає рядок для підключення до бази даних, у якому зазначено хост та порт бази даних, а також псевдонім користувача для підключення та пароль. Окремим значенням додано «Include Error Detail» для забезпечення розширеної інформації щодо помилок при роботі з базою даних у режимо розробки [40]. Для використання даного рядка, потрібно використати звертання до стандартного файлу конфігурації в файлі конфігурації проєкту Program.cs використовуючи методи об’єкту ConfigurationManager. Даний рядок підключення потрібно передати у відповідний метод при реєстрації контексту бази даних, який був описаниний вище як це представлено на лістингу 3.2.

Лістинг 3.2 – Використання рядку підключення

```
services.AddDbContext<ApplicationDbContext>(options =>
{
    string? connectionString =
builder.Configuration.GetSection("ConnectionStrings:Default
Connection").Value;
    if (string.IsNullOrEmpty(connectionString))
    {
        throw new NpgsqlException("Connection string is
unavailable");
    }
    options.UseNpgsql(connectionString!);
});
```

На цьому лістингу представлено реєстрації контексту бази даних та виклик конструктора відповідного класу для виконання підключення до бази даних. Перед усі потрібно розглянути реалізацію контексту бази даних для подального аналізу та реалізації системи, який представлений на лістингу 3.3.

Лістинг 3.3 – Контекст бази даних

```
public class ApplicationDbContext : DbContext
{
    public DbSet<PharmacyEntity> Pharmacies { get; set; }
    public DbSet<CategoryEntity> ProductCategories { get;
set; }
    public DbSet<ProductEntity> Products { get; set; }
    public DbSet<ProductTypeEntity> ProductTypes { get;
set; }
    public DbSet<SupplierEntity> Suppliers { get; set; }
    public DbSet<SupplyEntity> Supplies { get; set; }
    public DbSet<ProductSuppliesEntity> ProductSupplies {
get; set; }
    public DbSet<UserEntity> Users { get; set; }
    public DbSet<ClientEntity> Clients { get; set; }
    public DbSet<CheckEntity> Checks { get; set; }
    public DbSet<ProductCheckEntity> ProductChecks { get;
set; }
    public DbSet<PharmacyProductsEntity> PharmacyProducts {
get; set; }
```

Продовження лістингу 3.3

```
    public
ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options) : base(options)
    {
        Database.Migrate();
    }

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new
PharmacyConfiguration());
        modelBuilder.ApplyConfiguration(new
CategoryConfiguration());
        modelBuilder.ApplyConfiguration(new
ProductConfiguration());
        modelBuilder.ApplyConfiguration(new
ProductTypeConfiguration());
        modelBuilder.ApplyConfiguration(new
SupplierConfiguration());
        modelBuilder.ApplyConfiguration(new
SupplyConfiguration());
        modelBuilder.ApplyConfiguration(new
ProductSuppliesConfiguration());
        modelBuilder.ApplyConfiguration(new
UserConfiguration());
        modelBuilder.ApplyConfiguration(new
ClientConfiguration());
        modelBuilder.ApplyConfiguration(new
CheckConfiguration());
        modelBuilder.ApplyConfiguration(new
ProductCheckConfiguration());
        modelBuilder.ApplyConfiguration(new
PharmacyProductsConfiguration());
    }
}
```

Конструктор контексту потребує об’єкту типу DbContextOptions<T> який в подальшому передається в конструктор базового класу [42]. Виклик даного конструктора відбувається в коді нижнього рівня самого фреймворку, коли ми

викликаємо код з лістингу 3.2 передаючи рядок підключення з використанням відповідного методу. Також потрібно зазначити, що рядок підключення який представлено на лістингу коду 3.1 використовується для підключення саме до локальної бази даних PostgreSql, яка була вибрана в якості цільової бази даних на етапі планування та проєктування у відповідних розділах.

Контекст бази даних формує представлення про базу даних за допомогою вбуваних типів обгортаючи їх в загальний тип DbSet<T> [41]. Вони повинні бути представлені в якості публічних властивостей контесту бази даних. Також контекст бази даних, для явного вказання конфігурації таблиці, повинен перевизначати метод OnModelCreating. Усі конфігурації винесені в окрему папку. На рисунку 3.1 зображений перелік усі доступних конфігурацій у файловому вигляді.

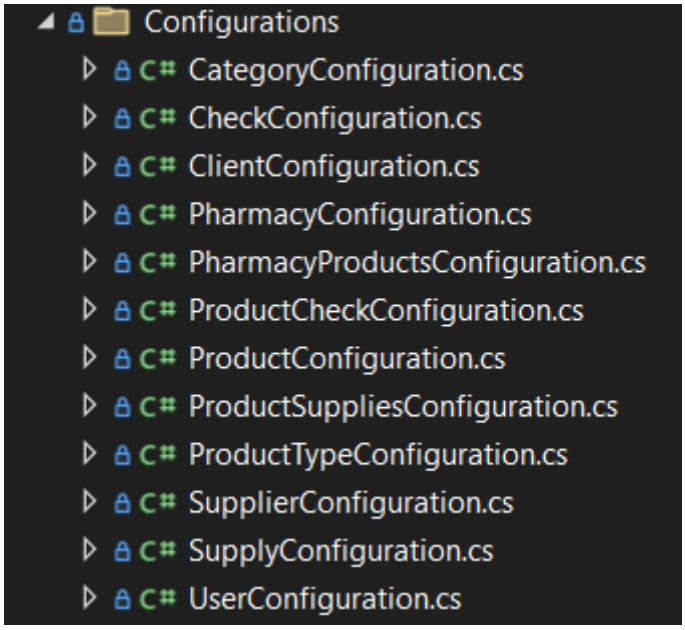


Рисунок 3.1 – Конфігурації таблиць у файловоми представлені

Використання даних конфігурацій представлено на лістнгу 3.3 у перевизначеному методі OnModelCreating [43]. Даний підхід спрощує читання коду та аналіз конфігурацій таблиць що спрощує їх доповнення та редагування.

Щоб створити базу даних та таблиці, а також зв’язати їх за допомогою конфігурацій, маючи готовий контекст бази даних як на лістингу коду 3.3 потрібно

зробити «міграцію». Міграція бази даних це процес зміни структури бази даних, таблиці або зв'язків між ними [44]. Фреймворк Entity Framework надає готовий інструментарій для побудови та виконання міграцій за допомогою бібліотеки Microsoft.EntityFrameworkCore.Tools [45], який потрібно встановити через менеджер пакетів NuGet у відповідний проєкт, де знаходиться контекст бази даних. Після цього потрібно відкрити консоль менеджера пакетів та виконати команди add-migration з назвою міграції та update-database відповідно. Після виконання зазначених команд база даних буде створена, а зв'язки налаштовані. Для забезпечення актуальності бази даних, при передачі проєкт з одного пристрою на інший, конструктор контексту бази даних повинен викликати відповідний метод для відстворення міграцій. Усі міграції зберігаються в окрему папку Migrations у відповідному проєкті, з якого було виконано команди міграцій. Наступні міграції повині виконуватись при зміні будь яких типів або структур, які мають будь-яке значення для відображення їх в базі даних або її структури.

Entity Framework забепечує співставлення таблиць бази даних із відповідними типами, які були зазначені в контексті бази даних. В цьому контексті дані типи носять назву Сутності та описуються в окремому проєкті PharmaCheck.Database. Файловий список усі сутностей зображено на рисунку 3.2. Даний список сутностей також можна зустріти в лістингу коду 3.3 в якості внутрішніх типів для обгортки DbSet<T> властивостей контексту бази даних.

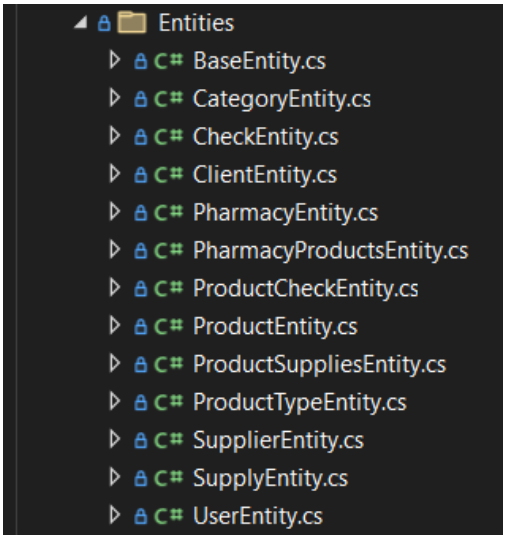


Рисунок 3.2 – Сутності бази даних

Налаштування зв'язків між сутностями виконується шляхов сказання посилання з одного типу на інший, а також явне вказання цього зв'язку за допомогою методів Entity Framework у відповідних конфігураціях на рисунку 3.1. Приклад такої сутності представлено на лістингу коду 3.4, а її конфігурація на 3.5 відповідно.

Лістинг 3.4 – Сутність продукту

```
public sealed record ProductEntity : BaseEntity
{
    public string Name { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
    public string Manufacturer { get; set; } =
string.Empty;
    public float Price { get; set; }
    public int Count { get; set; }

    public Guid TypeId { get; set; }
    public ProductTypeEntity ProductType { get; set; }

    public List<PharmacyProductsEntity> Pharmacies { get;
set; } = new List<PharmacyProductsEntity>();
    public List<ProductCheckEntity> Checks { get; set; } =
new List<ProductCheckEntity>();
    public List<ProductSuppliesEntity> Supplies { get; set;
} = new List<ProductSuppliesEntity>();
}
```

Також потрібно звернути увагу на наслідування від базового типу BaseEntity. Тип BaseEntity містить загальні властивості усіх сутностей, такі як дата створення, дата останнього редагування та первинний ключ типу даних Guid [46]. Це поширена та допустима практика при проєктуванні бази даних з використанням Entity Framework.

Лістинг 3.5 – Конфігурація таблиці продуктів

```
public sealed class ProductConfiguration :
IEntityTypeConfiguration<ProductEntity>
{
```

Продовження лістингу 3.5

```
    public void Configure(EntityTypeBuilder<ProductEntity>
builder)
    {
        builder.ToTable("Products").HasKey(product =>
product.Id);

        builder.HasOne<ProductTypeEntity>(product =>
product.ProductType)
            .WithMany(type => type.Products)
            .HasForeignKey(product => product.TypeId);
    }
}
```

У цій конфігурації опущено налаштування колонок відповідної таблиця, зокрема таких властивостей як максимальна довжина, назва колонки, тип даних тощо.

На цьому етапі розробка шару зберігання та доступу до даних вважається виконано, але потрібно звернути увагу на наявність різних реалізацій репозиторіїв для доступу до даних та винесення загальних операцій з запитами на рівень доступу до даних, але ця реалізація являється лише перебіжним процесом між доступом до даних та їх обробкою, оскільки використовує методи з різних частин програми.

3.2 Реалізація сервернох частини обробки даних інформаційної системи

У цьому підрозділі будуть розглянуті основні аспекти обробки інформації та основна логіка роботи різного цільового функціоналу інформаційної системи, зокрема наступний функціонал:

- створення та редагування продуктів;
- створення та управління постачальниками;
- реєсрація доставки продуктів;
- доставка продуктів;

— продаж продуктів;

— переміщення продуктів між точками або складом.

При плануванні прошарку обробки даних використовується підхід CQRS + Mediator, що дає змогу відійти від сервісного підходу, який є стандартним у подібних системах та більшеш акцентувати увагі саме на запитах та їх обробці. Прошарок обробки даних зв'язується з базою даних використовуючи різні реалізації репозиторіїв, куди, фактично, винесена основна логіка обробки програми. Кожна сутність має окремий репозиторій який працює з окермою таблицею у базі даних використовуючи реалізацію DbSet<T>. Наявність такої реалізації репозиторія відокремлює використання типу IQueriable<T> для запитів до бази даних на рівні прошарку доступу до даних [47]. Приклад такого репозиторію для сутності продукту представлено на лістингу 3.6.

Лістинг 3.6 – Репозиторій сутності продукту

```
public sealed class ProductRepository
{
    private const int PAGE_VOLUME = 50;

    private readonly ApplicationDbContext _dbContext;
    private readonly DbSet<ProductEntity> _table;

    public ProductRepository(ApplicationDbContext
dbContext)
{
        _dbContext = dbContext;
        _table = _dbContext.Products;
    }

    public async Task Create(ProductEntity entity)
    {
        entity.Id = Guid.NewGuid();
        entity.CreatedAt =
DateTimeOffset.Now.ToUniversalTime();
        entity.UpdatedAt =
DateTimeOffset.Now.ToUniversalTime();

        await _table.AddAsync(entity);
```

```
        await _dbContext.SaveChangesAsync();
    }
```

Продовження лістингу 3.6

```
    public async Task Delete(ProductEntity entity)
    {
        entity.DeletedAt =
DateTimeOffset.Now.ToUniversalTime();

        _table.Update(entity);
        await _dbContext.SaveChangesAsync();
    }

    public async Task Update(ProductEntity entity)
    {
        _table.Update(entity);
        await _dbContext.SaveChangesAsync();
    }

    public async Task<List<ProductEntity>> Get(Guid?
pharmacyId, string? name, string? description, string?
manufacturer, float? minPrice, float? maxPrice, int Page =
1)
    {
        IQueryable<ProductEntity> query =
_table.AsNoTracking();

        query = pharmacyId is not null ?
            query.Where(entity =>
entity.Pharmacies.Where(pp => pp.PharmacyId ==
pharmacyId.Value).Any()) :
            query;

        query = name is not null ?
            query.Where(entity =>
entity.Name.Contains(name)) :
            query;

        query = description is not null ?
            query.Where(entity =>
entity.Description.Contains(description)) :
            query;

        query = manufacturer is not null ?
```

```
            query.Where(entity => entity.Manufacturer ==
manufacturer) :
            query;
```

Продовження лістингу 3.6

```
        query = minPrice is not null ?
            query.Where(entity => entity.Price >=
minPrice.Value) :
            query;

        query = maxPrice is not null ?
            query.Where(entity => entity.Price <=
maxPrice.Value) :
            query;

        int skip = Page <= 1 ? 0 : PAGE_VOLUME * (Page -
1);
        return await
query.Skip(skip).Take(PAGE_VOLUME).ToListAsync();
    }
}
```

Даний репозиторій має в наявності декілька основних методів, які реалізовані явно, а також сам репозиторій не являється наслідником будь якого внутрішнього класу або реалізатором інтерфейсу, тобто, на цьому етапі одразу відкинута абстракція і поліморфізм. Навіщо це потрібно? Це запобіжний метод для оптимізації роботи з даними через явні методи які працюють з інтерфейсом IQueriable<T>. Даний інтерфейс представляє собою лише запит до бази даних, який може корегуватись, додаватись вибірка, сортування тощо. Цей запит може бути багаторівневим, про що свідчиить одна з стандартних реалізація патерну GenericRepository, коли вибірка даних може повертати результат у вигляді IQueriable<T> для подальшого опрацювання запиту. Зазвичай, такий репозиторій використовується в сервісах вищого рівня, що призводить до винесення даних за певну межу відповідного рівня доступу до даних, а значить, надає можливість для додаткових помилок. Звідси можна зробити висновок, що підхід який використовується у даній реалізації являється більш захищеним та професійним підходом.

Репозиторії самі у своїй реалізації не використовуються в середині механізму ін’єкції залежностей, для цього використовується підхід фабрики, реалізація якого представлена на лістингу 3.7.

Лістинг 3.7 – Фабрика репозиторіїв

```
public sealed class RepositoryFactory : IRepositoryFactory
{
    private readonly ApplicationDbContext _dbContext;

    public RepositoryFactory(ApplicationDbContext
dbContext)
    {
        _dbContext = dbContext;
    }

    public ProductTypeRepository NewProductTypeRepository()
=> new(_dbContext);
    public CategoryRepository NewCategoryRepository() =>
new(_dbContext);
    public SupplyRepository NewSupplyRepository() =>
new(_dbContext);
    public SupplierRepository NewSupplierRepository() =>
new(_dbContext);
    public PharmacyRepository NewPharmacyRepository() =>
new(_dbContext);
    public ProductRepository NewProductRepository() =>
new(_dbContext);
    public UserRepository NewUserRepository() =>
new(_dbContext);
    public CheckRepository NewCheckRepository() =>
new(_dbContext);
    public ProductCheckRepository
NewProductCheckRepository() => new(_dbContext);
    public PharmacyProductsRepository
NewPharmacyProductsRepository() => new(_dbContext);
    public ProductSupplyRepository
NewProductSupplyRepository() => new(_dbContext);
}
```

Як можна побачити з представленного коду, фабрика реалізує відповідний інтерфейс що надає можливість використання її в якості залежності в механзні

ін’єкції залежностей. Сама реалізації містить залежність лише контексту бази даних, який використовується для створення об’єктів відповідних репозиторіїв. Дана практика має деякі особливості та переваги, зокрема зменшення кількість залежних об’єктів та явну реалізації різних репозиторів, а також забирає необхідність в реєстрації усіх репозиторіїв в механізмі ін’єкції залежностей.

Використання даного механізму має місце в обробниках медіатора. Для використання підходу CQRS + Mediator не використовується явна реалізація даного патерну, натомість використовується бібліотека MediatR яка його реалізує. Для початку, проаналізуємо роботу такого механізму. Для отримання даних з клієнтської частини системи використовуються типи самого медіатора, тобто, реалізатори інтерфейлу IRequest<T>, що надає змогу одразу перенаправляти отриманий об’єкт в обробник запитів для подальшої обробки. На лістингу 3.8 представлений приклад кінцевої точки ProductController, який виконує функцію додавання товару в чек.

Лістинг 3.8 – Запит на додавання товару в чек

```
   [HttpPost("check/attach")]
    public async Task<IActionResult>
AttachToCheck([FromBody] AttachToCheckRequest request) =>
        await mediator.Send(request).Map<Result,
IActionResult>(
            result => result.IsError ?
            StatusCode((int)result.StatusCode,
ControllerResponse.ToErrorResult(result.ErrorMessage)) :
                NoContent());
```

Даний метод, який представлений у коді вище, являється так званою «кінцевою точкою» серверу та повертає деякі вказані дані, які згодом, за допомогою вбудованого в фреймворк asp.net серіалізатора, серіалізуються в JSON об’єкт для передачі його клієнтській частині системи. Він помічений атрибутом HttpPost з вказанням назви цієї кінцевої точки при побудові маршрутів серверу. HttpPost означає, що цей метод буде реагувати на вказаний тип HTTP запитів,

тобто, на HTTP POST запити [48]. Даний метод являється асинхронним та повертає реалізацію інтерфейсу IActionResult [49]. Дана кінцева точка приймає деякий набір даних, які винесені в окремий клас та помічений атрибутом FromBody для вказання місця, звідки будуть надходити дані, тобто, з тіла самого запиту. Внутрішні десеріалізатор asp.net буде співставляти JSON об'єкт, який надходитиме до кінцевої точки, та об'єктом вказаного типу. При збігу типів, буде ініціалізований вказаний об'єкт, з яким далі виконуватимуться деякі дії в середині методу. Як показани на лістингу 3.8, отриманий запит одразу передається на обробку в медіатор, який, в свою чергу, виконую внутрішню маршрутизацію між обробниками та викликає необхідний з них.

На лістингу коду 3.9 представлено реалізацію обробника запитів медіатора, який виконую функцію додавання товарів у вказаний чек. Даний клас повинен реалізовувати інтерфейс IRequestHandler<in TRequest, TResponse> що являється частиною внутрішньої логіки роботи бібліотеки [51].

Лістинг 3.9 – Обробника запиту додавання товару в чек

```
public sealed class AttachToCheckRequestHandler(
    IRepositoryFactory repositoryFactory)
    : IRequestHandler<AttachToCheckRequest, Result>
{
    private const string CheckOrProductNotFoundError =
"Check or product not found.";

    public async Task<Result> Handle(AttachToCheckRequest
request, CancellationToken cancellationToken)
    {
        ProductCheckRepository repository =
repositoryFactory.NewProductCheckRepository();

        try
        {
            await repository.Attach(request.CheckId,
request.ProductId);
        }
        Catch
```

Продовження лістингу 3.9

```
        {
            return
Result.Error(CheckOrProductNotFoundError,
ResultErrorStatusCode.NotFound);
        }
        return
Result.Ok(ResultSuccessStatusCode.NoContent);
    }
}
```

З представленого коду випливає висновок про структуру самих обробників. Потрібно звернути увагу також на залежність самого класу, який в конструкторі потребує ін’єкцію фабрики репозиторіїв, який потрібен для відтворення потрібних репозиторіїв. Обробник в середині методу створює необхідниий репозиторій та викликає необхідний метод для обробки запиту. Як можна побачити, сам процес максимально спрощений за рахунок продуманої структури проєкту, відповідних бібліотек та класів. На лістингу 3.10 представлено реалізацію саме цього методу відповідного репозиторію, який викликається обробником в лістингу 3.9.

Лістинг 3.10 – Реалізація додавання товару в чек

```
    public async Task Attach(Guid CheckId, Guid ProductId)
{
    await _table.AddAsync(new ProductCheckEntity()
    {
        CreatedAt =
DateTimeOffset.UtcNow.ToUniversalTime(),
        UpdatedAt =
DateTimeOffset.UtcNow.ToUniversalTime(),
        Id = Guid.NewGuid(),
        CheckId = CheckId,
        ProductId = ProductId
    });
    await _dbContext.SaveChangesAsync();
}
```

З представленого коду вище можемо зробити висновок щодо реалізації даної та подібних операцій щодо робити з базою даних та відповідними таблицями. Потрібно звернути увагу, що таблиця продуктів та чеків пов’язані зв’язком «Багато до Багатьох», тому у цьому коді відбувається робота саме з допоміжною таблицею у цьому зв’язку [52].

Тепер, коли було розглянуто приклад роботи серверу на не складному прикладі, варто розглянути більш складні функції серверу, для прикладу, такі як доставка товару. Обробник такого запиту представлено на лістингу коду 3.11.

Лістинг 3.11 – Доставка продуктів

```
    public async Task<Result> Handle(ApplySupplyRequest
request, CancellationToken cancellationToken)
    {
        SupplyRepository supplyRepository =
factory.NewSupplyRepository();
        SupplyEntity? supply = await
supplyRepository.GetById(request.Id);
        if (supply is null)
        {
            return Result.Error(SupplyNotFoundError,
ResultErrorStatusCode.NotFound);
        }
        PharmacyProductsRepository repository =
factory.NewPharmacyProductsRepository();
        try
        {
            await repository.ApplyRange(supply.PharmacyId,
supply.Products.Select(x => (x.ProductId, x.Count)));
        }
        catch (Exception ex)
        {
            return Result.Error(ApplyError,
ResultErrorStatusCode.InternalError);
        }
        return
Result.Ok(ResultSuccessStatusCode.NoContent);
    }
```

Логіка роботи цього обробника заключається в декількох кроках. Спершу отримується запис про саму доставку відповідного репозиторію, після чого робиться перевірка, чи дійсно такий об’єкт доставки існує. Після цього викликається метод репозиторію допоміжної таблиці між таблицями продуктів та аптек. На лістингу 3.12 представлено реалізацію запиту доставки товарів в аптеку.

Лістинг 3.12 – Доставка товару в аптеку

```
public async Task ApplyRange(Guid pharmacyId,
IEnumerable<(Guid id, int count)> products)
{
    using (var transaction = await
_dbContext.Database.BeginTransactionAsync())
    {
        List<Guid> productIds = products.Select(p =>
p.id).ToList();

        List<PharmacyProductsEntity> entities = await
_table
            .Where(entity => entity.PharmacyId ==
pharmacyId && productIds.Contains(entity.ProductId))
            .ToListAsync();

        foreach (PharmacyProductsEntity entity in entities)
        {
            (Guid id, int count) product =
products.FirstOrDefault(p => p.id == entity.ProductId);
            if (product != default)
            {
                entity.Count = product.count;
            }
        }

        await _dbContext.BulkInsertOrUpdateAsync(entities);
        await _dbContext.SaveChangesAsync();

        IEnumerable<Guid> existingProductIds =
entities.Select(e => e.ProductId).ToHashSet();
        IEnumerable<PharmacyProductsEntity> newProducts =
products.Where(p => !existingProductIds.Contains(p.id))
                                    .Select(p => new
PharmacyProductsEntity
```

```
                                    {
                                        CreatedAt =
DateTimeOffset.Now.ToUniversalTime(),
                                        Id = Guid.NewGuid(),
```

Продовження лістингу 3.12

```
                                        UpdatedAt =
DateTimeOffset.Now.ToUniversalTime(),
                                        PharmacyId =
pharmacyId,
                                        ProductId = p.id,
                                        Count = p.count
                                    });

        if (newProducts.Any())
        {
            await
_dbContext.BulkInsertOrUpdateAsync(newProducts);
            await _dbContext.SaveChangesAsync();
        }

        await transaction.CommitAsync();
    }
}
```

На цьому лістингу коду представлено досить складний для виконання запит до бази даних для оновлення великої кількості даних в рамках одного запиту такі запити робляться в рамках однієї транзації з використанням методів для масового оновлення або створення даних в базі даних, такі методи називаються Bulk методами. Спершу отримується список ідентифікаторів усіх продуктів, які потрібно доставити на точку, після чого отримуються усі продукти відносно цієї аптеки. Наступним кроком є основлення кількості товарів, які є на відповідній точці. Для цього в циклі виконується проходження по всіх продуктах та збільшується як кількість на відповідне значення. Після цього створюються об’єкти тих продуктів, які не були наявні на вказаній аптеці. Для них встановлюється відповідна кількість одиниць, яку потрібно доставити, та вносяться в базу даних. Цей процес розбитий на декілька умовних частин які є сукупною логікою для оновлення даних.

3.3 Користувацький додаток, мікросервісна архітектура та розгортання додатку

Для акцентування уваги на функціональних можливостях інформаційної системи, у даній кваліфікаційній роботі не буде акцентуватись увага на дизайні користувацього інтерфейсу та користувацькому досвіді роботи. Перед початком роботи слід звернути увагу, що користувацький додаток являється окремим проєктом, який потрібно буде зв’язати із центральним сервером у вигляді API серверу. Виходячи із цього, робота у даному підрозділі розпочнеться з розгортання додатку в Docker.

Docker – це технологія для контейнеризації програмного забезпечення в процесі якого додаток розміщується в так званому контейнері з усіма залежностями, які необхідні для його роботи. Після розміщення додатку з контейнері, цей контейнер можна легко передавати або розгортати з використанням програмного забезпечення Docker [53]. Кожний окремий сервіс повинен бути розміщений в оркемому контейнері відповідно, а між ними повинен бути налаштований відповідний зв’язок для здійснення запитів. Спершу потрібно додати Dockerfile для API серверу, це робиться шляхом вибору відповідного елемента меню [54] як на рисунку 3.3. Як цільову операційну систему потрібно вибрати Linux, оскільки більшість проєктів розгортаються саме на ньому, оскільки дана операційна система значно легша ніж операційна система Windows та має деякі дистриибутиви без графічного користувацього інтерфейсу, що також надає значної оптимізації при робочих публікаціяї програмного забезпечення. Також потрібно зазнчити, що є Linux дистрибутиви, які орієнтовані виключно на розгортання та підтримку веб-серверів, тому такий сервер потребує значно менші ресурси для роботи ніж сервери на операційній системі Windows, включаючи навість Windows Server та подібні операційні системи.

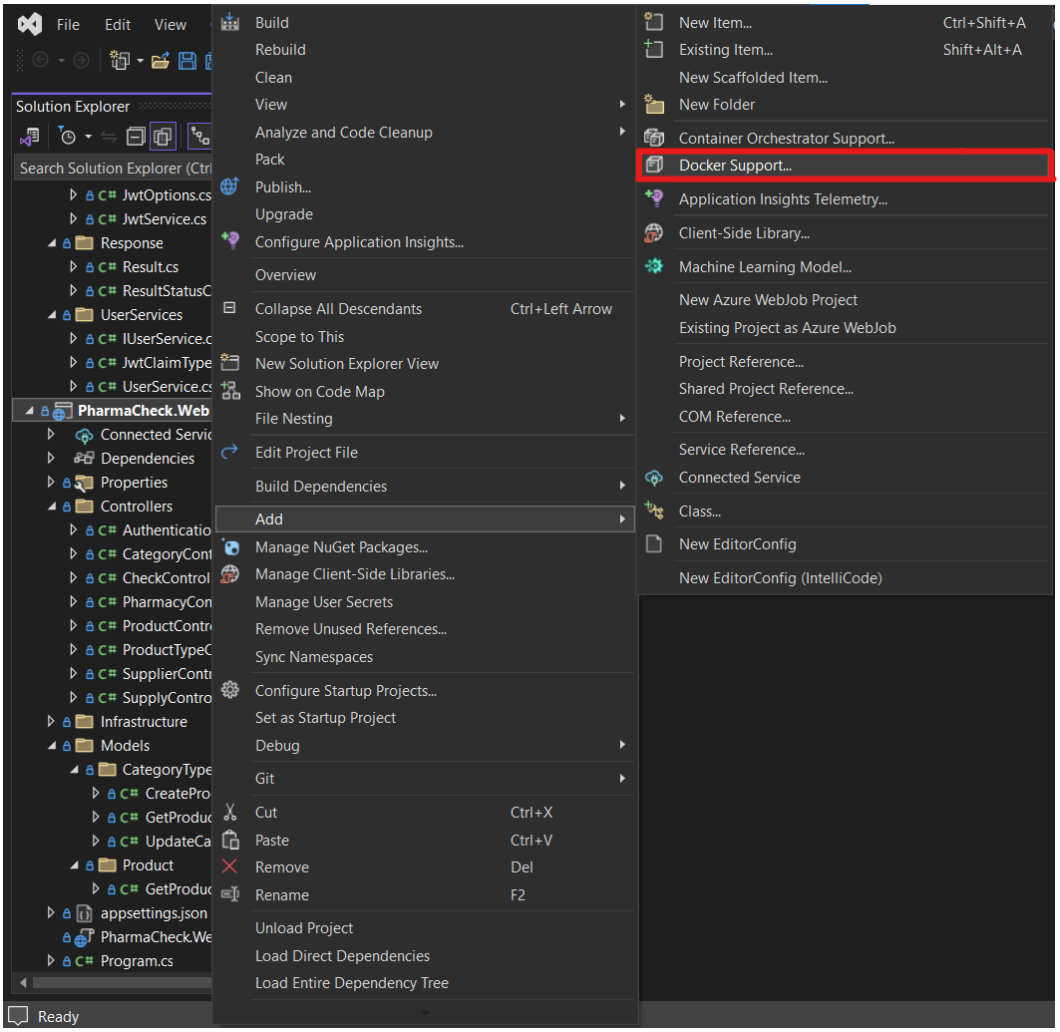


Рисунок 3.3 – Додавання підтримки Docker

Після вибору відповідного елементу меню в корінь рішення буде додано згенерований файл з назвою Dockerfile без розширення, код якого представлено на лістингу 3.X.

Лістинг 3.12 – Вміст Dockerfile

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 5180

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
```

Продовження лістингу 3.12

```
COPY ["PharmaCheck.Web/PharmaCheck.Web.csproj",
"PharmaCheck.Web/"]
COPY ["PharmaCheck.Domain/PharmaCheck.Domain.csproj",
"PharmaCheck.Domain/"]
COPY ["PharmaCheck.Services/PharmaCheck.Services.csproj",
"PharmaCheck.Services/"]
COPY
["PharmaCheck.EntityFramework/PharmaCheck.EntityFramework.c
sproj", "PharmaCheck.EntityFramework/"]
COPY ["PharmaCheck.Database/PharmaCheck.Database.csproj",
"PharmaCheck.Database/"]
RUN dotnet restore
"./PharmaCheck.Web/./PharmaCheck.Web.csproj"
COPY . .
WORKDIR "/src/PharmaCheck.Web"
RUN dotnet build "./PharmaCheck.Web.csproj" -c
$BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./PharmaCheck.Web.csproj" -c
$BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "PharmaCheck.Web.dll"]
```

Для запуску мікросервісних програм використовуються різні Dockerfile для кожного сервісу окремо та запускаються за допомогою одного docker-compose. Наразі в системі наявний лише один сервіс для запуску, але є сенс спершу зайнятися розгортанням саме його. Після написання docker-compose його код буде відповідним до лістингу коду 3.13.

Лістинг 3.13 – Файл запуску docker-compose

```
services:
  api:
    build:
      context: .
```

Продовження лістингу 3.13

```
      dockerfile: Dockerfile
    ports:
      - 5180:5180
    networks:
      - pharmacy

networks:
  pharmacy:
    driver: bridge
```

Наступний крок відповідає розробці користувацької програми для використання API серверу. Для спрощення написання користувацього інтерфейсу буде використання безкоштовний шаблон типу адміністративної панелі, який, зазвичай, використовується для реалізації користувацього інтерфейсу в програмах подібних до CRM систем. Після встановлення шаблону його слід переробити під потреби інформаційної системи з використанням фреймворку Bootstrap. Вигляд головного меню зображено на рисунку 3.4.

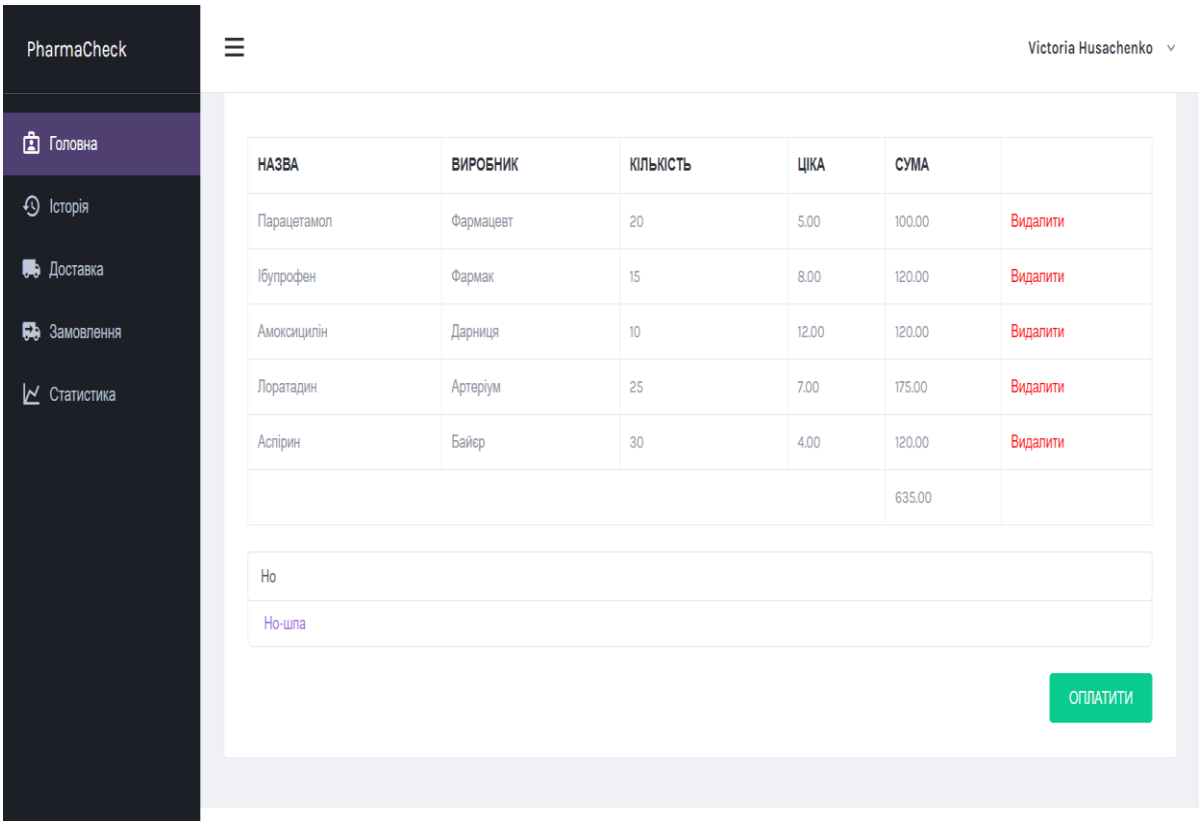


Рисунок 3.4 – Головна сторінка

Потрібно звернути увагу, що головна сторінка являється місцем для створення нового чека з можливістю додавання товарів та оплати самого чеку. Це зроблено задля полегшення роботи з системою та забезпечення швидкої навігації в програмі.На головній сторінці користувач може швидко й ефективно створити чек, додаючи необхідні товари з каталогу, а також проводити оплату безпосередньо в інтерфейсі програми. Крім того, на головній сторінці реалізована зручна система пошуку та додавання клієнтів. Це особливо корисно для бізнесів, де часто обслуговують постійних клієнтів або де потрібна швидка ідентифікація покупців для надання спеціальних умов або знижок. Дана форма також надає можливість додати клієнта по номеру телефону або зареєструвати нового клієнта в системі. Це дозволяє швидко знаходити існуючих клієнтів за номером телефону, що зручно для персоналу, особливо у великих магазинах чи ресторанах з великою кількістю постійних відвідувачів. На рисунку 3.5 зображено відповідне модальне вікно із пошуком клієнта по номеру телефону. Модальне вікно має інтуїтивно зрозумілий інтерфейс, де користувач може ввести номер телефону клієнта, після чого система автоматично знайде та відобразить інформацію про клієнта, якщо він вже зареєстрований в базі даних. У разі, якщо клієнт не знайдений, система пропонує можливість швидкої реєстрації нового клієнта, що мінімізує час обслуговування і покращує загальний користувацький досвід.

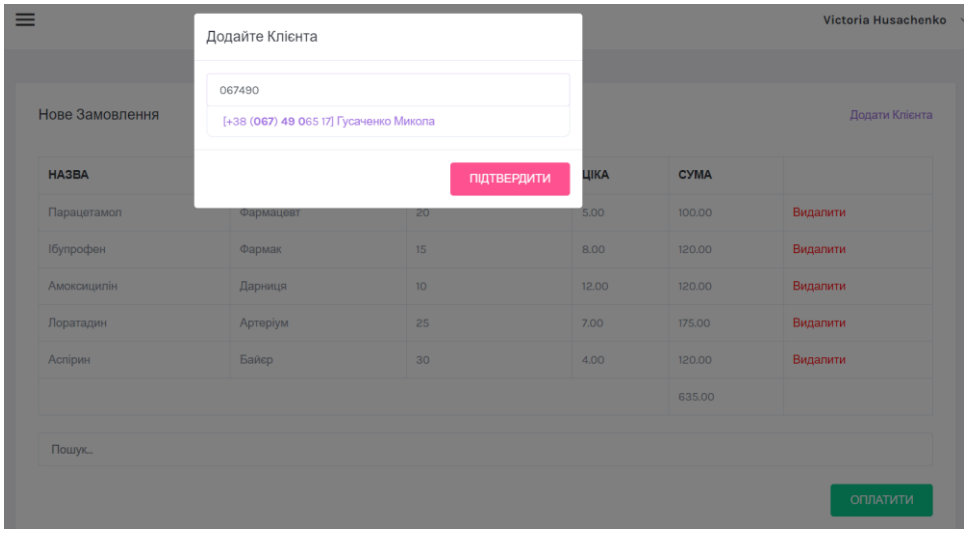


Рисунок 3.5 – Пошук клієнта

Кожний клієнт має може мати свою особисту знижку завждяка бонусним картам та іншим бонусним пропозиціям, які можуть додаватись в ході покращення та доповнення інформаційної системи. На рисунку 3.6 зображено знижену ціну на товарний чек встановленому користувачу.

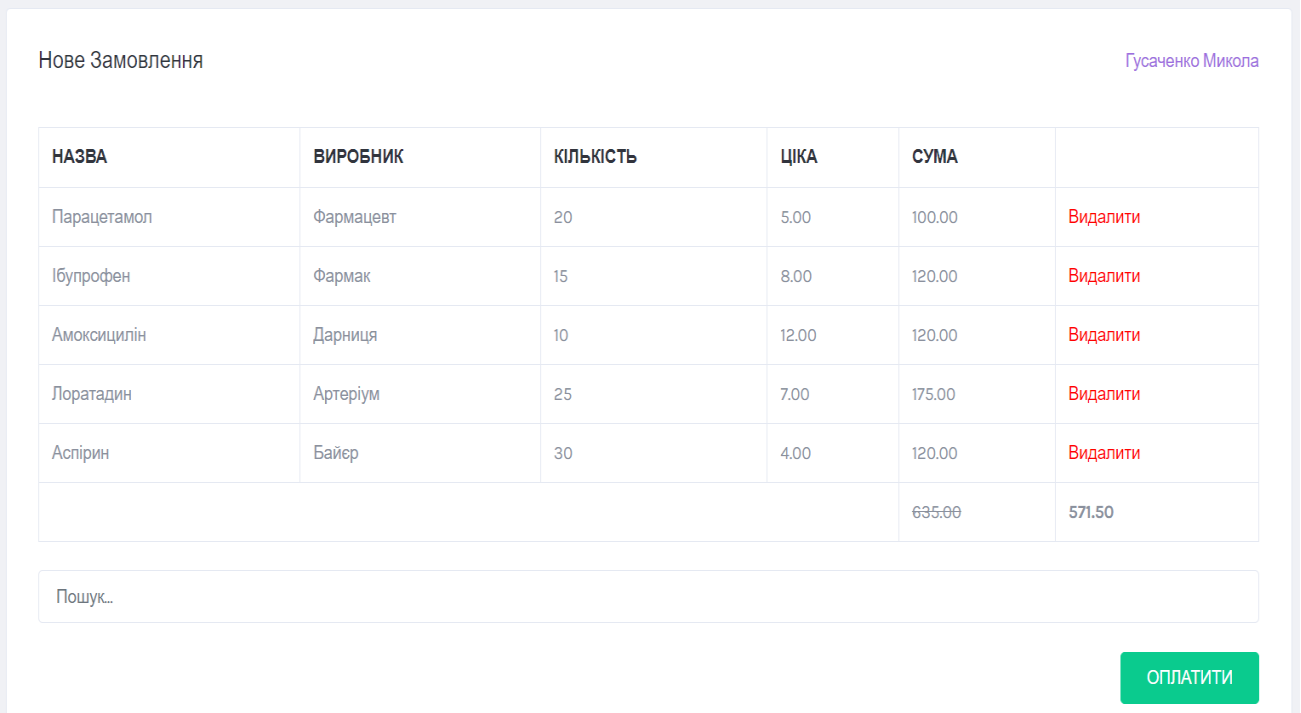


Рисунок 3.6 – Зниження суми

Одна з ключових можливостей інформаційної системи – це пошук та бронювання відповідних ліків та препаратів по всій мережі аптек або в інших мережах аптек. Така функціональність значно підвищує зручність для користувачів, дозволяючи їм швидко знайти потрібний товар, перевірити його наявність та забронювати для подальшого отримання в найближчій аптеці. Функціонал пошуку реалізований таким чином, що клієнтський додаток надсилає запит на спеціальну кінцеву точку API з пошуком товару за іменем при введенні кожного символу в рядок пошуку. Це забезпечує динамічний пошук в реальному часі, коли користувач отримує миттєві результати з пропозиціями щодо відповідних ліків та препаратів. Такий підхід дозволяє зменшити час на пошук потрібного товару та підвищує ефективність використання системи. Додатково,

система може надавати інформацію про наявність товару в різних аптеках, включаючи кількість одиниць на складі, можливість резервування та терміни доставки. Це дозволяє користувачам не тільки знаходити необхідні ліки, але й оперативно планувати їх отримання, що особливо важливо в умовах дефіциту або при потребі в негайному лікуванні. На рисунку 3.7 зображений результат одного із таких пошуків, де можна побачити, як система відображає список доступних ліків у відповідь на введений запит.

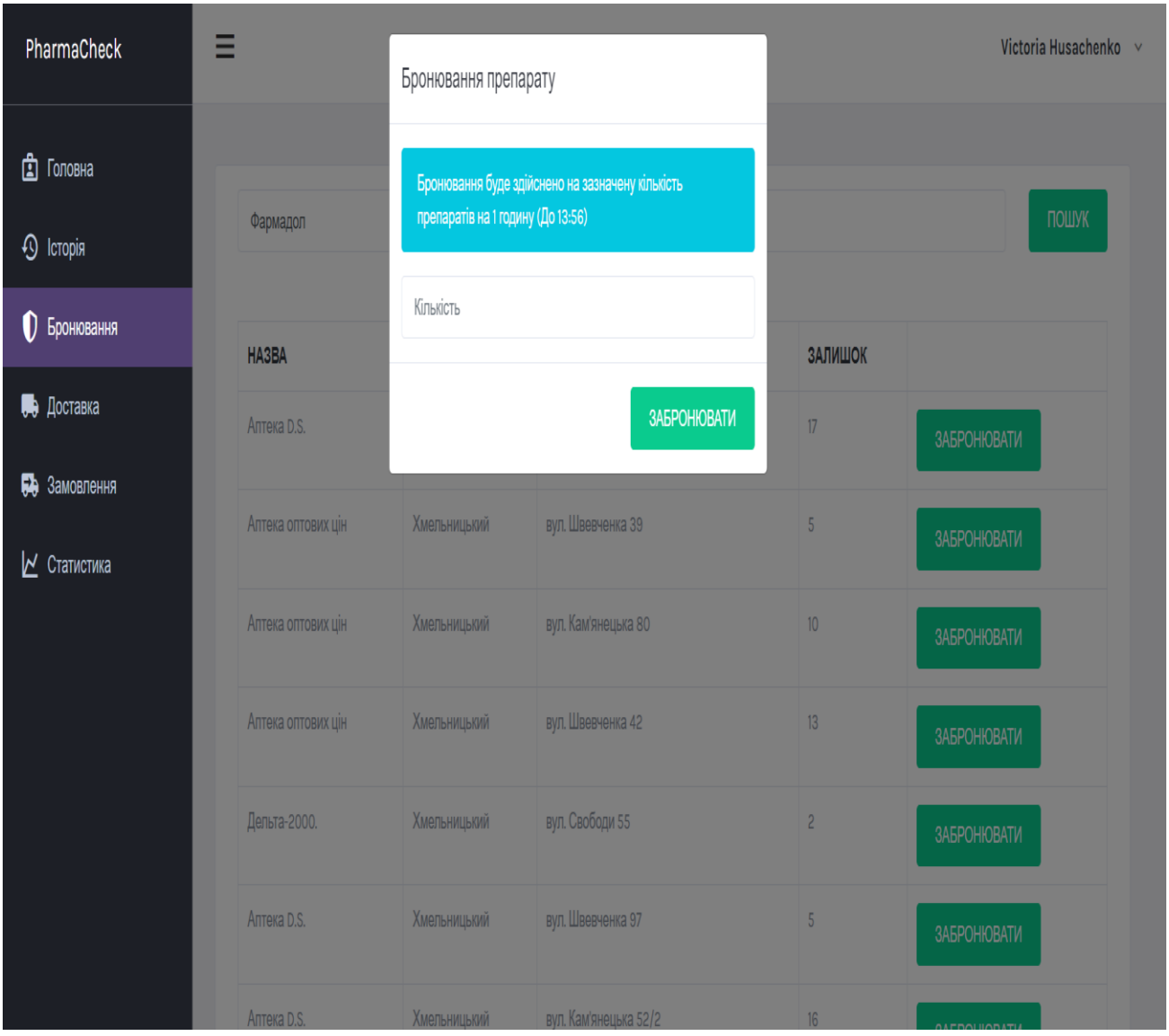


Рисунок 3.7 – Результат пошуку

На даному рисунку також можна побачити процес бронювання за вказаною адресою. При натисканні на кнопку пошуку робить запит на API сервер. Після успішного запиту дані виводяться в таблицю з вказанням інформації про наявні препарати. Бронювання робить динамічно, оскільки аптека, в якій бронюються препарати автоматично отримує сповіщення про дану дію. Задля забезпечення захисту від зловживання, бронювання ліків здійснюється лише на одну годину.

3.4 Висновок

Розділ, присвячений реалізації та тестуванню інструментальних засобів для створення і використання інформаційних систем, детально розкриває ключові аспекти розробки системи для обміну даними в мережі аптек. Впровадження цієї системи сприяє автоматизації бізнес-процесів, що зменшує потребу в ручній праці та мінімізує ймовірність помилок. Завдяки надійному зберіганню та обробці даних, система забезпечує своєчасне та точне оновлення інформації, що є критично важливим для ефективного управління запасами та обслуговування клієнтів.

Під час тестування було підтверджено функціональність системи, її надійність та готовність до використання в реальних умовах. Використання сучасних технологій, таких як ASP.NET Core, Entity Framework Core та PostgreSQL, дозволяє досягти високої продуктивності та масштабованості. Інформаційна система для обміну даними в мережі аптек не тільки покращує внутрішні процеси управління, але й підвищує загальний рівень обслуговування клієнтів, що є важливим для успіху аптечного бізнесу.

Таким чином, створена інформаційна система відповідає всім вимогам сучасного аптечного бізнесу, забезпечуючи ефективний обмін даними, оптимізацію бізнес-процесів та підвищення задоволеності клієнтів.

ВИСНОВКИ

У цій кваліфікаційній роботі було розроблено інформаційну систему «Обмін даними мережі аптек», яка призначена для автоматизації процесів обміну інформацією між аптеками в межах однієї мережі. Система забезпечує швидкий та безпечний обмін даними, оптимізує бізнес-процеси, підвищує ефективність роботи аптек та покращує якість обслуговування клієнтів.

Для реалізації поставленої мети було використано методи аналізу та проектування, включаючи визначення вимог до системи, розробку архітектури та проектування бази даних. На етапі програмування було реалізовано серверну частину на основі сучасних веб-технологій, таких як ASP.NET Core Web API і Entity Framework Core, та клієнтський інтерфейс на основі HTML та CSS. Впроваджено механізми захисту даних, включаючи шифрування, автентифікацію та авторизацію. Проведено функціональне та нефункціональне тестування для забезпечення стабільності та надійності системи.

У першому розділі кваліфікаційної роботи було проведено аналіз сучасного стану інформаційних систем, що включало дослідження існуючих рішень для управління аптечними мережами та визначення ключових проблем і вимог до нової системи. На основі цього аналізу було сформульовано функціональні та нефункціональні вимоги до системи та розроблено її архітектуру, включаючи базу даних, серверну та клієнтську частини.

У другому розділі було виконано проєктування інформаційної системи, зокрема аналіз даних для збереження, визначення схеми даних бази даних, проєктування взаємодії серверу та внутрішньої маршрутизації запитів та проєктування користувацького інтерфейсу.

У третьому розділі було розроблена та реалізована безпосередньо система для управління інформаційною. Реалізація включає в себе розробку бази даних та зв’язок з нею. Обробку інформації з бази даних та відправлення клієнтському додатку та обробку інформації з клієнтського додатку перед внесенням їх в базу

даних. Також було розроблено клієнтський додаток за допомогою Html, Css та JavaScript.

Система забезпечила швидкий та безпечний обмін даними між аптеками, що дозволило зменшити час на обробку замовлень та підвищити точність обліку лікарських засобів. Автоматизація процесів дозволила скоротити витрати часу та зменшити кількість людських помилок, що позитивно вплинуло на фінансові показники аптечної мережі. Завдяки системі клієнти отримали швидше та якісніше обслуговування, що підвищило їх задоволеність та лояльність.

Впровадження розробленої системи дозволить скоротити затрати часу на обробку даних та управління запасами, заощадити людські та грошові ресурси шляхом автоматизації рутинних процесів, підвищити ефективність управління аптечними мережами, прискорити швидкість та покращити якість обслуговування клієнтів.

Розроблений інструмент може бути використаний не лише у мережах аптек, але й у інших галузях, де необхідний ефективний обмін даними між структурними підрозділами, таких як логістика, торгівля, охорона здоров'я та інші. Результати впровадження показали високу ефективність системи, що підтверджено актами впровадження та позитивними відгуками користувачів. Наукові тези, представлені на конференціях, також підтвердили перспективність та новизну розробки.

Подальша робота може включати розширення функціональності системи шляхом інтеграції з іншими інформаційними системами, впровадження аналітичних інструментів для покращення управління запасами та прогнозування потреб, а також дослідження та впровадження нових методів захисту даних для підвищення безпеки системи. Всі матеріали викладені коротко, як підсумки виконаної роботи, та відповідають визначеним завданням кваліфікаційної роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Корченко О. Г., Давиденко А. М., Висоцька О. О. Метод автентифікації користувачів інформаційних систем за їх рукописним почерком з багатокроковою корекцією первинних даних: захист інформації 21.1. 2019. 51 с.

2. What is HTTPS? URL: https://www.cloudflare.com/ru-ru/learning/ssl/what-is-https/ (дата звернення: 10.02.2024).

3. Indrasiri K., Kuruppu D. gRPC: up and running: building cloud native applications with Go and Java for Docker and Kubernetes. O'Reilly Media, 2020. 33 p.

4. HTTP/2: Why you should switch to the new protocol. URL: https://tuthost.ua/en/blog/http-2-why-you-should-switch-to-the-new-protocol/ (дата звернення: 10.02.2024).

5. Гурбанов, Т. А. Архітектурний стиль програмного забезпечення REST. Diss. Національний авіаційний університет. 2022. 41 с.

6. Василенко, М. Ю. Графічний інтерфейс інтелектуальної системи керування трансляцією мережевих адрес на основі протоколу NAT. MS thesis. Сумський державний університет, 2020.

7. Ковальчук О. А. Розробка освітньої платформи на основі технології. Net. 2023. 45 с.

8. Мартиненко О. П. Концепції десекуляризації, постсекуляризації та множинних сучасностей у подоланні кризи теорії модернізації. 2021. 167 с.

9. Andrienko, Y. A., and S. U. Donetskii. Development of an algorithm for constructing the optimal path based on the floor plan for the measuring robot. *International Journal of Open Information Technologies.* 2023. Vol. 11.8. P. 73-78.

10. Majumder S., Kar S., Pal T. Uncertain multi-objective Chinese postman problem. *Soft Computing.* Vol. 23. 2019. P.11557-11572.

11. Що таке POS-термінал? URL: http://www.finosvita.com.ua/ua/useful_info/Pensoneram/SHCHo-take-Pos-terminal-/ (дата звернення: 12.02.2024).

12. STACK AND HEAP. Структури даних у .NET. URL: https://itvdn.com/ua/blog/article/stack-and-heap (дата звернення: 12.02.2024).

13. Клієнт (інформатика). URL: https://termin.in.ua/server/ (дата звернення: 14.02.2024).

14. Сервер – що це таке, як працює і навіщо потрібен. Види, функції та приклади. URL: https://www.wikidata.uk-ua.nina.az/ (дата звернення: 14.02.2024).

15. Методи передачі даних (GET і POST). URL: https://uk.php.brj.cz/metodi-peredaci-danih-get-i-post (дата звернення: 14.02.2024).

16. ASP: Що таке і як працює абстрактний примітивний сервіс? URL: https://polaridad.es/uk/asp (дата звернення: 14.02.2024).

17. Які є конвенції в REST API та для чого їх дотримуватись. URL: https://dou.ua/forums/topic/34550/ (дата звернення: 15.02.2024).

18. Характеристика асинхронних та синхронних сокетів. Інтерфейс класу Socket для даного типу сокетів. URL: https://studfile.net/preview/9351317/page:20/ (дата звернення: 16.02.2024).

19. Основні протоколи Мережі. Навіщо вони використовуються? URL: https://hyperhost.ua/info/uk/osnovni-protokoli-merezhi-navishcho-voni-vikoristovuyutsya (дата звернення: 16.02.2024).

20. DELETE. URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/DELETE (дата звернення: 17.02.2024).

21. Що таке CRUD простими словами: функції, переваги та приклади. URL: https://highload.today/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/ (дата звернення: 20.03.2024).

22. PostgreSQL Technical Insights (Part 1): Data Organization. URL: https://cloudberrydb.org/blog/postgresql-technical-insights-1 (дата звернення: 20.03.2024).

23. What Is a Data Warehouse? Warehousing Data, Data Mining Explained. URL: https://www.investopedia.com/terms/d/data-warehousing.asp (дата звернення: 22.03.2024).

24. Create the Data Access Layer. URL: https://learn.microsoft.com/en-us/aspnet/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/create_the_data_access_layer (дата звернення: 23.03.2024).

25. Chapter 20 Data Structure and Storage. URL: https://www.richardtwatson.com/open/Reader/_book/data-structure-and-storage.html (дата звернення: 27.03.2024).

26. What is PostgreSQL? URL: https://www.postgresql.org/about/ (дата звернення: 27.03.2024).

27. ADO.NET entity framework: raising the level of abstraction in data programming. URL: https://www.researchgate.net/publication/221213932_ADONET_entity_framework_raising_the_level_of_abstraction_in_data_programming (дата звернення: 28.03.2024).

28. ASP.NET Core Web API: Your Path to Building Powerful APIs. URL: https://medium.com/@fatih.akpiyal1/asp-net-core-web-api-your-path-to-building-powerful-apis-a5338ef3b81d (дата звернення: 28.03.2024).

29. Why I completely switched to VS Code. URL: https://sternschleuder.de/category/c/ (дата звернення: 28.03.2024).

30. Database Migration. URL: https://www.yiiframework.com/doc/guide/2.0/en/db-migrations (дата звернення: 28.03.2024).

31. Implement the infrastructure persistence layer with Entity Framework Core. URL: https://learn.microsoft.com/en-

us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implementation-entity-framework-core (дата звернення: 30.03.2024).

32. Enterprise Level Application Architecture with Web APIs using Entity Framework, Generic Repository Pattern and Unit of Work. URL: https://www.slideshare.net/slideshow/restful-day-1/49316498#5 (дата звернення: 30.03.2024).

33. Usage of ASP.NET Core HTTP GET, POST, PUT, DELETE Methods. https://medium.com/@fatih.akpiyal1/usage-of-asp-net-core-http-get-post-put-delete-methods-b9e0e08f726e (дата звернення: 30.03.2024).

34. Pattern: API Gateway / Backends for Frontends. URL: https://chatgpt.com/c/85d03fc7-d073-4d71-9a07-a3b121f42095 (дата звернення: 30.03.2024).

35. Generic Repository Pattern with ASP.NET MVC and EF. URL: https://www.slideshare.net/slideshow/mvc-generic-repo/49071163 (дата звернення: 30.03.2024).

36. Mediator. URL: https://refactoring.guru/design-patterns/mediator (дата звернення: 30.03.2024).

37. CQRS pattern. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs (дата звернення: 30.03.2024).

38. The DTO Pattern (Data Transfer Object). URL: https://www.baeldung.com/ (дата звернення: 05.04.2024).

39. Modernize after upgrading to .NET from .NET Framework. URL: https://learn.microsoft.com/en-us/dotnet/core/porting/modernize (дата звернення: 05.04.2024).

40. Fujitsu Enterprise Postgres 15 for x86 Application Development Guide. URL: https://www.postgresql.fastware.com/hubfs/_Global/Manuals/FEP-v15SP2forx86-ApplicationDevelopmentGuide.pdf (дата звернення: 09.04.2024).

41. Value Conversions. URL: https://learn.microsoft.com/en-us/ef/core/modeling/value-conversions?tabs=data-annotations (дата звернення: 21.04.2024).

42. Error while loading asp.net core MVC list view. URL: https://learn.microsoft.com/en-us/answers/questions/1048330/error-while-loading-asp-net-core-mvc-list-view (дата звернення: 21.04.2024).

43. Creating and Configuring a Model. URL: https://learn.microsoft.com/en-us/ef/core/modeling/ (дата звернення: 21.04.2024).

44. How do programmers manage the migration of data from the previous version of database to its new version? URL: https://www.quora.com/How-do-programmers-manage-the-migration-of-data-from-the-previous-version-of-database-to-its-new-version (дата звернення: 22.04.2024).

45. Microsoft.EntityFrameworkCore.Tools. URL: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Tools (дата звернення: 23.04.2024).

46. Inheritance. URL: https://learn.microsoft.com/en-us/ef/core/modeling/inheritance (дата звернення: 25.04.2024).

47. Avoid Proliferating DbContext or IQueryable in .NET Apps. URL: https://ardalis.com/avoid-dbcontext-iqueryable-proliferation/ (дата звернення: 25.04.2024).

48. WEB API Interview Questions. URL: https://ru.scribd.com/document/548589328/ASP-Net-WEB-API-Interview-questions (дата звернення: 05.05.2024).

49. Controller action return types in ASP.NET Core web API. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/action-return-types?view=aspnetcore-8.0 (дата звернення: 05.05.2024).

50. Implement the microservice application layer using the Web API. URL: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-

ddd-cqrs-patterns/microservice-application-layer-implementation-web-api (дата звернення: 05.05.2024).

51. Command & Query: Domain - Request handler mediator. URL: https://medium.com/@bogdan.darjan/command-and-query-domain-request-handler-mediator-c918bc40cb1f (дата звернення: 10.05.2024).

52. An Object-Oriented Database Model Approach For The Logical Design Of A Customer Order Entry System. URL: https://www.slideshare.net/slideshow/an-objectoriented-database-model-approach-for-the-logical-design-of-a-customer-order-entry-system/259669217 (дата звернення: 18.05.2024).

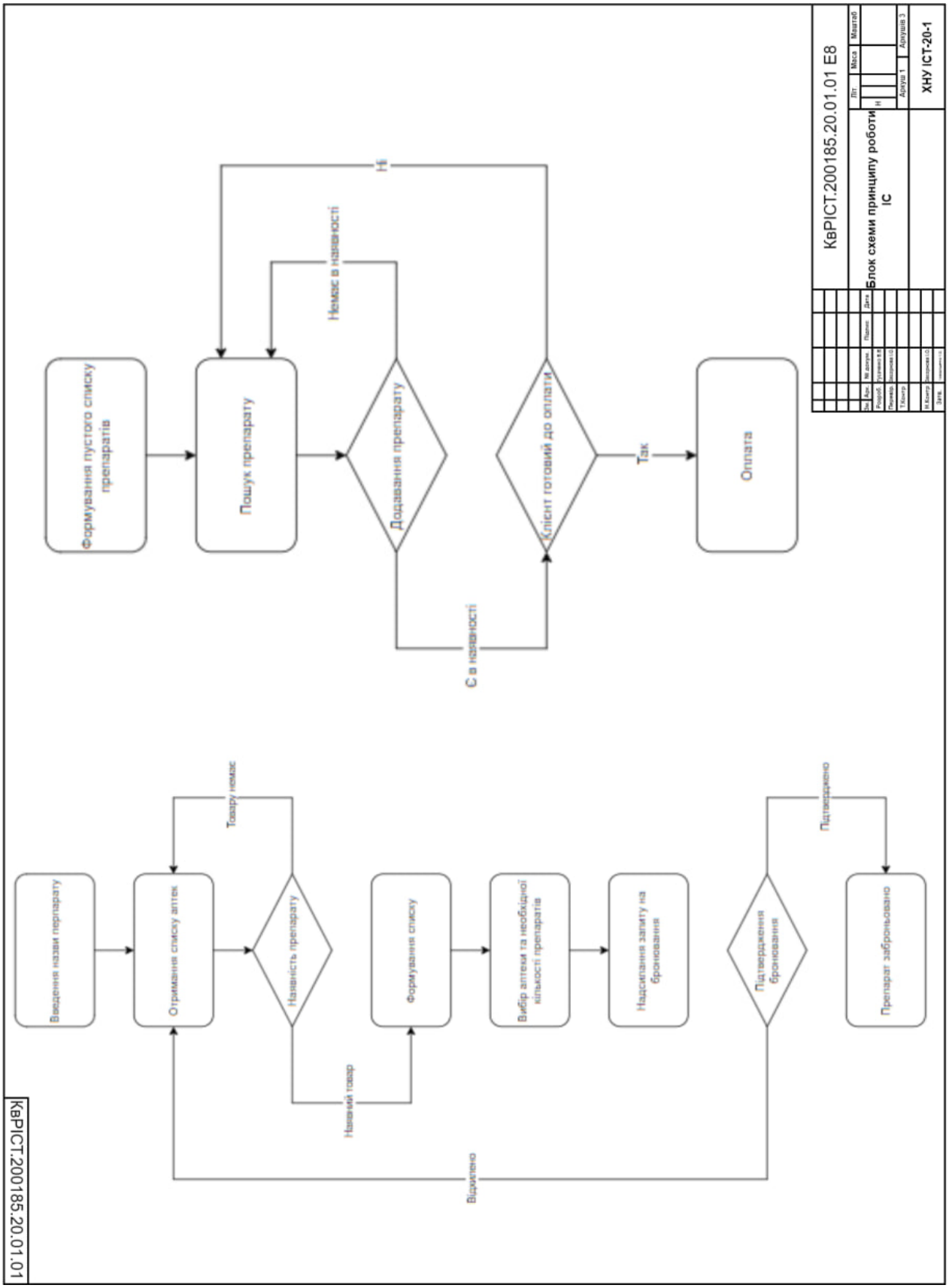
53. Docker: Accelerated Container Application Development. URL: https://www.docker.com/ (дата звернення: 18.05.2024).

54. Change Docker Desktop settings on Windows. URL: https://docs.docker.com/desktop/settings/windows/ (дата звернення: 20.05.2024).

Додаток А

(обов’язковий)

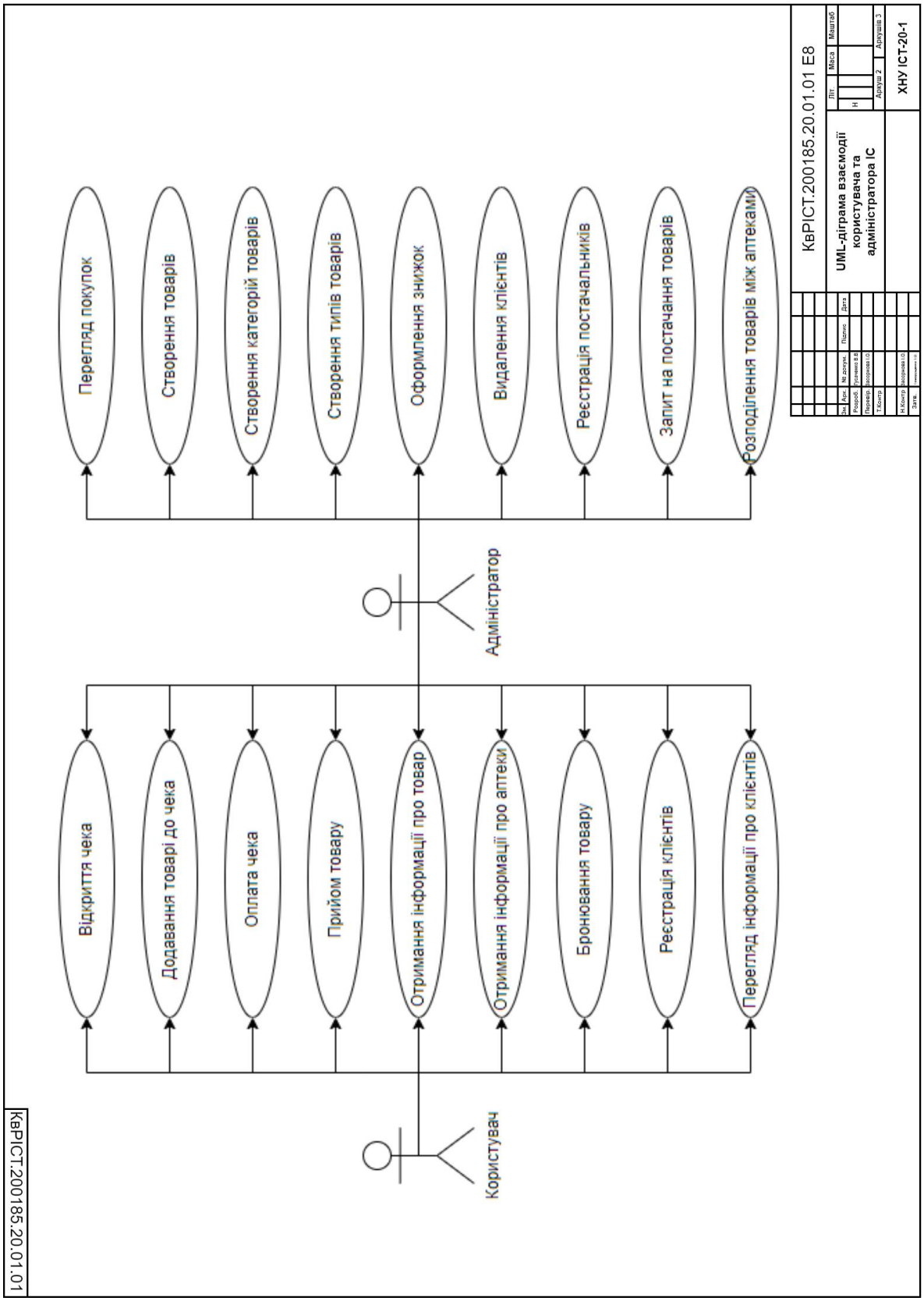
Копія креслення «Блок-схема принципу роботи ІС»



Додаток Б

(обов’язковий)

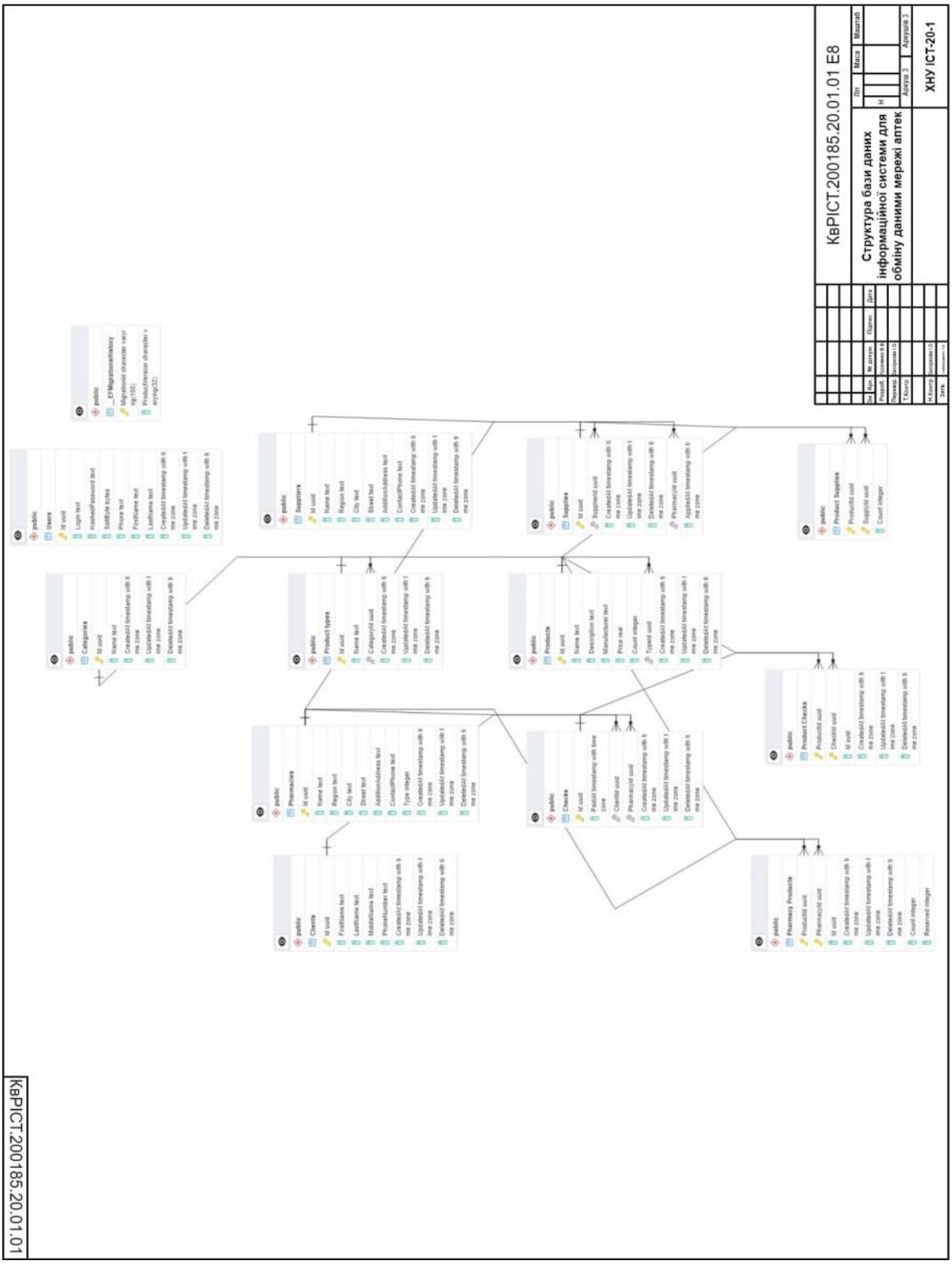
Копія креслення «UML діаграма взаємодії користувача ІС»



Додаток В

(обов’язковий)

Копія креслення «Структура бази даних інформаційної системи обміну даними мережі аптек»



UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016383159

Дата перевірки:
23.06.2024 09:28:11 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
23.06.2024 09:28:53 EEST

ID користувача:
100005591

Назва документа: **Гусаченко_Інформаційна система для обміну даними мережі аптек**

Кількість сторінок: **67** Кількість слів: **11283** Кількість символів: **91554** Розмір файлу: **853.84 KB** ID файлу: **1016193417**

4.43%
Схожість

Найбільша схожість: **1.71%** з джерелом з Бібліотеки (ID файлу: **1016193418**)

4.43% Джерела з Інтернету	297	Сторінка 69
1.71% Джерела з Бібліотеки	154	Сторінка 70

0.45% Цитат

Цитати	3	Сторінка 71
Посилання	1	Сторінка 71

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

Sun Jun 23 08:48:02 EEST 2024, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

ID: 132226 Назва: БКР Інформаційна система для обміну даними мережі аптек Додано в БД: 2024-06-23 Автора: В.В. Гусаченко Керівники: І.О. Засорнова Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	74391	619	2280 (3%)	32 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Гусаченко Вікторія Володимирівна

Тема: Інформаційна система для обміну даними мережі аптек

Спеціальність: 126 «Інформаційні системи та технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 57

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є забезпечення ефективного і безпечного обміну даними між аптечними мережами.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи здійснено теоретичний аналіз створення та функціонування інформаційної системи обміну даними в мережі аптек. Виділено ключові компоненти, що впливають на ефективність та надійність таких систем, а також можливі проблеми та виклики при їх впровадженні. Основними аспектами є архітектура систем, технології передачі даних, безпека інформації та інтеграція програмних продуктів. Особливу увагу приділено надійності та безперебійності функціонування системи, що критично для підтримання високого рівня обслуговування клієнтів.

В другому розділі кваліфікаційної роботи була реалізована серверна частина для обробки запитів та управління даними, а також створено клієнтський інтерфейс для зручного доступу користувачів до функцій системи.

В заключному третьому розділі було проведено тестування системи для виявлення та усунення помилок, а також оцінено її ефективність у реальних умовах.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: здійснений неповний аналіз інформаційної системи для обміну даними мережі аптек.

Завідувачу кафедри КІС
д-р.техн.наук, проф. Говорущенко Т. О.

Гусаченко Вікторія Володимирівнв
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи ІСТ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30.05.24

дата

підпис

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре 4.5 (B)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи)_____

Федосенко Леонід Петрович, зав. кафедри ІПЗ УДУ

"24" червня 2024 р. _____(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Інформаційна система для обміну даними мережі аптек

Автор: Гусаченко Вікторія Володимирівна

Спеціальність: 126 – Інформаційні системи та технології

Освітня програма: освітньо-професійна

Науковий керівник: Засорнова Ірина Олександрівна, к.т.н, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не становлять плагіату, оскільки:

1) Запозичення розміщені у розділах аналізу існуючих аналогів та прототипів, що не стосуються безпосередньо авторського дослідження та результатів роботи.
2) Усі запозичення є фрагментарними або мають належним чином оформлені посилання.
3) Окремі виявлені збіги є загальновживаними фразами або виразами, підтвердженням чого є збіг з певними джерелами для одного фрагмента речення.
4) Всі зафіксовані системою ознаки модифікації тексту стосуються комбінування латинських символів з україномовними скороченнями індексів у формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 4.43% і адресується до 222 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи — І. О. Засорнова

Гарант ОП — Є. Г. Гнатчук

Завідувач кафедри КІІС — Т. О. Говорущенко