Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра інженерії програмного забезпечення
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня
Бакалавр

(назва освітнього ступеня)

на тему: Розробка інформаційної системи лікарських препаратів та аналіз методів тестування

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Кривий Ю.Я. (прізвище та ініціали)

Керівник (підпис) Стефанишин І.М. (прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)
Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра інженерії програмного забезпечення
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Кривому Юрію Ярославовичу
(прізвище, ім'я, по батькові)

на тему: Розробка інформаційної системи лікарських препаратів та аналіз методів тестування

Керівник роботи Стефанишин І.М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи _____
3. Вихідні дані до роботи Реалізована інформаційна система лікарських препаратів та проведений аналіз методів тестування

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина
Аналіз предметної області та теоретичних основ розподілу ліків в світі та Україні
Визначення принципів роботи систем оплат
Реалізація інформаційної система та її тестування
Визначення основних аспектів охорони праці та безпеки життєдіяльності
Висновки роботи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,			
основи охорони праці	Лазарюк В.В. доц. каф. МТ		
Нормоконтроль	Стоянов Ю.М. к.т.н., ст. викл.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Аналіз завдання		
3	Виконання розділу 1		
4	Виконання розділу 2		
5	Виконання розділу 3		
6	Оформлення пояснювальної записки		
7	Оформлення графічного та презентаційного матеріалу		
8	Попередній захист		
9	Захист		

Студент _____ Кривий Ю.Я
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стефанишин І.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконала Кривий Юрій Ярославович, студент групи СП-41 Тернопільського національного технічного університету імені Івана Пулюя , присвячена розробці та тестуванню системи розподілу лікарських препаратів. Робота має обсяг 83 сторінок, включає 2 таблиці, 15 рисунків, 5 додатків, та бібліографію з 21 джерела.

Метою роботи є створення ефективної системи для оптимізації розподілу лікарських препаратів між медичними закладами. Система використовує передові методики інформаційних технологій та обробки даних для автоматизації процесів закупівлі, контролю залишків та логістики розподілу.

У роботі застосовані методики модульного та інтеграційного тестування, що забезпечують високий рівень надійності та функціональності системи. Розроблено користувацький інтерфейс, який дозволяє ефективно управляти всіма процесами в системі та забезпечує зручність використання.

Результати тестування підтверджують ефективність системи та її потенціал для подальшого розвитку та впровадження у медичні установи. Робота вносить важливий вклад у розвиток методів автоматизованого управління медичними ресурсами та оптимізації процесів розподілу.

Ключові слова: система розподілу лікарських препаратів, автоматизація, модульне тестування, інтеграційне тестування, медична логістика, інформаційні технології.

ABSTRACT

Bachelor's qualification work , carried out by KryvyiYuriy, a student of group SP-41 at the Ternopil National Technical University named after Ivan Puluj is dedicated to the development and testing of a pharmaceutical distribution system. The work consists of 83 pages, includes 15 figures, 5 appendices, and a bibliography with 21 sources.

The aim of the project is to create an effective system for optimizing the distribution of pharmaceuticals among medical institutions. The system employs advanced information technology and data processing methodologies to automate procurement, inventory control, and distribution logistics processes.

Modular and integration testing techniques applied in the work ensure a high level of system reliability and functionality. A user interface has been developed that allows for efficient management of all processes within the system and ensures ease of use.

Testing results confirm the system's efficiency and its potential for further development and implementation in medical facilities. This work contributes significantly to the development of automated medical resource management methods and optimization of distribution processes.

Keywords: pharmaceutical distribution system, automation, modular testing, integration testing, medical logistics, information technology.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

DB – Database

UI – User Interface

UX – User Experience

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

SQL – Structured Query Language

NLP – Natural Language Processing

ML – Machine Learning

AI – Artificial Intelligence

SaaS – Software as a Service

PaaS – Platform as a Service

IaaS – Infrastructure as a Service

JSON – JavaScript Object Notation

XML – Extensible Markup Language

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

UI/UX – User Interface/User Experience

QA – Quality Assurance

CI/CD – Continuous Integration/Continuous Deployment

ВСТУП

Розробка та оптимізація систем розподілу лікарських препаратів відіграє ключову роль у забезпеченні ефективності, безпеки та доступності медичного обслуговування. Сучасні інформаційні системи, що використовуються в ланцюгах поставок медикаментів, стикаються з рядом складнощів, що вимагають ретельного аналізу та інноваційних підходів до оптимізації. Основна мета таких систем полягає не тільки у впорядкуванні поставок і зберіганні препаратів, а й у забезпеченні швидкого реагування на змінювані потреби медичних закладів та пацієнтів. Це включає точне планування запасів, управління термінами придатності препаратів, а також координацію між різними ланками логістичних ланцюгів.

Проблеми, які існують в галузі, є багатогранними і охоплюють технічні, економічні та регуляторні аспекти. Технічні виклики включають інтеграцію різних інформаційних систем, що часто використовуються в ізольованих середовищах без належної синхронізації даних. Економічні складнощі пов'язані з високими витратами на впровадження та підтримку передових технологій у менш розвинених регіонах, де обмежений доступ до капіталу робить неможливим швидке оновлення застарілих систем. Регуляторні бар'єри включають необхідність дотримання національних та міжнародних стандартів безпеки, що постійно змінюються, а також забезпечення прозорості у веденні документації, що становить великий виклик для розподільчих мереж.

Однією з основних проблем є нестача стандартизації у процесах і технологіях, що ускладнює обмін даними та координацію між різними ланками ланцюга поставок. Це може призводити до перерв у доступності ліків, що особливо критично в періоди зростання попиту або глобальних здоров'я криз, як пандемія COVID-19. Активізація зусиль на шляху до стандартизації, розробка єдиних протоколів безпеки та обміну даними може сприяти підвищенню надійності і стабільності систем розподілу.

Мета даного дослідження полягає в аналізі, розробці та оптимізації систем розподілу лікарських препаратів з використанням сучасних інформаційних технологій для підвищення ефективності, надійності та безпеки медичного забезпечення. Особлива увага приділяється впровадженню ітераційних та модульних підходів до тестування цих систем, що дозволить знизити ризики пов’язані з помилками в програмному забезпеченні та забезпечити високу адаптивність системи до змінних умов експлуатації.

Перше завдання полягає в дослідженні існуючих систем розподілу лікарських препаратів. Аналіз структур, механізмів управління та технологічних рішень, які вже впроваджені в медичній індустрії, дозволить визначити основні недоліки та потенційні можливості для покращення. Друге завдання — розробка архітектури нової системи. Створення детальної моделі системи має враховувати сучасні вимоги до ефективності, безпеки та оперативності обробки даних. Третє завдання полягає в аналізі різних методик тестування, з порівнянням ітераційного, модульного та інших методів для визначення їхньої придатності у контексті розробки медичних інформаційних систем. Четверте завдання включає практичне застосування методів тестування. Імплементація вибраних методик на реальних проєктах дозволить виявити та усунути потенційні проблеми у функціонуванні системи до її офіційного запуску.

Завдяки виконанню цих завдань планується не тільки підвищити ефективність та безпеку системи розподілу лікарських препаратів, а й забезпечити гнучкість і масштабованість системи для відповіді на зростаючі потреби медичної індустрії.

Важливим є розуміння, що інновації та технологічні вдосконалення повинні йти рука об руку із забезпеченням високого рівня безпеки і дотримання етичних стандартів. Це вимагає балансу між впровадженням передових технологій і забезпеченням, що кожен крок у ланцюзі поставок виконується з дотриманням правил і норм, що гарантують безпеку та доступність лікарських препаратів для всіх, хто їх потребує.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз предметної області для систем розподілу лікарських препаратів охоплює широкий спектр аспектів, від технологічних до регуляторних. Важливо зрозуміти поточний стан індустрії, щоб ідентифікувати можливості для покращення та розвитку. Це включає детальне дослідження існуючих систем, використовуваних механізмів контролю за якістю, логістичних процедур, а також аналіз потреб кінцевих користувачів, що можуть включати лікарні, аптеки та інші медичні установи.

З підвищенням вимог до безпеки та ефективності, сучасні системи розподілу мають адаптуватися до швидко змінюваних умов ринку. Технологічні інновації, як-от впровадження автоматизації та штучного інтелекту, відіграють ключову роль у цьому процесі. Вони можуть допомогти оптимізувати логістику, покращити точність інвентаризації та забезпечити більш точне прогнозування попиту.

Крім технічних аспектів, аналіз предметної області також включає вивчення регуляторних вимог і стандартів. У різних країнах існують різні нормативні рамки, що впливають на управління ланцюгами поставок лікарських засобів. Розуміння цих вимог є критично важливим для розробки систем, що не тільки ефективні, але й повністю відповідають законодавчим стандартам.

Таким чином, глибокий аналіз предметної області допомагає створювати більш ефективні та безпечні системи розподілу лікарських препаратів, що сприяє підвищенню якості медичного обслуговування. Це забезпечує вирішення актуальних проблем індустрії та відкриває шлях для інновацій та покращення процесів у майбутньому.

Окрім вищезгаданих аспектів, аналіз предметної області включає вивчення етичних питань та практик корпоративної соціальної відповідальності, які все більше стають актуальними в галузі розподілу лікарських препаратів. Особливу увагу приділяється прозорості операцій та етичному використанню даних пацієнтів, що вимагає не тільки технологічних рішень для захисту інформації, але

й чітких корпоративних політик. Розуміння цих вимог та забезпечення їх виконання важливі для підтримки довіри між всіма зацікавленими сторонами і сприяння сталому розвитку в галузі.

1.1 Огляд систем розподілу лікарських препаратів

Огляд систем розподілу лікарських препаратів можна розглянути, аналізуючи приклади з України та інших країн, де кожна система має свої особливості та виклики.

В Україні система розподілу лікарських препаратів традиційно залежить від мережі дистриб'юторів, які поставляють препарати від виробників до аптек та медичних установ. Останнім часом уряд вжив заходів для модернізації цієї системи з метою підвищення її ефективності та прозорості. Однією з ключових ініціатив є впровадження електронної системи обліку та контролю за обігом медикаментів, що дозволяє відслідковувати кожну партію ліків від виробника до кінцевого споживача. Це спрямовано на боротьбу з контрабандою та підробками, а також на забезпечення доступності необхідних ліків для населення [1, 2].

На відміну від України, в країнах ЄС, наприклад, в Німеччині та Франції, система розподілу лікарських препаратів є більш централізованою та строго регульованою. У цих країнах діють жорсткі законодавчі вимоги до кожного етапу ланцюга поставок, а також високі стандарти якості та безпеки. Це забезпечує високий рівень захисту споживачів, але також збільшує витрати на логістику та управління запасами [3].

В США, система розподілу також відрізняється високим рівнем регуляції, але, на відміну від європейських моделей, більша частина лікарських препаратів поставляється через великі фармацевтичні мережі, які використовують передові технології для управління запасами та логістики. Особлива увага в США приділяється інноваційним методам доставки, таким як автоматизовані склади і

роботизовані системи сортування, що дозволяє знижувати витрати та підвищувати швидкість обслуговування пацієнтів.

Кожна з цих систем має свої переваги та недоліки, в залежності від регуляторного середовища, доступності технологій та культурних особливостей країни. Огляд цих систем дозволяє не тільки зрозуміти глобальні тенденції, але й визначити потенційні напрями для вдосконалення національних систем розподілу лікарських засобів [4,5].

1.1.1 Визначення основних типів систем розподілу

Визначення основних типів систем розподілу лікарських препаратів можна класифікувати на кілька категорій, кожна з яких відповідає різним логістичним потребам та регуляторним вимогам.

1) Прямий розподіл (Direct Distribution) - цей тип системи передбачає безпосередню доставку медикаментів від виробника до кінцевих споживачів, зазвичай великих медичних закладів або роздрібних аптек. Прямий розподіл зменшує кількість посередників, що може призвести до зниження вартості ліків та підвищення контролю за якістю продукції.

2) Дистриб'юторська система (Wholesale Distribution) - у цьому випадку, виробники постачають продукцію дистриб'юторам, які потім займаються її розподілом між аптеками, лікарнями та іншими медичними установами. Це найпоширеніший тип розподілу, який дозволяє ефективно керувати великими об'ємами товарів та забезпечувати їх доступність у різних регіонах.

3) Консигнаційний розподіл (Consignment Distribution) - у такій моделі лікарські препарати розміщуються на складі замовника, але залишаються власністю постачальника до моменту їх продажу або використання. Це знижує ризики для постачальників і дозволяє замовникам не вкладати кошти в запаси, які можуть не знайти попиту.

4) Централізована система (Centralized Distribution) - в цій системі усі лікарські засоби зберігаються в одному або кількох центральних складах, звідки вони розподіляються по потребі. Централізація сприяє кращому контролю запасів, оптимізації логістичних витрат та поліпшенню управління інвентарем.

5) Гібридна система (Hybrid Distribution) - комбінує елементи вищезгаданих систем для досягнення оптимальної ефективності. Наприклад, великі ланцюги аптек можуть використовувати прямі поставки для популярних препаратів, в той час як рідкісні ліки поставляються через дистриб'юторів.

Кожен з цих типів має свої переваги та недоліки і може бути більш або менш підходящим залежно від конкретних умов та потреб ринку. Вибір системи розподілу в значній мірі залежить від специфіки лікарських засобів, масштабів діяльності, географічного розташування та регуляторного середовища [6].

1.1.2 Аналіз сучасних технологій та методів управління ланцюгами поставок

Аналіз сучасних технологій та методів управління ланцюгами поставок є ключовим для підвищення ефективності та оптимізації витрат у різних галузях, включаючи фармацевтику. На прикладі України можна виділити кілька передових технологій та методів, які активно впроваджуються або вже використовуються в ланцюгах поставок лікарських препаратів.

1) ERP системи (Enterprise Resource Planning) - Багато українських компаній впроваджують інтегровані ERP-системи, що дозволяють автоматизувати та оптимізувати багато бізнес-процесів, включаючи управління запасами, логістику, фінанси та звітність. Системи, як SAP або Oracle, забезпечують кращий контроль над даними та ефективне управління ресурсами.

2) Системи управління складом (WMS, Warehouse Management Systems) - Сучасні WMS використовуються для підвищення точності запасів, оптимізації складських операцій та покращення обслуговування клієнтів. Ці системи

допомагають автоматизувати приймання, зберігання, підготовку та відвантаження товарів, знижуючи тим самим витрати та збільшуючи продуктивність.

3) Інформаційні технології для відстеження та простежуваності - Відстеження та простежуваність важливі для забезпечення якості та безпеки фармацевтичних продуктів. В Україні впровадження систем відстеження на основі RFID (Radio Frequency Identification) та баркодів дозволяє моніторити рух ліків від виробника до кінцевого споживача, забезпечуючи захист від контрафактної продукції.

4) Інтернет речей (IoT) - Інтернет речей стає все більш популярним у логістичних операціях, дозволяючи збирати великі обсяги даних з різних джерел в реальному часі. У фармацевтиці це може включати моніторинг температурних режимів під час транспортування чутливих препаратів або автоматизацію управління запасами.

5) Штучний інтелект (AI) та машинне навчання - Використання AI дозволяє оптимізувати ланцюги поставок за допомогою прогнозування попиту, автоматизації планування виробництва та підвищення точності управління запасами. Штучний інтелект може аналізувати великі обсяги даних для виявлення тенденцій та нерегулярностей, що значно покращує рішення про запаси та логістику.

Впровадження цих технологій в Україні ще перебуває на різних стадіях розвитку, але швидке прийняття інноваційних рішень відкриває значні можливості для підвищення ефективності та конкурентоспроможності на фармацевтичному ринку/

В Україні основною системою розподілу лікарських препаратів традиційно є дистриб'юторська система. Ця система передбачає, що виробники препаратів співпрацюють із дистриб'юторами, які виконують роль посередників між фармацевтичними компаніями та кінцевими пунктами продажу, такими як аптеки та медичні установи.

Дистриб'ютори відіграють ключову роль у ланцюзі поставок, забезпечуючи логістику, зберігання та розподіл лікарських засобів по всій країні. Вони

відповідають за те, щоб ліки були доставлені вчасно і в належних умовах, що є важливим для підтримки якості та ефективності медичного обслуговування.

Дистриб'юторська система дозволяє ефективно управляти великими обсягами лікарських засобів, забезпечує широкий охоплення та доступність препаратів у різних регіонах, включаючи віддалені та сільські території. Також ця модель сприяє конкуренції серед дистриб'юторів, що може призводити до покращення сервісу і зниження цін для кінцевих споживачів.

Однак, система також має певні недоліки, зокрема, залежність від дистриб'юторів може призводити до збільшення вартості лікарських засобів через нарахування додаткових маржі дистриб'юторами. Це також може ускладнювати контроль за якістю препаратів через можливі порушення у ланцюзі поставок.

З метою оптимізації та підвищення ефективності розподілу лікарських засобів, в Україні вживаються заходи для впровадження сучасних технологій та покращення регуляторної бази, що дозволяє підвищити прозорість і контроль у цій сфері.

1.2 Теоретичні основи ітераційного та модульного тестування

Теоретичні основи ітераційного та модульного тестування є важливими для розуміння та застосування цих методів в процесах розробки та підтримки програмних продуктів, зокрема у системах розподілу лікарських препаратів.

Ітераційне тестування базується на принципі поступового додавання функціональних можливостей та їх тестування протягом кількох ітерацій розробки продукту. Кожна ітерація включає планування, розробку, тестування та аналіз результатів, що дозволяє систематично виявляти та виправляти помилки на ранніх стадіях процесу розробки. Цей підхід підвищує адаптивність процесу до змін у вимогах та допомагає краще управляти ризиками [7].

Модульне тестування, відоме також як компонентне тестування, полягає в перевірці окремих частин програми, називаних модулями або компонентами, на відповідність їх функціональних специфікацій. Модульне тестування дозволяє ізолювати частини програми та переконатися у їх коректній роботі перед тим, як вони будуть інтегровані з іншими частинами системи. Це сприяє виявленню помилок на ранніх етапах, зниженню взаємозалежності між модулями і полегшенню процесу інтеграції [8].

Обидва ці методи вимагають детального планування та ретельного виконання тестових сценаріїв. Важливим аспектом ітераційного та модульного тестування є використання автоматизованих інструментів, що дозволяє значно підвищити швидкість та ефективність тестування. Впровадження цих методів у розробку інформаційних систем, зокрема тих, що використовуються для розподілу лікарських препаратів, може значно збільшити надійність та ефективність кінцевих продуктів.

1.2.1 Опис та порівняння методологій тестування

Опис і порівняння різних методологій тестування програмного забезпечення дають змогу краще зрозуміти, які підходи найбільш ефективні для конкретних типів проєктів та вимог. Дві основні категорії тестування включають статичні та динамічні методи, кожен з яких має свої унікальні характеристики та призначення.

Статичне тестування відбувається без виконання програмного коду. Основні інструменти статичного аналізу включають ревізії коду, аналіз вимог і дизайну, а також інструменти статичного аналізу коду, які можуть автоматично виявляти потенційні проблеми та вразливості [9].

Динамічне тестування, навпаки, здійснюється шляхом запуску програми. Цей метод виявляє помилки, які проявляються тільки під час виконання програми. Динамічне тестування може бути поділено на кілька типів, зокрема:

— Функціональне тестування – перевіряє відповідність програми її визначеним специфікаціям і вимогам.

— Тестування на відмову – спрямоване на визначення умов, при яких програма може зазнавати збоїв.

— Регресивне тестування – переконується, що нові зміни не порушують існуючу функціональність.

— Тестування продуктивності – оцінює, наскільки швидко система обробляє запити під навантаженням.

Важливість вибору методології тестування залежить від специфіки проєкту. Наприклад, для критичних систем, де помилка може призвести до значних фінансових втрат або навіть загрози життю, рекомендується використання як статичного, так і динамічного тестування для забезпечення високого рівня якості.

Загалом, комбінування різних методів тестування забезпечує більш повне і всебічне виявлення помилок, що сприяє створенню більш надійного та ефективного програмного забезпечення. Вибір методології повинен базуватися на цілях тестування, доступних ресурсах, термінах проєкту та критичності системи.

1.2.2 Застосування ітераційного та модульного тестування в різних типах проєктів

Ітераційне та модульне тестування відіграють важливу роль у розробці програмного забезпечення, кожен з цих методів підходить для застосування в певних типах проєктів залежно від їх специфікацій та вимог.

Ітераційне тестування ефективне у великих, складних проєктах, де вимоги можуть змінюватися протягом розробки. Цей підхід дозволяє командам розробників адаптуватися до змін, вносячи корективи в проєкт на ранніх стадіях, що сприяє зниженню кінцевих витрат на виправлення помилок.

Приклади застосування:

— Розробка програмного забезпечення: Проєкти, які передбачають розробку великих програмних систем, де етапи планування, розробки, тестування та перегляду повторюються в кілька ітерацій.

— Стартапи: Ітераційний підхід ідеально підходить для стартапів, які швидко адаптуються до ринкових умов та користувацьких вимог.

Модульне тестування застосовується для перевірки окремих компонентів або модулів програми до їх інтеграції в загальну систему. Це дозволяє забезпечити, що кожен модуль працює правильно незалежно, що спрощує виявлення джерел помилок після інтеграції.

Приклади застосування:

— Розробка банківських та фінансових систем: Модульне тестування є критично важливим, оскільки вимоги до безпеки і точності в таких системах дуже високі.

— Інтеграція сторонніх сервісів: Коли проєкт включає інтеграцію сторонніх компонентів, модульне тестування допомагає забезпечити, що кожен інтегрований компонент функціонує як очікується.

Обидва методи можуть бути застосовані разом у рамках гнучких методологій розробки, таких як Agile, де ітераційне тестування використовується для перевірки функціоналу на рівні всієї системи після кожної ітерації розробки, а модульне тестування — для перевірки окремих компонентів перед їх інтеграцією.

Застосування цих тестувальних методологій забезпечує більшу гнучкість і контроль у процесі розробки, дозволяючи командам швидше реагувати на помилки та зміни, а також підвищувати якість та надійність кінцевого продукту.

Ітераційне та модульне тестування застосовуються у багатьох реальних проєктах, де кожен із цих підходів використовується для забезпечення якості та ефективності розробки програмного забезпечення. Нижче наведено два приклади великих проєктів, де ці методи тестування були використані для досягнення високих стандартів якості та надійності.

Приклад 1 (Spotify): Spotify використовує ітераційний підхід у своїй розробці, постійно впроваджуючи нові функції та оновлення для своїх додатків.

Розробники проводять тестування нових функціональностей у коротких циклах, щоб швидко адаптувати продукт до відгуків користувачів.

Приклад 2 (Microsoft Windows): Microsoft застосовує модульне тестування для перевірки окремих компонентів своєї операційної системи Windows перед їх інтеграцією. Такий підхід забезпечує високу стабільність та безпеку системи, дозволяючи ефективно ідентифікувати та виправляти помилки в окремих модулях.

Приклад 3 (Google Chrome): Розробка браузера Google Chrome включає використання ітераційного тестування для швидкого випуску нових версій та модульне тестування для перевірки окремих компонентів, таких як рендеринг движок або система управління вкладками. Це забезпечує високу надійність браузера та швидке впровадження нововведень.

Обидва ці проєкти демонструють, як ітераційне та модульне тестування можуть бути ефективно застосовані в різних областях для покращення якості продукту та оптимізації процесу розробки. Ітераційне тестування ідеально підходить для проєктів, де необхідно швидко адаптуватися до змінюваних вимог, тоді як модульне тестування забезпечує детальне перевірення окремих компонентів перед їхнім введенням в експлуатацію, знижуючи ризики, пов’язані з інтеграцією.

1.3 Ітераційне тестування: методики і застосування

Ітераційне тестування є одним із центральних елементів гнучких методологій розробки програмного забезпечення, таких як Agile. Цей підхід дозволяє командам розробників і тестувальників ефективно працювати разом у коротких циклах, постійно вдосконалюючи продукт і виправляючи помилки на ранніх стадіях.

Методики ітераційного тестування включають кілька ключових компонентів, що допомагають забезпечити його ефективність:

1) Планування ітерацій: Кожна ітерація починається з планування, де команда визначає, які функції будуть розроблені та тестовані протягом наступного циклу. Планування також включає приоритизацію завдань та розподіл ресурсів.

2) Розробка та неперервне тестування: Розробники і тестувальники працюють паралельно, що дозволяє швидко виявляти та усувати помилки. Тестування виконується не тільки після завершення розробки, але й протягом усього процесу кодування.

3) Щоденні зустрічі (Daily Stand-ups): Короткі щоденні наради допомагають тримати команду в курсі поточного стану проєкту, виявляти проблеми та обговорювати рішення.

4) Демонстрації та зворотний зв'язок: В кінці кожної ітерації проводяться демонстрації розробленого функціоналу для замовників або користувачів, щоб отримати їхній зворотний зв'язок і внести необхідні корективи в наступні ітерації.

5) Ретроспективи: Команда аналізує, що було зроблено добре, що можна покращити та як оптимізувати процеси в майбутньому. Це сприяє неперервному вдосконаленню команди та проєкту.

Ітераційне тестування широко застосовується у різноманітних проєктах, особливо де велике значення має швидка адаптація до змін у вимогах і ринкових умовах:

— Програмне забезпечення як послуга (SaaS): Швидкі цикли розробки і регулярне оновлення продукту.

— Стартапи: Можливість швидко адаптуватися до змін на ринку і користувацьких вподобань.

— Великі корпоративні системи: Необхідність управління великими обсягами даних та складністю системи.

Ітераційне тестування дозволяє командам не тільки покращувати якість продукту, але й забезпечувати його відповідність до постійно змінюваних вимог користувачів та замовників, зменшуючи ризики та витрати на розробку.

1.3.1 Переваги і недоліки ітераційного тестування

Ітераційне тестування, як важлива частина гнучких методологій розробки, пропонує значні переваги, але також має деякі недоліки, які слід враховувати під час планування та впровадження цього підходу в проєкти (див. рис. 1.1).

Переваги ітераційного тестування:

1) Гнучкість до змін: Ітераційне тестування дозволяє командам швидко адаптуватися до змін у вимогах та користувацьких очікуваннях. Завдяки коротким циклам розробки, можливо оперативно вносити корективи без значних витрат.

2) Раннє виявлення помилок: Тестування, що відбувається паралельно з розробкою в рамках кожної ітерації, допомагає виявляти та усувати помилки на ранніх стадіях, що значно знижує вартість їх виправлення.

3) Покращення задоволеності клієнта: Часті демонстрації продукту та залучення замовника до процесу перегляду та внесення змін забезпечують вищий рівень задоволеності клієнтів, оскільки вони бачать постійний прогрес та мають можливість впливати на кінцевий продукт.

4) Неперервне вдосконалення: Ретроспективи в кінці кожної ітерації сприяють неперервному аналізу та вдосконаленню процесів, що підвищує ефективність команди.

Недоліки ітераційного тестування:

1) Управління обсягом робіт: Ітераційне тестування може призвести до проблем з управлінням обсягу робіт, оскільки постійні зміни вимог можуть спричинити "зміщення обсягу" (scope creep), де постійно додаються нові функції.

2) Залежність від дисципліни команди: Ефективність ітераційного тестування сильно залежить від самодисципліни та здатності команди дотримуватись планів і графіків. Без чіткої організації роботи можливе зниження продуктивності.

3) Потреба в постійній участі замовника: Ітераційне тестування вимагає активної участі замовника протягом усього процесу розробки. Не всі клієнти готові або можуть дозволити собі витрачати час на таку тісну співпрацю.

4) Витрати на управління: Ітераційний процес може вимагати додаткових витрат на управління проєктом і координацію діяльності, особливо у великих командах.

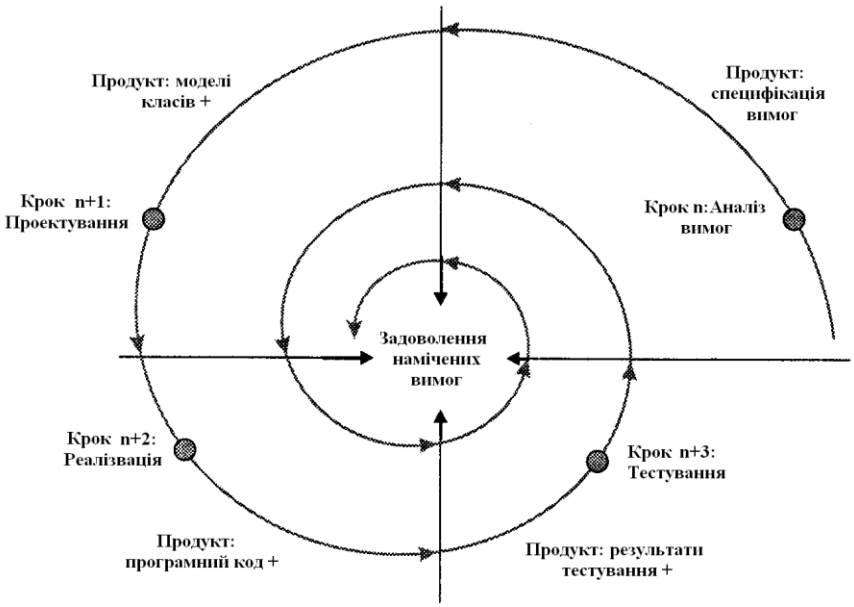

Рисунок 1.1 – Ітераційна модель розробки

Ітераційне тестування пропонує значні переваги для проєктів, де важлива швидка адаптація до змін і висока взаємодія з клієнтом, однак вимагає чіткого управління процесом та високої залученості всіх учасників проєкту для досягнення успіху.

1.3.2 Сценарії використання в інформаційних системах

Ітераційне тестування є важливим інструментом у гнучкій розробці програмного забезпечення, особливо у великих та складних проєктах. Цей метод

дозволяє командам швидко адаптуватися до змін у вимогах, поступово вдосконалюючи продукт і виправляючи помилки на ранніх стадіях. Такий підхід демонструється на прикладах різних інформаційних систем.

Наприклад, у розробці CRM-систем, ітераційне тестування дозволяє команді концентруватися на одній ключовій функціональності на кожній ітерації, такій як інтеграція з електронною поштою або автоматизація продажів. Тестування інтеграції з зовнішніми API та обробка взаємодії з даними забезпечує, що кожен новий елемент функціональності діє ефективно та безпечно в реальних умовах.

Ще один приклад використання ітераційного тестування - оновлення системи електронної документації у державній установі. Це включає впровадження нових модулів безпеки, як двофакторна аутентифікація, шифрування даних та інші функції, що підвищують безпеку. Ітераційний підхід дозволяє поступово вводити ці зміни, забезпечуючи надійність системи та мінімізуючи ризики для існуючих функцій.

Також, ітераційне тестування має велике значення у впровадженні нових банківських платформ, зокрема для обробки онлайн-транзакцій. Тут ітерації зосереджені на тестуванні відповідності системи до фінансових стандартів безпеки та її здатності взаємодіяти з іншими фінансовими системами. В результаті, банк може гарантувати, що нова платформа не тільки ефективна, але й відповідає усім необхідним регуляторним вимогам.

Ітераційне тестування надає значні переваги для розробки та управління інформаційними системами, дозволяючи забезпечити високу якість та відповідність продукту до змінюваних вимог користувачів та ринкових умов.

1.4 Ітераційне тестування: методики і застосування

Модульне тестування, також відоме як юніт-тестування, є фундаментальним компонентом розробки програмного забезпечення, який дозволяє перевіряти

найменші тестовані частини програми, зазвичай окремі функції або модулі, на предмет правильності їх виконання. Цей підхід грає ключову роль у забезпеченні високої якості кінцевого продукту.

Принципи модульного тестування

1) Ізоляція: Кожен модуль тестується окремо від інших частин програми. Це забезпечує, що помилки виявлені тільки в одному тестованому компоненті, без впливу зовнішніх факторів.

2) Автоматизація: Модульні тести зазвичай пишуться та виконуються автоматично за допомогою спеціалізованого програмного забезпечення, що дозволяє їх легко інтегрувати в щоденні процеси розробки та неперервної інтеграції.

3) Документація: Хорошо написані модульні тести можуть слугувати формою документації, оскільки вони показують, як призначено використовувати реальні функції системи.

4) Повторне використання: Одні і ті ж тести можуть використовуватися впродовж усього життєвого циклу програми для перевірки, що зміни в коді не порушили існуючу функціональність.

Переваги модульного тестування

1) Виявлення помилок на ранніх стадіях: Модульні тести допомагають розробникам виявляти та виправляти помилки на ранніх стадіях розробки, що знижує витрати на виправлення помилок у майбутньому.

2) Покращення дизайну коду: Розробка модульних тестів вимагає від розробників ретельно продумувати інтерфейси та взаємодії між компонентами програми, що часто призводить до більш чистого та ефективнішого коду.

3) Забезпечення стабільності: Автоматизовані модульні тести забезпечують мережу безпеки, що дозволяє розробникам вносити зміни в код, не боячись ненароком порушити існуючу функціональність.

4) Спрощення рефакторингу: Модульні тести дозволяють розробникам рефакторити код, знаючи, що їхні зміни можна швидко перевірити, що забезпечує збереження функціональності програми.

5) Зменшення залежностей: Ізоляція модульних тестів від зовнішніх залежностей забезпечує, що тести не впливають один на одного, що призводить до більш стабільних та надійних тестових циклів.

Модульне тестування є незамінним інструментом в сучасній розробці програмного забезпечення, особливо у великих та складних проєктах, де потрібно забезпечити високу якість та надійність продукту.

1.4.1 Детальний опис процесу модульного тестування

Модульне тестування, також відоме як юніт-тестування, є ключовою складовою процесу розробки програмного забезпечення. Цей метод фокусується на перевірці окремих модулів програми, щоб забезпечити їхню правильну роботу перед інтеграцією з іншими частинами системи. Нижче наведено детальний опис процесу модульного тестування:

1) Визначення модулів для тестування

Першим кроком у модульному тестуванні є визначення, які компоненти або модулі програми будуть тестуватися. Модуль може бути функцією, методом класу, або навіть цілим класом, залежно від архітектури програми та рівня складності.

2) Написання тестових випадків

Для кожного модуля пишуться тестові випадки, що визначають очікувану поведінку при різних умовах. Тестові випадки мають бути якомога більш охоплюючими, включаючи стандартні сценарії, межові умови, а також потенційно некоректні вхідні дані.

3) Ізоляція модуля

Для забезпечення точності тестів, модуль потрібно ізолювати від інших частин системи. Це означає, що зовнішні залежності, такі як бази даних, файлові системи чи інші модулі, заміщаються моками або стабами. Така ізоляція допомагає

переконатись, що тести не впливають один на одного та фокусуються лише на функціональності тестуваного модуля.

4) Виконання тестів

Модульні тести зазвичай автоматизовані та виконуються за допомогою фреймворків, таких як JUnit для Java, pytest для Python, або jest для JavaScript. Тести можуть бути виконані вручну розробником або автоматично як частина процесу неперервної інтеграції.

5) Аналіз результатів

Після виконання тестів аналізуються результати для виявлення помилок у коді. Успішне проходження тестів означає, що модуль відповідає встановленим вимогам, тоді як невдача може вказувати на потребу у додатковій розробці чи виправленні.

6) Підтримка та рефакторинг

Модульні тести слід підтримувати в актуальному стані разом із змінами, які вносяться в код. Вони також служать підтримкою при рефакторингу, оскільки допомагають переконатися, що зміни не вплинули на існуючу функціональність.

Модульне тестування не тільки допомагає забезпечити надійність та безпеку програмного забезпечення, але й сприяє більшій продуктивності розробників, зменшуючи час, необхідний для виявлення та виправлення помилок у майбутньому. Це засіб для досягнення високої якості продукту та задоволення вимог користувачів.

1.4.2 Вплив на ефективність і надійність систем

Модульне тестування має значний вплив на ефективність та надійність інформаційних систем. Цей метод тестування дозволяє розробникам систематично перевіряти кожен компонент програми, що сприяє створенню більш стабільних та надійних програмних продуктів.

Вплив на ефективність

1) Зменшення часу на виправлення помилок: Модульні тести допомагають виявляти помилки на ранніх стадіях розробки, коли виправити їх набагато простіше і дешевше. Це значно зменшує загальний час, витрачений на дебагінг і рефакторинг в пізнішій фазі проєкту.

2) Підвищення продуктивності розробників: Автоматизація модульних тестів дозволяє розробникам швидше впроваджувати нові функції та впевнено робити зміни в коді, знаючи, що існуюча функціональність захищена тестами.

3) Оптимізація процесів розробки: Модульне тестування сприяє вибудовуванню більш ефективних процесів розробки, стимулюючи використання кращих практик програмування і проєктування, що призводить до більш чистого, організованого і модульного коду.

Вплив на надійність

1) Покращення якості коду: Модульні тести вимагають, щоб код був написаний з урахуванням тестування, що часто веде до кращої архітектури і вищої якості коду. Це, в свою чергу, знижує ймовірність помилок і підвищує надійність системи.

2) Забезпечення стабільності: Завдяки ізоляції модулів і тестуванню їх у відриві від інших частин програми, модульне тестування допомагає забезпечити стабільність всієї системи. Зміни в одному модулі не впливають на інші, що знижує ризик виникнення системних помилок.

3) Гарантування безперервної роботи: Надійна система — це система, яка здатна працювати безперервно і стабільно. Модульні тести перевіряють критичні компоненти на міцність перед тим, як вони будуть інтегровані в більші системи, забезпечуючи тим самим безперервність бізнес-процесів.

4) Зниження ризиків у складних системах: Великі та складні системи часто мають високий рівень взаємозалежностей між модулями. Модульне тестування дозволяє контролювати ці залежності та забезпечувати, що кожен модуль функціонує як очікується, знижуючи загальний ризик збоїв системи.

Модульне тестування, таким чином, є незамінним інструментом для підвищення ефективності та надійності розробки програмного забезпечення, допомагаючи розробникам створювати кращі продукти з меншими витратами на підтримку та виправлення помилок.

1.5 Інші методики тестування

Крім модульного тестування, існує багато інших методик тестування програмного забезпечення, кожна з яких має свої унікальні переваги та застосування. Ось деякі з них:

1) Інтеграційне тестування

Ця методика зосереджена на тестуванні взаємодії між модулями або відділеннями системи. Інтеграційне тестування перевіряє, що різні частини програми працюють разом як очікується. Це особливо важливо для великих систем, де окремі модулі розробляються різними командами.

2) Системне тестування

Системне тестування оцінює поведінку цілої системи або програми в контексті її кінцевого призначення або вимог. Це включає перевірку відповідності функціональних і нефункціональних вимог, таких як проєктні специфікації, надійність, ефективність, та зручність використання.

3) Приймальне тестування

Приймальне тестування здійснюється з метою перевірки, чи готовий продукт до впровадження та чи відповідає він очікуванням клієнтів. Це часто включає тестування з реальними користувачами або зацікавленими сторонами та служить останнім етапом перевірки перед розгортанням продукту.

4) Навантажувальне тестування

Цей тип тестування спрямований на перевірку поведінки системи під значним навантаженням, наприклад, великою кількістю користувачів, операцій або

запитів. Навантажувальне тестування допомагає визначити, на якому рівні навантаження система може почати діяти нестабільно або завершити свою роботу.

5) Регресійне тестування

Регресійне тестування використовується для забезпечення того, що зміни в коді, такі як виправлення помилок або оновлення функціоналу, не порушили існуючу функціональність. Це важливо для виявлення непередбачених ефектів від змін, здійснених в програмі.

6) Безпекове тестування

Безпекове тестування спрямоване на виявлення вразливостей у програмі, які можуть бути використані зловмисниками. Це включає перевірку на різні типи атак, такі як SQL ін'єкції, переповнення буфера, неправильне управління сесіями та інші потенційні загрози.

7) Юзабіліті тестування

Цей вид тестування фокусується на оцінці зручності користування програмою. Юзабіліті тестування вимірює, наскільки легко кінцеві користувачі можуть виконувати задачі за допомогою програми, виявляючи проблеми, пов'язані з інтерфейсом користувача.

Кожен із цих методів має своє специфічне застосування та може бути використаний окремо або в комбінації з іншими для забезпечення високої якості та надійності програмного продукту.

1.5.1 Аналіз статичного та динамічного тестування

Для системи розподілу лікарських препаратів, що включає комплексні логістичні та інформаційні процеси, важливо використовувати ефективні методи тестування. Статичне та динамічне тестування є двома ключовими підходами, кожен з яких має свої переваги та недоліки, залежно від контексту застосування.

Статичне тестування — це аналіз коду та документації без виконання коду. Це включає перевірку синтаксису, перевірку стилів програмування, код-рев’ю, та аналіз виконуваності коду. Для системи розподілу лікарських препаратів, статичне тестування може допомогти виявити проблеми з безпекою, такі як вразливості до ін'єкцій коду, проблеми з доступом до даних, та інші питання, що можуть порушити конфіденційність пацієнтської інформації.

Переваги:

— Виявлення помилок на ранніх стадіях розробки.

— Зменшення витрат на виправлення помилок.

— Покращення якості коду та документації.

Недоліки:

— Не може ідентифікувати проблеми, які з'являються тільки під час виконання програми.

— Може не враховувати взаємодію з зовнішніми системами.

Динамічне тестування включає виконання програмного коду з метою виявлення помилок. Воно може бути реалізовано через модульне, інтеграційне, системне чи приймальне тестування. У контексті системи розподілу лікарських препаратів, динамічне тестування може допомогти переконатися, що всі компоненти системи взаємодіють правильно, що процеси відповідають регуляторним вимогам, і що система ефективно управляє логістичними потоками.

Переваги:

— Виявлення помилок, які не можуть бути знайдені через статичне тестування.

— Тестування функціональності системи в умовах, максимально наближених до реальних.

— Перевірка інтеграції системи з іншими аплікаціями та сервісами.

Недоліки:

— Зазвичай більш часомістке та дороге у порівнянні зі статичним тестуванням.

— Може вимагати складних середовищ для тестування.

Ефективність та надійність системи розподілу лікарських препаратів значною мірою залежать від якості проведеного тестування. Статичне тестування ідеально підходить для раннього виявлення синтаксичних та логічних помилок, в той час як динамічне тестування незамінне для перевірки взаємодії та інтеграції різних системних компонентів. Використання обох методів тестування забезпечує глибоке розуміння та високу якість розробки системи.

1.5.2 Аналіз статичного та динамічного тестування

Тестування великих даних у медичних інформаційних системах є критично важливим процесом, оскільки воно забезпечує надійність, безпеку та ефективність обробки та аналізу величезних обсягів медичних даних. Ці дані включають пацієнтську інформацію, результати клінічних досліджень, записи про лікування та іншу критично важливу інформацію. Ось як застосовується тестування великих даних у медичних інформаційних системах:

Медичні інформаційні системи часто інтегрують дані з різноманітних джерел, таких як електронні медичні записи, лабораторні системи, системи управління пацієнтами тощо. Тестування великих даних допомагає переконатися, що інтеграція виконується без помилок, дані правильно об'єднуються, та система може ефективно обробляти дані з різних джерел.

Якість даних є життєво важливою для медичних інформаційних систем, оскільки помилки в даних можуть призвести до невірних медичних висновків та лікування. Тестування включає валідацію точності, повноти та консистенції даних. Перевіряється, чи дані відповідають визначеним стандартам та чи правильно вони відображаються для кінцевих користувачів.

Медичні інформаційні системи повинні бути здатні обробляти великі обсяги даних без збоїв у роботі, особливо під час піків навантаження, які можуть відбуватися в критичних ситуаціях. Стрес-тестування визначає межі системи та її

здатність витримувати велике навантаження, забезпечуючи, що система залишається стабільною та швидкою під час критичних операцій.

Оскільки медичні дані є вкрай чутливими, тестування безпеки є обов'язковим. Це включає перевірку на вразливості до зламу, захисту від несанкціонованого доступу, та забезпечення конфіденційності даних. Тестування даних у цьому контексті також має забезпечити, що всі механізми шифрування, аутентифікації та контролю доступу функціонують належним чином.

Медичні інформаційні системи мають відповідати численним регуляторним вимогам, таким як HIPAA в США або GDPR в Європі, які стосуються зберігання та обробки медичних даних. Тестування великих даних має забезпечити, що системи не лише відповідають цим стандартам на момент запуску, але й здатні підтримувати цю відповідність у довгостроковій перспективі, включаючи здатність правильно реагувати на запити щодо даних пацієнтів, такі як виправлення, видалення або перенесення даних.

У сфері медичних даних, де дослідження та аналітика відіграють велику роль, важливо забезпечити, що аналітичні засоби, які використовуються для витягування знань з великих обсягів даних, працюють коректно. Тестування великих даних має включати перевірку аналітичних звітів на точність і надійність, а також оптимізацію процесів обробки даних, щоб забезпечити швидке та ефективне вирішення запитів.

Автоматизація тестування великих даних у медичних системах дозволяє регулярно перевіряти великі обсяги даних без значних витрат часу та ресурсів, що важливо для забезпечення постійної якості і надійності в реальному часі. Автоматизовані інструменти можуть швидко ідентифікувати аномалії та помилки в даних, дозволяючи швидко вжити заходів.

Застосування тестування великих даних у медичних інформаційних системах є ключовим для забезпечення їх ефективності, надійності та безпеки. Це важливо не тільки для покращення медичного обслуговування, але й для захисту конфіденційної інформації пацієнтів.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА ЗАСТОСУВАННЯ МЕТОДИК ТЕСТУВАННЯ

Розробка інформаційної системи в медичній сфері вимагає не тільки глибоких знань у галузі охорони здоров'я та технологій, але й забезпечення високого рівня точності, безпеки даних та їх захисту. На етапі планування та розробки системи особлива увага приділяється вибору методик тестування, які зможуть ефективно виявити потенційні помилки та вразливості в системі. Вибір методів тестування значною мірою залежить від специфіки проєкту, обсягу даних, які оброблятиме система, та вимог до її функціональності.

В процесі розробки інформаційних систем особливу увагу приділяють модульному тестуванню, яке дозволяє перевіряти окремі компоненти системи до їх інтеграції. Це забезпечує виявлення та усунення помилок на ранніх етапах, перш ніж вони стануть частиною більшої системи. На наступному етапі використовується інтеграційне тестування, яке зосереджене на перевірці взаємодії між модулями. Цей вид тестування критично важливий для забезпечення того, що всі частини системи ефективно співпрацюють між собою, забезпечуючи надійне та точне функціонування відповідно до встановлених вимог.

2.1 Архітектура системи

Використання клієнт-серверної моделі та бази даних MySQL для архітектури медичної інформаційної системи відкриває широкі можливості для ефективного управління медичними даними. Ця архітектура забезпечує чітке розділення відповідальностей між клієнтами (наприклад, робочими станціями лікарів, мобільними додатками пацієнтів) і сервером, який управляє логікою обробки даних та їх збереженням.

Клієнт-серверна архітектура дозволяє централізовано управляти бізнес-логікою, безпекою та доступом до даних, що особливо важливо у медичних системах з високими вимогами до конфіденційності та безпеки інформації. Сервери не тільки забезпечують обробку запитів від багатьох клієнтів, але й виконують функції з валідації даних, виконання запитів до баз даних і повернення результатів. Такий підхід знижує навантаження на клієнтські системи та покращує загальну продуктивність системи.

MySQL, як система управління базами даних, відіграє ключову роль у зберіганні та обробці медичних даних. Її висока продуктивність, надійність та легкість використання роблять MySQL ідеальним вибором для медичних застосунків, де потрібна швидка обробка великих обсягів інформації. Крім того, MySQL підтримує розширені можливості безпеки, такі як шифрування даних та управління доступом, що є критично важливим для захисту чутливих медичних даних [9].

Інтеграція клієнт-серверної архітектури з MySQL також сприяє легкому масштабуванню медичних інформаційних систем. За міру зростання обсягу даних чи кількості користувачів системи, сервери можуть бути налаштовані на збільшення потужності обробки або зберігання, а нові клієнтські додатки легко інтегрувати без потреби зміни основної інфраструктури. Таке рішення забезпечує гнучкість та адаптивність системи в динамічному медичному середовищі [10, 11].

Враховуючи ці аспекти, клієнт-серверна модель разом з базою даних MySQL становить міцну основу для медичних інформаційних систем, забезпечуючи їх ефективність, надійність, та масштабованість.

2.1.1 Визначення вимог до системи

Визначення вимог до медичної інформаційної системи є критичним етапом, який впливає на всі аспекти проєктування, розробки та використання системи. Ось детальний перелік ключових вимог, які слід розглянути:

Функціональні вимоги:

— Управління пацієнтськими даними: Система повинна забезпечувати можливість реєстрації, збереження, оновлення та видалення інформації про пацієнтів, включаючи медичну історію, діагнози, лікування, алергії та контактні дані.

— Інтеграція з іншими системами: Система має інтегруватися з лабораторними системами, системами обліку ліків, страховими програмами та іншими медичними інформаційними системами для забезпечення повної синхронізації даних.

— Обробка запитів: Система має ефективно обробляти запити від користувачів, забезпечуючи швидкий доступ до необхідної інформації з мінімальними затримками.

— Звітність: Система має генерувати стандартні та налаштовувані звіти для медичних працівників і адміністрації.

Нефункціональні вимоги:

— Безпека: Система має відповідати найвищим стандартам безпеки для захисту конфіденційних медичних даних, включаючи шифрування даних, аутентифікацію користувачів та аудит доступу.

— Надійність: Система повинна бути стійкою до помилок та забезпечувати високу доступність і точність обробки даних, з мінімальним ризиком втрати інформації.

— Масштабованість: Система має бути гнучкою до масштабування, щоб вона могла адаптуватися до зростаючого числа користувачів, обсягів даних та інтеграції з новими системами.

— Швидкодія: Час відгуку системи на запити користувачів має бути мінімальним, щоб забезпечити ефективне використання системи в умовах високого навантаження.

Вимоги до користувацького інтерфейсу:

— Інтуїтивність: Інтерфейс повинен бути простим і зрозумілим для користувачів різного рівня компетентності.

— Доступність: Інтерфейс повинен бути доступним для користувачів з обмеженими можливостями, включаючи налаштування для людей із порушеннями зору або слуху.

— Адаптивність: Інтерфейс повинен коректно відображатися на різних пристроях, включаючи настільні комп'ютери, планшети та мобільні телефони.

Ці вимоги визначають основу для створення ефективної, надійної та безпечної медичної інформаційної системи, яка може значно покращити якість медичного обслуговування та операційну ефективність.

2.1.2 Опис структури і компонентів системи

Структура та компоненти медичної інформаційної системи, яка використовує клієнт-серверну архітектуру та базу даних MySQL, є фундаментальними для забезпечення її ефективності, безпеки та надійності.

Сервер бази даних (Database Server)

MySQL Database Server: Серцем системи є сервер бази даних MySQL, який зберігає всі медичні, адміністративні та оперативні дані. Це включає дані про інформацію про лікарів, деталі ліки, інформацію про рецепти та інвентаризаційні дані. База даних налаштована так, щоб забезпечити високу продуктивність, надійність зберігання даних і строгі заходи безпеки, включаючи шифрування та контроль доступу.

Клієнтські додатки (Client Applications)

Веб-додаток: Основний інтерфейс для користувачів, таких як медичний персонал і адміністратори, є веб-додаток, доступний через стандартний веб-браузер.

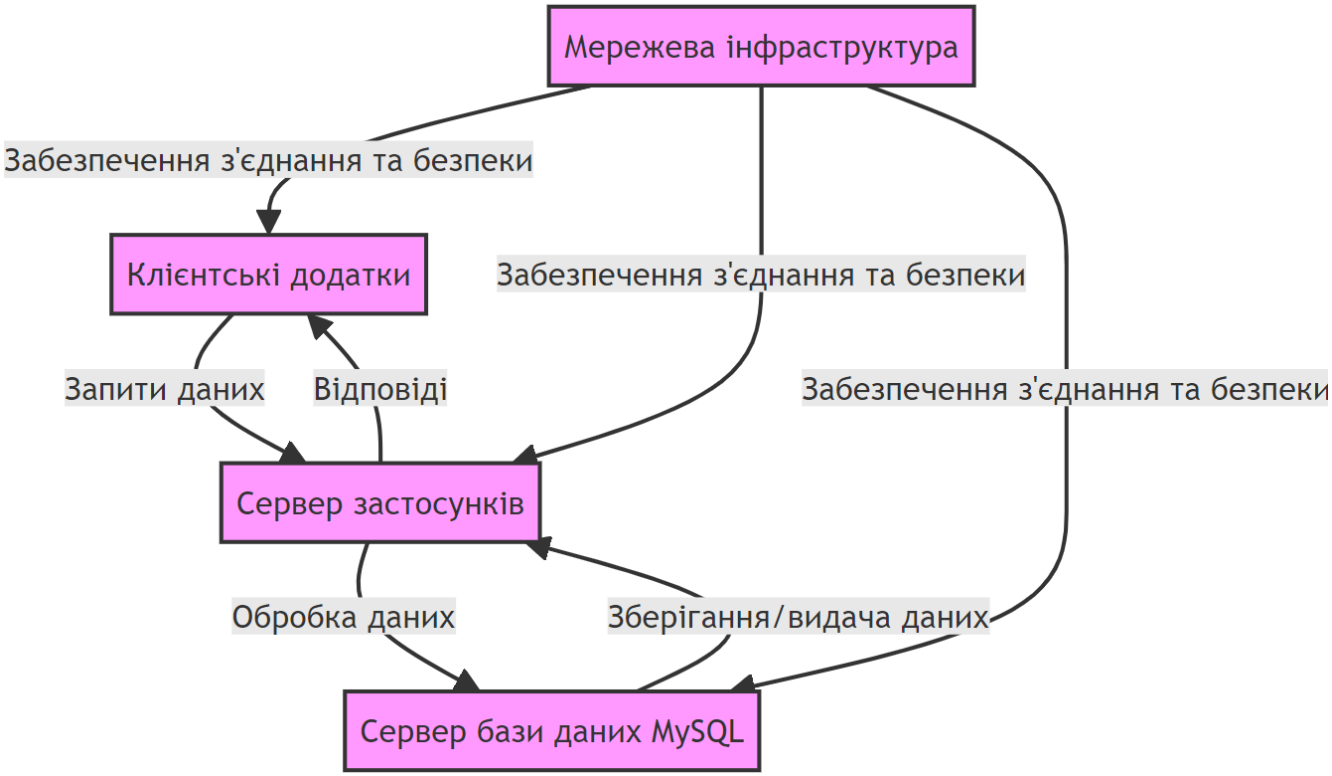

Рисунок 2.1 – Основні компоненти системи

Сервер застосунків (Application Server)

Логіка застосунку: Всі бізнес-логіки та обробки даних виконуються на сервері застосунків, який спілкується з сервером бази даних і клієнтськими додатками. Це включає модулі обробки запитів, управління сесіями, обробки транзакцій та забезпечення безпеки. Сервер застосунків також відповідає за виконання бізнес-правил, валідацію даних та забезпечення взаємодії різних модулів системи.

Мережева інфраструктура

З'єднання та безпека: Мережева інфраструктура забезпечує надійні та безпечні з'єднання між клієнтськими додатками та серверами. Вона включає застосування мережевих протоколів, шифрування передачі даних та встановлення

брандмауерів та інших систем безпеки для захисту від несанкціонованого доступу та інших кіберзагроз.

Ця структура та компоненти системи формують міцну основу для ефективної медичної інформаційної системи, забезпечуючи високу продуктивність, надійність та безпеку при обробці чутливих медичних даних.

2.1.3 Варіанти використання

Інтуїтивно зрозумілим є використання діаграм варіантів використання для зображення загальних функціональних можливостей системи (див. рис. 2.2).

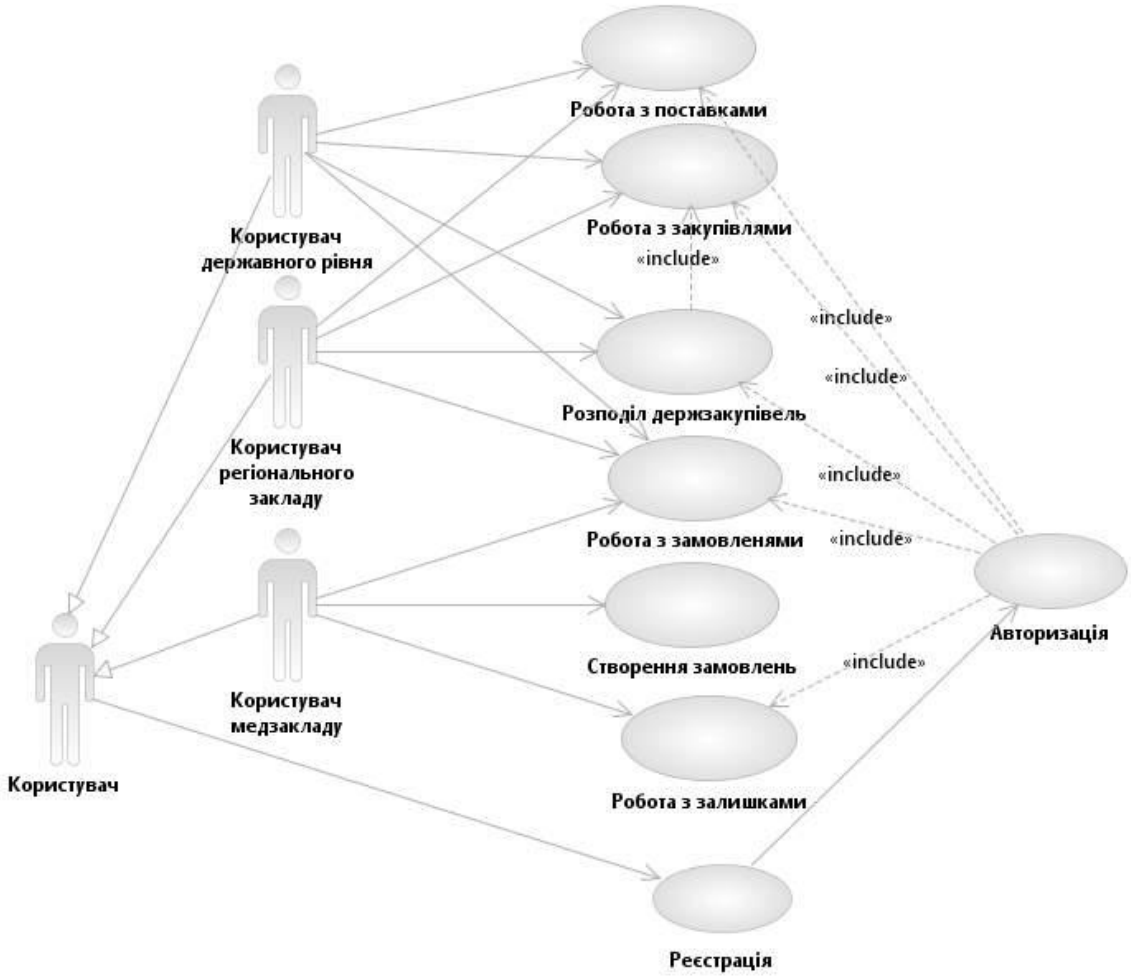

Рисунок 2.2 – Основні варіанти використання системи

На діаграмі представлено кілька основних типів користувачів системи: користувач державного рівня, користувач регіонального закладу та користувач медзакладу. Кожен із цих користувачів має доступ до різних функцій системи, таких як:

— Робота з постачальниками
— Робота з закупівлями
— Робота з замовленнями
— Створення замовлень
— Робота з заліками
— Авторизація
— Реєстрація

Ці функції взаємодіють між собою, як зазначено на діаграмі, де деякі функції включають ("include") інші, свідчачи про те, що виконання однієї функції автоматично вимагає виконання інших.

Застосування цієї діаграми в контексті медичної інформаційної системи

1) Організація доступу: Ця діаграма допоможе визначити, які ролі та дозволи мають бути налаштовані для кожного типу користувача, щоб забезпечити ефективне управління доступом до різних частин системи.

2) Модульне проєктування: Розробка може бути модульною, з чітко визначеними залежностями та інтерфейсами між модулями, відповідно до функцій, вказаних на діаграмі.

3) Інтеграція функцій: Понимання зв'язків між функціями допоможе у проєктуванні бази даних і логіки застосунку, забезпечуючи плавне взаємодію компонентів системи.

4) Тестування: Діаграма також може служити основою для розробки сценаріїв тестування, оскільки вона відображає основні шляхи взаємодії користувачів із системою.

Варто детально розглянути кожен з цих варіантів використання та проаналізувати їх основні функції, залежності між ними та режими доступу між

різними варіантами. Зведення відомостей про варіанти використання можна побачити на таблиці 2.1.

Таблиця 2.1 – Опис варіантів використання

Назва функції	Опис функції	Залежні функції	Ролі доступу
Робота з постачами	Управління інформацією про постачальників та наявність товарів на складі.	Робота з закупівлями	Користувач регіонального закладу
Робота з закупівлями	Замовлення товарів та ліків, управління договорами з постачальниками.	Створення замовлень	Користувач медзакладу
Робота з замовленнями	Обробка і відстеження замовлень, планування доставок.	Робота з постачами, Авторизація	Користувач медзакладу
Створення замовлень	Формування замовлень на основі потреб закладу.	Робота з закупівлями	Користувач медзакладу
Робота з заліками	Управління фінансовими заліками між закладами та постачальниками.	Робота з постачами	Користувач державного рівня
Авторизація	Перевірка та підтвердження ідентифікаційних даних користувачів для доступу до системи.	Робота з замовленнями	Всі ролі
Реєстрація	Внесення нових користувачів до системи, налаштування прав доступу.	Авторизація	Користувач державного рівня

Варто розглянути основні варіанти використання детальніше:

1) Варіанти використання в управлінні закупівлями:

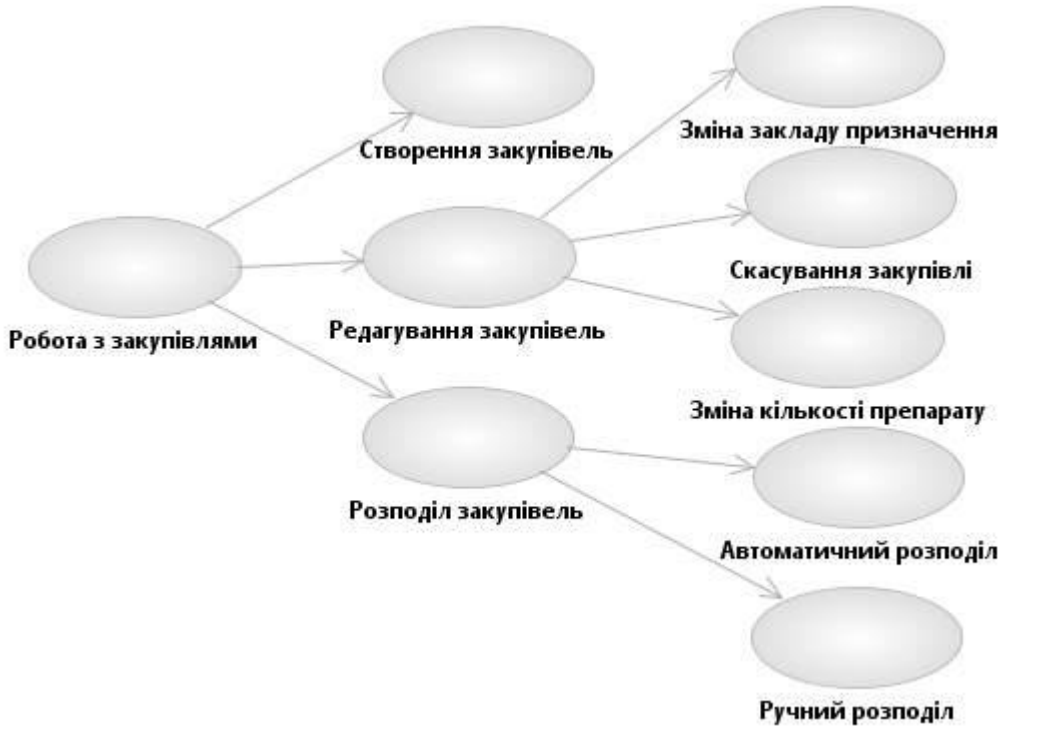

Рисунок 2.3 – Опис роботи з закупівлями

— Робота з закупівлями. Цей варіант використання є основним вузлом, який описує загальний процес взаємодії з закупівлями в системі.

— Створення закупівель. Це дія дозволяє користувачам ініціювати нові закупівлі. Включає процес внесення даних про необхідні медикаменти чи інші товари, які потрібно закупити.

— Редагування закупівель. Функція дає змогу модифікувати існуючі записи закупівель. Можна змінювати кількість товарів, ціни або інші параметри запланованих закупівель.

— Скасування закупівель. Дозволяє користувачам скасувати заплановані закупівлі, що може бути необхідно у випадку зміни планів або помилок у замовленні.

— Зміна закладу призначення. Цей сценарій дозволяє змінювати місце доставки або використання закупленого товару, що може бути корисно у випадку змін в організаційній структурі або потребах.

— Зміна кількості препарату. Варіант використання для корекції кількості медикаментів у вже зроблених закупівлях, наприклад, на основі оновленої інформації про потреби або доступність.

— Автоматичний розподіл. Система автоматично розподіляє закуплені товари між відділеннями або закладами на основі встановлених алгоритмів або правил.

— Ручний розподіл. Дає можливість користувачам вручну керувати розподілом товарів, що може бути потрібно для специфічних випадків або особливих вимог.

2) Розподіл держзакупівель(див. рис. 2.4)

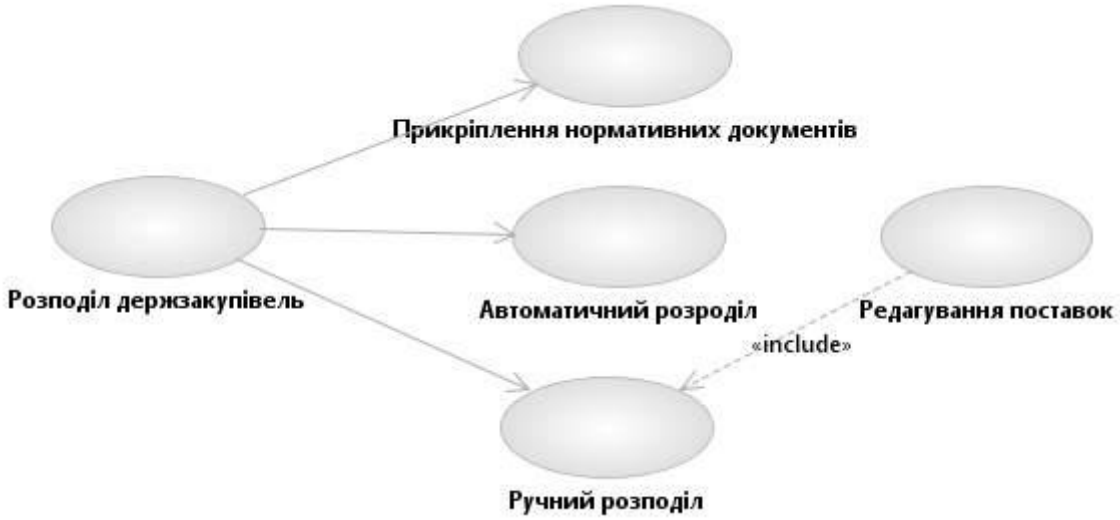

Рисунок 2.4 – Опис роботи з держзакупівлями

На наданій діаграмі варіантів використання показано різні процеси, пов'язані з розподілом державних закупівель у медичній інформаційній системі. Діаграма демонструє, як різні функції системи співпрацюють для забезпечення ефективного управління розподілом ресурсів. Ось детальний опис кожного компонента на діаграмі:

— Розподіл держзакупівель. Центральний вузол, який управляє процесом розподілу закуплених медичних товарів між різними регіональними та місцевими

медичними установами на основі попередньо встановлених критеріїв або замовлень.

— Прикріплення нормативних документів. Процес прикріплення важливих нормативних документів, які регулюють закупівлі та розподіл медичних товарів. Це може включати законодавчі акти, стандарти якості та інші регуляторні документи, необхідні для забезпечення правильного і законного розподілу ресурсів.

— Автоматичний розподіл. Автоматизація процесу розподілу з метою збільшення ефективності та зменшення людських помилок. Система використовує алгоритми для визначення оптимального розподілу товарів згідно з потребами та доступністю.

— Ручний розподіл. Забезпечує можливість ручного втручання в процес розподілу, дозволяючи користувачам вносити зміни або коригувати автоматичні розподіли відповідно до специфічних потреб або умов.

— Редагування поставок. Функція, що дозволяє модифікувати параметри вже запланованих поставок, такі як кількість товарів, терміни поставок або навіть види товарів, що поставляються. Ця функція включена ("include") в процес автоматичного та ручного розподілу, оскільки вона безпосередньо впливає на їх виконання.

Ця діаграма допомагає визначити як основні, так і допоміжні процеси управління ресурсами в системі медичних закупівель, підкреслюючи важливість гнучкості та контролю у взаємодії з такими ресурсами. Вона надає цінний огляд для розробників системи, адміністраторів, та кінцевих користувачів, що займаються логістикою та розподілом медичних товарів.

3) Робота з поставками:

На цій діаграмі представлено деталізований процес управління поставками в системі. Вона відображає взаємозв'язок між основними діями, які залучені в цей процес, включаючи створення, редагування, розподіл, та додавання інформації про поставки. Ось детальний опис кожного компонента діаграми (див. рис. 2.5):

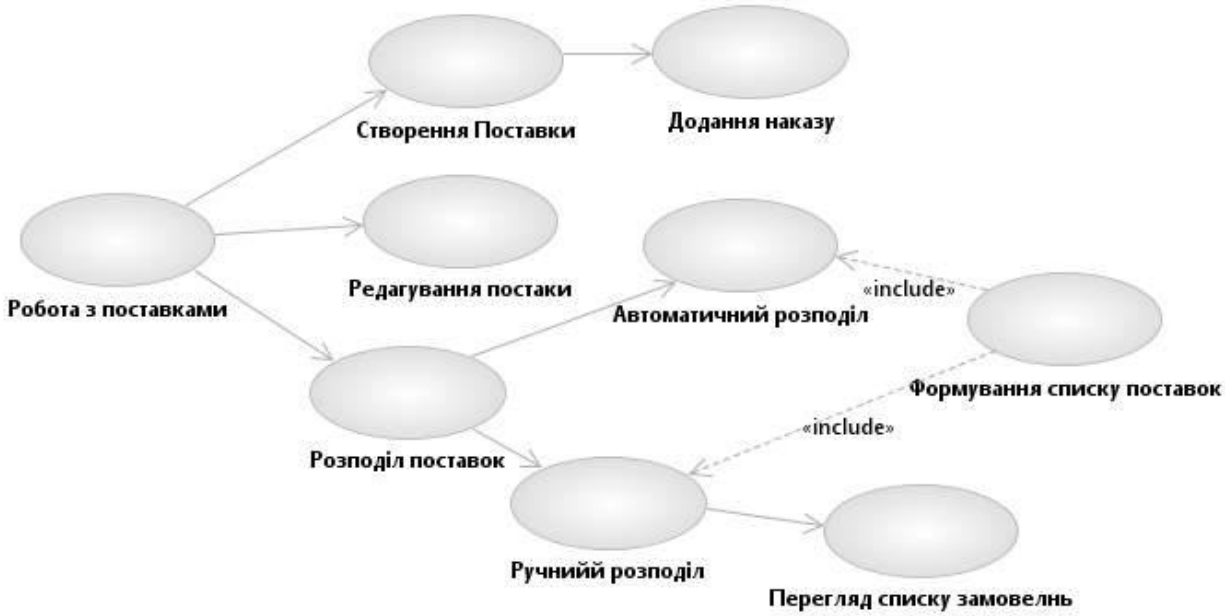

Рисунок 2.5 – Опис роботи з поставками

— Робота з поставками. Головний вузол, що включає загальне управління процесами, пов'язаними з поставками медичних товарів.

— Створення Поставки. Цей процес включає ініціацію нових поставок, внесення даних про нові медикаменти або медичні товари, які потребують закупівлі.

— Редагування поставки. Модифікація деталей існуючих поставок, таких як кількість товарів, терміни доставки, постачальники тощо.

— Розподіл поставок. Цей процес забезпечує ефективний розподіл поставлених товарів між різними медичними установами.

— Ручний розподіл. Дозволяє користувачам вручну керувати розподілом поставок, на основі конкретних потреб або змін у планах.

— Автоматичний розподіл. Автоматизований процес розподілу, який використовує алгоритми для розподілу поставок згідно з встановленими правилами або критеріями.

— Формування списку поставок. Збір інформації про всі планові поставки в єдиний список для легшого відстеження і управління.

— Перегляд списку замовлень. Перегляд та аналіз замовлень, що були зроблені медичними установами, для оптимізації майбутніх поставок.

— Додання наказу. Внесення нових наказів до системи, які можуть вплинути на планування і виконання поставок.

Ця діаграма є інструментом для забезпечення чіткого розуміння процесу управління поставками в медичній інформаційній системі, підкреслюючи як автоматизовані, так і ручні компоненти процесу для оптимального розподілу та використання ресурсів.

4) Додаткові варіанти використання

У доповнення до основних варіантів використання, представлених вище, додаток Б містить розширений опис додаткових сценаріїв взаємодії, які важливі для повноцінної роботи системи. Ці додаткові варіанти використання включають, але не обмежуються наступним:

— Процеси верифікації та затвердження замовлень. Детальний опис процедур перевірки та затвердження замовлень перед їх фіналізацією та виконанням поставок.

— Управління непередбаченими обставинами при доставці. Сценарії, які розглядають випадки затримок, пошкоджень товарів під час транспортування та інших непередбачених обставин, включаючи їх вирішення.

— Зворотний зв'язок від користувачів. Процеси збору та аналізу зворотного зв'язку від користувачів системи для покращення сервісу та внесення коректив у роботу системи.

— Деталізоване ведення журналів аудиту. Механізми ведення журналів дій користувачів для забезпечення відповідності нормам безпеки та можливості аудиту діяльності в системі.

— Керування пріоритетами замовлень. Функціональність для управління пріоритетами замовлень, що дозволяє віддавати перевагу невідкладним або критично важливим поставкам у разі необхідності.

Ці додаткові варіанти використання забезпечують гнучкість та адаптивність системи до змінних потреб закладів охорони здоров'я та їх оперативних вимог.

Вони є важливою частиною загальної стратегії управління медичними ресурсами, направленої на підвищення ефективності, зниження ризиків та покращення якості обслуговування пацієнтів.

Для детального ознайомлення з кожним з цих сценаріїв зверніться до додатку N, де ви знайдете вичерпну інформацію, процесні схеми та вказівки для впровадження та використання у повсякденній діяльності медичного закладу.

Цей текст можна включити в документацію для забезпечення користувачів системи всією необхідною інформацією про додаткові можливості та сценарії використання, що не включені безпосередньо у головне тіло документа.

2.1.4 Структура бази даних

Для забезпечення правильної роботи системи розподілу ліків було розроблено базу даних, що відповідає основним вимогам системи (див. рис. 2.6). Опис основних сутностей:

1) Medicines (Ліки)

— id: Унікальний ідентифікатор ліків.

— name: Назва медичного препарату.

— type: Тип ліків.

— unitName: Назва одиниці виміру.

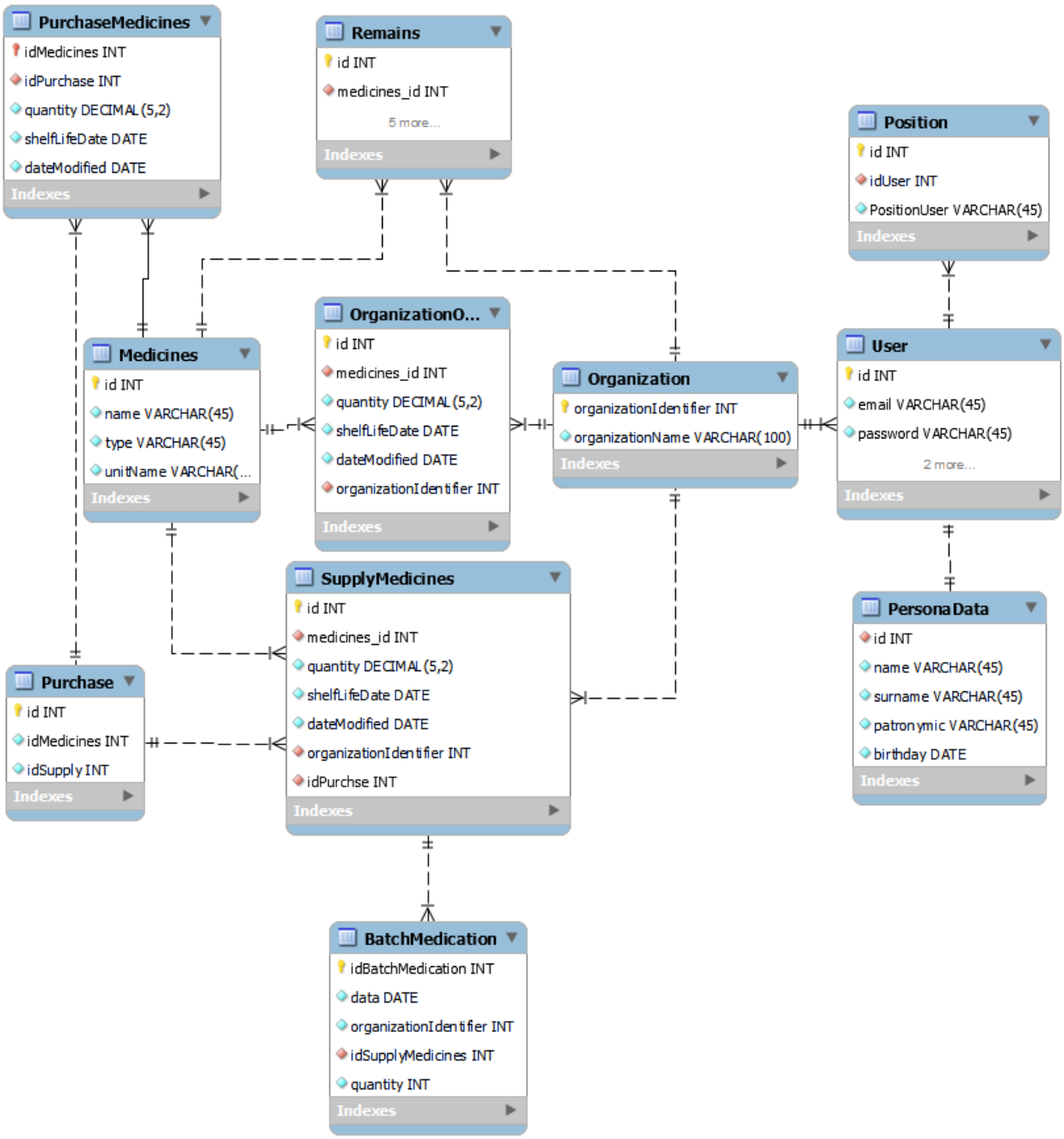

Рисунок 2.6 – Схема бази даних

2) Purchase (Закупівля)

— id: Унікальний ідентифікатор закупівлі.

— idMedicines: Ідентифікатор медичного препарату, який закуповується.

— idSupply: Ідентифікатор поставки.

— dateModified: Дата оновлення запису.

3) PurchaseMedicines (Закуплені ліки)

— idMedicines: Ідентифікатор ліків.

— idPurchase: Ідентифікатор закупівлі.

— quantity: Кількість закуплених ліків.

— shelfLifeDate: Термін придатності ліків.

— dateModified: Дата оновлення запису.

4) Remains (Залишки)

— id: Унікальний ідентифікатор залишків.

— medicines_id: Ідентифікатор ліків.

5) Organization (Організація)

— organizationIdentifier: Унікальний ідентифікатор організації.

— organizationName: Назва організації.

6) SupplyMedicines (Поставка ліків)

— id: Ідентифікатор поставки.

— medicines_id: Ідентифікатор ліків.

— quantity: Кількість ліків у поставці.

— shelfLifeDate: Термін придатності ліків.

— organizationIdentifier: Ідентифікатор організації, якій належить поставка.

7) BatchMedication (Партія медикації)

— idBatchMedication: Ідентифікатор партії медикації.

— date: Дата створення партії.

— organizationIdentifier: Ідентифікатор організації.

— idSupplyMedicines: Ідентифікатор поставки ліків.

— quantity: Кількість ліків у партії.

8) User (Користувач)

— id: Унікальний ідентифікатор користувача.

— email: Електронна пошта користувача.

— password: Пароль користувача.

9) PersonaData (Персональні дані)

— id: Ідентифікатор персональних даних.

— name: Ім'я особи.

— surname: Прізвище особи.

— patronymic: По батькові.

— birthday: Дата народження.

10) Position (Посада)

— id: Ідентифікатор посади.

— idUser: Ідентифікатор користувача, який обіймає посаду.

— PositionUser: Назва посади користувача.

Ця структура бази даних є комплексною та забезпечує повне управління всіма аспектами закупівель, поставок, а також інформацією про користувачів та організації в медичній системі. Вона підтримує ефективне зберігання даних і дозволяє різним сторонам системи доступ до актуальної інформації для своєчасного реагування на потреби медичного закладу.

2.1.5 Класова організація системи

В процесі розробки було розроблено класову структуру проєкту. На шашій діаграмі представлено об'єктно-орієнтовану структуру медичної інформаційної системи, що включає різні класи та зв'язки між ними, використовувані для управління закупівлями, поставками, залишками та організаційною інформацією(див. рис. 2.7) [12].

Опис класів та їх зв'язків:

— DataManager (Менеджер Даних)

Центральний клас, що відповідає за управління всіма даними в системі. Він зв'язаний з усіма основними класами, такими як Purchase, Supply, Medicines, і т.д., що дозволяє виконувати CRUD (Створення, Читання, Оновлення, Видалення) операції з даними.

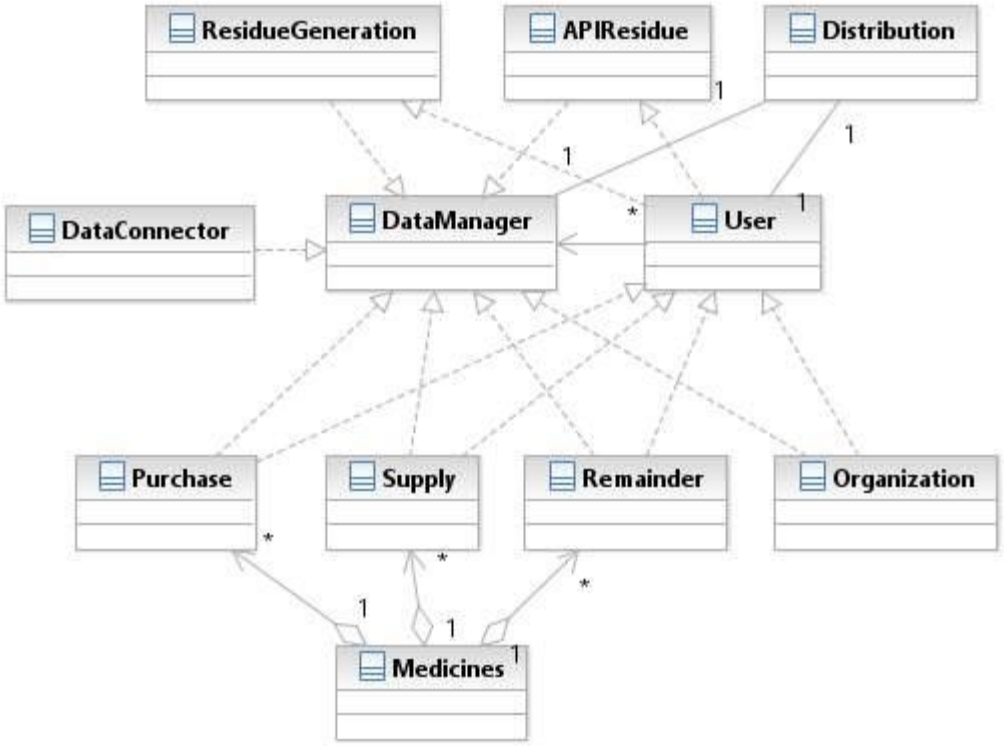

Рисунок 2.7 – Діаграма класів

— User (Користувач)

Клас, який представляє користувачів системи. Має взаємозв'язок з DataManager, що дозволяє користувачам доступ до функціоналу системи залежно від їхніх ролей і дозволів.

— DataConnector (З'єднувач Даних)

Служить мостом для зв'язку DataManager з зовнішніми джерелами даних або іншими системами, забезпечуючи інтеграцію та обмін даними.

— Medicines (Медикаменти)

Клас, що представляє медичні препарати. Зберігає інформацію про всі ліки, які закуповуються, постачаються, та залишки в організації.

— Purchase (Закупівля)

Управляє інформацією про закупівлі ліків, включаючи деталі закупівель та постачальників.

— Supply (Поставка)

Керує інформацією про поставки ліків до організації, включаючи дати та кількості.

Remainder (Залишки)

Відслідковує залишки ліків в організації, забезпечуючи актуальну інформацію про наявність.

— Organization (Організація)

Зберігає інформацію про організації, які беруть участь у процесі закупівлі та розподілу медикаментів.

Зв'язки між класами:

— Класи, такі як Purchase, Supply, Medicines, Remainder, і Organization мають взаємозв'язки, які індикують, що вони можуть взаємодіяти один з одним через DataManager. Це означає, що DataManager виступає як центральний вузол для управління усіма процесами та даними.

— Класи, такі як DataConnector, забезпечують інтеграцію з зовнішніми системами, розширюючи можливості DataManager.

— У системі використовуються також спеціалізовані класи для управління специфічними процесами, як ResidueGeneration та APIResidue для генерації та управління залишками, відповідно.

Ця об'єктно-орієнтована структура дозволяє ефективно моделювати та управляти складними процесами у медичній інформаційній системі, підвищуючи її гнучкість та масштабованість.

2.2 Застосування методик тестування

Для нашої медичної інформаційної системи, використання ефективних методик тестування є ключовим для забезпечення якості та надійності важливих системних компонентів. Викладення практичного застосування цих методик на реальних прикладах допомогло зрозуміти, як вони впливають на успішне

впровадження та експлуатацію системи. Ось приклади застосування різних технік тестування для критичних компонентів системи [13]:

1) Компонент "DataManager"

— Модульне тестування

Використання JUnit для тестування індивідуальних функцій в DataManager, забезпечуючи, що вони коректно обробляють дані з різних джерел і відправляють їх до відповідних компонентів (див. рис. 2.8).

```
public class DataManagerTest {
    private DataManager dataManager;
    @Before
    public void setUp() {
        dataManager = new DataManager();
    }
    @Test
    public void testFetchDataReturnsCorrectData() {
        assertEquals("Expected data", dataManager.fetchData("sourceId"));
    }
}
```

Рисунок 2.8 – Тестування fetchData

— Інтеграційне тестування

Застосування тестів для перевірки взаємодії між DataManager та базою даних, а також з іншими компонентами як DataConnector(див. рис. 2.9).

2) Компонент "User"

— Функціональне тестування

Перевірка всіх користувацьких сценаріїв, включаючи реєстрацію, вхід в систему, і взаємодію з інтерфейсом для зміни даних користувача (див. рис. 2.10).

```
public class DataManagerIntegrationTest {
    @Mock
    private Database database;
    @Mock
    private DataConnector dataConnector;
    private DataManager dataManager;
    @Before
    public void setUp() {
        MockitoAnnotations.initMocks(this);
        dataManager = new DataManager(database, dataConnector);
    }
    @Test
    public void testDataManagerInteractionWithDatabase() {
        when(database.getData("query")).thenReturn("data");
        assertEquals("data", dataManager.fetchDataFromDB("query"));
    }
    @Test
    public void testDataManagerInteractionWithDataConnector() {
        when(dataConnector.connect("apiEndpoint")).thenReturn("apiData");
        assertEquals("apiData", dataManager.fetchDataFromAPI("apiEndpoint"));
    }
}
```

Рисунок 2.9 – Тестування взаємодії DataManager з базою даних та DataConnector.

```
public class UserTest {
    private User user;
    @Before
    public void setUp() {
        user = new User("username", "password");
    }
    @Test
    public void testUserLogin() {
        assertTrue("User should be logged in with correct credentials", user
            .login("username", "password"));
    }
    @Test
    public void testUserChangeInfo() {
        user.changeInfo("newEmail@example.com");
        assertEquals("newEmail@example.com", user.getEmail());
    }
}
```

Рисунок 2.10 – Тестування процесів реєстрації та входу користувача.

— Тестування безпеки

Застосування тестів на проникнення та вразливості для переконання в тому, що особисті дані користувачів захищені від несанкціонованого доступу(рис. 2.11).

```
public class UserSecurityTest {
    private User user;
    @Before
    public void setUp() {
        user = new User("username", "password");
    }
    @Test
    public void testSqlInjection() {
        String maliciousInput = "1' OR '1'='1";
        assertFalse("Login should fail with SQL injection", user.login("username"
            , maliciousInput));
    }
}
```

Рисунок 2.11 – Тестування безпеки.

3) Компонент "Medicines"

— Тестування бази даних

Перевірка правильності CRUD операцій для сутностей ліків, зокрема перевірка обмежень інтегритету даних(див. рис. 2.12).

```
public class MedicinesDBTest {
    private MedicinesRepository repo;
    private Database database;
    @Before
    public void setup() {
        database = mock(Database.class);
        repo = new MedicinesRepository(database);
    }
    @Test
    public void testCreateMedicine() {
        Medicine medicine = new Medicine("Aspirin", "Pill");
        when(database.insert(any(Medicine.class))).thenReturn(true);
        assertTrue(repo.addMedicine(medicine));
    }
    @Test
    public void testUpdateMedicine() {
        Medicine medicine = new Medicine("Ibuprofen", "Pill");
        when(database.update(any(Medicine.class))).thenReturn(true);
        assertTrue(repo.updateMedicine(medicine));
    }
    @Test
    public void testDeleteMedicine() {
        when(database.delete(anyInt())).thenReturn(true);
        assertTrue(repo.deleteMedicine(1));
    }
}
```

Рисунок 2.12 – Тестування CRUD операцій.

— Тестування продуктивності

Аналіз відгуків системи на запити пов'язані з великою кількістю записів медикаментів для забезпечення вимог швидкодії. Тестування продуктивності реалізовано за допомогою JMeter для тестування навантаження:

4) Компоненти "Supply" та "Purchase"

— Тестування бізнес-логіки

Перевірка логіки обробки поставок та закупівель, забезпечуючи, що всі бізнес-правила дотримуються(див. рис. 2.13).

```
public class SupplyPurchaseLogicTest {
    private PurchaseManager purchaseManager;
    private SupplyManager supplyManager;
    @Before
    public void setup() {
        purchaseManager = new PurchaseManager();
        supplyManager = new SupplyManager();
    }
    @Test
    public void testPurchaseProcess() {
        Order order = new Order(1, 100, "Paracetamol");
        Purchase purchase = purchaseManager.processOrder(order);
        assertNotNull(purchase);
        assertEquals(100, purchase.getQuantity());
    }
    @Test
    public void testSupplyProcess() {
        Supply supply = supplyManager.receiveSupply(1, 100, "Paracetamol");
        assertNotNull(supply);
        assertEquals(100, supply.getQuantity());
    }
}
```

Рисунок 2.13 – Тестування бізнес-логіки.

— Енд-ту-енд тестування

Симуляція повного процесу закупівлі та доставки медикаментів від початку до кінця, забезпечуючи, що взаємодія між компонентами відповідає очікуванням(див. рис. 2.14).

```
public class EndToEndTest {
    private OrderSystem orderSystem;
    @Before
    public void setup() {
        orderSystem = new OrderSystem();
    }
    @Test
    public void testEndToEndFlow() {
        Customer customer = new Customer("Hospital X");
        Order order = new Order(customer, "Aspirin", 200);
        orderSystem.placeOrder(order);
        orderSystem.processOrder();
        orderSystem.shipOrder();
        assertTrue(order.isDelivered());
    }
}
```

Рисунок 2.14 – Тестування бізнес-логіки.

Кожен із цих підходів допомагає гарантувати, що кожний аспект системи функціонує належним чином і що система в цілому ефективна та надійна для кінцевих користувачів у медичній сфері.

2.3 Застосування методик тестування

Аналіз результатів тестування та оцінка якості роботи системи зазвичай включають збір та аналіз даних, отриманих в процесі різних типів тестування. Нижче представлено таблицю, яка використовувалася для збору та аналізу результатів тестування медичної інформаційної системи. Таблиця включає кілька критеріїв, таких як тип тестування, кількість виконаних тестів, кількість успішних проходжень тестів, кількість помилок, відсоток успіху та зауваження.

Таблиця 2.2 – Результати тестування

Тип тестування	Кількість тестів	Успішні проходження	Помилки	Відсоток успіху	Зауваження
Модульне тестування	100	98	2	98%	Мінімальні помилки в логіці
Інтеграційне тестування	75	72	3	96%	Проблеми зі зв'язком між модулями
Функціональне тестування	50	45	5	90%	Помилки в користувацькому інтерфейсі
Тестування продуктивності	30	27	3	90%	Потребує оптимізації продуктивності
Тестування безпеки	40	38	2	95%	Виявлено незначні уразливості

Аналіз результатів:

1) Модульне тестування

— Показало високу надійність індивідуальних компонентів.

— Рекомендується переглянути і виправити логічні помилки.

2) Інтеграційне тестування

— Хороші результати, але потрібно вдосконалити взаємодію між модулями для зниження кількості помилок.

3) Функціональне тестування

— Націлений на перевірку користувацьких сценаріїв, виявлення помилок у користувацькому інтерфейсі вимагає подальшої уваги.

4) Тестування продуктивності

— Система показує задовільні результати, але рекомендується оптимізація для підвищення продуктивності, особливо при великих обсягах даних.

5) Тестування безпеки

— Важливо продовжувати роботу над зміцненням захисту системи, навіть незначні уразливості можуть призвести до серйозних проблем.

Ця таблиця і результати аналізу допоможуть керівництву та розробникам зосередитися на ключових областях для покращення перед запуском системи в експлуатацію.

У цьому розділі ми ретельно розглянули процес розробки та тестування медичної інформаційної системи, акцентуючи увагу на критичній важливості інтеграції різноманітних методик тестування на всіх етапах процесу. Ефективність і надійність системи, що були підтверджені через численні тести, забезпечують не тільки виконання встановлених технічних вимог, а й гарантують високий рівень безпеки та користувацького задоволення. Отже, постійне вдосконалення процесів тестування та аналіз їх результатів є ключовим для успішного масштабування та адаптації системи до змінних потреб сучасного медичного середовища.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Динамічні явища на поверхні землі.

Динамічні явища на поверхні Землі включають широкий спектр природних процесів, які можуть мати значний вплив на безпеку людей та інфраструктури. Ці явища охоплюють геологічні, метеорологічні та гідрологічні зміни, кожне з яких має свої специфічні ризики та можливості для запобігання катастрофам [14].

Усі надзвичайні ситуації в Україні регулюються відповідно до національного законодавства, зокрема згідно з наступними ключовими документами: Кодекс цивільного захисту України, положення про підсистему реагування на надзвичайні ситуації та проведення аварійно-рятувальних та інших невідкладних робіт у рамках єдиної державної системи цивільного захисту, положення про штаб з ліквідації наслідків надзвичайної ситуації та організацію оперативно-технічної і звітної документації, наказ Державної служби України з надзвичайних ситуацій [15-18].

До динамічних явищ на поверхні землі належать абіотичні та біотичні небезпеки. Одним із проявів природних небезпек є стихійні лиха.

Стихійні лиха - це природні явища, які мають надзвичайний характер та призводять до порушення нормальної діяльності населення, загибелі людей, руйнування і знищення матеріальних цінностей.

За причиною виникнення стихійні лиха поділяють на: тектонічні (пов'язані з процесами, які відбуваються в надрах земної кори), топологічні (пов'язані з процесами, які відбуваються на поверхні землі), метеорологічні (пов'язані з процесами, які відбуваються в атмосфері).

Види абіотичних небезпек: літосферні, гідросферні, атмосферні, космічні. Серед літосферних небезпек розрізняють землетруси, вулкани, зсуви, селі.

— Землетрус — підземні поштовхи та коливання земної поверхні, зумовлені раптовими зміщеннями і розривами в корі або у верхній частині мантії, які передаються на великі відстані у вигляді пружних коливань.

— Зсув - сповзання мас гірських порід вниз по схилу, яке виникає через порушення рівноваги. Зсуви бувають повільні (см/доба), середньої швидкості (м/год), швидкі (км/год).

— Селі - це паводки з великою концентрацією ґрунту, мінеральних частин, каміння, уламків гірських порід (від 10-15 до 75% об'єму потоку). За складом матеріалу, що переносить потік, розрізняють: грязьові, грязекам'яні, водокам'яні.

Гідросферні небезпеки - повені, снігові лавини, шторми, цунамі.

— Повінь — значне затоплення місцевості внаслідок підйому рівня води в річці, озері, водосховищі. В Україні повені спостерігаються в басейнах Дніпра, Дністра, Прип'яті. В останні роки найчастіше спостерігаються в Закарпатті (Західний Буг та Тиса).

— Снігова лавина - величезна маса снігу, яка зсувається або падає зі стрімких гірських схилів, захоплюючи різні об'єкти, що трапляються на шляху. Лавина супроводжується утворенням передлавинної поверхневої хвилі, що мас найбільшу руйнівну силу. Розрізняють сухі (зимові) та мокрі (весняні) снігові лавини.

— Шторм - тривалий, дуже сильний вітер, що спричиняє значні руйнування на суші та велике хвилювання на морі.

— Цунамі - великі хвилі, що виникають на поверхні океану під час підводних землетрусів. Атмосферні небезпеки - бурі, урагани, тайфуни, цунамі, смерчі, морози, засуха тощо.

— Ураган - вітер руйнівної сили зі швидкістю 35 м/с.

— Буря - шторм, тривалий, дуже сильний вітер (понад 20 м/с), спричинений зазвичай циклоном.

— Смерч - сильний локальний атмосферний вихор (діаметр до 1000м), в якому повітря обертається зі швидкістю до 100 м/с.

3.2 Організація ведення робіт в аварійних умовах

Організація ведення робіт в аварійних умовах є надзвичайно важливим аспектом забезпечення безпеки та ефективності в небезпечних ситуаціях. Одні із основних кроків такій ситуації [18, 19]:

— оцінка ризиків: Перед початком будь-яких аварійних робіт необхідно провести оцінку ризиків. Визначте потенційні небезпеки, що виникають у зв'язку з аварійною ситуацією, і розробіть план заходів для зниження цих ризиків.

— створення команди: Сформуйте команду, яка буде відповідальна за проведення аварійних робіт. Команда повинна складатися з кваліфікованих фахівців з необхідними навичками та досвідом в роботі з аварійними ситуаціями.

— встановлення комунікації: Забезпечте ефективну систему комунікації всередині команди та з іншими відповідними структурами (пожежна служба, поліція, медична допомога тощо). Засоби зв'язку повинні бути надійними та доступними, щоб забезпечити швидкий обмін інформацією.

— планування та координація: Розробіть детальний план робіт, включаючи послідовність дій, розподіл завдань, використання необхідного обладнання та ресурсів. Координуйте дії команди таким чином, щоб досягти ефективного вирішення проблеми.

— навчання та підготовка працівників: Працівники, які беруть участь у роботах в аварійних умовах, повинні мати необхідні навички та знання. Забезпечте їм належне навчання з процедур безпеки, використання спеціального обладнання та заходів, які слід вживати під час аварій.

Складовою частиною процесу ліквідації наслідків НС є рятування людей. Цей процес представляє собою взаємопов'язаний комплекс робіт, які за характером виконання діляться на три специфічні групи: рятувальні, спеціальні і допоміжні.

Рятувальні роботи, що безпосередньо пов'язані із рятуванням людей включають у себе:

— пошук постраждалих у місцях їхнього можливого блокування;

— деблокування постраждалих (забезпечення доступу до них);

— надання постраждалим домедичної допомоги;

— евакуація постраждалих із місць блокування.

— Спеціальні роботи включають:

— гасіння пожеж;

— ліквідацію аварії на комунально-енергетичних і технологічних мережах;

— улаштування проїздів (проходів) у завалах;

— зміцнення (обвалення) нестійких конструкцій.

У результаті виконання спеціальних робіт створюються умови найбільш сприятливі для виконання рятувальних робіт і запобігання додаткового ураження людей. Допоміжні роботи пов'язані з інженерною та організаційною підготовкою ділянки рятувальних робіт та робочих місць. До них відносяться:

— розчищення майданчиків;

— установлення на них техніки;

— огородження, попереджувальних знаків;

— освітлення робочих місць.

У практиці рятування постраждалих при обваленні будівель використовуються наступні рятувальні технології [20]:

— пошук постраждалих за допомогою спеціально навчених собак (кінологічний спосіб);

— пошук постраждалих за допомогою спеціальних приладів;

— деблокування постраждалих із завалу, що складається із дрібних уламків, способом розбирання завалу зверху;

— деблокування постраждалих із завалу, що складається із великих уламків, способом розбирання завалу зверху;

— деблокування постраждалих із завалу способом суцільного горизонтального розбирання;

— деблокування постраждалих способом улаштування лазу у завалі;

— деблокування постраждалих із завалених приміщень;

— деблокування постраждалих з верхніх поверхів будівлі з використанням вертольоту;

— деблокування постраждалих з верхніх поверхів будівлі із застосуванням автодрабин;

— порятунок постраждалих з верхніх поверхів будівлі за допомогою автовишок та автопідйомників;

— порятунок постраждалих з верхніх поверхів будівлі по збереженим або тимчасово відновленим сходовим маршам;

— порятунок постраждалих з верхніх поверхів будівлі з використанням канатної дороги;

— деблокування постраждалих з верхніх поверхів будівлі з використанням альпіністських засобів.

У загальному вигляді процес рятування постраждалих може бути представлений як комплексний технологічний процес, що включає етапи [21]:

— загальна спеціальна розвідка осередку ураження та об'єкта робіт;

— підготовчі роботи;

— аварійно-технічні роботи;

— пошуково-рятувальні роботи;

— роботи з деблокування та витягання постраждалих;

— надання домедичної та лікарської допомоги, медична евакуація поранених;

— евакуація, упізнання та поховання загиблих.

На кожному з наведених технологічних етапів здійснюються відповідні види робіт, а вони, у свою чергу виконуються певними способами. Найбільш складним технологічним етапом при обваленні будівель і споруд є інженерні роботи з деблокування та витягання постраждалих.

ВИСНОВКИ

У процесі розробки та тестування системи розподілу лікарських препаратів ми досягли значного прогресу у вдосконаленні її функціональності та безпеки. Застосування сучасних методологій розробки дозволило нам швидко адаптуватися до змінюваних вимог та удосконалити систему з високим ступенем точності. Тестування системи в різних режимах, включаючи модульне, інтеграційне, функціональне та тестування безпеки, підтвердило її надійність і відповідність встановленим стандартам якості.

Проведений аналіз результатів тестування демонструє, що система ефективно справляється з завданнями, для яких вона була розроблена. Водночас ми ідентифікували декілька напрямків для подальших удосконалень, особливо у контексті забезпечення вищого рівня автоматизації та оптимізації робочих процесів. Ці напрямки включають покращення інтерфейсів взаємодії з користувачами та розширення функціоналу системи для забезпечення більшої гнучкості в управлінні даними.

У ході даного дослідження було проведено розробку та глибокий аналіз системи розподілу лікарських препаратів, що демонструє значний потенціал для підвищення ефективності управління медичними ресурсами. Система була створена з використанням передових технологій та об'єктно-орієнтованих підходів, забезпечуючи її високу адаптивність та інтеграційну здатність із існуючими медичними інформаційними системами.

Особлива увага в проєкті приділялася аспектам безпеки і конфіденційності даних, що є критично важливими для медичної сфери. Використання комплексних тестів, включаючи модульне, інтеграційне та функціональне тестування, підтвердило високу надійність системи та її спроможність ефективно обробляти дані різного роду без порушення цілісності та конфіденційності інформації.

Результати тестування та аналіз даних виявили деякі області для подальшого удосконалення, зокрема потребу в оптимізації деяких процесів та поліпшенні

користувацького інтерфейсу для забезпечення більшої зручності використання. Ці висновки надають важливі настанови для подальшої роботи та розвитку системи.

Завершення цього проєкту відкриває шляхи для його практичного застосування у медичних установах, що може суттєво вдосконалити процеси управління запасами лікарських засобів, зменшити витрати та підвищити якість обслуговування пацієнтів.

На основі проведеного дослідження та отриманих результатів, є декілька ключових рекомендацій для подальших розробок та досліджень у сфері медичних інформаційних систем, зокрема систем розподілу лікарських препаратів.

Перш за все, рекомендується продовжити роботу над оптимізацією інтерфейсу користувача. Це може включати розробку більш інтуїтивно зрозумілих меню, покращення візуального представлення даних, та створення більш зручних інструментів для введення та аналізу даних. Ці зміни допоможуть забезпечити більшу ефективність використання системи медичним персоналом.

Важливим напрямком розвитку є підвищення масштабованості системи. Рекомендується розробити рішення, які дозволяють системі легко адаптуватися до збільшення обсягів даних та користувачів без втрати продуктивності. Це може включати використання хмарних технологій, розподілені бази даних та інші сучасні підходи до обробки великих обсягів даних.

Крім того, слід звернути увагу на подальше вдосконалення алгоритмів машинного навчання для аналізу даних, які можуть виявити складні зв'язки та тенденції в даних, що надходять в систему. Це дозволить не тільки поліпшити точність прогнозувань та прийняття рішень, але й надати можливість для раннього виявлення потенційних проблем в логістиці та управлінні запасами.

Застосування цих рекомендацій може суттєво підвищити потенціал системи та її цінність для медичних установ, забезпечуючи високий рівень обслуговування пацієнтів та оперативне управління медичними ресурсами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Наказ України «Про затвердження Порядку розміщення на інформаційних стендах у закладах охорони здоров'я інформації щодо наявності лікарських засобів, витратних матеріалів, медичних виробів та харчових продуктів для спеціального дієтичного споживання, отриманих за кошти державного та місцевого бюджетів, благодійної діяльності і гуманітарної допомоги» [Електронний ресурс] // №459 від 26.04.2017 р. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0841-17

2. Наказ України «Про здійснення заходів з контролю використання лікарських засобів та медичних виробів, закуплених за бюджетні кошти » [Електронний ресурс] // №459 від 26.04.2017 р. – Режим доступу: https://eliky.in.ua/web/media/orders/moz_order_%E2%84%96509_(02-06-2016).pdf

3. Dumez H., Minvielle E. L'e-santé rend-elle la démocratie sanitaire pleinement performative? (2017) Systèmes d'Information et Management 22(1):9-37

4. Закупівлі ліків [Електронний ресурс] // Режим доступу: https://moz.gov.ua/zakupivli-likiv

5. «Медзакупівлі України» представили стратегію розвитку на 2021-2023 рр [Електронний ресурс] // Режим доступу: https://moz.gov.ua/article/reform-plan/medzakupivli-ukraini-predstavili-strategiju-rozvitku-na-2021-2023-rr

6. Відомості про лікарські засоби/препарати, придбані за бюджетні кошти, відомості про розподілення таких ліків між закладами охорони здоров'я та їх залишки в кожному з них [Електронний ресурс] // Режим доступу: https://data.gov.ua/pages/835-recm-health-care-medicine

7. Passwords vs. Private Keys, Johannes Weber [Електронний ресурс] // Режим доступу: https://blog.webernetz.net/ passwords-vs-private-keys/

8. Hassan A. Afyouni Database Security and Auditing: Protecting Data Integrity and Accessibility / Hassan A. Afyouni – Course Technology, 2012. — 429 pages.

9. David Knox Oracle Database 12C Security / David Knox – McGraw-Hill Education, 2016 — 768 pages.

10. MySQL [Електронний ресурс] — Режим доступу: https://www.mysql.com/

11. Database security issues [Електронний ресурс] — Режим доступу: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2014/july/top-10-common-database-security-issues/

12. C# documentation [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/

13. Безпека SQL Server [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/dotnet/framework/data/adonet/sql/sql-server-security

14. Кодекс цивільного захисту України [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/5403-17#Text

15. Про затвердження Положення про підсистему реагування на надзвичайні ситуації, проведення аварійно-рятувальних та інших невідкладних робіт єдиної державної системи цивільного захисту : наказ Міністерства внутрішніх справ України від 04.05.2016 №356.

16. Про затвердження Положення про штаб з ліквідації наслідків надзвичайної ситуації та видів оперативно-технічної і звітної документації штабу з ліквідації наслідків надзвичайної ситуації : наказ Міністерства внутрішніх справ України від 26.12.2014 №1406.

17. Про організацію роботи штабу з ліквідації наслідків надзвичайної ситуації та забезпечення його готовності : наказ Державної служби України з надзвичайних ситуацій від 16.03.2015 №149.

18. Рекомендації для роботодавців щодо організації виконання робіт підвищеної небезпеки під час воєнних (бойових) дій [Електронний ресурс] – Режим доступу: https://dsp.gov.ua/faq/rekomendatsii-dlia-robotodavtsiv-shchodo-orhanizatsii-vykonannia-robit-pidvyshchenoi-nebezpeky-pid-chas-voiennykh-boiovykh-dii/

19. Аветисян В.Г., Тригуб В.В., Грицина І.М., Остапов К.М. Організації аварійнорятувальних робіт : курс лекцій – Х: НУЦЗУ, 2017. – 141 с.

20. Організація аварійно-рятувальних робіт : навчальний посібник /Р.Т. Ратушний, В.Б. Лоїк, О.Д. Синельніков, В.М. Ковальчук – Львів: Видавництво ЛДУ БЖД, 2020. – 394 с.

21. Методичні рекомендації щодо розроблення планів локалізації і ліквідації аварій та їх наслідків : наказ Державної служби України з надзвичайних ситуацій від 17 травня 2022 року за №253

ДОДАТКИ

Додаток А - Публікація у науковому виданні

VII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.41
Кривий Ю. – т. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

АНАЛІЗ ПІДХОДІВ МОДУЛЬНИХ ТА ІТЕРАЦІЙНИХ ПІДХОДІВ ДО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Науковий керівник: маг. Стефанишин І.М.

Kryvy Y.
Ternopil Ivan Puluj National TechnicalUniversity

ANALYSIS OF MODULAR AND ITERATIVE APPROACHES TO SOFTWARE TESTING

Supervisor: m., Stefanyshyn V.

Ключові слова: Тестування, модульність, ітераційність.
Key words: Testing, modularity, iterability.

У сфері розробки програмного забезпечення, тестування відіграє вирішальну роль у забезпеченні якості та надійності кінцевого продукту. Різні підходи до тестування, зокрема модульні та ітераційні методики, мають свої переваги та обмеження, які впливають на вибір методу тестування залежно від характеристик проекту та цілей розробки.

Модульний підхід до тестування зосереджується на перевірці окремих компонентів програмного забезпечення перед їх інтеграцією в більш складні системи. Цей метод дозволяє виявляти помилки на ранніх етапах розробки, що сприяє більш легкому їх виправленню та зниженню загальних витрат на тестування. Модульне тестування виконується розробниками, які пишуть тестові випадки для кожного компонента незалежно, що забезпечує високий рівень деталізації та ізоляції помилок.

Ітераційний підхід, з іншого боку, передбачає повторне тестування програмного забезпечення на кожному етапі його розробки. Це означає, що продукт регулярно оцінюється на предмет помилок та вдосконалень протягом кількох ітерацій, що допомагає забезпечити більшу стабільність та надійність системи перед її випуском. Ітераційний процес залучає різні рівні тестування, включаючи інтеграційне, системне та приймальне, що дозволяє всебічно оцінити продукт.

В ітераційному підході важливою є регулярна взаємодія з клієнтом чи кінцевим користувачем для збору зворотного зв'язку після кожної ітерації. Це дозволяє оперативно вносити корективи до продукту відповідно до потреб користувача. У контексті модульного тестування, залучення зацікавлених сторін може бути більш обмеженим, оскільки фокус зосереджений на внутрішній якості та функціональності окремих модулів.

Автоматизація тестування може істотно підвищити ефективність обох підходів. У модульному тестуванні автоматизовані тестові скрипти можуть швидко виконувати рутинні перевірки окремих компонентів. У рамках ітераційного підходу, автоматизація допомагає управляти більшою кількістю тестів через короткі релізні цикли, забезпечуючи високу якість кінцевого продукту.

Континуальна інтеграція (CI) ідеально підходить для ітераційного підходу, оскільки вона передбачає регулярну збірку та тестування продукту при кожному оновленні коду. Це сприяє виявленню та виправленню помилок на ранніх стадіях розробки. Хоча CI також може бути використане у модульному тестуванні, його ефективність особливо висока при поєднанні з ітераційними методами.

Аналізуючи обидва підходи, важливо зазначити, що вибір методики тестування має базуватися на конкретних потребах проекту. Модульне тестування є ефективним для проектів, де велика кількість незалежних компонентів потрібно протестувати окремо, тоді як ітераційний підхід краще підходить для комплексних систем, де потрібна постійна взаємодія та відповідність між усіма частинами проекту.

Важливим аспектом обох методик є також вплив на час розробки та витрати. Модульне тестування може зменшити загальний час розробки, дозволяючи паралельну роботу над різними частинами проекту, тоді як ітераційний метод може вимагати більше часу та ресурсів через необхідність постійного перегляду та оцінювання прогресу.

Завершення проекту залежить від вдалої інтеграції обраного підходу до тестування у загальну стратегію розробки, що сприяє досягненню високих стандартів якості та відповідності до очікувань замовника.

Література:

1. Roman, S., & Petrenko, A. (2014). "Python Testing: Beginner's Guide." Packt Publishing, Birmingham, UK.
2. Kaner, C., & Bond, W.P. (2012). "Lessons Learned in Software Testing: A Context-Driven Approach." Wiley, Hoboken, NJ
3. Rainsberger, J. B. (2014). "Integration Tests Are a Scam.
4. Crispin, L., & Gregory, J. (2014). "Agile Testing: A Practical Guide for Testers and Agile Teams." Addison-Wesley Professional, Boston, MA.
5. Crispin, L., & Gregory, J. (2014). "Agile Testing: A Practical Guide for Testers and Agile Teams." Addison-Wesley Professional.

Додаток Б – Додаткові графічні елементи

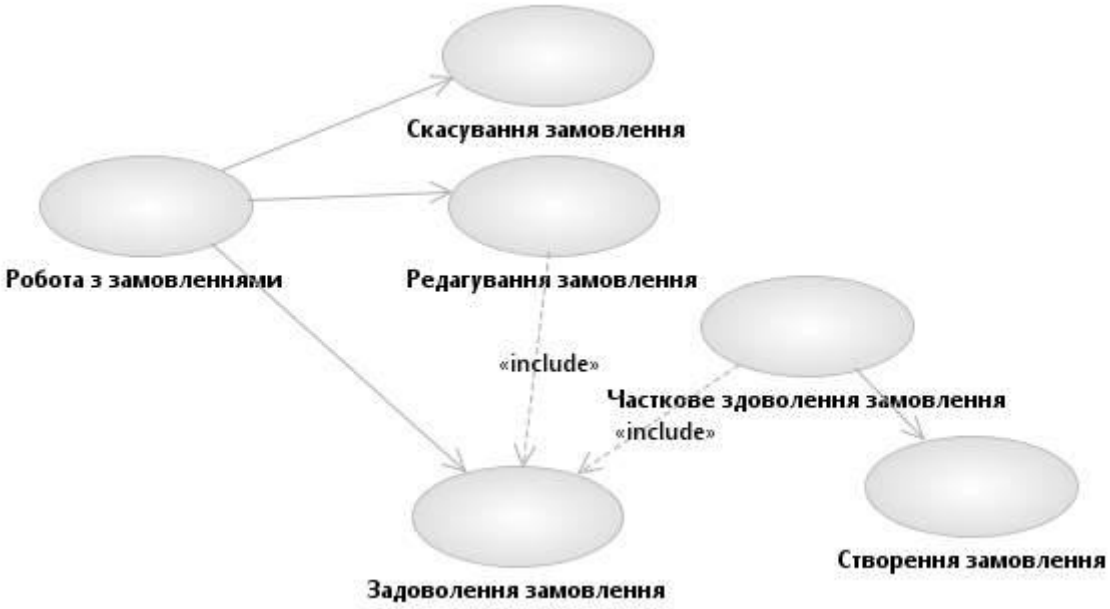

Рисунок Б.1 - Діаграма варіанту використання «Робота з замовленнями»

Рисунок Б.2 - Діаграма варіанту використання «Створення замовлення»

Рисунок Б.3 - Діаграма варіанту використання «Робота з залишками»

Лістинг коду В.1 - Клас DataManager

```
using System;
using System.Data.SqlClient;
public class DataManager
{
    private string connectionString;
    public DataManager(string connectionString)
    {
        this.connectionString = connectionString;
    }
    public void SaveData(string data)
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();
            var command = new SqlCommand("INSERT INTO Data (Info) 
VALUES (@data)", connection);
            command.Parameters.AddWithValue("@data", data);
            command.ExecuteNonQuery();
        }
    }
    public string LoadData()
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();
            var command = new SqlCommand("SELECT TOP 1 Info FROM Data 
ORDER BY ID DESC", connection);
            return command.ExecuteScalar().ToString();
        }
    }
}
```

Лістинг коду В.2 - Клас User

```
public class User
{
    public string Username { get; private set; }
    public string Password { get; private set; }
    public User(string username, string password)
    {
        Username = username;
        Password = password;
    }
    public bool CheckPassword(string password)
    {
        return password == Password;
    }
    public void ChangePassword(string newPassword)
    {
        Password = newPassword;
    }
}
```

Лістинг коду В.3 - Клас MedicationManager

```
using System.Collections.Generic;
public class MedicationManager
{
    private List<Medication> medications = new List<Medication>();
    public void AddMedication(Medication medication)
    {
        medications.Add(medication);
    }
    public bool RemoveMedication(Medication medication)
    {
        return medications.Remove(medication);
    }
    public List<Medication> GetAllMedications()
    {
        return medications;
    }
}
```

Лістинг коду В.4 - Клас SupplyManager

public class SupplyManager

```
{
    private List<Supply> supplies = new List<Supply>();
    public void ReceiveSupply(Supply supply)
    {
        supplies.Add(supply);
    }
    public IEnumerable<Supply> GetCurrentSupplies()
    {
        return supplies;
    }
}
```

Лістинг коду В.5 - Взаємодія між компонентами

```
public class AuthenticationService
{
    private DataManager dataManager;
    public AuthenticationService(DataManager dataManager)
    {
        this.dataManager = dataManager;
    }
    public bool AuthenticateUser(string username, string password)
    {
        var user = dataManager.LoadUser(username);
        return user != null && user.CheckPassword(password);
    }
}
```

Лістинг коду В.6 - Клас ErrorHandling

```
public class ErrorHandling
{
    public void HandleError(Exception exception)
    {
        // Логування деталей помилки
        LogError(exception);
```

```
            // Відображення дружнього повідомлення користувачу
            ShowFriendlyMessage();
        }
        private void LogError(Exception exception)
        {
            Console.WriteLine($"Error: {exception.Message}");
        }
        private void ShowFriendlyMessage()
        {
            Console.WriteLine("An error occurred. Please try again
later.");
        }
    }
```

Лістинг коду В.7 - Клас Configuration

```
    public class Configuration
    {
        public string GetDatabaseConnectionString()
        {
            return "Server=myServerAddress;Database=myDataBase;User
Id=myUsername;Password=myPassword;";
        }
        // Можливість додати додаткові конфігураційні параметри
    }
```

Лістинг коду В.8 - Клас Reporting

```
    public class Reporting
    {
        public void GenerateReport(List<Data> reportData)
        {
            // Псевдокод для демонстрації генерації звіту
            foreach (var data in reportData)
            {
                Console.WriteLine($"Report Entry: {data.Details}");
            }
        }
    }
```

Лістинг коду В.9 - Клас Notification

```
public class Notification
{
    public void SendNotification(string message, string userContact)
    {
        // Псевдокод для відправки сповіщення
        Console.WriteLine($"Sending notification to {userContact}:
{message}");
    }
}
```

Лістинг коду В.10 - Клас HealthCheck

```
public class HealthCheck
{
    public bool CheckDatabaseConnection()
    {
        try
        {
            // Перевірка з'єднання з базою даних
            Console.WriteLine("Checking database connection...");
            // Логіка для перевірки з'єднання
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Database connection failed:
{ex.Message}");
            return false;
        }
    }
    public bool CheckServiceAvailability()
    {
        // Перевірка доступності зовнішніх сервісів
        return true; // Припустимо, що сервіс доступний
    }
}
```

Лістинг коду В.11 - Клас ConfigManager

```
public class ConfigManager
{
    Dictionary<string, string> settings = new Dictionary<string, string>();
    public string GetSetting(string key)
    {
        if (settings.ContainsKey(key))
        {
            return settings[key];
        }
        return null;
    }
    public void SetSetting(string key, string value)
    {
        settings[key] = value;
    }
}
```

Лістинг коду Г.1 – Створення таблиці Medicines

```
CREATE TABLE Medicines (
    idMedicines INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(45),
    type VARCHAR(45),
    unitName VARCHAR(45)
);
```

Лістинг коду Г.2 – Створення таблиці Purchase

```
CREATE TABLE Purchase (
    id INT AUTO_INCREMENT PRIMARY KEY,
    idMedicines INT,
    idSupply INT,
    FOREIGN KEY (idMedicines) REFERENCES Medicines(idMedicines),
    FOREIGN KEY (idSupply) REFERENCES SupplyMedicines(id)
);
```

Лістинг коду Г.3 – Створення таблиці SupplyMedicines

```
CREATE TABLE SupplyMedicines (
    id INT AUTO_INCREMENT PRIMARY KEY,
    medicines_id INT,
    quantity DECIMAL(5,2),
    shelfLifeDate DATE,
    dateModified DATE,
    organizationIdentifier INT,
    FOREIGN KEY (medicines_id) REFERENCES Medicines(idMedicines)
);
```

Лістинг коду Г.4 – Створення таблиці PurchaseMedicines

```
CREATE TABLE PurchaseMedicines (
    idMedicines INT,
    idPurchase INT,
    quantity DECIMAL(5,2),
```

```
    shelfLifeDate DATE,
    dateModified DATE,
    PRIMARY KEY (idMedicines, idPurchase),
    FOREIGN KEY (idMedicines) REFERENCES Medicines(idMedicines),
    FOREIGN KEY (idPurchase) REFERENCES Purchase(id)
);
```

Лістинг коду Г.5 – Створення таблиці Remains

```
CREATE TABLE Remains (
    id INT AUTO_INCREMENT PRIMARY KEY,
    medicines_id INT,
    FOREIGN KEY (medicines_id) REFERENCES Medicines(idMedicines)
);
```

Лістинг коду Г.6 – Створення таблиці Organization

```
CREATE TABLE Organization (
    organizationIdentifier INT AUTO_INCREMENT PRIMARY KEY,
    organizationName VARCHAR(100)
);
```

Лістинг коду Г.7 – Створення таблиці OrganizationOfMedicines

```
CREATE TABLE OrganizationOfMedicines (
    id INT AUTO_INCREMENT PRIMARY KEY,
    medicines_id INT,
    quantity DECIMAL(5,2),
    shelfLifeDate DATE,
    dateModified DATE,
    organizationIdentifier INT,
    FOREIGN KEY (medicines_id) REFERENCES Medicines(idMedicines),
    FOREIGN KEY (organizationIdentifier) REFERENCES
Organization(organizationIdentifier)
);
```

Лістинг коду Г.8 – Створення таблиці User

```
CREATE TABLE User (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    email VARCHAR(45),
    password VARCHAR(45)
);
```

Лістинг коду Г.9 – Створення таблиці Position

```
CREATE TABLE Position (
    id INT AUTO_INCREMENT PRIMARY KEY,
    idUser INT,
    positionUser VARCHAR(45),
    FOREIGN KEY (idUser) REFERENCES User(id)
);
```

Лістинг коду Г.10 – Створення таблиці PersonaData

```
CREATE TABLE PersonaData (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(45),
    surname VARCHAR(45),
    patronymic VARCHAR(45),
    birthday DATE,
    idUser INT,
    FOREIGN KEY (idUser) REFERENCES User(id)
);
```

Лістинг коду Г.11 – Створення таблиці BatchMedication

```
CREATE TABLE BatchMedication (
    idBatchMedication INT AUTO_INCREMENT PRIMARY KEY,
    data DATE,
    organizationIdentifier INT,
    idSupplyMedicines INT,
    quantity INT,
    FOREIGN KEY (organizationIdentifier) REFERENCES
Organization(organizationIdentifier),
    FOREIGN KEY (idSupplyMedicines) REFERENCES SupplyMedicines(id)
);
```

Додаток Д – Диск із кваліфікаційною роботою бакалавра