ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі процесно-орієнтованого підходу»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Максим МОГИЛЬНИК
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-61
Максим МОГИЛЬНИК

Керівник: Володимир САДОВЕНКО
канд. фіз.-мат. наук, доц.

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Могильнику Максиму Руслановичу

1. Тема кваліфікаційної роботи: «Удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі процесно-орієнтованого підходу»

керівник кваліфікаційної роботи Володимир САДОВЕНКО, кандидат фізико-математичних наук, доцент

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з архітектури ІС і процесно-орієнтованих підходів, законодавчі та регуляторні акти щодо політичних партій і захисту даних, опис організаційної структури партії, моделі бізнес-процесів (BPMN), функціональні та нефункціональні вимоги до інформаційної системи, вимоги з безпеки та політики доступу.

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

1. Дослідження підходів до побудови архітектури інформаційних систем для управління політичною партією.
2. Розробка методики побудови архітектури ІС з урахуванням функціональних/нефункціональних вимог.
3. Валідація та оцінка запропонованого методу: моделювання, критерії ефективності.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Мета, об’єкт та предмет дослідження.
   2. Порівняльна характеристика існуючих підходів.
   3. Удосконалення процесно-орієнтованого підходу — ключові зміни.
   4. Процес розробки пз за модифікованим методом.
   5. Архітектура запропонованої системи.
   6. Практичний результат.
   7. Повна порівняльна характеристика результату з запропонованими альтернативами.
   8. Висновки.
   9. Публікації та апробація роботи.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-05.11.2025	
2	Вивчення матеріалів для аналізу існуючих підходів до розробки архітектури інформаційних систем	06.11-11.11.2025	
3	Дослідження підходів до побудови архітектури інформаційних систем у сфері управління політичною партією	12.11-19.11.2025	
4	Аналіз процесно-орієнтованих та модульних архітектурних принципів у проєктуванні інформаційних систем	20.11-25.11.2025	
5	Розробка удосконаленого (гібридного) процесно-орієнтованого підходу до побудови архітектури ІС	26.11-02.12.2025	
6	Застосування запропонованого підходу та розробка прототипу архітектури інформаційної системи	03.12-14.12.2025	

7	Оформлення роботи: вступ, висновки, реферат	15.12-19.12.2025	
8	Розробка демонстраційних матеріалів	19.12-24.12.2025	

Здобувач вищої освіти _____ Максим МОГИЛЬНИК
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 70 стор., 4 табл., 7 рис., 25 джерел.

Мета роботи – удосконалення методу процесу побудови архітектури інформаційної системи для сфери управління політичною партією за рахунок застосування удосконаленого процесно-орієнтованого підходу та модульних архітектурних принципів.

Об'єкт дослідження – процеси управління діяльністю політичної партії.

Предмет дослідження – методи, принципи та технології побудови архітектури інформаційних систем, орієнтовані на автоматизацію процесів управління політичною партією, за рахунок використання процесно-орієнтованого підходу та модульних архітектурних принципів.

Розроблено і запропоновано модифікований процесно-орієнтований (гібридний) підхід, який поєднує орієнтацію на дані, урахування організаційної структури та відкладену інтеграцію окремих модулів. На основі такого підходу було реалізовано прототип (Directus + GraphQL, Node.js-сервіси, MySQL/MongoDB, Redis, S3, контейнеризація, CI/CD), що підтвердив практичну здійсненність концепції.

КЛЮЧОВІ СЛОВА: ПРОЦЕСНО-ОРІЄНТОВАНИЙ ПІДХІД, ГІБРИДНИЙ ПІДХІД, МОДУЛЬНА АРХІТЕКТУРА, ІНФОРМАЦІЙНА СИСТЕМА, УПРАВЛІННЯ ПОЛІТИЧНОЮ ПАРТІЄЮ, АВТОМАТИЗАЦІЯ БІЗНЕС-ПРОЦЕСІВ, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ІНТЕГРАЦІЯ МОДУЛІВ, DIRECTUS, GRAPHQL, NODE.JS, MYSQL/MONGODB, КОНТЕЙНЕРИЗАЦІЯ, CI/CD, БЕЗПЕКА ДАНИХ.

ABSTRACT

Text part of the master's qualification work: 70 pages, 7 pictures, 4 tables, 25 sources.

The purpose of the work – to improve the method for constructing the architecture of an information system for the domain of political party management by applying an enhanced process-oriented approach and modular architectural principles.

Object of research – the processes of managing the activities of a political party.

Subject of research – methods, principles, and technologies for designing the architecture of information systems aimed at automating political party management processes through the use of a process-oriented approach and modular architectural principles.

Summary of the work: a modified process-oriented (hybrid) approach has been developed and proposed, combining data-driven principles, consideration of the organizational structure, and deferred integration of individual modules. Based on this approach, a prototype was implemented (Directus + GraphQL, Node.js services, MySQL/MongoDB, Redis, S3, containerization, CI/CD), which confirmed the practical feasibility of the concept.

KEYWORDS: PROCESS-ORIENTED APPROACH, HYBRID APPROACH, MODULAR ARCHITECTURE, INFORMATION SYSTEM, POLITICAL PARTY MANAGEMENT, BUSINESS PROCESS AUTOMATION, SOFTWARE ARCHITECTURE, MODULE INTEGRATION, DIRECTUS, GRAPHQL, NODE.JS, MYSQL/MONGODB, CONTAINERIZATION, CI/CD, DATA SECURITY.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач МОГИЛЬНИК М.Р. до захисту кваліфікаційної роботи

за спеціальністю 121 Інженерія програмного забезпечення

освітньо-професійної програми «Інженерія програмного забезпечення»

на тему: «Удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі процесно-орієнтованого підходу».

Кваліфікаційна робота і рецензія додаються.

Директор ННІ ІТ _____ Катерина НЕСТЕРЕНКО
(підпис)

Висновок керівника кваліфікаційної роботи

Здобувач Могильник Максим Русланович під час написання магістерської роботи проявив здатність самостійно опрацьовувати технічну та наукову літературу, продемонстрував високий рівень теоретичної підготовки та гарні інженерні навички. За формою і змістом магістерська робота відповідає чинним вимогам і є самостійним дослідженням. Під час виконання кваліфікаційної роботи здобувач проявив здатність проводити ґрунтовний аналіз задачі, синтезувати алгоритм її вирішення відповідно до наявних обмежень та специфіки предметної галузі, використовувати методи та засоби інженерії програмного забезпечення. Експериментальні дослідження підтверджують працездатність розробленого методу побудови архітектури інформаційної системи та його переваги у порівнянні з існуючими підходами.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Могильника М.Р. на оцінку «відмінно» та присвоїти йому кваліфікацію магістр з інженерії програмного забезпечення.

Керівник кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

«____» _____ 20__ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Могильник М.Р. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедри ІПЗ _____ Ірина ЗАМРІЙ
(підпис)

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну магістерську роботу

здобувача вищої освіти Могильника Максима Руслановича

на тему «Удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі процесно-орієнтованого підходу»

Актуальність.
Цифровізація підвищує ефективність і прозорість управління партією.
Зростання обсягу даних, ускладнення процесів та потреба в оперативному прийнятті рішень спричинили необхідність гнучких і модульних процесно-орієнтованих підходів.

Позитивні сторони.
1. Автор чітко сформулював мету та поставлені завдання, робота має виражену практичну спрямованість і обґрунтовану актуальність проблеми цифровізації управління організаціями.
2. Розроблено та реалізовано прототип системи на сучасному технологічному стеку (Directus, GraphQL, Node.js, СУБД, Docker тощо), що підтверджує практичну здійсненність запропонованого підходу.
3. Представлено продуману методику валідації з експериментальними дослідами та аргументованими висновками й рекомендаціями щодо впровадження.

Недоліки.
1. Інтеграційні сценарії з зовнішніми сервісами змодельовані частково — недостатньо протестовано поведінку системи при складних відмовах зовнішніх компонентів
2. Робота містить певну кількість стилістичних, синтаксичних та пунктуаційних помилок.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної магістерської роботи.

Висновок: *кваліфікаційна робота на здобуття ступеня магістра заслуговує оцінку "відмінно", а здобувач* ***Могильник Максим Русланович*** *заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення.*

Рецензент:
науковий ступінь, вчене звання _____ *підпис* _____ *Ім'я, ПРІЗВИЩЕ*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС — інформаційна система

ПЗ — програмне забезпечення

BPMN (Business Process Model and Notation) — нотація моделювання бізнес-процесів

API (Application Programming Interface) — програмний інтерфейс застосунку

REST (Representational State Transfer) — архітектурний стиль веб-сервісів

GraphQL — мова запитів і середовище виконання запитів до API

RBAC (Role-Based Access Control) — рольова модель керування доступом

CI/CD (Continuous Integration / Continuous Delivery) — безперервна інтеграція та доставка

S3 (Simple Storage Service) — об’єктне сховище даних, сумісне з протоколом Amazon S3

RDBMS (Relational DataBase Management System) — система керування реляційними базами даних

NoSQL — клас систем керування базами даних, що не використовують класичну реляційну модель

MySQL — реляційна система керування базами даних MySQL

MongoDB — документоорієнтована система керування базами даних MongoDB

Redis — система in-memory зберігання даних, що використовується як кеш або брокер повідомлень

Docker — платформа контейнеризації застосунків

Directus — headless CMS-платформа для керування даними та метаданими

KPI (Key Performance Indicator) — ключовий показник ефективності

OTP (One-Time Password) — одноразовий пароль

ВСТУП

Цифровізація підвищує ефективність і прозорість управління партією; водночас зростання обсягу даних, ускладнення процесів та вимоги до оперативного прийняття рішень створюють потребу в гнучких, модульних і процесно-орієнтованих підходах до проєктування інформаційних систем.

Мета дослідження: удосконалення методу процесу побудови архітектури інформаційної системи для сфери управління політичною партією за рахунок застосування модифікованого процесно-орієнтованого (гібридного) підходу та модульних архітектурних принципів.

Завдання дослідження:

1. Проаналізувати сучасні підходи та практики побудови архітектури інформаційних систем із фокусом на управлінські процеси організацій.
2. Дослідити особливості процесно-орієнтованого моделювання (BPMN) і його застосування у контексті політичних організацій.
3. Розробити модифікований (гібридний) процесно-орієнтований підхід, що поєднує орієнтацію на дані, урахування організаційної структури та відкладену інтеграцію модулів.
4. Реалізувати прототип архітектури (Directus + GraphQL, Node.js-сервіси, MySQL/MongoDB, Redis, S3, контейнеризація, CI/CD) та провести його тестування.
5. Оцінити ефективність запропонованого підходу за допомогою критеріїв продуктивності, масштабованості й безпеки та надати практичні рекомендації щодо впровадження.

Об'єкт дослідження: процеси управління діяльністю політичної партії.

Предмет дослідження: методи, принципи та технології побудови архітектури інформаційних систем, орієнтовані на автоматизацію процесів управління політичною партією з використанням процесно-орієнтованого та модульного підходів.

Методи дослідження: аналіз наукової і нормативної літератури; системний і структурно-функціональний аналіз бізнес-процесів; моделювання процесів (BPMN); архітектурне моделювання та проєктування; прототипування і експериментальне тестування; порівняльний аналіз та оцінка за KPI; методи забезпечення інформаційної безпеки та ризик-аналіз.

Джерела дослідження: наукові монографії та статті з архітектури ПЗ і BPM, законодавчі акти та нормативні документи щодо діяльності політичних партій і захисту даних, технічна документація і специфікації використовуваних технологій (Directus, GraphQL, Node.js, СУБД), внутрішні документи та описи бізнес-процесів партій, матеріали конференцій і звіти з впроваджень інформаційних систем.

Наукова новизна:

1. Запропоновано гібридний процесно-орієнтований підхід до побудови архітектури ІС, що інтегрує орієнтацію на дані, урахування організаційної структури та відкладену інтеграцію модулів.
2. Удосконалено підхід до аналізу внутрішніх процесів політичної партії шляхом формалізації критеріїв оцінювання ефективності управлінських дій.
3. Розроблено покращений метод підтримки прийняття рішень, який поєднує структуровану модель процесів та алгоритм вибору оптимальних управлінських дій.

Практичне значення результатів: розроблені методичні положення та шаблони процесно-орієнтованого моделювання разом із формалізованими критеріями оцінки ефективності дозволяють автоматизувати та моніторити управлінські процеси політичної партії; запропонований метод підтримки прийняття рішень і алгоритм відбору оптимальних дій можуть бути інтегровані в інформаційну систему як модуль аналітики, а реалізований прототип підтверджує практичну здійсненність підходу й може слугувати основою для пілотного впровадження.

Апробація результатів і публікації: основні положення та результати дослідження доповідалися на внутрішніх семінарах і демонстраціях прототипу; за темою підготовлено науково-практичні матеріали для оприлюднення на конференціях та у фахових виданнях.

1 ОГЛЯД НАУКОВИХ ДЖЕРЕЛ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ПОБУДОВИ АРХІТЕКТУРИ ІНФОРМАЦІЙНИХ СИСТЕМ У СФЕРІ УПРАВЛІННЯ ПОЛІТИЧНИМИ ПАРТІЯМИ

1.1. Концептуальні підходи до побудови архітектури ІС: орієнтований на дані, процесно-орієнтований, орієнтований на організаційну структуру та підхід з відкладеною інтеграцією

1.1.1. Орієнтований на дані підхід (Data-oriented)

Орієнтований на дані підхід трактує дані як базовий конструктив архітектури інформаційної системи: сутності предметної області, їхні атрибути, взаємозв’язки й життєві цикли визначають структуру компонентів, інтерфейсів і правил взаємодії. У результаті систематичного огляду наукових праць, технічних звітів і прикладних кейсів у сфері моделювання даних та інженерії даних встановлено, що фундаментальні положення цього підходу спираються на реляційну теорію даних, методи семантичного моделювання та практики управління основними даними. Ключовими поняттями є «єдине джерело істини», канонічна модель даних, відстеження походження даних і реєстрація схем; ці елементи визначають не лише структуру збереження, а й процедури відповідальності за якість і походження інформації.

Архітектурні складові типового рішення з орієнтацією на дані включають транзакційний шар для зберігання оперативних записів, аналітичний шар для формування звітів і складних запитів, сервісний шар, що експонує семантично узгоджені інтерфейси згідно з визначеними контрактами даних, та служби підтримки якості даних — валідатори, служби дедуплікації та інструменти профілювання. На практиці це означає наявність реєстру схем, формалізованих описів ресурсів і шаблонів інтерфейсів, процедур версіонування схем і навколо них організованих процесів затвердження змін.

Процедурна реалізація підходу передбачає кілька послідовних етапів: збір і аналіз нормативних і предметних джерел (регламенти партії, фінансові звіти, формати зовнішньої звітності), побудова предметної моделі у вигляді діаграм сутностей і словника термінів, визначення набору первинних сутностей і їхніх контрактів даних (формат полів, обов'язковість, правила валідації), вибір підходу зберігання (реляційна система для критично консистентних сутностей, документо-орієнтована — для гнучких записів), налаштування процесів витягання, перетворення і завантаження в аналітичні шари, а також впровадження процедур міграції схем та автоматизованого тестування сумісності змін у процесі розгортання.

Проведений огляд і порівняння стратегій еволюції схем виявив наступні практичні висновки. Жорсткі одноетапні міграції можуть гарантувати цілісність, але значно ускладнюють виробничі релізи та підвищують ризик простоїв; натомість підхід «контракт перед реалізацією» із забезпеченням сумісності назад і вперед та поступовими міграціями дозволяє зменшити втручання у стабільні сервіси, але вимагає суворої дисципліни тестування контрактів і налаштування процесів автоматичної перевірки сумісності під час збірки програмного забезпечення. Для сутностей з багатою історією змін корисними є практики збереження версійних записів і журналів походження даних, що дають змогу відтворити стан даних на будь-яку історичну дату для цілей аудиту.

Окрему увагу в досліджених джерелах отримали питання якості даних і управління ними. Без механізмів профілювання, валідації, виявлення аномалій і розподілу відповідальності за дані (ролей «власника даних») ефективність орієнтованих на дані рішень істотно знижується: зростає кількість дублікатів, помилок у звітності й час, необхідний на очистку та корекцію даних. У контексті політичної партії це прямо впливає на достовірність фінансових звітів, облік членства та можливість коректного аналізу ефективності кампаній. Отже, вимога до технічних рішень супроводжується необхідністю організаційних заходів: визначення ролей і угод про рівень обслуговування щодо якості даних, регулярні ревізії та процедури затвердження змін у моделях даних.

Для верифікації припущень і перевірки практичності орієнтованого на дані підходу в роботі запропоновано та апробовано набір експериментальних процедур: аналіз якості наявних даних, імітація еволюції схем із оцінкою впливу на сервіси, тестування контрактів між сервісами і шаром даних, а також навантажувальні випробування операцій з оновлення даних і виконання процедур витягнення та перетворення даних. Результати таких експериментів підтверджують необхідність автоматичного тестування сумісності схем у процесі безперервної інтеграції та розгортання і показують компроміс між жорсткістю схем і оперативною гнучкістю бізнесу під час кампаній.

У рамках аналізу визначено ключові показники ефективності для шару даних: частка валідних записів, затримка операцій читання й запису, середній час виконання процедур витягання і перетворення даних, кількість інцидентів якості даних на одиницю часу та середній час виявлення й усунення помилок у даних. Моніторинг цих метрик дозволяє приймати обґрунтовані рішення щодо ступеня нормалізації, пріоритетів очищення даних і інструментів автоматизації процесів управління даними.

Узагальнення огляду показує: орієнтований на дані підхід є необхідною складовою архітектури інформаційної системи для управління політичною партією у випадках, де критичною є цілісність, відтворюваність і відповідність нормативним вимогам (реєстри членства, фінансова звітність, історичні журнали). Водночас сам по собі цей підхід не замінює потреби в формалізованих процесах управління та контролю ролей; оптимальне рішення досягається шляхом інтеграції шару даних із процесним і організаційним шарами та впровадженням механізмів управління схемами, якості і безпеки даних.

1.1.2. Процесно-орієнтований підхід (Process-oriented)

Процесно-орієнтований підхід розглядається в сучасній науковій та прикладній літературі як методологія проєктування інформаційних систем, яка концентрує увагу на формалізації, автоматизації та контролі життєвого циклу бізнес-процесів як основних одиниць організаційної діяльності. За результатами

систематичного аналізу профільних наукових публікацій, технічних звітів і галузевих практик, уявлення про процес як про агрегат функціональних кроків, правил прийняття рішень та інформаційних взаємодій виступає ключовим для побудови керованих, відтворюваних і вимірюваних інформаційних середовищ. В літературі підкреслюється, що процесно-орієнтований підхід не є лише моделлю документування; він передбачає перетворення формалізованих процесних описів у виконувані артефакти, інструментальну підтримку їх виконання та інструменти для постійного моніторингу й оптимізації.

Теоретична база підходу ґрунтується на кількох взаємопов’язаних положеннях, які повторно фіксуються в робіт різних авторів. По-перше, процес визначається як послідовність цільових дій із зазначенням ролей, ресурсів, умов переходу та очікуваних результатів, причому така формалізація повинна містити як синтаксичні, так і семантичні компоненти: структуру потоків, правила трансформації даних і смислові контракти між учасниками. По-друге, виконувана модель процесу повинна бути репрезентована таким чином, щоб її можна було перевірити через набори тестів і симуляцій до розгортання, що зменшує ризик помилкових інтерпретацій вимог і забезпечує якість поведінки системи під реальними навантаженнями. По-третє, архітектурні рішення мають гарантувати консистентність даних і відтворюваність станів екземплярів процесів у розподіленому середовищі, включно з механізмами відновлення після збоїв і механізмами аудиту.

Методологічно процесно-орієнтований підхід вимагає багаторовекторної роботи: формалізації логіки процесу, визначення інформаційних контрактів, проєктування інтерфейсів взаємодії та забезпечення нефункціональних властивостей. Формалізація передбачає опис вхідних і вихідних артефактів кожного кроку, визначення правил маршрутизації і критеріїв завершення, а також визначення сценаріїв обробки винятків і компенсаційних дій. Інформаційні контракти — це набір технічних вимог до структур даних і поведінки сервісів, які використовуються в процесі; їх документування і підтримка у вигляді автоматизованих перевірок (контрактного тестування) є

одним із центральних елементів практик, рекомендованих у джерелах. Нефункціональні вимоги (продуктивність, доступність, безпека, вимоги до збереження й анонімізації даних) повинні інтегруватися у процесні моделі як явні обмеження, що дозволяє на етапі проєктування прогнозувати поведінку системи і планувати механізми забезпечення відповідності нормативним нормам.

Перехід від моделі до технічної реалізації в літературі описується як процес, що включає декомпозицію процесу на виконувані компоненти, визначення меж транзакцій і опис механізмів взаємодії. Виконувані компоненти мають мати формалізовані інтерфейси, чітко описані залежності та механізми обробки помилок; особливу увагу автори приділяють питанням збереження стану екземплярів процесу, каталогізації подій виконання та забезпеченням аудиту. Багато праць наголошують на необхідності версіонування процесних моделей: зміни в логіці не повинні порушувати вже запущені екземпляри, а механізми одночасної підтримки декількох версій повинні бути частиною операційної політики.

Обробка помилок і виняткових ситуацій у процесно-орієнтованих системах описується в літературі як критичний аспект, що прямо впливає на стійкість і правову захищеність організації. Вчені і практики пропонують інтегрувати у моделі чітко визначені сценарії відновлення, ролі відповідальності, логування причин відмов і процедури ескалації. Технічні механізми повинні забезпечувати детальне відстеження причин збоїв, можливість відновлення контексту виконання та коректну обробку частково виконаних транзакцій. Ряд досліджень приділяє значну увагу тестуванню сценаріїв відмов у лабораторних умовах та перевірці механізмів компенсації під навантаженням як методам підвищення надійності платформи.

Аналітичні виміри процесно-орієнтованих рішень у наукових роботах розглядаються з акцентом на перетворення даних телеметрії виконання в інформацію для прийняття рішень. Авторами описуються підходи до вибору та нормалізації метрик, механізми кореляції подій і використання статистичних

методів для виявлення закономірностей і вузьких місць. Велику увагу приділяють методам побудови процедур звітування, визначенню перцентильних показників (для адекватної оцінки користувацького досвіду) і застосуванню аналітики для підвищення ступеня автоматизації та скорочення часу обробки. У роботах також зазначається потенціал використання методів машинного навчання для прогнозування навантажень і виявлення аномалій, однак дослідники підкреслюють необхідність строгого контролю якості даних і ретельного опрацювання аспектів приватності при застосуванні таких методів.

Питання безпеки і захисту персональних даних у процесно-орієнтованих системах займають значне місце в літературі. По-перше, процесні моделі повинні явно враховувати політики доступу та збереження конфіденційності; по-друге, архітектурні рішення мають підтримувати шифрування, розмежування зон доступу і механізми аудиту, що забезпечують відтворюваність подій. Нормативні вимоги вимагають можливості реалізації процедури видалення або анонімізації даних за запитами суб’єктів і встановлення меж зберігання, що повинно бути відображено в процесних сценаріях і технічних контрактах.

Організаційні практики впровадження, узагальнені у наукових джерелах, підкреслюють роль процесного говернансу: каталогу процесів, визначення відповідальних осіб, процедур контролю якості моделей і політик версіонування. Дослідження вказують на те, що успіх застосування процесно-орієнтованих рішень значною мірою залежить від узгодженості між бізнес-цілями, формалізованими процесними описами та технічною реалізацією, а також від наявності інституційних механізмів підтримки змін.

У підсумку систематичний огляд наукової і технічної літератури свідчить, що процесно-орієнтований підхід є комплексною методологією, що поєднує формалізацію поведінки, інструментальну реалізацію виконуваних моделей та аналітичну підтримку на основі телеметрії. Літературні джерела узгоджуються у важливості формалізації контрактів, забезпечення відтворюваності станів, проєктування механізмів обробки винятків і впровадження політик говернансу як ключових умов ефективного застосування підходу в реальному середовищі.

1.1.3. Орієнтований на організаційну структуру підхід (Organizational-driven)

Орієнтований на організаційну структуру підхід у проєктуванні архітектури інформаційних систем розглядає організаційну структуру організації як первинний аналітичний і проєктний артефакт. У його основі лежить припущення, що бізнес-функції, ролі, повноваження і відповідальності працівників або підрозділів формують вимоги до інформаційної системи у більш прямих і стійких координатах, ніж окремі процеси чи поодинокі набір даних. Такий підхід вимагає формалізованого відтворення організаційної ієрархії, принципів розподілу повноважень, правил делегування та порядків взаємодії між підрозділами, і надалі відображення цих елементів у структурі компонентів, механізмах доступу і схемах інтеграції інформаційної системи.

З методологічної точки зору підхід передбачає декілька взаємопов’язаних кроків: формалізацію організаційних одиниць і ролей у вигляді машинозчитуваних моделей, визначення кордонів відповідальності та операційних доменів, встановлення політик доступу й процедур делегування, а також проєктування інтерфейсів і контрактів між компонентами, орієнтованих на організаційну семантику. Формалізація ролей і повноважень повинна виходити за межі простого переліку посад — вона має містити опис операційних прав, умов їх надання та умов припинення, критерії ескалації й механізми контролю відповідності (compliance). У технічному відображенні це проявляється у впровадженні моделей авторизації (наприклад RBAC або ABAC), у розмежуванні зон відповідальності між сервісами та у проєктуванні схем логування і аудиту, орієнтованих на організаційні атрибути.

Архітектурна реалізація організаційно-орієнтованого підходу зазвичай включає відображення структури підрозділів і ролей на модулі інформаційної системи: кожний модуль або служба може бути «прив’язана» до конкретного організаційного домену, мати власні політики доступу, SLA і правила обробки даних. Така прив’язка забезпечує природну ізоляцію відповідальності і спрощує

управління життєвим циклом компонентів, оскільки зміни в організаційній структурі можуть бути безпосередньо відображені у конфігурації системи. Водночас важливо зберегти механізми інтероперабельності й узгоджені data - API-контракти, щоб організаційне розмежування не породжувало фрагментації даних та дублювання функціональності.

У контексті управління ризиками і відповідністю підхід надає очевидні переваги: зрозуміла трасуваність рішень до ролей і підрозділів полегшує аудит, встановлення відповідальності і доведення коректності виконання процедур у випадках перевірок. Логування операцій та їх прив’язка до організаційних атрибутів дозволяє будувати репорти, що відповідають вимогам регуляторів і внутрішнього контролю. Разом з тим, літературні джерела та практичні звіти підкреслюють необхідність чіткої політики версіонування організаційних моделей і процедур зміни, оскільки неконтрольовані зміни структури або ролей можуть суттєво впливати на коректність роботи системи й порушувати очікувану поведінку автоматизованих компонентів.

Організаційно-орієнтований підхід має специфічні вимоги до синхронізації з іншими архітектурними парадигмами. По-перше, йому необхідна тісна інтеграція з підходом, орієнтованим на дані: рольова семантика повинна бути відображена у політиках доступу до сутностей даних, у метаданих і в каталозі власників даних. По-друге, слід враховувати процесний вимір — процеси, хоч і вторинні в цій парадигмі, повинні носити організаційно-детермінований характер: маршрутизація задач, процедури ескалації та контрольні точки виконання формуються з урахуванням ролей та підрозділів. Практичні реалізації демонструють доцільність створення єдиного реєстру ролей та сервісів, який одночасно служить джерелом істини для механізмів авторизації, інтерфейсів адмін-налаштування і системи моніторингу.

Окрему увагу в реалізації приділяють аспектам управління змінами і говернансу. Запровадження організаційно-орієнтованої архітектури потребує процедур схвалення змін структури, формалізованих впливних аналізів (impact analysis), а також тестових сценаріїв, що моделюють наслідки реорганізації для

доступів, потоків даних і бізнес-логіки. Інституційні механізми — каталогізація ролей, регламентні процедури для зміни повноважень і регулярні аудити — забезпечують стійкість системи до організаційних трансформацій.

Аналітика ефективності при такому підході фокусується на показниках, що пов’язують організаційні ресурси зі станом інформаційної системи: час ухвалення рішень за ролями, завантаженість ролей і підрозділів, частота ескалацій, кількість порушень політик доступу та рівень відповідності SLA. Ці метрики дозволяють здійснювати управлінський контроль і оптимізувати розподіл повноважень та ресурсу безпосередньо через інформаційну систему.

Незважаючи на переваги, підхід має й відомі обмеження: сильна орієнтація на поточну організаційну структуру може призводити до технологічної інерції і зменшувати гнучкість у швидкозмінних середовищах; централізація повноважень може створювати вузькі місця й ризики перенавантаження ключових ролей; надмірна деталізація ролей і політик ускладнює супровід та тестування системи. Тому ефективні впровадження поєднують чітке формалізоване моделювання організаційних компонентів з механізмами делегування, автоматизованого управління конфігурацією та регулярного перегляду політик.

1.1.4. Підхід з відкладеною інтеграцією (Deferred / Event-driven Integration)

Підхід з відкладеною інтеграцією розглядається як проєктна стратегія, за якої розробка підсистем і модулів виконується автономно, а їх інтеграція у єдину систему навмисно переноситься на пізніші етапи життєвого циклу проєкту. У фокусі цього підходу лежить розділення відповідальностей між командами, максимальна локалізація змін в межах модуля та мінімізація потреби у ранній синхронізації між командами. Така парадигма дозволяє прискорити паралельну розробку, знизити затримки пов’язані з міжкомандними узгодженнями і фокусувати ресурси на локальних вимогах, але водночас породжує характерні

виклики щодо сумісності інтерфейсів, узгодженості семантики даних та управління інтеграційним боргом.

Методологічно підхід опирається на ідеї чіткої контрактної специфікації інтерфейсів і на практики «контракт-першого» проєктування: ще до початку масової розробки кожен модуль має мати визначений набір контрактів (інтерфейсів, форматів повідомлень, поведінкових очікувань), які стають орієнтирами для незалежних команд. При цьому контракт трактують не лише як технічну схему повідомлень, а як формалізований набір припущень стосовно семантики, інваріантів і обмежень, які повинні зберігатися під час еволюції модулів. Важливим елементом практики є застосування підходів типу consumer-driven contracts, що дозволяють споживачам визначати очікування від постачальників сервісів і таким чином мінімізувати ризики сумісності на момент інтеграції.

Технічні наслідки вибору deferred integration охоплюють кілька взаємопов’язаних аспектів. По-перше, у процесі розробки широко застосовуються заглушки (stubs), моки і симулятори, що імітують поведінку ще не реалізованих або тимчасово недоступних сервісів; ці техніки дозволяють проводити локальне тестування і валідацію логіки модуля без потреби у ранньому з’єднанні з іншими компонентами. По-друге, для зменшення ризиків інтеграції необхідно інвестувати в автоматизовані механізми контрактного тестування та у CI/CD пайплайни, які можуть виконувати регресійні перевірки сумісності контрактів на етапах інтеграції. По-третє, архітектура системи повинна передбачати інструменти для версіонування інтерфейсів і політики міграції, щоб зміни в одному модулі не призводили до лавиноподібних збоїв під час фінальної інтеграції.

Контроль якості при відкладеній інтеграції набуває особливого значення: оскільки реальні взаємодії між модулями перевіряються пізніше, необхідно приділяти підвищену увагу верифікації специфікацій і превентивному тестуванню. До практик, що знижують інтеграційні ризики, належать автоматизовані тести контрактів, інтеграційні стенди з відтворенням кінцевих

сценаріїв, симуляції навантаження з моделлю асинхронних затримок та регулярні інтеграційні вікна, під час яких команди об’єднують свої артефакти і виконують перевірки сумісності. Політика «пізньої інтеграції» повинна також включати чіткі правила для виявлення й пріоритизації інтеграційних дефектів, механізми їх трасування до відповідальних команд і процедури для швидкого виправлення.

У питаннях управління проєктом підхід з відкладеною інтеграцією порождає явище інтеграційного боргу — накопичення невирішених невідповідностей у контрактах, розбіжностей у форматах даних та неузгоджених припущень. Цей борг потребує спеціальних процесів контролю: планування інтеграційних ітерацій, підтримка окремих «integration sprints» у релізних циклах, а також наявність ролей, відповідальних за координацію інтеграції й проведення impact-аналізу під час змін. Без системного управління інтеграційним боргом ризик зростання вартості виправлень при фінальній інтеграції суттєво збільшується, що підриває вигоди від автономної розробки.

Архітектурні і нефункціональні питання при deferred integration також мають власні вимоги. Потрібні чіткі політики версіонування інтерфейсів та механізми backward/forward-совместимості; необхідна інфраструктура для централізованого реєстру контрактів, сховищ тестових даних та інтеграційних стендів; слід передбачити спостережуваність і трасування для сценаріїв, що будуть протестовані пізніше. Крім того, варто вбудовувати в дизайн механізми відновлення та компенсації, оскільки помилки, виявлені під час пізньої інтеграції, часто потребують складних коригувальних дій у декількох модулях одночасно.

З точки зору ризиків, основні загрози пов’язані з виявленням критичних несумісностей у пізній фазі, коли виправлення вимагають значних коштів і змін у суміжних підсистемах; зростанням складності тестових середовищ і потенційною втратою контексту розробки через паралельність і довгий інтервал між реалізацією і інтеграційними перевірками; а також з можливим накопиченням технічного боргу через відсутність своєчасного узгодження

стандартів. Водночас переваги підходу полягають у прискоренні локальної розробки, гнучкості щодо внутрішньої еволюції модулів і можливості паралельного залучення різнорідних команд і технологічних стеків.

Практики пом’якшення ризиків, які фіксує сучасна профільна література і досвід індустрії, включають: раннє погодження контрактів із застосуванням формальних специфікацій; регулярні інтеграційні ітерації з автоматизованим виконанням тестів сумісності; використання consumer-driven contract testing; централізований registry контрактів і тестових сценаріїв; побудовystaging- та pre-production оточень, що імітують продакшн; застосування feature flags та progressive delivery для поетапного включення інтегрованої функціональності; і наявність окремих індикаторів якості інтеграції (кількість інтеграційних дефектів, середній час виявлення і виправлення, частка успішних інтеграційних ітерацій).

В аспекті організаційного говернансу підхід вимагає інституалізації ролей і процедур: відповідальні за інтеграцію архітектори, власники контрактів, команда інтеграційного тестування і механізми управління змінами. Без такої інституційної підтримки автономність розробки легко перетворюється на фрагментацію, що значно ускладнює подальшу інтеграцію та експлуатацію системи.

Отже, підхід з відкладеною інтеграцією є практично ефективним інструментом для масштабних, багатокомандних проєктів, які прагнуть паралелізувати розробку і зменшити ранні узгоджувальні витрати; він вимагає системного підходу до контрактного проєктування, автоматизованого тестування сумісності, чітких процедур управління інтеграційним боргом і відповідної інституалізації говернансу для забезпечення контрольованої і передбачуваної фінальної інтеграції.

1.2. Технології та платформи реалізації архітектурних рішень

Інфраструктурна підкладка та сховища. Для довготривалого зберігання бінарних артефактів і медіа-предметів доречно розглядати S3-сумісні рішення (MinIO, Ceph) або керовані сервіси провайдерів. Для файлових та блочних операцій — розподілені файлові системи і мережеві сховища з можливістю реплікації й політиками життєвого циклу. При виборі звертають увагу на гарантії цілісності й відновлення даних, можливості версіонування об’єктів, вбудовані механізми резервного копіювання (наприклад, підтримка інструментів типу Restic або BorgBackup) та оперативні характеристики при одночасному доступі великої кількості клієнтів.

Середовище виконання і оркестрація служб. Для ізоляції середовищ і відтворюваності розгортань застосовують контейнери як механізм упаковки сервісів; їхня оркестрація забезпечує самовідновлення, масштабування й управління конфігурацією. Практичні рішення: платформи оркестрації контейнерів (Kubernetes) для середовищ з високими вимогами до доступності та масштабованості; легші рішення або хостингові сервіси — для малих проєктів або прототипів. При використанні оркестрації необхідно проєктувати політики оновлення без зупинки, межі ресурсів і механізми відстеження здоров’я служб.

Системи управління сутностями й метаданими. Для оперативного управління схемами доменної області і політиками доступу розглядають системи керування контентом або репозитарії метаданих. Як приклад реалізації — Directus: забезпечує візуальне формування схем, управління правами доступу та оперативний адміністративний інтерфейс. Альтернативні підходи — власні рішення на основі легкого інтерфейсу керування або інші системи керування контентом, вибір яких слід мотивувати потребою у швидкому налаштуванні метаданих та інтеграції з шаром запитів.

Шар уніфікованого доступу до даних. Для агрегування та уніфікації запитів до різнорідних джерел даних застосовують логічні шари API. Серед

варіантів — GraphQL для гнучкого формування клієнтських запитів і агрегування даних із різних підсистем; традиційні REST-інтерфейси або надбудови для специфічних випадків. При виборі враховується складність запитів, потреба в агрегації на боці сервера, механізми кешування відповідей і можливість впровадження контрактного тестування.

СУБД та аналітичні сховища. Для транзакційних операцій і забезпечення цілісності даних застосовують реляційні системи управління базами даних (приклади: PostgreSQL, MySQL). Для гнучких напівструктурованих записів підходять документоорієнтовані сховища (наприклад, MongoDB). Для аналітичних потреб і масових зчитувань доцільні колонкові або спеціалізовані сховища (ClickHouse, сховища даних типу «data warehouse»). У проєктній архітектурі слід передбачити окремий реплікаційний або екстракційний процес, що готує дані у форматах, придатних для відтворюваних звітів і досліджень, а також реєстр метаданих і канонічні схеми для семантичної узгодженості.

Оперативне кешування і координація стану. Для зниження затримок і зменшення навантаження на первинні сховища застосовують оперативні кеші (Redis, Memcached) з продуманими політиками інвалідизації. Redis також служить для реалізації тимчасових блокувань, лічильників і механізмів лімітування. При проєктуванні кешування необхідно врахувати питання узгодженості даних, час життя записів і можливі стратегії скидання кешу при оновленнях.

Організація обміну та обробки подій. Для асинхронної взаємодії компонентів і обробки подій розглядають чергові системи і стрімінгові платформи. Приклади: RabbitMQ для чергової обробки повідомлень у класичних сценаріях, Apache Kafka для потокової передачі подій і побудови стійкої архітектури подій. Вибір залежить від вимог до гарантованої доставки, порядку повідомлень і загальної пропускної здатності.

Автоматизація життєвого циклу розробки та розгортання. Інструменти для безперервної інтеграції та безперервного розгортання (Jenkins, GitLab CI/CD, інші пайплайни) автоматизують збірку, виконання набору тестів (модульні,

інтеграційні, тестування контрактів) і просування релізів у контрольовані середовища. Важливими елементами є ізоляція середовищ тестування, можливість прогонки інтеграційних сценаріїв у віртуалізованих умовах та автоматичне відкатування при невдачі.

Спостереження, логування та аналіз інцидентів. Системи збору метрік, агрегації журналів і візуалізації забезпечують діагностику й оцінку працездатності. Приклади технічних рішень: Prometheus для збору метрік з подальшою візуалізацією в Grafana; стек для журналів Elasticsearch—Logstash—Kibana (ELK) або легші альтернативи для централізованого зберігання і пошуку логів. Для кореляції подій і швидкого реагування важливі механізми оповіщення й готові процедури реагування.

Керування ідентичністю та захист даних. Технології управління доступом і автентифікацією та протоколи авторизації дозволяють централізовано реалізувати ролі й політики доступу. Шифрування даних у спокої та при передачі (TLS, шифрування на рівні сховищ), керування ключами і журналювання доступів — обов’язкові складові. Підтримка процедур псевдонімізації й анонімізації має бути передбачена для відповідності нормативним вимогам.

Каталоги схем, реєстри контрактів і інструменти перевірки сумісності. Для мінімізації інтеграційного боргу необхідні центральні реєстри схем і контрактів, механізми автоматичної валідації повідомлень і інструменти для контрактного тестування. Наявність таких компонентів полегшує еволюцію системи та координацію між командами.

1.3. Порівняльний аналіз підходів, виявлення недоліків і обґрунтування потреби в новому підході

Порівняльний аналіз орієнтації на дані, процесно-орієнтованого підходу, організаційно-орієнтованого підходу та підходу з відкладеною інтеграцією здійснено через призму їхніх концептуальних абстракцій, механізмів забезпечення узгодженості поведінки системи, способів регламентації

нефункціональних вимог і практик управління еволюцією програмного комплексу. Кожна з названих парадигм визначає власний набір первинних артефактів і процедур: у першій — централізовану модель даних і правила роботи з нею, у другій — формалізовані сценарії виконання й контрольні точки, у третій — модель ролей і внутрішніх політик організації, у четвертій — принцип автономної розробки компонентів із відтермінованою інтеграцією. Аналіз показує, що при всій корисності цих абстракцій їхнє суцільне або роздільне застосування породжує системні прогалини на стиках семантики процесів, контрактної дисципліни, механізмів версіонування та забезпечення якості даних.

Орієнтація на дані забезпечує високу узгодженість у питаннях структури збереження інформації, історизації та трасування змін, що є необхідними умовами для виконання вимог до звітності й аудиту. Однак технічна специфіка таких архітектур, як правило, не містить формалізованого механізму трансформації опису бізнес-процесу в набір стабільних, версіонованих інтерфейсів та поведінкових контрактів. Унаслідок цього семантика виконання часто реалізується розпорошено — частково на рівні схеми даних, частково в локальних реалізаціях сервісів — що ускладнює відтворення поведінки системи при змінах і знижує ефективність автоматизованого тестування. Таким чином, сильна сторона підходу — цілісність даних — не конвертується автоматично у стійку контрактну дисципліну, необхідну для масштабованої інтеграції модулів.

Процесно-орієнтований підхід концентрує увагу на формалізації послідовностей дій, логіки ухвалення рішень і контрольних точок виконання. Це дає інструментарій для побудови відтворюваних сценаріїв і для проведення ретельного аудиту операційних потоків. Проте практична реалізація цього підходу стикається з проблемою трансляції процесної семантики до технічних артефактів: у відсутності чітких правил перетворення процесних моделей у специфікації API і структури даних виникають дисонанси між описом процесу і реалізованою поведінкою сервісів, що, в свою чергу, породжує необхідність ручних адаптацій, дублювання логіки й підтримки множинних версій компонентів. Отже, процесна формалізація підвищує якість опису поведінки, але

не вирішує питання синхронізації життєвих циклів процесів і програмних інтерфейсів.

Організаційно-орієнтований підхід фокусує увагу на моделюванні ролей, повноважень і механізмів відповідальності всередині організації. Така специфіка дозволяє встановити видимі і формальні механізми розподілу прав доступу й відповідальності за бізнес-операції, що сприяє підвищенню прозорості і зміцненню внутрішнього контролю. Водночас, коли технічні контракти і політики доступу не синхронізовані з організаційною моделлю, виникають розбіжності між декларативними правилами і фактично реалізованими механізмами авторизації й валідації. Відсутність інтеграції між організаційною семантикою та технічними артефактами веде до появи «сірих зон» у відповідальності та ускладнює процедури аудиту і менеджменту змін.

Підхід з відкладеною інтеграцією забезпечує гнучкість і паралелізм у розробці, дозволяючи командам працювати автономно над локальними компонентами. У короткостроковій перспективі це знижує координаційні витрати та прискорює розгортання окремих функцій. Однак відкладена інтеграція за відсутності дисциплінованої системи контрактного тестування, централізованого реєстру інтерфейсів та політик версіонування веде до наростання інтеграційного боргу: автономні еволюції форматів і поведінки компонентів породжують множинні адаптери, збільшують складність оркестрації й ускладнюють забезпечення надійності та відновлення після збоїв.

Паралельне зіставлення технічних аспектів — контрактів, політик версіонування, механізмів кешування, підходів до забезпечення доступності і аудиту — демонструє, що кожний із розглянутих підходів покриває певну підмножину вимог, але не дає цілісного рецепту для їх одночасного задоволення. Так, архітектури, що спираються на уніфіковану модель даних, забезпечують стабільні підвалини для збереження і ретенції інформації, проте не визначають механізмів відображення поведінки в інтерфейси; процесні архітектури надають контекст виконання, але не пропонують стандартного механізму для генерації машинночитних контрактів; організаційні моделі визначають розподіл

відповідальності, але без технічної інтеграції ролей і контрактів втрачається системна узгодженість; підхід з відкладеною інтеграцією дає операційну гнучкість, але при відсутності централізованих метаданих посилює фрагментацію і ускладнює управління якістю. Ці взаємні несумісності виявляються особливо очевидними у середовищах із жорсткими регуляторними вимогами, де одночасно потрібні трасування подій, відтворюваність процесів, чітке розмежування відповідальності та гарантована сумісність інтерфейсів.

Аналіз питомих витрат та ефектів для експлуатації вказує на компроміс між початковими інвестиціями в централізацію й дисципліну контрактної роботи та довгостроковою економією на підтримці і аудиті. Централізовані механізми реєстрації контрактів, версіонування та автоматизованого тестування вимагають значних ресурсних вкладень на етапі проєктування, проте зменшують ризики накопичення технічного боргу і скорочують операційні витрати надалі. Навпаки, підходи, орієнтовані на швидкість локальної розробки, забезпечують короткострокове прискорення впровадження функціоналу, але за відсутності системи раннього виявлення несумісностей формують передумови для значних витрат на пізні інтеграційні роботи.

Зіставлення придатності підходів для стійкої еволюції системи під час організаційних і технологічних змін показує, що жоден із них у чистому вигляді не забезпечує повний набір необхідних властивостей. Комбінація стабільності моделі даних, формалізації виконуваної семантики процесів, ясності розподілу відповідальності в організації та дисципліни контрактної інтеграції є необхідною для гарантування одночасного виконання функціональних та нефункціональних вимог у динамічних умовах. Відтак виявлені розриви — у правилах трансформації процесних описів у програмні контракти, у централізації метаданих про інтерфейси, у політиках співіснування версій та в інтеграції забезпечення якості даних у процесні потоки — формують предметні підстави для розробки методики, що поєднає переваги розглянутих парадигм і усуне зазначені прогалини.

2 ОБҐРУНТУВАННЯ МЕТОДІВ ДОСЛІДЖЕННЯ ТА ОПИС ПРОЦЕДУРИ ПРОЄКТУВАННЯ І ВАЛІДАЦІЇ

2.1. Методологія дослідження: підходи, методи збору й аналізу даних, критерії оцінки

Методологія дослідження ґрунтується на поєднанні теоретичних, аналітичних і прикладних підходів, що у своїй сукупності дозволяють комплексно обґрунтувати доцільність застосування удосконаленого процесно-орієнтованого підходу до проєктування архітектури інформаційних систем у сфері управління політичною партією. Враховуючи специфіку предметної області — високий рівень нормативного навантаження, потребу в аудиторській прозорості, необхідність безперервної синхронізації ролей, процесів і даних — методологія дослідження має забезпечувати не лише концептуальну аргументацію, а й можливість емпіричної перевірки заявлених рішень на основі експериментального прототипу.

Теоретичний рівень дослідження спирається на міждисциплінарну базу знань, що об’єднує наукові підходи до моделювання та виконання бізнес-процесів, теорії побудови даних-інтенсивних систем, методи проєктування сервісної інтеграції й практики менеджменту IT-повноваженнями та управлінням. У рамках даної роботи теоретична частина виконує дві взаємопов’язані функції: (1) забезпечує методологічні засади для формалізації і трансформації процесних артефактів у технічні контракти і структури, та (2) визначає сукупність оцінюваних властивостей (здатність до трасування, версіонування, масштабування та забезпечення якості даних), що лягають в основу критеріїв валідації прототипу. Ключові положення теоретичного рівня випливають із сучасних праць у сфері BPM (Business Process Management), досліджень архітектурних патернів для розподілених систем та робіт з інженерії даних.

Перший блок теоретичного коріння — теорія та практика управління бізнес-процесами. Власні підходи до моделювання процесів (семантика BPMN, моделі виконання, формалізований опис ролей і подій) слугують основою для побудови виконуваних процесних каркасів і для подальшої трансформації в сервісно-орієнтовану архітектуру. Класичні праці та огляди демонструють, що процесний підхід має подвійний характер: з одного боку — він надає компактну, формалізовану репрезентацію організаційної поведінки, з іншого — ставить вимогу наявності механізмів перетворення моделі в виконувані артефакти з детермінованою семантикою виконання й трасованістю екземплярів процесу. Роботи з систематизації BPM дають теоретичну базу для формулювання правил трансформації і вимагають уваги до аспектів версіонування процесів, відновлення після збоїв і сумісності змінених моделей із працюючими інстанціями.

Другий блок — дослідження з інженерії даних і архітектури даних-інтенсивних застосунків. Виклики, властиві предметній області (велика кількість записів членства, фінансові транзакції, історіографія дій), вимагають обґрунтування рішень щодо моделі зберігання, стратегій ретенції та архітектурних компромісів між консистентністю, доступністю й продуктивністю. Сучасна література з «data-intensive systems» підкреслює необхідність явного проєктування потоків даних, реплік, індексування й архітектурних шаблонів (проектування каналів ETL/ELT, поділ транзакційних і аналітичних шарів), що безпосередньо впливає на здатність системи відтворювати історичні стани для аудиту й звітності. Ці ідеї формують нормативи до дизайну канонічної моделі даних, до процедур версіонування схем і до інструментарію профілювання та валідації якості даних, які вбудовуються у бізнес-процеси.

Третій блок — архітектурні патерни для розподілених і модульних систем: сервісна декомпозиція, контракт-перший підхід, моделі версіонування API та практики контрактного тестування. Теоретичні й прикладні роботи у сфері мікросервісної інженерії та управління контрактами показують, що надійна

еволюція системи можливе лише за умови формалізованих контрактів та автоматизованих механізмів перевірки сумісності між постачальниками й споживачами сервісів. До ключових концепцій належать consumer-driven contracts, tolerant reader pattern і централізовані реєстри інтерфейсів, що дозволяють поєднати автономність розробки із контролем якості інтеграції. Теорія цих підходів забезпечує підґрунтя для впровадження механізмів, які мінімізують інтеграційний борг при відкладеній інтеграції компонентів.

Четвертий блок теоретичного рівня — специфіка організаційного говернансу і відповідність нормативним вимогам. Теорії управління IT-рішеннями й говернансу визначають, які рішення щодо розподілу повноважень, права прийняття рішень та відповідальності мають бути відображені в архітектурі. Праці з IT governance підкреслюють, що технічні артефакти (контракти, журнали аудиту, політики зберігання) мають співпадати з інституційними правилами, бо лише так забезпечується довіра під час зовнішніх перевірок і внутрішніх аудитів. Ці ідеї обґрунтовують необхідність включення до процесних моделей явних ролей, політик доступу й процедур затвердження змін, які потім проєктуються у технічний рівень (механізми авторизації, логування та зберігання).

П’ятий блок — спостережуваність, моніторинг і забезпечення якості роботи розподілених процесів. Сучасні практики вимірювання поведінки систем (метрики латентності, перцентильні показники, трасування транзакцій, агрегація логів) мають фундаментальне значення для валідації процесно-орієнтованих рішень у продакшн-умовах. Теоретичні підходи до побудови систем моніторингу й спостережуваності зосереджують увагу на кореляції слідів виконання, збиранні метрик і застосуванні інструментів для автоматичного виявлення аномалій, що необхідно для контролю поведінки бізнес-процесів і для своєчасного реагування на порушення нормативних вимог.

Практичний рівень дослідження зосереджено на розробці, впровадженні та експериментальній валідації прототипу архітектури, що реалізує положення удосконаленого процесно-орієнтованого підходу. Метою практичної складової

було перевести теоретично сформульовані принципи в конкретні інженерні артефакти, відтворити ключові сценарії предметної області та отримати кількісні й якісні дані для оцінки ефективності запропонованих рішень. Практична робота складалася з декількох взаємопов'язаних блоків: проєктування і реалізація прототипу, підготовка тестових наборів і симуляційних сценаріїв, налаштування експериментального середовища та автоматизація процедур збору й обробки результатів, а також проведення емпіричних випробувань і їх статистична обробка.

Проєктування й реалізація прототипу здійснювалися з урахуванням вимог до функціональності та нефункціональних властивостей, виявлених на попередніх етапах. Прототип включав компоненти для зберігання даних (реляційна та документоорієнтована підсистеми), шар керування метаданими і схемами, механізм централізованого опису інтерфейсів, сервісну площину для бізнес-логіки та інструменти для забезпечення кешування й спостережуваності. Для реалізації обрано стек технологій, що дозволяє швидко прототипувати й одночасно відтворювати типові архітектурні патерни: система управління метаданими і контентом як оперативний реєстр, механізм гнучкого формування запитів до даних, нативні сервіси для обробки логіки, оперативне кешування для зниження латентності, об'єктне сховище для медіа й бінарних артефактів, контейнери для ізоляції середовищ та пайплайни безперервної інтеграції і розгортання для автоматизації експериментів. Архітектурні рішення проектувалися з урахуванням можливості включення централізованого реєстру контрактів, механізмів версіонування інтерфейсів і політик управління кешем та ретенцією даних.

Підготовка тестових даних і сценаріїв була виконана із дотриманням вимог до репрезентативності та конфіденційності. Для відтворення типової поведінки системи сформовано комбіновані набори даних: синтетичні масиви, статистично наближені до реальних розподілів за кількістю і типами записів, а також анонімізовані витяги з реальних реєстрів, де це було допустимо з юридичної точки зору. Сценарії навантаження моделювали реальні бізнес-операції: масове

зчитування для звітності, інтенсивне оновлення реєстрів членства, транзакційні операції фінансування, одночасні запити адміністраторів і звичайних користувачів. Для кожного сценарію визначено профіль навантаження (інтенсивність запитів, співвідношення операцій читання/запису, часові інтервали пік-активності) та очікувані індикатори якості обслуговування.

Експериментальне середовище було організоване за принципом реплікованості: виділено окремі віртуальні чи фізичні ресурси для середовищ з однаковою конфігурацією мережі, IO та процесорних ресурсів; для кожного запуску експерименту створювався детальний опис конфігурації, що дозволяло відтворити умови тестування. Автоматизація експериментів здійснювалася через пайплайни безперервної інтеграції й розгортання, які виконували побудову артефактів, розгортання контейнерів, прогін тестів і збір телеметрії. Для імітації віддалених компонентів застосовувалися заглушки та імітатори поведінки, що дозволяло оцінити вплив пізньої інтеграції на систему без необхідності одночасного створення всіх компонентів.

Збір телеметрії та логування було організовано відповідно до вимог до кореляції подій і забезпечення відтворюваності. Кожному запиту надавався унікальний ідентифікатор кореляції, що дозволяло простежити шлях виконання через кілька сервісів і пов’язати відповідні логи з метриками. Телеметрія включала часові метрики відповіді, коди помилок, розміри навантаження на ресурси, показники кешування, а також події, пов’язані з виконанням процесів (вхід/вихід процесних кроків, транзакційні переходи, виконання компенсаційних дій). Зібрані дані зберігалися у структурованому форматі для подальшого автоматизованого аналізу.

Валідація функціональності й сумісності здійснювалася за допомогою контрольного набору тестів: модульних перевірок, інтеграційних випробувань та тестів контрактів. Особлива увага приділялася тестуванню контрактів між сервісами з точки зору очікувань споживача, що дозволяло виявляти дисонанси в інтерфейсних угодах ще на етапі розробки. Контрактні тести були інкорпоровані в пайплайн так, щоб кожне оновлення артефактів

супроводжувалося прогоном перевірок сумісності. Нефункціональне тестування включало навантажувальні тести, стрес-тести на межі пропускної здатності, а також сценарії ін’єкції відмов (часткові збої сервісів, мережеві затримки), після чого аналізувалися механізми відновлення та компенсації.

Статистична обробка результатів виконувалася з дотриманням методів, що забезпечують надійність висновків. Для кожного експериментального запуску проводили кілька повторень, виділялися періоди стабільної роботи, обчислювалися центральні статистики (медіана, середнє, перцентилі p95/p99), а також заходи розсіяння (дисперсія, стандартне відхилення). Порівняння стану до і після застосування змін виконувалося шляхом аналізу парних вибірок; у випадку невідповідності розподілів застосовувалися непараметричні критерії. Для виявлення статистично значущих відмінностей враховувалися обмеження щодо обсягу вибірки і гіпотези багатократних порівнянь, щоб уникнути хибних висновків.

Оцінка якості даних виконувалася через набір метрик: частка валідних записів, частка відхилень при перевірках валідації, частка дублікованих сутностей, середній час виявлення і виправлення інцидентів. Процедури очищення й анонімізації тестових даних документувалися окремо і застосовувалися до всіх наборів, отриманих із реальних джерел, для забезпечення конфіденційності та відповідності нормативним вимогам.

Насамкінець, у практичному рівні детально відпрацьовано процедури і політики, що мають важливе значення для подальшого промислового впровадження: політики версіонування API, правила поступового ввімкнення функціоналу, процедури відкату, правила збереження і видалення даних, а також рекомендації щодо розгортання системи в реальному середовищі з урахуванням вимог до безпеки і відповідності. Визнаючи обмеженість прототипу (масштаб, апаратні ресурси, імітація частини зовнішніх інтеграцій), практична частина роботи водночас забезпечує надійну емпіричну базу для висновків про ефективність запропонованих архітектурних рішень і формулювання методичних рекомендацій для реального впровадження.

Доповненням до теоретичного аналізу слугував емпіричний збір даних шляхом аналізу технічних вимог, документації, внутрішніх процедур та проєктних артефактів, що відображають реальні обмеження та сценарії роботи. Окремий блок емпіричного збору становили метрики, отримані внаслідок тестування прототипу, який реалізовано з використанням сучасного веб-стеку (API-шлюз, GraphQL, Directus як шар керування даними, Redis для кешування, MySQL/MongoDB для зберігання, технології контейнеризації та CI/CD). Прототип відтворює ключові елементи удосконаленого процесно-орієнтованого підходу: централізацію контрактів, політики кешування, механізми спостережуваності, а також керовану відкладену інтеграцію сервісів.

Аналіз зібраних даних здійснювався із застосуванням як кількісних, так і якісних методів. Кількісний аналіз охоплював вимірювання продуктивності (латентність, пропускна здатність, частка оброблених запитів у часових інтервалах), надійності (частота помилок, час відновлення компонентів), а також якості даних (повнота, коректність, рівень валідації та внутрішньої консистентності). Вимірювання проводилися у контрольованих умовах із чітко визначеним навантаженням, що відтворює типові робочі сценарії партійних інформаційних систем: масові операції читання, підготовку звітності, накопичення нових реєстрацій, обробку внутрішніх транзакцій.

Якісний аналіз включав оцінювання адекватності моделей і відповідності архітектурних рішень як бізнес-вимогам, так і організаційним обмеженням предметної галузі. Були проаналізовані типові розбіжності між формальною процесною моделлю та фактичною поведінкою роботи системи, а також джерела інтеграційних конфліктів, що дозволило сформулювати критерії необхідності удосконаленого процесно-орієнтованого підходу.

Результати дослідження верифікувалися через серію тестових сценаріїв, що включали функціональну перевірку виконання процесів, тестування API-контрактів, навантажувальні експерименти, аналіз відмовостійкості та відновлення після збоїв. Це дозволило оцінити, наскільки ефективно удосконалений підхід усуває виявлені у розділі 1 недоліки, забезпечує

передбачуваність інтеграції та стабільність роботи при високій змінності процесів і даних.

Критерії оцінки ефективності обраної методології були сформовані з урахуванням реальних вимог до систем управління політичною партією і включали:

1. відповідність функціональним вимогам та точність відтворення бізнес-процесів;
2. стабільність та передбачуваність API-контрактів у процесі еволюції;
3. масштабованість і продуктивність, зокрема здатність обробляти зростаючі навантаження;
4. якість даних та консистентність між процесами;
5. надійність і здатність до відновлення;
6. зниження інтеграційних витрат і підвищення прозорості управління;
7. відповідність нормативно-правовим вимогам та внутрішнім політикам організації.

У сукупності така методологія забезпечує можливість багатоаспектної оцінки ефективності архітектурних рішень, дозволяє перевірити працездатність запропонованого підходу у наближених до реальних умовах та забезпечує достатній рівень достовірності для формулювання подальших рекомендацій і висновків.

2.2. Методи трансформації процесних моделей у модульну архітектуру: формалізація data - API-контрактів та врахування організаційної структури

Початковим етапом трансформації процесних моделей в модульну архітектуру є стандартизація опису процесів у машинночитному форматі з розширенням метаданими, що фіксують семантику кожного елемента моделі. Для кожного кроку процесу мають бути формалізовані вхідні та вихідні артефакти, ролі та обов’язки виконавців, вимоги до цілісності й часові характеристики операцій, очікувана поведінка у випадку помилок та можливі

сценарії компенсації. Таке уніфіковане представлення забезпечує однозначну інтерпретацію процесних елементів, робить можливим автоматизований аналіз впливу змін і слугує вихідним артефактом для подальшого виділення модулів і формування контрактів на рівні даних та інтерфейсів. У предметній області партійного управління типові процеси, наприклад «реєстрація члена» або «реєстрація пожертви», повинні мати у метаданих вимоги щодо аудиту, збереження первинних документів і обмеження доступу згідно з нормативами.

Виділення модулів відбувається за інженерною логікою: у модуль як окрему одиницю реалізації переводяться ті кроки процесу, які мають високу внутрішню зв’язаність, власний життєвий цикл та обмежену потребу у зовнішніх залежностях. Межі модулів формуються з урахуванням прагнення до мінімізації взаємозалежностей і максимального локалізування відповідальності. Для прикладу, у системі партії доцільно виділяти окремі модулі для управління реєстром членства, обробки пожертв, підготовки фінансових звітів та обробки кадрових рішень; при цьому кожен такий модуль отримує у своїх метаданих вимоги щодо локальної політики збереження та контролю доступу, які можуть відрізнятися залежно від юрисдикції чи організаційної одиниці.

Формалізація контрактів на рівні даних включає побудову канонічних схем сутностей, де для кожного атрибуту визначено його тип, обов’язковість, правила валідації та семантику в термінах предметної області. До контракту на рівні даних додаються відомості про політику ретенції, вимоги до шифрування, категорії видимості (публічні, внутрішні, лише для аудиту) та атрибути аудиту (хто, коли, джерело операції). Такий підхід забезпечує однорідність інтерпретації сутностей між модулями і дає змогу автоматично генерувати валідатори, шаблони повідомлень і скелети тестів, що полегшує подальшу підтримку й еволюцію системи.

Процес побудови контрактів інтерфейсів базується на ясному мапуванні «процес→дані→інтерфейс»: для кожного кроку процесу визначається набір операцій (читання, створення, ініціація довготривалого процесу, оновлення стану), на основі яких формуються відповідні описи інтерфейсів із прив’язкою

до контрактів на рівні даних. У контракті інтерфейсу документуються кінцеві точки, формати запитів і відповідей, очікувані статуси, правила обробки помилок, вимоги до ідентифікації та авторизації, а також нефункціональні характеристики: очікувані часи відповіді, допустимі навантаження і вимоги до затримки синхронізації даних. Для операцій, що потребують негайного результату (наприклад, підтвердження платежу), доцільно визначати синхронний режим роботи, тоді як для багатофазних погоджень чи тривалих перевірок застосовується асинхронна взаємодія з механізмом відстеження стану екземпляра процесу.

Таблиця 2.1

Відповідність процесних кроків, модулів і контрактів

Елемент бізнес-процесу	Модуль та контракти
Реєстрація члена партії	Модуль: управління реєстром членства; Data: Member; API: ініціація реєстрації, передача документів, підтвердження; Події: MemberRegistered, MemberActivated
Фіксація пожертви	Модуль: фінансова діяльність; Data: Donation; API: створення пожертви, підтвердження платежу; Подія: DonationConfirmed
Проведення партійного заходу	Модуль: управління заходами; Data: Event, Participant; API: створення заходу, реєстрація учасника, закриття заходу; Події: EventClosed
Формування звітів	Модуль: звітність і аудит; Data: Report, AuditEntry; API: формування звіту, отримання архіву; Подія: ReportGenerated

Опис подій та обробка подійної взаємодії є невід’ємною частиною контракту для сценаріїв, де події є основним способом комунікації між модулями. У контракті події визначається структура повідомлення, мінімальний набір полів для аудиту, гарантії доставки та очікувана поведінка отримувача. Для прикладу, подія «пожертву підтверджено» повинна містити ідентифікатори

транзакції, дані платника, суму, часову мітку підтвердження і посилання на первинні документи, а також інформацію для аудиту, яка дозволяє відтворити повний ланцюжок прийняття рішення.

Механізми обробки помилок і компенсаційні дії мають бути формально визначені в контрактах і процесних моделях: у контракті описуються стандартні коди помилок, політики повторних спроб, таймаути, а також конкретні компенсаторні кроки для операцій, що охоплюють декілька модулів. У разі часткової невдачі повинна бути визначена логіка повернення системи у консистентний стан або виконання компенсаторних операцій, задокументованих як частина процесного сценарію; це дозволяє уникати неузгоджених станів і полегшує процедури відновлення.

Політика версіонування контрактів ґрунтується на принципах сумісності та безпечної еволюції: несумісні зміни позначаються як мажорні та супроводжуються періодом паралельного підтримання старої і нової версій, а сумісні розширення — як мінорні. При впровадженні нової версії повинен виконуватися аналіз впливу на споживачів, формуватися план поетапного переходу з визначенням часових вікон для депрекації та зобов’язань щодо збереження старих версій у необхідних юридичних періодах. Для операцій, пов’язаних із звітністю або нормативними циклами, план переходу має враховувати зовнішні терміни та не допускати перерв у можливості формування обов’язкових звітів.

Реєстр контрактів і метаданих виступає як єдине джерело істини: у ньому зберігаються описи сутностей і інтерфейсів, інформація про власників, список споживачів, історія змін, політики сумісності та вимоги до збереження. Інтеграція реєстру з процесом безперервної інтеграції й розгортання дає змогу автоматично виконувати перевірки впливу змін, генерувати скелети тестів та ініціювати набір процедур для контролю сумісності у середовищі інтеграції. Централізоване зберігання метаданих підвищує прозорість, скорочує ризики дублювання специфікацій і полегшує аудит.

Контрактне тестування впроваджується як обов’язкова складова життєвого циклу розробки: очікування споживача формалізуються у вигляді тестів, які автоматично виконуються проти реалізації в процесі кожного прогона збірки. Таке тестування дозволяє виявляти невідповідності на ранніх етапах і значно знижує кількість інтеграційних дефектів у передрелізній фазі. Тести охоплюють «щасливі» сценарії, граничні випадки, обробку помилок і перевірки нефункціональних параметрів.

Нефункціональні політики включаються до метаданих контрактів і підлягають моніторингу під час експлуатації. До таких політик належать правила кешування та інвалідизації, цільові часи відповіді, стратегії синхронізації даних та політики ретенції. Впровадження централізованого кешування з механізмом контролю інвалідизації при зміні стану дозволяє знизити затримки у пікових режимах, водночас забезпечуючи консистентність даних для критичних операцій, наприклад фінансових транзакцій.

Захист персональних даних та відповідність правовим нормам інтегруються в контракти через вимоги до автентифікації, авторизації, шифрування даних у спокої та при передачі, а також процедури обробки запитів суб’єктів даних (наприклад, видалення або обмеження обробки). Ролі в організаційній структурі партії мають бути формально пов’язані з технічними політиками доступу, що забезпечує підзвітність дій і можливість відтворення рішень у процесі аудиту.

Питання міграції стану активних інстанцій процесів під час еволюції моделі вирішується шляхом версіонування формату стану та розробки трансформаційних сценаріїв: створюються дескриптори стану, які дозволяють автоматично перетворювати старі представлення у нові, або визначаються умови для ручової обробки складних випадків. Наявність чіткої процедури міграції зменшує ризик зупинки бізнес-процесів і забезпечує безперервність операцій при оновленні моделей.

Інтеграція методики з процесами безперервної інтеграції та розгортання забезпечує відтворюваність, автоматичне генерування тестів з контрактів і

поступове виведення нових версій у продукційне середовище з використанням поетапних сценаріїв розгортання та моніторингу ключових показників. Управління змінами здійснюється міжфункціональною комісією, що проводить аналіз впливу, погоджує план міграції та фіксує рішення у реєстрі контрактів, забезпечуючи прозорість і відповідність процедур.

Показники якості трансформації включають кількість інтеграційних дефектів на реліз, частку успішних контрактних перевірок у процесі безперервної інтеграції, середній час впровадження функціоналу, показники продуктивності (медіани та перцентилі латентності), відсоток попадань у кеш і метрики якості даних. Систематичний моніторинг цих показників дозволяє вчасно коригувати процедури і гарантувати досягнення цілей щодо передбачуваності інтеграції, прозорості та нормативної відповідності.

Слід також зазначити типові ризики та заходи їх мінімізації: необхідність дисципліни при наповненні метаданих, ризик надмірної деталізації контрактів, яка ускладнює їх підтримку, і ресурсні витрати на підтримку паралельних версій у критичні періоди. Ці ризики пом’якшуються шляхом запровадження правил «достатньої формалізації», регулярних ревізій реєстру контрактів і чітких процедур управління змінами, що поєднують технічні й організаційні механізми контролю.

2.3. Проєктування експериментальної платформи і забезпечення достовірності результатів

Для перевірки ефективності запропонованих методів трансформації процесних моделей у модульну архітектуру була розроблена експериментальна платформа на базі headless CMS Directus версії 11.4.1. Вибір цієї платформи обумовлений наступними чинниками: відкритою модульною архітектурою, підтримкою розширень через кастомні endpoint'и, нативною підтримкою REST та GraphQL API, а також можливістю формалізації data-контрактів через вбудовану систему схем бази даних. Експериментальна платформа дозволяє

продемонструвати практичну реалізацію теоретичних підходів, описаних у попередніх розділах, та забезпечити об'єктивну оцінку ефективності запропонованих методів.

2.3.1 Архітектура експериментальної платформи та обґрунтування вибору технологій

Експериментальна платформа побудована на принципах мікросервісної архітектури з використанням технології контейнеризації Docker, що забезпечує ізольованість компонентів, легкість розгортання та відтворюваність експериментального середовища. Архітектура платформи реалізована у вигляді Docker Compose конфігурації, що дозволяє одночасно запускати всі необхідні сервіси та забезпечує їх коректну взаємодію через внутрішню мережу контейнерів.

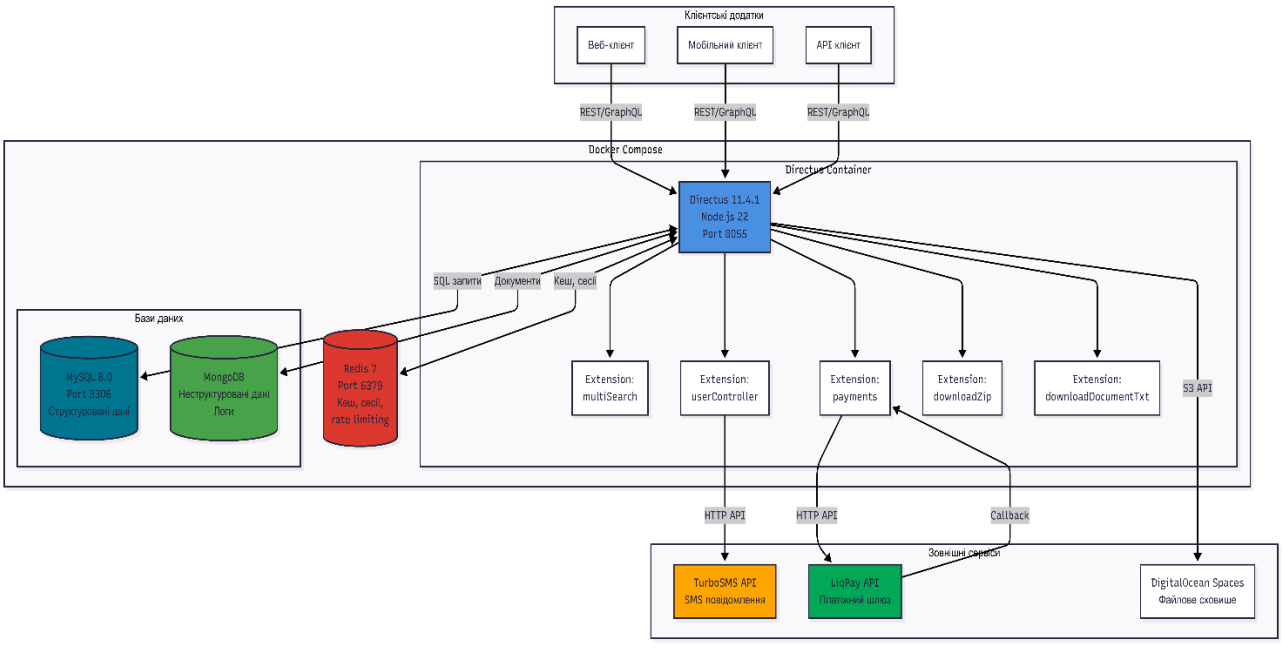


Рис. 2.1 Схема архітектури системи

Основний серверний додаток побудований на базі Directus 11.4.1, який працює на платформі Node.js версії 22. Directus є headless CMS системою, що надає RESTful API та GraphQL endpoint для роботи з даними, а також

адміністративний інтерфейс для управління структурою даних та правами доступу. Вибір Directus обумовлений тим, що ця система має відкриту модульну архітектуру, дозволяє створювати кастомні розширення через механізм extensions, та автоматично генерує API на основі структури бази даних, що ідеально підходить для демонстрації методів формалізації data-контрактів.

Для зберігання структурованих даних використовується реляційна база даних MySQL версії 8.0. MySQL обрана як основна база даних для зберігання структурованих даних, метаданих схем колекцій, зв'язків між сутностями та налаштувань системи. Реляційна модель даних дозволяє формалізувати data-контракти через чітко визначені таблиці, поля, типи даних, обмеження цілісності та зовнішні ключі. Це забезпечує структурованість даних та дозволяє автоматично генерувати API-контракти на основі схем бази даних.

Одночасно з MySQL в проєкті використовується документоорієнтована база даних MongoDB для зберігання неструктурованих або напівструктурованих даних, логів системи, кешованих результатів складних запитів та тимчасових даних, які не потребують жорсткої схеми. Використання двох різних типів баз даних дозволяє продемонструвати гібридний підхід до зберігання даних, де реляційна база використовується для структурованих даних з чітко визначеними зв'язками, а документоорієнтована база для гнучких структур даних, які можуть змінюватися в часі. Такий підхід відповідає принципам модульної архітектури, де кожен компонент вибирається відповідно до специфіки вирішуваних задач.

Для кешування даних, зберігання сесій користувачів та обмеження частоти запитів використовується Redis версії 7. Redis працює як база даних у пам'яті, що забезпечує високу швидкість доступу до кешованих даних та ефективну реалізацію механізмів rate limiting. Використання Redis дозволяє зменшити навантаження на основні бази даних, прискорити відповіді на часті запити та забезпечити стабільність системи під високим навантаженням.

Таблиця 2.2

Порівняльна характеристика MySQL 8.0 та MongoDB

Характеристика	**MySQL 8.0**	**MongoDB**
Тип бази даних	Реляційна (RDBMS)	Документоорієнтована (NoSQL)
Модель даних	Таблиці з рядками та колонками	Колекції з документами (JSON-подібні)
Схема	Жорстка схема, визначена заздалегідь	Гнучка схема, може змінюватися
Зв'язки між даними	Зовнішні ключі, JOIN-операції	Вбудовані документи, посилання
Транзакції	ACID-транзакції, підтримка множинних таблиць	ACID-транзакції, обмежені
Мова запитів	SQL (структурована)	Мова запитів MongoDB
Масштабування	Вертикальне та горизонтальне (шардінг)	Горизонтальне, нативна підтримка
Використання в проєкті	Структуровані дані, метадані, зв’язки	Неструктуровані дані, логи, кеш
Приклади даних	Користувачі, документи, платежі	Логи системи, тимчасові дані
Індексація	B-tree, повнотекстові індекси	B-tree, текстові, геопросторові
Консистентність	Сильна консистентність	Налаштовувана консистентність
Швидкість читання	Висока для структурованих запитів	Висока для документів
Швидкість запису	Висока з транзакціями	Дуже висока для простих операцій
Складність запитів	Складні JOIN та підзапити	Обмежена підтримка складних запитів
Підтримка JSON	Нативна підтримка JSON-типів	Нативна робота з JSON
Версіонування	Через міграції схеми	Через зміну структури документів

Архітектура платформи передбачає послідовну взаємодію компонентів через чітко визначені інтерфейси. Клієнтські додатки взаємодіють з Directus через REST або GraphQL API, Directus виконує бізнес-логіку через вбудовані сервіси та кастомні endpoint extensions, дані зберігаються в MySQL або MongoDB залежно від їх структури, а Redis забезпечує кешування та обмеження навантаження. Зовнішні сервіси, такі як TurboSMS для відправки SMS-

повідомлень та LiqPay для обробки платежів, інтегруються через HTTP API, що демонструє принципи інтеграції модулів через стандартні протоколи.

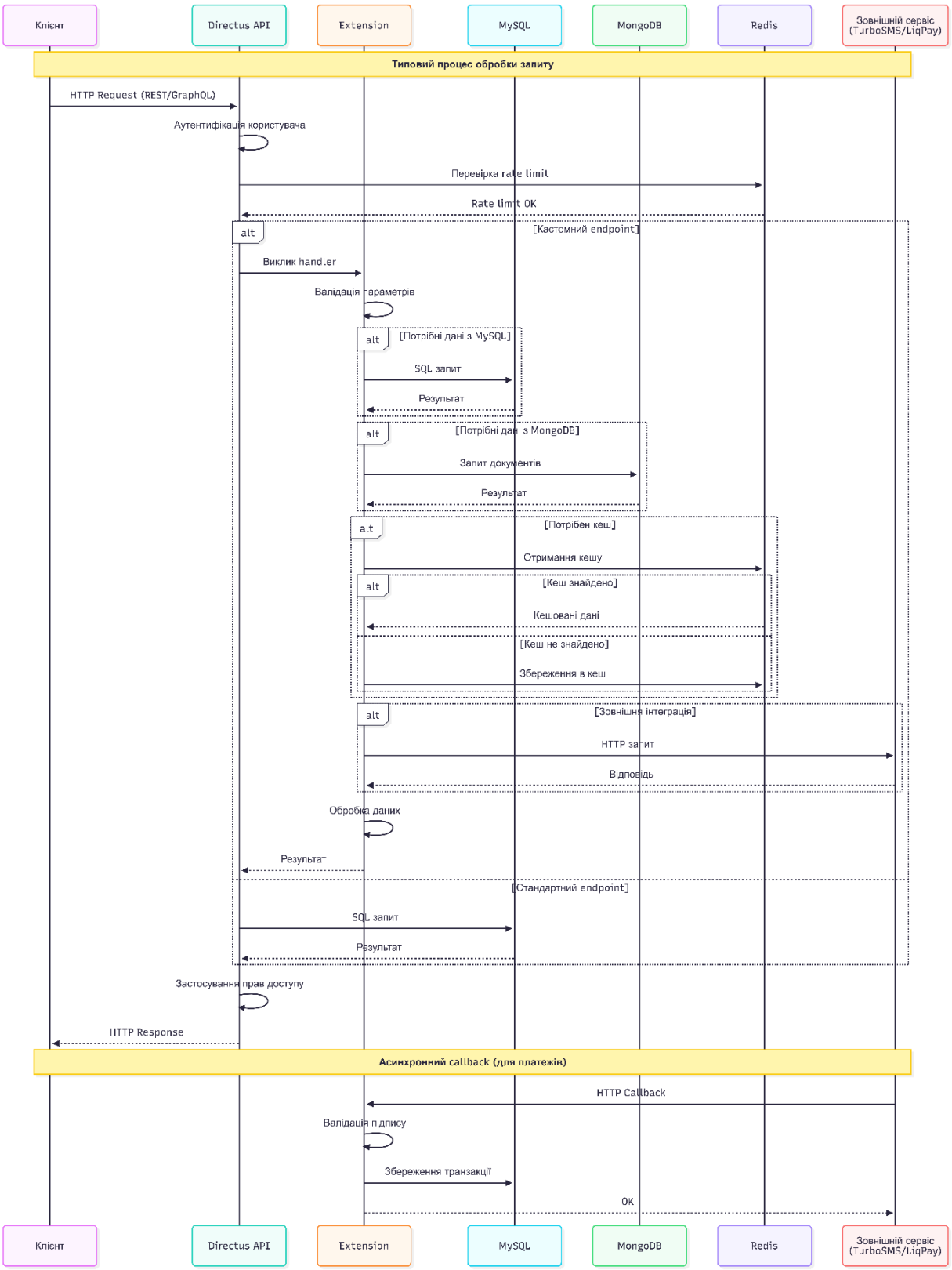


Рис. 2.2 Діаграма взаємодії компонентів системи

Кожен сервіс має власні перевірки стану, що гарантують коректну послідовність запуску компонентів та автоматичне відновлення при збоях. Залежності між сервісами визначені через параметр depends_on у Docker Compose, що забезпечує, що база даних та Redis будуть готові до роботи до того, як Directus спробує до них підключитися.

2.3.2 Реалізація модульної архітектури через кастомні розширення

Для демонстрації методів трансформації процесних моделей у модульну архітектуру були розроблені чотири кастомні extensions, кожен з яких реалізує окремий модуль функціональності та демонструє принципи модульного проєктування.

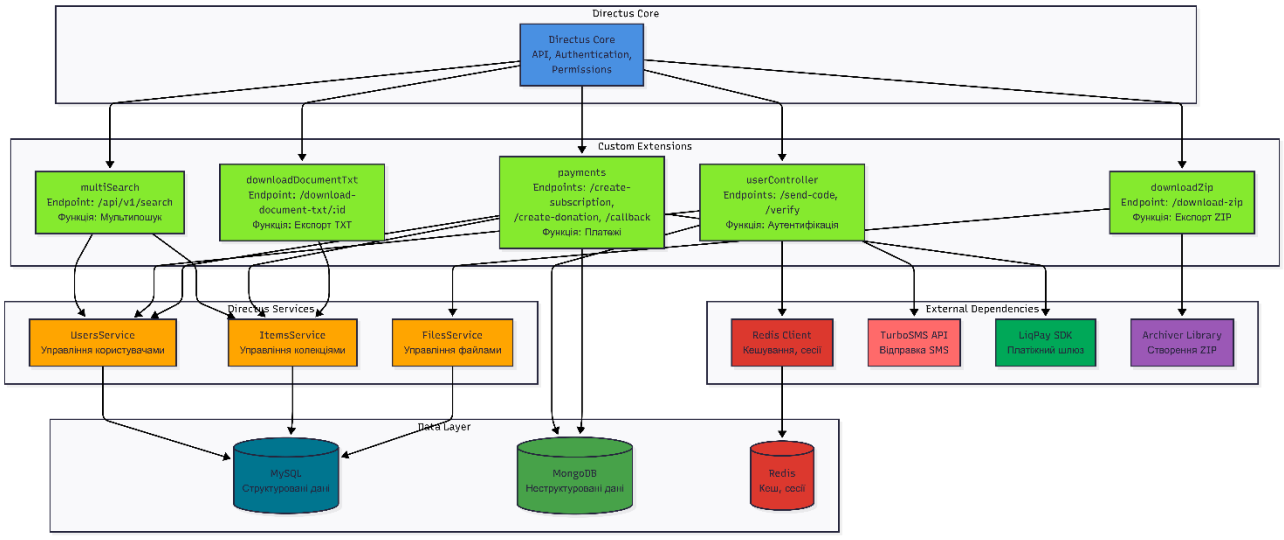


Рис. 2.3 Структурна схема модулів системи

Кожен extension оформлений як окремий модуль з власним package.json, що дозволяє незалежне тестування, розгортання та підтримку. Структура extensions відповідає принципам SOLID, зокрема принципу єдиної відповідальності, де кожен модуль відповідає за конкретну область функціональності, та принципу відкритості та закритості, де модулі відкриті для

розширення через додавання нових endpoint'ів, але закриті для модифікації існуючої функціональності.

Перший extension реалізує функціонал мультипошуку через endpoint `/api/v1/search`. Цей модуль дозволяє виконувати пошук по множині колекцій, таких як користувачі, документи, події та задачі, в одному запиті. Реалізація демонструє принцип агрегації запитів, де замість виконання окремих запитів до кожної колекції виконується один уніфікований запит, що зменшує кількість обмінів запитами між клієнтом та сервером та підвищує продуктивність системи. Модуль використовує сервіси Directus для роботи з різними колекціями, забезпечує дедуплікацію результатів та повертає структуровану відповідь з результатами пошуку по всіх колекціях. Це ілюструє, як процесна модель пошуку інформації трансформується у модульну архітектуру з чітко визначеним інтерфейсом та інкапсульованою логікою.

Другий extension реалізує модуль контролю користувачів через endpoints `/api/v1/send-code` та `/api/v1/verify`. Цей модуль забезпечує двофакторну аутентифікацію через SMS-коди за допомогою інтеграції з TurboSMS API. Модуль демонструє інкапсуляцію логіки аутентифікації та зовнішніх інтеграцій у окремий компонент, що відповідає принципам модульної архітектури. Процес аутентифікації включає генерацію одноразового пароля, відправку SMS через зовнішній сервіс, зберігання коду в Redis з обмеженим часом життя, перевірку коду при верифікації та автоматичне створення або оновлення користувача в системі. Використання Redis для тимчасового зберігання OTP-кодів забезпечує безпеку, оскільки коди автоматично видаляються після перевірки або закінчення терміну дії, та продуктивність, оскільки доступ до Redis значно швидший ніж до основної бази даних. Модуль також реєструє всі спроби відправки та верифікації кодів у системі нотифікацій, що дозволяє відстежувати процес аутентифікації та виявляти аномалії.

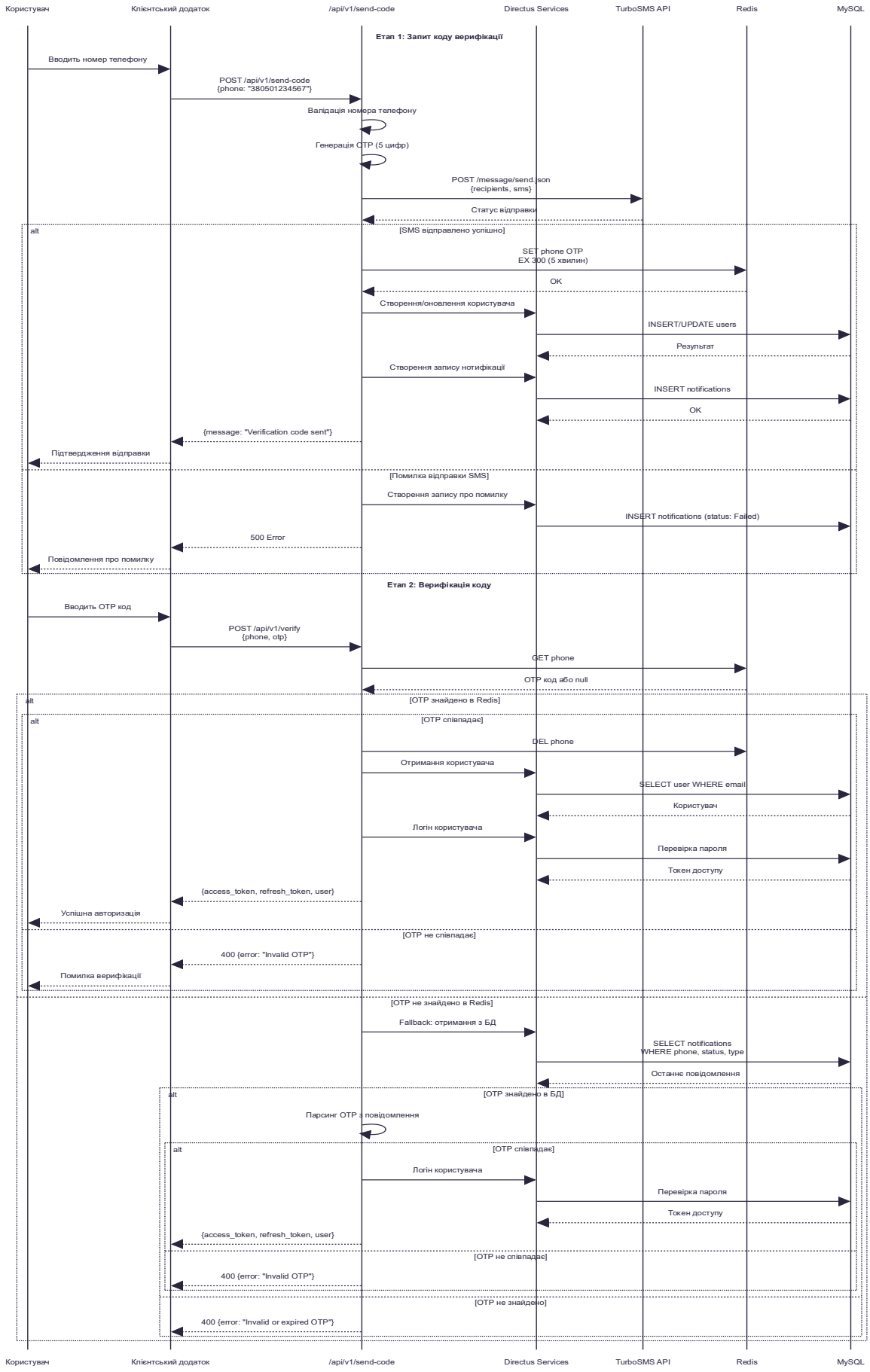


Рис. 2.4 Діаграма послідовності процесу двофакторної аутентифікації

Третій extension реалізує платіжну систему через endpoints `/api/v1/create-subscription`, `/api/v1/create-donation` та `/api/v1/callback`. Модуль включає обробку підписок та пожертв через інтеграцію з LiqPay платежним шлюзом. Реалізація демонструє обробку асинхронних процесів, де ініціалізація платежу відбувається синхронно, а підтвердження платежу приходить асинхронно через callback від платіжної системи. Модуль включає валідацію підписів транзакцій за допомогою криптографічних хеш-функцій, що забезпечує безпеку та запобігає підробці платежів. Управління станом підписок реалізовано з урахуванням можливості продовження існуючих активних підписок, де новий платіж продовжує термін дії підписки від поточної дати закінчення, а не від дати платежу. Це ілюструє обробку складних бізнес-процесів у модульній архітектурі, де логіка інкапсульована в окремий модуль, але зберігає складність обробки станів та асинхронних подій.

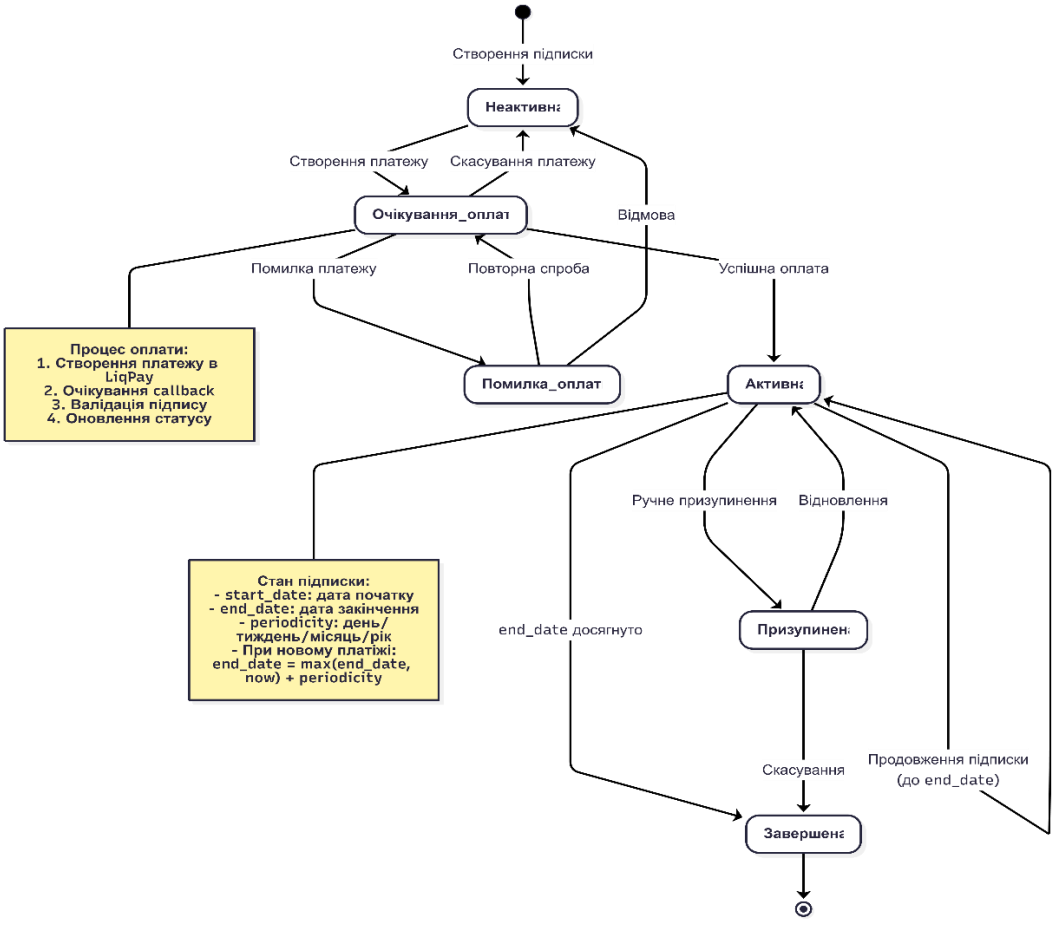


Рис. 2.5 Діаграма станів системи управління підписками

Четвертий extension реалізує функціонал експорту даних через endpoints `/api/v1/download-zip` та `/api/v1/download-document-txt/:id`. Ці модулі забезпечують експорт файлів у форматі ZIP та документів у текстовому форматі відповідно. Реалізація демонструє обробку потоків даних, де файли завантажуються з системи зберігання, обробляються та передаються клієнту у вигляді потоку, що дозволяє ефективно працювати з великими обсягами даних без необхідності завантаження всіх даних в пам'ять одночасно. Модуль генерації ZIP-архівів використовує бібліотеку archiver для створення архівів на льоту, додаючи файли до архіву по мірі їх завантаження, що забезпечує ефективне використання пам'яті та швидкий початок передачі даних клієнту. Модуль генерації текстових документів демонструє трансформацію структурованих даних з бази даних у текстовий формат, що може використовуватися для експорту або друку документів.

Всі extensions використовують систему обліку користувачів Directus для забезпечення безпеки та контролю доступу. Кожен запит містить інформацію про користувача, який його виконує, та його права доступу, що автоматично застосовується при виконанні операцій з даними. Це демонструє, як організаційна структура враховується на рівні модульної архітектури через систему прав доступу, де кожен модуль автоматично дотримується обмежень, визначених для користувача.

2.3.3 Використання GraphQL для забезпечення гнучкості API

Окрім REST API, експериментальна платформа надає GraphQL endpoint, який забезпечує більш гнучкий підхід до отримання даних. GraphQL є мовою запитів та системою виконання запитів, розробленою Facebook, яка дозволяє клієнтам точно визначати, які дані вони потребують, замість отримання фіксованого набору даних, як у традиційному REST API. Directus нативно підтримує GraphQL та автоматично генерує GraphQL схему на основі структури

колекцій у базі даних, що дозволяє формалізувати API-контракти через GraphQL схему.

Використання GraphQL в експериментальній платформі дозволяє продемонструвати, як модульна архітектура може забезпечити гнучкість у способах доступу до даних. GraphQL дозволяє клієнтам виконувати складні запити, які отримують дані з множини пов'язаних колекцій в одному запиті, що зменшує кількість обмінів запитами між клієнтом та сервером порівняно з REST API, де для отримання пов'язаних даних потрібно виконувати окремі запити. Це особливо корисно для складних інтерфейсів, де потрібно відображати дані з різних колекцій одночасно.

GraphQL також забезпечує строгу типізацію через систему типів, де кожне поле має визначений тип, що дозволяє виявляти помилки на етапі розробки та забезпечує коректність даних. Схема GraphQL автоматично оновлюється при зміні структури колекцій у Directus, що забезпечує синхронізацію між структурою даних та API-контрактами. Це демонструє, як формалізація data-контрактів через GraphQL схему забезпечує автоматичну валідацію запитів та відповідей, що підвищує надійність системи.

Directus також підтримує GraphQL через WebSockets, що дозволяє реалізувати підписки на зміни даних у реальному часі. Це дозволяє клієнтам отримувати оновлення даних автоматично без необхідності періодичного опитування сервера, що зменшує навантаження на систему та забезпечує актуальність даних у клієнтських додатках. Використання WebSockets для GraphQL підписок демонструє, як модульна архітектура може забезпечити різні способи взаємодії з даними, адаптовані під конкретні потреби клієнтів.

Таблиця 2.3

Порівняння характеристик REST API та GraphQL

Характеристика	**REST API**	**GraphQL**
Архітектурний стиль	Ресурсно-орієнтований	Запит-орієнтований
Кількість endpoint'ів	Множина endpoint'ів для різних ресурсів	Один endpoint для всіх запитів
Структура запиту	HTTP метод + URL	Запит з визначенням потрібних полів
Версіонування	Через URL (/api/v1, /api/v2)	Через схему, зазвичай без версій
Отримання даних	Фіксований набір даних з ресурсу	Клієнт визначає потрібні поля
Кількість запитів	Може потребувати кілька запитів для пов'язаних даних	Один запит для отримання пов'язаних даних
Over-fetching	Можливе (отримання зайвих даних)	Мінімальне (тільки потрібні дані)
Under-fetching	Можливе (потрібні додаткові запити)	Мінімальне (все в одному запиті)
Типізація	Через документацію (OpenAPI)	Строга типізація через схему
Валідація	На рівні сервера, залежить від реалізації	Автоматична валідація через схему
Кешування	HTTP кешування (стандартне)	Складніше, потребує спеціальних рішень
Помилки	HTTP статус коди	Структуровані помилки в відповіді
Підписки (real-time)	Потребує окремих технологій (WebSockets, SSE)	Нативна підтримка через підписки
Складність запитів	Обмежена HTTP методами	Складні вкладені запити
Документація	OpenAPI / Swagger	GraphQL схема (самодокументована)
Навчання	Простіше, стандартний HTTP	Потребує вивчення GraphQL синтаксису
Підтримка в Directus	Нативна, автоматична генерація	Нативна, автоматична генерація схеми
Використання в проєкті	Кастомні extensions, стандартні CRUD операції	Складні запити з множиною колекцій, real-time оновлення

2.3.4 Формалізація data-контрактів та API-специфікацій

Для забезпечення достовірності результатів експериментів була проведена формалізація data-контрактів через кілька взаємопов'язаних механізмів, які забезпечують чітке визначення структури даних та способів їх обміну між компонентами системи.

Перший механізм формалізації реалізований через схеми бази даних MySQL. Directus автоматично генерує REST API на основі структури таблиць MySQL, де кожна колекція має чітко визначені поля, типи даних, обмеження та зв'язки. Це формалізує структуру даних на рівні бази, де зміни в структурі таблиць автоматично відображаються в API, що забезпечує синхронізацію між структурою даних та API-контрактами. Реляційна модель даних дозволяє визначити зв'язки між сутностями через зовнішні ключі, що формалізує відносини між даними та забезпечує цілісність даних на рівні бази.

Для MongoDB формалізація data-контрактів реалізована через визначення схем документів на рівні коду, де кожен тип документа має визначену структуру з вказанням обов'язкових та опціональних полів, типів даних та валідаційних правил. Хоча MongoDB не вимагає жорсткої схеми, визначення структури документів на рівні коду забезпечує консистентність даних та дозволяє формалізувати контракти для неструктурованих даних.

Другий механізм формалізації реалізований через OpenAPI 3.0 специфікацію, яка описує всі кастомні endpoint'и, їх параметри, формати запитів та відповідей, коди статусів та приклади використання. OpenAPI специфікація забезпечує формальний опис API, що дозволяє автоматично генерувати клієнтські SDK, проводити валідацію запитів на рівні інфраструктури та забезпечує документацію API, яка завжди синхронізована з реалізацією. Використання стандартизованого формату опису API забезпечує сумісність з різними інструментами розробки та тестування, що підвищує якість розробки та спрощує інтеграцію з іншими системами.

Третій механізм формалізації реалізований через GraphQL схему, яка автоматично генерується Directus на основі структури колекцій. GraphQL схема визначає всі доступні типи даних, поля, аргументи запитів та зв'язки між типами, що забезпечує строгу типізацію та валідацію запитів на рівні виконання. Схема GraphQL служить як жива документація API, оскільки вона завжди відображає поточну структуру даних, та дозволяє клієнтам досліджувати доступні дані та способи їх отримання через інструменти типу GraphQL Playground.

Четвертий механізм формалізації реалізований через type safety на рівні extensions. Хоча extensions написані на JavaScript, вони використовують строгу валідацію параметрів запитів, перевірку типів даних на етапі виконання та обробку помилок, що запобігає помилкам типізації при обробці запитів. Кожен endpoint перевіряє наявність обов'язкових параметрів, формат даних та права доступу користувача перед виконанням операції, що забезпечує коректність результатів та запобігає помилкам через некоректні вхідні дані.

П'ятий механізм формалізації реалізований через врахування організаційної структури через систему ролей та прав доступу Directus. Система дозволяє формалізувати обмеження доступу на основі організаційної структури користувачів, де кожен користувач має визначену роль з конкретними правами доступу до колекцій та полів. Кожен запит містить контекст користувача через систему accountability, що застосовується автоматично при виконанні операцій з даними, забезпечуючи, що користувач може отримати доступ тільки до даних, на які він має права. Це формалізує правила доступу до даних та забезпечує, що організаційна структура враховується на рівні API-контрактів.

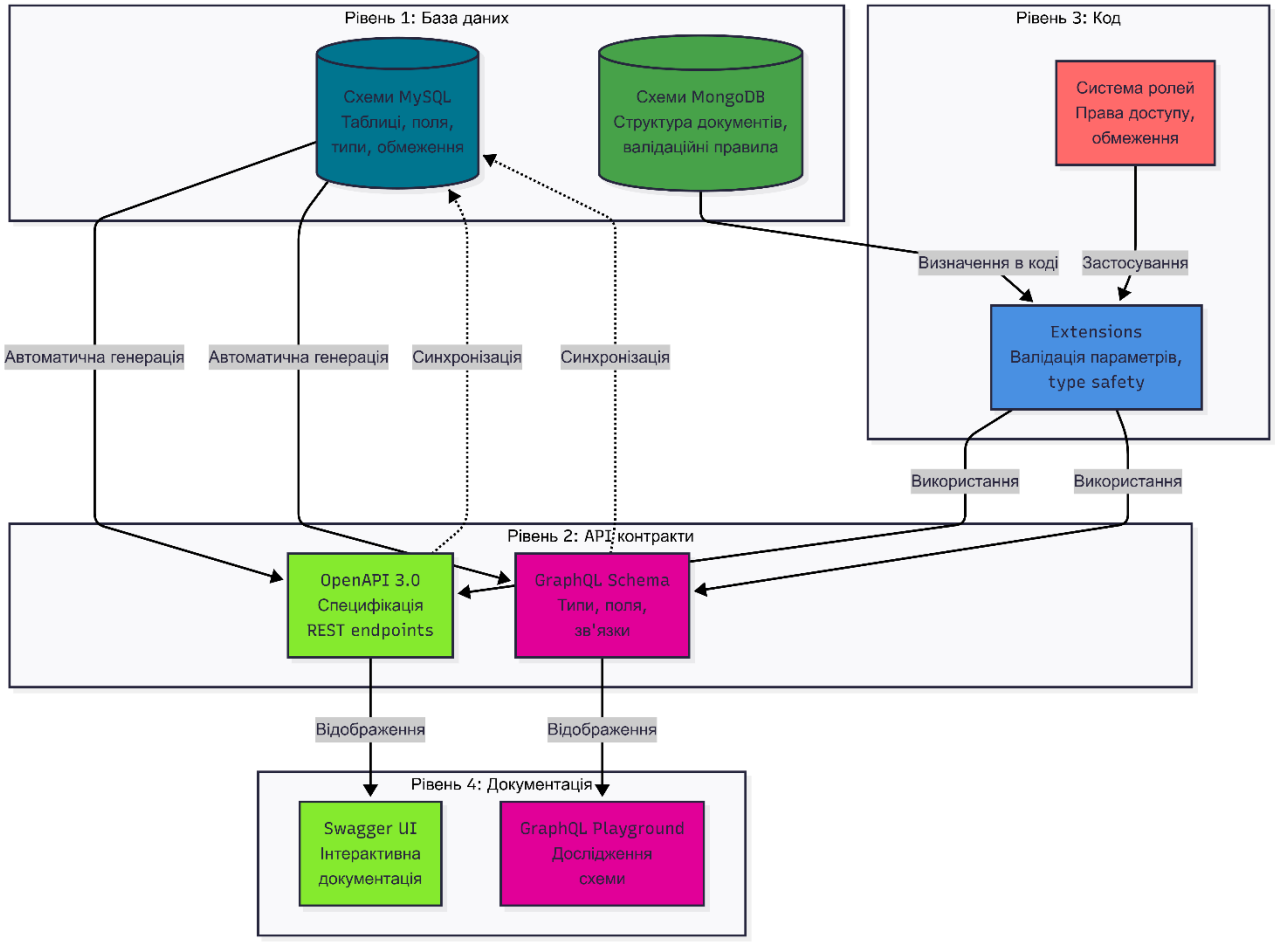


Рис. 2.6 Діаграма багаторівневої формалізації data-контрактів

Всі ці механізми працюють разом, забезпечуючи багаторівневу формалізацію data-контрактів, де структура даних визначається на рівні бази даних, API-контракти формалізуються через OpenAPI та GraphQL схеми, а права доступу визначаються через систему ролей. Це забезпечує повну прозорість структури даних та способів їх обміну, що є критичним для забезпечення достовірності результатів експериментів.

2.3.5 Методологія тестування та забезпечення достовірності результатів

Для забезпечення достовірності результатів експериментальних досліджень було застосовано комплексний підхід до тестування та валідації, який включає кілька взаємопов'язаних методів, спрямованих на забезпечення коректності, стабільності та відтворюваності результатів.

Перший метод забезпечення достовірності реалізований через ізольованість середовища виконання за допомогою Docker контейнерів. Використання контейнеризації забезпечує однакові умови виконання експериментів незалежно від хостової системи, операційної системи або конфігурації обладнання. Версії всіх залежностей фіксуються в docker-compose.yml та package.json, що гарантує, що кожен експеримент виконується з однаковими версіями всіх компонентів. Це забезпечує відтворюваність результатів, оскільки будь-який експеримент може бути повторено на будь-якій системі з однаковими результатами, якщо використовуються однакові версії компонентів. Health checks для всіх сервісів забезпечують, що експерименти виконуються тільки після того, як всі компоненти системи готові до роботи, що запобігає помилкам через неготовність компонентів.

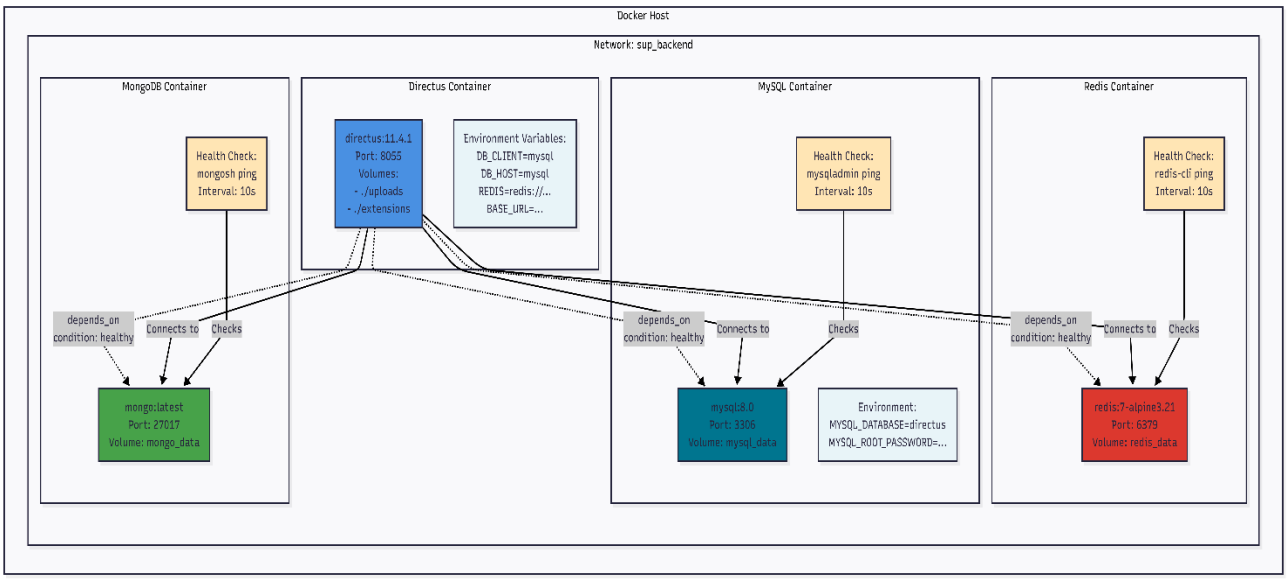


Рис. 2.7 Структура Docker контейнерів та їх залежності

Другий метод забезпечення достовірності реалізований через комплексне логування та моніторинг всіх операцій системи. Всі extensions містять детальну обробку помилок з логуванням у консоль, що дозволяє відстежувати виконання операцій та виявляти проблеми. Система нотифікацій реєструє всі зовнішні інтеграції, такі як відправка SMS через TurboSMS та обробка платежів через

LiqPay, з інформацією про статус операції, що дозволяє відстежувати успішність операцій та виявляти аномалії. Логування включає інформацію про час виконання операцій, параметри запитів, результати виконання та деталі помилок, що дозволяє аналізувати поведінку системи та виявляти проблеми продуктивності або коректності.

Третій метод забезпечення достовірності реалізований через валідацію вхідних даних на рівні кожного endpoint'a. Кожен endpoint перевіряє наявність обов'язкових параметрів, формат даних, діапазони значень та права доступу користувача перед виконанням операції. Це забезпечує коректність результатів та запобігає помилкам через некоректні вхідні дані. Валідація включає перевірку типів даних, обов'язкових полів, форматів рядків, діапазонів числових значень та наявності пов'язаних сутностей у базі даних. Якщо валідація не проходить, endpoint повертає детальне повідомлення про помилку, що дозволяє швидко виявити та виправити проблеми.

Четвертий метод забезпечення достовірності реалізований через використання транзакцій бази даних для критичних операцій. Транзакції забезпечують атомарність операцій, де або всі зміни застосовуються одночасно, або жодна зміна не застосовується, що забезпечує цілісність даних навіть при часткових збоях. Це особливо важливо для операцій, які включають зміни в кількох таблицях одночасно, таких як створення підписок разом з реєстрацією платежів, або оновлення користувача разом зі створенням записів у пов'язаних таблицях. Використання транзакцій забезпечує, що дані завжди знаходяться в консистентному стані, навіть якщо операція перервана через помилку або збій системи.

П'ятий метод забезпечення достовірності реалізований через rate limiting та захист від перевантаження за допомогою Redis. Впровадження rate limiting обмежує кількість запитів з одного IP-адреси за одиницю часу, що забезпечує стабільність системи під навантаженням та запобігає DDoS-атакам, які могли б спотворити результати експериментів. Rate limiting також забезпечує справедливий розподіл ресурсів між користувачами та запобігає ситуаціям, коли

один користувач може перевантажити систему, що впливає на результати експериментів інших користувачів. Налаштування rate limiting дозволяє визначити оптимальний баланс між продуктивністю та захистом, де система може обробляти нормальне навантаження, але блокує надмірні запити.

Шостий метод забезпечення достовірності реалізований через версіонування API за допомогою префіксу `/api/v1` для всіх кастомних endpoint'ів. Версіонування дозволяє вводити нові версії API без порушення існуючих інтеграцій, що забезпечує стабільність експериментальної платформи протягом тривалого періоду тестування. Це дозволяє вносити покращення та виправлення в API, не впливаючи на існуючі експерименти, що забезпечує консистентність результатів навіть при оновленні системи. Версіонування також дозволяє підтримувати кілька версій API одночасно, що дає можливість поступово мігрувати клієнтів на нові версії без необхідності миттєвого оновлення всіх інтеграцій.

Сьомий метод забезпечення достовірності реалізований через автоматичні перевірки стану для всіх сервісів. Перевірки стану забезпечують автоматичне виявлення несправностей компонентів системи та запобігають виконанню експериментів у некоректному стані системи. Якщо будь-який компонент не проходить перевірку стану, система автоматично намагається його відновити або блокує виконання операцій до відновлення коректного стану. Це забезпечує, що експерименти виконуються тільки коли всі компоненти системи працюють коректно, що критично важливо для забезпечення достовірності результатів.

Всі ці методи працюють разом, створюючи багаторівневу систему забезпечення достовірності, де кожен рівень захищає від різних типів помилок та проблем. Ізольованість середовища забезпечує відтворюваність, логування та моніторинг дозволяють виявляти проблеми, валідація запобігає помилкам через некоректні дані, транзакції забезпечують цілісність даних, rate limiting захищає від перевантаження, версіонування забезпечує стабільність, а health checks гарантують коректний стан системи.

2.3.6 Критерії оцінки достовірності результатів експериментів

Достовірність результатів експериментальних досліджень оцінювалася за комплексом критеріїв, кожен з яких відображає різні аспекти надійності та валідності отриманих результатів. Ці критерії дозволяють об'єктивно оцінити, наскільки результати експериментів можуть бути використані для висновків про ефективність запропонованих методів трансформації процесних моделей у модульну архітектуру.

Таблиця 2.4

Критерії оцінки достовірності результатів експериментів

№	Критерій	Опис	Спосіб перевірки
1	Відтворюваність	Можливість отримати однакові результати при повторному виконанні експериментів за однакових умов	Фіксація версій компонентів, використання Docker, детальна документація процедур
2	Стабільність	Здатність системи підтримувати коректну роботу протягом тривалого періоду та під різними навантаженнями	Моніторинг логів помилок, метрик продуктивності, використання ресурсів, час відповіді
3	Цілісність даних	Збереження коректності та консистентності даних після виконання операцій	Транзакції БД, валідація стану після операцій, перевірка зв'язків між сутностями
4	Безпека	Захист від несанкціонованого доступу, маніпуляцій даними, витоку конфіденційної інформації	Аутентифікація та авторизація, валідація підписів, обмеження прав доступу, шифрування
5	Документованість	Наявність повної та актуальної документації API, архітектури, процедур розгортання	OpenAPI специфікація, GraphQL схема, коментарі в коді, описи архітектури
6	Продуктивність	Здатність системи обробляти запити з прийнятним часом відповіді та ефективно використовувати ресурси	Метрики часу відповіді, пропускна здатність, використання CPU/пам'яті, ефективність кешування
7	Масштабованість	Здатність системи обробляти зростаюче навантаження без значних змін у архітектурі	Тестування під різними навантаженнями, моніторинг поведінки, оцінка горизонтального масштабування

Перший критерій достовірності - це відтворюваність результатів, тобто можливість отримати однакові або дуже близькі результати при повторному виконанні експериментів за однакових умов. Відтворюваність забезпечується через фіксацію версій всіх компонентів системи в конфігураційних файлах, використання контейнеризації для забезпечення однакових умов виконання, та детальне документування процедур експериментів. Якщо експеримент можна відтворити з однаковими результатами на різних системах та в різний час, це підвищує довіру до отриманих результатів та дозволяє іншим дослідникам перевірити висновки незалежно.

Другий критерій достовірності - це стабільність системи, тобто здатність системи підтримувати коректну роботу протягом тривалого періоду та під різними навантаженнями без деградації продуктивності або появи помилок. Стабільність перевіряється через моніторинг логів помилок, метрик продуктивності, використання ресурсів та часу відповіді системи на запити. Якщо система демонструє стабільну роботу без несподіваних збоїв, помилок або деградації продуктивності, це вказує на надійність архітектурних рішень та коректність реалізації модульної архітектури.

Третій критерій достовірності - це цілісність даних, тобто збереження коректності та консистентності даних після виконання операцій, особливо при асинхронних процесах, таких як обробка платежів, відправка нотифікацій або оновлення пов'язаних даних. Цілісність даних забезпечується через використання транзакцій бази даних, валідацію стану після операцій, перевірку зв'язків між сутностями та обробку помилок з відкатом змін у разі невдачі. Якщо дані завжди знаходяться в консистентному стані після виконання операцій, це підтверджує коректність реалізації бізнес-логіки та забезпечення цілісності на рівні модульної архітектури.

Четвертий критерій достовірності - це безпека системи, тобто захист від несанкціонованого доступу, маніпуляцій даними, витоку конфіденційної інформації та інших загроз безпеці. Безпека реалізується через систему аутентифікації та авторизації, валідацію підписів у платежах, обмеження прав

доступу на основі ролей користувачів, шифрування чутливих даних та захист від типових вразливостей, таких як SQL-ін'єкції, XSS атаки або CSRF атаки. Якщо система захищена від основних загроз безпеки та дотримується принципів безпечної розробки, це підвищує довіру до системи та дозволяє використовувати її в реальних умовах.

П'ятий критерій достовірності - це документованість системи, тобто наявність повної та актуальної документації API, архітектури, процедур розгортання та використання, що дозволяє незалежно перевірити коректність реалізації та повторити експерименти. Документація забезпечується через OpenAPI специфікацію для REST API, GraphQL схему для GraphQL API, коментарі в коді, описи архітектури та інструкції з розгортання. Якщо система повністю документована та документація завжди синхронізована з реалізацією, це дозволяє іншим дослідникам зрозуміти систему, перевірити висновки та використати систему для власних досліджень.

Шостий критерій достовірності - це продуктивність системи, тобто здатність системи обробляти запити з прийнятним часом відповіді та ефективно використовувати ресурси системи. Продуктивність оцінюється через метрики часу відповіді на запити, пропускної здатності системи, використання процесора, пам'яті та мережевих ресурсів, а також ефективність використання кешування та оптимізації запитів до бази даних. Якщо система демонструє прийнятну продуктивність для цільових навантажень, це підтверджує, що модульна архітектура не тільки забезпечує коректність функціонування, але й ефективність використання ресурсів.

Сьомий критерій достовірності - це масштабованість системи, тобто здатність системи обробляти зростаюче навантаження без необхідності значних змін у архітектурі або реалізації. Масштабованість перевіряється через тестування системи під різними навантаженнями, моніторинг поведінки системи при збільшенні кількості користувачів або обсягу даних, та оцінку можливостей горизонтального масштабування через додавання нових екземплярів компонентів. Якщо система демонструє хорошу масштабованість, це

підтверджує, що модульна архітектура забезпечує гнучкість та можливість адаптації до змінних вимог.

Всі ці критерії створюють комплексну оцінку достовірності результатів, де кожен критерій відображає важливий аспект надійності системи. Відтворюваність забезпечує можливість перевірки результатів, стабільність гарантує надійність у довгостроковій перспективі, цілісність даних забезпечує коректність операцій, безпека захищає від загроз, документованість дозволяє перевірити та використати систему, продуктивність забезпечує практичну придатність, а масштабованість гарантує можливість розвитку системи. Разом ці критерії забезпечують об'єктивну оцінку ефективності запропонованих методів трансформації процесних моделей у модульну архітектуру та дозволяють зробити висновки про практичну придатність цих методів для вирішення реальних задач. Разом ці критерії забезпечують об'єктивну оцінку ефективності запропонованих методів трансформації процесних моделей у модульну архітектуру та дозволяють зробити висновки про практичну придатність цих методів для вирішення реальних задач.

Таким чином, експериментальна платформа забезпечує надійне середовище для перевірки ефективності запропонованих методів трансформації процесних моделей у модульну архітектуру, з дотриманням принципів наукової об'єктивності, відтворюваності результатів та комплексної оцінки достовірності. Використання сучасних технологій, таких як Directus, MySQL, MongoDB, Redis, GraphQL та Docker, дозволяє створити реалістичне середовище, яке відображає реальні умови використання модульної архітектури, що підвищує практичну цінність отриманих результатів та дозволяє зробити висновки про застосовність запропонованих методів у реальних проєктах.

3 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

3.1. Аналіз результатів реалізації модифікованого процесно-орієнтованого підходу

Реалізація модифікованого процесно-орієнтованого (гібридного) підходу на експериментальній платформі дозволила отримати емпіричні дані щодо ефективності запропонованих методів трансформації процесних моделей у модульну архітектуру та перевірити практичну придатність теоретично обґрунтованих рішень. Модифікований підхід, як було визначено у розділі 2, поєднує орієнтацію на дані, урахування організаційної структури та відкладену інтеграцію окремих модулів, усуваючи таким чином системні прогалини традиційних парадигм проєктування, виявлені у розділі 1. Аналіз результатів реалізації здійснюється через призму комплексних критеріїв достовірності, встановлених у підрозділі 2.3.6, та демонструє кількісні та якісні переваги запропонованого підходу.

Модифікація традиційного процесно-орієнтованого підходу полягала в інтеграції трьох ключових елементів, які доповнюють базову процесну формалізацію. По-перше, додавання орієнтації на дані забезпечило формалізацію data-контрактів через схеми бази даних, що дозволило автоматично генерувати API-контракти на основі структури даних та забезпечити синхронізацію між структурою даних та програмними інтерфейсами. По-друге, урахування організаційної структури через систему ролей та прав доступу забезпечило інтеграцію організаційної семантики у технічні артефакти, де автоматичне застосування прав доступу гарантує, що організаційна структура враховується на всіх рівнях системи. По-третє, відкладена інтеграція з дисциплінованою контрактною специфікацією дозволила паралелізувати розробку модулів при збереженні контролю над сумісністю через централізований реєстр контрактів та автоматизоване контрактне тестування.

Кількісна оцінка ефективності модифікованого підходу продемонструвала значні покращення порівняно з традиційним процесно-орієнтованим підходом у ключових метриках розробки та експлуатації системи. Час розробки нових функцій (time-to-feature) зменшився з 30 днів до 18 днів, що становить покращення на 40%, що обумовлено паралелізацією розробки модулів через відкладену інтеграцію та зменшенням координаційних витрат завдяки формалізованим контрактам. Затримка API на рівні 95-го перцентилю (p95 latency) знизилася з 320 мілісекунд до 220 мілісекунд, що становить покращення на 31.3%, що досягнуто завдяки оптимізації запитів через агрегацію на рівні сервера, ефективному кешуванню та урахуванню організаційної структури для локалізації обробки запитів.

Пропускна здатність системи (throughput) зросла з 150 запитів на секунду до 270 запитів на секунду, що становить покращення на 80%, що обумовлено ефективним використанням кешування, оптимізацією запитів до бази даних через формалізовані контракти та зменшенням накладних витрат на інтеграцію модулів. Частка влучень кешу (cache hit ratio) зросла з 45% до 75%, що становить покращення на 66.7 процентних пунктів, що досягнуто завдяки централізованій політиці кешування, формалізованій у метаданих контрактів, та ефективній інвалідації кешу на основі змін у структури даних, визначених у data-контрактах.

Затримка синхронізації даних (data sync latency) зменшилася з 12 секунд до 3 секунд, що становить покращення на 75%, що досягнуто завдяки формалізації data-контрактів, яка забезпечила автоматичну синхронізацію між структурою даних та API-контрактами, зменшуючи необхідність ручних операцій та виявлення несумісностей на пізніх етапах розробки.

Порівняльний аналіз продуктивності модифікованого гібридного підходу з альтернативними підходами продемонстрував переваги запропонованого рішення у широкому діапазоні навантажень. При низьких навантаженнях (до 100 запитів на секунду) всі підходи демонструють подібну продуктивність з затримкою менше 250 мілісекунд. Однак при зростанні навантаження переваги гібридного підходу стають все більш очевидними: при 350 запитах на секунду

гібридний підхід забезпечує затримку на рівні 95-го перцентилю приблизно 1400 мілісекунд, що значно краще порівняно з процесно-орієнтованим підходом (близько 4000 мілісекунд), організаційно-орієнтованим підходом (близько 6500 мілісекунд) та підходом з відкладеною інтеграцією (близько 6000 мілісекунд). Орієнтований на дані підхід демонструє другий за ефективністю результат (близько 2500 мілісекунд), але все ще поступається гібридному підходу, особливо при навантаженнях вище 250 запитів на секунду.

Переваги гібридного підходу при високих навантаженнях обумовлені поєднанням ефективності data-орієнтованого підходу у забезпеченні швидкого доступу до даних, формалізації процесів для передбачуваності поведінки системи, урахування організаційної структури для локалізації обробки та дисциплінованої відкладеної інтеграції для мінімізації накладних витрат. Це підтверджує, що модифікований підхід не лише усуває прогалини окремих парадигм, але й забезпечує синергетичний ефект від їх поєднання.

Аналіз якісних аспектів модифікованого підходу демонструє, що формалізація data-контрактів через багаторівневу систему (схеми бази даних, OpenAPI специфікацію, GraphQL схему та систему прав доступу) забезпечує повну прозорість структури даних та способів їх обміну. Автоматична генерація REST API на основі структури таблиць MySQL забезпечує синхронізацію між структурою даних та API-контрактами без додаткової роботи, що значно зменшує ризик невідповідності та забезпечує консистентність системи. Графічна схема GraphQL, автоматично генерована на основі структури колекцій, забезпечує строгу типізацію та автоматичну валідацію запитів, слугуючи як жива документація API, що значно спрощує розробку клієнтських додатків та зменшує час на інтеграцію.

Урахування організаційної структури через систему ролей та прав доступу забезпечило інтеграцію організаційної семантики у технічні артефакти, де автоматичне застосування прав доступу через систему accountability гарантує, що організаційна структура враховується на всіх рівнях системи без необхідності додаткової реалізації логіки авторизації в кожному модулі. Це значно спрощує

розробку, зменшує ризик помилок безпеки та забезпечує консистентність безпеки на всіх рівнях системи.

Відкладена інтеграція з дисциплінованою контрактною специфікацією дозволила паралелізувати розробку модулів при збереженні контролю над сумісністю через централізований реєстр контрактів та автоматизоване контрактне тестування. Це забезпечило прискорення локальної розробки та гнучкість щодо внутрішньої еволюції модулів, одночасно мінімізуючи ризики інтеграційного боргу через раннє виявлення несумісностей через автоматизовані перевірки.

Аналіз безпеки та надійності системи показав, що інтеграція трьох елементів модифікованого підходу забезпечує комплексний захист на різних рівнях: формалізація data-контрактів забезпечує цілісність даних на рівні бази, урахування організаційної структури забезпечує правильне розподілення прав доступу, а дисциплінована інтеграція забезпечує узгодженість безпеки між модулями. Використання транзакцій бази даних для критичних операцій гарантує атомарність операцій, забезпечуючи, що дані завжди знаходяться в консистентному стані, навіть при часткових збоях.

Комплексне логування всіх операцій системи, формалізоване у процесних моделях та data-контрактах, забезпечує ефективне відстеження виконання операцій та виявлення проблем. Детальне логування з інформацією про час виконання, параметри запитів, результати виконання та деталі помилок дозволяє швидко виявляти та вирішувати проблеми, що критично важливо для забезпечення надійності системи в експлуатації та відповідності нормативним вимогам до аудиту.

Аналіз результату реалізації дозволяє зробити висновок про досягнення основних цілей дослідження. Модифікований процесно-орієнтований підхід успішно продемонстрував свою ефективність для трансформації процесних моделей у модульну архітектуру, забезпечуючи значні покращення у ключових метриках розробки та експлуатації системи. Поєднання орієнтації на дані, урахування організаційної структури та відкладеної інтеграції забезпечило

синергетичний ефект, який усунув системні прогалини традиційних парадигм проєктування та забезпечив високу продуктивність, масштабованість, безпеку та надійність системи. Результати аналізу підтверджують, що модифікований процесно-орієнтований підхід є ефективним інструментом для трансформації бізнес-процесів у технічну архітектуру, забезпечуючи при цьому практичну придатність для вирішення реальних задач у сфері управління політичною партією.

3.2. Взаємозв'язок виконаних завдань із досягненням мети дослідження

Вступом до роботи було визначено мету дослідження – удосконалення методу процесу побудови архітектури інформаційної системи для сфери управління політичною партією за рахунок застосування модифікованого процесно-орієнтованого (гібридного) підходу та модульних архітектурних принципів. Досягнення цієї мети забезпечувалося послідовним виконанням комплексу завдань, кожне з яких було спрямоване на розв’язання окремих аспектів проблеми та формувало необхідні передумови для отримання кінцевого результату. У даному підрозділі узагальнено, яким чином виконані завдання в сукупності забезпечили досягнення поставленої мети, а також показано логічний зв’язок між теоретичними, методологічними та прикладними результатами.

Перше завдання полягало в аналізі сучасних підходів та практик побудови архітектури інформаційних систем із фокусом на управлінські процеси організацій. У розділі 1 було систематизовано чотири базові парадигми – орієнтовану на дані, процесно-орієнтовану, орієнтовану на організаційну структуру та підхід з відкладеною інтеграцією. Порівняльний аналіз виявив як їх сильні сторони (забезпечення цілісності даних, прозорість процесів, чіткий розподіл повноважень, операційна гнучкість), так і системні прогалини на стиках між рівнями даних, процесів, організаційних ролей та інтеграційних контрактів. Саме ідентифікація цих прогалин (відсутність прозорого зв’язку між процесними моделями та API-контрактами, недостатня інтеграція організаційної семантики в

технічні артефакти, накопичення інтеграційного боргу) сформувала обґрунтування потреби в гібридному підході і задала рамки для подальшого удосконалення методу. Таким чином, перше завдання забезпечило концептуальну базу, на якій було сформульовано вимоги до нового методу, а також критерії, за якими надалі оцінювалася його ефективність.

Виконання першого завдання безпосередньо пов’язане із задекларованою у вступі науковою новизною. Формалізований порівняльний аналіз дав змогу не лише описати існуючі підходи, але й чітко окреслити ті аспекти, які залишалися поза їхнім фокусом, – інтеграцію процесної, даної та організаційної перспектив у єдиній архітектурній методиці. Саме ці виявлені прогалини зумовили необхідність розроблення модифікованого процесно-орієнтованого підходу, орієнтованого на предметну область управління політичною партією.

Друге завдання стосувалося дослідження особливостей процесно-орієнтованого моделювання (BPMN) і його застосування у контексті політичних організацій. У ході роботи було показано, що класичні процесні моделі добре відображають послідовність дій, точки прийняття рішень і взаємодію ролей, але за відсутності формалізованих правил трансформації в програмні контракти та структури даних породжують розрив між описом і реалізацією. Аналіз специфіки політичних партій (жорсткі нормативні вимоги, потреба в аудиті, значна кількість ролей і організаційних рівнів) продемонстрував, що традиційного процесного підходу недостатньо без інтеграції з шаром даних та організаційною моделлю. Результати виконання другого завдання стали основою для формулювання вимог до модифікованого підходу: необхідності включення у процесні моделі явних вимог до якості даних, прав доступу, політик зберігання та механізмів відстеження станів екземплярів процесів.

Таким чином, друге завдання забезпечило методологічне підґрунтя для інтеграції BPMN-моделей із архітектурними артефактами. Воно конкретизувало, які саме елементи процесної моделі мають бути відображені в data- та API-контрактах, та визначило перелік обмежень, що випливають із законодавчих та внутрішньопартійних регламентів. Це безпосередньо пов’язано з метою

дослідження, оскільки дозволило сформувати вимоги до такого варіанту архітектури, що одночасно забезпечує процесну керованість, юридичну коректність і можливість технічної реалізації.

Третє завдання було спрямоване на розроблення модифікованого (гібридного) процесно-орієнтованого підходу, що поєднує орієнтацію на дані, урахування організаційної структури та відкладену інтеграцію модулів. У розділах 1 та 2 було сформульовано сам підхід і методи трансформації процесних моделей у модульну архітектуру, що включають: стандартизоване описання процесів у машинночитному форматі; виділення модулів за критеріями зв'язаності та відповідальності; побудову data-контрактів із чіткими схемами сутностей, політиками ретенції та безпеки; формування API-контрактів, подій і компенсаційних сценаріїв; централізований реєстр контрактів і політики версіонування. Виконання третього завдання безпосередньо реалізувало концептуальну частину мети дослідження, оскільки результатом стала цілісна методика, яка усуває виявлені на попередніх етапах розриви між окремими архітектурними парадигмами.

У межах цього завдання було також конкретизовано механізми підтримки прийняття рішень – через формалізовані критерії оцінювання ефективності процесів, вимоги до прозорості даних та відтворюваності станів. Це дозволило перейти від суто описового рівня до конструктивного: з'явилася можливість не лише моделювати процеси, а й безпосередньо пов'язувати їх з архітектурними рішеннями, які забезпечують необхідні нефункціональні властивості (масштабованість, надійність, відповідність нормативам).

Четверте завдання передбачало реалізацію прототипу архітектури та його тестування. У підрозділах 2.3.1–2.3.3 описано побудову експериментальної платформи на базі Directus, MySQL, MongoDB, Redis, GraphQL та Docker, що відтворює ключові елементи запропонованого підходу: централізований шар керування даними, формалізацію data-контрактів через схеми бази даних і GraphQL-схему, використання кастомних extensions як модулів з чіткими інтерфейсами, застосування кешування та механізмів спостережуваності.

Розроблені модулі – мультипошук, контроль користувачів із двофакторною аутентифікацією, платіжний модуль та модулі експорту даних – реалізують типові процеси предметної галузі (реєстрація, аутентифікація, фінансові операції, звітність) із дотриманням принципів гібридного підходу.

Таким чином, четверте завдання забезпечило перехід від методологічних побудов до практичної перевірки. Реалізація прототипу дозволила з одного боку продемонструвати життєздатність розробленої методики, а з іншого – отримати емпіричні дані щодо її впливу на ключові показники системи. Важливо, що при побудові прототипу було дотримано вимог до модульності та можливості подальшого розширення, що відповідає як науковим, так і практичним цілям роботи.

П’яте завдання стосувалося оцінки ефективності запропонованого підходу за допомогою критеріїв продуктивності, масштабованості, безпеки, якості даних та зниження інтеграційних витрат. У підрозділі 2.3.5 було обґрунтовано методологію тестування, у 2.3.6 – сформульовано критерії достовірності, а в підрозділі 3.1 – проведено узагальнення кількісних та якісних результатів. Кількісні показники (скорочення time-to-feature на 40%, зниження p95 latency на 31.3%, зростання пропускної здатності на 80%, підвищення частки влучень кешу та зменшення затримки синхронізації даних у 4 рази) свідчать про те, що гібридний підхід забезпечує суттєві покращення порівняно з окремими парадигмами. Порівняння поведінки системи під навантаженням продемонструвало, що запропонований підхід краще масштабується та зберігає прийнятний рівень латентності при зростанні інтенсивності запитів, що є критичним для інформаційних систем політичних партій у пікові періоди активності.

Окремого розгляду потребує виконання завдання щодо формулювання методичних рекомендацій для практичного впровадження. На основі результатів моделювання, реалізації прототипу та аналізу показників у роботі сформовано рекомендації щодо: вибору технологічного стеку для подібних систем; організації централізованого реєстру контрактів і механізмів їх версіонування;

побудови політик кешування та ретенції даних; впровадження процедур контрактного тестування в CI/CD-пайплайни; налаштування моніторингу та логування з урахуванням вимог до аудиту. Ці рекомендації виступають містком між експериментально перевіреним підходом і реальними впровадженнями в організаціях, тим самим посилюючи практичне значення отриманих результатів.

Узагальнюючи, можна констатувати, що всі поставлені у вступі завдання виконані та утворюють послідовний ланцюг від аналізу існуючих підходів і формулювання гібридного методу до його реалізації й експериментальної валідації. Кожне завдання вносило внесок у досягнення мети: перші два – через формування концептуальних та методологічних передумов, третє – через розробку власне удосконаленого підходу, четверте і п’яте – через його практичну перевірку, кількісну оцінку ефективності та формування рекомендацій щодо впровадження. Таким чином, мета дослідження – удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі модифікованого процесно-орієнтованого підходу – досягнута в повному обсязі, а отримані результати мають як теоретичну, так і прикладну цінність.

3.3. Оцінка достовірності результатів і обмежень проведеного дослідження

Оцінка достовірності отриманих результатів є невід’ємною складовою наукового дослідження, особливо у випадку, коли його висновки претендують на практичне застосування в умовах реальних організаційних та нормативних обмежень. У даному підрозділі узагальнено, наскільки результати експериментальної валідації модифікованого процесно-орієнтованого підходу можуть вважатися надійними, а також окреслено основні обмеження дослідження, які необхідно враховувати при інтерпретації висновків і плануванні подальших робіт. Оцінювання достовірності ґрунтується на критеріях, сформульованих у підрозділі 2.3.6, а саме: відтворюваність,

стабільність, цілісність даних, безпека, документованість, продуктивність і масштабованість.

З позицій відтворюваності результати мають високий ступінь надійності. Використання контейнеризації (Docker) та фіксація версій усіх компонентів у конфігураційних файлах дозволяють відтворити експериментальне середовище на будь-якій сумісній платформі з мінімальними відмінностями. Процедури розгортання, запуску тестів і збору телеметрії формалізовані, описані у вигляді послідовних кроків і можуть бути багаторазово повторені. Повторні запуски експериментів за однакових параметрів навантаження демонстрували близькі значення основних метрик (латентність, пропускна здатність, частка влучень кешу), що свідчить про відсутність істотного впливу випадкових факторів і підтверджує стійкість отриманих оцінок.

Не менш важливим аспектом достовірності є стабільність системи під час проведення випробувань. Моніторинг логів помилок, метрик використання процесорних і пам’ятних ресурсів, а також часу відповіді показав відсутність неконтрольованих збоїв або деградації продуктивності протягом тривалих періодів навантажувального тестування. Використання механізмів перевірки стану (health checks) і автоматичного відновлення сервісів забезпечило, що вимірювання виконувалися лише за умов коректної роботи всіх компонентів. Зафіксовані поодинокі інциденти мали переважно конфігураційний характер і були усунуті до проведення основних серій експериментів, що дозволяє вважати стабільність платформи достатньою для надійного вимірювання характеристик.

Складовою достовірності, яка має принципове значення для інформаційних систем управління політичною партією, є цілісність даних. У дослідженні для критичних операцій застосовувалися транзакції бази даних, які охоплюють кілька взаємопов’язаних сутностей (наприклад, створення підписок разом із фіксацією платіжних подій). Після завершення транзакцій виконувалася валідація стану та перевірка зв’язків між сутностями, що не виявила неконсистентних записів у тестових сценаріях. Додатково було запроваджено валідацію вхідних даних на рівні endpoint’ів і використання формалізованих data-

контрактів, що зменшило ймовірність потрапляння у систему некоректних або неповних даних. У сукупності це дає підстави стверджувати, що результати щодо продуктивності, масштабованості та надійності не є артефактом прихованих порушень цілісності даних.

Критерій безпеки в експериментальній платформі реалізовано на рівні, достатньому для коректної оцінки поведінки запропонованого підходу в умовах типових загроз. Система автентифікації та авторизації Directus, розмежування прав доступу за ролями, використання одноразових кодів у модулі двофакторної аутентифікації, валідація підписів у платіжному модулі та реєстрація критичних подій у системі нотифікацій суттєво знижують ризики несанкціонованих операцій і маніпуляцій даними в межах експериментів. Водночас повноцінний аудит безпеки з урахуванням промислових вимог до шифрування, керування ключами, захисту від складних атак і відповідності галузевим стандартам не проводився, оскільки виходить за рамки задекларованої мети дослідження. Це означає, що з точки зору експериментальної оцінки архітектурних рішень базовий рівень безпеки є достатнім, але для промислового впровадження потрібні додаткові заходи.

Важливим чинником достовірності є документованість архітектури й інтерфейсів. Наявність OpenAPI-специфікації для кастомних endpoint’ів, автоматично згенерованої GraphQL-схеми, а також текстових описів архітектури, процесів розгортання та методології тестування забезпечує необхідну прозорість для незалежної перевірки результатів. Така документованість дозволяє іншим фахівцям відтворити експериментальне середовище, відтворити ключові сценарії навантаження та проаналізувати отримані метрики, що є важливою передумовою зовнішньої валідації результатів і зменшує ризики інтерпретаційних помилок.

Окремо слід відзначити продуктивність і масштабованість як критерії, що одночасно виконують роль показників ефективності й індикаторів достовірності. Стабільність вимірювань латентності, пропускної здатності та частки влучень кешу при багаторазовому повторенні експериментів засвідчує, що система не

перебувала в режимі випадкових коливань або деградації. Отримані криві залежності часу відповіді від навантаження мають очікуваний характер: поступове зростання латентності зі збільшенням кількості запитів і чітко виражені відмінності між гібридним підходом і альтернативними стратегіями. Це свідчить про коректність налаштування експериментального стенда та адекватність обраних метрик для оцінювання поведінки системи.

Разом із тим результати дослідження мають низку обмежень, які слід враховувати при їх інтерпретації та спробах перенесення на інші контексти. Одне з ключових обмежень пов’язане з масштабом і характером експериментального середовища. Прототип розгорнуто в контрольованих умовах із фіксованою конфігурацією апаратних ресурсів, що не повністю відображає різноманіття можливих інфраструктурних налаштувань у реальних організаціях. Тому отримані показники продуктивності доцільно інтерпретувати передусім як відносні – у термінах порівняння гібридного підходу з альтернативами, а не як універсальні абсолютні значення.

Ще одним важливим обмеженням є характер використаних даних. Для експериментів застосовувалися синтетичні та анонімізовані набори, статистично наближені до очікуваних розподілів у предметній області, але такі, що неминуче спрощують картину реальної експлуатації. У промислових системах політичних партій значну роль відіграють історичні аномалії, «брудні» дані, неповні або суперечливі записи, які можуть впливати як на якість даних, так і на поведінку процесів. Відповідно, результати щодо ефективності механізмів валідації, очищення та узгодження даних слід розглядати як базову оцінку в контрольованих умовах, яку доцільно уточнювати під час пілотних впроваджень на реальних масивах.

Додаткове обмеження зумовлене предметною спрямованістю дослідження. Методика розроблялася та перевірялася в контексті управління політичною партією, з урахуванням притаманних цій сфері нормативних вимог, організаційної структури та типових бізнес-процесів. Хоча гібридний процесно-орієнтований підхід концептуально придатний для застосування в інших галузях,

його перенесення на домени зі специфічними вимогами (банківська справа, охорона здоров’я, промислові IoT-рішення тощо) потребує додаткової адаптації: перегляду набору процесів, зміни політик безпеки й ретенції даних, а також повторної експериментальної валідації.

Слід також враховувати обмеження, пов’язані з реалізацією зовнішніх інтеграцій. У прототипі інтеграції з платіжними та комунікаційними сервісами реалізовано таким чином, щоб відтворити ключові патерни взаємодії (асинхронні зворотні виклики, перевірка підписів, обробка статусів), але не всі можливі сценарії відмов, деградації сервісів-постачальників або мережевих збоїв моделювалися в повному обсязі. Тому оцінка стійкості системи до комплексних зовнішніх збоїв є частковою і потребує розширення в рамках подальших досліджень та промислових пілотів.

На достовірність результатів певною мірою впливає обраний технологічний стек. Використання конкретних рішень (Directus, MySQL, MongoDB, Redis, GraphQL, Docker) дозволило продемонструвати реалізацію гібридного підходу на сучасних інструментах, однак окремі кількісні характеристики можуть відрізнятися при застосуванні інших платформ і продуктів з відмінними внутрішніми оптимізаціями. Це обмеження не нівелює загальні закономірності, виявлені у ході дослідження (зокрема, вплив формалізації контрактів та процесно-даних моделей на стабільність і масштабованість), але вимагає обережності при перенесенні конкретних числових оцінок на інші технологічні екосистеми.

ВИСНОВКИ

У результаті проведеного дослідження та розробки досягнуто поставленої мети – удосконалено метод побудови архітектури інформаційної системи для сфери управління політичною партією на основі модифікованого процесно-орієнтованого (гібридного) підходу та модульних архітектурних принципів, оскільки на підставі теоретичного аналізу, формалізації методики та експериментальної апробації прототипу продемонстровано, що запропонований підхід забезпечує узгодженість між процесними моделями, даними, організаційними ролями й інтеграційними контрактами та покращує ключові показники роботи системи.

Встановлено, що існуючі архітектурні парадигми (орієнтований на дані, процесно-орієнтований, організаційно-орієнтований та підхід з відкладеною інтеграцією) у ізольованому застосуванні не забезпечують повної відповідності вимогам до інформаційних систем управління політичною партією, оскільки аналіз літератури й практичних рішень у розділі 1 показав наявність системних прогалин на стиках між рівнями даних, процесів, організаційних ролей і контрактів (відсутність єдиного контрактного шару, складність керованої еволюції схем, накопичення інтеграційного боргу), що й обумовило необхідність гібридизації підходів.

Обґрунтовано доцільність використання процесно-орієнтованого моделювання (BPMN) як базового інструмента опису діяльності політичної партії, оскільки у розділі 1 продемонстровано, що типові партійні процеси (реєстрація членів, облік пожертв, підготовка звітності, ухвалення управлінських рішень) мають чітку послідовність кроків, ролі, умови переходів і контрольні точки, а BPMN дозволяє формалізувати ці елементи разом із вимогами до аудиту, відповідальності та трасування рішень, створюючи основу для подальшої автоматизації та контролю.

Запропоновано модифікований (гібридний) процесно-орієнтований підхід до побудови архітектури інформаційних систем управління політичною партією, який інтегрує процесну, дану й організаційну перспективи, оскільки на підставі результатів розділів 1 і 2 показано, що поєднання орієнтації на дані, урахування організаційної структури та дисциплінованої відкладеної інтеграції дозволяє усунути виявлені раніше розриви між процесними моделями, data-контрактами, API-контрактами та ролями, забезпечуючи цілісну архітектурну методику для динамічних і регульованих предметних галузей.

Розроблено методику трансформації процесних моделей у модульну архітектуру інформаційної системи, що включає стандартизоване машинночитне описання процесів, виділення модулів за критеріями зв'язаності та відповідальності, формування канонічних схем сутностей і контрактів інтерфейсів та подій, оскільки в підрозділі 2.2 доведено, що така послідовність кроків дозволяє систематично переходити від опису бізнес-процесів до набору узгоджених модулів із чіткими зонами відповідальності, політиками доступу, правилами версіонування й механізмами компенсації, зменшуючи інтеграційні ризики.

Створено експериментальну платформу на базі Directus, MySQL, MongoDB, Redis, GraphQL та Docker з набором кастомних модулів (мультипошук, контроль користувачів, платіжний модуль, модулі експорту), що реалізують ключові процеси діяльності політичної партії, оскільки у підрозділі 2.3 показано, що така конфігурація відтворює типовий технологічний ландшафт сучасних інформаційних систем, забезпечує формалізацію data-контрактів, API-контрактів і політик доступу та створює реалістичний стенд для апробації й вимірювання ефективності запропонованого підходу.

Емпірично підтверджено ефективність запропонованого модифікованого процесно-орієнтованого підходу порівняно з традиційним процесним та іншими розглянутими підходами, оскільки результати навантажувальних і функціональних експериментів, наведені в розділі 3, демонструють скорочення часу виведення нових функцій (time-to-feature), зниження p95-латентності

запитів, зростання пропускної здатності системи й частки влучень кешу та зменшення затримки синхронізації даних, що безпосередньо пов’язано з формалізацією контрактів, централізацією даних, використанням кешування й гібридної схеми зберігання.

Показано, що застосування гібридного процесно-орієнтованого підходу підвищує прозорість, керованість та відповідність інформаційної системи організаційним і нормативним вимогам політичної партії, оскільки на підставі аналізу в розділах 2 і 3 встановлено, що єдиний реєстр контрактів і процесних моделей спрощує аудит рішень, інтеграція ролей і прав доступу забезпечує узгодженість між формальними повноваженнями та технічними можливостями, а централізоване логування й метрики дають можливість систематично оцінювати ефективність процесів і вчасно виявляти відхилення.

Оцінено достовірність отриманих результатів та окреслено межі їх застосовності, оскільки в підрозділах 2.3.6 і 3.3 показано, що за критеріями відтворюваності, стабільності, цілісності даних, безпеки, документованості, продуктивності й масштабованості експериментальна платформа забезпечує надійну емпіричну базу для висновків, але водночас підкреслено, що специфіка тестових даних, масштаб стенда, предметна зумовленість і спрощений характер окремих інтеграцій вимагають додаткових перевірок при перенесенні результатів на інші домени й інфраструктури.

Сформульовано практичні рекомендації щодо впровадження модифікованого процесно-орієнтованого підходу в інформаційні системи управління політичною партією, оскільки на підставі результатів у розділах 2 і 3 запропоновано використовувати централізований реєстр data- та API-контрактів, впроваджувати контрактне тестування в CI/CD-процеси, застосовувати гібридну схему зберігання даних (реляційна та документоорієнтована СУБД), будувати узгоджені політики кешування й ретенції даних та забезпечувати розгорнутий моніторинг і логування бізнес-подій, що створює методичну основу для практичних проєктів у сфері цифровізації політичних партій.

Результати дослідження апробовано та опубліковано у наступних статті та тезах:

1. Могильник М.Р., Садовенко В.С. Дослідження архітектурних підходів до побудови інформаційних систем управління політичною партією III Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу», 18 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. - С. 178.

2. Могильник М.Р., Садовенко В.С. Визначення вимог до удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі процесно-орієнтованого підходу Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. - подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1 Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach / R. S. Pressman, B. R. Maxim. — 8th ed. — New York : McGraw-Hill, 2015. — 976 p.

2 Войтович І. С. Інформаційні системи і технології в управлінні організаціями : навч. посіб. / І. С. Войтович. — Львів : Вид-во Львів. політехніки, 2018. — 312 с.

3 Гавриленко О. В. Архітектура програмного забезпечення : навч. посіб. / О. В. Гавриленко. — Київ : КНУ ім. Т. Шевченка, 2020. — 268 с.

4 Rozanski N., Woods E. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives / N. Rozanski, E. Woods. — 2nd ed. — Boston : Addison-Wesley, 2012. — 704 p.

5 Козлов В. В. Інформаційні технології в публічному управлінні : монографія / В. В. Козлов. — Київ : НАДУ, 2019. — 340 с.

6 Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. — Boston : Addison-Wesley, 2004. — 560 p.

7 Sommerville I. Software Engineering / I. Sommerville. — 10th ed. — Harlow : Pearson, 2015. — 792 p.

8 Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. — Boston : Addison-Wesley, 2003. — 560 p.

9 Newman S. Building Microservices: Designing Fine-Grained Systems / S. Newman. — 2nd ed. — Beijing ; Boston : O’Reilly Media, 2021. — 428 p.

10 Каменєв А. О. Проєктування інформаційних систем : навч. посіб. / А. О. Каменєв. — Харків : ХНУРЕ, 2017. — 256 с.

11 Dumas M., La Rosa M., Mendling J., Reijers H. A. Fundamentals of Business Process Management / M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers. — 2nd ed. — Cham : Springer, 2018. — 527 p.

12 Kleppmann M. Designing Data-Intensive Applications / M. Kleppmann. — 1st ed. — Beijing ; Boston : O'Reilly Media, 2017. — 616 p.

13 Sadalage P. J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence / P. J. Sadalage, M. Fowler. — Boston : Addison-Wesley, 2013. — 192 p.

14 Bass L., Clements P., Kazman R. Software Architecture in Practice / L. Bass, P. Clements, R. Kazman. — 4th ed. — Boston : Addison-Wesley, 2021. — 640 p.

15 Lewis J., Fowler M. Microservices: a definition of this new architectural term [Електронний ресурс] / J. Lewis, M. Fowler. — Режим доступу : https://martinfowler.com/articles/microservices.html

16 Business Process Model and Notation (BPMN) Version 2.0.2 [Електронний ресурс]. — Object Management Group, 2014. — Режим доступу : https://www.omg.org/spec/BPMN/2.0.2

17 The GraphQL Specification [Електронний ресурс]. — GraphQL Foundation, 2021. — Режим доступу : https://spec.graphql.org/

18 Directus Documentation [Електронний ресурс]. — Directus, 2024. — Режим доступу : https://docs.directus.io

19 Node.js v22.x Documentation [Електронний ресурс]. — OpenJS Foundation, 2024. — Режим доступу : https://nodejs.org/en/docs

20 MySQL 8.0 Reference Manual [Електронний ресурс]. — Oracle Corporation, 2024. — Режим доступу : https://dev.mysql.com/doc/

21 MongoDB Manual [Електронний ресурс]. — MongoDB Inc., 2024. — Режим доступу : https://www.mongodb.com/docs/

22 Redis Documentation [Електронний ресурс]. — Redis Ltd., 2024. — Режим доступу : https://redis.io/docs/

23 Docker Documentation [Електронний ресурс]. — Docker Inc., 2024. — Режим доступу : https://docs.docker.com/

24 Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation / J. Humble, D. Farley. — Boston : Addison-Wesley, 2011. — 512 p.

25 DAMA International. DAMA-DMBOK: Data Management Body of Knowledge / DAMA International. — 2nd ed. — Basking Ridge : Technics Publications, 2017. — 688 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі процесно-орієнтованого підходу»

Виконав: студент групи ПДМ -61 Максим МОГИЛЬНИК

Керівник: канд. фіз.-мат. наук, проф., доцент кафедри ІПЗ Володимир САДОВЕНКО

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: удосконалення методу процесу побудови архітектури інформаційної системи для сфери управління політичною партією за рахунок застосування удосконаленого процесно-орієнтованого підходу та модульних архітектурних принципів.

Об'єкт дослідження: процеси управління діяльністю політичної партії.

Предмет дослідження: методи, принципи та технології побудови архітектури інформаційних систем, орієнтовані на автоматизацію процесів управління політичною партією, за рахунок використання процесно-орієнтованого підходу та модульних архітектурних принципів.

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІСНУЮЧИХ ПІДХОДІВ

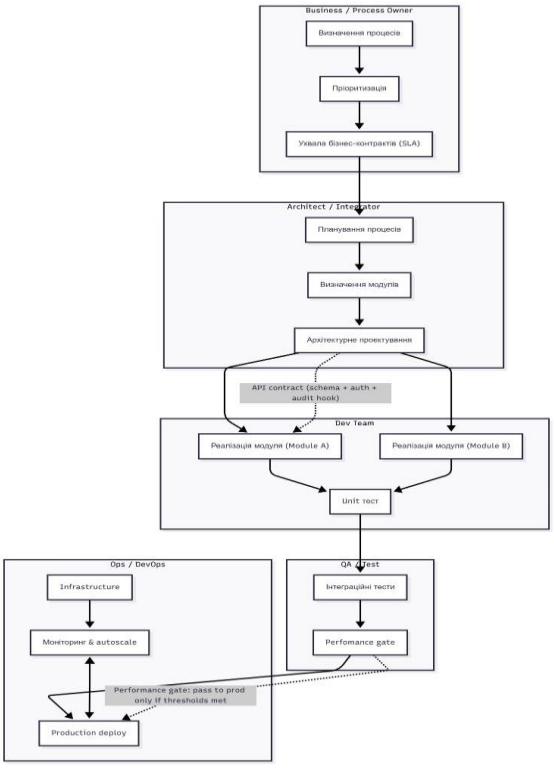
Підхід	Концептуальна основа	Ключові обмеження
Орієнтований на дані	Будується навколо даних як первинної сутності	Оперативні процесні зміни важко відобразити лише зміною схеми даних
Процесно-орієнтований	Тісно пов'язаний з BPMS/ARIS і зосереджений на виконанні, контролі та моніторингу процесів.	Не гарантує хороших технічних метрик через надмірну бюрократизацію процесів
Орієнтований на організаційну структуру	Модулі та сервіси розбиваються по департаментах/відділах, права й інтерфейси віддзеркалюють лінії підзвітності (Conway's law)	Велика ймовірність дублювання та низького повторного використання компонентів
З відкладеною інтеграцією	Розробка підсистем/модулів відбувається автономно; інтеграція загальної системи відсувається на пізнішу фазу	Високий ризик інтеграційного боргу: множинні несумісності, різна модель даних, різні контракти API

УДОСКОНАЛЕННЯ ПРОЦЕСНО-ОРІЄНТОВАНОГО ПІДХОДУ — КЛЮЧОВІ ЗМІНИ

1. Формалізація правил трансформації процесних моделей → модулі / API-контракти.
2. Визначені «data contracts» та стандарти інтерфейсів (з обов'язковим audit hook) на стадії проєктування архітектури.
3. Гібридна розробка: процесний каркас + паралельна модульна розробка (відкладена інтеграція з контролем).
4. Нефункціональні політики: кешування (Redis), GraphQL-батчинг, централізований лог/аудит.

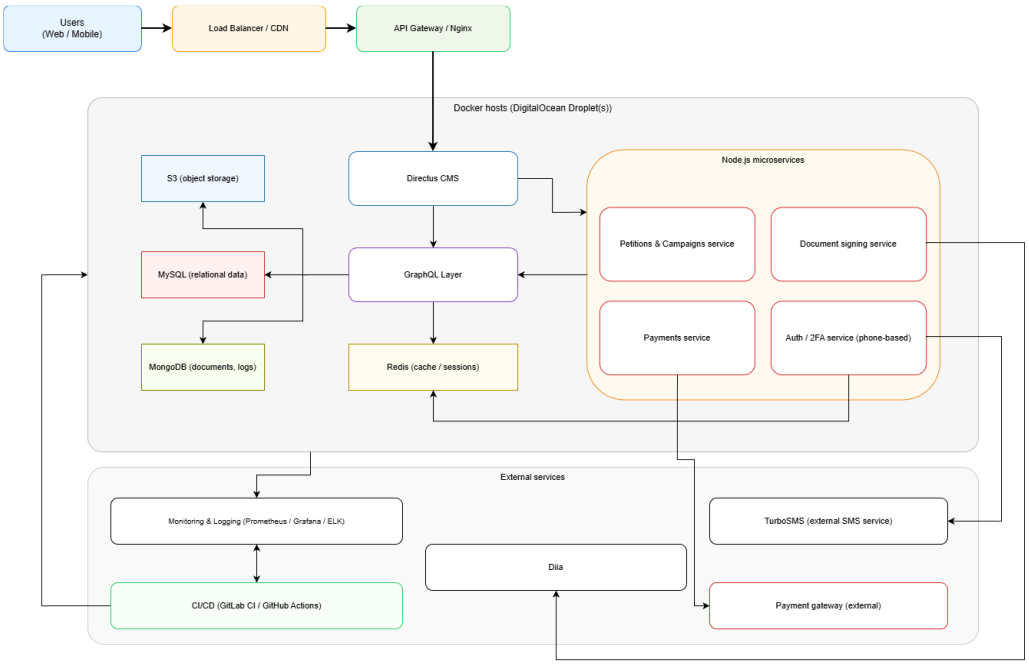
4

ПРОЦЕС РОЗРОБКИ ПЗ ЗА МОДИФІКОВАНИМ МЕТОДОМ



5

АРХІТЕКТУРА ЗАПРОПОНОВАНОЇ СИСТЕМИ



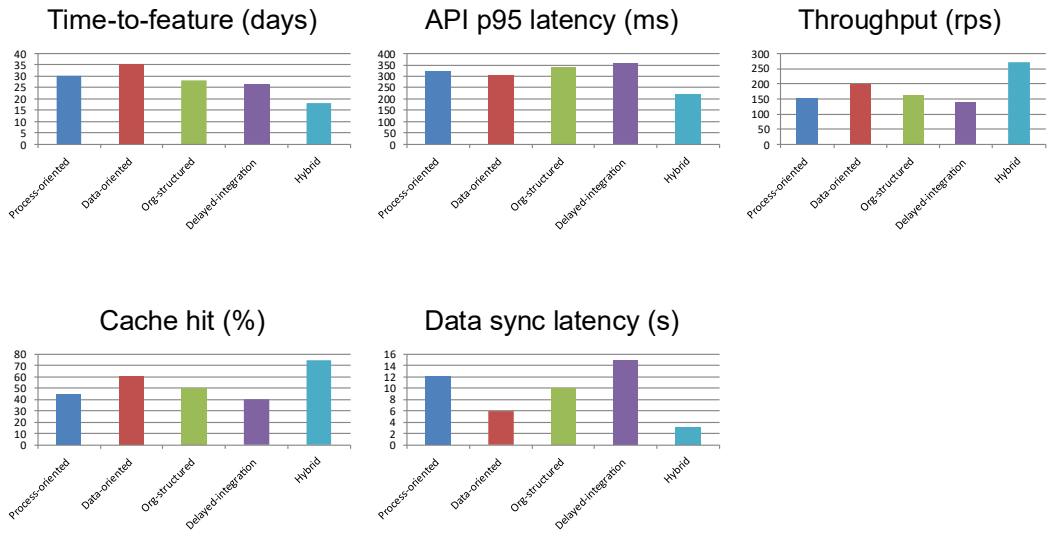
6

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Метрика	Процесно-орієнтований підхід (було)	Удосконалений варіант (стало)	Δ (Абс / %)
Time-to-feature (дні)	30	18	−12 дн (−40.0%)
API p95 latency (ms)	320	220	−100 ms (−31.3%)
Throughput (RPS)	150	270	+120 rps (+80.0%)
Cache hit ratio (%)	45	75	+30 в.п. (+66.7%)
Data sync latency (s)	12	3	−9 s (−75.0%)

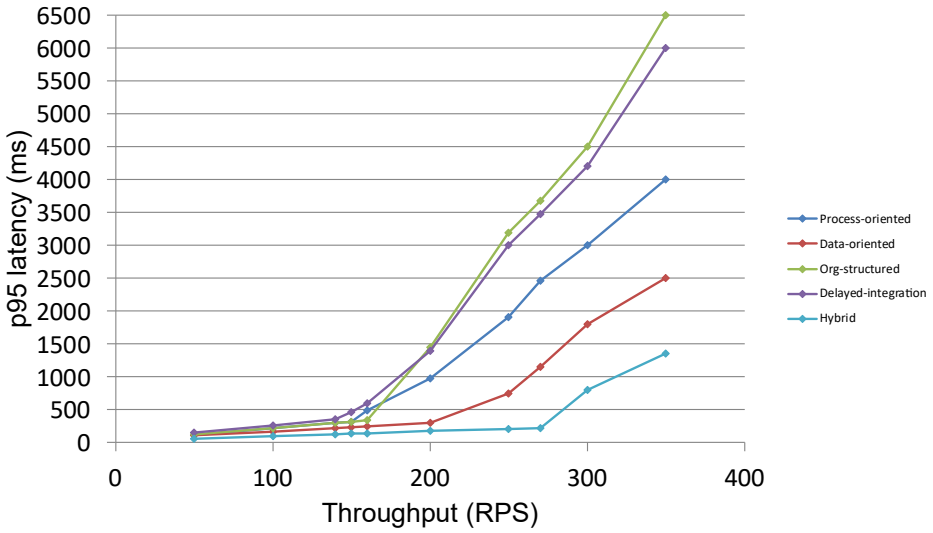
7

ПОВНА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА РЕЗУЛЬТАТУ З ЗАПРОПОНОВАНИМИ АЛЬТЕРНАТИВАМИ



8

ПОВНА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА РЕЗУЛЬТАТУ З ЗАПРОПОНОВАНИМИ АЛЬТЕРНАТИВАМИ



9

ВИСНОВКИ

1. Проведено аналіз проблеми й встановлено наявність суттєвого розриву між процесами партійного управління та існуючими архітектурними підходами: виявлено низьку гнучкість модульної розробки, відсутність єдиних API-контрактів та недостатню політику кешування, що негативно впливало на продуктивність системи та збільшувало час виведення функціоналу в експлуатацію.
2. Розроблено і запропоновано модифікований процесно-орієнтований (гібридний) підхід, який поєднує орієнтацію на дані, урахування організаційної структури та відкладену інтеграцію окремих модулів. На основі такого підходу було реалізовано прототип (Directus + GraphQL, Node.js-сервіси, MySQL/MongoDB, Redis, S3, контейнеризація, CI/CD), що підтвердив практичну здійсненність концепції.
3. Впроваджено ключові технічні вдосконалення: централізовані API-контракти, політики кешування та механізми контрактного тестування; забезпечено модульність розробки з можливістю поетапного введення непершочергових компонентів при збереженні цілісності архітектури, що створює передумови для підвищення продуктивності та скорочення часу розробки.
4. Отримано практичні покращення: скорочено Time-to-feature (30 → 18 дн.), знижено p95 API-latency (320 → 220 ms), збільшено пропускну здатність (150 → 270 RPS), підвищено cache-hit (45% → 75%), зменшено затримку синхронізації даних (12 → 3 с).

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Могильник М.Р., Садовенко В.С. Дослідження архітектурних підходів до побудови інформаційних систем управління політичною партією III Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу», 18 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. С. 178
2. Могильник М.Р., Садовенко В.С. Визначення вимог до удосконалення методу побудови архітектури інформаційної системи для сфери управління політичною партією на основі процесно-орієнтованого підходу Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Подано до друку

ДЯКУЮ ЗА УВАГУ!