Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделей процесів розгортання мікросервісів
в ІТ-проєктах валідації електронних фармацевтичних документів
(тема)

Виконав:
здобувач 2 року навчання,
групи УПГІТм-23-1
Владислав КРИВЕНКО
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Управління проєктами
в галузі інформаційних технологій
(повна назва освітньої програми)

Керівник: проф. каф. ІУС Сергій ЧАЛИЙ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри ІУС _____ Костянтин ПЕТРОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп’ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Управління проєктами в галузі інформаційних технологій
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“ 21 ” квітня 20 25 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кривенку Владиславу Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження моделей процесів розгортання мікросервісів в IT-проєктах валідації електронних фармацевтичних документів

затверджена наказом по університету від “ 28 ” березня 2025 р. № 235Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії “ 04 ” червня 2025 р.

3. Вихідні дані до роботи Публічні набори даних структур електронних фармацевтичних документів, технічна документація до стандартів валідації фармацевтичних документів, журнали логів розгортання мікросервісів, наукові статті розгортання мікросервісної архітектури

4. Перелік питань, що потрібно опрацювати у роботі аналіз процесів розгортання мікросервісів, дослідження структури процесу розгортання мікросервісів, аналіз моделей розгортання мікросервісів, аналіз задачі валідації фармацевтичних документів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	21.04.2025 – 24.04.2025	Виконано
2	Огляд й аналіз наукових джерел	24.04.2025 – 27.04.2025	Виконано
3	Постановка задачі	27.04.2025 – 30.04.2025	Виконано
4	Теоретичне дослідження	01.05.2025 – 03.05.2025	Виконано
5	Практична реалізація	04.05.2025 – 06.05.2025	Виконано
6	Підготовка пояснювальної записки	07.05.2025 – 15.05.2025	Виконано
7	Захист роботи	04.06.2023	Виконано

Дата видачі завдання 21 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. каф. ІУС Сергій Чалий
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 123 сторінок, 46 рисунків, 14 таблиць, 12 лістингів, 29 джерел.

ВАЛІДАЦІЯ, ЕЛЕКТРОННИЙ ФАРМАЦЕВТИЧНИЙ ДОКУМЕНТ, МІКРОСЕРВІСНА АРХІТЕКТУРА, МОДЕЛЬ РОЗГОРТАННЯ.

Об’єкт дослідження – процеси розгортання мікросервісів в IT-проєктах валідації електронних фармацевтичних документів.

Мета роботи – дослідження моделей розгортання мікросервісів з метою вибору оптимального підходу до побудови надійної, масштабованої системи для валідації електронних фармацевтичних документів.

Методи дослідження – аналіз та порівняння моделей розгортання мікросервісів, проєктування комбінованої моделі розгортання.

У роботі проведено аналіз існуючих моделей з використанням мікросервісної архітектури, виконано порівняння моделей, виявлені переваги та недоліки існуючих моделей розгортання. Проведено дослідження існуючих стандартів щодо валідації фармацевтичних документів. Проведено порівняння підходів щодо перевірки фармацевтичної документації, виявлені переваги та недоліки кожного з підходів. Проведено аналіз проєкту, орієнтованого на оптимізацію процесу перевірки документів, використовуючи міжнародні стандарти щодо постачання фармацевтичних документів.

Отримана комбінована модель розгортання системи, архітектурою якої є мікросервіси, демонструє масштабованість та гнучкість, оскільки кожен сервіс у системі може бути розгорнутий незалежно від інших. При використанні отриманої моделі спостерігається значно зменшена кількість простоїв системи, що гарантує цілісність та безперечність розгортань.

ABSTRACT

Master’s thesis: 123 pages, 46 figures, 14 tables, 12 listings, 29 sources.

DEPLOYMENT MODEL, ELECTRONIC PHARMACEUTICAL DOCUMENT, MICROSERVICE ARCHITECTURE, VALIDATION.

The object of research is the processes of deploying microservices in IT projects for the validation of electronic pharmaceutical documents.

The purpose of the work is to explore microservices deployment models to choose the optimal approach to building a reliable, scalable system for validating electronic pharmaceutical documents.

Research methods – analysis and comparison of microservices deployment models, design of a combined deployment model.

The paper analyzes existing software deployment models with microservice architecture, compares models, identifies advantages and disadvantages of existing deployment models. Researched existing standards for validation of pharmaceutical documents. Comparisons of approaches to verifying pharmaceutical documentation are carried out, identifies advantages and disadvantages of each approach. Analysis of a project focused on optimizing the document verification process using international standards for the delivery of pharmaceutical documents.

The resulting combined deployment system model, which is based on a microservice architecture, demonstrates scalability and flexibility, since each service in the system can be deployed independently of the others. Using the resulting model, the number of simple systems is significantly reduced, which guarantees the integrity and indisputability of the deployment.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ – програмне забезпечення;

API – application programming interface;

CD – continuous delivery;

CI – continuous integration;

CTD – common technical document;

ECTD – electronic common technical Document;

HTTP – hypertext transfer protocol;

JSON – javascript object notation;

PDF – portable document format;

SCTD – structured common technical document;

SMB – server message block;

SOLID – single responsibility, open-closed, liskov substitution, interface segregation, dependency injection;

WBS – work breakdown structure;

XML – extensible markup language;

XSD – XML schema definition;

YAML – yet another markup language.

ВСТУП

У сучасному світі системи та мережі поширюють свій вплив у всіх сферах діяльності людини. Інформаційні технології надають широкі можливості кожному користувачу, оскільки він може використовувати їх будь-яким способом.

Зазвичай, бізнес має деякі вимоги щодо таких систем, а такі вимоги можуть бути звичайними, що в наш час дуже велика рідкість, оскільки швидкість розвитку сучасних технологій не стоїть на місці. Критерії щодо систем і бізнесу змінюються майже кожен день. Така поведінка обумовлена швидкістю розвитку нових систем і різних технологій. Не на останньому місці стоїть й сам користувач, який зазвичай досить критично ставиться до різних систем та їх функціоналу, тому, як правило, бізнес працює з деякої командою IT-фахівців, які допомагають реалізувати всі ідеї. В умовах сучасного світу процес доставки програмного коду займає одну з передових позицій в інформаційних технологіях, тому що деякі моделі та підходи надають можливість оптимізувати цей процес для підвищення ефективності роботи.

Однак, не всі моделі розгортання є гарним рішенням в тій чи іншій архітектурі або задачі. Кожна з моделей має свої переваги та недоліки, які суттєво впливають на процес розгортання.

Прикладом є система, яка займається оптимізацією такої задачі, як перевірка фармацевтичних документів. Сьогодні така бізнес-задача є досить критичною, оскільки це питання стосується здоров’я людини, а медицина знаходиться у постійному розвитку, як й інформаційні технології. Саме тому питання розробки моделі розгортання мікросервісів в IT-проєктах, орієнтованих на валідацію електронних фармацевтичних документів є досить важливим.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз процесів розгортання мікросервісів

1.1.1 Дослідження структури процесу розгортання мікросервісів

У реаліях 2025 року сучасне суспільство потребує значної взаємодії з різними програмними продуктами, які сприяють оптимізації багатьох бізнес-процесів. Ці процеси, як правило, складні і вимагають багато часу для реалізації за допомогою ручних методів. Оптимізація є важливим аспектом не лише в людській сфері, але й в операціях інформаційних технологій, які можуть потребувати багато часу для досягнення певних результатів.

Сфера охорони здоров’я складається з багатьох важливих для життя бізнес-процесів. Наприклад, процедура реєстрації нових ліків є важким процесом, і передбачає багато взаємодії з різними зацікавленими сторонами, такими, як регуляторні органи та організації. Перш за все, необхідно зібрати значний обсяг даних про лікарський засіб, а саме: дані про його склад, властивості, клінічні оцінки, безпеку та ефективність – ці напрямки передбачають підготовку та оформлення різних типів документів. Основне питання полягає в тому, чи може існувати процес оптимізації, який би відповідав усім вимогам щодо стандартів цих документів.

Розробка такого рішення дозволить корпоративним компаніям пропонувати своїм клієнтам комплексні послуги перевірки фармацевтичної документації з урахуванням багатьох регіональних стандартів. Такі системи перевірки залишаться актуальними в майбутньому, оскільки технологічний прогрес ще більше підвищить швидкість роботи. Цей прогрес буде означати створення новіших, ефективніших алгоритмів і параметрів перевірки [1].

Основну увагу треба зосередити на створені архітектури корпоративного рішення, оскільки вона повинна відповідати сучасним

стандартам щодо створення програмного забезпечення (ПЗ). Використання мікросервісної архітектури є досить гарним рішенням, оскільки вона надає можливість створювати корпоративні рішення з можливістю легко масштабуватись при будь-якій необхідності. Питання підтримки та масштабування ПЗ сьогодні є досить важливим серед розробників. Підбір правильної архітектури надає можливість одразу використовувати технології та інтеграції з існуючими системами, що, звичайно, є вагомою перевагою при створенні ПЗ.

Кожне рішення з будь-якою архітектурою має власні особливості створення програмного коду, використання різноманітних шаблонів проєктування, різних технік збірки та доставки ПЗ. Мікросервісна архітектура не є виключенням, оскільки вона має досить багато особливостей, які треба брати до уваги не тільки при створенні такого ПЗ, а й при збірці та доставці [2].

Першим постає питання про збереження програмного коду, який необхідно використовувати при збірці та доставці. Це питання може мати декілька рішень, оскільки необхідно вирішити, яку систему контролю версій використовувати для збереження всього програмного коду. Як тільки це питання вирішується, необхідно визначати середу розгортання, яка буде відповідати фінальному результату доставки ПЗ. При вирішення питання щодо середи розгортання часто виникають питання щодо локації розгортання, мережевих налаштувань, типу застосунку тощо. Це досить важливі аспекти при доставці програмного коду кінцевому користувачу, оскільки всі налаштування середи спряють безпечному використанню кінцевого корпоративного рішення. Як тільки питання середи розгортання вирішено, необхідно перейти до технологій, які будуть направлені на оптимізацію процесу доставки ПЗ. Зазвичай, вони мають всі необхідні інструменти для створення деяких сценаріїв, які виконуються за деяких умов. Такі сценарії необхідні для того, щоб почати процес доставки ПЗ до середи розгортання. Після виконання сценаріїв розгортання програмний код

повинен потрапити до середи, де він вже готовий до використання. Звісно, після доставки необхідно провести тестування та моніторинг системи на правильність виконання поставлених задач. Загальний процес доставки ПЗ на основі одного мікросервісу зображено на рисунку 1.1 [3].

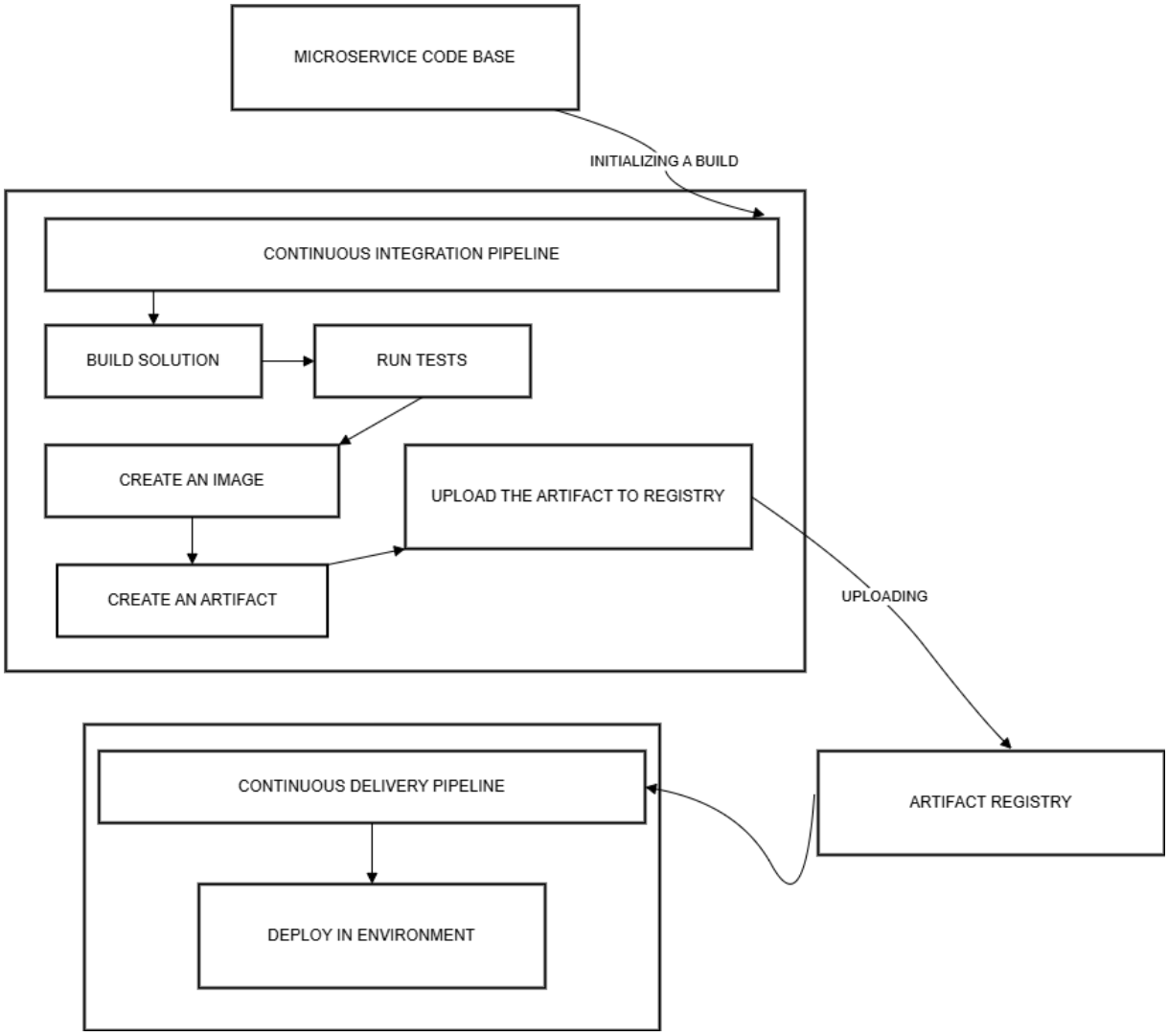

Рисунок 1.1 – Процес розгортання мікросервісу

Отже, при роботі з будь-якою архітектурою необхідно мати певний алгоритм, який відповідає за процес оптимізації збірки та доставки ПЗ. Цей алгоритм може бути різний для кожного застосунку, які треба брати до уваги при створенні моделі розгортання цього застосунку.

1.1.2 Класифікація процесів розгортання мікросервісів

На сьогодні існує чимало різних моделей розгортання ПЗ, яке побудоване, використовуючи мікросервісну архітектуру, кожна з яких має свої переваги та недоліки, саме тому необхідно дослідити та знайти найбільш оптимальну модель, яка вирішує поставлену задачу з використання сучасних технологій.

Найбільш відомий процес розгортання – це «Manual Deployment», або модель мануального розгортання. Навіть сьогодні існують корпоративні рішення, які підтримують цю модель розгортання. Зазвичай, це досить старе ПЗ, яке було створено при відсутності технологій, які б дозволяли оптимізувати процес розгортання. Ця модель передбачає наявність спеціалістів, які зможуть розгортати ПЗ власноруч. Цей підхід не є оптимальним, оскільки існує великий ризик помилки при перенесенні змін у програмному коді до середи розгортання, оскільки це, зазвичай, ручне копіювання файлів, виконання скриптів тощо. При збільшення застосунку цей процес стає тільки складнішим, тому така модель розгортання не є популярною серед IT-спільноти [4].

Більш сучасним процесом розгортання, який використовує чимало спеціалістів, є процес під назвою «Scripted Deployment», який частково відсікає мануальну роботу при розгортанні програмного застосунку. Весь процес розгортання відбувається за допомогою скриптів, які можна написати та виконувати в процесі розгортання ПЗ. Завдяки написаним скриптам відбувається процес автоматизації розгортання, що сприяє зниженню помилок, які може зробити спеціаліст, при мануальному розгортанні [5].

Найсучаснішим процесом розгортання є «Automated Deployment», який вже повністю виключає мануальну роботу спеціаліста, адже весь процес розгортання відбувається за рахунок обчислювальних можливостей

комп’ютера. Такий підхід забезпечує швидкість і повну автоматизацію процесу розгортання програмного коду. На даний момент реалізація такого процесу розгортання можлива за допомогою різних технологій таких, як конвеєри, що, зазвичай, потребує знання DevOps [6]

Порівняння процесів розгортання мікросервісів представлено у таблиці 1.1.

Таблиця 1.1 – Порівняння процесів розгортання мікросервісів

Назва процесу	Переваги	Недоліки	Ключова різниця
Manual Deployment	Простота, контроль	Високий ризик помилок, важко масштабувати	Все виконується вручну, без будь-якої автоматизації
Scripted Deployment	Менше помилок, повторюваність	Потрібна підтримка скриптів	Часткова автоматизація за допомогою скриптів
Automated Deployment	Повна автоматизація, швидкість	Потрібне налаштування, знання DevOps	Повністю автоматизований процес

1.2 Аналіз моделей розгортання мікросервісів

На даний час мікросервісна архітектура займає ключову позицію, оскільки забезпечує надійність системи, модульність, еластичність і масштабованість. Еволюція хмарних технологій надає широкий спектр можливостей для використання архітектури мікросервісів, оскільки хмара дозволяє створювати різноманітні середовища з різною конфігурацією

інфраструктури. Хмарні сервіси пропонують широкий спектр функцій, які можна і потрібно використовувати для створення справді надійних і універсальних корпоративних систем з використанням мікросервісної архітектури.

Сучасні моделі, які дозволяють оптимізувати процес розгортання, є частиною індустрії розробки ПЗ, оскільки вони дозволяють зменшити витрати, одночасно підвищуючи надійність і доступність системи в цілому.

Монолітне розгортання є однією з стандартних моделей розгортання ПЗ. У цій моделі розгортання вся система розгортається як єдине ціле, тобто, всі компоненти системи повинні походити з однієї кодової бази і, що більш важливо, вони є інтегрованим процесом. Однією з найвагоміших переваг монолітної моделі розгортання є те, що її легко як розробляти, так і розгортати, оскільки вся кодова база працює в одній структурі. Це дозволяє зменшити складність налаштування механізмів взаємодії між сервісами, що часто необхідно в архітектурі мікросервісів.

Оскільки всі компоненти системи виконуються в одному процесі, між компонентами системи немає витрат на мережевий запит, що зменшує накладні витрати на обмін даними. А при роботі з такою архітектурою немає необхідності використовувати велику кількість серверів, оскільки вся система працює як одне ціле, знижуючи вартість інфраструктури та уникаючи необхідності запровадження складних процесів Continuous Integration (CI) і Continuous Delivery (CD) [5].

Звичайно, ця модель має низку істотних недоліків, таких як проблема масштабування, оскільки всі елементи повинні бути масштабовані разом, навіть якщо потрібно змінити лише одну частину системи. Другою проблемою з монолітом є проблеми з обслуговуванням і оновленням, оскільки будь-яка зміна в коді повинна включати всю систему, що реконструюється, і будь-яка проблема може призвести до збою всієї системи. Слід мати на увазі, що під час оновлення необхідно перезапустити всю систему, що призведе до простою.

Модель почергового розгортання припускає, що нова версія системи повільно переходить на зміну попередній, почергово оновлюючи кожен конкретний екземпляр. Цей підхід ефективно запобігає простою системи, підвищує загальну високу доступність системи та мінімізує пов’язані з цим ризики. Стратегія окреслює ряд етапів, які є вирішальними для розгортання цієї моделі.

Початковий стан визначає поточну версію системи, другий стан визначає початок процесу оновлення, який розгортається на заданій кількості серверів, а решта серверів продовжують працювати, таким чином забезпечуючи доступність системи. Третій стан визначає фазу перевірки, на якій можна спостерігати за роботою серверів, які мають оновлену версію системи. Якщо перевірка пройдена, оновлюється наступна частина серверів. Після перевірки виконується повне розгортання – четвертий крок, який передбачає, що нова версія системи повністю замінює стару. Після успішного повного розгортання часто виконується повний моніторинг функціонування системи [7].

Переваги цієї моделі розгортання полягають у надзвичайно високій доступності системи, оскільки систему не потрібно переводити в автономний режим. Якщо в новій версії є якісь помилки, її можна повернути до версії безпосередньо перед повним розгортанням, тому проблеми з новою версією можуть вплинути лише на обмежену групу користувачів.

Схема моделі почергового розгортання дійсно має кілька недоліків, одним із найважливіших з яких є велика тривалість розгортання. Може виникнути багато проблем із сумісністю між новою та існуючою версіями, оскільки необхідно передбачити співіснування обох версій без будь-яких конфліктів. Найчастіше проблема може виникнути під час роботи з базою даних, оскільки вона повинна підтримувати обидві версії.

Загалом, така модель використовується в ситуаціях, коли система повинна працювати безперервно, а саме оновлення не вимагає радикальних змін. Досить часто цей підхід використовується з хмарними системами,

керованими Kubernetes.

Модель інкрементального розгортання – це стратегія розгортання, у якій нова версія системи доступна лише деяким підмножинам користувачів системи, а після повної перевірки та тестування вона поетапно випускається для всіх користувачів. Модель також визначає деякі фази розгортання, які необхідно пройти, щоб система була доступна для користувачів [7].

Початковий етап – це найперший, де всі запити обробляються ще старою версією системи. Другий етап – це розгортання нової версії, яка працює паралельно, але здатна обробляти певну кількість запитів. Третій етап – це моніторинг, де ретельно вивчаються показники продуктивності та відгуки користувачів. Після успішного етапу моніторингу виконується етап масштабування, під час якого більшість трафіку направляється до нової версії системи. Останнім етапом є повне оновлення, яке передбачає повну стабільність оновленої системи, де нова версія повністю замінює стару.

Якщо нова версія системи має проблеми, вони вплинуть лише на невелику частину користувачів системи, що дає змогу оцінити, як впливають зміни на продуктивність системи – одна з переваг такої моделі. Така модель повністю гнучка у масштабуванні, оскільки можна плавно перенаправляти трафік до різних версій однієї системи.

Одним із найбільших недоліків цієї моделі є те, що вона ускладнює процес розгортання, оскільки включає механізми для балансування Інтернет-трафіку. Ця модель також викликає проблеми сумісності під час розгортання нової версії, оскільки обидві версії повинні підтримуватися.

Модель інкрементального розгортання широко використовується в ситуаціях, коли є потреба зменшити ризики або часте розгортання нових версій системи. Модель дозволяє поступово вносити зміни, таким чином мінімізуючи ризики та дозволяючи швидко скасувати зміни у разі виникнення проблем. Незважаючи на складність реалізації, гнучкість моделі робить її дуже придатною для проєктів, які часто оновлюються та мають велику кількість користувачів.

Модель паралельного розгортання з перемиканням мережевого трафіку – це безперервний підхід до розгортання, у якому дві версії системи працюють паралельно, і користувачі можуть негайно переходити від однієї версії до іншої.

Перше середовище – це існуюча стабільна версія системи, яка обробляє весь трафік, який їй надсилається. Друге середовище – це нова версія системи, яка тестується перед тим, як увесь трафік буде направлено до нової версії системи [8].

Початкова фаза моделі – це розгортання нової версії, де нова система розгортається в цьому середовищі, тоді як попереднє середовище з наявною версією продовжує працювати. Другим кроком цього шаблону є тестування нового середовища, де можна виконувати автоматичне тестування разом із спостереженням за роботою системи та її ефективністю. Якщо припустити, що всі системи працюють за планом, другий етап передбачає маршрутизацію трафіку для задоволення нових запитів, при цьому стара версія служить резервною копією на випадок повернення. Якщо нова система стабільна, середовище старої версії може бути виведено з експлуатації.

Переваги моделі включають скорочення часу простою, оскільки споживачі ізольовані від процесу розгортання. У разі виникнення проблеми з новим випуском модель дозволяє миттєво розгортати систему до попереднього стану. Модель дозволяє легко контролювати версії та безпечно їх розгортати, дозволяючи тестувати нову версію на реальних даних без ризику для всієї групи користувачів, що є значним покращенням порівняно з іншими моделями розгортання.

Недоліки включають складну конфігурацію розгортання та високі витрати на інфраструктуру, оскільки два середовища повинні працювати одночасно.

Таким чином, модель паралельного розгортання з перемиканням мережевого трафіку є дуже ефективним підходом до оновлення систем без

простоїв і ризиків. Вона особливо добре підходить для критично важливих систем, де необхідно підтримувати безперервність бізнесу, однак для цього потрібні додаткові ресурси. Якщо компанія готова інвестувати в ці процеси, ця техніка розгортання є однією з найбільш надійних для впровадження.

Модель декларативного розгортання з використанням GitOps – це декларативний підхід до автоматизації розгортання ПЗ, де Git є єдиним джерелом для керування всім середовищем. Ця модель пропонує декларативний опис, яка охоплює розробку незалежних файлів конфігурації з метою вдосконалення різних процесів. Конфігурації зберігаються в централізованому сховищі, що дозволяє об’єднати весь робочий процес автоматизації. Завдання конфігурації варіюються від простих завдань, таких як фіксація коду та виконання тестів, до складних операцій, пов’язаних із поданням звітів групі керування [9].

Перевагами є прозорість і контроль, оскільки всі зміни керуються версіями за допомогою Git. Це сприяє стабільності, оскільки система постійно перевіряє, чи фактичний стан відповідає даним, які зберігаються в Git. Налаштування надають численні можливості для створення складних централізованих сховищ різного характеру, що підвищує безпеку, оскільки репозиторій може керувати доступом до всіх ресурсів. Важливою перевагою є те, що Git можна інтегрувати з багатьма різними технологіями, які можуть покращити процес розгортання ПЗ, що робить цю модель дуже гнучкою.

Складність конфігурації є вагомим недоліком саме тому, що потрібно добре налаштувати репозиторії й всі оператори. Саме цей недолік сприяє додатковим витратам.

Порівняння моделей розгортання наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння сучасних моделей розгортання

Модель	Переваги	Недоліки	Особливості
1	2	3	4
Монолітне розгортання	Простота розробки та розгортання, одна кодова база, низькі витрати на інфраструктуру, менше складнощів з CI/CD	Складність масштабування, високий ризик збоїв через єдину точку, весь застосунок треба перезапускати, важке обслуговування	Єдиний блок коду, оновлюється весь додаток одразу
Почергове розгортання	Висока доступність системи, безпечне оновлення без повного зупинення, легка відмова у разі проблем, підходить для Kubernetes	Тривалий час розгортання, складність сумісності між версіями, можливі конфлікти при спільному використанні ресурсів	Версії працюють паралельно в Kubernetes або іншому кластері
Інкрементальне розгортання	Мінімізація ризиків, поступовий контроль якості, можна швидко відкотитися, добре підходить для частих релізів	Ускладнене балансування трафіку, необхідність підтримки двох версій, можливі проблеми сумісності	Версія оновлюється поступово: частинами або групами користувачів
Модель паралельного розгортання з перемиканням мережевого трафіку	Мінімальний час простою, швидке перемикання між версіями, легкий відкат, тестування на реальних даних без впливу на всіх користувачів	Подвоєні інфраструктурні витрати, складна конфігурація, не підходить для часто оновлюваних систем, якщо ресурси обмежені	Одночасно існують дві ідентичні версії, між якими перемикається трафік

Кінець таблиці 1.2

1	2	3	4
Модель декларативного розгортання з використанням GitOps	Прозорість і контроль завдяки Git, постійна перевірка відповідності стану, висока безпека і гнучкість, інтеграція з різними технологіями	Складна конфігурація репозиторіїв	Інфраструктура керується через Git як єдине джерело істини

Отже, базуючись на результатах порівняння моделей розгортання, можна зробити висновок, що звичайні моделі розгортання хоча і мають певні переваги, але присутні й суттєві недоліки, які можуть вплинути на оптимізацію процесу розгортання. Рішення такої проблеми є створення комбінованої моделі розгортання, яка буде мати переваги декількох моделей, доповнюючи можливості одна одної, що спричиняю вирішенню суттєвих недоліків.

1.3 Аналіз задачі валідації фармацевтичних документів

Сьогодні вже існує декілька стандартів, які можуть бути використані для валідації фармацевтичних документів. Кожен з таких стандартів має власні недоліки та переваги, які повинні бути враховані при створенні результуючого рішення. Оскілки подання фармацевтичних документів визначено вищім регуляторним органом, необхідно, щоб кожна специфікація враховувала всі аспекти подання. Якщо будь-який стандарт не враховує всі вимоги регуляторного органу, він не може бути використаний для створення результуючого рішення.

Common Technical Document (CTD) – це технічна специфікація для гармонізації, яка полегшує подання різноманітної технічної документації на лікарські засоби до регуляторних органів. Стандарт був розроблений в рамках процесу Міжнародної конференції з гармонізації технічних вимог до реєстрації лікарських препаратів для використання людиною та опублікований у 2002 р. Розробка технічної документації мала на меті полегшити подання та забезпечити покращений зв'язок між регуляторними органами та фармацевтичними компаніями в різних країнах, що гарантує уніфікований та послідовний процес подання. Стандарт об'єднує всі документи, які фармацевтичні компанії повинні подавати для схвалення нових ліків, шляхом узгодження технічної інформації в єдиний формат, спрощуючи процес подання до багатьох регуляторних органів. Зазначені цілі реалізуються шляхом визначення чітких компонентів [10].

Частина перша визначає адміністративну інформацію та дані про призначення лікарського засобу. У розділі міститься докладний опис продукту, інформація про компанію, яка заявляє про його виробництво, а також клінічні рекомендації щодо показань і протипоказань, дозування та можливих побічних ефектів. Основна концепція цього розділу – це надання інструкцій для пацієнтів із маркуванням ліків .

Друга частина містить дані щодо опису та детального складу лікарського засобу. Необхідно описати стислу інформацію про хімічну структуру діючої лікарської речовини, детально описати виробничі процеси, які використовуються для виготовлення продукту, надати докази стабільності препарату, а також встановити та оголосити стандарти якості та оцінки на місцях виробництва. Іншим важливим моментом є обмін інформацією щодо контролю якості на кожному етапі виробництва.

Третя частина вимагає детальних звітів про доклінічні дослідження, а саме: токсикологічні оцінки, дослідження на тваринних моделях, фармакологічні властивості продукту, а також дані щодо мутагенності, канцерогенності та репродуктивної токсичності.

Четверта частина визначає всі дані з клінічних досліджень лікарського засобу. Вона включає в себе результати клінічних випробувань, проведених на людях. Важливо включити дані щодо безпеки та ефективності продукту, а також результати тестування на різних групах пацієнтів з урахуванням демографічних характеристик.

П’ята і остання частина містить допоміжну інформацію щодо фармакокінетики, фармакодинаміки та біоеквівалентності. Цей розділ повинен додати більше інформації щодо всмоктування, розподілу, метаболізму та виведення препарату. Крім того, необхідно надати інформацію про оцінку взаємодії лікарських засобів, щоб мати можливість моделювати дію препарату в організмі. Стандарти подання зазначені на рисунку 1.2 [10].

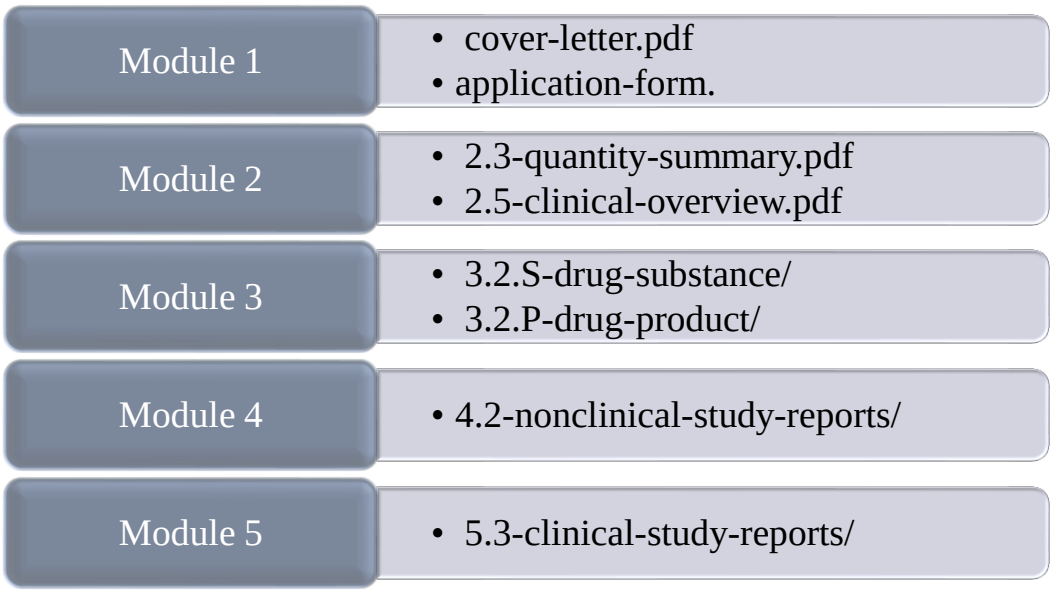

Рисунок 1.2 – Структура подання CTD

Таким чином, CTD дозволяє гармонізувати документацію шляхом створення єдиного стандарту для подання документації на фармацевтичну продукцію. Стандарт спрощує процес реєстрації ліків, надаючи чітко визначений формат документації, якого повинні дотримуватися всі

фармацевтичні компанії. Поєднання цих двох аспектів сприяє гармонізації між регуляторними органами та фармацевтичними компаніями, які прагнуть продавати свої ліки в Сполучених Штатах, Європейському Союзі чи Японії. Загалом цей документ виконує ключову функцію у стандартизації процесу реєстрації фармацевтичних продуктів, що значно полегшує вихід на нові ринки. Недолік такого підходу полягає в тому, що даний стандарт не глобалізує пункти подання та не має чітких вказівок щодо електронного формату. Структура CTD представлена на рисунку 1.3 [10].

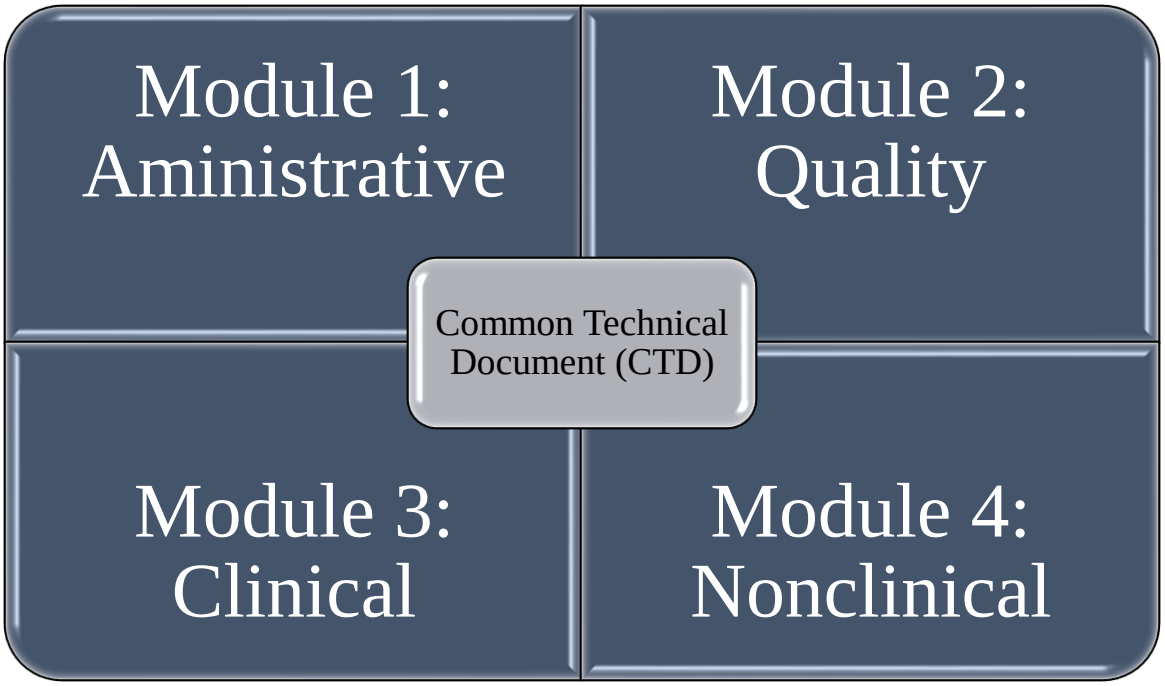

Рисунок 1.3 – Структура CTD

Structured Common Technical Document (SCTD) – це вдосконалена та стандартизована форма загального технічного документа, розробленого з єдиною метою полегшення подання документів та реєстрації лікарських засобів у різних країнах. Мінімальна вимога цього документа полягає в тому, щоб він був спрямований на гармонізацію процедур, необхідних для подання до різних регуляторних органів [11].

У той час як CTD є міжнародним стандартом для подання технічних

фармацевтичних документів для Сполучених Штатів, Європейський Союз і Японія використовують, SCTD – це особливий варіант оригінального стандарту, результат міжнародної співпраці для подальшої гармонізації технічної документації та процедур реєстрації ліків у країнах, які не завжди дотримуються міжнародних стандартів.

Основні характеристики включають сумісність і гнучкість, оскільки SCTD гарантує відповідність усім світовим стандартам, одночасно спрощуючи процедуру модифікації технічної документації для подання до регуляторних органів, які можуть мати власні унікальні вимоги. Це, у свою чергу, дає змогу фармацевтичним компаніям подавати заявки з обмеженими змінами до документації на фармацевтичні продукти, таким чином, забезпечуючи її зручність і доступність для глобального використання. Важливим аспектом є уніфікація тестування, яку забезпечує SCTD, адже це дозволяє спростити процес перевірки нових препаратів.

Загалом, SCTD є дуже важливим винаходом у розвитку світової фармацевтичної промисловості, оскільки специфікація додатково стандартизує та спростить процес реєстрації ліків у різних країнах. Завдяки цьому технічному документу фармацевтичні компанії зможуть зменшити потребу у значних витратах, а процеси управління між підприємствами та регуляторними органами стануть більш ефективними. Головним недоліком є неможливість використати електронний формат фармацевтичних документів. Структура SCTD представлена на рисунку 1.4 [11].

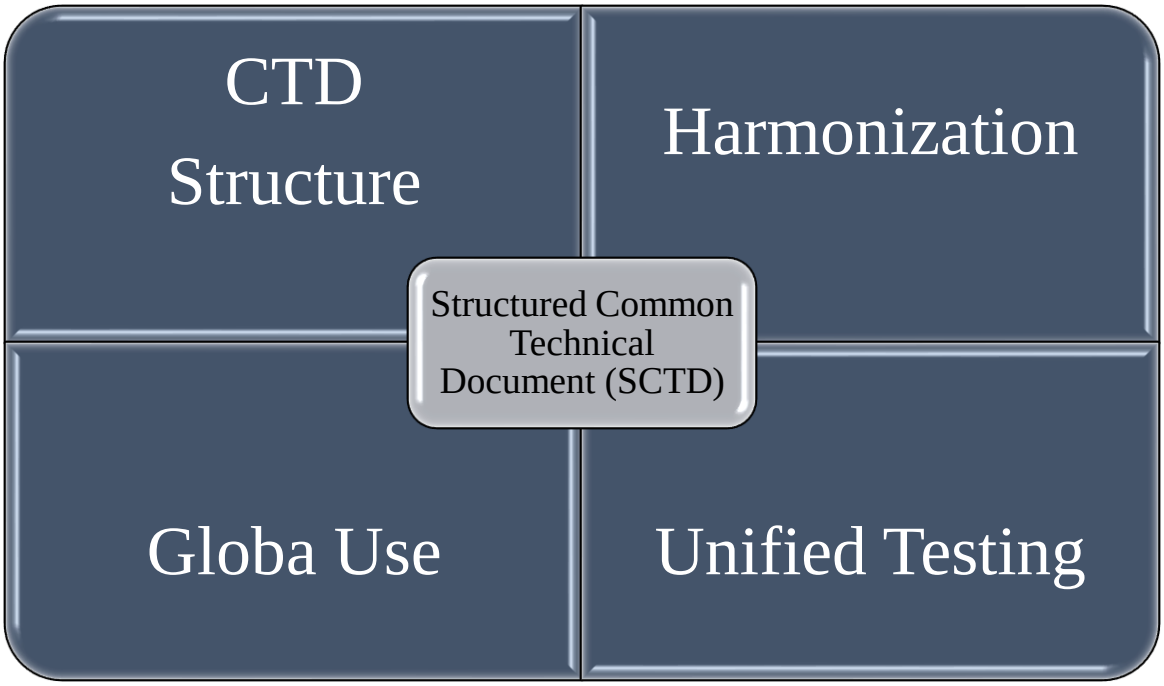

Рисунок 1.4 – Структура SCTD

Electronic Common Technical Document (ECTD) представляє собою електронну версію CTD. Це новий стандарт для подання технічної документації при реєстрації лікарських засобів до регуляторних органів. Дана специфікація переводить стандартизацію та оптимізацію подання технічної документації при реєстрації лікарських засобів на новий рівень [12].

Основна характеристика цієї специфікації полягає в заміні всіх паперових документів електронною версією, що значно покращує подання, зберігання та подальшу обробку документів. Ця зміна дозволяє регуляторним органам ефективніше спілкуватися з фармацевтичними компаніями. Використання ECTD сприяє інтернаціоналізації процесів реєстрації лікарських засобів, підвищує доступність інформації та прозорість, а також прискорює обробку заявок. ECTD використовує ту саму структуру частин, що й оригінальний CTD, але вони подаються в електронному вигляді. Структура подання ECTD зазначена на рисунку 1.5 [12].

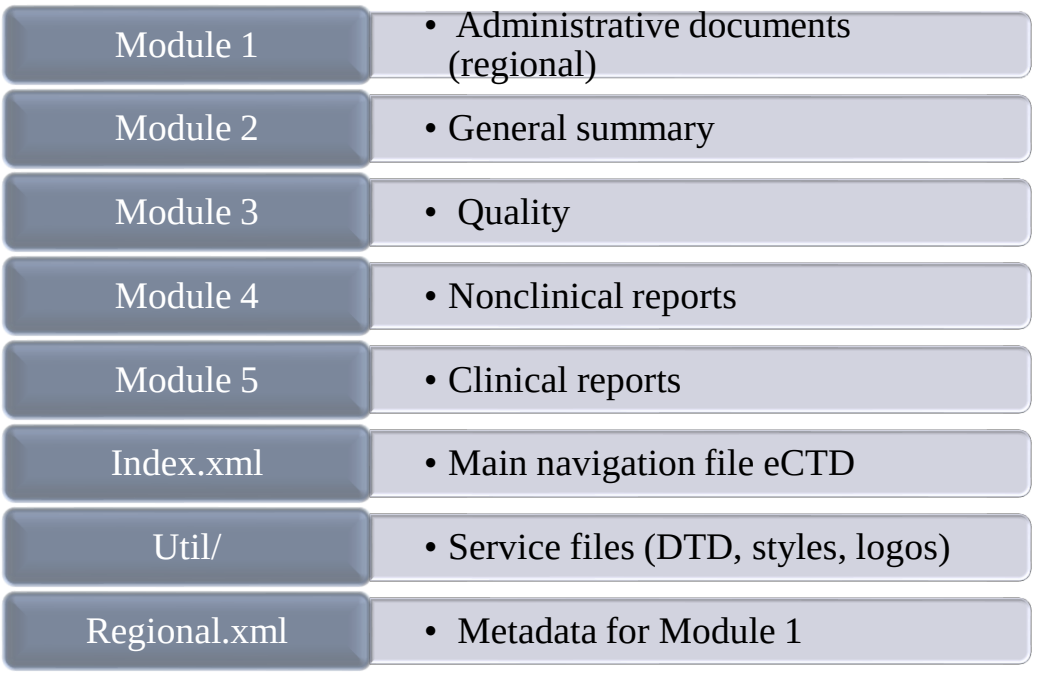

Рисунок 1.5 – Структура подання ECTD

Перевагами ECTD є ефективність і швидкість обробки, оскільки ECTD є електронним представленням документації, контролюючі органи можуть обробляти подані документи набагато швидше. Формат дозволяє автоматично перевіряти документ на відповідність стандартам і досить легко визначати наявність помилок або відсутню інформацію. А відсутність необхідності обробляти величезну кількість паперових документів полегшує навантаження на контролюючих органів.

Розробка такої специфікації дозволяє фармацевтичним компаніям значно заощадити, оскільки вся документація зберігається в електронному вигляді, що значно спрощує процес оновлення та подання додаткових матеріалів.

ECTD підтримує глобальну сумісність, що є значної перевагою оскільки, специфікація підтримується більшістю великих регуляторних органів, таких як США, країнами Європейського Союзу, Японією, Канадою та Австралією. Це дає можливість подавати документи до різних організацій без необхідності змінювати формат.

Відповідно, стандартизована технічна специфікація, яка використовує електронне подання, служить важливим інструментом у процесі глобалізації та стандартизації фармацевтичної реєстрації. Така техніка підвищує ефективність фармацевтичних фірм, а також органів, що регулюють лікарські засоби, за рахунок підвищення ефективності, зменшення витрат і прискореного подання та розгляду заявок на нові ліки. Основним недоліком цього підходу є складність реалізації програмними засобами. Структура ECTD представлена на рисунку 1.6 [12].

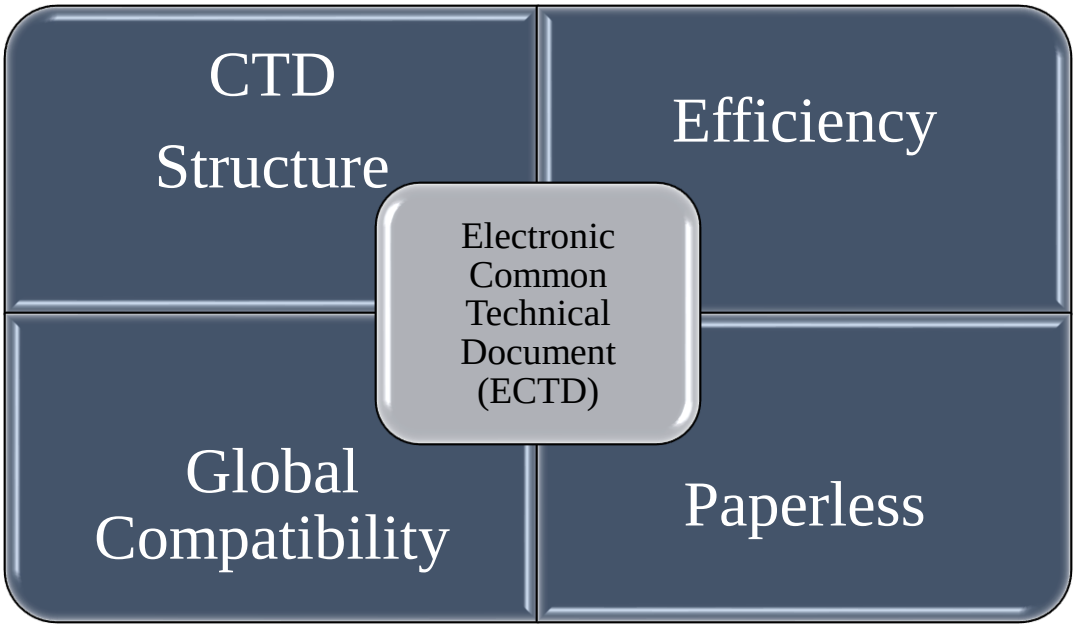

Рисунок 1.6 – Структура ECTD

Отже, CTD, SCTD та ECTD мають чимало спільних характеристик, але технології кардинально відрізняються коли мова йде про переваги та недоліки. Детальне порівняння технологій наведено у таблиці 1.3.

Таблиця 1.3 – Порівняння CTD, SCTD та ECTD

Критерій	CTD	SCTD	ECTD
1	2	3	4
Формат	Portable Document Format (PDF), паперовий	PDF та Extensible Markup Language (XML)	XML або JavaScript Object Notation (JSON)
Подання	Поштова	Через регуляторні портали	Автоматизована
Навігація	Ручна	XML-індексація	Повністю структурована
Підтримка версій	Ручне	Послідовності	Вбудоване відстеження змін у даних
Зручність для регуляторів	Низька	Середня	Висока, оптимізована для автоматичного аналізу
Машиночитаність	Немає	Часткова (структура)	Повна (структура та вміст машиночитані)
Можливість повторного використання даних	Обмежена	Часткова	Повна

Кінець таблиці 1.3

1	2	3	4
Міжнародна підтримка	Широка	Дуже поширена	У розробці, підтримка зростає
Автоматизована обробка	Немає	Обмежена	Підтримує інтеграцію з IT-системами
Складність впровадження	Найпростіше	Середня	Найскладніше
Загальні витрати на впровадження	Низькі	Середні	Високі (але виправдані для великих компаній)
Автоматична валідація	Немає	Часткова (через валідатори)	Повна, машиночитана

1.4 Постановка задачі

Актуальність дослідження полягає в необхідності вдосконалення існуючих моделей розгортання IT-проєктів з використанням мікросервісної архітектури внаслідок таких ключових недоліків існуючих підходів, як складність управління інфраструктурою при наявності великої кількості мікросервісів, складність масштабування, високий ризик збоїв, тривалий час розгортання та ускладнене балансування мережевого трафіку. Зокрема, у сфері фармацевтичної комерції де характерними є такі недоліки, як відсутність інтеграцій з різними медичними системами, високий рівень вимог до безпеки зберігання або передачі медичних даних та складність забезпечення відповідності нормативним вимогам. Однак у випадку

мікросервісної архітектури, процес розгортання повинен враховувати сучасні технічні вимоги та інтеграції з іншими технологіями. Це обумовлює потребу в створенні комбінованої моделі, яка поєднує переваги кількох підходів з урахуванням специфіки мікросервісної архітектури та галузевих вимог. Такий комбінований підхід дозволить забезпечити високу надійність, адаптивність до змін і зниження ризиків під час оновлення системи.

Об’єктом дослідження кваліфікаційної роботи є процес розгортання мікоросервісів в задачах валідації електронних фармацевтичних документів.

Метою даної роботи є дослідження процесу розгортання мікоросервісів в ІТ-проєктах валідації електронних фармацевтичних документів для підвищення ефективності доставки ПЗ.

Для досягнення мети магістерської роботи необхідно вирішити такі задачі дослідження:

– аналіз процесів розгортання мікросервісів;

– дослідження структури процесу розгортання мікросервісів;

– аналіз моделей розгортання мікросервісів;

– аналіз задачі валідації фармацевтичних документів;

– удосконалення моделі розгортання мікросерівсів з використанням комбінованого підходу розгортання ПЗ для валідації електронних фармацевтичних документів;

– розробка системи для валідації електроних фармацевтичних документів;

– розробка комбінованої моделі розгортання мікросервісів;

– експериментальна перевірка отриманої удосконаленого моделі.

2 КОМБІНОВАНА МОДЕЛЬ ПРОЦЕСУ РОЗГОРТАННЯ МІКРОСЕРВІСІВ

2.1 Модель паралельного розгортання з перемиканням мережевого трафіку

Модель паралельного розгортання з перемиканням мережевого трафіку надає досить багато можливостей для реалізації гнучкого, масштабованого, а саме головне – надійного розгортання без простоїв, які досить можуть виникати при розгортанні ПЗ.

Дана модель, насамперед, передбачає наявність двох повністю ізольованих середовищ для розгортання ПЗ. Середовища можуть бути як і прості з використанням різних хмарних технологій, які постачаються як послуги, так і досить складними з використання хмарної інфраструктури. Склад середовища, насамперед, визначається необхідними технологіями для роботи системи. До складу можуть входити віртуальні машині, віртуальні мережі, шини повідомлень, сховища об’єктів тощо. Перше середовище, яке іноді називають «зеленим» середовищем, представляє ізольоване середовище, де буде розгорнуто нову версію ПЗ. Друге середовище, які іноді називають «блакитним» середовищем, представляє собою активну інфраструктуру, які працюють в поточний момент, тобто, обробляють запроси користувачів [13].

Іншим, але досить важливим, компонентом моделі є система маршрутизації або регулювальник мережевого навантаження. Цей окремий елемент моделі забезпечує миттєве перемикання мережевого трафіку між двома ізольованими середовищами. Даний компонент надає можливість перемикати мережевий трафік після того, як нове середовище повністю буде відповідати стандартам, що забезпечують коректну роботу сервісу в цьому середовищі.

Система моніторингу та логування – це ще один компонент моделі

паралельного розгортання з перемиканням мережевого трафіку. Система моніторингу надає можливість в реальному часі відстежити внутрішню роботу обчислювальних ресурсів, на яких працює сервіс, що дозволяє знаходити різноманітні помилки на рівні інфраструктури. Цей елемент може використовуватися не тільки для відстеження реальних показників інфраструктури, але й для повного моніторингу сервісу, що дає можливість відстежити різни помилки на рівні ПЗ [14]

Останній, але ключовий елемент, моделі розгортання представляє собою CI/CD-конвеєр, який забезпечує автоматизацію процесу розгортання ПЗ. Цей інструмент дозволяє декларативно створити конвеєри, які займаються збором і розгортанням.

Усі компоненти моделі паралельного розгортання з перемиканням мережевого трафіку зображені на рисунку 2.1.

Рисунок 2.1 – Компоненти моделі паралельного розгортання з перемиканням мережевого трафіку

Для глибшого аналізу моделі паралельного розгортання з перемиканням мережевого трафіку доцільно подати її формалізованому вигляді (2.1), який описує основні компоненти системи, їх взаємозв'язки та функціональну взаємодію між собою:

$$M = \{E_1, E_2, T, O, P\}, \quad (2.1)$$

де M – модель розгортангня;

E_1 – поточне середовище;

E_2 – нове середовище з оновленою версією ПЗ;

R – механізм маршрутизації трафіку між середовищами;

T – правило або логіка перемикання трафіку між середовищами;

O – система моніторингу для оцінки стану;

P – CI/CD-конвеєр.

Загальний процес розгортання можна представити у вигляді формалізованого виразу (2.2):

$$M = E_1 \rightarrow P \rightarrow O(E_2), \quad (2.2)$$

де M – модель розгортангня;

E_1 – поточне середовище;

E_2 – нове середовище з оновленою версією ПЗ;

O – система моніторингу для оцінки стану;

P – CI/CD-конвеєр.

У разі успішного завершення розгортання нової версії ПЗ та відсутності помилок при тестуванні, система переходить до наступного етапу – перемикання трафіку на оновлене середовище. Формалізований процес виглядає:

$$O(E_2) = ready => T \rightarrow E_2,$$

де O – система моніторингу для оцінки стану;

E_2 – нове середовище з оновленою версією ПЗ;

T – правило або логіка перемикання трафіку між середовищами.

Оскільки безпечне оновлення системи, критично важливим є механізм повернення на стабільну версію у випадку виникнення помилок у новому середовищі. Формалізований процес можна описати за формулою (2.3):

$$O(E_2) = error => T \rightarrow E_1, \quad (2.3)$$

де O – система моніторингу для оцінки стану;

E_1 – поточне середовище;

T – правило або логіка перемикання трафіку між середовищами;

Основними проблемами при розгортанні ПЗ з мікросервісною архітектурою можуть бути наступні проблеми:

– простої системи при розгортанні ПЗ;

– недостатня надійність та контроль релізів;

– неможливість моментального відновлення після оновлення ПЗ;

Модель паралельного розгортання з перемиканням мережевого трафіку завдяки своїм компонентам та технології перемикання трафіку вирішує ці проблеми. Усі компоненти, працюючи як єдине ціле, надають певні особливості, як слід брати до уваги, при використанні моделі. Першою особливістю моделі є атомарне перемикання мережевого трафіку, яке дозволяє перенаправити трафік практично миттєво. Другою особливістю є можливість поєднання з різними моделями розгортання, що може надати більш гнучке та масштабоване рішення для управління розгортанням ПЗ з використанням мікросервісної архітектури [15].

До недоліків моделі паралельного розгортання з перемиканням мережевого трафіку можна віднести високе споживання ресурсів через використання двох ізольованих середовищ, що може спричинити додаткові витрати на підтримку інфраструктури. Управління автоматизацію або

скриптів, які використовує конвеєр автоматизації, також відноситься до недоліків даної моделі розгортання.

Виходячи з недоліку про додаткове споживання ресурсів, які потенційно можуть створити додаткові витрати на підтримку двох середовищ одночасно, можна зробити висновок щодо обмежень використання цієї моделі розгортання. Головним обмеження є те, що ця модель не підходить для систем із дуже великим розміром, оскільки підтримка інфраструктури великого розміру це досить складна задача.

Загалом, модель паралельного розгортання з перемиканням мережевого трафіку надає значно більше переваг, чим недоліків, а можливості вирішувати основні проблеми при розгортанні ПЗ з мінімальними обмеженнями роблять цю модель однією з найпривабливіших для використання в сучасних DevOps-практиках.

2.2 Модель декларативного розгортання з використанням GitOps

Модель декларативного розгортання з використанням GitOps – це повністю сучасна модель розгортання ПЗ, в основі якої лежить декларативній підхід до опису розгортання ПЗ. Усі описи знаходяться в репозиторії Git, що виступає єдиним джерелом правди в цій моделі. На даному етапі вся конфігурація, параметри налаштування фізичних серверів описується у вигляді конфігураційних файлів, які повинні зберігатися в репозиторії Git. Після того, як вся конфігурація визначена та збережена в репозиторії, агент-оператор, що слідкує за всіма змінами в репозиторії, автоматично виконує розгортання шляхом синхронізації поточного стану системи з станом у Git [16]

Дана модель складається з декількох важливих компонентів, які виконують свої певні обов’язки. Першим, але фундаментальним для цієї

моделі, компонентом є репозиторій Git. Цей елемент представляє звичайний репозиторій, який працює за правилами Git. Головною задачею цього компонента є збереження декларативних файлів конфігурації, які використовуються для визначення як інфраструктури, так і ПЗ. Репозиторій Git представлений як єдине джерело правди при використанні цієї моделі.

Наступним компонентом є агент-оператор, який виконує перевірку репозиторії Git для порівняння поточного стану середовища із новими змінами в репозиторії. Якщо такі зміни знайдені, то починається новий процес розгортання. Цей компонент працює з іншим важливим елементом моделі – CI/CD-конвеєр, який виконує автоматизацію процесу розгортання.

Останнім компонентом моделі є середовище розгортання, яке представляє собою звичайну інфраструктуру де працює ПЗ. Усі компоненти декларативної моделі розгортання з використанням GitOps зображені на рисунку 2.2 [17].

Рисунок 2.2 – Компоненти моделі декларативного розгортання з використанням GitOps

З метою чіткішого формального подання архітектури та логіки GitOps-підходу, модель можна описати у формалізованому вигляді (2.4):

$$M = \{R, S, A, E, P\}, \tag{2.4}$$

де M – модель розгортання;

R – репозиторій, містить декларативні файли конфігурацій;

S – стан системи, якого має бути досягнуто в середовищі виконання;

A – агент-оператор, який відтворює всі зміни у середовищі;

E – середовище розгортання;

P – CI/CD-конвеєр, який виконує автоматизацію процесу розгортання ПЗ.

Загальний процес розгортання за допомогою моделі декларативного розгортання з використанням GitOps можна представити у вигляді формалізованого виразу:

$$M = R \ \rightarrow A(S) \rightarrow P \ \rightarrow E,$$

де M – модель розгортання;

R – репозиторій, містить декларативні файли конфігурацій;

S – стан системи, якого має бути досягнуто в середовищі виконання;

A – агент-оператор, який відтворює всі зміни у середовищі;

E – середовище розгортання;

P – CI/CD-конвеєр, який виконує автоматизацію процесу розгортання ПЗ.

Модель декларативного розгортання з використанням GitOps часто порівнюється із звичайним використанням CI/CD-конвеєром, але це принципово різні моделі розгортання. Це підтверджують і відмінностям між виконанням різних процесів у цих моделях, порівняння яких відображена у таблиці 2.1.

Таблиця 2.1 – Порівняння GitOps і традиційного CI/CD

Процес	GitOps	CI/CD
Підхід до розгортання	Pull-підхід	Push-підхід
Керування інфраструктурою	Декларативне, тобто, весь стан зберігається у Git	Імперативне або змішане
Розгортання	Повністю автоматизоване, через GitOps-оператор	Тригер або складні налаштування
Відкат змін	Простий Git Revert	Складніший процес, не завжди передбачено
Моніторинг стану	Оператор постійно звіряє реальний стан із Git	Моніторинг виконується окремо

Модель декларативного розгортання з використанням GitOps визначає чимало різних переваг при використанні її у реальних проєктах. Прозорість, яка досягається шляхом збереження усіх декларативних конфігурацій у репозиторії Git, моделі розгортання дозволяє легко проводити аудит та відстежувати всі зміни, які потенційно могли потрапити у результуючий реліз ПЗ. Відстеження змін за допомогою механізму Git надає змогу створити процес відтворення попередньої версії ПЗ, що сприяє швидкому відновленню при помилках у процесі розгортання. При будь-якому розгортанні важливо відзначити такий аспект, як безпека, яка досягається шляхом використання повністю автоматизованого процесу розгортання за допомогою агенту-оператора, який надає змогу мінімізувати мануальні втручання до серверів [18].

До недоліків моделі можна віднести складність реалізації, оскільки вона потребує знання в області розробки CI/CD-конвеєрів з використання Git-технологій. Останнім, але суттєвим недоліком є проблема при

зберігання таких важливих для системи даних, як паролі, підключення до серверів – всі ці дані представляють критично важливу інформацію, яка повинна зберігатися безпечно. Використовуючи репозиторії Git, не завжди вдається зберігати такого роду дані в безпеці, тому потрібні окремі підходи для вирішення цього недоліку.

Загалом, модель декларативного розгортання з використанням GitOps забезпечує надійне, відтворюване та контрольоване управління інфраструктурою. Це дає змогу досягти високої стабільності, швидкості релізів, що особливо актуально в проєктах, орієнтованих на мікросервісну архітектуру. Слід враховувати й те, що ця модель може бути інтегрована з різними DevOps-практиками, що одразу надає можливість масштабування моделі процесу розгортання.

2.3 Розробка комбінованої моделі процесу розгортання мікросервісів

У ході дослідження моделі паралельного розгортання з перемиканням мережевого трафіку та моделі декларативного розгортання з використанням GitOps було виявлено особливості, переваги та недоліки кожної з моделей. Результатом дослідження є таблиця 2.2, яка відображає різні ознаки моделей розгортання [19].

Таблиця 2.2 – Порівняння особливостей моделей розгортання

Ознака	Модель паралельного розгортання з перемиканням мережевого трафіку	Модель з використанням GitOps
1	2	3
Механізм розгортання	Виведення з експлуатації одного середовища та зміна трафіку	Внесення змін у Git, зміна у середовищі

Кінець таблиці 2.2

1	2	3
Механізм розгортання	Виведення з експлуатації одного середовища та зміна трафіку	Внесення змін у Git, зміна у середовище
Контроль версій конфігурацій	Частково або мануальний контроль	Повноцінний контроль через Git
CI/CD інтеграція	Можна реалізувати з Jenkins, GitLab CI	Git виступає як тригер; інтегрується з CI або самодостатньо працює з Git
Наявність одночасного розгортання	Обидві версії одночасно працюють, але лише одна обслуговує користувачів	Застосовується лише описаний у Git стан
Можливість швидкого відновлення	Миттєвий за допомогою миттєвого переключення трафіку на стару версію	Оператор за командою Git Revert автоматично застосовує зміни
Зміні середовища та паролі	Подаються окремо, часто в CI/CD	Керуються через зашифровані секрети у Git
Управління середовищем	Два повністю ізольованих середовища	Одне середовище з актуальний станом

Модель паралельного розгортання з перемиканням мережевого трафіку має наступний недолік: високе споживання ресурсів, що сприяє збільшенню витрат на підтримку інфраструктури. А відсутність одного зазначеного централізованого місця управління конфігурацією розгортання інфраструктури робить підтримку розгортання великої кількості сервісів досить складною задачею. Модель декларативного розгортання з

використанням GitOps має два вагомих недоліки, перший з яких – це складність реалізації, а другий – потреба додаткових технологій для збереження критично важливих даних.

Однак, переваги обох моделей розгортання надають широкий спектр можливостей для створення комплексної моделі розгортання з досить високим рівнем гнучкості та масштабування, що є ключовими потребами в IT-проєктах валідації електронних фармацевтичних документів. В ході аналізу кожної з моделей було виявлено можливість вирішити певні недоліки моделей шляхом використання можливостей іншої моделі розгортання. В таблиці 2.3 зазначена проблема та її вирішення використовуючи комбінацію моделей розгортання.

Таблиця 2.3 – Підходи вирішення проблем впровадженням комбінованої моделі розгортання

Проблема	Модель	Рішення
Збереження критично важливої інформації	Модель декларативного розгортання з використанням GitOps	Використання таких технологій, які можуть бути інтегровані з моделлю паралельного розгортання
Складність управління конфігурацією	Модель паралельного розгортання	Використання декларативного підходу щодо опису конфігурації
Складність підтримки інфраструктури	Модель паралельного розгортання	Використання єдиного джерела – Git

Отже, можна зробити висновок, що запропонована комбінована модель розгортання мікросервісів, побудована на основі двох моделей, а саме: модель паралельного розгортання та модель декларативного розгортання з використанням Git, поєднує в собі компоненти обох моделей, які наведені на рисунку 2.3, та пропонує процес, який вирішує проблеми поодиноких моделей розгортання [20]. Можливість масштабування моделі дозволяє використовувати інтеграції з різними сучасними технологіям для поліпшення процесу автоматизація, що є критичним важливим аспектом при вирішенні задачі валідації електронних фармацевтичних документів.

Компоненти комбінованої моделі розгортання зображені на рисунку 2.3.

Рисунок 2.3 – Компоненти комбінованої моделі розгортання

Для більш точного опису архітектури комбінованої моделі, що поєднує модель паралельного розгортання з перемиканням мережевого трафіку з моделлю декларативного розгортання з використанням GitOps, доцільно використати формалізований підхід, який відображає всі ключові компоненти системи та їх взаємозв'язки.

$$M = \{R, S, A, E_1, E_2, T, O, P\},$$

де M – модель розгортання;

R – репозиторій, який містить декларативні файли конфігурацій;

S – стан системи, якого має бути досягнуто в середовищі виконання;

A – агент-оператор, який відтворює всі зміни у середовищі;

P – CI/CD-конвеєр, який виконує автоматизацію процесу розгортання;

E_1 – поточне середовище;

E_2 – нове середовище з оновленою версією ПЗ;

R – механізм маршрутизації трафіку між середовищами;

T – правило або логіка перемикання трафіку між середовищами;

O – система моніторингу для оцінки стану.

Таке представлення дозволяє формалізувати логіку автоматизованого інструменту, що застосовує зміни до середовища, яке тимчасово не обслуговує запити реальних користувачів. Такий підхід забезпечує можливість безпечного тестування нової версії ПЗ без ризику порушення роботи продуктивної системи, оскільки можна провести перемикання трафіку на поточне середовище. Загальний процес розгортання за допомогою комбінованої моделі розгортання можна представити у вигляді формалізованого виразу, що дозволить відстежити взаємодію компонентів моделі:

$$M = R \to A(S) \to P \to E_2 \to O(E_2),$$

де M – модель розгортання;

R – репозиторій, який містить декларативні файли конфігурацій;

S – стан системи, якого має бути досягнуто в середовищі виконання;

A – агент-оператор, який відтворює всі зміни у середовищі;

P – CI/CD-конвеєр, який виконує автоматизацію процесу розгортання;

E_2 – нове середовище з оновленою версією ПЗ;

O – система моніторингу для оцінки стану.

3 РОЗРОБКА ПРОЄКТУ СИСТЕМИ ВАЛІДАЦІЇ ЕЛЕКТРОННИХ ФАРМАЦЕВТИЧНИХ ДОКУМЕНТІВ З ВИКОРИСТАННЯМ КОМБІНОВАНОЇ МОДЕЛІ РОЗГОРТАННЯ

3.1 Ініціалізація та розробка концепції автоматизації валідації електронних фармацевтичних документів

Автоматизація процесу розгортання ПЗ є важливою ініціативою, оскільки це дозволяє оптимізувати та поліпшити весь процес розробки. Це особливо важливо в задачах валідації фармацевтичних документів, оскільки ПЗ для виконання задач перевірки повинно завжди працювати та мати всі останні оновлення. Впровадження автоматизації спрощує обробку та обмін інформацією між різними структурами, такими як фармацевтичні компанії та регуляторні органи, що може призвести до оптимізації адміністративних процесів [21].

Розробка і впровадження моделі для автоматизації розгортання ПЗ з мікросервісною архітектурою може значно поліпшити ефективність та точність управління цим бізнес-процесом. ІТ-компанії матимуть можливість зберегти значні ресурси і вдосконалити роботу організацій, що використовують систему для перевірки фармацевтичних документів. Це також призведе до швидшого та ефективнішого виконання робіт, що сприятиме підвищенню іміджу компанії в очах замовників.

Така модель розгортання допомагає вирішувати завдання щодо розгортання інформаційної системи, яка займається процесом перевірки фармацевтичних документів з вражаючою швидкістю. Це ставить певні вимоги до кваліфікаційного рівня персоналу, зокрема, до проєктних менеджерів, які повинні мати необхідні знання для ефективного управління ресурсами, як трудовими, так і матеріальними. Результатом такого проєкту є готовий документ, який містить опис модулів для створення інформаційної системи валідації фармацевтичних документів. Такий

документ служить цінним інструментом для визначення вигідності проєкту або виявлення можливих проблем, з якими може зіткнутися компанія під час впровадження системи. Тим самим продукт надає інформацію про необхідність виконання роботи для компанії або, у випадку вже запущеного проєкту [21].

Однією з ключових умов успішного впровадження методів та стратегій управління проєктом є чітке визначення та розуміння його цілей. Встановлення конкретних цілей та їх детальний опис становлять фундаментальний етап для подальшої роботи над проєктом. Ефективне досягнення бажаних результатів у визначений термін при встановлених умовах виконання проєкту стає можливим саме завдяки визначенню чітких цілей проєкту. Однією з головних цілей цього проєкту є розробка плану створення комбінованої моделі розгортання ПЗ з використанням мікросервісної архітектури для автоматизації валідації електронних фармацевтичних документів. Окрім цього, серед інших поставлених завдань можна виокремити: підвищення ефективності взаємодії між фармацевтичними компаніями та регуляторними органами; зменшення витрат; прискорення обробки і звітності; інтеграція з іншими бізнес-системами. Щодо ключових результатів проєкту то до них належить:

– розраховані необхідні ресурси та вартість для того, щоб виконати проєкт;
– писаний життєвий цикл проєкту;
– проаналізована предметна область та визначена мета проєкту;
– створено основні плани для виконання проєкту;
– визначені основні ризики створення проєкту;
– створена діаграма проєкту.

Для успішної реалізації проєкту необхідно дотримуватися таких обмежень:

– обмеження в часі: Важливо виконувати завдання відповідно до

визначених дат завершення, які визначають загальний термін проєкту;

– обмеження щодо області розробки: Важливо дотримуватися обмеженого обсягу роботи, який може виконати команда. Занадто велика кількість кінцевих результатів або функцій може призвести до нестачі часу.

Першочергово необхідно визначити основні цілі, завдання, відповідальності, структуру, тривалість проєкту тощо. Для виконання цього завдання потрібно створити статут проєкту, який визначає ключові аспекти проєкту. Основна інформація наведена у таблиці 3.1

Таблиця 3.1 – Основна інформація за проєктом

Назва проєкту	Створення комбінованої моделі розгортання ПЗ з використанням мікросервісної архітектури для автоматизації валідації електронних фармацевтичних документів
Планова тривалість проєкту	6 місяців
Плановий бюджет проєкту	760 000 грн.
Стратегічна мета проєкту	Автоматизація системи валідації електронних фармацевтичних документів
Власник проєкту	Шипакова Маргарита Андріївна
Керівник проєкту	Шевченко Сергій Віталійович
Причини ініціації проєкту	Відсутність ефективних та швидких механізмів валідації фармацевтичних документів

Метою проєкту є розробка автоматизованої системи, котра зможе забезпечити ефективний механізм валідації електронних фармацевтичних документів. Для виконання поставленої мети визначено наступні критерії досягнення: складений план робіт з розробки системи, виявлені варіанти

концепцій, проведений аналіз предметної області, розрахована вартість та терміни реалізації [21].

Вимоги замовника:

– провести аналіз основних модулів, для яких будуть розраховуватися ризики;

– провести аналіз існуючих обмежень для обраних модулів;

– здійснити зміни, що коригують напрямок діяльності команди;

– оцінити необхідність виконання задачі оцінки ризиків;

– оцінити можливість виконання задачі ресурсами команди;

– створити діаграма проєкту.

В документі повинні бути зазначені такі вимоги до кінцевого продукту:

– виявлені вимоги до системи;

– проаналізовані вимоги;

– сформульовані вимоги;

– визначені особливості предметної області;

– можливі концепції вирішення;

– особливості реалізації системи.

Склад учасників ІТ-проєкту, як правило, формується з урахуванням індивідуальних особливостей, масштабів, технічних вимог та цілей конкретного проєкту. Важливо зазначити, що набір ролей та ступінь залучення окремих спеціалістів можуть суттєво змінюватися залежно від етапу реалізації, специфіки ПЗ, що розробляється, та організаційної структури команди. Для більшої наочності та кращого розуміння внутрішньої структури проєкту, у таблиці 3.2 представлено попередній перелік учасників, а також розподіл ролей між ними згідно з наявними компетенціями та обсягом відповідальності кожного члена команди.

Таблиця 3.2 – Учасники проєкту та розподіл їх ролей

Учасник проєкту	Роль	Відповідальність
Замовник	Замовник	Замовляння проєкту
Власник	Власник компанії	Керування компанією
Менеджер	Менеджер проєкту	Планування, організація процесів
Керівник	Керівник проєкту	Керування процесом
Розробники	Команда розробників	Розробка системи
Тестувальники	Команда QA	Тестування системи

Для уточнення умов, в яких відбуватиметься проект необхідно визначити допустимі припущення проєкту, оскільки це фундаментальні твердження, які вважаються істинними для планування та реалізації проєкту, хоча на момент створення плану вони ще не підтверджені фактами або доказами. Припущення проєкту щодо створення системи валідації електронних фармацевтичних документів:

– роботи повинні бути виконані згідно з встановленим графіком проєкту;

– учасники проєкту будуть відповідати вимогам та дотримуватися термінів виконання, встановлених замовником;

– проєкт передбачає консультаційну підтримку з боку замовника;

– проєкт отримає організаційну підтримку від менеджера проєкту;

– в організації наявні експерти, які взяли участь у проєкті;

– розуміння організаційної дисципліни присутнє з боку замовника та керівника проєкту.

Обмеження проєкту становлять один із ключових факторів, які впливають на процес прийняття рішень під час планування, організації та реалізації ІТ-проєкту. Вони визначають межі можливого, встановлюють рамки для ресурсів, строків, технічних рішень і формують вимоги до складу команди. У контексті розробки автоматизованої системи валідації

електронних фармацевтичних документів з використанням комбінованої моделі розгортання ПЗ, обмеження відіграють важливу роль у визначенні технічних, організаційних та нормативних аспектів реалізації [21, 22]:

– стандартний робочий час в організації з 10:00 до 19:00, включаючи перерву з 13:00 до 14:00;

– менеджер проєктів, який керує декількома проєктами, може реагувати з затримкою;

– збільшення вартості проєкту не повинно перевищувати 7% від початкового обсягу;

– порушення плану на більше ніж 5 днів є обмеженням часового характеру.

Запланована вартість проєкту – 760 000 грн. Вартість узгоджена між керівником і замовником.

У процесі планування ІТ-проєкту особливу увагу було приділено формуванню детального переліку операцій, необхідних для якісного виконання окремих завдань, зокрема задачі з оцінки ризиків. Враховуючи важливість цього етапу для забезпечення стабільності, надійності та ефективності майбутньої системи, усі відповідні дії було сплановано й структуровано відповідно до методології управління проєктами. Таблиця 3.3 містить узагальнений перелік операцій, передбачених планом, які безпосередньо стосуються реалізації задачі оцінки ризиків у межах даного проєкту.

Таблиця 3.3 – Операції за планом для задачі оцінки ризиків

Операція	Опис змісту робіт
1	2
Ознайомлення з проєктом	Ознайомлення з метою, описом, задачами, необхідними навичками для проєкту

Продовження таблиці 3.3

1	2
Аналіз вимог	Обговорення критеріїв прийому та елементів, які повинні бути у проєкті, опис вимог до задачі. Визначення документів, які будуть створені як результат роботи над проєктом
Визначення учасників проєкту	Визначення учасників з необхідними навичками для роботи над проєктом
Ознайомлення учасників	Ознайомлення учасників з проєктом, введення в курс подальших дій
Особливості реалізації	Запит до експерта, виділеного для проєкту на проходження анкетування для цього проєкту. По результатам анкетування визначення життєвого циклу проєкту та моделі проєктування
Концепція вирішення	Визначення існуючих концепцій, вибір концепції, які найбільш підходить до оцінки ризиків, вивчення їх переваг і недоліків
Особливості створення	Визначення вимог до проєкту, нормативів роботи з документацією, формування базових і розширених планів проєкту
Аналіз модулів задач	Визначення модулів, для яких оцінка ризиків буде проведена у першу чергу, формування складу документа, визначення правил оформлення та опис обраних модулів замовнику з аналізом вибраних модулів

Кінець таблиці 3.3

1	2
Метод додаткового зберігання інформації	Визначити, чи буде розроблятися база даних для проєкту, якщо так, то залучення нових учасників до проєктування, розробка листу посилань на функціональні модулі задачі, планування доступу користувачів до результатів проєктування
Рецензування експертом	Виявлення помилок та розстановка їх градації, оцінка ризиків, за результатами рецензії проведення робіт з виправлення помилок, починаючи з критичних
Висновки	Після рецензування остаточне оформлення звіту проєкту, формування презентації і докладу до неї. Презентація результатів роботи замовнику

Задля визначення та організації цілей завдань проєкту потрібно використати такий інструмент управління проєктами як дерево цілей. Цей метод дозволяє розкрити зв’язки між вищим рівнем стратегічних цілей і більш деталізованими завданнями на нижчому рівні. Дерево цілей проєкту виглядає як ієрархічна структура, де кожна ціль розкладається на більш малі цілі до досягнення остаточного результату [22]. Схема представляє з себе набір ієрархічно складених задач, котрі мають зв’язки між собою у вигляді ієрархічного дерева. Дерево цілей зображено на рисунку 3.1.

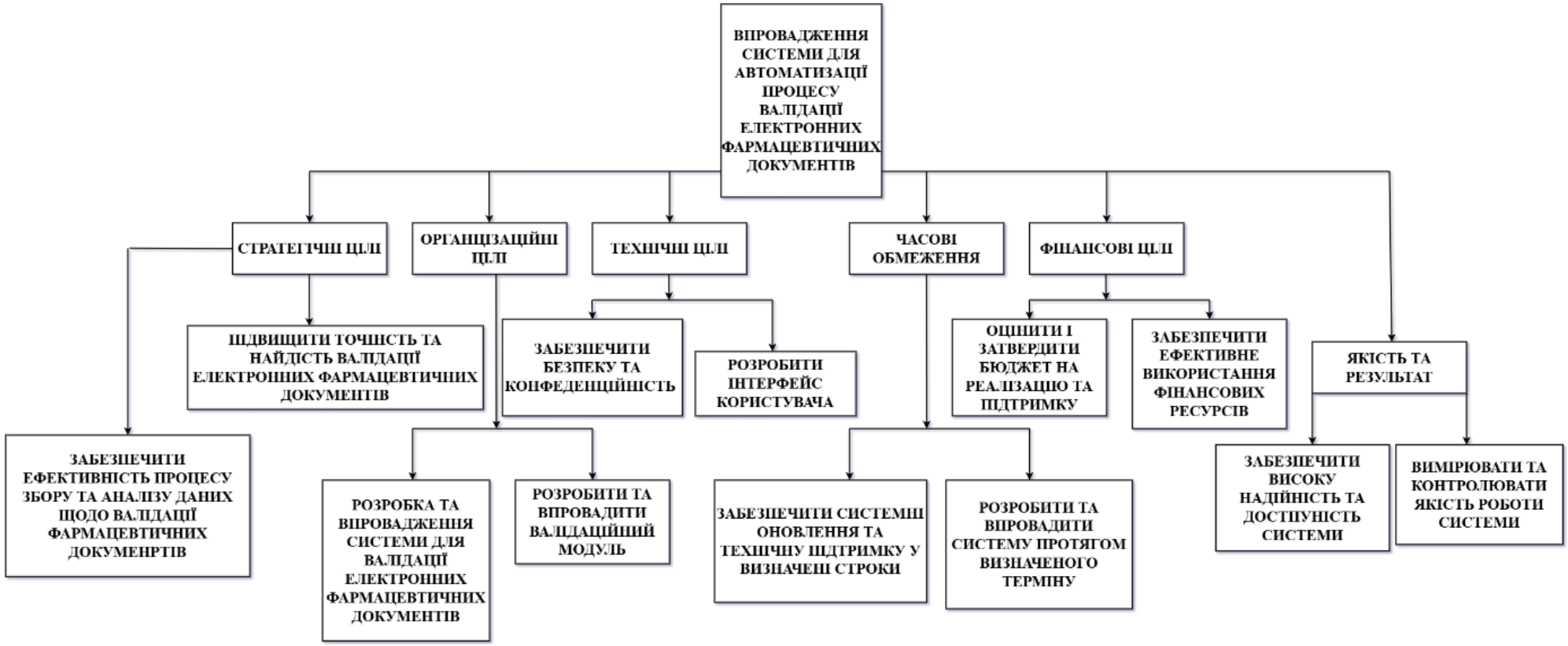

Рисунок 3.1 – Дерево цілей проєкту

3.2 Розробка життєвого циклу системи валідації електронних фармацевтичних документів

Життєвий цикл проєкту визначає послідовність етапів від періоду концепції проєкту до його завершення, розглядаючи його як послідовність фаз, подій і етапів, кожен з яких має свою назву та часові межі. Об'єкт управління, будь то автоматизація чи інформатизація, пройде три стани: початковий, цільовий і кінцевий. Початковий стан виникає в момент появи ідеї проєкту або початку фінансування створення системи. Цільовий стан пов'язаний з моментом початку функціонування об'єкта відповідно до своєї цілі. Кінцевий стан асоційований з припиненням його діяльності через фізичне чи моральне старіння, зміни або перетворення на якісний новий об'єкт. Цей процес нормативно регулюється документами і завершується випуском документації на інформаційну систему. Ця документація включає повний опис моделі інформаційної системи, що відповідає вимогам, на певному етапі, та виготовленням компонентів інформаційної системи або введенням інформаційної системи в промислову експлуатацію [22]. В таблиці 3.4 визначено життєвий цикл проєкту, його окремі фази та етапи.

Таблиця 3.4 – Життєвий цикл проєкту

Фаза	Ініціація	Планування	Виконання	Завершення
1	2	3	4	5
Перелік основних робіт	Визначення цілей, задач, відповідальності, команди проєкту	Визначення розкладу, бюджету, ризиків	Розробка програмної системи та тестування	Презентація замовнику, передача документів

Кінець таблиці 3.4

1	2	3	4	5
Ключові віхи	Затвердження вимог та команди проєкту	Затвердження розкладу, бюджету та інших ресурсів	Закінчення додатку, тестування	Прийняття замовником готового продукту
Складності	Чітке визначення задач та вимог, що бажає замовник	Підготовка до передачі результатів замовнику	Оцінка очікуванням	Створення підсумкової звітності

Учасниками проєкту є фізичні та юридичні особи, чиї інтереси пов'язані з успішною реалізацією проєкту.

Замовником проєкту є відділ організації, який виступає інвестором і відповідає за розробку проєкту. Відділ фінансових розрахунків та інвестицій, очолюваний директором з фінансів, виступає в ролі інвестора. Департамент з розробки та реєстрації, під керівництвом відповідного керівника, надає дозвіл на ініціалізацію та проєктування проєкту.

Керівник проєктів організації виступає як замовник проєкту, а експерт з розробки системи консультує та проводить анкетування для визначення моделі проєкту та інформації для розробки завдання.

Основними учасниками проєкту, безпосередньо залученими до його виконання, є менеджер проєктів, команда розробників та тестувальників, а також устаткування, які забезпечують технічну базу для команди розробників у рамках виділеного бізнес-проєкту.

Для ефективної реалізації ІТ-проєкту, зокрема такого, як створення

системи автоматизації валідації електронних фармацевтичних документів, надзвичайно важливою є чітка структура команди та розмежування функціональних ролей. Кожна роль у проєктній команді виконує свою унікальну функцію, яка сприяє досягненню поставлених цілей, дотриманню строків, забезпеченню якості кінцевого продукту [23]. Визначення ролей дозволяє уникнути дублювання обов'язків, зменшує ймовірність конфліктів між учасниками, а також забезпечує ефективне управління ресурсами та взаємодію між усіма сторонами, зацікавленими в результаті. Визначені ролі проєкту:

- ВО – відповідальна особа;
- В – виконавець;
- ПР – приймання роботи;
- КР – координація робіт;
- К – контроль;
- У – узгодження.

3.3 Структуризація проєкту системи валідації електронних фармацевтичних документів

Створення ієрархічної структури робіт – це процес розчленування результатів та завдань проєкту на менші компоненти, якими легше керувати. Основна перевага цього підходу полягає в тому, що він визначає структуру того, що необхідно виконати, розкриваючи завдання проєкту на найнижчому рівні деталізації. Деталізація спрямована на точне представлення завдань потреб плану, які передаються виконавцям. Для розробки основних елементів проєкту використовуються дерево цілей, дані про життєвий цикл та учасників проєкту [23]. Основні завдання проєкту

розробляються за допомогою структурної схеми Work Breakdown Structure (WBS), що використовується для розбиття цілого проєкту на менші, керовані частини. Структурна декомпозиція робіт зображена на рисунку 3.2.

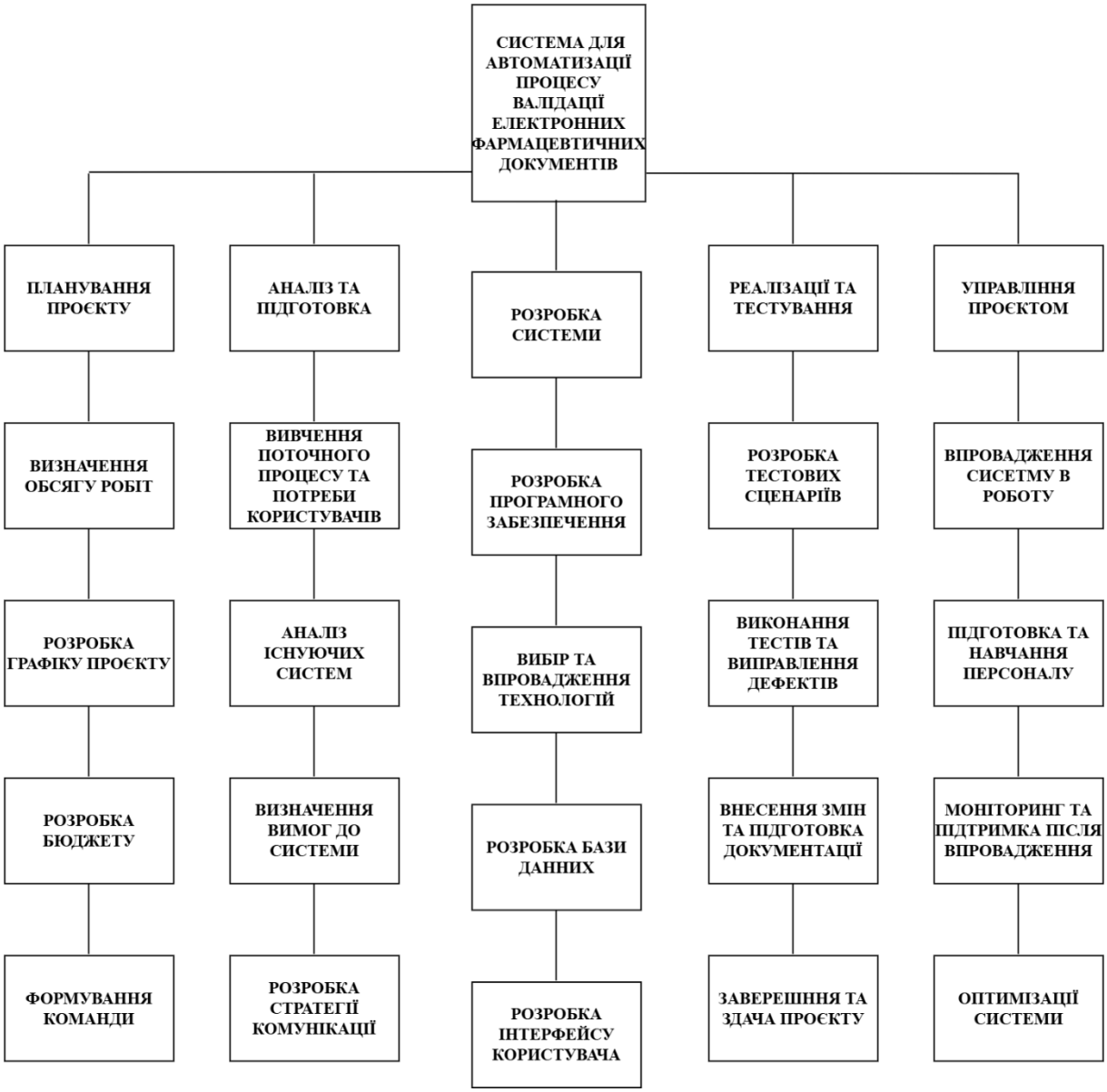

Рисунок 3.2 – Структура декомпозиції робіт

Ієрархічна структура ресурсів – це система ієрархічного відображення персональних та матеріальних ресурсів команди, яка розподіляє їх за категоріями та типами. Ця структура використовується для планування та

контролю в ході роботи над проєктом, а також для ефективного управління. Кожен рівень ієрархії надає все більш детальний опис ресурсів, поки не досягається достатня деталізація, щоб використовувати цю інформацію разом з ієрархічною структурою робіт для планування, моніторингу та контролю роботи. На рисунку 3.3 зображена ієрархічна структура ресурсів.

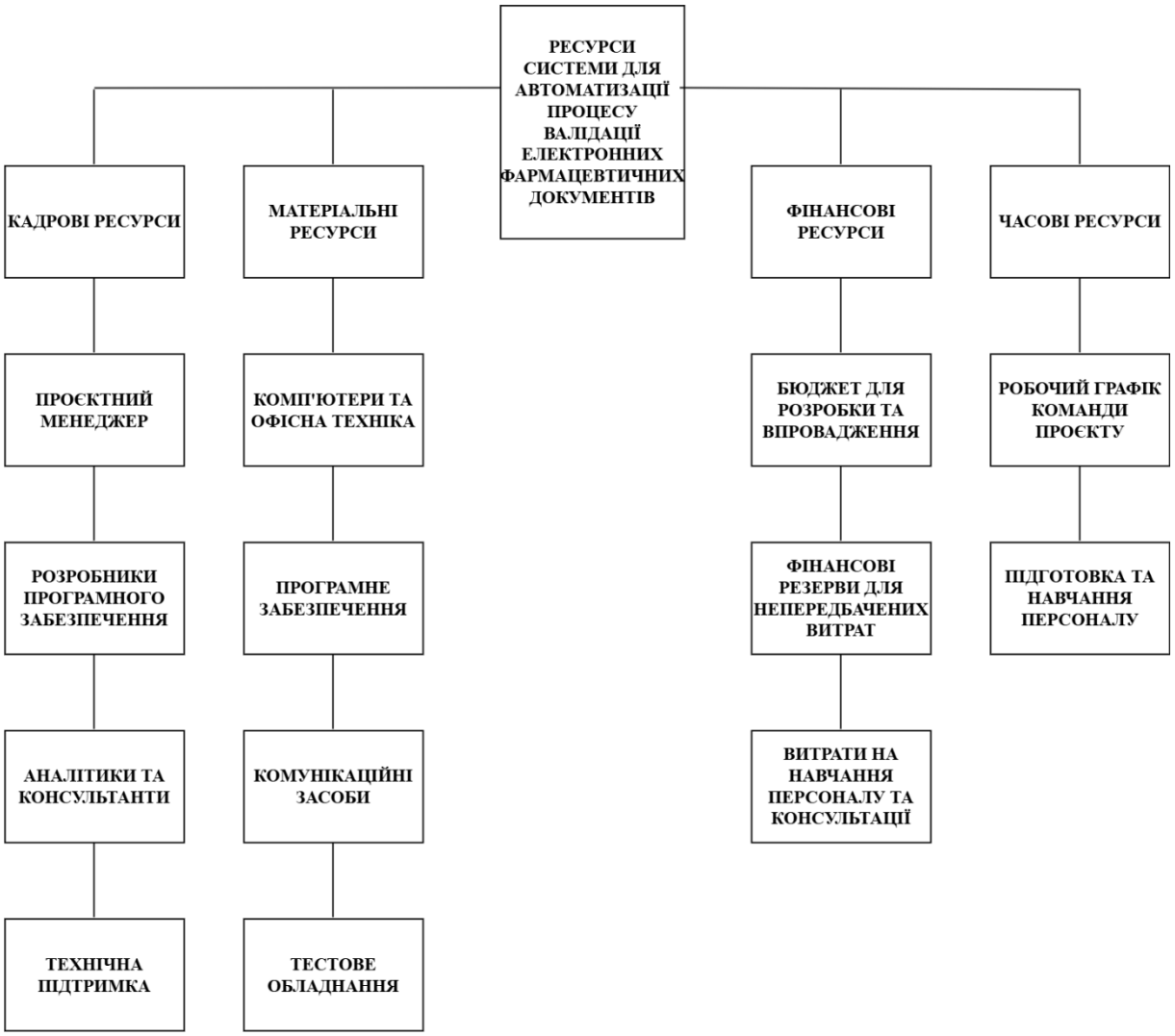

Рисунок 3.3 – Ієрархічна структура ресурсів проєкту

Фінансові ресурси проєкту визначають можливості проєкту, враховуючи кількість ресурсів, які можуть бути задіяні, потребу в них для виконання робіт у визначений термін, а також вплив на продуктивність

колективу розробників, оскільки ефективність безпосередньо залежить від ПЗ проєкту. Оцінюючи виділені фінансові ресурси на початку проєкту, було створено розрахунок плану вартості проєкту, який зазначено у таблиці 3.5.

Таблиця 3.5 – План вартості проєкту

№	Ресурс	Витрати, грн.
1	Проєктний менеджер	70.000
2	Розробник ПЗ	120.000
3	Аналітик	60.000
4	Технічна підтримка	50.000
5	Комп’ютери та офісна техніка	40.000
6	ПЗ	120.000
7	Комунікаційні засоби	20.000
8	Тестове обладнання	30.000
9	Бюджет для розробки та Впровадження	70.000
10	Фінансові резерви	30.000
11	Витрати на навчання персоналу	20.000
12	Робочий графік команди Персоналу	40.000
13	Приміщення	50.000
Загалом	720 000	

Отже, сума вартості на 6 місяців становить 720 000 грн., що не перевищує бюджет проєкту. Опрацьовуючи кожен ресурс, враховано технічні та організаційні потреби задля забезпечення реалізації проєкту. Значна частина виділена на управлінські та людські ресурси. План витрат є збалансованим та відповідає вимогам успішного впровадження проєкту.

3.4 Проєктування системи валідації електронних фармацевтичних документів

Проєктування інформаційної системи – це процес розробки архітектури, структури та компонентів інформаційної системи для вирішення певних бізнес-задач. Цей етап включає в себе детальне визначення вимог, розробку концепційних та технічних моделей, визначення структури бази даних, вибір технологій.

На етапі проєктування відбувається внесення головних чинників, які будуть виконуватися на етапі планування проєкту. Розроблену логіко-інформаційну схему управління процесом проєктування відображено у таблиці 3.6.

Таблиця 3.6 – Логіко-інформаційна схема управління процесом проєктування

№	Задача	Вихідна інформація	Виконавець	Споживач результату
1	2	2	3	4
1	Аналіз потреб та визначення функціональності системи	Вимоги до системи	Бізнес-аналітик	Команда проєкту, замовник
2	Проєктування бази даних та інтерфейсу системи	Специфікація вимог, архітектурна схема	Архітектор	Розробники, тестувальники

Кінець таблиці 3.6

1	2	3	4	5
3	Розробка ПЗ	Діаграми бази даних, прототип інтерфейсу	Розробники	Розробники, тестувальники
4	Тестування та відладка	Вихідний код, інтеграційні тести	Тестувальники	Розробники, замовник
5	Моніторинг та підтримка системи	Звіти про роботу системи, запити на технічну підтримку, заяви на зміни	Технічна підтримка	Користувачі, технічна підтримка
6	Аудит та оптимізація системи	Запити на оптимізацію, аналіз роботи системи	Архітектор	Розробники, технічна Підтримка

Однією з необхідних дій є визначення часових рамок виконання різних етапів проєкту та встановлення зв’язків між ними шляхом створення розрахунку мережевого графіку проєкту. За допомогою такого методу можна візуалізувати послідовність та тривалість завдань, а також визначити критичний шлях проєкту. Розрахунок мережевого графіку проєкту наведено у таблиці 3.7.

Таблиця 3.7 – Розрахунок мережевого графіку проєкту

№	Задача	Затрати, грн	Тривалість, дні	Початок	Кінець
1	Аналіз потреб та визначення функціональності системи	30.000	15	01.01.2024	20.01.2024
2	Проєктування бази даних та інтерфейсу	60.000	30	21.01.2024	19.02.2024
3	Розробка ПЗ	120.000	60	20.02.2024	20.04.2024
4	Тестування та відладка	80.000	40	21.04.2024	30.05.2024
5	Впровадження та навчання персоналу	40.000	20	31.05.2024	19.06.2024
6	Моніторинг та підтримка системи	30.000	15	20.06.2024	04.07.2024
7	Аудит та оптимізація системи	20.000	10	05.07.2024	14.07.2024
8	Підготовка та здача проєкту	80.000	30	15.07.2024	14.08.2024
9	Підготовка документації	30.000	15	15.08.2024	29.08.2024
10	Підготовка та проведення презентацій	40.000	20	30.08.2024	18.09.2024
11	Завершення та оцінка	30.000	15	19.09.2024	03.10.2024

3.5 Керування часом, вартістю та ресурсами проєкту

Microsoft Project – це інструмент управління проєктами, розроблений корпорацією Microsoft. Його призначення полягає в тому, щоб надати менеджерові проєкту засоби для створення планів, розподілу ресурсів, відстеження прогресу та аналізу обсягів робіт. Також він допомагає генерувати звіт завдяки інтегрованому модулю, зокрема можна створити звіт щодо критичного шляху проєкту [24].

Згідно з визначеними етапами та роботами було створено план проєкту. З метою створення оптимального плану проєкту, початок виконання робіт проєкту було назначено на 1 січня 2024 року. Було створено ресурси проєкту за допомогою ПЗ Microsoft Project.

Microsoft Project першочергово дозволяє створити проєкт, вести всі необхідні дані про проєкт, створити необхідні роботи та ресурсі. Це дозволяє вести подальшу обробку та роботу з проєктом. Створені ресурси зображені на рисунку 3.4.

Ім'я ресурсу	Тип	Одиниця вимірюван матеріалів	Ініціали	Група	Макс. одиниц	Звич. ставка	Понад. ставка	Витрати,	Нараху	Основний календар
Management	Робота		M		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard
Project Manager	Робота		P		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard
Analyst	Робота		A		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard
Developer	Робота		D		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard
Testers	Робота		T		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard
Trainers	Робота		T		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard
Technical Communicators	Робота		T		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard
Deployment Tea	Робота		D		100%	$0.00/год	$0.00/год	$0.00	Пропорц	Standard

Рисунок 3.4 – Створені ресурси

Після створення плану проєкту, отримано діаграму Ганта, яка наведена на рисунку 3.5. На діаграмі визначено критичний шлях (червона лінія на діаграмі). Критичний шлях виник через те, що на даному етапі

ресурси перенавантажені, тому необхідно провести оптимізацію плану проєкту, а саме назначити попередників. В такому випадку роботи будуть виконуватися паралельно. Після проведення оптимізація отримано нову діаграму Ганта, яка зображена на рисунку 3.6.

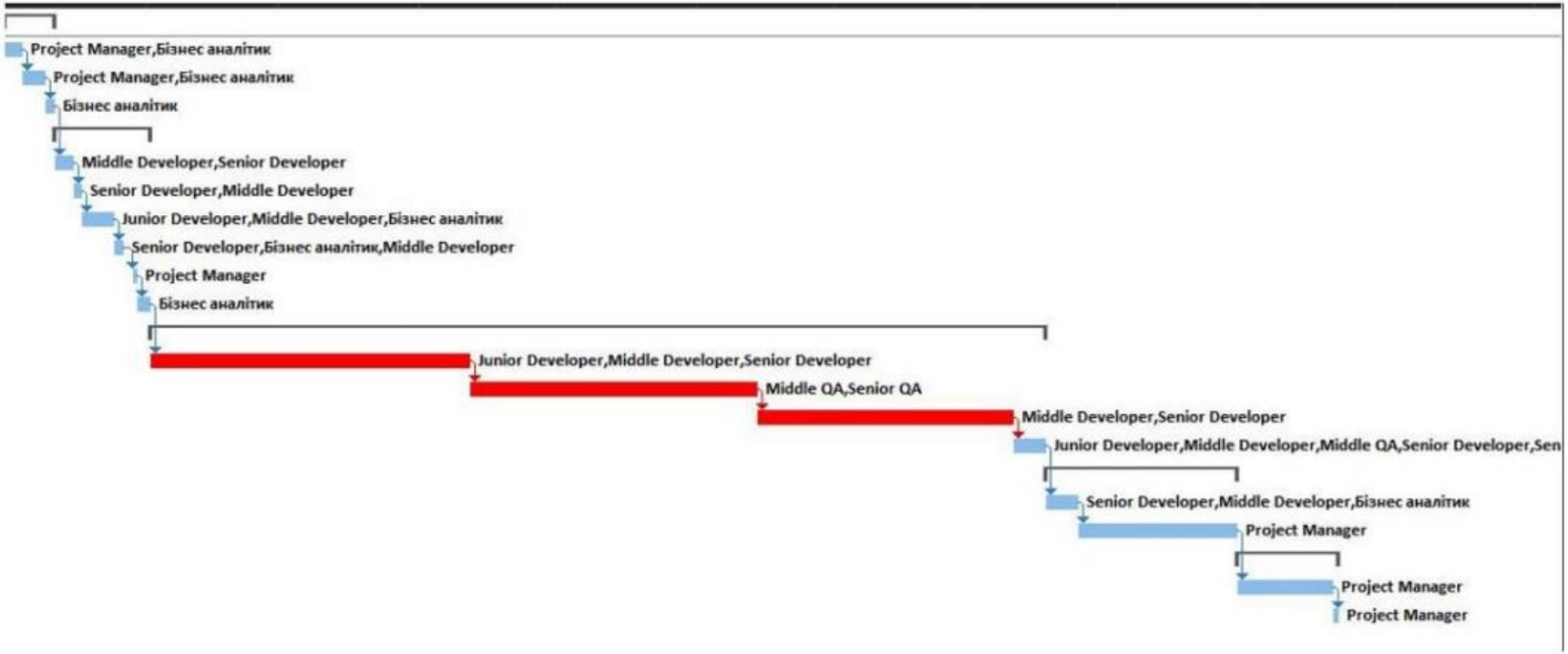

Рисунок 3.5 – Діаграма Ганта до оптимізації ресурсів

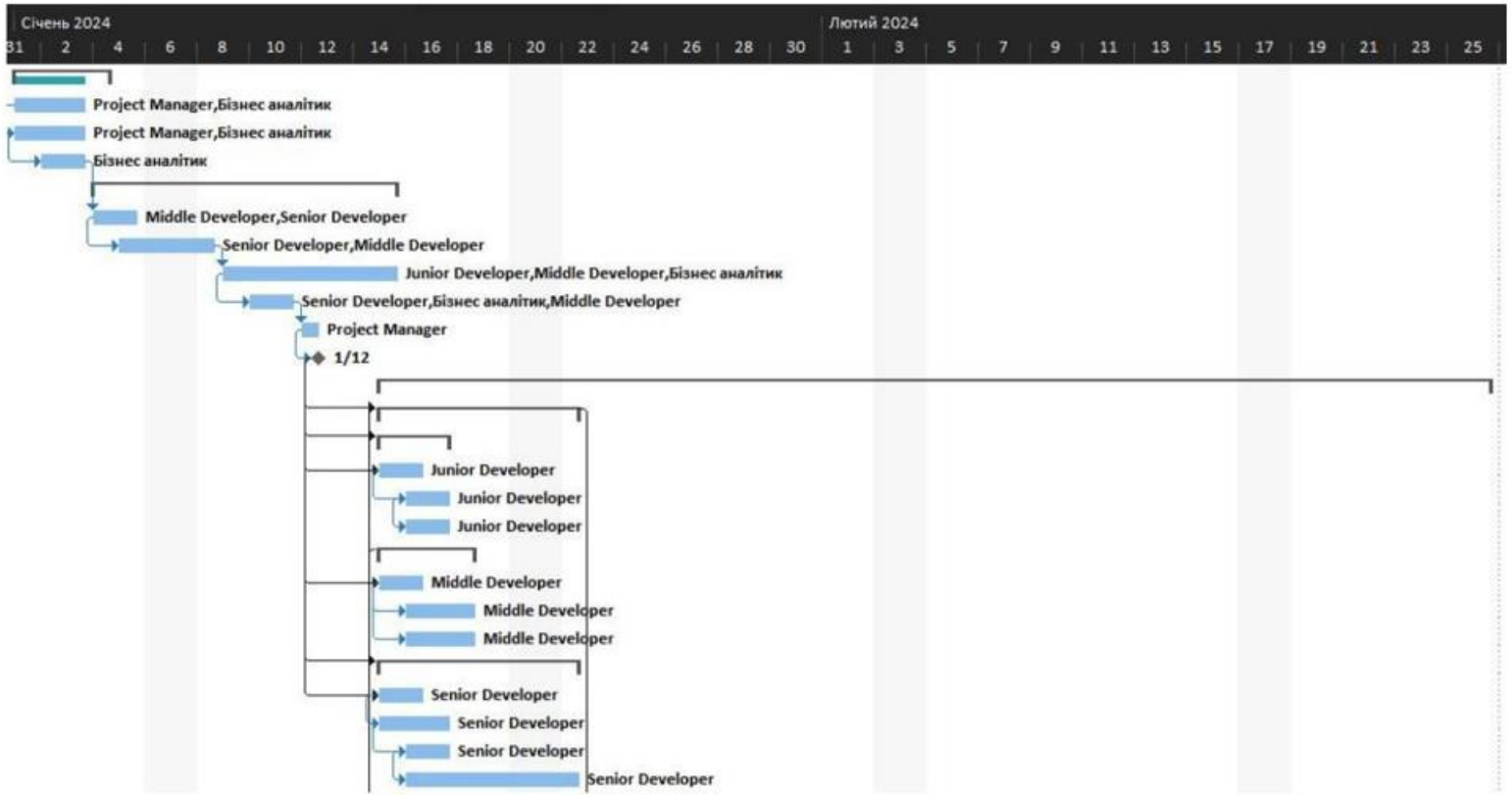

Рисунок 3.6 – Діаграма Ганта після оптимізації ресурсів

Під час планування проєкту були визначені основні етапи виконання робіт, включаючи визначення вимог, розподіл завдань, розробку графіка, оцінку ризиків та визначення ресурсів. Аналіз ризиків дозволив передбачити можливі труднощі та прийняти заходи для їх вирішення заздалегідь. Важливим етапом було визначення технічних аспектів системи, вибір архітектурних рішень та оцінка технічних вимог. Загалом, цей проєкт підкреслює важливість правильного підходу до етапу планування для успішної реалізації комплексних інноваційних проєктів у сфері автоматизації валідації електронних фармацевтичних документів.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА МОДЕЛІ ПРОЦЕСУ РОЗГОРТАННЯ МІКРОСЕРВІСІВ

4.1 Проєктування архітектури системи валідації електронних фармацевтичних документів

Створення архітектури системи – це комплексна задача, при якій необхідно брати до уваги чимало речей, особливо коли мова йде про рішення, що стосуються оптимізації бізнес-процесів у сфері охорони здоров’я. Зазвичай, такі рішення потребують проєктування архітектури з застосуванням сучасних технологій для гнучкого масштабування та підтримки доступності системи. Для реалізації інфраструктури системи можна використовувати хмарні технології, які надають обчислювальні послуги при їх необхідності. До хмарних послуг може відноситися надання таких ресурсів як: фізичні сервери, сховища, бази даних, мережі, ПЗ, аналітика та інтелект – все це постачається через публічну мережу Інтернет [26]. Ці ресурси зберігаються в, так званій, хмарі, яка, є мережею хмарних серверів по всьому світу. Ключовим моментом є те, що компанії можуть отримати доступ до цих ресурсів незалежно від свого місцезнаходження, тобто з різних місць по всьому світу. Це забезпечує швидші інновації, гнучкі ресурси та економію масштабу. Компанії зазвичай платять лише за хмарні послуги, які вони використовують, що допомагає їм знизити експлуатаційні витрати, ефективніше керувати інфраструктурою та масштабуватися відповідно до змін потреб бізнесу.

Першим і фундаментальним етапом створенні хмарної архітектури системи – це визначення організації. Архітектура Microsoft Azure забезпечує механізм ідентифікації організацій завдяки унікальній ідентифікаційній директорії, який називається тенантом. Тенант – це повністю ізольована середа або корпоративний каталог хмари Microsoft Azure, який однозначно

визначає одну певну організацію в рамках хмари. Ця середа надає можливість керувати користувачами або групами користувачів, ролями, автентифікацією та доступом до всіх ресурсів у хмарі. Тенант визначає таке поняття, як підписка – окремий контейнер для ресурсів хмари Azure, за які ведеться окремий облік споживання та нараховуються платежі. Загалом, одна організація може мати декілька різних підписок, які відповідають за різні аспекти середи або ресурси [26].

Після визначення організації необхідно створити ресурсну групу, яка буде зберігати всі хмарні ресурси, які будуть використані системою. Ресурсна група – це важливий та ключовий аспект Azure. Головною ціллю групи є поєднання різних типів ресурсів. Групи дозволяють керувати ресурсами як єдиним цілим, і спрощують організацію так контроль над усіма ресурсами в групі. Цей контейнер є досить специфічним, оскільки він дозволяє не тільки зберігати ресурси в одному місці, але й керувати доступом до ресурсів, застосовуючи ролі або політики доступу. Логічне збереження усіх ресурсів в одному місці дозволяє забезпечити збереження метаданих про кожен окремий ресурс у групі в одному вибраному регіоні, що є досить важливим аспектом для деяким послуг Azure.

Першим, але досить головним з точки зору безпеки хмарним ресурсом, виступає віртуальна мережа Azure, оскільки майже кожна IT-інфраструктура має свою власну приватну мережу, що дозволяє створювати стратегії передачі даних з максимальною безпекою. Створення локальної мережі це досить складна задача, оскільки потрібно забезпечити всі необхідні умови, що зв’язати декілька фізичних комп’ютерів, для відтворення локальної корпоративної мережі. Для вирішення цієї задачі використовується хмарні технології, які дозволяють створити та налаштувати аналог мережі у хмарі. На високому рівні віртуальна мережа Azure – це контейнер, який забезпечує ізоляцію, розділення трафіку та працює майже так, як і звичайна локальна мережа, дозволяючи

розташовувати фізичні сервери, які можуть взаємодіяти між собою, в декількох підмережах, які знаходяться в одній віртуальній мережі, а взаємодія різних ресурсів Azure можлива завдяки протоколу TCP/IP та через приватні IP-адреси через мережеву шину Azure, яка забезпечує безпеку даних. Отже, мережеві налаштування є досить важливою та критичною задачею в рамках проєктування архітектури, оскільки мережеві налаштування є фундаментальною базою кожної корпоративної системи, які можуть бути створені за допомогою Azure, що надає чимало різних додаткових конфігурацій, які можуть бути застосовані в різноманітних сценаріях [26].

Планування архітектури є важливим аспектом при розробці ПЗ, а вибір архітектури, орієнтованої на хмарні технології, надає такі переваги, як забезпечення масштабованості та доступності, оптимізація витрат компанії, централізоване управління безпека, логічна організація та контроль хмарних ресурсів, що відповідає сучасним стандартам та технологіям. Архітектура системи валідації електронних фармацевтичних документів зображена на рисунку 4.1.

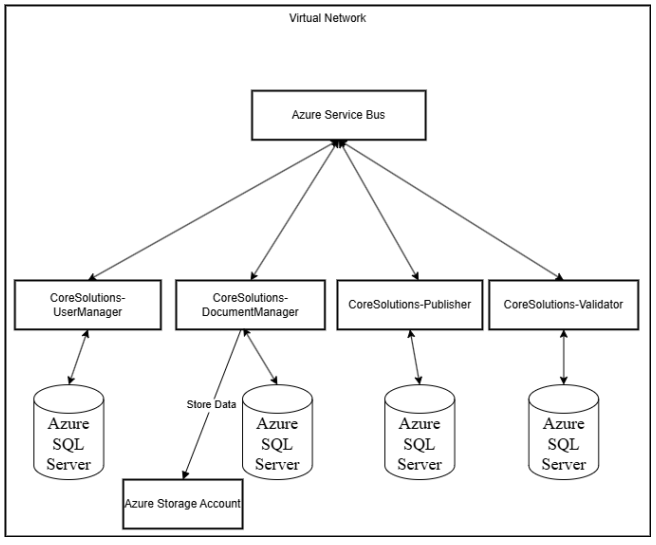

Рисунок 4.1 – Архітектура системи валідації фармацевтичних документів

4.2 Опис мікросервісу «CoreSoultion-UserManagement»

Мікросервіс «UserManagement» відіграє важливу роль у системі, оскільки його відповідальність – це створення користувачів, збереження облікових їх облікових записів і повне управління користувачами в системі, що включає визначення прав та можливостей кожного окремого користувача в системі. Управління користувачами в системі – це одна з головних задач цього сервісу, але він виконує й інші важливі задачі, а саме: автентифікація користувачів, авторизація користувачів та управління доступом до інших сервісів у системі – всі ці задачі є досить важливими з точки зору безпеки у системі [27].

Реалізація такого сервісу повинна мати централізована сховище даних користувачів, ресурсів і політик, які необхідні для забезпечення доступу до інших ресурсів у системі. При реалізації сервісу використовується система управління базами даних Microsoft SQL Server [28].

Архітектура сервісу «UserManagement» має декілька рівнів, кожен з яких відповідає за свої головні задачі. Це дає змогу забезпечити надійне розділення концептів та задач між різними рівнями компоненту, що сильно спрощує тестування кожного окремого рівня. Кожен з цих рівнім має свої власні алгоритми, класи, які створені для вирішення поставленої задачі в рамках цього сервісу [29].

Даний сервіс виділяє наступні рівні:

– рівень «Abstract» – надає базові моделі даних та ключові контракти для реалізації наступним рівням системи;

– рівень «BusinessLayer» – надає класи, що реалізують бізнес-логіку для вирішення задач;

– рівень «DataAccessLayer» – визначає класи, які виконують взаємодію з базою даних;

– рівень «Server» – фактичний серверний застосунок, який обробляє запити користувачів.

Кожен з цих рівнів має певну відповідальність, яка є важливою для побудови гнучкого сервісу з можливістю підтримки та масштабування.

Рівень «Abstract» є ключовим рівнем для визначення всіх ключових фундаментальних моделей та класів реалізацій, на основі яких будується бізнес-логіка сервісу. Даний рівень визначає наступні важливі такі важливі компоненти:

– ролі користувачів, яку можуть бути використані у системі;

– конфігурації API-сервісів для взаємодії;

– базову конфігурації серверу;

– контракти для реалізації компонентів;

– моделі даних;

– специфікації;

– конфігураційні компоненти, які налаштовують компоненти, які реалізують визначені контракти.

Цей рівень є ключовим для реалізації сервісу управління користувачами, оскільки він визначає всі фундаментальні аспекти для побудови результуючого рішення.

Рівень «DataAccessLayer» – це рівень, який відповідає за пряму роботу з специфічною системою управління базами даних, а саме Microsoft SQL Server. На даному етапі реалізуються усі важливі класи для створення запитів до бази даних, отримання необхідної інформації та перетворення цієї інформації у необхідний застосунку формат [28].

Збереження інформації щодо міграцій за допомогою технології Entity Framework – це одна задач цього рівня. Уся інформація зберігається у спеціальних класах, що дає можливість використовувати ці компоненти в таких механізмах, як конвеєр для застосування необхідної міграції в базі даних.

Створення конфігураційних класів, які описують фактичну таблицю у базі даних, є іншою задачею цього рівня. Створені класи конфігурацій повністю описують таблицю, її властивості, обмеження тощо. Такий підхід надає можливість тримати всю конфігурації бази даних в одному місці, та організовувати процес міграцій ефективніше, оскільки міграції використовують цю інформацію для створення необхідної схеми даних. Реалізація конфігураційної функції для таблиці «UserRole» приведена у лістингу 4.1

Лістинг 4.1 – Реалізація конфігураційної функції для таблиці «UserRole» (файл UserRoleConfiguration.cs)

```
public void Configure
(EntityTypeBuilder<ApplicationUserRole> builder)
      {
          builder.
               HasOne(ur => ur.Role).
               WithMany(r => r.Users)
               .HasForeignKey(ur =>ur.RoleId)
               .HasPrincipalKey(r => r.Id);

          builder.
               HasOne(ur => ur.User).
               WithMany(u => u.Roles).
               HasForeignKey(ur => ur.UserId).
               HasPrincipalKey(u => u.Id);
      }
```

Остання, але ключова, задача цього рівня – визначення класу підключення до фактичної бази даних. Цей компонент об’єднує попередні задачі для створення підключення до необхідної бази даних, а завдяки правильно налаштованим конфігураційним компонентам передає всю

необхідну інформацію на рівень нижче, що дає змогу правильно виконувати всі операції, що стосуються пошуку даних, їх перетворення в об’єкти платформи .NET. Реалізації цього функціоналу приведена у лістингу 4.2.

Лістинг 4.2 – Реалізація підключення до бази даних (файл ApplicationPersistedGrantDbContext.cs)

```
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
     optionsBuilder.
               UseSqlServer(this.connectionStringOptions.
               SqlServerConnection, sql => sql.
     MigrationsAssembly(typeof(ApplicationPersistedGrantDbConte
xt).Assembly.FullName));
}
```

Рівень «BusinessLayer» відповідає за визначення усіх необхідних компонентів, які надають програмні рішення бізнес-проблеми. Саме цей рівень є ключовим у сервісі «UserManagement», оскільки саме тут зосереджена уся бізнес-логіка, яка вирішує поставлені задачі щодо управління користувачами, їх правами та політками доступу. На даному рівні також визначені компоненти для реалізації механізмів автентифікації та авторизації у системі. Ця частина сервісу є складнішою, оскільки саме тут комбінують та використовуються усі попередні рівні для створення єдиного рішення з використанням слабкої залежності компоненту від компоненту. Саме це є причиною визначення окремого програмного компоненту який комбінує та реєструє всі компоненти та їх залежності в одному місці. Це дає можливість підключити всі необхідні залежності та компоненти на серверному рівні за допомогою одно виклику функції. Програмна реалізація такої функції приведена у лістингу 4.3.

Лістинг 4.3 – Реалізація функції розширення для підключення необхідних реалізацій (файл ServiceCollectionExtensions.cs)

```
public static
IServiceCollection AddBusinessLayerServices
<TUser, TRole, TKey>(this IServiceCollection services)
        where TUser : IdentityUser<TKey>
        where TRole : IdentityRole<TKey>,new()
        where TKey : IEquatable<TKey>
{
     services.TryAddScoped
     <IRoleService<TRole, TKey>, RoleService<TRole, TKey>>();

     services.TryAddScoped
     <IUserService<TUser, TKey>, UserService<TUser, TKey>>();

     services.TryAddScoped
     <IAuthenticationService<TUser, TKey>,
     AuthenticationService<TUser, TKey>>();

     services.TryAddScoped
     <IUserRoleService<TUser, TRole, TKey>,
     UserRoleService<TUser, TRole, TKey>>();

     return services;
}
```

Рівень «Server» відповідає за обробку фактичних запитів з використанням Hypertext Transfer Protocol (HTTP) від користувачів, які мають намір автентифікуватися у системі для подальшої взаємодії з її функціоналом. Серверний рівень визначає такі компоненти як:

– клас конфігурації серверу, який конфігурує сервер для реального використання;

– контролери, які відповідають за фактичну обробку HTTP-запитів;

– опис інфраструктури серверу, який описує важливі аспекти серверу, наприклад, конвеєр оброки HTTP-запиту.

Конфігураційний компонент серверу є досить важливим аспектом в створенні будь-якого серверного застосунку на платформі .NET, оскільки він відповідає за конфігурацію інфраструктури серверу та всіх необхідних компонентів бізнес-логіки.

Контролери серверного рівня відповідають за оброку HTTP-запитів від окремих користувачів. Даний сервіс використовує контролери, які орієнтовані на обробку запитів щодо управління користувачами, їх ролями та політками доступу у системі. Контролер, що відповідає за автентифікацію окремих користувачів у системі є найважливішим компонентом цього сервісу, оскільки він є вхідною точкою до системи. До основного функціоналу цього контролеру входять наступні функції: реєстрація користувача, виконання автентифікації та вихід з системи, що означає повне видалення токену безпеки з браузеру. Реалізація функції, що відповідає за вхід у систему, наведена у лістингу 4.4.

Лістинг 4.4 – Реалізація функції, відповідальної за вхід у систему (файл AuthenticationController.cs)

```
[HttpPost][AutoValidateAntiforgeryToken]
public async Task<IActionResult> Login(LoginUserModel
userModel)
{
     if(!this.ModelState.IsValid) return this.View(userModel);
     var user = await this.userService.
     FirstOrDefaultAsync(u => u.UserName==userModel.UserName);
```

Кінець лістингу 4.4

```
     if(user == null)
     {
          this.AddErrorToModelState(
               IdentityErrors.InvalidUserName());
          return this.View(userModel);
     }

     var signiors = await this.authenticationService.
          SignInWithPasswordAsync(user, userModel.Password);

     if (signInResult.Succeeded) return
     this.Redirect(userModel.ReturnUrl ?? new
     PathString($"/Home/Index"));


     AddErrorToModelState(IdentityErrors.InvalidPassword());
     return View(userModel);
}
```

4.3 Опис мікросервісу «CoreSoultion-Validator»

Мікросервіс «Validator» є ключовим у всій системі, оскільки саме він надає можливість виконувати перевірки як і окремих частин документів, так і всього документу. Створення двох окремих перевірок надає можливість оптимізувати перевірку, оскільки при точковому виправленні документів, перевірка буде займати набагато менше.

Архітектура сервісу «Validator» побудована за багаторівневим принципом, де кожен рівень відповідає за виконання чітко визначених

функцій. Такий підхід забезпечує чітке розмежування обов’язків між складовими сервісу, що значно полегшує процес розробки та тестування окремих частин системи. Кожен рівень містить власні алгоритми й класи реалізації бізнес-логіки, спеціально спроєктовані для ефективного вирішення завдань у межах своєї відповідальності.

Даний сервіс виділяє наступні рівні:

– рівень «Abstract» – надає моделі даних та контракти для реалізації;

– рівень «AzureBlobStorage» – надає можливість викликати функції Azure Storage Account для роботи з файлами;

– рівень «BusinessLayer» – надає головні класи з бізнес-логікою для вирішення задач;

– рівень «Server» – фактичний серверний застосунок, який обробляє запити користувачів.

Рівень «Abstract» відіграє базову роль у формуванні основних моделей і компонентів, які слугують фундаментом для побудови бізнес-логіки та решти частин сервісу. Саме на цьому рівні визначаються ключові елементи, необхідні для подальшого розвитку архітектури сервісу

– список кодів помилок при перевірці;

– допустимі розширення документів для обробки;

– постачальники валідаційних параметрів;

– абстракції, які забезпечують слабку залежність.

Створення такого рівня в архітектурі цього сервісу є надзвичайно важливим з кількох причин. По-перше, він забезпечує чітку структуризацію коду, відокремлюючи базові моделі, інтерфейси та загальні компоненти від прикладної логіки та реалізацій. Це сприяє високому рівню повторного використання коду, полегшує супровід і тестування системи, а також зменшує залежність між модулями. По-друге, рівень створює стабільний контракт, на який можуть покладатися інші частини системи – наприклад, рівень бізнес-логіки. Завдяки цьому зміни в реалізації не призводять до

масових змін у всьому коді. Такий підхід відповідає Single responsibility, Open-closed, Liskov substitution, Interface segregation і Dependency injection (SOLID) принципам. і є основою для побудови масштабованої та легко підтримуваної архітектури.

Усі класи рівня «Abstract» зазначені на рисунку 4.2, який представляє діаграму класів рівню.

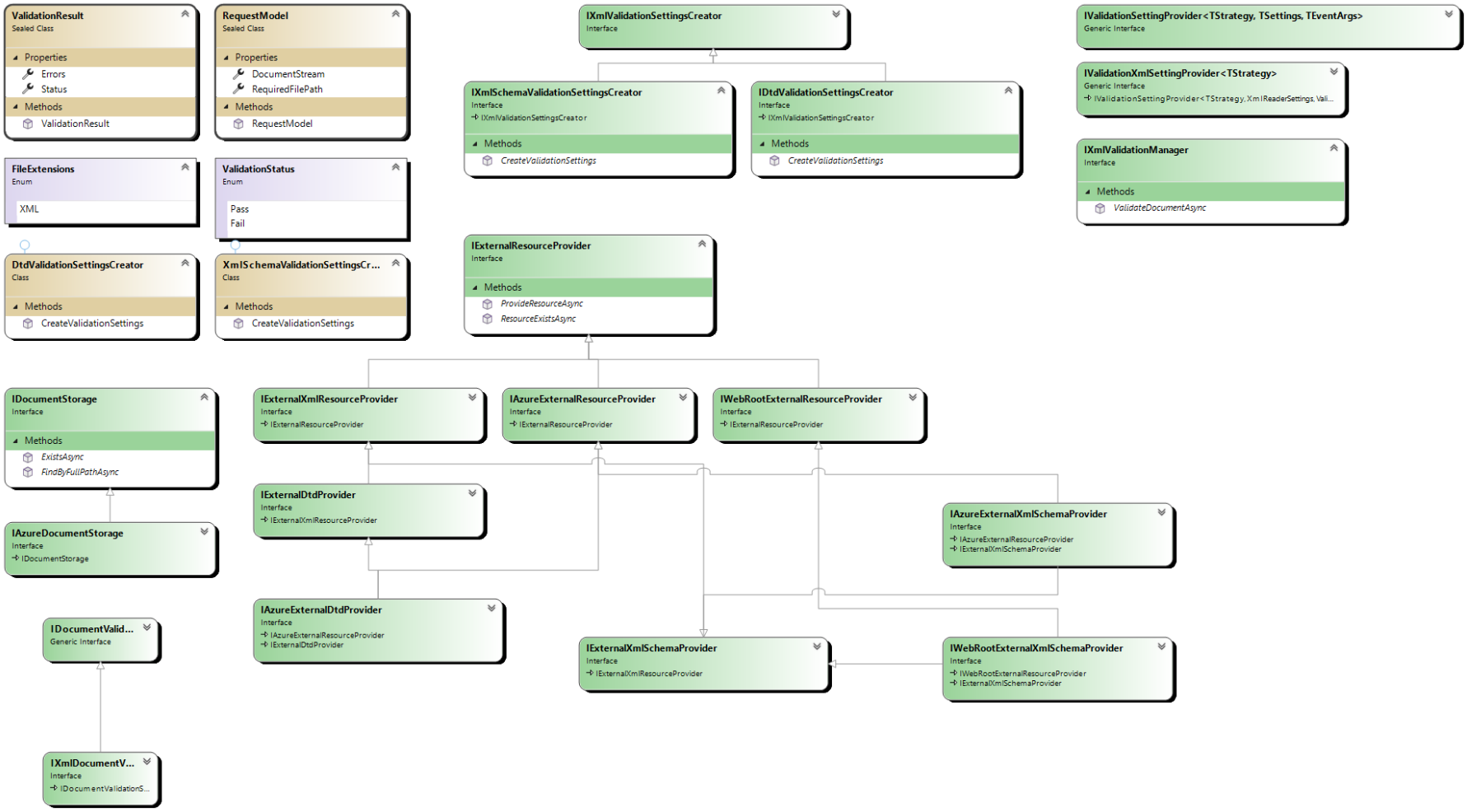

Рисунок 4.2 – Діаграма класів рівня «Abstract» сервісу «CoreSoultion-Validator»

Рівень «AzureBlobStorage» надає можливість використовувати компоненти Microsoft Azure за допомогою сучасного Application Programming Interface (API), створеного на платформі .NET. Створення такого рівня необхідне з архітектурної точки зору, оскільки це повністю ізольована одиниця, яка підключається до основного рівню бізнес-логіки, розширяючи його можливості, надаючи інструменти для роботи з Azure Storage Account.

Архітектура рівня побудована відповідно до принципів об'єктно-орієнтованого проєктування та підходів SOLID, зокрема розділення відповідальності та відкритості до розширення. Базовим компонентом функціоналу виступає компонент автентифікації, який визначає контракт для отримання автентифікованого клієнта Azure. Його реалізації поділяються на дві основні моделі автентифікації: через облікові дані та через ключ доступу. Обидві моделі успадковують спільний абстрактний клас, який інкапсулює спільну логіку автентифікації. Конкретні класи реалізують логіку, залежно від типу автентифікації, використовуючи налаштування, задані в конфігураційних компонентах. Реалізація автентифікації на основі облікового запису зазначена у лістингу 4.5.

Лістинг 4.5 – Реалізація функції автентифікації на основі облікового запису Azure (файл AzureCredentialsBlobAuthenticator.cs)

```
public virtual BlobServiceClient AuthenticateClient(){
     var parameters = new object[]
     {
          new Uri(this.options.FullyQualifiedNamespace),
          new DefaultAzureCredential(),
          new BlobClientOptions()
     };
     return base.Authenticate(parameters);
 }
```

Для взаємодії з самим сховищем документів визначено контракт, який містить методи для перевірки наявності файлів та їх завантаження. Конкретна реалізація цього контракту працює з автентифікованим клієнтом, наданим відповідним автентифікатором. Запити до сервісу описуються через інтерфейс, який містить необхідні властивості: ім’я контейнера та шлях до файлу – все це надає можливості розширювати базові компоненти та їх функціонал без порушення основних принципів SOLID. Реалізації функції завантаження документу з Azure Storage Account приведена у лістингу 4.6.

Лістинг 4.6 – Реалізація функції завантаження документу з Azure Storage Account запису Azure (файл AzureBlobService.cs)

```
public virtual async Task<BinaryData>
DownloadBlobAsync(IBlobRequestModel requestModel,
                  CancellationToken token = default)
{
     token.ThrowIfCancellationRequested();
     ArgumentNullException.ThrowIfNull(requestModel,
nameof(requestModel));

     var containerClient = client.
     GetBlobContainerClient(requestModel.ContainerName);

     var blobClient = containerClient.
     GetBlobClient(requestModel.RequiredFilePath);

     var downloadedContent =  await blobClient.
     DownloadContentAsync().ConfigureAwait(false);
     return downloadedContent.Value.Content

}
```

З метою зручної реєстрації служб у контейнері реалізацій застосовано метод розширення. Статичний клас, що містить метод, який реєструє необхідні реалізації в контейнері. Додатково використовується класс налаштування, який забезпечує розширену конфігурацію та гнучке налаштування залежностей, включаючи вибір механізму автентифікації та реєстрацію сервісів. Для обробки помилок, пов’язаних із відсутністю файлів у сховищі, передбачено класс, який дозволяє створювати спеціалізовані виключення з інформативними повідомленнями. Отже, таким чином, спроєктований рівень забезпечує модульний, масштабований і розширюваний механізм взаємодії з Azure Blob Storage, що дозволяє ефективно інкапсулювати логіку доступу до хмарного сховища в корпоративних застосунках.

У складі архітектури ПЗ було реалізовано окремий бізнес-рівень або «BusinessLayer», основною метою якого є інкапсуляція логіки обробки XML документів, їх валідація за схемами XML Schema Definition (XSD), а також взаємодія з зовнішніми джерелами ресурсів. Архітектура бізнес-рівня розроблена з урахуванням принципів розширюваності та повторного використання коду, що досягається завдяки активному використанню абстрактних класів, інтерфейсів, а також шаблонів проєктування.

Центральне місце в бізнес-рівні займає система валідації XML-документів, реалізована через окремий класс, який координує процес перевірки документів, взаємодіючи з колекцією стратегій, представлених у вигляді реалізацій абстрактного класу. Цей підхід забезпечує підтримку кількох типів документів із можливістю застосування до кожного з них власної логіки обробки. Реалізація стратегії валідації приведена у лістингу 4.7.

Лістинг 4.7 – Реалізація стратегії валідації (файл DtdXmlDocumentValidationStrategy.cs)

```
public override bool CanProcess(Stream documentStream)
{
     var xmlDocument = CreateDocument(documentStream);
     var documentType = xmlDocument.DocumentType;

     return documentType?.
     PublicId != null || documentType?.SystemId != null;
}
```

Для зберігання та отримання XML-документів використовується окремий класс, який здійснює пошук і перевірку існування документів у зовнішньому сховищі. Ресурси, необхідні для валідації, можуть надходити з зовнішніх джерел, зокрема через компоненти постачальників, що реалізують логіку доступу до зовнішніх ресурсів за допомогою різних стратегій. Окрему роль відіграє компонент валідатору, який безпосередньо виконує перевірку документів, створюючи відповідний XML-процесор на основі отриманих налаштувань. Реалізації функції валідації документу приведена у лістингу 4.8.

Лістинг 4.7 – Реалізація стратегії валідації (файл XmlValidationManager.cs)

```
public async Task<ValidationResult>
ValidateDocumentAsync(string documentFullPath, Stream
documentStream)
{
     var requiredStrategy = strategies
     .Where(validator => validator.CanProcess(documentStream))
     .FirstOrDefault();
```

Кінець лістингу 4.7

```
 if(requiredStrategy == null)
     throw new ValidationStrategyNotFoundException();

var strategyType = requiredStrategy.GetType();
var settingsProvider = this.settingsProviders
     .Where(provider => provider.GetType().
     GetInterfaces().Any(type => type ==
typeof(IValidationXmlSettingProvider<>)
     .MakeGenericType(strategyType)))
     .FirstOrDefault();

var requiredValidationSettings = await settingsProvider.
CreateSettingsAsync
     (documentFullPath, ValidationEventHandler);

await requiredStrategy.
ProcessAsync(documentStream, requiredValidationSettings);

return this.validationResult;
}
```

Таким чином, бізнес-рівень є ядром системи валідації документів та гнучкого керування зовнішніми ресурсами. Його реалізація надає можливість масштабування, розширення типів документів і стратегій їх обробки без потреби змінювати існуючу реалізацію, що відповідає принципам відкритості-закритості та інверсії залежностей.

4.4 Опис мікросервісів «CoreSoultion-DocumentManager» та «CoreSolution-Publisher»

Мікросервіси «CoreSoultion-DocumentManager» та «CoreSolution-Publisher» мають спільну архітектуру, яка дозволяє використовувати всі властивості та особистості платформи .NET для побудови гнучких та масштабованих сервісів. Архітектура складається з декількох модулів, які об’єднані між собою для забезпечення слабої зв’язаності компонентів. Основні компоненти архітектури мікросервісів зображені на рисунку 4.3:

– рівень «Abstract» – надає базові моделі даних усім рівням системи;

– рівень «Contracts» – надає контракти для реалізації компонентів системи;

– рівень «BusinessLayer» – надає реалізації компонентів за контрактом з бізнес-логікою для вирішення задач;

– рівень «Server» – фактичний серверний застосунок, який обробляє запити користувачів.

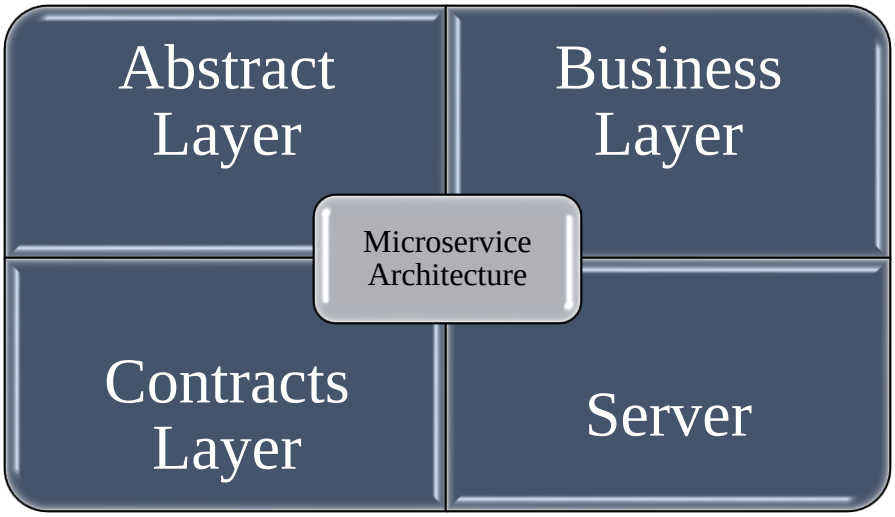

Рисунок 4.3 – Архітектура мікросервісів «DocumentManager» та «Publisher»

Мікросервіс «CoreSoultion-DocumentManager» забезпечує можливість виконувати так операції над документами, як створення, оновлення видалення та перегляд. Це фундаментальні операції, які необхідні для роботи системи валідації електронних фармацевтичних документів, оскільки вони надають змогу швидко керувати будь-яким документом в системі. Створення окремого сервісу, відповідального за всю роботу з документами, дозволяє централізувати всю бізнес-логіку, яка відноситься до управління документами, що дозволяє забезпечити високий рівень масштабування сервісу. Ключовою задачею сервісу є надання основних та безпечних операцій для роботи з документами, що особливо важливо при роботи с фармацевтичними даними, де потрібна висока доступність та надійність системи. Головною перевагою є безпечна та гнучка інтеграція з такими технологіями, як Server Message Block (SMB). Реалізація операції зчитування файлу документу з використанням SMB наведено у лістингу 4.8.

Лістинг 4.8 – Реалізація методу зчитування файлу за допомогою SMB (файл SMBFileStorage.cs)

```
private MemoryStream ReadSharedFile(object fileHandler){
     var memoryStream = new MemoryStream();
     var bytesRead = 0;
     while (true)
     {
          var status = fileStorage.ReadFile(out var data,
     fileHandler, bytesRead, options.MaxReadSize);
          if (status == NTStatus.STATUS_END_OF_FILE ||
data.Length == 0) break;
          bytesRead += data.Length;
          memoryStream.Write(data, 0, data.Length);
      }
     return memoryStream;}
```

Відповідальний за роботу з SMB модуль складається з усіх основних класів для підтримки роботи цього протоколу у системі, таким чином, мікросервіс «Document-Manager» інтегрується з технології, яка може обробляти декілька документів, які є спільними для всіх співробітників в організації, що значно спрощує керування складними ієрархіями документів у системі. Основні класи модулю SMB наведені на рисунку 4.4.

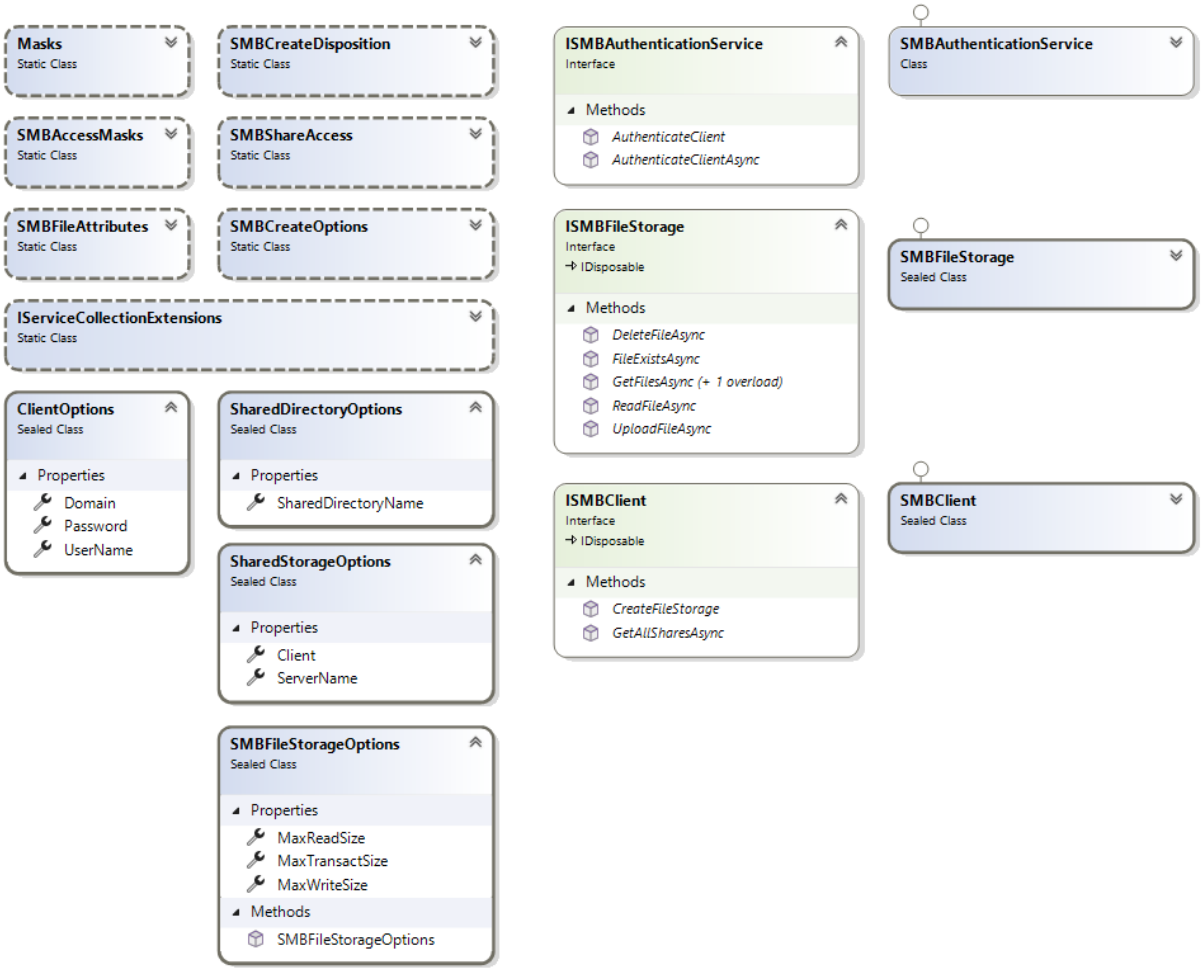

Рисунок 4.4 – Діаграма класів модулю SMB

Загалом, мікросервіс «CoreSoultion-DocumentManager» забезпечує роботу з ієрархією документів, що може бути збережена або в Azure Blob Storage, або Azure File Share. Ключовим аспектом створення мікросервісу є можливість додавати нові місця для збереження. Це підтверджує

властивість масштабування сервісу, що є критично важливим при розробці системи для валідації електронних фармацевтичних документів.

Мікросервіс «CoreSoultion-Publisher» надає функціонал публікування документа до централізованого ізольованого сховища документів, яке присутнє у кожного клієнта системи. Основний фунціонал сервісу забезпечує можливість безпечно пройти попередній процес валідації документу перед публікацією до сховища. Публікація документу є досить складним процесом, який треба виконувати з використанням приватної мережі для створення границі безпеки при передачі фармацевтичної документації, а головною задачею сервісу є гарантування безпечної публікації документу. Інкапсуляція цього процесу в окремий сервіс дозволяє досягти модульності та масштабованості архітектури, що є критично важливим аспектом у сфері обробки фармацевтичних документів, оскільки необхідно постійно підтримувати актуальність системи, використовуючи інтеграції з різними сучасними технологіями. Реалізація процесу публікації наведена у лістингу 4.9.

Лістинг 4.9 – Реалізація процесу публікації (файл SMBFileStorage.cs)

```
    private async Task CreateSharedFileFromStreamAsync(object
fileHandler, Stream fileStream)
    {
        var writeOffset = 0;
        while (fileStream.Position < fileStream.Length)
        {
            var buffer = new byte[options.MaxWriteSize];
            var bytesReadCount = await
fileStream.ReadAsync(buffer, 0, buffer.Length);
            if (bytesReadCount < options.MaxWriteSize)
                Array.Resize(ref buffer, bytesReadCount);
        var status = fileStorage.WriteFile(out var
```

Кінець лістингу 4.9

```
numberBytesWritten, fileHandler, writeOffset, buffer);
UnableUpdateSharedFileException.ThrowIfFailed(status);
            writeOffset += bytesReadCount;
        }
}
```

4.5 Реалізація комбінованої моделі розгортання та експериментальна перевірка комбінованої моделі розгортання

Комбінована модель розгортання реалізується за допомогою конфігураційних файлів Yet Another Markup Language (YAML) для опису інфраструктури. Для декларативного створення конвеєру розгортання необхідно визначати основні задачі, які буде вирішувати модель розгортання мікросервісу комбінованої моделі, а саме:

- отримання актуальних даних з репозиторія;
- збірка мікросервісу;
- конфігурація сервісу для релізу;
- створення артефакту;
- публікація артефакту до централізованого сховища артефактів;
- запуск нового конвеєру розгортання;
- отримання останнього артефакту;
- підключення до середи розгортання;
- оновлення середи, використовуючи нову кодову базу;
- перевірка застосунку (Health Check);
- перемикання трафіку.

Для оптимізації процесу створення інфраструктури системи

необхідно реалізувати новий конвеєр, який має змогу підключитися до організації в хмарі Azure, а потім відтворити нові ресурси інфраструктури, яку було задекларовано в репозиторії. Зображення головного репозиторію, відповідального за розгортання інфраструктури зображено на рисунку 4.5.

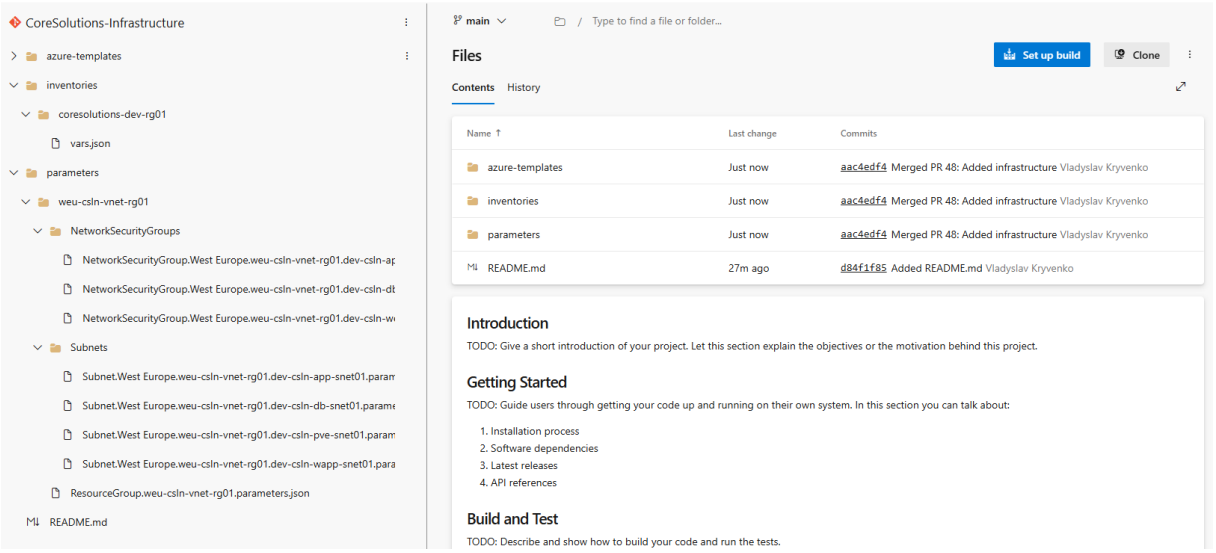

Рисунок 4.5 – Зображення центрального репозиторію, відповідального за конфігурацію інфраструктури

Комбінована модель надає одну ключову перевагу, яка визначає центральне місце збереження конфігурації для інфраструктури системи. Декларативний спосіб визначення інфраструктури дає змогу створити все налаштування, використовуючи YAML-файли з описом усіх необхідних хмарних компонентів. Це забезпечує прозорість, повторюваність та контрольованість процесів розгортання, знижує ймовірність помилок, спричинених людським фактором, та підвищує рівень автоматизації процесів розгортання. Визначення мікросервісів для розгортання за допомогою декларативного підходу наведено у лістингу 4.10.

Лістинг 4.10 – Визначення мікросервісів для розгортання (файл vars.yml)

```
"AppService_documentPublisher": {
     "name": "dev-csln-documentpublisher-wapp01",
     "currentStack": "dotnet",
     "FxVersion": "DOTNETCORE|8.0",
     "ins_webAppVersion": "1.1.3",
     "ins_webAppSettings": []
},
"PrivateEndpoint_documentPublisher": {
     "peName": "dev-csln-documentpublisher-wapp01-pve01",
     "peResourceName": "dev-csln-documentpublisher-wapp01"
},
"AppService_documentManager": {
     "name": "dev-csln-AppService_documentManager-wapp01",
     "currentStack": "dotnet",
     "FxVersion": "DOTNETCORE|8.0",
     "ins_webAppVersion": "1.1.3",
     "ins_webAppSettings": []
},
"PrivateEndpoint_documentManager": {
     "peName": "dev-csln-AppService_documentManager-wapp01-
     pve01",
     "peResourceName": "dev-csln-AppService_documentManager-
     wapp01"
},
"AppService_validator": {
     "name": "dev-csln-AppService_validator-wapp01",
     "currentStack": "dotnet",
     "FxVersion": "DOTNETCORE|8.0",
     "ins_webAppVersion": "1.1.3",
     "ins_webAppSettings": []
},
```

Кінець лістингу 4.10

```
"PrivateEndpoint_validator": {
     "peName": "dev-csln-AppService_validator-wapp01-pve01",
     "peResourceName": "dev-csln-AppService_validator-wapp01"
},
"AppService_userManager": {
     "name": "dev-csln-AppService_validator-wapp01",
     "currentStack": "dotnet",
     "FxVersion": "DOTNETCORE|8.0",
     "ins_webAppVersion": "1.1.3",
     "ins_webAppSettings": []
},
"PrivateEndpoint_userManager": {
     "peName": "dev-csln-AppService_usermanager-wapp01-pve01",
     "peResourceName": "dev-csln-AppService_usermanager-wapp01"
}
```

У контексті комбінованої моделі, яка передбачає централізоване керування конфігурацією інфраструктури, створення конвеєру збірки для окремого мікросервісу відіграє ключову роль у забезпеченні безперервної інтеграції та підвищення загальної стабільності. Цей конвеєр автоматизує процес збирання та підготовки артефактів мікросервісу до подальшого процесу розгортання. Це дозволяє гарантувати, що кожна зміна в кодовій базі мікросервісу проходить через стандартизований процес перевірки. Реалізація конвеєру збірки наведена у лістингу 4.11.

Лістинг 4.11 – Реалізація конвеєру збірки мікросервісу (файл build.yml)

```
stages:
  - stage: Building CoreSolutions-Validator Microservice
```

Продовження лістингу 4.11

```
    jobs:
      - job: ${{ parameters.jobName }}
        dependsOn: ${{ parameters.dependsOn }}
        condition: ${{ parameters.condition }}
        pool:
          vmImage: ${{ parameters.agentVmImage }}
        steps:
          - task: UseDotNet@2
            displayName: 'Installing .NET SDK'
            inputs:
              packageType: sdk
              version: ${{ parameters.dotnetVersion }}
              installationPath: $(Agent.ToolsDirectory)/dotnet

          - task: DotNetCoreCLI@2
            displayName: 'Restoring NuGet Packages'
            inputs:
              azureSubscription:
SkillupSubscriptionServiceConnection
              command: restore
              projects: ${{ parameters.projectSrcDir
}}/**/*.csproj
              vstsFeed: ea23c167-0a7e-4532-8a6d-3c2312e2108d
              includeNuGetOrg: false
              noCache: true

          - task: DotNetCoreCLI@2
            displayName: 'Building CoreSolutions-Validator
Microservice'
            inputs:
              command: build
              projects: ${{ parameters.projectSrcDir
```

Кінець лістингу 4.11

```
}}/**/*.csproj
              configuration: ${{ parameters.buildConfiguration
}}
              arguments: '--version-suffix ${{
parameters.assembliesVersion }}'

          - task: DotNetCoreCLI@2
            displayName: 'Publishing CoreSolutions-Validator
Microservice'
            inputs:
              command: publish
              zipAfterPublish: True
              publishWebProjects: True
              arguments: '--configuration ${{
parameters.buildConfiguration }} --output
$(Build.ArtifactStagingDirectory)'

          - task: PublishBuildArtifacts@1
            displayName: 'Publishing Build Artifact of
CoreSolutions-Validator Microservice'
            inputs:
              PathtoPublish: $(Build.ArtifactStagingDirectory)
              ArtifactName: NetCoreApplicationLast
              publishLocation: Container

          - task: mspremier.PostBuildCleanup.PostBuildCleanup-
task.PostBuildCleanup@4
            displayName: 'Cleaning Agent Directories'
            continueOnError: true
            condition: succeededOrFailed()
```

Конвеєр, відповідальний за розгортання мікросервісу, є основною складовою моделі процесу розгортання мікросервісу. Його створення має критичне значення для забезпечення надійного, відтворюваного та контрольованого процесу розгортання змін у продуктивне середовище. Реалізація конвеєру розгортання приведена у лістингу 4.12.

Лістинг 4.12 – Реалізація конвеєру розгортання мікросервісу (файл deploy.yml)

```
stages:
  - stage: Deploy
    displayName: "Deploing ASP.NET Core Application"
    jobs:
      - job: ${{ parameters.jobName }}
        dependsOn: ${{ parameters.dependsOn }}
        condition: ${{ parameters.condition }}
        pool:
          vmImage: ${{ parameters.agentVmImage }}
        steps:

        - task: DownloadBuildArtifacts@1
          inputs:
            displayName: 'Downloading Build Artifacts for
CoreSolutions-Validator Microservice'
            buildType: 'specific'
            project: 'ea23c167-0a7e-4532-8a6d-3c2312e2108d'
            pipeline: '9'
            buildVersionToDownload: 'latest'
            downloadType: 'single'
            artifactName: 'NetCoreApplicationLast'
            itemPattern:
            downloadPath: '$(System.ArtifactsDirectory)'
            cleanDestinationFolder: true
```

Продовження лістингу 4.12

```
        - task: ExtractFiles@1
          inputs:
            displayName: 'Extracting Files'
            archiveFilePatterns:
'$(System.ArtifactsDirectory)/**/*.zip'
            destinationFolder:
'$(System.ArtifactsDirectory)/application'
            cleanDestinationFolder: true
            overwriteExistingFiles: false

        - task: replacetokens@6
          inputs:
            displayName: 'Replacing Secrets for CoreSolutions-
Validator Microservice'
            root: '$(System.ArtifactsDirectory)/application/'
            sources: 'appsettings.json'
            tokenPattern: 'custom'
            tokenPrefix: '{'
            tokenSuffix: '}'
            logLevel: 'error'
            missingVarLog: 'error'
            ifNoFilesFound: 'error'

        - task: ArchiveFiles@2
          inputs:
            displayName: 'Archiving CoreSolutions-Validator
Microservice Build'
            rootFolderOrFile:
'$(System.ArtifactsDirectory)/application/'
            includeRootFolder: false
            archiveType: 'zip'
            archiveFile:
```

Кінець лістингу 4.12

```
'$(System.ArtifactsDirectory)/drop/$(Build.BuildId).zip'
            replaceExistingArchive: true

        - task: AzureRmWebAppDeployment@4
          inputs:
            displayName: 'Deploying CoreSolutions-Validator
Microservice to Azure Web App'
            ConnectionType: 'AzureRM'
            azureSubscription: 'CoreSoultions-Sub'
            appType: 'webApp'
            WebAppName: 'dev-csln-AppService_validator-wapp01'
            packageForLinux:
'$(System.ArtifactsDirectory)/**/$(Build.BuildId).zip'
```

Налаштування комбінованої моделі розгортання дає можливість розширяти модель, оскільки для її створення використовується декларативний підхід, що забезпечує гнучкість та масштабованість моделі.

Для розгортання мікросервісу необхідно зробити зміни у кодовій базі, як тільки змін надійдуть, почнеться процес безперервного розгортання. Спершу комбінована модель розгортання створить новий конвеєр для обробки змін у центральному сховищі. Результатом виконання конвеєру буде готова збірка мікросервісу, яка знаходиться у сховищі артефактів, яке використовується для подальшого розгортання. Детальний процес збрів відображений на рисунку 4.6.

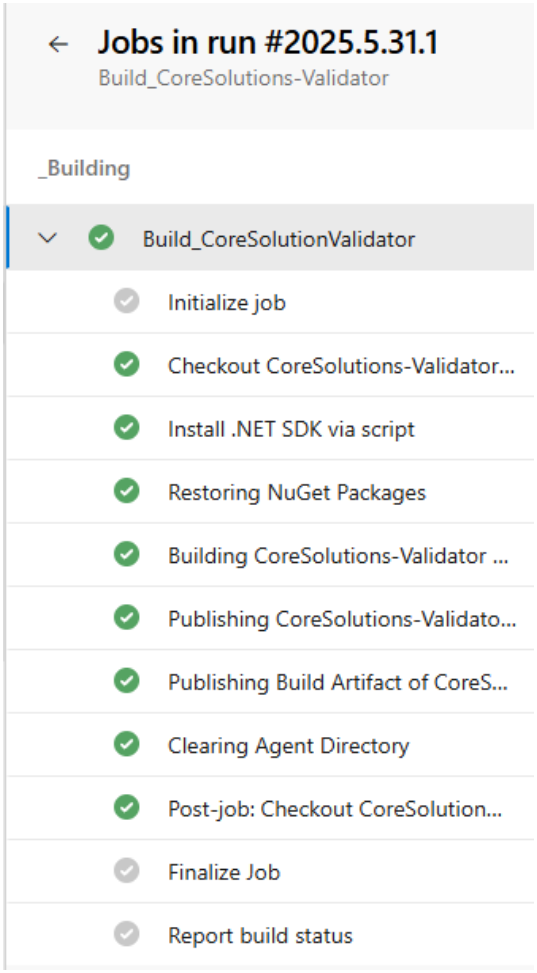

Рисунок 4.6 – Результат виконання конвеєру збірки

Наступним кроком комбінована модель визначить результат виконання конвеєру збірки, і, якщо артефакт мікросервісу був опублікований у сховище, розпочинає розгортання за допомогою конвеєру розгортання. Результатом виконання конвеєру розгортання, що зображено на рисунку 4.7, є мікросервіс, який працює за допомогою хмарних технологій, а саме – Azure Web App. А рисунок 4.8 відображає успішне розгортання мікросервісів за допомогою комбінованої моделі розгортання.

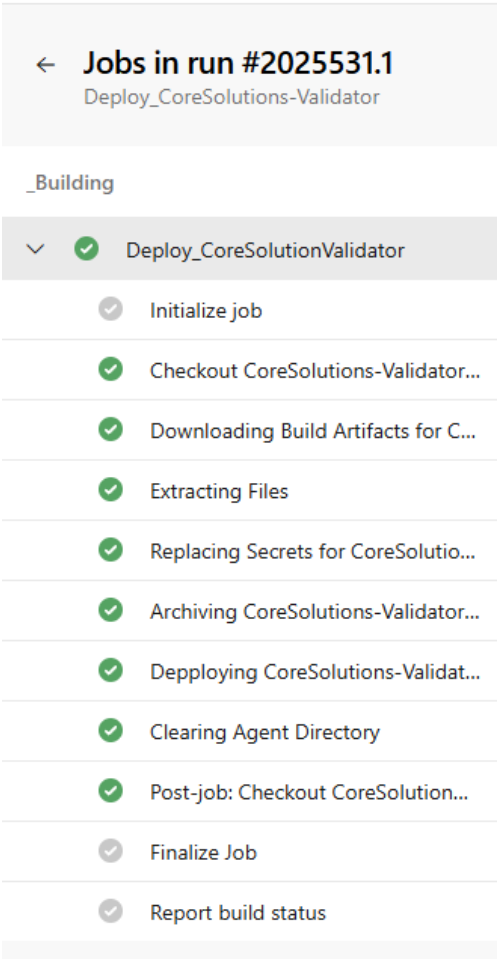

Рисунок 4.7 – Результат виконання конвеєру розгортання

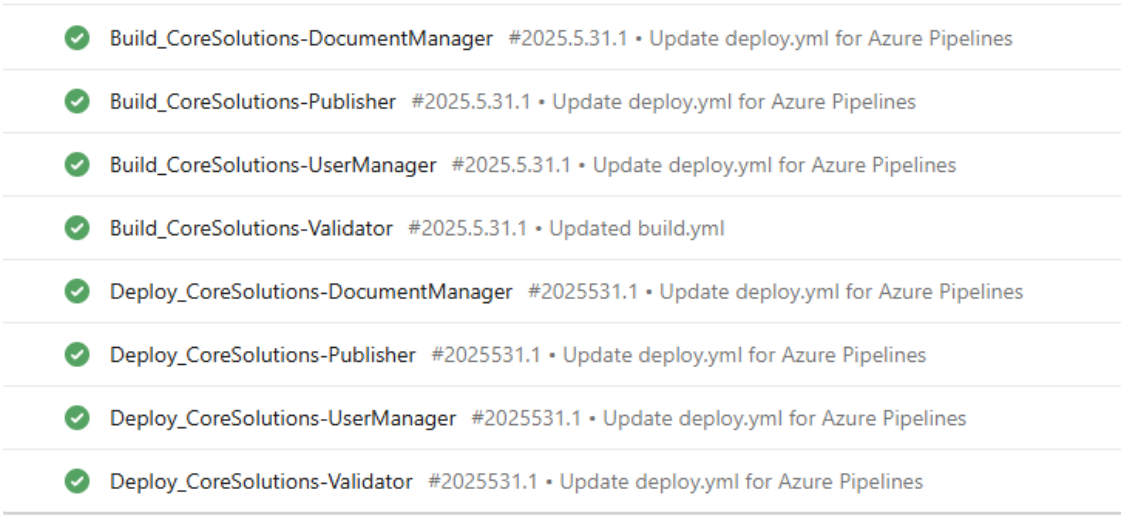

Рисунок 4.8 – Результат розгортання мікросервісів

Експериментальна перевірка комбінованої моделі розгортання дає можливість порівняти моделі розгортання мікросервісів в IT-проєктах валідації електронних фармацевтичних документів.

Результати аналізу різних моделей розгортання мікросервісів у середовищі Microsoft Azure показали, що розроблена комбінована модель розгортання забезпечує високу керованість, стабільність та мінімізацію простою системи, що є критичним для IT-проєктів валідації електронних фармацевтичних документів. Саме завдяки комбінованій моделі розгортання забезпечується безперервна доступність системи, не беручи до уваги Microsoft Azure обмеження, а використання GitOps дозволяє оптимізувати керування інфраструктурою та змінами через систему контролю версій.

Час повного розгортання сервісів у такій моделі становить від 3 до 5 хвилин, в залежності від типу сервісу, при цьому система не зазнає простою, оскільки перемикання трафіку відбувається після повної готовності нового середовища. Повне розгортання всієї інфраструктури за допомогою GitOps у поєднанні з Microsoft Azure становить 38-45 хвилин, що є швидшим порівняно з іншими моделями розгортання, завдяки автоматизації та декларативному опису інфраструктури, платформи та системи.

Таблиця 4.1 порівнює моделі за ключовими особливостями. Результатом розгортання є система, яка може виконувати валідацію електронних фармацевтичних документів, графічний інтерфейс якої наведено на рисунку 4.9.

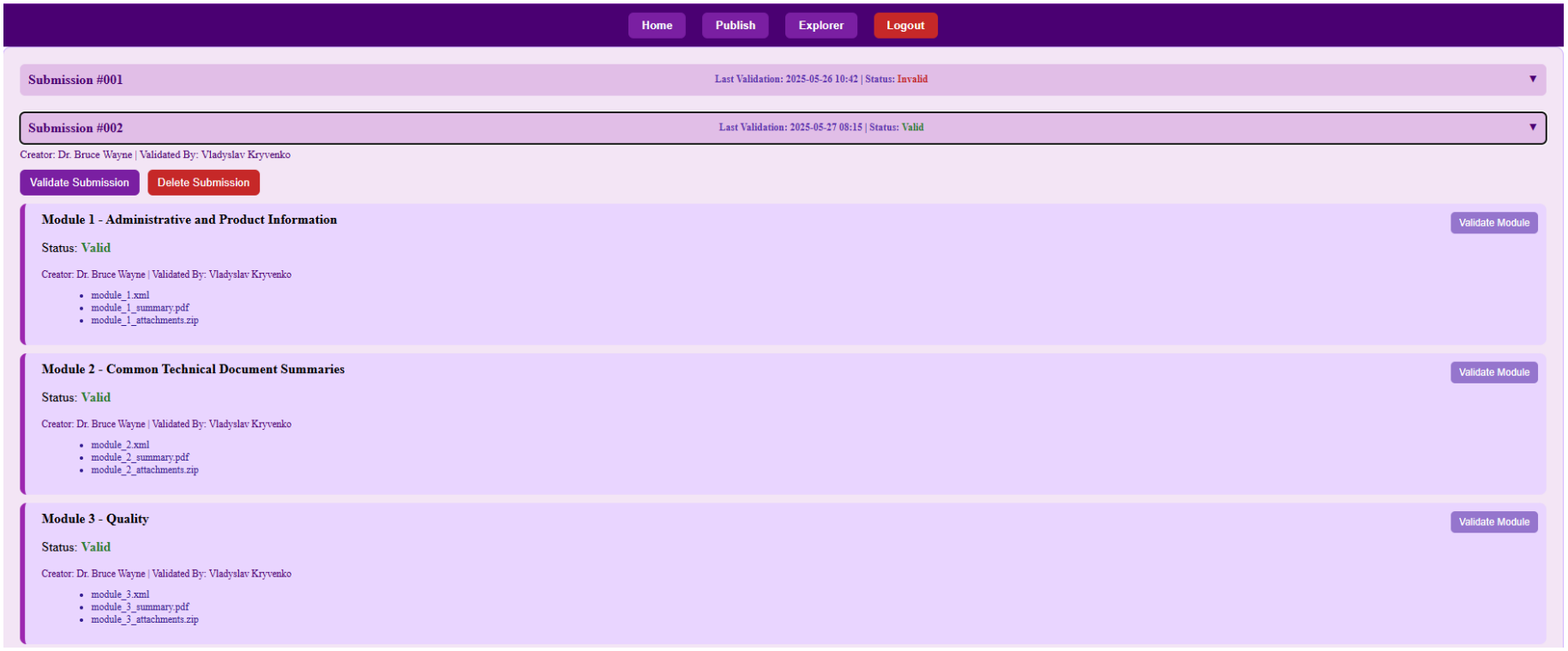

Рисунок 4.9 – Графічний інтерфейс системи

Таблиця 4.1 – Експериментальні результати порівняння моделей розгортання

Модель	Час розгортання сервісів	Орієнтований час простою	Автоматизація розгортання
Монолітне розгортання	60-120 хвилин (в залежності від наявності помилок)	60-120 хвилин (в залежності від наявності помилок)	Відсутня. Все виконується мануально
Почергове розгортання	20-30 хвилин	1-3 хвилини	Оновлення відбуваються поетапно
Інкрементальне розгортання	15-25 хвилин	1-3 хвилини	Оновлюються лише змінені частини застосунку
Комбінована модель	10-15 хвилин	до 1 хвилини	Два середовища працюють паралельно, розгортання сервісів описано декларативним шляхом. Вся конфігурація інфраструктури може бути розгорнута знову та швидко

ВИСНОВКИ

У магістерський кваліфікаційній роботі вирішено задачу вдосконалення моделей розгортання IT-проєктів, орієнтованих на мікросервісну архітектуру, з огляду на такі недоліки чинних моделей, як складність управління інфраструктурою при наявності великої кількості мікросервісів, складність масштабування, високий ризик збоїв, тривалий час розгортання та ускладнене балансування мережевого трафіку. Особливо вирішення цих проблем є важливим для сфери автоматизованої обробки фармацевтичної комерції фармацевтичних документів, де спостерігаються наступні обмеження, зокрема відсутність інтеграції з різними медичними системами, високий рівень вимог до безпеки зберігання або передачі медичних даних та складність забезпечення відповідності нормативним вимогам.

Для досягнення мети дослідження вирішено задачі аналізу процесів розгортання мікросервісів, дослідження структури процесу розгортання мікросервісів, аналізу моделей розгортання мікросервісів, аналізу валідації фармацевтичних документів, розробки комбінованої моделі розгортання мікросервісів.

Отримано комбіновану модель паралельного розгортання з перемиканням мережевого трафіку та використанням декларативного підходу GitOps для розгортання мікросервісів. Перевагами комбінованої моделі є повна відтворюваність та контроль, підвищена надійність та зменшення ризиків, автоматизація процесу розгортання, безперервна доставка без витрат доступності, безпечне впровадження нових функцій, що є критично важливо в IT-проєктах валідації електронних фармацевтичних документів.

Розроблений проєкт системи валідації електронних фармацевтичних

документів з використанням комбінованої моделі розгортання мікросервісів включає в себе ініціалізацію та розробку концепції автоматизації валідації електронних фармацевтичних документів, розробку життєвого циклу системи, структуризацію проєкту, проєктування системи валідації електронних фармацевтичних документів та керування часом, вартістю та ресурсами проєкту.

Розроблено систему валідації електронних фармацевтичних документів, що орієнтована на хмарні технології, та характеризується наступними модулями: модуль управління документами, модуль публікації документів, модуль валідації фармацевтичних документів та модуль управління користувачами в системі.

Експериментальна перевірка комбінованої моделі розгортання надає детальну різницю між різними моделями розгортання, і підкреслює переваги комбінованої моделі розгортання мікросервісів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. OpsLevel Team. 4 Microservice Deployment Patterns That Improve Availability. URL: https://www.opslevel.com/resources/4-microservice-deployment-patterns-that-improve-availability (дата звернення: 23.04.2025).

2. Simform Team. How Does Microservices Architecture Work? A Complete Guide. URL: https://www.simform.com/blog/how-does-microservices-architecture-work/ (дата звернення:23.04.2025).

3. KongHQ Team. Essential Guide to Understanding Microservices Architecture. URL: https://konghq.com/blog/learning-center/what-are-microservices (дата звернення: 24.04.2025).

4. AutoMQ Team. Fully Automated Deployment vs. Manual Deployment. URL: https://www.automq.com/blog/fully-automated-deployment-vs-manual-deployment-comparison (дата звернення: 25.04.2025).

5. Codegenitor. Manual Deployment Mayhem: A Step-by-Step Guide to Automate DevOps Pipeline. URL: https://codegenitor.medium.com/manual-deployment-mayhem-a-step-by-step-guide-to-automate-your-devops-pipeline-19d3fa43fce7 (дата звернення: 25.04.2025).

6. Acronis Team. The Ultimate Guide to Automated Software Deployment for MSPs and IT Teams. URL: https://www.acronis.com/en-us/blog/posts/automated-software-deployment-tools/ (дата звернення: 01.04.2025).

7. DeployHQ Team. The Ultimate Guide to Streamlined Deployments. URL: https://www.deployhq.com/blog/the-ultimate-guide-to-streamlined-deployment (дата звернення: 26.04.2025).

8. Daffodil Software. Automated CI/CD vs. Manual Launch: Optimizing. URL: https://insights.daffodilsw.com/blog/automated-ci/cd-vs.-manual-launch-optimizing-software-deployment (дата звернення: 01.04.2025).

9. Carlos Arguelles. Blue-Green Deployment from the Trenches. URL: https://www.infoq.com/articles/blue-green-deployments/ (дата звернення: 26.04.2025).

10. ICH Harmonization for better health. CTD. URL: https://www.ich.org/page/ctd (дата звернення: 03.05.2025).

11. ICH Harmonization for better health. Welcome to the ICH. URL: https://www.ich.org/ (дата звернення: 03.05.2025).

12. U.S. Food and Drug. Electronic Common Technical Document. Published on 01.10.2024 URL: https://www.fda.gov/drugs/electronic-regulatory-submission-and-review/electronic-common-technical-document-ectd (дата звернення: 04.05.2025).

13. Christian Posta. Blue-Green Deployments, A/B Testing, and Canary Releases. URL: https://blog.christianposta.com/deploy/blue-green-deployments-a-b-testing-and-canary-releases/ (дата звернення: 05.05.2025).

14. Omkar Pasalkar. Unleashing the Power of Blue-Green Deployments: A Deep Dive. URL: https://www.linkedin.com/pulse/unleashing-power-blue-green-deployments-deep-dive-omkar-pasalkar-1mwgf (дата звернення: 05.05.2025).

15. Stine J. A. Digital Validation of Pharmaceutical Data Using Cloud Microservices. URL: https://www.researchgate.net/publication/345276517 (дата звернення: 07.05.2025).

16. Atlassian. Deployment Automation: What Is It & How to Start URL: https://www.atlassian.com/devops/frameworks/deployment-automation дата звернення: 07.05.2025).

17. Spot.io Team. Understanding GitOps: Principles, Workflow, and Deployment Types. URL: https://spot.io/resources/gitops/understanding-gitops-principles-workflows-deployment-types/ (дата звернення: 09.05.2025).

18. Keerthi K. Introduction to GitOps: Simplifying Deployment workflow. URL: https://medium.com/@iamkeerthik/introduction-to-gitops-simplifying-deployment-automation-de082ae01707 (дата звернення: 09.05.2025).

19. Navya CloudOps. DevOps Zero to Hero – Day 20: Deployment Strategies. URL: https://medium.com/@navya.cloudops/devops-zero-to-hero-day-20-deployment-strategies-e6712b4801e4 (дата звернення: 09.06.2025).

20. Google Cloud Team. Google Cloud Hybrid Deployment Archetype URL: https://cloud.google.com/architecture/deployment-archetypes/hybrid (дата звернення: 10.05.2025).

21. Atlassian Team. IT Project Management: Tools, and Best Practices URL: https://www.atlassian.com/work-management/project-management/it-project-management (дата звернення: 12.05.2025).

22. Управління ризиками під час впровадження інформаційних систем та технологій URL: https://moz.gov.ua/uk/upravlinnya-rizikami-pid-chas-vprovadzhennya-informacijnih-sistem-ta-tehnologij (дата звернення: 12.05.2025).

23. Борисов О.В, Данченко О.Б, Харута В.С Технологія вибору ефективної методології управління IT-проєктом. Вісник Національного технічного університету «ХПІ» Серія Стратегічне управління, управління портфелями, програмами та проектами. 2022. № 2(6) С. 7–13. URL: https://doi.org/10.20998/2413-3000.2022.6.2 (дата звернення: 13.05.2025).

24. Катренко А. В. Управління IT-проєктами. Книга 1. Стандарти, моделі та методи управління проєктами: підручник. Львів: Новий Світ – 2000, 201 (дата звернення: 13.05.2025).

26. Cloud Foundry Documentation. Using blue-green deployment. Published on 23.10.2021. URL: https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html (дата звернення: 14.05.2025).

27. Assis Zang. ASP.NET Basics: ASP.NET Core Overview. Published in 25.05.2023. URL: https://www.telerik.com/blogs/aspnet-core-basics-aspnetcore-overview (дата звернення: 14.05.2025).

28. Mahesh Sabnis. Using Entity Framework Core to access Stored Procedures having User Defined table Types as Parameters. URL:

https://www.devcurry.com/2019/08/stored-procedures-udt-types-ef-core.html = (дата звернення: 15.05.2025).

29. George Rus. Web Resources Access Protection Using ASP.NET Identity. URL: https://www.todaysoftmag.com/article/1591/web-resourcesaccess-protection-using-asp-net-identity (дата звернення: 16.04.2023).