# БАКАЛАВРСЬКА РОБОТА

БР. ІП - 22.00.00.000 ПЗ

**Група ІП-21-2**

**Вирстюк Роман**

**2025**

**Івано-Франківський національний технічний університет нафти і газу**

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

**Вирстюк Роман Ярославович**

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

# БАКАЛАВРСЬКА РОБОТА

**Імплементація SOA на рівні корпоративних сервісів для підвищення гнучкості інтеграції**

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього рівня Вирстюк Р.Я.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Піх Марія Михайлівна, асистент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

**Допущено до захисту**
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

**Івано-Франківськ – 2025**

**Івано-Франківський національний технічний університет нафти і газу**

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ:**
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

# ЗАВДАННЯ

## НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

**Вирстюку Роману Ярославовичу**

(прізвище, ім’я, по-батькові)

**1. Тема проекту (роботи) “ Імплементація SOA на рівні корпоративних сервісів для підвищення гнучкості інтеграції”**

керівник проекту (роботи) Піх М.М., асистент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

**2. Строк подання студентом проекту (роботи)** 10 червня 2025 р.

**3. Вихідні дані до проекту (роботи)** Результати і матеріали отримані під час проходження переддипломної практики

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

1. Аналіз предметної області використання SOA на рівні корпоративних сервісів
2. Інженерія вимог та фази розробки програмного забезпечення
3. Сервісно-орієнтована архітектура та системна інтеграція додатків корпоративного рівня
4. Системна інтеграція та роль Enterprise Service Bus
5. Імплементація SOA на рівні корпоративних сервісів для підвищення гнучкості інтеграції

**5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)**

1. Високорівнева схема топології системи (рис. 1.1)
2. Життєвий цикл розробки програмного забезпечення (рис. 1.2)
3. Спіральна модель процесу інженерії вимог (рис. 1.3)
4. Етапи сервісної інженерії (рис. 1.4)
5. Базова функціональність SOA (рис. 1.5)

**6. Консультанти розділів проекту (роботи)**

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

**7. Дата видачі завдання** **05 травня 2025 р.**

**Керівник** _____
(підпис)

**Завдання прийняв до виконання** _____
(підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проекту (роботи) | Строк виконання етапів проекту | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області використання SOA на рівні корпоративних сервісів | 07.05.2025 | виконано |
| 2 | Інженерія вимог та фази розробки програмного забезпечення | 17.05.2025 | виконано |
| 3 | Сервісно-орієнтована архітектура та системна інтеграція додатків корпоративного рівня | 27.05.2025 | виконано |
| 4 | Системна інтеграція та роль Enterprise Service Bus | 02.06.2025 | виконано |
| 5 | Імплементація SOA на рівні корпоративних сервісів для підвищення гнучкості інтеграції | 05.06.2025 | виконано |
| 6 | Оформлення пояснювальної записки дипломної роботи завідувачем кафедри | 10.06.2025 | виконано |

**Студент – дипломник** _____
(підпис)

**Керівник роботи** _____
(підпис)

# АНОТАЦІЯ

Бакалаврська робота містить 77 сторінок, 27 рисунків, список використаних джерел із 37 найменуваннями, 1 додаток.

**Мета роботи** полягає у дослідженні, моделюванні та реалізації інтеграційного рішення на базі SOA з використанням ESB на рівні корпоративних сервісів.

**Об'єкт дослідження** - процеси інтеграції корпоративних інформаційних систем у межах великої організації.

**Предмет дослідження** - методи, засоби та принципи впровадження сервісно-орієнтованої архітектури на базі технологій SOA та ESB для інтеграції корпоративних сервісів.

**В першому** розділі проведено опис системної інтеграції із застосуванням SOA і ESB дозволяє забезпечити ефективне поєднання розподілених сервісів у рамках складних корпоративних інформаційних систем

**В другому розділі** показано використання веб-сервісів та ESB як інструментів реалізації SOA забезпечує стандартизовану, масштабовану та незалежну від технологій модель інтеграції.

**В третьому розділі** описана імплементація розробленої SOA-архітектури яка підтвердила її ефективність у підвищенні гнучкості та керованості інтеграційних процесів на рівні корпоративних сервісів.

**Висновок:** у роботі досліджено концептуальні та практичні аспекти впровадження сервісно-орієнтованої архітектури (SOA) на рівні корпоративних сервісів із використанням шини підприємства (Enterprise Service Bus, ESB)

**КЛЮЧОВІ СЛОВА:** СИСТЕМНА ІНТЕГРАЦІЯ, СЕРВІСНО-ОРІЄНТОВАНА АРХІТЕКТУРА, SOA, ENTERPRISE SERVICE BUS, ESB, КОРПОРАТИВНІ СЕРВІСИ, ІНЖЕНЕРІЯ ВИМОГ, ВЕБ-СЕРВІСИ

# ANNOTATION

The bachelor's thesis contains 77 pages, 27 figures, a list of used sources with 37 names, 1 appendix.

**The purpose of the work is to study**, model and implement an integration solution based on SOA using ESB at the corporate services level.

**The object of the research** is the processes of integration of corporate information systems within a large organization.

**The subject of the research** is methods, tools and principles of implementing a service-oriented architecture based on SOA and ESB technologies for the integration of corporate services.

**The first section** describes system integration using SOA and ESB, which allows for the effective combination of distributed services within complex corporate information systems.

**The second section** shows the use of web services and ESB as tools for implementing SOA, which provides a standardized, scalable and technology-independent integration model.

**The third section** describes the implementation of the developed SOA architecture, which confirmed its effectiveness in increasing the flexibility and manageability of integration processes at the corporate services level.

**Conclusion:** The paper explores the conceptual and practical aspects of implementing service-oriented architecture (SOA) at the corporate services level using the Enterprise Service Bus (ESB)

**KEYWORDS:** SYSTEM INTEGRATION, SERVICE-ORIENTED ARCHITECTURE, SOA, ENTERPRISE SERVICE BUS, ESB, ENTERPRISE SERVICES, REQUIREMENTS ENGINEERING, WEB SERVICES

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SOA - Service-Oriented Architecture

ESB - Enterprise Service Bus

ERP - Enterprise Resource Planning

BPEL - Business Process Execution Language

WSDL - Web Services Description Language

XML - Extensible Markup Language

REST - Representational State Transfer

SOAP - Simple Object Access Protocol

TOGAF - The Open Group Architecture Framework

SOMA - Service-Oriented Modeling and Architecture

# ВСТУП

У сучасних умовах цифрової трансформації бізнесу організації стикаються з проблемою ефективної взаємодії між гетерогенними інформаційними системами. Потреба в забезпеченні надійної, гнучкої та масштабованої інтеграції стала критично важливою для підвищення продуктивності бізнес-процесів і адаптації до змін. Сервісно-орієнтована архітектура (SOA) у поєднанні з Enterprise Service Bus (ESB) пропонує концептуальну та технологічну основу для досягнення цієї мети. Водночас, впровадження SOA вимагає глибокого аналізу предметної області, чіткої інженерії вимог, методично обґрунтованого проєктування та поетапної реалізації. Це зумовлює актуальність дослідження, спрямованого на розробку інтеграційного рішення корпоративного рівня на базі SOA, що дозволяє ефективно поєднати існуючі додатки, зменшити технологічну фрагментацію та підвищити адаптивність IT-інфраструктури.

Інтеграція корпоративних інформаційних систем — один з ключових викликів, що стоїть перед сучасними організаціями в умовах стрімкого розвитку інформаційних технологій. Різнорідність платформ, мов програмування, архітектурних рішень та бізнес-вимог породжує складнощі у забезпеченні безперебійної взаємодії між окремими компонентами корпоративної IT-інфраструктури. З огляду на це, все більше компаній звертаються до концепції сервісно-орієнтованої архітектури (SOA), яка дозволяє створювати гнучкі, масштабовані та повторно використовувані рішення для системної інтеграції.

Особливе місце в реалізації SOA займає технологія Enterprise Service Bus (ESB), яка функціонує як посередник між сервісами, забезпечуючи маршрутизацію повідомлень, трансформацію даних, управління безпекою та моніторинг. Такий підхід дозволяє не лише знизити загальну складність IT-середовища, а й спростити його розвиток і супровід.

**Актуальність роботи**

У сучасних умовах цифрової трансформації компанії дедалі частіше стикаються з необхідністю об'єднання розрізнених інформаційних систем і сервісів у єдину цілісну інфраструктуру. Сервісно-орієнтована архітектура (SOA) та інструменти системної інтеграції, зокрема Enterprise Service Bus (ESB), стають критично важливими для побудови гнучких, масштабованих та адаптивних рішень. Саме тому дослідження можливостей впровадження SOA на рівні корпоративних сервісів є вкрай актуальним з огляду на зростаючі вимоги до ефективності бізнес-процесів, зниження витрат на розробку та підтримку програмного забезпечення, а також забезпечення гнучкої інтеграції в гетерогенних IT-середовищах.

**Мета цієї роботи** полягає у дослідженні, моделюванні та реалізації інтеграційного рішення на базі SOA з використанням ESB на рівні корпоративних сервісів.

Для досягнення поставленої мети було проаналізовано теоретичні основи архітектурних рішень, розглянуто життєвий цикл програмного забезпечення, виконано інженерію вимог, розроблено концептуальну модель, спроєктовано і реалізовано окремі етапи інтеграції, а також здійснено валідацію результатів.

**Завдання дослідження**

- Проаналізувати сучасні підходи до системної інтеграції та ролі SOA.
- Дослідити архітектурні принципи SOA та ESB у контексті корпоративного програмного забезпечення.
- Розробити концептуальну модель інтеграційного рішення.
- Виконати інженерію вимог до системи.
- Реалізувати прототип системної інтеграції на основі SOA.
- Провести валідацію та аналіз ефективності запропонованого рішення.

**Об’єкт дослідження** - процеси інтеграції корпоративних інформаційних систем у межах великої організації.

**Предмет дослідження** - методи, засоби та принципи впровадження сервісно-орієнтованої архітектури на базі технологій SOA та ESB для інтеграції корпоративних сервісів.

**Методи дослідження**

- Теоретичний аналіз наукових джерел;
- Моделювання програмних архітектур;
- Інженерія вимог;
- Проєктування та реалізація програмного забезпечення;
- Метод експертної оцінки результатів;
- Аналіз ефективності реалізованої архітектури.

**Наукова новизна**

Запропоновано методологію проєктування гнучкої SOA-архітектури для інтеграції корпоративних сервісів з урахуванням практичних вимог безпеки, масштабованості та адаптивності.

**Практичне застосування**

Результати дослідження можуть бути використані підприємствами для впровадження системної інтеграції, модернізації існуючих IT-систем, а також у навчальному процесі при викладанні дисциплін, пов’язаних з архітектурою ПЗ.

Бакалаврська робота містить 77 сторінок, 27 рисунків, 3 розділи список використаних джерел із 37 найменуваннями, 1 додаток.

# РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВИКОРИСТАННЯ SOA НА РІВНІ КОРПОРАТИВНИХ СЕРВІСІВ

## 1.1. Сервісно-орієнтована архітектура (SOA) та Enterprise Service Bus (ESB) як інструменти ефективної системної інтеграції

Підвищена залежність функціонування бізнес-процесів та організацій від їхніх інформаційних систем призвела до суттєвого зростання складності архітектури відповідної IT-інфраструктури. Ключовими факторами, що детермінують цю тенденцію, є, зокрема, процеси корпоративної консолідації (злиття та поглинання компаній) та прискорення динаміки ринкових змін, що вимагають оперативної адаптації та взаємодії між різнорідними програмними комплексами. У такому контексті ефективна інтеграція систем набула статусу критично важливого елемента забезпечення операційної ефективності, стратегічної гнучкості та конкурентоспроможності сучасних підприємств.

Традиційні підходи до інтеграції, що часто базуються на точкових зв'язках або тісно зв'язаних інтерфейсах, виявляються недостатніми для подолання викликів, пов'язаних з гетерогенністю технологій, дублюванням даних, складністю підтримки та масштабування. Виникає нагальна потреба у забезпеченні більш надійного, ефективного та гнучкого механізму комунікації та обміну інформацією між системами. Це формує вимогу до вдосконалення інтеграційних рішень.

Означена проблемна область стала методологічним підґрунтям для поточного дослідження. Основною метою дослідження було подолання викликів системної інтеграції шляхом розробки архітектурного рішення, що базується на сучасних парадигмах. Зокрема, дослідження зосередилося на потенціалі сервісно-орієнтованого підходу (Service-Oriented Architecture, SOA) як основи для побудови стійкої та гнучкої інтеграційної платформи,

здатної задовольнити поточні потреби та забезпечити адаптацію до майбутніх змін.

В рамках роботи було сформульовано гіпотезу щодо цінності та ефективності застосування принципів SOA у поєднанні з імплементацією архітектурного патерну Enterprise Service Bus (ESB). Мета дослідження полягала в емпіричній валідації цієї гіпотези шляхом створення та апробації прототипу інтеграційного рішення, що ілюструє та демонструє переваги такого підходу.

Результатом роботи стала розробка функціонального прототипу, що використовує технології веб-сервісів та ESB для реалізації необхідної інтеграційної функціональності. Емпірична оцінка прототипу підтвердила, що використання сервісно-орієнтованого підходу сприяло досягненню поставлених вимог до інтеграції. Хоча SOA є лише одним з компонентів комплексного рішення, його застосування відіграло значущу роль у процесі розробки та формуванні архітектури прототипу, надавши йому необхідної гнучкості та масштабованості. Це надає емпіричні докази на користь застосування SOA (зокрема, через ESB) як дієвого підходу для вирішення складних завдань інтеграції в сучасних гетерогенних IT-ландшафтах.

## 1.2. Передумови використання сервісно-орієнтованої архітектури для інтеграції корпоративних додатків

Сучасний розвиток цивілізації неможливо уявити без ubiquitous computing та глобальних мережевих з'єднань. Прогрес та інновації у галузі інформаційних технологій (IT) досягли значних масштабів і продовжують демонструвати експоненційне зростання. Інженери програмного та апаратного забезпечення постійно розширюють межі технологічних можливостей. Хоча точне визначення причинно-наслідкового зв'язку між вимогами до програмного забезпечення та інноваціями в апаратному

забезпеченні є предметом дискусій, можна стверджувати, що вони перебувають у симбіотичних відносинах і мають обмежену функціональну цінність ізольовано одне від одного. Обчислювальна система без програмного забезпечення не здатна виконувати корисні функції, так само як програмне забезпечення потребує апаратної платформи для свого виконання та розробки.

Зі зростаючою залежністю бізнес-процесів та організацій від IT, а також зі значним збільшенням обсягів зібраних та оброблених даних за останні роки, що, зокрема, стимулювало розвиток бізнес-аналітики, архітектура інфраструктури інформаційних систем суттєво ускладнилася. Фактори, такі як процеси злиття та поглинання (M&A) та динамічна природа ринкових умов, які вимагають підвищеної гнучкості та адаптивності, зробили інтеграцію систем критично важливим аспектом діяльності сучасних підприємств. Попит на інтероперабельність систем та ефективну інтеграцію створює тиск на розробників щодо створення ефективних рішень для автоматизації подій та підвищення архітектурної гнучкості. Розробка проміжного програмного забезпечення (middleware), яке забезпечує обмін інформацією між гетерогенними системами, стала важливим кроком у задоволенні потреби систем у взаємодії. На вищому рівні, коли додатки та системи розподілені географічно, функціонують на різних платформах та використовують різнорідне програмне забезпечення, інтеграція зазвичай класифікується як інтеграція корпоративних додатків (Enterprise Application Integration, EAI).

Для підвищення гнучкості інтеграційних рішень та сприяння повторному використанню компонентів критично важливо мінімізувати жорстке зв'язування між системами та додатками, оскільки надмірна залежність обмежує їхнє подальше застосування та модифікацію. Одним з ефективних підходів до вирішення цієї задачі є використання сервісно-орієнтованої архітектури (Service-Oriented Architecture, SOA). SOA дозволяє

створювати слабозв'язані сервіси, які можуть взаємодіяти між собою у гнучкий спосіб. Зі зростанням масштабу та складності інтеграційних рішень, особливо в умовах географічно розподілених систем, виникає потреба у централізації управління взаємодіями для спрощення їх моніторингу та підтримки. Ця потреба часто задовольняється шляхом впровадження архітектурного патерну Enterprise Service Bus (ESB), який, серед іншого, забезпечує централізоване управління та маршрутизацію сервісів більш ефективним чином.

Описаний контекст формує методологічне підґрунтя для поточного дослідження, яке спрямоване на подолання викликів системної інтеграції шляхом застосування сучасного та гнучкого архітектурного підходу. Дослідження проводиться на базі середньої за розміром міжнародної компанії, що спеціалізується на наданні IT-рішень та послуг підтримки різним галузям. З міркувань конфіденційності назва компанії в рамках даного дослідження не розкривається.

Основна задача дослідження полягає у заміщенні існуючого успадкованого додатку на нове, більш функціональне та архітектурно досконаліше рішення. Результатом роботи буде розроблений прототип, який слугуватиме початковою точкою для подальшої імплементації повноцінного додатку. Фінальна розробка може бути здійснена як внутрішніми ресурсами компанії, так і залученими зовнішніми консультантами, залежно від наявних фінансових та часових обмежень.

## 1.3. Актуальність проблеми системної інтеграції та завдання дипломної роботи

Компанія оперує рядом дискретних інформаційних систем, які потребують надійної та ефективної взаємодії для обміну даними. Наслідком цього є виникнення потреби у значному покращенні інтеграційних

можливостей та впровадженні додаткової функціональності. Це, у свою чергу, вимагає розробки нового архітектурного рішення, яке характеризуватиметься високим ступенем гнучкості та здатністю адаптуватися до еволюційних вимог у майбутньому.

Існуюче інтеграційне рішення реалізоване у вигляді програмного скрипту, що забезпечує передачу даних між системами. Цей скрипт здійснює вибірку інформації з однієї системи, виконує перевірку на наявність змін та, у разі їх виявлення, відправляє оновлені дані до іншої системи для збереження у відповідній базі даних. Цей процес виконується з періодичністю один раз на добу. Початково розроблений як тимчасове, "швидке" рішення, з часом цей додаток еволюціонував шляхом інкрементального додавання функціональності у відповідь на нові вимоги. Така еволюція призвела до створення системи, що не тільки є складною в обслуговуванні та модифікації, але й демонструє обмежену здатність ефективно обробляти нові або майбутні функціональні вимоги. Більше того, через свою спеціалізованість та тісну прив'язку до конкретного середовища, її придатність для повторного використання в інших підрозділах компанії практично відсутня, що є одним із ключових недоліків, які має усунути пропоноване дослідження.

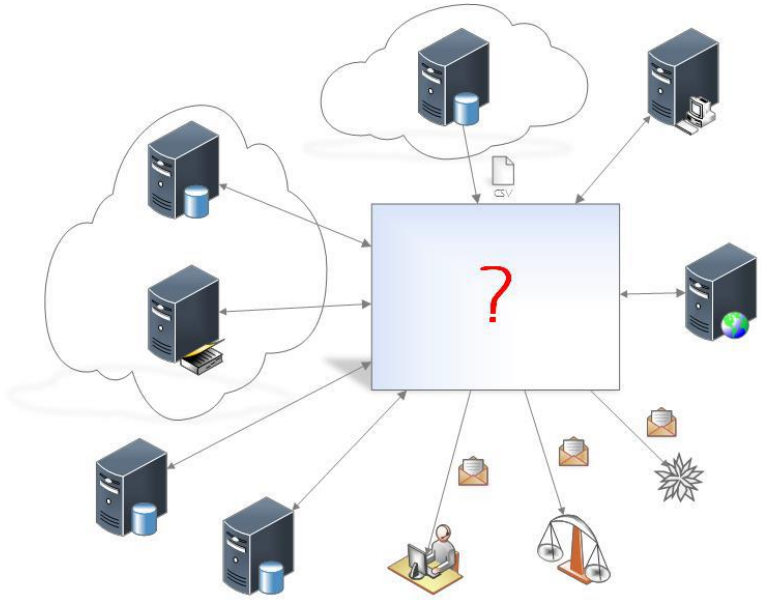


Рисунок 1.1 - Високорівнева схема топології системи

На ілюстративній діаграмі топології (рисунок 1.1) центральний елемент позначає поточний інтеграційний компонент, реалізований на основі скрипту, який підлягає заміщенню на більш досконале архітектурне інтеграційне рішення.

З наукової точки зору, ключова проблема полягає у визначенні оптимальних способів використання можливостей, що надаються сучасними парадигмами, такими як сервісно-орієнтована архітектура (SOA), та інструментальними засобами, зокрема Enterprise Service Bus (ESB), для ефективної інтеграції систем.

На високорівневому концептуальному рівні, мета дослідження полягає у розробці стійкої та адаптивної архітектури, здатної задовольнити всі актуальні потреби інтеграції систем. З метою забезпечення максимальної гнучкості, окремою метою є формалізація та визначення сервісно-орієнтованої архітектури, яка дозволить досягти безшовної та ефективної інтеграції систем, а також забезпечить можливість їх подальшого розширення та масштабування у майбутньому. На початковій фазі реалізації рішення процес інтеграції має бути повністю автоматизований та відтворювати існуючу функціональність скриптового рішення. Додавання нової функціональності передбачається на наступних ітераціях розвитку системи. Впровадження сервісно-орієнтованої архітектури, з її внутрішньою властивістю слабкого зв'язування компонентів, розглядається як критичний фактор для досягнення успішного результату.

Крім того, у разі успішної апробації результатів дослідження, існують плани щодо реплікації розробленого рішення в інших структурних підрозділах компанії. Відповідно, підготовка рішення до максимально можливого масштабування та здатності обробляти додаткові майбутні вимоги є важливим критерієм успіху проєкту.

Однак, справжня дослідницька мета полягає у встановленні того, чи використання сервісної орієнтації у поєднанні з імплементацією Enterprise

Service Bus не тільки задовольнить поставлені проєктом вимоги, але й принесе суттєву додаткову цінність. Дане дослідження базується на дедуктивному методологічному підході, деталізація якого представлена у розділі, присвяченому методам дослідження. Відповідно до обраного підходу, робоча гіпотеза дослідження узгоджується з його основною метою.

Хоча наукова література містить численні приклади інтеграції систем, кожний конкретний сценарій інтеграції є унікальним, що обумовлює специфіку кожного рішення. Таким чином, дане дослідження не ставить за мету створення нової фундаментальної теорії чи революціонізацію існуючого наукового підходу, а скоріше спрямоване на надання ще одного емпіричного прикладу успішної інтеграції систем, а також надання підтримки для сформульованої вище гіпотези. Очікуваним результатом, у контексті проблем, окреслених у попередньому підрозділі, є розроблений та побудований на основі доступної теорії прототип інтеграційного рішення.

Завданням також є надати читачеві уявлення про релевантні техніки та методології, необхідні для реалізації подібних проєктів з розробки інтеграційного програмного забезпечення. Слід підкреслити, що через різноманіття доступних методологій проєктування, спосіб, яким було проведено дане конкретне дослідження, є лише одним з багатьох можливих підходів, і читачеві, що планує ініціювати подібний процес, рекомендується ознайомитися з альтернативами перед вибором конкретної методології. Мотивація вибору підходу, застосованого в даній роботі, буде представлена разом з описом відповідних методів.

Для більш чіткого формулювання, дослідницьке питання, на яке спрямована дана робота, звучить наступним чином:

"При проєктуванні нового інтеграційного рішення, яким чином використання сервісної орієнтації та Enterprise Service Bus сприяє створенню додаткової цінності ".

## 1.4. Теоретичні аспекти системної інтеграції, SOA та життєвого циклу розробки програмного забезпечення

Метою даного підрозділу є надання необхідного теоретичного підґрунтя для успішної розробки прототипу. Розділ містить огляд ключових концепцій, що стосуються системної інтеграції, сервісно-орієнтованої архітектури, а також процесу розробки програмного забезпечення та його типових фаз. Особлива увага приділяється важливим аспектам, таким як збір та аналіз вимог, з метою забезпечення міцної теоретичної бази для подальших етапів проєкту.

На високорівневому концептуальному рівні, фундаментальною метою розробки нового програмного забезпечення, або вдосконалення та модифікації існуючого, є надання бізнес-цінності та задоволення актуальних потреб організації. У контексті мінливих та зростаючих вимог до додаткової функціональності та підвищення продуктивності, виникає нагальна потреба у розробці архітектурного рішення, що забезпечить легку підтримку, можливість майбутніх оновлень та інтеграцію додаткових систем. Для забезпечення стійкої цінності таке рішення має характеризуватися масштабованістю, тобто здатністю ефективно розширюватися відповідно до майбутніх бізнес-потреб.

З моменту появи комп'ютерної ери, методології розробки програмного забезпечення пройшли значний еволюційний шлях, зазнавши численних змін та вдосконалень. Ранні підходи, що базувалися на парадигмі водоспадної моделі, поступово поступалися місцем більш гнучким методологіям, що домінують сьогодні. Цю зміну можна розглядати як пряму реакцію на обмеження водоспадної моделі, яка могла призводити до нездатності проєктів генерувати цінність, особливо у випадках, коли початкові вимоги змінювалися або були недостатньо чітко сформульовані, або коли процеси розробки не відповідали суворо лінійній послідовності.

Ключовою характеристикою гнучкої розробки (Agile Development) є прийняття зміни як невід'ємного елемента процесу, що базується на усвідомленні динамічності та мінливості сучасного бізнес-середовища, яке неминуче спричиняє зміни у вимогах. Крім того, ітеративний характер гнучких процесів, що передбачає створення функціональних "інкрементів" на кожній ітерації, дозволяє здійснювати безперервну оцінку та вдосконалення продукту. До популярних гнучких методологій належать SCRUM, XP (Extreme Programming) та AUP (Agile Unified Process). До переваг використання гнучких підходів відносять: скорочення часу виведення продукту на ринок, підвищення прозорості процесу розробки , зростання задоволеності замовника та більш ефективне використання часу та ресурсів.

Типовий процес розробки програмного забезпечення включає чотири основні фази: специфікація програмного забезпечення, проектування та реалізація, валідація та еволюція. Іноді виділяють п'ять дещо відмінних фаз: комунікація, планування, моделювання, побудова та розгортання. Незважаючи на значне перекриття між цими моделями, ключова відмінність полягає у явному виділенні фази еволюції як продовженого вдосконалення, що в іншій моделі інтегровано в пізніші фази. Обидві моделі передбачають безперервне, тобто ітеративне застосування для поступового створення інкрементальних результатів. Третій варіант життєвого циклу розробки програмного забезпечення (ЖЦРПЗ або SDLC) представлений на рисунку 1.2.

Різні фази розробки програмного забезпечення будуть коротко описані нижче, починаючи з фази специфікації програмного забезпечення (що охоплює комунікацію) або інженерії вимог. Остання буде розглянута дещо детальніше з огляду на її критичну значущість для успішної реалізації проєкту. Важливо зазначити, що хоча фази описуються окремо, на практиці їхні границі часто є розмитими, що призводить до їх взаємопроникнення та інтеграції.

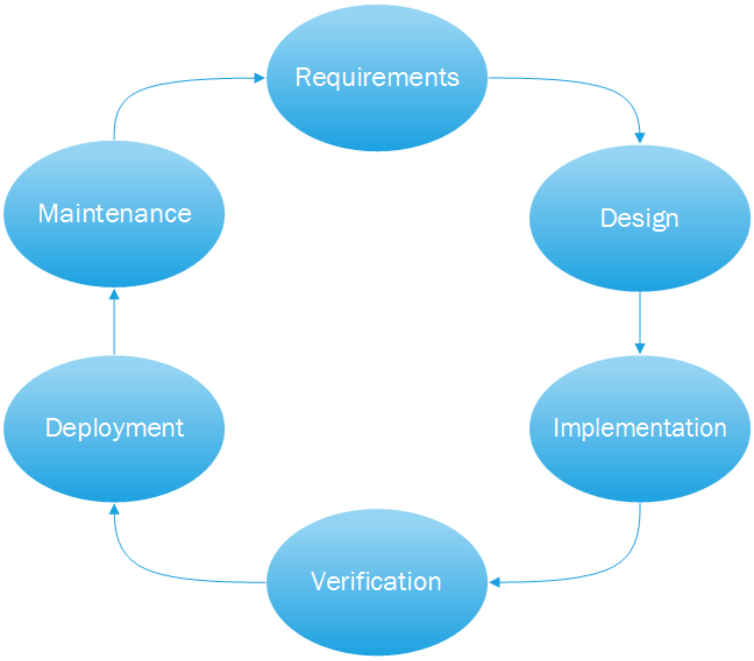


Рисунок 1.2 – Життєвий цикл розробки програмного забезпечення

## 1.5. Інженерія вимог та фази розробки програмного забезпечення

Успіх будь-якого великомасштабного проєкту суттєво залежить від ефективності попередньої фази, присвяченої збору вимог від зацікавлених сторін, незалежно від специфіки проєкту. Саме вимоги визначають вектор розвитку проєкту ; без їх чіткого визначення досягнення поставлених цілей та завдань є вкрай ускладненим або неможливим.

### *1.5.1. Визначення вимог*

Вимога може бути сформульована як обмеження, яке необхідно виконати для отримання схвалення від відповідної зацікавленої сторони, або як певна функція, що має бути реалізована у програмному забезпеченні. Ключовими характеристиками якісних вимог є їх чіткість, однозначність, послідовність та пріоритезація. Формулювання вимог має бути лаконічним та конкретним, унеможливлюючи неоднозначні інтерпретації. Вимоги мають бути взаємоузгодженими (несуперечливими) та чітко ранжованими за ступенем важливості. Для пріоритезації може використовуватися метод

MoSCoW, який передбачає категоризацію вимог за спаданням важливості: "Must have" (обов'язкові), "Should have" (бажані), "Could have" (можливі) та "Won't have" (ті, що не будуть реалізовані на даному етапі).

Зазвичай вимоги класифікують за двома основними категоріями: функціональні та нефункціональні. Функціональні вимоги, значною мірою залежно від типу програмного забезпечення, що розробляється, визначають функціональну поведінку системи, включаючи опис функцій, вхідних та вихідних даних. Нефункціональні вимоги, у свою чергу, стосуються атрибутів якості системи, таких як продуктивність, зручність використання, надійність, безпека та доступність. Для групування цих типів вимог іноді використовується модель FURPS+, що походить з Unified Process (USDP), де вони об'єднуються під терміном "вимоги до якості".

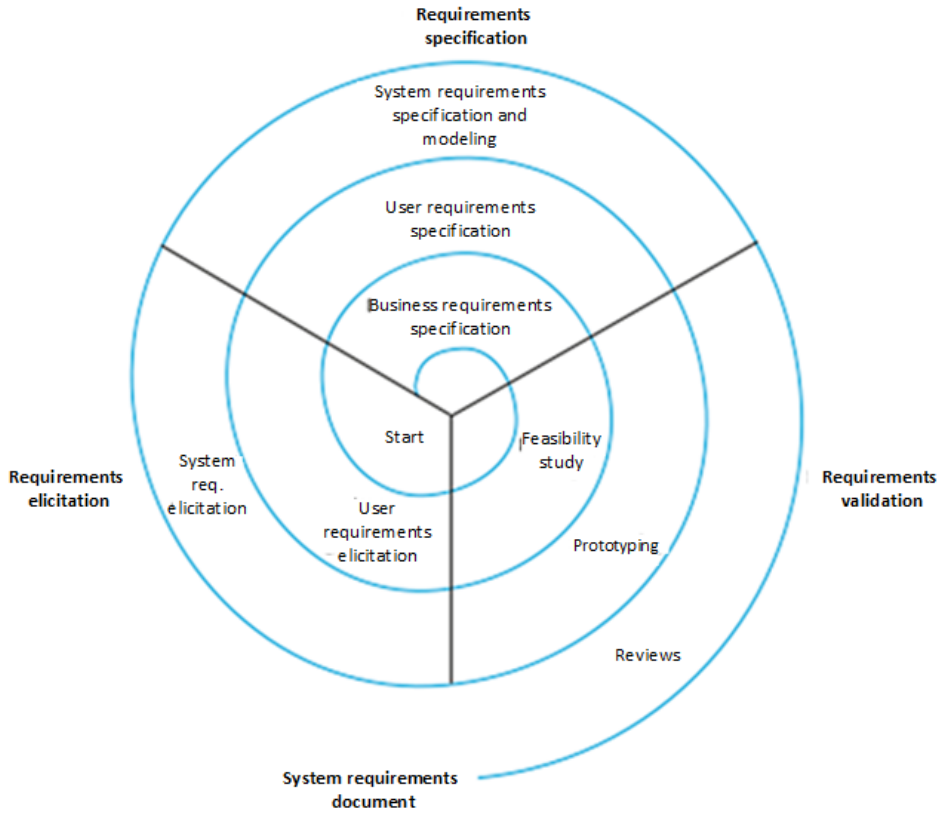


Рисунок 1.3 – Спіральна модель процесу інженерії вимог

Процес збору та управління вимогами для програмного проєкту відомий як інженерія вимог. Цей процес включає низку послідовних фаз. Ці

фази включають: ініціалізацію, виявлення, розробку, переговори, специфікацію та валідацію. Часто визначають фази інженерії вимог наступним чином: дослідження можливості (оцінка бізнес-корисності), виявлення та аналіз, специфікація та валідація. Спіральна модель процесу інженерії вимог представлена на рисунку 1.3.

### *1.5.2. Забезпечення якості вимог*

Простий збір вимог не гарантує успіху проєкту; ключовим аспектом успішного процесу інженерії вимог є забезпечення високої якості зібраних вимог, що робить їх корисними та надає чітке керівництво для подальших етапів розробки програмного забезпечення. У цьому процесі важливу роль відіграють експерти в предметній області – група осіб, чий досвід є критичним для досягнення бажаного результату.

Quality Function Deployment (QFD) – це техніка управління, яка застосовується для максимізації задоволеності клієнта через глибоке розуміння його потреб та пріоритетів. Таке розуміння має пронизувати весь життєвий цикл розробки програмного забезпечення. Вимоги також мають важливе значення на завершальній стадії проєкту, під час фази оцінки, коли здійснюється аналіз та оцінка проєкту на предмет відповідності всім початково визначеним вимогам.

### *1.5.3. Планування*

Фаза планування охоплює діяльність, що виконується до початку фактичної реалізації артефакту чи прототипу. Основною метою цієї фази є створення "дорожньої карти" або плану проєкту, який слугуватиме керівництвом для подальшого процесу розробки. План проєкту може включати опис обсягу робіт (statement of work), необхідні ресурси, ідентифіковані ризики, перелік кінцевих продуктів (deliverables) та графік проєкту. На цьому етапі можуть вирішуватися технічні питання, такі як вибір

мови програмування, стандарти кодування, вибір середовища розробки та програмних інструментів. Для великих проєктів ця фаза може включати процеси пошуку, переговорів та залучення кадрових ресурсів, формування команд та визначення потреби у зовнішніх консультантах.

*1.5.4. Проектування / Моделювання*

Як випливає з назви, очікуваним результатом цієї фази є модель проектування. Основою для розробки цієї моделі є вимоги, зібрані на етапі інженерії вимог, що визначають цілі проектування. Модель має відображати загальну архітектуру системи та принципи взаємодії між її компонентами. Загальна архітектура є критично важливим елементом, що пов'язує фазу вимог з подальшою реалізацією, оскільки вона інкапсулює основні структурні компоненти та їх взаємозалежності. Тому часто спостерігається значне перекриття між цими двома фазами. Модель проектування може розглядатися як "міст" між цими етапами; вона має бути достатньо абстрактною, щоб не перевантажувати загальну картину зайвими деталями, але водночас достатньо деталізованою для надання необхідної інформації розробникам. Типовим рішенням для узгодження цих суперечливих вимог є розробка моделей на різних рівнях абстракції.

Проектування та моделювання зазвичай здійснюються на високому та низькому рівнях. Високорівневе проектування має на меті надати загальний огляд пропонованої системи без надмірної деталізації, зберігаючи цілісне бачення. Низькорівневе проектування, навпаки, передбачає значно вищий ступінь деталізації, включаючи технічні рішення та рекомендації для реалізації (наприклад, структура баз даних, атрибути класів), відповідаючи на питання "як", а не тільки "що", як на високому рівні. Хоча границі між цими рівнями можуть бути не завжди чіткими, низькорівневе проектування однозначно характеризується вищим рівнем деталізації. Цільова аудиторія для моделей також відрізняється: високорівневі моделі призначені для

керівників проєктів та інших зацікавлених сторін, тоді як низькорівневі – для програмістів та розробників.

### *1.5.5. Реалізація*

Після завершення всіх попередніх робіт настає етап фактичної побудови програмного забезпечення. Ця фаза передбачає, що компанія прийняла рішення про розробку програмного забезпечення власними силами, а не придбання готового комерційного продукту (COTS). Якщо моделі, розроблені на попередній фазі, мають достатній ступінь деталізації, вони слугують чітким керівництвом, роблячи фазу реалізації відносно прямолінійною.

Хоча конкретні методи та інструменти реалізації можуть варіюватися залежно від обраної мови програмування та технологічного стеку, основна мета залишається незмінною – створення працездатного програмного забезпечення. У гнучких методологіях цей процес здійснюється шляхом серії ітерацій, де кожна ітерація призводить до створення більш повної частини програмного забезпечення зі збільшенням функціональних можливостей. За можливості слід проводити аналіз потенціалу повторного використання існуючого коду, програмних компонентів або систем, щоб уникнути дублювання зусиль.

### *1.5.6. Розгортання*

Розгортання нового програмного забезпечення – це процес введення нового продукту в його цільове операційне середовище. Тестування зазвичай здійснюється безперервно протягом усього процесу побудови та є невід'ємною частиною фази розгортання, забезпечуючи коректне функціонування нового програмного забезпечення. З огляду на значну варіативність процесу розгортання, що залежить від розміру та типу програмного забезпечення, апаратної платформи, кількості залучених систем

тощо, критично важливим є розробка плану, оптимально адаптованого до поточних потреб. План розгортання має передбачати потенційні проблеми та способи їх оперативного усунення. Рекомендується мати план відкату (rollback plan), який може бути негайно реалізований для відновлення системи до попереднього стабільного стану, якщо процес розгортання відбуватиметься не за планом.

Процес міграції користувачів на нову систему або додаток відомий як перехід (transition). Можна виділити чотири основні методи переходу: поетапне впровадження, поступовий перехід, інкрементальне розгортання та паралельне тестування. Метою цих методів є забезпечення максимально плавного та безпечного процесу переходу для кінцевих користувачів.

### *1.5.7. Валідація та еволюція*

Вимоги, отримані на етапі інженерії вимог, слугують основою як для внутрішньої перевірки продукту (verification), так і для валідації замовником. Валідація замовником передбачає офіційне підтвердження відповідності поставленого продукту початково сформульованим вимогам (Stephens, 2015). Фактичний процес валідації та верифікації також реалізується через різноманітні процедури тестування, критерії успіху для яких, як правило, визначаються у специфікації вимог.

Після успішної побудови та розгортання системи або програмного забезпечення розпочинається заключна фаза, яка називається еволюцією або підтримкою. Ця фаза охоплює широкий спектр активностей, від усунення незначних дефектів (виправлення помилок) та адаптації до мінливих вимог до покращення продуктивності та додавання нової функціональності. Основні завдання підтримки можна класифікувати за чотирма категоріями: профілактична, коригуюча, адаптивна та досконала підтримка. Найбільша частка часу та ресурсів зазвичай витрачається на досконалу підтримку, яка зосереджена на реалізації нових або вдосконалених функцій.

До двох третин загального бюджету програмного забезпечення може витрачатися протягом цієї фази. Це пов'язано зі значними інвестиціями компаній у програмні системи та їх подальшою залежністю від них. Оскільки створення або придбання програмного забезпечення є дорогим на початковому етапі, для отримання віддачі від інвестицій (ROI) система має експлуатуватися протягом значного періоду часу, часто понад 10 років. Відповідно, значна частина бюджету спрямовується на вдосконалення існуючого програмного забезпечення замість його повної заміни.

Проте, для більшості систем настає момент, коли численні модифікації, спрямовані на відповідність мінливим вимогам та змінам у середовищі, роблять кожне подальше вдосконалення все менш економічно ефективним. Це свідчить про необхідність виведення програмного забезпечення з експлуатації та його повної заміни. Етап виведення з експлуатації може включати міграцію або утилізацію даних та інтелектуальної власності, а також безпечне знищення конфіденційної інформації та носіїв даних.

# РОЗДІЛ 2. СЕРВІСНО-ОРІЄНТОВАНА АРХІТЕКТУРА ТА СИСТЕМНА ІНТЕГРАЦІЯ ДОДАТКІВ КОРПОРАТИВНОГО РІВНЯ

## 2.1. Сервісно-орієнтована архітектура та її реалізація за допомогою веб-сервісів

Процес інженерії сервісів, спрямований на розробку програмних сервісів, типово включає три основні фази з відповідними підфазами. Ці фази зображені на рисунку 2.1.

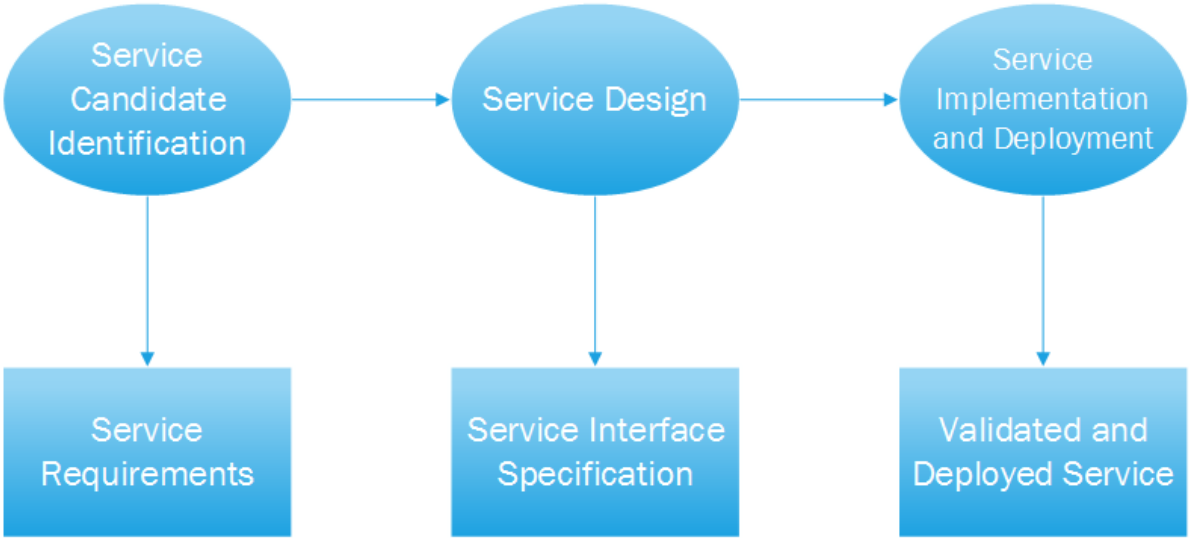


Рисунок 2.1 - Етапи сервісної інженерії

- Визначення сервісу. На цій фазі ідентифікуються кандидати на реалізацію сервісів та формалізуються їхні функціональні та нефункціональні вимоги.

- Проектування сервісу. Ця фаза включає проектування інтерфейсів сервісу (визначення доступних операцій) та форматів повідомлень, що використовуються для взаємодії.

- Реалізація та розгортання сервісу. На фінальній фазі здійснюється побудова сервісу (як нового компонента або як інтерфейсу до існуючої системи), його тестування та подальше розгортання в цільовому середовищі.

### *2.1.2. Ключові принципи проектування сервісно-орієнтованої архітектури (SOA)*

Сервісно-орієнтована архітектура (SOA) є концептуальною архітектурною парадигмою, призначеною для забезпечення взаємодії та комунікації між різнорідними та розподіленими програмними системами шляхом використання сервісів. Важливо відзначити, що концепція сервісів не обмежується лише технологічним аспектом, а є більш загальною концепцією, що застосовується як у сфері бізнес-процесів, так і в архітектурі програмного забезпечення. SOA функціонує як модель проектування, яка передбачає інкапсуляцію логіки додатків у дискретні сервіси, що взаємодіють між собою через стандартизований або загальний протокол.

Завдяки своїй незалежності, окремі елементи, що складають SOA (сервіси), можуть бути легко переміщені, замінені або адаптовані для інтеграції в нові контексти та середовища. Ключові принципи проектування, які лежать в основі сервісної орієнтації та забезпечують її гнучкість та ефективність і включають:

- Повторне використання (Reusability): Сервіси розробляються з розрахунком на багатократне використання в різних бізнес-процесах або додатках.

- Безстанність (Statelessness): Сервіс не зберігає інформацію про попередні взаємодії з конкретним споживачем, що спрощує масштабування та управління.

- Автономність (Autonomy): Кожен сервіс має контроль над своєю логікою та ресурсами, функціонуючи як незалежний підрозділ.

- Абстракція (Abstraction): Зовнішній інтерфейс сервісу приховує деталі внутрішньої реалізації, надаючи споживачам лише необхідну інформацію про його функціональність.

- Можливість виявлення (Discoverability): Сервіси можуть бути легко знайдені споживачами через відповідні механізми (наприклад, реєстри сервісів).

- Слабка зв'язаність (Loose Coupling): Сервіси мають мінімальну залежність один від одного, що дозволяє змінювати один сервіс без суттєвого впливу на інші.

- Композиційність (Composability): Сервіси можуть бути об'єднані (скомпоновані) для формування складніших бізнес-процесів або додатків.

Ці принципи уможливлюють надання функціональності додатків та їх споживання у вигляді сервісів, сприяючи гнучкості та інтероперабельності.

### *2.1.2. Сервіси у контексті розподілених систем*

Розподілені системи та розподілені обчислення виникають з необхідності спільної роботи щонайменше двох комп'ютерних систем для надання певного бізнес-сервісу або виконання бізнес-процесу/завдання. Прикладами можуть слугувати процес оформлення замовлення, де дані про продукт і інформація про клієнта зберігаються у різних системах, або процес обробки платежу, що вимагає даних з CRM-системи, бази даних продуктів та фінансової системи. Системи, залучені до розподіленого процесу, можуть значно відрізнятися за типом, програмним забезпеченням, апаратним забезпеченням, фізичним розташуванням тощо.

У рамках SOA, сервіс є незалежним "будівельним блоком", який використовується для представлення та інкапсуляції функціональності середовища додатків. Він чітко відокремлює зовнішню поведінку сервісу ("що" він робить) від його внутрішньої реалізації ("як" він це робить), що є

суттю принципу абстракції і зазвичай не має значення для споживачів сервісу.

### *2.1.3. Веб-сервіси як механізм реалізації SOA*

Веб-сервіси (Web Services), що базуються на відкритих стандартах, є одним з найбільш поширених механізмів експонування функціональності програмних додатків. Вони суттєво спрощують здійснення віддалених процедурних викликів (RPC) та обмін даними (EDI) між системами. Веб-сервіси часто використовуються для інтеграції з успадкованими (legacy) системами, дозволяючи зробити функціональність старих систем доступною через стандартизований веб-сервісний інтерфейс без необхідності модифікації самої успадкованої системи. Це значно підвищує доступність даних та сприяє ефективнішому спільному використанню організаційних ресурсів. Внутрішня незалежність веб-сервісів та інкапсульована в них логіка усувають прив'язку до конкретної платформи чи постачальника, що робить їх високо універсальними.

Згідно з визначенням Консорціуму Всесвітньої павутини (W3C): веб-сервіс – це програмна система, призначена для підтримки інтероперабельної взаємодії між машинами через мережу. Він має інтерфейс, описаний у машинооброблюваному форматі (зокрема, WSDL). Інші системи взаємодіють з веб-сервісом у спосіб, визначений його описом, використовуючи повідомлення SOAP, які зазвичай передаються за допомогою HTTP з XML-серіалізацією разом з іншими веб-пов'язаними стандартами.

З цього визначення випливає, що HTTP, як найпоширеніший протокол комунікації в Інтернеті, використовується веб-сервісами як узгоджена "спільна мова" для взаємодії. Крім того, їхня незалежність та автономність роблять веб-сервіси ідеальним засобом для реалізації слабозв'язаних бізнес-функцій, які можуть бути скомпоновані у більш складні архітектурні структури з використанням цих відкритих стандартів.

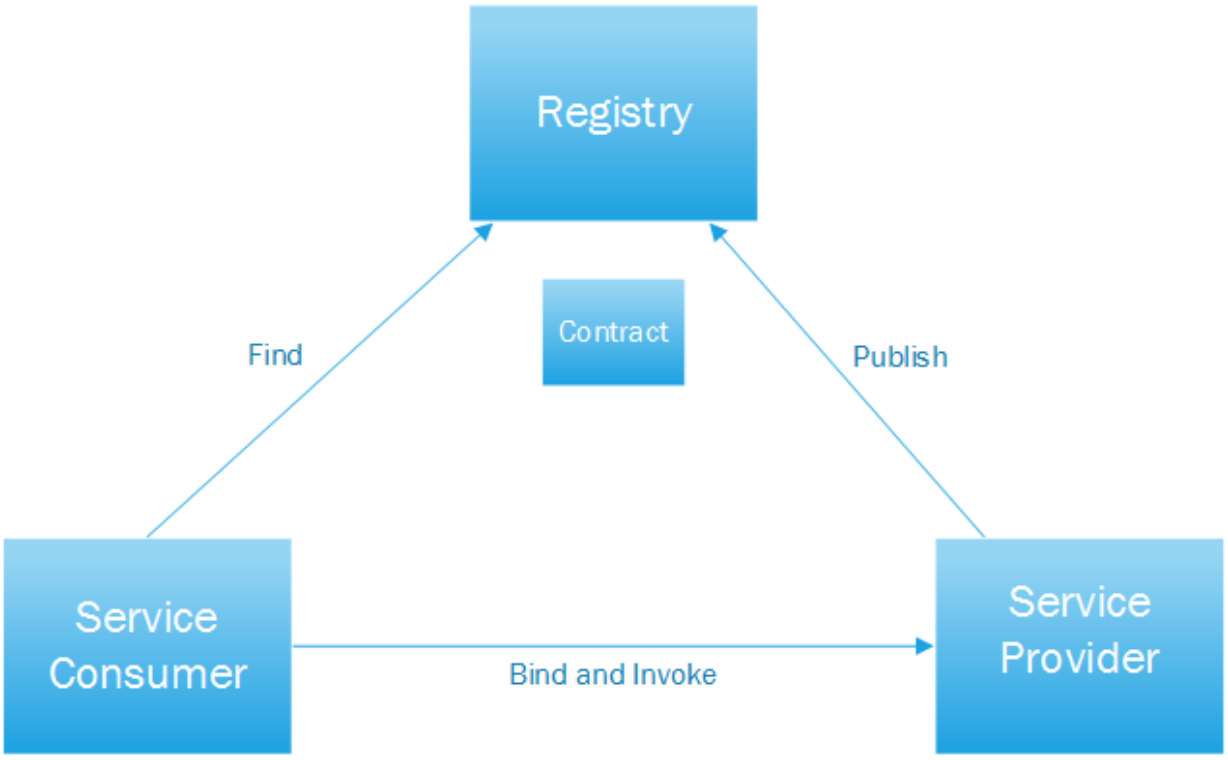


Рисунок 2.2 – Базова функціональність SOA

Як проілюстровано на рисунку 2.2, сервіс може виступати у ролі споживача (запитувача) або постачальника (надавача). Сервіс-споживач викликає функціональність іншого сервісу, який у цьому випадку виступає у ролі сервісу-постачальника. Один і той же сервіс може приймати обидві ролі за необхідності. Для забезпечення можливості виявлення може використовуватися реєстр сервісів, де постачальники публікують описи своїх сервісів, дозволяючи споживачам їх знаходити та використовувати їхню логіку. Взаємодія здійснюється на основі взаємної угоди, що визначає умови обміну інформацією – так званого контракту сервісу.

На завершення слід зазначити, що хоча веб-сервіси стали майже синонімом SOA, вони не є єдиним доступним засобом для реалізації сервісно-орієнтованої архітектури. Серед інших механізмів реалізації SOA слід виділити: Enterprise Service Bus (ESB), компоненти, сервіси RESTful та інші існуючі підходи.

## 2.2. Системна інтеграція та роль Enterprise Service Bus

Різноманіття параметрів, що характеризують компоненти інтеграційного сценарію (такі як апаратне забезпечення, програмне забезпечення, бізнес-логіка, додатки, мови програмування, платформи, постачальники тощо), неодмінно надає кожному такому сценарію унікального характеру. Відповідно, кожне інтеграційне рішення також є унікальним.

Історичний розвиток комп'ютерних систем відзначився переходом від монолітних архітектур, де всі необхідні для бізнес-процесів додатки та дані розташовувалися на одній машині, до розподілених багаторівневих архітектур (від 1-рівневих до n-рівневих). Ця еволюція включала розподіл логіки, представлення та даних, що стало основою для концепцій "товстих" та "тонких" клієнтів. Такий поділ на рівні, де кожен рівень відповідає за виконання певних функцій, проілюстровано на Рисунку 6. Ця архітектурна трансформація стала ключовим кроком у розвитку розподілених обчислень.

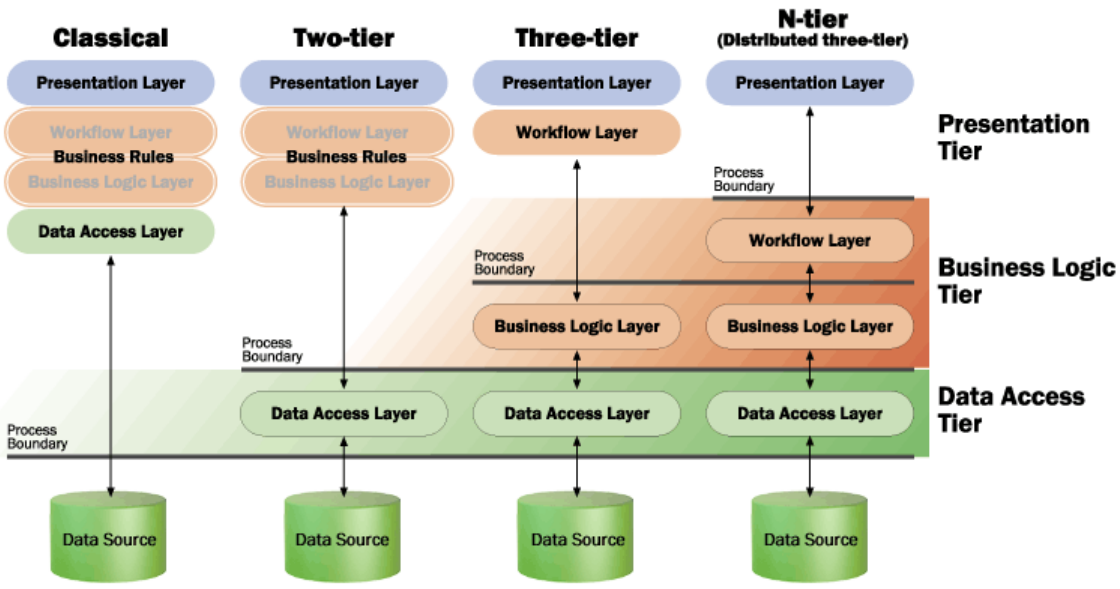


Рисунок 2.3 – Ілюстрація 1-, 2-, 3- та n-рівневих архітектурних моделей

Інтеграція додатків на рівні підприємства визначається як інтеграція корпоративних додатків (Enterprise Application Integration, EAI). Цей термін стосується як загальної концепції взаємозв'язку великих програмних систем, так і конкретних програмних продуктів, розроблених для цієї мети, причому перше визначення є більш поширеним. Раніше організації часто стикалися з проблемою утворення великих ізольованих фрагментів IT-інфраструктури, відомих як "силоси" або "острови", де однакові дані дублювалися та зберігалися у кількох фізично різних місцях. EAI спрямована на усунення цієї непотрібної надмірності шляхом забезпечення спільного використання даних та бізнес-процесів між усіма системами та додатками в організації, переважно за допомогою стандартизованих інтерфейсів, таких як веб-сервіси. Інтеграція в рамках EAI може здійснюватися на одному з чотирьох основних рівнів: рівні даних, рівні додатків, рівні методів (функціональності) або рівні користувацького інтерфейсу. Інтеграція технологій, що забезпечують взаємодію та координацію розподілених гетерогенних систем, дозволяє об'єднувати різнорідні додатки для реалізації складніших та ефективніших бізнес-процесів.

Традиційна архітектура EAI часто реалізовувалася з використанням брокера повідомлень та двигуна оркестрації. Однак, з часом ці реалізації були значною мірою заміщені більш досконалою архітектурою, що використовує Enterprise Service Bus (ESB), яка розглядається далі.

Enterprise Service Bus (ESB) є інтеграційною платформою, що може функціонувати як динамічне середовище для хостингу та управління взаємодією сервісів та додатків. ESB базується на відкритих стандартах та використовує шинну архітектуру, що є відмінністю від більш ранньої моделі EAI типу "хаб-спік". Це архітектурне рішення сприяє зниженню ризику прив'язки до конкретного постачальника (vendor lock-in).

Основні функціональні можливості, які типово надає ESB включають:

1. Прозорість розташування (Location Transparency): Забезпечення комунікації між споживачем та постачальником сервісу через ESB без необхідності для споживача знати фізичне розташування постачальника.

2. Конвертація транспортного протоколу (Transport Protocol Conversion): Здатність трансформувати повідомлення між різними транспортними протоколами, що використовуються інтегрованими додатками.

3. Трансформація повідомлень (Message Transformation): Можливість зміни формату повідомлень для забезпечення сумісності між гетерогенними компонентами, написаними різними мовами або використовують різні структури даних.

4. Маршрутизація повідомлень (Message Routing): Визначення оптимального шляху доставки повідомлень до відповідних додатків-приймачів у складних інтеграційних сценаріях.

5. Безпека (Security): Надання механізмів аутентифікації, авторизації та шифрування повідомлень, що є особливо критичним при комунікації через відкриті мережі (наприклад, Інтернет з використанням протоколу HTTP/S).

6. Моніторинг та управління (Monitoring & Management): Надання функціональності для нагляду за процесами, додатками та потоками повідомлень, що циркулюють через шину, з огляду на критичність ESB у загальній системі інтеграції.

Наявність цих ключових можливостей позиціонує ESB як технологію, що ефективно функціонує як засіб реалізації сервісно-орієнтованої архітектури. Таким чином, ESB може розглядатися як еволюція або розширення концепції EAI. Системна інтеграція, побудована на принципах сервісно-орієнтованої архітектури з використанням ESB як центрального компонента, сьогодні широко визнається багатьма практиками та дослідниками як сучасна форма проміжного програмного забезпечення (middleware).

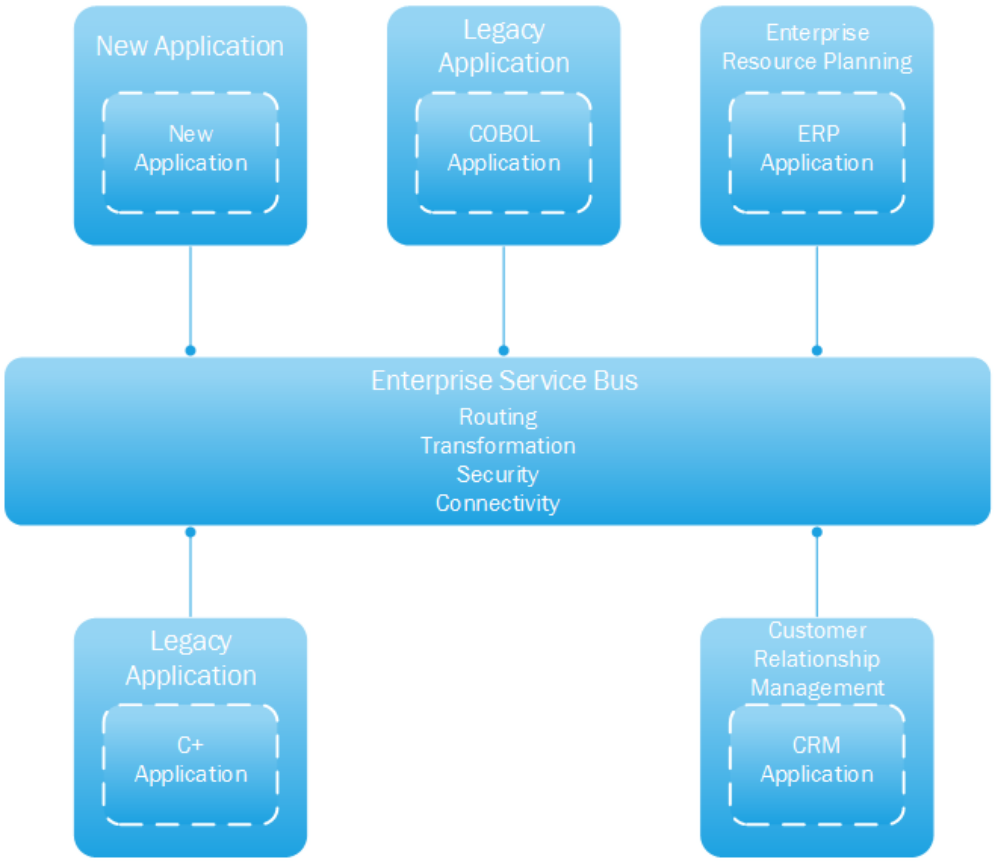


Рисунок 2.4 – Інтеграція додатків з використанням ESB

Рисунок 2.4 ілюструє типовий приклад інтеграції додатків та сервісів за допомогою ESB, демонструючи його роль як центрального брокера повідомлень та оркестратора взаємодій.

## 2.3. Огляд теоретичних засад та практичного застосування SOA та ESB в системній інтеграції

У науковій літературі представлено значну кількість прикладів реалізації системної інтеграції. Даний розділ присвячений огляду деяких з них, а також аналізу бізнес-аспектів, переваг, викликів та тенденцій у цій сфері, зокрема стосовно сервісно-орієнтованої архітектури (SOA) та Enterprise Service Bus (ESB).

Визначення бізнес-цінності інтеграції систем є критичним на початкових етапах проєкту, зважаючи на суттєві витрати, пов'язані з її

реалізацією. Для успішного впровадження необхідно глибоке розуміння бізнес-процесів компанії. Обґрунтування необхідності інтеграції вимагає зіставлення потенційних переваг (таких як підвищення продуктивності, зниження рівня помилок, оптимізація процесів) з витратами на впровадження. Оцінка економічного впливу від реалізації SOA є складним завданням через множинність факторів, що впливають на результат. З метою вирішення цієї проблеми в [12] розробили модель для опису економічного зв'язку між SOA та потенційною бізнес-цінністю. На підставі свого дослідження вони дійшли висновку, що основні переваги випливають з покращень в операційній IT-інфраструктурі, але також можуть мати вплив на стратегічному рівні.

Одним з головних бізнес-драйверів впровадження системної інтеграції є можливість оптимізації бізнес-процесів. Згідно з [13], використання EAI уможливлює повторне використання додатків, що вже функціонують в організації. Інструментальні засоби, такі як Mule або Spring, є типовими при проектуванні та реалізації інтеграційних рішень, які можуть базуватися на архітектурних шаблонах. Разначають, що властива SOA здатність до повторного використання існуючих компонентів може суттєво скоротити час розробки інтеграційних рішень, що, на вищому рівні, сприяє підвищенню організаційної гнучкості та надійності. Впровадження EAI з використанням веб-сервісів на основі відкритих стандартів може зменшити вартість, час та складність взаємодії між системами.

В [14] дійшли висновку, що застосування сервісно-орієнтованого підходу на етапах проектування та реалізації їхнього дослідницького проєкту забезпечило такі переваги, як масштабованість та гнучкість. Крім того, цей підхід виявився ефективним для динамічної збірки додаткових компонентів у проміжне програмне забезпечення (middleware) з використанням спільного API для обробки взаємодії між розподіленими компонентами.

Акцентуючи на тому, що веб-сервіси переважно застосовуються на вищому рівні інтеграції даних, в [15] запропонували рішення, орієнтоване на інтеграцію віддалених пристроїв на нижчому рівні. Проєкт, що досліджував інтеграцію даних з віддалених терміналів у водопостачальній компанії, продемонстрував використання проміжного програмного забезпечення для вирішення інтеграційних завдань шляхом виділення даних для комунікації. У цій архітектурі верхній рівень відповідав лише за представлення даних, тоді як нижній рівень займався передачею даних.

Оскільки веб-сервіси доступні через мережу Інтернет, і єдиною технічною передумовою для їх використання є надійне інтернет-з'єднання, інтеграція систем та бізнес-процесів може відносно легко масштабуватися, перетинаючи регіональні та національні кордони. У своїй статті [16] спочатку стверджують, що сучасна інфраструктура SOA часто будується з використанням технології веб-сервісів, а потім демонструють її застосування на глобальному рівні. У таких глобально розподілених середовищах ESB використовується як центральний компонент інфраструктури, при цьому архітектурний підхід залишається мінімалістичним.

В [17] припускають, що спільне використання даних може бути оптимізоване завдяки використанню проміжного програмного забезпечення для сервісів даних (Data Service Middleware, DSM), яке уможливлює виявлення існуючих гетерогенних джерел даних. Експорт джерел даних у вигляді сервісів на етапі розробки додатків може скоротити час, необхідний для інтеграції, за рахунок використання вже існуючих ресурсів даних. Це означає, що у випадках, коли для надання інформації або підтримки бізнес-процесів, що залежать від розподілених даних, потрібні різноманітні джерела даних, їхнє представлення як сервісів є цілком життєздатним варіантом.

Дослідження щодо потенційних факторів успіху міграції успадкованих систем (legacy systems) у сервісно-орієнтоване середовище вказує на

можливість досягнення певних позитивних ефектів завдяки застосуванню нових принципів архітектурного проектування. Ці фактори включають:

- готовність компанії до міграції в SOA (не всі системи та процеси однаково придатні для цієї трансформації);

- стратегію міграції (вибір між повторним використанням існуючих компонентів, можливо, через експонування їх як веб-сервісів, або їх повною переробкою);

- управління SOA (SOA governance), що стосується, зокрема, використання бізнес-орієнтованих або технічних Угод про рівень обслуговування (Service Level Agreements, SLA) для визначення меж та умов міграції успадкованих систем.

Рівень зрілості сервісної орієнтації в компанії може значно варіюватися, особливо на початкових етапах міграції. Відповідно, потенційна бізнес-цінність, що реалізується завдяки сервісному забезпеченню бізнес-функцій, може суттєво відрізнятися залежно від контексту. У [18] виділяють чотири різні категорії зрілості сервісної орієнтації, від низького до високого рівня трансформації бізнесу: найнижчий рівень відповідає простому використанню веб-сервісів, другий – сервісно-орієнтованій інтеграції функціональності, третій – IT-трансформації на рівні підприємства, і найвищий рівень – трансформації бізнесу на вимогу (on-demand business transformation). З цього випливає, що переваги можуть бути отримані на різних рівнях відповідно до рівня інвестованих зусиль та ресурсів, і існує пряма кореляція між інвестиціями та потенційною віддачею.

При переході до архітектури SOA важливо враховувати, чи має загальна IT-архітектура бути узгоджена з існуючою організаційною структурою. Надмірні компроміси в архітектурі можуть призвести до створення субоптимальної IT-інфраструктури.

В [19] обговорюють відмінності та спільні риси між домінуючою сьогодні парадигмою об'єктно-орієнтованого програмування (ООП) та

потенційним переходом до сервісно-орієнтованого програмування (СОП) під впливом сервісної орієнтації загалом. Вони дійшли висновку про значні переваги інтеграції принципів SOA в ООП і запропонували "сервісно-готовий підхід" (service-ready approach) як спосіб подолання розриву між цими двома перспективами програмування. Важливим внеском є прагнення до зменшення складності, а не тільки до задоволення вимог, що досягається шляхом адаптації методів розробки до сервісно-орієнтованого мислення. Це свідчить про те, що сучасні практики розробки програмного забезпечення мають включати сервісно-орієнтоване мислення, а не розглядати його як опцію. Включення цих принципів на більш ранніх етапах виявиться корисним, оскільки є майже неминучим, що програмне забезпечення, принаймні частково, стане компонентом більшої сервісно-орієнтованої архітектури в майбутньому. Це має бути ключовим фактором при визначенні вимог до довговічності та інтегрованості додатків.

Підсумовуючи, інтеграція систем з використанням сервісно-орієнтованого підходу має значний потенціал для створення довгострокової бізнес-цінності. Хоча найбільші дивіденди, ймовірно, будуть отримані на рівні всього підприємства, переваги та покращення можуть бути досягнуті й на нижчих рівнях інтеграції. Для відповідності очікуванням та реалізації потенційних переваг необхідна ретельна попередня оцінка та планування. Важливо глибоко зрозуміти та задокументувати бізнес-процеси, а також оцінити їх придатність до сервісного забезпечення.

Повторне використання як існуючих, так і майбутніх додатків є важливим бізнес-драйвером, що сприяє загальному успіху. Здатність до легкого масштабування може стати вирішальним фактором при виборі інтеграційної стратегії, готуючи організацію до майбутніх подій та сценаріїв. Включення ESB в корпоративну інфраструктуру надає можливість організувати розподілені сервіси та спростити їхнє управління. Традиційні методи розробки програмного забезпечення мають перейняти принципи та

використовувати переваги сервісно-орієнтованої парадигми. Замість витіснення існуючих підходів, слід прагнути до синтезу різних перспектив у симбіотичне ціле шляхом адаптації перевірених методологій.

# РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ SOA НА РІВНІ КОРПОРАТИВНИХ СЕРВІСІВ ДЛЯ ПІДВИЩЕННЯ ГНУЧКОСТІ ІНТЕГРАЦІЇ

## 3.1. Дедуктивний підхід до формування теоретичного підґрунтя та розробки інтеграційного рішення

Дослідження базувалося на застосуванні дедуктивного підходу, спрямованого на першочергове формування міцного теоретичного підґрунтя. Це підґрунтя слугувало основою для подальшої роботи та методологічним каркасом. Вибір дедуктивного підходу обґрунтований характером дослідного питання, яке передбачало тестування висунутої гіпотези в рамках вже існуючої теорії, а не розробку нової.

Отримані теоретичні знання були інтегровані у процес побудови та використовувалися як керівництво для оптимального управління проєктом розробки програмного забезпечення в даному контексті.

З цією метою було проведено пошук та аналіз релевантної наукової літератури. Відбір здійснювався з урахуванням актуальності для різних галузей знань, що були визнані критично важливими для успішного завершення роботи. Література включала монографії, статті з академічних журналів та публікацій, а також документи з електронних джерел. Більшість статей було отримано через електронну бібліотеку LTU за допомогою пошуку за ключовими словами, що включали:

- Інтеграція систем
- SOA (Сервіс-орієнтована архітектура)
- Розробка програмного забезпечення
- ESB (Корпоративна сервісна шина)

Критеріями відбору літератури слугували її релевантність, достовірність, наукова значущість та відповідність ключовим термінам дослідження. Хоча існують різні погляди на окремі питання, аналіз

літератури підтвердив обґрунтованість та загальноприйнятість основних передумов, на яких ґрунтується дана робота. Як зазначено у вступі, метою дослідження було не визначення єдиноможливого рішення (враховуючи існування множинності підходів у розробці програмного забезпечення), а розробка ефективного та відповідного рішення, що задовольняє вимоги специфічного контексту.

Загалом, дослідження відповідало етапам процесу розробки програмного забезпечення, описаним у теоретичному розділі, задля отримання рішення, що задовольняє визначені вимоги та відповідає на поставлене питання.

Методологічна послідовність реалізації проєкту включала наступні етапи:

1. Збір та аналіз даних
2. Встановлення вихідних умов (базової лінії)
3. Технічний аналіз існуючої системи/програми
4. Проектування та розробка
5. Проектування рішення
6. Реалізація (Імплементація)
7. Розгортання
8. Тестування
9. Валідація рішення
10. Аналіз результатів

## 3.2. Методологія реалізації проекту

### *3.2.1. Збір та аналіз даних*

Збір емпіричних даних було здійснено на етапі інженерії вимог у процесі виявлення. Початковим етапом було накопичення достатнього обсягу інформації щодо існуючої ситуації, зацікавлених сторін, а також технічних та

економічних обмежень з метою формування уявлення про стан "як є". Інформація отримувалася шляхом проведення особистих та онлайн-зустрічей, інтерв'ю та аналізу наявної документації. Результати цих досліджень були консолідовані у базову модель, що стала вихідною точкою для подальшої роботи. Ключовою метою встановлення базових умов було з'ясування способів комунікації між залученими системами на поточному етапі, як на низькому технічному рівні, так і на вищому архітектурному рівні.

Виявлені поточні обмеження та недоліки призвели до формування нових вимог, які були зібрані та документовані під час фази інженерії вимог.

Для вивчення архітектури та структури поточної програми було організовано зустріч з її початковим розробником. Мета полягала у отриманні детального розуміння внутрішньої будови програми, її архітектурної композиції, а також організації процесів та функцій. Цей аналіз, у поєднанні з аналізом зібраних вимог, став вхідними даними для проєктування нового рішення.

### *3.2.2. Проектування та розробка*

Як зазначалося вище, проектування та реалізація рішення відбувалися згідно з етапами моделі розробки програмного забезпечення та були застосовані в ітеративній формі. Це передбачало проведення кількох циклів перегляду та уточнення проєктування і реалізації для досягнення фінального рішення.

Фактичне рішення було спроєктоване на основі вимог проєкту, раніше визначених обмежень та доступних ресурсів. З метою забезпечення можливості майбутньої інтеграції додаткових систем було прийнято рішення про включення веб-сервісів. Ці сервіси можуть слугувати точкою доступу до процесу через мережу Інтернет з використанням протоколу HTTP.

Реалізація здійснювалася в ітеративній та інкрементальній формі. Окремі функціональні блоки розроблялися, тестувалися та інтегрувалися у

систему з поступовим розширенням її можливостей для відповідності вимогам. Цей підхід був обраний як найбільш доцільний, оскільки він не лише дозволяв проводити ізольоване тестування кожного інкременту перед його інтеграцією з іншими компонентами, але й підвищував ймовірність отримання працездатної системи у визначені терміни, навіть якщо не всі функції були імплементовані.

Додаток було розгорнуто локально на сервері компанії після встановлення та конфігурації необхідного програмного забезпечення. Для цього було створено віртуальний сервер з метою мінімізації потенційного впливу на інші процеси у випадку виникнення непередбачених ситуацій.

Тестування проводилося у двох окремих фазах: тестування окремих функцій та компонентів, а також тестування завершеного продукту. Перша серія тестів виконувалася як невід'ємна частина процесу розробки. Методологія гнучкої розробки забезпечує можливості для тестування кожного інкременту перед переходом до наступного. Цей підхід має переваги, оскільки процес тестування інтегрований у розробку, а не відкладається на кінцеві етапи, коли часові ресурси обмежені. Крім того, такий підхід є органічним для розробника, оскільки тестування окремих функцій та методів є прямолінійним, а його важливість очевидна.

На другій фазі здійснювалося комплексне тестування всього потоку даних, від читання вхідного файлу з записами співробітників до запиту однієї бази даних та запису в дві інші. Тестування проводилося у двох різних середовищах: середовищі розробника та тестовому середовищі компанії.

На завершальному етапі здійснювалося зіставлення реалізованого рішення з початково встановленими вимогами з метою отримання підтвердження від зацікавлених сторін щодо адекватного задоволення визначених умов та вимог. Цей процес валідації відбувся під час зустрічі між автором роботи та власником проєкту в компанії.

*3.2.3. Аналіз результатів*

Результати фази реалізації, а саме розроблений прототип та показники його функціонування в конкретному контексті, були зіставлені з результатами аналізу літератури. Це дозволило оцінити відповідність отриманих практичних результатів теоретичним положенням, а також виявити можливі відхилення чи розбіжності. Ключові висновки літературного огляду використовувалися як критерії для перевірки відповідності аспектів розробленого рішення.

Однією з ключових вимог було створення рішення, яке легко масштабується для інтеграції додаткової функціональності, компонентів або систем. Також існували очікування щодо універсальності рішення для впровадження у різних інтеграційних контекстах в інших регіонах/країнах. Для повного розуміння вимог, що могли б виникнути в інших країнах, був необхідний додатковий збір та аналіз даних. Однак, через обмеження у часі та фінансових ресурсах, цей аспект дослідження не був реалізований, що відобразилося на обсязі проєкту.

Іншою вимогою була можливість інтеграції додаткових систем, оцінка якої є складною. Незважаючи на важливість цієї вимоги, тестування здатності інших систем взаємодіяти та обмінюватися інформацією з розробленим рішенням виходить за межі обсягу даної дисертації і тому не було враховано на цьому етапі. Слід зазначити, проте, що імплементація веб-сервісів забезпечує можливість такої інтеграції, принаймні частково.

Оскільки значна частина розробки та побудови здійснювалася "поза офісом", доступ до програмних інструментів був обмежений інструментами з відкритим кодом та безкоштовним програмним забезпеченням. Як наслідок, вибір програмних продуктів був обмежений виключенням комерційних рішень, що вимагають придбання ліцензій. Це потенційно могло мати негативний вплив, оскільки деякі продукти, що могли б дати кращі результати, не розглядалися. Однак, з огляду на те, що бажаним результатом

є отримання працездатної архітектури у даному контексті, а не обов'язково найкращої чи найбільш ефективної, цим потенційним недоліком можна знехтувати. Крім того, перевагою використання систем з відкритим кодом є їхня орієнтація на відкриті стандарти, тоді як деякі пропрієтарні рішення використовують власні протоколи, що суттєво обмежує гнучкість. Більш того, бюджетні обмеження не дозволяли придбання будь-яких ліцензій на той момент.

При оцінці достовірності (validity) та надійності (reliability), доречними є наступні концепції. Підтвердження достовірності шукалося шляхом обговорення вимог та результатів з представниками компанії. Використання перевірених методологій також сприяло підвищенню достовірності. Перенесення результатів (transferability) може бути можливим, якщо результати дослідження узгоджуються та підтверджуються висновками літературного огляду, тобто досягається певний рівень конвергенції. Щодо надійності, вважається ймовірним, що при повторенні дослідження в інших умовах, але за тих самих обставин і контексту, результат буде аналогічним або принаймні дуже схожим. Це пояснюється тим, що у роботі використовувалися добре встановлені та загальновизнані теорії, представлені у відповідних теоретичних розділах та літературному огляді.

### 3.3. Концептуальне проєктування та аналіз поточної системи

Цей розділ представляє результати роботи, що охоплюють початкові етапи життєвого циклу розробки програмного забезпечення, включно з етапом проєктування. Він починається з опису встановлення базових умов (базової лінії), після чого переходить до етапу інженерії вимог у рамках процесу розробки програмного забезпечення.

Перед початком етапу інженерії вимог було необхідно сформувати чітке розуміння поточної ситуації. Огляд існуючого стану, представлений на

рисунку 3.1, ілюструє взаємозв'язок різних систем. У цій конфігурації скрипт активується після отримання інформації у форматі файлу CSV, обробляє її та зберігає дані у базі даних, а також ініціює надсилання електронних листів відповідним отримувачам. Важливо зазначити, що поточне проміжне програмне забезпечення (middleware) складається з комплексу кількох скриптів.

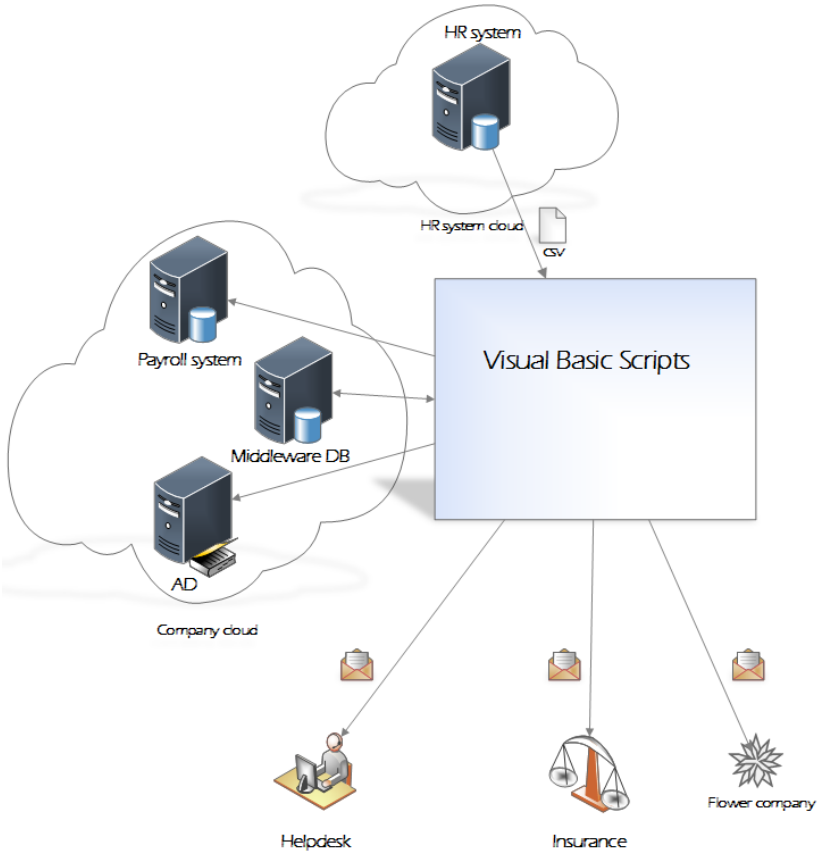


Рисунок 3.1 – Діаграма топології базової лінії

На рисунку 3.2 представлена діаграма активності, що відображає логіку процесу.

Детальний опис етапів процесу наведено нижче:

1. Отримання файлу даних із системи управління персоналом (HR).
2. Запуск скрипту імпорту.
3. Збереження копії файлу у базі даних проміжного програмного забезпечення.

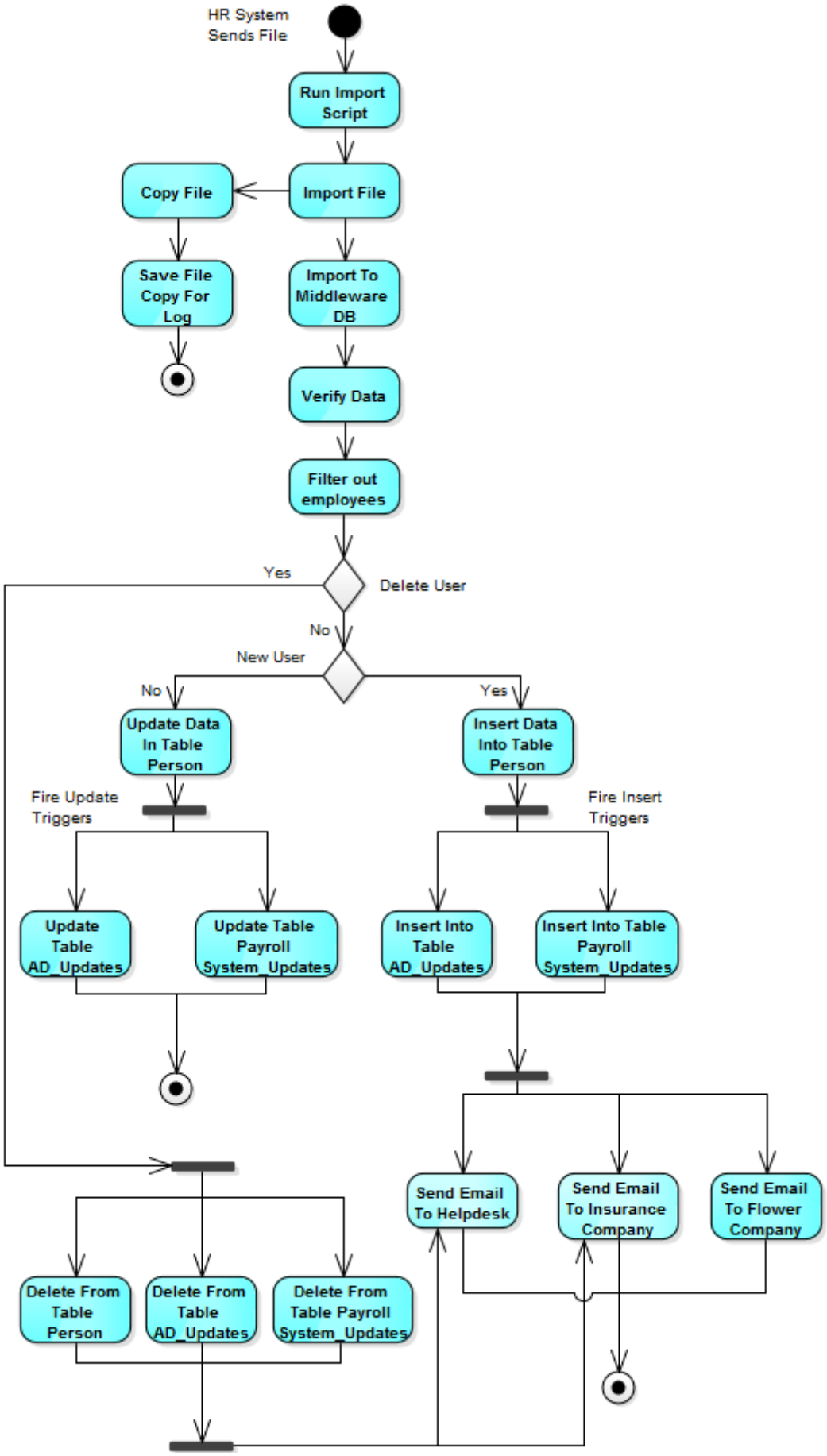


Рисунок 3.2 – Діаграма активності, що показує логіку операції.

4. Валідація отриманих даних.

5. Фільтрація записів про співробітників.

6. Порівняння даних з імпортованого файлу з даними, що зберігаються у базі даних проміжного програмного забезпечення.

7. У разі виявлення змін здійснюється оновлення бази даних та активуються відповідні тригери, які викликають: a. Скрипт оновлення для служби каталогів Active Directory (AD). b. Скрипт оновлення для системи обліку заробітної плати.

8. При додаванні нового користувача до AD, надсилаються електронні повідомлення до служби підтримки, страхової компанії та компанії, що надає послуги доставки квітів.

9. При видаленні користувача з AD, надсилаються електронні повідомлення до служби підтримки.

Скрипти розміщені на сервері бази даних проміжного програмного забезпечення та запускаються за розкладом як заплановані завдання у визначений час. У випадку найму нового співробітника, йому або їй надається корпоративне страхування та компліментарний букет квітів, що реалізується через функціонал надсилання листів до відповідних компаній. Служба підтримки отримує сповіщення про нових користувачів і відповідає, зокрема, за створення облікових записів користувачів та призначення відповідних ліцензій в Office 365.

Системи, залучені до цього процесу, включають систему управління персоналом (HR), систему обліку заробітної плати (Payroll System), Active Directory (AD) та базу даних проміжного програмного забезпечення. Таблиці в базі даних проміжного програмного забезпечення містять дані сутностей: Person, AD update та Payroll System update.

Необхідно відзначити, що існуюча архітектура системи HR та способи її інтеграції з іншими системами створили певні обмеження при розробці прототипу. Зокрема, передача файлів даних з регулярними інтервалами, що

відповідає парадигмі пакетної обробки даних, замість динамічної обробки змін у реальному часі, зумовлена саме цією архітектурою. Наразі API для підтримки динамічної обробки відсутній, тому запропонований прототип використовує ту ж структуру передачі файлів, що й поточне рішення.

## 3.4. Виконання інженерії вимог

Даний етап роботи присвячений процесу збору та специфікації вимог до проєкту, який здійснювався відповідно до спіральної моделі інженерії вимог. Збір користувацьких вимог проводився шляхом організації зустрічей з кінцевими користувачами, розробником існуючої системи та IT-менеджером.

### *3.4.1. Специфікація бізнес-вимог*

Бізнес-вимоги, ідентифіковані на цьому етапі, випливають із бізнес-процесів, для яких система має забезпечити інфраструктурну підтримку, і були визначені наступним чином:

- Реєстрація нового співробітника (процес адаптації, on-boarding)
- Завершення трудових відносин (процес звільнення, leaving)
- Внесення змін до інформації про співробітника
- Оновлення персональних даних (адреса, контактна інформація, дані найближчих родичів тощо)
- Зміни у даних про заробітну плату
- Дослідження можливостей (Feasibility study)

Обґрунтуванням проєкту слугує зростаюча потреба компанії у підвищенні ефективності системної комунікації та забезпеченні більш надійної та дієвої обробки інтеграційних процесів. Розробник поточної програми вказав на вичерпання можливостей подальшого додавання коду для реалізації нових функцій без ризику порушення стабільності її роботи. Фактично, програма досягла ліміту свого розвитку та потребує заміни.

*3.4.2. Виявлення та специфікація вимог користувача*

Вимоги, ідентифіковані на цьому етапі, представлені нижче з короткими поясненнями:

- Відстежуваність (Traceability) – Усі транзакції мають бути відстежуваними, потенційно шляхом зберігання у окремій таблиці бази даних.

- Повідомлення (Notification) – Система повинна надавати чіткі повідомлення у випадку неуспішного завершення транзакції, деталізуючи причину збою.

- Масштабованість (Scalability) – Рішення має бути легко масштабованим (як у бік збільшення, так і зменшення можливостей) для інтеграції, наприклад, інших систем та/або бізнес-процесів.

- Реалізація всіх поточних функцій (All current functions implemented) – Увесь функціонал існуючого скрипту повинен бути включений до нового рішення.

- Легше ідентифікувати обов'язкові поля форми (Easier to identify mandatory form fields) – З точки зору користувача, має бути інтуїтивно зрозуміло, яка інформація є обов'язковою для введення в систему.

- Покращена надійність (Improved reliability) – Система повинна забезпечувати вищу надійність для гарантування безперебійної обробки транзакцій.

- Покращена продуктивність (Improved performance) – Необхідне швидше та частіше виконання транзакцій; транзакції мають оброблятися частіше, ніж у поточній системі, бажано у режимі, наближеному до реального часу, замість щоденної пакетної обробки.

- Краще оброблення змін (Better change handling) – Реалізація нових функцій та адаптація до змінних вимог має бути спрощеною.

Резюме функціональних та нефункціональних вимог представлено у таблиці 3.1.

Таблиця 3.1 – Зведений перелік вимог

| **Функціональні** | **Нефункціональні** |
|---|---|
| Відстежуваність | Покращена надійність |
| Повідомлення | Покращена продуктивність |
| Масштабованість | Краще оброблення змін |
| Всі поточні функції реалізовані | Обов'язкові поля форми легше ідентифікувати |

Таблиця 3.2 демонструє пріоритезацію вимог відповідно до методу MoSCoW, де "1" позначає найвищий пріоритет, що відповідає обов'язковій вимозі ("Must have"), а далі за спаданням.

Таблиця 3.2 – Пріоритезація вимог за методом MoSCoW

| **1** | **2** | **3** | **4** |
|---|---|---|---|
| Поточні функції | Відстежуваність | Покращена продуктивність | Обов'язкові поля форми |
| Масштабованість | Повідомлення | Покращена надійність | |
| | Краще оброблення змін | | |

*3.4.3. Виявлення та специфікація системних вимог*

Бажаною платформою для розміщення нового проміжного програмного забезпечення є хмарна інфраструктура з використанням існуючої корпоративної платформи Microsoft Azure. Проте, якщо цей варіант виявиться неможливим або недоцільним з певних причин, допускається розміщення системи локально (on-premise).

Вимоги, описані у таблиці 3.1, були валідовані у консультації з одним із зацікавлених сторін, зокрема власником проєкту. Метою перегляду було

отримання формального підтвердження вимог перед переходом до наступної фази процесу розробки. Крім того, процес перегляду сприяв встановленню пріоритезації вимог, представленої у таблиці 3.2. Слід зазначити, що не всі зацікавлені сторони були залучені до процесу валідації, оскільки власник проєкту вважав їхню участь на цьому етапі необов'язковою.

## 3.5. Планування проєкту

На даному етапі основним результатом стало формування плану проєкту, що включав графік виконання робіт, перелік необхідних ресурсів, а також зведення технічних обмежень проєкту та обґрунтування прийнятих рішень.

Структура плану проєкту відповідала послідовності етапів процесу розробки програмного забезпечення. План розглядався як попередній, оскільки визначені часові рамки етапів слід інтерпретувати як орієнтовні оцінки тривалості, а не фіксовані дати. Крім того, застосування ітеративного підходу зумовило значне перекриття багатьох етапів та їхню одночасну реалізацію, особливо це стосується фаз проєктування/моделювання та реалізації. Графік виконання основних етапів представлено у таблиці 3.3.

Таблиця 3.3 – Графік виконання етапів проекту

| Етап | Дата початку | Кінцевий термін |
|---|---|---|
| Інженерія вимог | 2025-02-15 | 2025-03-04 |
| Планування | 2025-02-22 | 2025-02-28 |
| Проектування та моделювання | 2025-03-05 | 2025-04-24 |
| Реалізація | 2025-04-04 | 2025-05-08 |
| Розгортання | 2025-05-09 | 2025-05-15 |
| Валідація | 2025-05-12 | 2025-05-22 |

Перелік ресурсів, задіяних у проєкті, включав:

- Технічні ресурси:
  - Середовище розробки
  - Програмне забезпечення
  - Додаткове обладнання/інструменти
- Персонал:
  - Зацікавлені сторони (стейкхолдери)
  - Розробник/дослідник (автор)
- Часові ресурси

Були прийняті наступні ключові технічні рішення:

Мова програмування: Java. Обрано завдяки її платформенній незалежності та сумісності з Mule ESB, який також базується на Java.

Веб-сервіс: Заснований на SOAP з використанням XML. Рішення на користь SOAP порівняно з RESTful веб-сервісами прийнято з двох причин: Mule забезпечує кращу підтримку для SOAP-сервісів, і SOAP надає розширені можливості безпеки, які часто відсутні у більш легковагих RESTful сервісів.

Шина даних (ESB): Mule.

Система управління базами даних (СУБД): MySQL (використовувалася у тестовому середовищі). Вибрано як рішення з відкритим кодом, що володіє всією необхідною функціональністю для цілей тестування та реалізації даного проекту.

Середовище розміщення: Windows.

Операційна система: Windows 10, 64-бітна.

Обґрунтування вибору Java, SOAP-сервісів та MySQL було продиктоване як технічними перевагами цих рішень (платформенна незалежність Java, функціональність та безпека SOAP, можливості MySQL як відкритого рішення), так і сумісністю компонентів (Java та Mule ESB).

## 3.6. Проектування та моделювання

Мета на цьому етапі полягала в тому, щоб надати модель проектування, яка б показала, як рішення має бути реалізоване на наступному етапі. Модель пройшла дві ітерації перед тим, як досягла кінцевого стану, як показано на рисунку 3.3.

### *3.6.1. Ітерація 1*

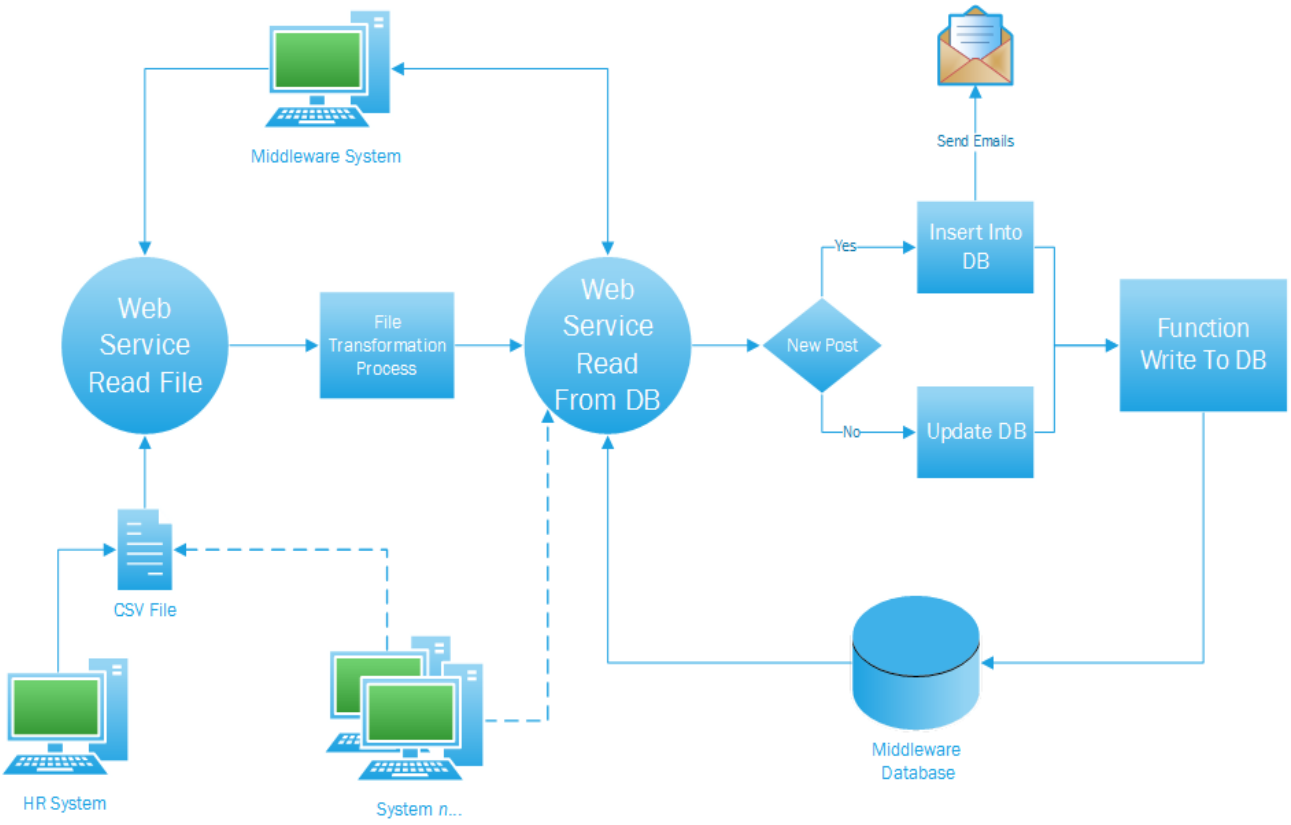


Рисунок 3.3 - Потік процесу, ітерація 1

Перший проектний ескіз зображений на рисунку 3.3, що є високорівневою ілюстрацією потоку процесу. У цій першій ітерації ідея полягала в тому, щоб використовувати два різні веб-сервіси для обробки: перший займається читанням файлу CSV у пам'ять, а другий обробляє доступ до читання з бази даних. Для запису в базу даних, тобто оновлення або

вставки нових записів, це не експонується як веб-сервіс, а скоріше як внутрішня функція системи, ідея полягає в тому, що це безпечніше. Перед записом у базу даних вибір між оновленням вже існуючого співробітника або вставкою нового визначає, чи надсилаються електронні листи до страхової компанії, служби підтримки та квіткової компанії відповідно.

Веб-сервіси служать як точки доступу для поточних систем, тобто проміжного програмного забезпечення та системи HR, але також можуть бути використані іншими внутрішніми або зовнішніми системами, зображеними на фігурі як система n.

Поділ функціональності на два окремі веб-сервіси був спроектований з урахуванням того, що це створить більш інтуїтивний потік повідомлень з більш чітким розмежуванням відповідальностей. Як стане очевидним у наступній ітерації проекту, однак, це довелося поступитися іншими міркуваннями.

### *3.6.2. Ітерація 2*

У другій ітерації два веб-сервіси були об'єднані в один, який містить усі методи та функції окремих веб-сервісів (рис. 3.4). Один веб-сервіс читає файл CSV, а потім оновлює або вставляє нового співробітника в базу даних і контролює надсилання електронних листів, як і раніше. Основною причиною об'єднання веб-сервісів в один був збільшення продуктивності системи. Маючи необхідність спілкуватися з базою даних двічі в потоці для виконання всіх необхідних завдань, як це було зроблено в першій ітерації, процес був значно повільнішим.

З усіма доступом до бази даних, що відбувається в "чорному ящику" веб-сервісу, було отримано значне покращення продуктивності. Це покращення продуктивності довелося зважувати проти ризику експонування доступу до запису в базу даних через інтернет.

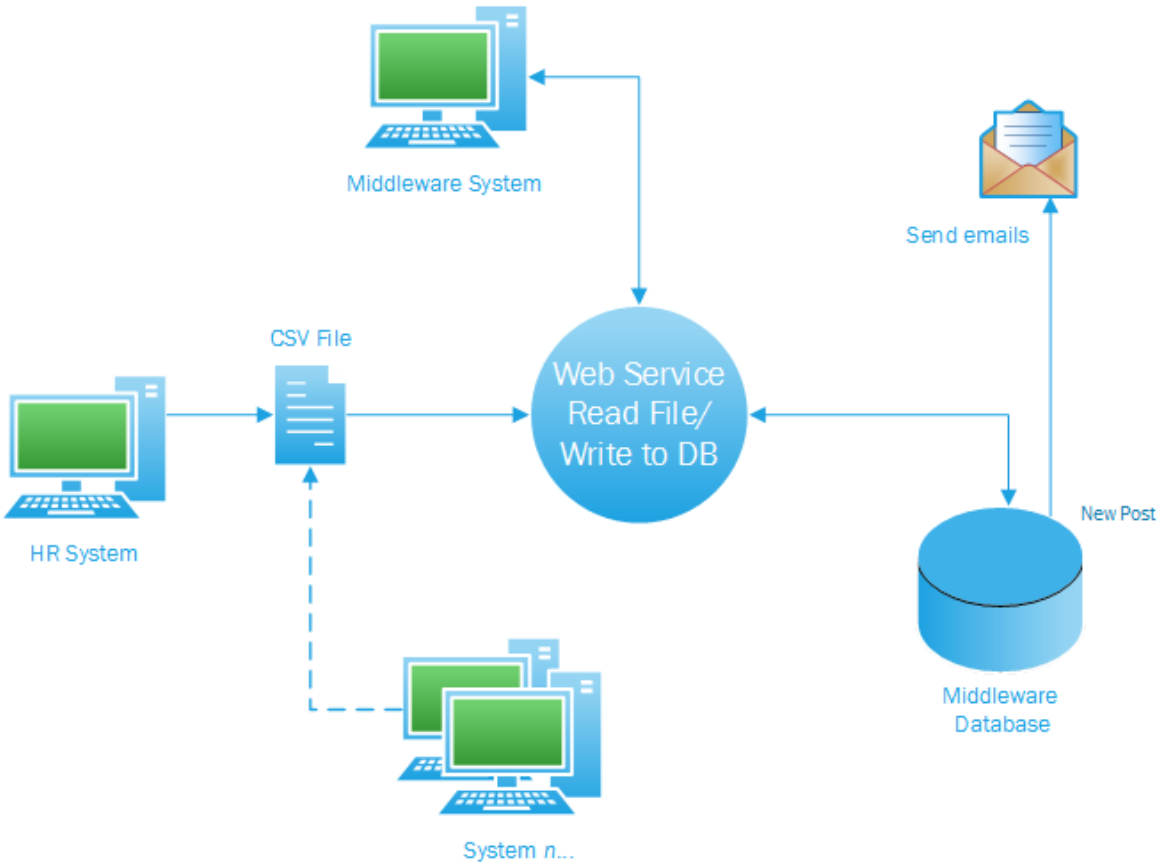


Рисунок 3.4 - Потік процесу після другої ітерації

Фактичний потік на трохи більш деталізованому рівні можна побачити на діаграмі активності на рисунку 3.5, що можна порівняти з оригінальним потоком з рисунку 3.2. Хоча і не передаючи всі деталі з міркувань читабельності, рівень абстракції все ще надає хороший огляд всього процесу від початку до кінця. Основні відмінності, на які слід звернути увагу, полягають у тому, що тут використовується лише один тригер бази даних для оновлення іншої бази даних, і що якщо будь-які винятки викидаються під час процесу, спочатку зберігається файл помилки, а потім він надсилається електронною поштою власнику системи.

Речі, які навмисно відсутні на рисунку 3.4, то на 3.5 це зображення кількох баз даних та потоку завершення роботи співробітника, який, до речі, вирішується шляхом перемикання стовпця під назвою active в базі даних замість фактичного видалення запису. Цей спосіб реалізації

використовується з статистичних причин, якщо виникне потреба в даних попередніх співробітників в якийсь момент у майбутньому.

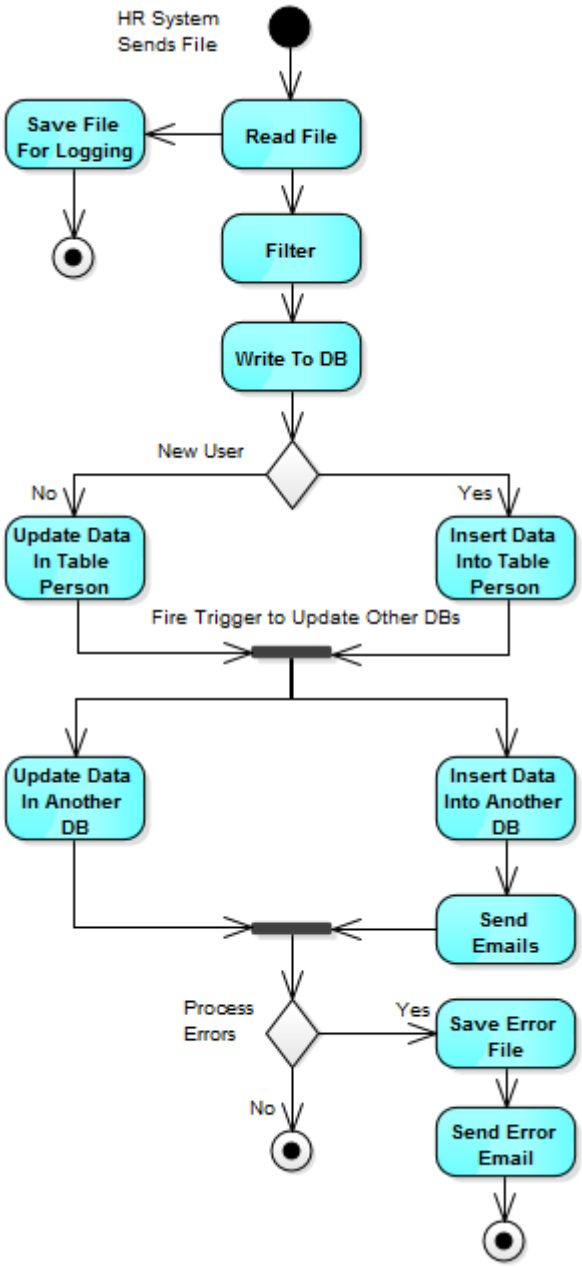


Рисунок 3.5 - Діаграма активності переглянутої моделі дизайну

*3.6.3. Вплив на безпеку*

При використанні веб-сервісів, які спілкуються через інтернет, вони роблять це через небезпечний протокол HTTP. Тут повідомлення надсилаються у вигляді звичайного тексту, що означає, що вони видимі для

всіх, що, очевидно, є небажаним, особливо якщо йдеться про чутливу інформацію. Щоб вирішити це, зазвичай застосовується шифрування, щоб зберегти конфіденційність даних. Щоб забезпечити, що користувачі веб-сервісів є довіреними, тобто що вони аутентифіковані та авторизовані постачальником сервісу, може бути використаний метод аутентифікації на основі, наприклад, паролів, токенів, цифрових підписів або сертифікатів. Повідомлення також повинні бути захищені від зміни, тобто має бути можливість зберегти та перевірити цілісність. Інша важлива проблема - це доступність сервісу; якщо сервіс стає недоступним на певні періоди часу через, наприклад, втрату інтернет-з'єднання або простою сервера, як це вплине на бізнес, які наслідки та наслідки.

При роботі конкретно з безпекою веб-сервісів, WS-Security (Web Service Security) - це збірка стандартів безпеки, розроблених для застосування в контексті SOA. Він може бути розглянутий як розширення протоколу повідомлень SOAP. Деякі з найважливіших та найчастіше використовуваних стандартів - це WS-Policy, SAML, XML Signature та XML Encryption. Крім того, WS-Security є частиною WSIT, яка також містить послуги оптимізації повідомлень та надійних повідомлень. WSIT буде використаний під час реалізації безпеки, оскільки він інтегрований в IDE, який використовується.

У цьому проекті, фокус безпеки буде спрямований на захист конфіденційності та цілісності даних, а також на захист веб-сервісу від несанкціонованого доступу. Оскільки питання доступності значною мірою залежить від того, хто і як розміщує сервіс, це не буде розглянуто тут, оскільки це більше питання для самої компанії, чи вони вибирають розмістити сервіс на локальному сервері або на зовнішньому хмарному сервері.

Однією з переваг використання ESB є інтегрована підтримка безпеки, яка зробить можливою реалізацію додаткових функцій безпеки, якщо це буде вирішено в майбутньому.

## 3.7. Імплементація та валідація

Спочатку цей розділ описує процес імплементації, а також результат у вигляді діаграми разом з відповідним поясненням потоку повідомлень. Потім коротко описується розгортання додатка та його оцінка.

### *3.7.1. Процес імплементації*

Реалізація почалася з побудови класів Java, які містили б об'єкти та методи, необхідні для досягнення бажаних операцій. Були створені методи для читання файлів та спілкування з базою даних разом з додатковими "допоміжними" методами. Повна діаграма класів показана на рисунку 3.6. Ці класи становили основу для подальшої розробки веб-сервісу з відповідними методами.

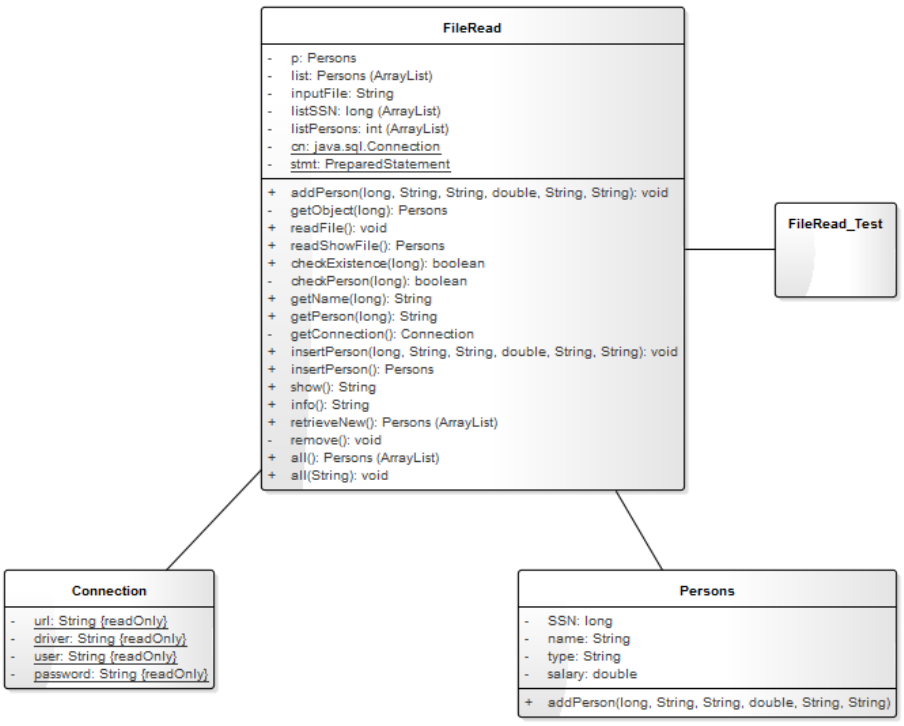


Рисунок 3.6 – Діаграма класів

Оскільки реалізація відбувалася одночасно з пізнішими етапами фази проектування, рішення перейти від двох веб-сервісів до одного вплинуло на макет класів. Спочатку було розділення класів, де один займався читанням та трансформацією файлів, а інший — читанням з бази даних та записом у неї. Коли два веб-сервіси були об'єднані в один, ці два класи також були об'єднані в один з тих же причин.

Як описано в розділі тестування, всі методи та функціональність програми були повністю протестовані під час розробки, як окремо, так і в комбінації. Для цього був написаний окремий клас тестування. Крім того, були побудовані дві бази даних, кожна з яких містила по одній таблиці з сімома та трьома стовпцями відповідно.

Кілька слів про обробку помилок у прототипі, яка була зведена до мінімуму, можуть бути доречними тут. Це було зроблено з двох причин: економія часу під час розробки та, оскільки вхідними даними для процесу був файл, що містить записи співробітників, насправді не було необхідності реалізовувати будь-яку особливу обробку помилок в іншому місці потоку на цей момент часу.

Після завершення веб-сервісу наступним кроком було налаштування середовища ESB, яке мало, серед іншого, споживати веб-сервіс та виконувати функції обробки помилок та надсилання електронних листів. Це також було зроблено ітеративно, тобто різні функції, такі як налаштування обробки помилок, надсилання електронних листів, споживання веб-сервісів, трансформація повідомлень тощо, спочатку тестувалися окремо, перш ніж бути зібраними в повний потік. Результат показано графічно на рисунку 3.7, за яким слідує пояснення кроків з повним кодом, який можна знайти в додатку В, XML-код додатку Mule.

Опис потоку процесу:

1. Веб-сервіс опитує папку на наявність нових файлів кожні 60 секунд.

2. Після доступу до файлу, файл перейменовується з поточною датою та оригінальним ім'ям файлу і переміщується в іншу папку для резервного копіювання.

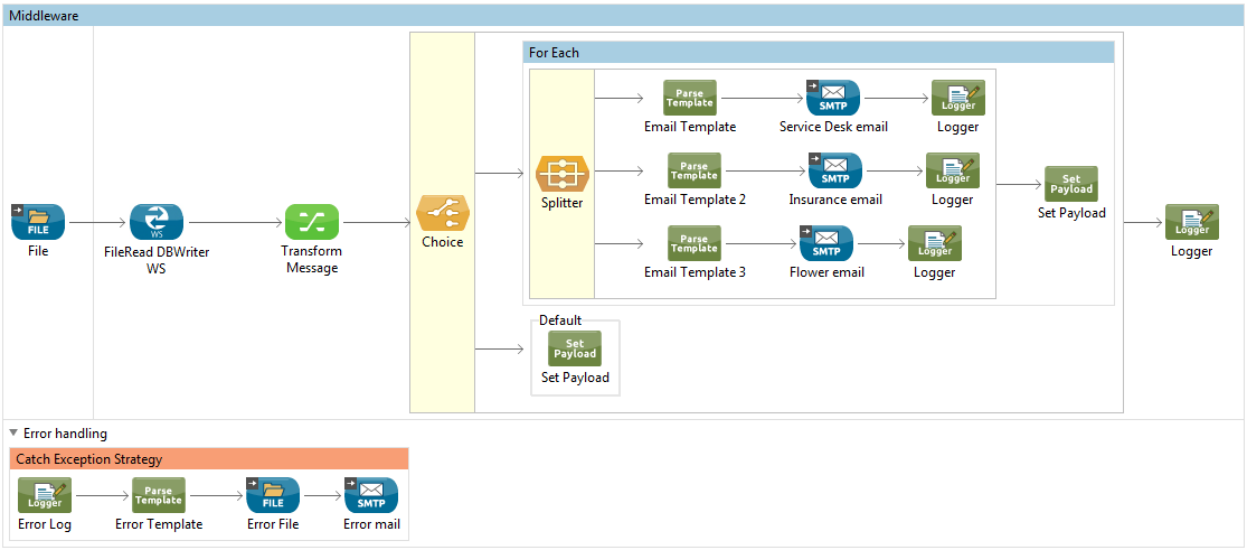


Рисунок 3.7 - Кінцева імплементація в Mule ESB

3. Веб-сервіс виконує операції "чорного ящика": a. Читає файл b. Фільтрує співробітників, які не належать до правильного філіалу c. Записує файл у базу даних d. Тригер у базі даних виконує команду вставки або оновлення відповідно в іншу базу даних e. Повертає нових співробітників, якщо такі є

4. Вивід з веб-сервісу трансформується з XML у String для подальшої обробки.

5. Три різні шаблони електронних листів заповнюються змінними співробітників з бази даних.

6. Електронні листи надсилаються з шаблоном як тілом повідомлення трьом отримувачам відповідно. Якщо крок 3e повертає більше одного результату, кроки 4-6 повторюються, поки всі результати не будуть оброблені.

7. Винятки, які можуть бути викинуті під час процесу, спочатку зберігаються у файл у локальному каталозі помилок, а потім надсилаються як вкладення електронної пошти власнику системи.

Пояснення деяких особливостей Mule

A. Елемент вибору є розгалуженням, яке дозволяє потоку йти різними шляхами на основі певних критеріїв. У цьому випадку верхній шлях вибирається, якщо вивід з веб-сервісу містить будь-які елементи, тобто нові співробітники.

Нижній шлях, який також є шляхом за замовчуванням, вибирається, коли вивід є нульовим, тобто коли не додаються нові співробітники, інформація просто оновлюється.

B. Розділювач, або scatter-gather, як називається елемент у Mule, спочатку розділяє потік на три окремі, які виконуються паралельно, перш ніж нарешті агрегувати корисне навантаження повідомлення в одне повідомлення. Його функція тут полягає в тому, щоб надіслати ті самі дані співробітника для заповнення трьох шаблонів електронних листів, які надсилаються трьом різним отримувачам відповідно.

C. Для кожного працює як цикл, розділяючи колекцію на елементи перед обробкою їх ітеративно та повертаючи оригінальне повідомлення в потік.

Якщо вивід з веб-сервісу є списком, який містить кілька елементів списку, кожен елемент надсилається через потік один за одним, поки всі елементи не будуть оброблені.

D. Якщо під час виконання потоку виникають будь-які помилки або викидаються винятки, помилки обробляються стратегією винятків повідомлень, як описано вище в кроці номер 7.

E. Елемент логування використовується для логування виводу з потоку повідомлень, наприклад, у вигляді повідомлень про статус або винятків.

*3.7.2. Реалізація безпеки*

Функції безпеки, які були реалізовані на серверній стороні додатку, тобто самому веб-сервісі, використовували вищезгадану платформу WSIT. Фактичний метод, обраний для реалізації, був аутентифікація користувача з симетричними ключами, оскільки це задовольняло потреби в конфіденційності та цілісності. Шифрування з використанням симетричного ключа означає, що один секретний ключ використовується як для шифрування, так і для дешифрування, де ключ є спільним між відправником та отримувачем, що в цьому випадку відповідає серверній та клієнтській сторонам відповідно. Рівень безпеки при використанні ключа для шифрування повідомлень значною мірою залежить від розміру або довжини цього ключа, де коротший ключ легше зламати, ніж більший, та алгоритму шифрування, який використовується. Для прототипу був використаний 192-бітний ключ та базовий набір алгоритмів, який вважався достатнім для цієї мети; доступне більш просунуте шифрування, якщо це буде необхідно.

*3.7.3. Розгортання*

Процес розгортання (deployment) було виконано у два етапи. На першому етапі було завантажено, інстальовано та конфігуровано необхідне програмне забезпечення для функціонування додатку на віртуальному сервері компанії. Після цього було встановлено та налаштовано інтеграційну шину Mule ESB та створено тестові бази даних. Створення структури двох тестових баз даних та відповідних таблиць здійснювалося за допомогою скриптів мови визначення даних (DDL).

Другий етап передбачав розгортання безпосередньо веб-сервісу на сервері додатків, роль якого у даному випадку виконував Glassfish. Оскільки інсталяція та розгортання всіх компонентів здійснювалися у виділеному віртуальному середовищі, був відсутній ризик інтерференції додатку з іншими існуючими програмами чи процесами у разі виникнення

позаштатних ситуацій. Відповідно, необхідність у розробці плану відкату (rollback plan) чи резервного плану (backup plan) була відсутня.

*3.7.4. Валідація*

Процес валідації (validation) полягав у верифікації відповідності кожного з пунктів вимог, зібраних на етапі інженерії вимог (див. таблицю 3.2), та оцінці ступеня їх задоволення реалізованим рішенням. Серед вимог двох найвищих категорій пріоритетності (згідно з методом MoSCoW), всі, окрім вимоги щодо кращої обробки змін, були підтверджені та верифіковані як реалізовані у прототипі. Причиною відсутності валідації вимоги щодо кращої обробки змін переважно була відсутність репрезентативного методу оцінки та кількісного вимірювання її присутності на даному етапі. Показники покращення продуктивності та надійності не були встановлені/оцінені, оскільки відповідні метрики та тести продуктивності/надійності не проводилися, і їхнє об'єктивне та коректне проведення було б складним.

Валідація у ширшому контексті, що стосується підтвердження відповідності результатів теоретичним положенням, буде розглянута в наступному розділі.

## 3.8. Аналіз та обговорення результатів проєктування та реалізації

Незважаючи на те, що змодельований бізнес-процес представляв собою сценарій інтеграції обмеженого масштабу, існувало обґрунтоване очікування, що в розробленому прототипі будуть продемонстровані певні покращення. Серед ключових вимог були масштабованість рішення та покращене повідомлення про події, зокрема в контексті обробки помилок.

Включення веб-сервісу як точки входу, до якої можуть підключатися інші системи, створює архітектуру, здатну до розширення та масштабування прямим чином. Можна припустити, що використання лише одного веб-

сервісу може суттєво обмежити зручність використання та можливості взаємодії для інших систем порівняно з реалізацією кількох сервісів. Однак, зважаючи на те, що завдання полягало у створенні прототипу, а не повноцінної виробничої системи, переваги архітектурного підходу та його наслідки залишаються очевидними та чітко артикульованими.

Застосування ESB забезпечило ефективні вбудовані можливості обробки помилок, що значно перевершує функціонал поточної програми. Гнучкість конфігурації для отримання інформації з різних етапів потоку повідомлень, а також можливість вибору формату та обсягу відображуваної інформації є прикладами переваг, досягнутих завдяки впровадженню ESB.

Хоча розроблений веб-сервіс використовує лише один метод для виконання основного функціоналу, він містить кілька додаткових методів (таблиця 3.4).

Таблиця 3.4 – Методи веб-сервісу

| **Назва** | **Опис** |
|---|---|
| read | Читає файл у пам'ять |
| readShow | Читає файл у пам'ять та відображає результати як рядок |
| showFile | Показує співробітників зі списку |
| showPersons | Читає файл у пам'ять та відображає результати як об'єкти |
| info | Відображає всі записи з бази даних |
| getPerson | Отримує одного співробітника з бази даних |
| checkExistence | Запитує базу даних на наявність певного співробітника |
| all | Читає файл у пам'ять, записує його в базу даних, повертає нові записи (Це метод, використаний у рішенні) |

Ці методи можуть бути використані у двох сценаріях: вони можуть викликатися іншими споживачами сервісів шляхом побудови додаткових інтеграційних потоків, або на їх основі можуть бути створені менші веб-

сервіси, кожен з яких включатиме певний набір цих методів. Перший підхід є кращим, коли потрібна додаткова функціональність між взаємодіючими системами. Другий може виявитися більш ефективним для контролю доступу споживачів до конкретних методів сервісу. Це також відкриває потенціал для повторного використання коду, оскільки деякі сервіси можуть покращити або розширити функціонал, використовуючи існуючий код як основу. Прикладами є можливість ініціювати потік без вхідного файлу, вставляти записи до бази даних з джерел, відмінних від файлів, або отримувати дані безпосередньо з бази даних. Розробка графічного інтерфейсу користувача, що дозволяє вибирати методи для виклику, надасть доступ до функціоналу сервісу не тільки іншим системам, але й кінцевим користувачам. Важливо відзначити, що обидва ці підходи демонструють здатність розробленого рішення до масштабування та інтеграції більшої кількості систем, тим самим підвищуючи його гнучкість та загальну корисність.

При проєктуванні рішення попереднє поглиблене розуміння самого бізнес-процесу виявилося цінним, оскільки це сприяло адаптації та оптимізації інтеграційного потоку. Модифікація та спрощення потоку були зумовлені двома факторами: не всі функції були необхідні для побудови прототипу, і, що важливіше, оригінальний потік став надмірно складним через поступове, несистемне додавання функцій та можливостей з часом, і тому потребував оптимізації.

Аналогічно, хоча сам прототип не інтенсивно використовував попередній додаток, певні його компоненти можуть бути інтегровані у ширші, функціонально багатші майбутні рішення. Оскільки Mule ESB надає функціональність для включення скриптів (повністю або частково) як будівельних блоків у потік процесу, існують можливості для повторного використання існуючого коду.

Використання ESB як центрального вузла, до якого можуть підключатися сервіси, процеси та функції, сприяє підвищенню рівня

організації в загальному інтеграційному середовищі. Хоча цей ефект може бути менш очевидним у прототипі з його обмеженим функціоналом, переваги стануть більш вираженими при додаванні нових сервісів та можливостей. Управління інтегрованими системами буде істотно спрощене, особливо якщо компоненти розподілені між різними філіями компанії, хостинговими майданчиками або навіть перебувають у різних юрисдикціях. Графічний інтерфейс управління ESB забезпечує візуалізацію складових компонентів та їхніх зв'язків.

Проєктування класів Java, які підтримують веб-сервіс, було реалізовано в об'єктно-орієнтованому стилі. Це означає, що кожен клас спрямований на виконання специфічного завдання через певний об'єкт або набір методів з метою точного моделювання сутностей реального світу. Однак, стверджувати, що проєктування також здійснювалося з повним прийняттям сервіс-орієнтованої парадигми на нижчих рівнях реалізації, було б перебільшенням. У проєкті настільки обмеженого масштабу, як цей, вплив СОА-принципів не був суттєво помітним або значущим на рівні коду. Тим не менш, якщо цей прототип послужить основою для покращеного та більш розвиненого майбутнього інтеграційного рішення, був зроблений початковий крок до реалізації сервіс-орієнтованої архітектури.

Фінальне розгортання веб-сервісу було виконано локально на сервері компанії, на противагу використанню хмарного провайдера, такого як Azure. Причина цього полягала у спрощенні процесів розгортання, конфігурації та тестування додатку. Таким чином, веб-сервіс був доступний лише для локальних ресурсів, але вважається, що для демонстрації функціональності та загальної корисності прототипу цього є достатнім.

Хоча безпека для прототипу була реалізована мінімальним чином, існують значні можливості для посилення заходів безпеки у разі потреби. Незважаючи на наявність певних механізмів безпеки для RESTful веб-сервісів, їхні аналоги на основі SOAP мають значно більше функцій безпеки,

що, як було зазначено раніше, стало однією з причин вибору саме цього підходу для реалізації сервісів.

Отже, дана робота була спрямована на демонстрацію застосування сервісно-орієнтованої архітектури (СОА) як засобу оптимізації інтеграції систем. Використання веб-сервісів та інтеграційної шини підприємства (ESB) було ідентифіковано як ключовий підхід для досягнення цієї мети. За результатами дослідження було встановлено, що застосування принципів сервісної орієнтації спільно з впровадженням ESB здатне забезпечити суттєву цінність для проєктів інтеграції систем.

Розроблений прототип ефективно використовує веб-сервіс та ESB для реалізації функціоналу, що відповідає визначеним вимогам. Відповідність деяких із цих вимог була досягнута зокрема завдяки застосуванню сервісної орієнтації, яка відіграла важливу роль у процесі побудови та розвитку інтеграційного рішення.

Сервісно-орієнтована архітектура уможливлює створення гнучких та масштабованих систем, які легко адаптуються до змінюваних вимог та умов функціонування. Використання ESB сприяє централізації управління сервісами та забезпечує ефективну інтеграцію різнорідних систем і додатків.

Підсумовуючи, можна констатувати, що застосування сервісної орієнтації та ESB являє собою ефективний метод покращення інтеграції систем та забезпечення суттєвої цінності для організацій. Дане дослідження ілюструє, як ці технології можуть бути ефективно застосовані для досягнення успішної інтеграції систем та підвищення загальної продуктивності та ефективності бізнес-процесів.

# ВИСНОВКИ

В дипломній роботі було здійснено всебічний аналіз концепції сервісно-орієнтованої архітектури (SOA) та її впровадження на рівні корпоративних сервісів з метою забезпечення гнучкої та ефективної інтеграції інформаційних систем.

У першому розділі досліджено теоретичні засади SOA, розглянуто роль Enterprise Service Bus (ESB) у процесі інтеграції, а також обґрунтовано актуальність проблематики системної інтеграції у сучасному корпоративному середовищі. Особливу увагу було приділено взаємозв'язку між сервісно-орієнтованим підходом і життєвим циклом розробки програмного забезпечення, зокрема інженерії вимог.

Другий розділ поглибив розуміння архітектурних принципів SOA, визначив роль веб-сервісів як ключового механізму реалізації інтеграційних рішень, а також описав функціональні особливості ESB як проміжного шару, що забезпечує зв'язність, масштабованість і керованість сервісів у корпоративному середовищі. Було також проаналізовано практичні приклади застосування SOA та ESB, що надало змогу окреслити основні напрямки успішної реалізації системної інтеграції.

У третьому розділі здійснено безпосередню імплементацію SOA-рішення відповідно до розробленої методології. Застосовано дедуктивний підхід до побудови теоретичного підґрунтя проєкту, проведено інженерію вимог, концептуальне проєктування, планування, моделювання, реалізацію та валідацію інтеграційного рішення. Значну увагу приділено безпеці при реалізації сервісів, що є критично важливим аспектом для корпоративного IT-середовища. Результати проєктування та реалізації були проаналізовані й інтерпретовані, що дозволило зробити обґрунтовані висновки щодо ефективності застосованого підходу.

У результаті дослідження підтверджено, що використання SOA у поєднанні з ESB сприяє зниженню складності інтеграції, покращенню повторного використання сервісів, підвищенню гнучкості змін у бізнес-процесах та забезпеченню масштабованості корпоративних IT-систем. Запропоноване рішення демонструє практичну цінність і може бути використане як основа для побудови інтеграційних архітектур у різних галузях діяльності. Подальші дослідження можуть бути спрямовані на оптимізацію продуктивності сервісів, розширення засобів управління якістю сервісів та автоматизацію процесів інтеграції.

## ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Erl, T. (2005). Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR.
2. Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. Proceedings of the Fourth International Conference on Web Information Systems Engineering, 3–12.
3. Josuttis, N. M. (2007). SOA in Practice: The Art of Distributed System Design. O'Reilly Media.
4. Barry, D. K. (2003). Web Services and Service-Oriented Architectures: The Savvy Manager's Guide. Morgan Kaufmann.
5. Bieberstein, N., Bose, S., Fiammante, M., Jones, K., & Shah, R. (2006). Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap. IBM Press.
6. Garlan, D., & Shaw, M. (1994). An introduction to software architecture. Advances in Software Engineering and Knowledge Engineering, 1, 1–39.
7. Lankhorst, M. (2005). Enterprise Architecture at Work: Modelling, Communication and Analysis. Springer.
8. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., & Holley, K. (2008). SOMA: A method for developing service-oriented solutions. IBM Systems Journal, 47(3), 377–396.
9. Linthicum, D. S. (2003). Next Generation Application Integration: From Simple Information to Web Services. Addison-Wesley.
10. Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. Computer, 40(11), 38–45.
11. Demirkan, H., & Goul, M. (2006). Towards the Service-Oriented Enterprise Vision: Bridging Industry and Academics. Communications of the ACM, 49(12), 49–54.

12. Cardoso, J., Voigt, K., & Winkler, M. (2009). Service engineering for the internet of services. Enterprise Information Systems, 3(3), 225–239.
13. Yoon, V. Y., Carter, L., & Kang, D. (2015). SOA governance for flexibility and alignment in enterprise architecture. Journal of Computer Information Systems, 55(3), 59–66.
14. Hojaji, F., & Shirazi, M. R. (2010). A novel framework for agile SOA governance. 2010 IEEE International Conference on Service-Oriented Computing and Applications, 1–6.
15. Krafzig, D., Banke, K., & Slama, D. (2004). Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall.
16. Pulier, E., & Taylor, H. (2006). Understanding Enterprise SOA. Manning Publications.
17. Arsanjani, A., & Holley, K. (2006). The service integration maturity model: Achieving flexibility in the transformation to SOA. IBM Systems Journal, 45(4), 703–724.
18. Mohammadi, K., & Jowkar, A. (2013). An Enterprise SOA Maturity Model for Evaluating Service-Oriented Architecture Adoption. International Journal of Computer Applications, 71(14), 30–36.
19. Quartel, D., Steen, M. W. A., & Lankhorst, M. (2012). Application and project portfolio valuation using enterprise architecture and business requirements modelling. Enterprise Information Systems, 6(2), 189–213.
20. Bell, M. (2008). Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley.
21. Schekkerman, J. (2004). How to Survive in the Jungle of Enterprise Architecture Frameworks. Trafford.
22. Boh, W. F., & Yellin, D. (2006). Using enterprise architecture standards in managing information technology. Journal of Management Information Systems, 23(3), 163–207.

23. Sprott, D., & Wilkes, L. (2004). Understanding service-oriented architecture. CBDI Forum White Paper, 1(1), 1–32.
24. Zimmermann, O., Milinski, S., Craes, M., & Oellermann, F. (2004). Second generation web services-oriented architecture in production in the finance industry. IBM Systems Journal, 44(4), 703–719.
25. Joshi, A., & Rai, A. (2000). Impact of the alignment between information technology and supply chain integration on firm performance. Decision Sciences, 31(3), 413–447.
26. Jeston, J., & Nelis, J. (2008). Business Process Management: Practical Guidelines to Successful Implementations. Routledge.
27. Zhang, L.-J., & Zhou, Q. (2009). Enterprise-level SOA and service-based business process management: A case study. IEEE International Conference on Services Computing, 609–616.
28. Zeng, L., Lei, H., Chang, H., & Berbner, R. (2007). Business service management: Challenges and directions. Proceedings of the IEEE International Conference on Services Computing, 241–248.
29. Dijkman, R., Dumas, M., & Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. Information and Software Technology, 50(12), 1281–1294.
30. Fehling, C., Leymann, F., Retter, R., Schupeck, W., & Arbitter, P. (2014). Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer.
31. Dustdar, S., & Schreiner, W. (2005). A survey on web services composition. International Journal of Web and Grid Services, 1(1), 1–30.
32. Taylor, H., Yochem, A., Phillips, L., & Martinez, F. (2009). Event-Driven Architecture: How SOA Enables the Real-Time Enterprise. Addison-Wesley.

33. Lewis, G. A., Morris, E. J., & O'Brien, L. (2008). A Framework for Evaluating SOA Governance. Carnegie Mellon Software Engineering Institute.
34. Jamshidi, P., Pahl, C., & Mendonça, N. C. (2017). Pattern-based multi-cloud architecture migration. Software: Practice and Experience, 47(10), 1365–1389.
35. Tan, W. L., & Wang, Y. (2011). Measuring agility of business process reengineering in SOA-based e-commerce. Decision Support Systems, 51(3), 554–563.
36. Bhat, M., & Saha, D. (2005). SOA: The missing link between IT and business. Infosys Technologies White Paper.
37. Marks, E. A., & Bell, M. (2006). Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology. Wiley.

# ДОДАТКИ

# Додаток А

## XML Code

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:mulexml="http://www.mulesoft.org/schema/mule/xml"
xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
        xmlns:tracking="ttp://www.mulesoft.org/schema/mule/ee/tracking"xmlns:file="http://www.mulesoft.org
        /schema/mule/file" xmlns:dw="http://www.mulesoft.org/schema/mule/ee/dw"
        xmlns:ws="http://www.mulesoft.org/schema/mule/ws"
        xmlns:metadata="http://www.mulesoft.org/schema/mule/metadata"
        xmlns="http://www.mulesoft.org/schema/mule/core"
        xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
        xmlns:spring="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.mulesoft.org/schema/mule/smtp
http://www.mulesoft.org/schema/mule/smtp/current/mule-smtp.xsd
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-
current.xsd
http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/current/mule.xsd
http://www.mulesoft.org/schema/mule/ws http://www.mulesoft.org/schema/mule/ws/current/mule-ws.xsd
http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/current/mule-file.xsd
http://www.mulesoft.org/schema/mule/ee/dw http://www.mulesoft.org/schema/mule/ee/dw/current/dw.xsd
http://www.mulesoft.org/schema/mule/ee/tracking
http://www.mulesoft.org/schema/mule/ee/tracking/current/mule-tracking-ee.xsd
http://www.mulesoft.org/schema/mule/xml http://www.mulesoft.org/schema/mule/xml/current/mule-xml.xsd">
    <ws:consumer-config name="File_Read_WS" wsdlLocation="C:\Users\...\Info.wsdl" service="info"
port="infoPort" serviceAddress="http://....?" doc:name="Web Service Consumer"/>
    <file:connector name="File" autoDelete="false" streaming="true" validateConnections="true"
doc:name="File"/>
     <smtp:gmail-connector name="Gmail" validateConnections="true" doc:name="Gmail"/>
    <flow name="Middleware">
        <file:inbound-endpoint path="C:\Users\...." connector-ref="File" responseTimeout="10000"
doc:name="File" pollingFrequency="60000"/>
        <ws:consumer config-ref="File_Read_WS" operation="all5" doc:name="FileRead DBWriter WS"/>
        <dw:transform-message metadata:id="a7797fb7-57d0-4163-9e7a-7071074ab96d" doc:name="Transform
Message">
            <dw:set-payload><![CDATA[%dw 1.0
%output application/java
%namespace ns0 http://DB_Reader/
---
payload.ns0#all5Response.*return map ((return , indexOfReturn) -> {
        SSN: return.SSN,
        name: return.name,
        type: return.type,
        salary: return.salary,
        address: return.address,
        department: return.department
})]]></dw:set-payload>
            <dw:set-variable variableName="SSN"><![CDATA[%dw 1.0
%output application/java
%namespace ns0 http://DB_Reader/
---
payload.ns0#all5Response.return.SSN]]></dw:set-variable>
        </dw:transform-message>
        <choice doc:name="Choice">
            <when expression="#[flowVars.SSN != 0]">
                <foreach doc:name="For Each">
                    <scatter-gather doc:name="Splitter">
                        <processor-chain>
                            <parse-template location="C:\Users\....\Mail.txt" metadata:id="b7d894eb-465b-
47f7-a542-b49fc4fb53d9" doc:name="Email Template"/>
```

```
                        <smtp:outbound-endpoint host="smtp.gmail.com" port="587" user="...."
password="***" connector-ref="Gmail" to="...." from="...." subject="Service Desk New Employee"
responseTimeout="10000" doc:name="Service Desk email"/>
                        <logger message="#[payload]" level="INFO" doc:name="Logger"/>
                    </processor-chain>
                    <processor-chain>
                        <parse-template location="C:\Users\..\Mail2.txt" doc:name="Email Template 2"/>
                        <smtp:outbound-endpoint host="smtp.gmail.com" port="587" user="...."
password="***" connector-ref="Gmail" to="...." from="...." subject="Insurance Company New Employee"
responseTimeout="10000" doc:name="Insurance email"/>
                        <logger message="#[payload]" level="INFO" doc:name="Logger"/>
                    </processor-chain>
                    <processor-chain>
                        <parse-template location="C:\Users\..\Mail3.txt" doc:name="Email Template 3"/>
                        <smtp:outbound-endpoint host="smtp.gmail.com" port="587" user="...."
password="***" connector-ref="Gmail" to="...." from="...." subject="Flower Company New Employee"
responseTimeout="10000" doc:name="Flower email"/>
                        <logger level="INFO" doc:name="Logger" message="#[payload]"/>
                    </processor-chain>
                </scatter-gather>
                <set-payload value="Employees successfully added" doc:name="Set Payload"/>
            </foreach>
        </when>
        <otherwise>
            <set-payload value="No new employees" doc:name="Set Payload"/>
        </otherwise>
    </choice>
    <logger message="3: #[payload]" level="INFO" doc:name="Logger"/>
    <catch-exception-strategy doc:name="Catch Exception Strategy">
        <logger message="Failure: " level="ERROR" doc:name="Error Log"/>
        <parse-template location="C:\Users\....\Error.txt" doc:name="Error Template"/>
        <file:outbound-endpoint path="C:\Users\..."
outputPattern="#[server.dateTime.format(&quot;YYYY-MM-
dd&quot;)]_ERROR_#[message.inboundProperties.originalFileName]" responseTimeout="10000" doc:name="Error
File"/>
    <smtp:outbound-endpoint host="smtp.gmail.com" port="587" user="...." password="***" connector-
ref="Gmail" to="...." from="...." subject="Error notification" responseTimeout="10000" doc:name="Error
mail"/>
     </catch-exception-strategy>
    </flow>
</mule>
```

## Методи веб-сервісу

```
@WebService
public class EmployeeService {
    @WebMethod
    public String getEmployeeDetails(String employeeId) {
        // Implementation to get employee details
        return "Employee Details";
    }

    @WebMethod
    public void updateEmployeeDetails(String employeeId, String newDetails) {
        // Implementation to update employee details
    }
}
```

# MySQL Database trigger

```
DELIMITER &&

CREATE TRIGGER before_insert_employees
BEFORE INSERT
ON `employees`.`person`
FOR EACH ROW
BEGIN
    INSERT INTO `AD`.`person`(SSN, name, address)
    VALUES (NEW.SSN, NEW.name, NEW.address)
    ON DUPLICATE KEY UPDATE
    name = VALUES(name), address = VALUES(address);
END&&

DELIMITER ;
```

**БІБЛІОГРАФІЧНА ДОВІДКА**

**Тема дипломної роботи:** “ Імплементація SOA на рівні корпоративних сервісів  для підвищення гнучкості інтеграції ”

Обсяг пояснювальної записки: __77__ аркушів.

Дата закінчення роботи:  10  червня  2025 р.

Підпис студента _____