ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Методика застосування генеративного штучного інтелекту для проєктування інформаційних систем»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2024
Мулько Андрій Владиславович
Керівник: Слюсарь Ігор Іванович
Рецензент: Муравльов Володимир В’ячеславович

Полтава – 2025 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти другий (магістерський)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Юрій УТКІН
«08» листопада 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Мулька Андрія Владиславовича

1. Тема кваліфікаційної роботи:
«Методика застосування генеративного штучного інтелекту для проєктування інформаційних систем»,
Керівник роботи: к. т. н., доцент, доцент кафедри інформаційних систем та технологій Слюсарь Ігор Іванович.
Затверджено наказом закладу вищої освіти від «31» жовтня 2025 року № 1332-ст

2. Строк подання здобувачем вищої освіти роботи «09» грудня 2025 р.

3. Вихідні дані до роботи: наукові джерела наукометричних баз, дані інтернет-ресурсів щодо генеративного штучного інтелекту, моделей Google Gemini, мови UML, інструментів візуалізації PlantUML, Mermaid.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1. Аналіз особливостей застосування генеративного штучного інтелекту для проєктування інформаційних систем
Розділ 2. Розробка методики застосування генеративного штучного інтелекту для проєктування інформаційних систем
Розділ 3. Рекомендації щодо інтеграції генеративного ШІ у практику проєктування інформаційних систем

5. Перелік графічного матеріалу: схеми, рисунки, діаграми за темою та об’єктом дослідження.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Калініченко О. В., к. е. н., доцент, доцент кафедри економіки та публічного управління	24.11.2025	04.12.2025

7. Дата видачі завдання «08» листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1.	Вибір і затвердження теми роботи	29.10.2024 р.	
2.	Складання та погодження розгорнутого плану та завдання на кваліфікаційну роботу	30.10.2024 р. – 08.11.2024 р.	
3.	Опрацювання джерел інформації	11.11.2024 р. – 27.12.2024 р.	
4.	Збір, вивчення і обробка інформації, необхідної для виконання роботи	30.12.2024 р.– 19.01.2025 р.	
5.	Виконання теоретико-методологічного розділу роботи	17.02.2025 р.– 16.05.2025 р.	
6.	Виконання дослідницько-аналітичного розділу роботи	02.06.2025 р.– 13.07.2025 р.	
7.	Виконання проєктно-рекомендаційного розділу роботи	08.09.2025 р.– 14.11.2025 р.	
8.	Оцінювання економічної ефективності результатів дослідження	24.11.2025 р.– 04.12.2025 р.	
9.	Оформлення тексту роботи	05.12.2025 р.– 08.12.2025 р.	
10.	Попередній захист роботи на кафедрі	09.12.2025 р.	
11.	Доопрацювання роботи з урахуванням зауважень і пропозицій	10.12.2025 р.- 14.12.2025 р.	
12.	Нормоконтроль	15.12.2025 р. – 16.12.2025 р.	
13.	Захист кваліфікаційної роботи	18.12.2025 р.	

Здобувач вищої освіти **Андрій МУЛЬКО**

Керівник роботи **Ігор СЛЮСАРЬ**

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

МУЛЬКО АНДРІЙ ВЛАДИСЛАВОВИЧ

«МЕТОДИКА ЗАСТОСУВАННЯ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ПРОЄКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ»

Освітньо-професійна програма
Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Ступінь вищої освіти Магістр

РЕФЕРАТ
кваліфікаційної роботи на здобуття кваліфікації –
магістр з інформаційних систем та технологій

Полтава – 2025 року

Кваліфікаційна робота складається із вступу, 3 розділів, висновків, списку використаних джерел (41 найменування), додатків. Кваліфікаційна робота містить 6 таблиць, 19 рисунків, викладена на 77 сторінках.

Основний зміст роботи

У першому розділі «Аналіз особливостей застосування генеративного штучного інтелекту для проєктування інформаційних систем» проведено аналіз існуючих підходів до автоматизації (CASE, MDA, Low-code) та виявлено їх обмеженість у вирішенні творчих задач («концептуальна прогалина»). Проаналізовано можливості сучасних LLM та методів їх адаптації (Fine-Tuning, RAG) у контексті проєктування ІС.

У другому розділі «Розробка методики застосування генеративного штучного інтелекту для проєктування інформаційних систем» розроблено методику GAID (GenAI-Assisted Iterative Design). Визначено концептуальну модель, що базується на принципах «Human-in-the-Loop» та ітеративному уточненні. Описано компоненти методики, алгоритми взаємодії з RAG-системою та розроблено шаблони промпт-інжинірингу для різних етапів життєвого циклу.

У третьому розділі «Рекомендації щодо інтеграції генеративного ШІ у практику проєктування інформаційних систем» виконано практичну апробацію методики на прикладі системи бронювання квитків. Згенеровано технічне завдання, Use Cases, діаграми компонентів та послідовності (PlantUML/Mermaid). Проведено декомпозицію мікросервісної архітектури та виконано економічне обґрунтування, яке підтвердило скорочення витрат на проєктування у 3,75 рази.

Висновки

Проведений глибокий аналіз теоретичних засад виявив фундаментальне обмеження існуючих засобів автоматизації, які, залишаючись високоефективними при обробці чітко структурованих даних, виявляються неспроможними подолати «концептуальну прогалину», що виникає при спробі інтерпретації нечітких, абстрактних або неформалізованих вимог замовника. У цьому контексті генеративний штучний інтелект демонструє здатність вирішити зазначену проблему, трансформуючи свою роль з пасивного інструменту на повноцінного інтелектуального партнера архітектора, здатного до семантичного розуміння контексту та пропозиції варіативних рішень.

Ключовим досягненням роботи стала розробка методики GAID, яка чітко регламентує процес використання GenAI, базуючись на моделі циклічної ітеративної взаємодії «запит – генерація – верифікація», що дозволяє контрольовано підвищувати якість результату. Невід’ємною складовою

запропонованого підходу є використання технології RAG (Retrieval-Augmented Generation), яка забезпечує роботу моделі з актуальними корпоративними даними, що критично мінімізує ризики виникнення «галюцинацій» та підвищує довіру до автоматично згенерованого контенту.

Практична реалізація цієї методики дозволила успішно автоматизувати складні процеси створення проєктної документації та генерації візуальних моделей за сучасним підходом Diagrams as Code, що забезпечує синхронізацію архітектурних схем із програмним кодом. Такий підхід не лише радикально прискорив процес проєктування, але й забезпечив гнучкість архітектури, її легку масштабованість та гарантовану відповідність суворим стандартам інформаційної безпеки.

Економічна оцінка впровадження довела беззаперечну ефективність запропонованого підходу, оскільки автоматизація значного обсягу рутинних операцій дозволила скоротити загальні трудовитрати на 75%, перенаправляючи інтелектуальний ресурс команди на вирішення стратегічних завдань. Розрахунки підтвердили, що інвестиції в розробку спеціалізованих промптів та налаштування технічного середовища окупаються в найкоротші терміни, фактично ще до завершення першого пілотного проєкту.

Таким чином, результатами роботи є методика застосування генеративного ШІ для підтримки процесів проєктування інформаційних систем; рекомендації щодо інтеграції генеративного ШІ у практику проєктування інформаційних систем. Вони можуть бути використані для автоматизації рутинних етапів розробки, підвищення ефективності роботи команд розробників та подальших досліджень за даною тематикою.

Список публікацій здобувача

1. Мулько А. Формування вимог засобами штучного інтелекту в процесі проєктування інформаційної системи. *Сучасні аспекти та перспективні напрямки розвитку науки: матеріали X Міжнародної студентської наукової конференції* (жовтень 2025 р. м. Луцьк), 2025. С. 242, 243.

2. Мулько А. Вплив сучасних інтелектуальних технологій на ефективність проєктування інформаційних систем. *Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXII щорічного міждисциплінарного семінару* (листопад 2025 р., м. Полтава), 2025. С. 79, 80.

АНОТАЦІЯ

Мулько А. В. «Методика застосування генеративного штучного інтелекту для проєктування інформаційних систем». Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття ступеня вищої освіти магістр за освітньо-професійною програмою Інформаційні управляючі системи та технології спеціальності 126 Інформаційні системи та технології. Полтавський державний аграрний університет, Полтава, 2025.

Розроблено методику застосування генеративного ШІ для підтримки процесів проєктування інформаційних систем; рекомендації щодо інтеграції генеративного ШІ у практику проєктування інформаційних систем.

Вони можуть бути використані для подальших досліджень за даною тематикою та при проєктуванні інформаційних систем.

Ключові слова: генеративний штучний інтелект, проєктування інформаційних систем, RAG, LLM, prompt engineering, GAID, автоматизація розробки.

ANNOTATION

Mulko A. V. "Methodology of using generative artificial intelligence for information systems design." Qualification work on manuscript rights.

Qualification work for obtaining a master's degree of higher education under the educational and professional program Information management systems and technologies specialty 126 Information systems and technologies. Poltava State Agrarian University, Poltava, 2025.

A methodology for using generative AI (GAID) for information systems design based on Human-in-the-Loop principles has been developed; recommendations for integrating GenAI and RAG tools into the software development process have been substantiated.

They can be used to automate the creation of technical documentation, architectural modeling, and optimize the work of development teams.

Keywords: Generative AI, Information Systems Design, RAG, LLM, Prompt Engineering, GAID, Development Automation.

ЗМІСТ

ВСТУП

Актуальність теми кваліфікаційної роботи підтверджується необхідністю застосування сучасних інтелектуальних технологій, зокрема генеративного штучного інтелекту, у процесах проєктування інформаційних систем. Використання генеративного ШІ дозволяє автоматизувати рутинні етапи розробки, прискорити створення прототипів та підвищити загальну ефективність роботи команд розробників. Інтеграція цих технологій сприяє значному підвищенню продуктивності та оптимізації витрат. Однак, питання розробки чіткої методики застосування генеративного ШІ для проєктування інформаційних систем потребують додаткових досліджень. Все це свідчить про актуальність теми роботи.

Зв'язок роботи з науковими програмами, темами. Робота відповідає дослідженням в межах науково-дослідної ініціативної тематики «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» (ДРН 0123U105060, 2023-2028 рр.), що реалізується на кафедрі інформаційних систем та технологій, тематиці досліджень навчально-дослідної лабораторії інтелектуальних систем, комп'ютерних мереж та інтернет речей кафедри інформаційних систем та технологій Полтавського державного аграрного університету.

Метою кваліфікаційної роботи є підвищення ефективності проєктування інформаційних систем за рахунок застосування генеративного штучного інтелекту.

Завданнями кваліфікаційної роботи є:

– аналіз сучасних методів і підходів до автоматизації проєктування інформаційних систем;

– дослідження можливостей застосування генеративних моделей ШІ у процесах проєктування інформаційних систем;

– синтез методики застосування генеративного ШІ для проєктування інформаційних систем;

– формування рекомендацій щодо інтеграції генеративного ШІ у практику проєктування інформаційних систем;

– оцінка ефективності запропонованих рішень.

Об'єктом дослідження є процес проєктування та розроблення інформаційних систем із використанням інтелектуальних технологій.

Предметом дослідження є інструментарій генеративного ШІ, що застосовуються для автоматизації етапів проєктування інформаційних систем.

Методами дослідження для досягнення поставленої мети виступає комплекс загальнонаукових теоретичних підходів. Для вивчення сучасного стану проблеми, аналізу можливостей інструментів генеративного ШІ та існуючих підходів до проєктування інформаційних систем використовувалися методи аналізу, синтезу та порівняльного аналізу. Для структурування задач проєктування та відповідних їм інструментів ШІ застосовано методи класифікації та систематизації. Ключовим методом для розробки самої методики є моделювання, яке дозволило створити логічно-послідовну модель застосування генеративного ШІ на різних етапах проєктування інформаційних систем.

Інформаційна база кваліфікаційної роботи сформована з ресурсів, що містять інформацію про методики застосування генеративного ШІ, а також інструментарій для проєктування інформаційних систем.

Елементи наукової новизни роботи полягають в розробці методики застосування генеративного ШІ для підтримки процесів проєктування інформаційних систем, яка поєднує концепції системного аналізу, автоматизованої генерації коду та природно – мовної інженерії.

Практична значущість роботи полягає в розробці рекомендацій щодо інтеграції генеративного ШІ у практику проєктування інформаційних систем, вони можуть бути використані для подальших досліджень за даною тематикою та при проєктуванні інформаційних систем.

Апробація результатів відбувалася в рамках X Міжнародної студентської наукової конференції «Сучасні аспекти та перспективні напрямки розвитку науки» (жовтень 2025 р., м. Луцьк) та XXII щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій» (листопад 2025 р., м. Полтава).

Структура кваліфікаційної роботи логічно пов’язана з завданнями досліджень і містить вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг пояснювальної записки кваліфікаційної роботи складає 77 сторінок формату А4. Вона містить 19 рисунків і 6 таблиць.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ ЗАСТОСУВАННЯ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ПРОЄКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Існуючі підходи до автоматизації процесів проєктування інформаційних систем

Інформаційна система (ІС) є фундаментальним компонентом сучасної організації, що забезпечує збір, зберігання, обробку, аналіз та розповсюдження інформації для підтримки процесів прийняття рішень та операційної діяльності. У найзагальнішому визначенні, ІС – це взаємопов'язана сукупність апаратних засобів, програмного забезпечення, даних, персоналу та комунікаційних каналів, організованих для досягнення конкретної мети. Класифікація ІС може здійснюватися за різними критеріями, що відображають їхню складність, призначення та ін. [1]. Наприклад, це може бути за рівнем управління; за сферою функціонування; за архітектурою.

Процес створення та розвитку інформаційної системи описується її життєвим циклом (Software Development Life Cycle, SDLC). Це структурована послідовність етапів, що забезпечує контроль, якість та керованість розробки. Класичні моделі (наприклад, «водоспадна») включають такі фази [2]:

– планування та аналіз вимог – визначення цілей системи, збір та формалізація функціональних і нефункціональних вимог від зацікавлених сторін;

– архітектурне проєктування – визначення загальної структури системи, її компонентів, інтерфейсів та технологічного стеку;

– детальне проєктування – опис логіки кожного модуля, структури баз даних (ER-діаграми), проєктування інтерфейсів користувача (UI/UX);

– реалізація (кодування) – написання програмного коду відповідно до проєктних специфікацій;

– тестування: верифікація та валідація системи – включає модульне, інтеграційне, системне та приймальне тестування для виявлення та виправлення дефектів;

– впровадження (розгортання) – інсталяція системи у робоче середовище, міграція даних та навчання кінцевих користувачів;

– експлуатація та супровід – забезпечення безперебійної роботи системи, виправлення помилок, що виникають, та внесення модифікацій чи оновлень.

Гнучкі методології (Agile), такі як Scrum або Kanban, пропонують ітеративний підхід, де ці етапи повторюються у коротких циклах (спринтах), дозволяючи швидше адаптуватися до змін вимог [3].

Етап проєктування є одним із найбільш трудомістких та критично важливих у життєвому циклі ІС. Помилки, допущені на цьому етапі, мають найвищу вартість виправлення. Тому історично виникло багато підходів для автоматизації та підвищення ефективності цього процесу [4].

CASE технології – це набір програмних інструментів, що автоматизують окремі або всі етапи життєвого циклу ІС. Вони стали першою серйозною спробою систематизувати та автоматизувати проєктування, вони являють собою комплекс програмних інструментів, розроблених для автоматизації різноманітних етапів життєвого циклу інформаційної системи. Вони виступають як системна підтримка для розробників, допомагаючи в аналізі вимог та моделюванні бізнес – процесів і структур даних за допомогою таких формалізованих нотацій, як UML, DFD та ERD. Крім того, ці інструменти часто здатні генерувати програмний код безпосередньо зі створених специфікацій, виконувати реверс-інжиніринг для аналізу існуючих систем та слугувати централізованим середовищем для управління всією проєктною документацією. Використання таких засобів приносить значні позитивні ефекти у процес розробки. В першу чергу, вони сприяють

покращенню загальної якості проєктування, оскільки вимагають застосування стандартизованих формалізованих нотацій (наприклад, UML). Це також забезпечує централізоване зберігання всіх проєктних артефактів у єдиному репозиторії, що покращує комунікацію в команді та цілісність даних. Важливим аспектом є й автоматизація рутинних завдань, таких як створення «каркасу» коду або генерація DDL – скриптів для баз даних, що звільняє час розробників для складніших завдань [5, 6].

Незважаючи на ці переваги, впровадження CASE – технологій пов'язане з певними труднощами. Часто повнофункціональні пакети (так звані Upper/Lower CASE) вирізняються високою вартістю та значною складністю впровадження у робочі процеси компанії. До того ж, ці інструменти нерідко нав'язують жорсткі методології розробки, що обмежує гнучкість команди. Також поширеною проблемою є обмежена гнучкість згенерованого коду, який часто потребує суттєвого ручного доопрацювання для відповідності специфічним вимогам проєкту.

Серед відомих прикладів таких інструментів можна назвати IBM Rational Rose, ERwin Data Modeler та Enterprise Architect.

Моделі MDA (Model-Driven Architecture), запропоновані консорціумом OMG (Object Management Group), представляють собою підхід до розробки програмного забезпечення. Ключова ідея цього підходу полягає в тому, що моделі стають первинними артефактами розробки, а програмний код генерується з них автоматично [7].

Ця методологія спирається на декілька фундаментальних концепцій. PIM (Platform-Independent Model), або модель, незалежна від платформи, яка описує бізнес – логіку та структуру системи, абстрагуючись від конкретних технологій. PSM (Platform-Specific Model), або модель, специфічна для платформи, що деталізує реалізацію системи на конкретній технологічній базі (наприклад, Java EE або .NET). Зв'язок між ними забезпечують трансформації – набір автоматизованих правил для перетворення PIM на одну чи декілька PSM, а також для подальшого перетворення PSM на код.

Застосування MDA дозволяє підвищити загальний рівень абстракції під час розробки, а також надає можливість повторно використовувати – логіку (PIM) для створення рішень на різних платформах. Це, у свою чергу, забезпечує покращену портативність та довговічність систем, оскільки PIM залишається стабільною, навіть коли базові технології змінюються.

Проте, практичне впровадження MDA стикається з серйозними викликами. Існує складність у визначенні повних та коректних правил трансформацій для складних систем. Крім того, такий підхід вимагає наявності висококваліфікованих фахівців, що володіють навичками метамоделювання. Також варто відзначити обмежену підтримку з боку існуючих інструментів для повноцінного циклу генерації коду у випадку складних, масштабних систем.

Середовища low-code / no-code (LCNC) є сучасним підходом до розробки, що фокусується на мінімізації або повній відсутності ручного кодування. Це досягається шляхом використання візуальних інтерфейсів, конструкторів (drag–and–drop) та великого набору готових компонентів.

Такі платформи зазвичай використовуються для швидкої розробки бізнес – додатків, автоматизації внутрішніх процесів організації, а також для оперативного створення прототипів та мобільних додатків.

Головні сильні сторони LCNC – платформ є радикальне скорочення часу розробки (Time–to–Market). Другим фактором висока доступність, оскільки вони дозволяють так званим «громадянським розробникам» (наприклад, бізнес–аналітикам чи менеджерам) створювати функціональні додатки, не маючи глибоких знань у програмуванні. Це, в свою чергу, призводить до зниження загальних витрат на розробку та подальший супровід відносно простих систем [8, 9].

Водночас, цей підхід має низку суттєвих обмежень. Користувачі часто стикаються з обмеженою кастомізацією та гнучкістю, коли виникає потреба вийти за рамки стандартних компонентів. Можуть виникати проблеми з масштабованістю та продуктивністю у високонавантажених системах. Існує

також значний ризик «прив'язки» до вендора (vendor lock–in), що ускладнює перехід на інші рішення, та складнощі з інтеграцією нестандартних систем. Яскравими прикладами таких платформ є Mendix, OutSystems, Microsoft Power Apps та Bubble.

Аналіз вищезгаданих підходів (CASE, MDA, LCNC) виявляє спільну рису, вони спрямовані на автоматизацію формалізованих, структурованих та рутинних аспектів проєктування. CASE-засоби ефективні, коли бізнес-процеси вже описані, і їх потрібно перетворити на формальні моделі (UML, DFD) та згенерувати з них каркас коду. MDA фокусується на перетворенні (трансформації) однієї моделі (PIM) в іншу (PSM), а потім у код. Це процес, що вимагає чітко визначених правил. LCNC-платформи надають візуальний конструктор, де розробка зводиться до композиції заздалегідь створених, функціональних блоків.

В усіх цих випадках інструменти автоматизації вступають у гру тоді, коли концептуальна робота вже виконана. Вони не допомагають на найбільш ранніх та творчих етапах, де відбувається перетворення нечіткої бізнес-ідеї на структуровану специфікацію [10]. Цей феномен можна назвати «концептуальною прогалиною» (Conceptual Gap) в традиційних засобах автоматизації: інтерпретація нечітких вимог – розуміння та структурування вимог, викладених природною мовою, виявлення в них суперечностей, двозначностей та неповних сценаріїїв; генерація альтернативних рішень – мозковий штурм та пропозиція різних архітектурних підходів (наприклад, моноліт проти мікросервісів) для конкретної бізнес – проблеми; створення унікальної бізнес-логіки – написання складних, нестандартних алгоритмів, які не є частиною типових «блоків» чи патернів; концептуальне моделювання – створення початкових моделей даних (ERD) або API-контрактів на основі лише текстового опису бізнес-домену. Вона полягає у відсутності інструментальної підтримки для таких когнітивних завдань [11]. Для кращого розуміння природи цієї проблеми доцільно наочно відобразити межі можливостей існуючих інструментів. Наведена нижче схема ілюструє розрив

між рівнем абстрактного людського мислення, де формуються ідеї, та рівнем формальної логіки, де працюють традиційні засоби автоматизації, підкреслюючи зону, що залишається неохопленою (рис. 1.1).

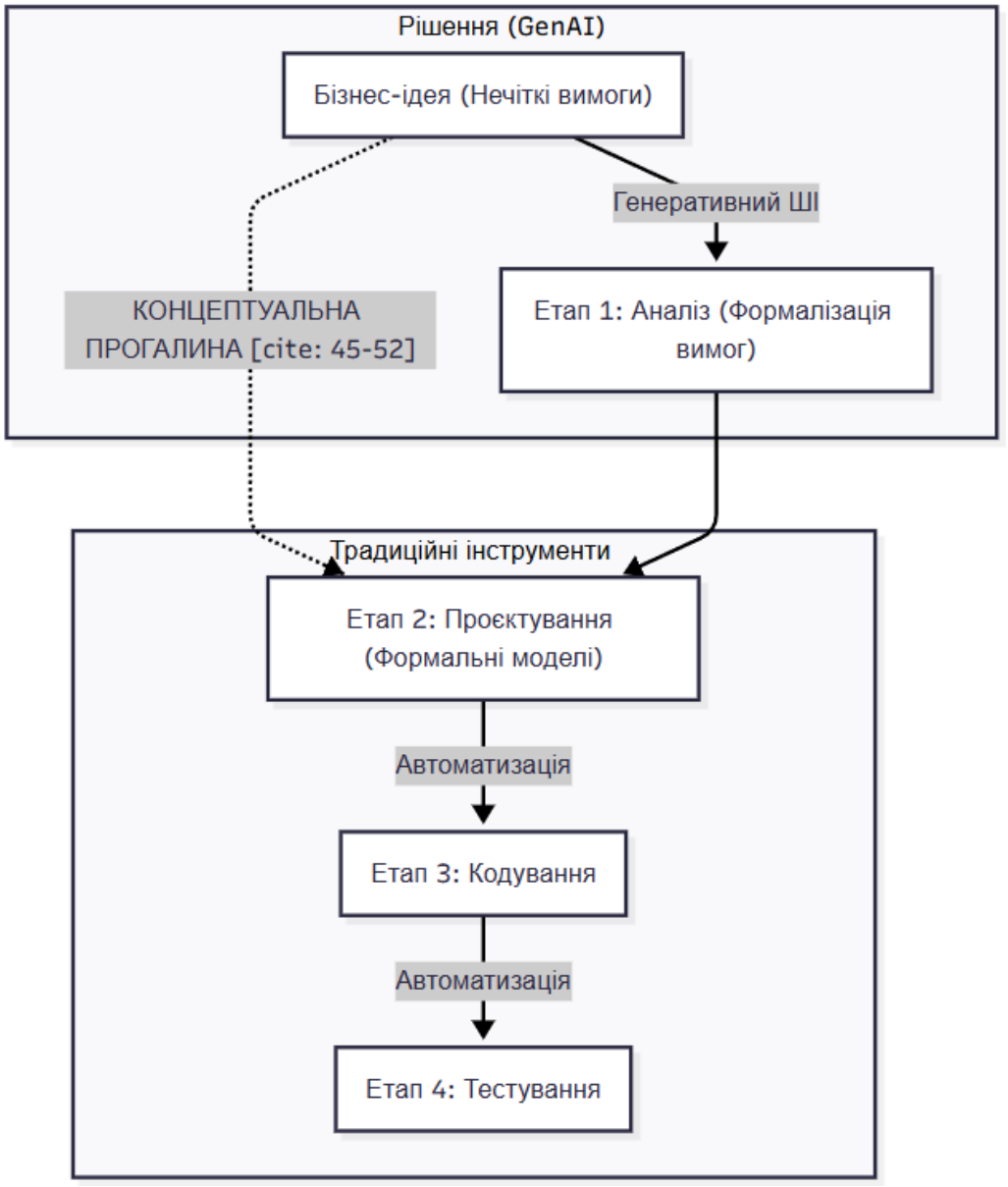


Рисунок 1.1 – Візуалізація «Концептуальної прогалини» в автоматизації проєктування

Традиційні підходи автоматизують те, що вже вирішено, але не допомагають як це вирішити. Вони працюють з формальними моделями, а не з семантикою ідей. Саме цю «концептуальну прогалину» покликаний закрити генеративний штучний інтелект, який, на відміну від попередніх технологій,

оперує безпосередньо природною мовою – універсальним носієм ідей та вимог.

1.2 Технології генеративного штучного інтелекту

Генеративний штучний інтелект (GenAI) – це клас моделей машинного навчання, здатних створювати новий, оригінальний контент (текст, зображення, код, аудіо) на основі вивчених закономірностей з тренувальних даних. На відміну від традиційних аналітичних моделей ШІ, що класифікують або прогнозують, генеративні моделі створюють [12].

LLM – це тип нейронних мереж, що зазвичай базуються на архітектурі Transformer та навчаються на величезних масивах текстових даних, таких як весь інтернет, книги та програмний код.

Вони функціонують, вивчаючи статистичні зв'язки між словами (токенами) і, таким чином, вчаться «передбачати» наступне слово у послідовності. Ця, на перший погляд, проста здатність, при екстремальному масштабуванні до мільярдів і трильйонів параметрів, призводить до виникнення емерджентних властивостей. До них належить здатність до міркування, узагальнення, перекладу та генерації програмного коду.

Процес їх підготовки проходить у два основні етапи: спочатку pre-training (загальне навчання на всіх наявних текстах), а потім fine-tuning (доналаштування на вузьких, конкретних завданнях, наприклад, «питання – відповідь» або генерація коду). Сучасні моделі для покращення якості відповідей та дотримання інструкцій також використовують RLHF (Reinforcement Learning from Human Feedback).

У контексті проєктування інформаційних систем, LLM знаходять широке застосування для генерації коду за текстовим описом, написання технічної документації, проведення рефакторингу, пояснення складних ділянок коду, генерації SQL-запитів та створення тест-кейсів.

Дифузійні моделі є домінуючою технологією для генерації високоякісних зображень, хоча їхній фундаментальний принцип може бути застосований і до інших типів даних.

Процес їх навчання відбувається у два етапи. Спочатку йде прямий процес (forward), під час якого до чистого зображення поступово додається «шум», аж поки воно не стане повністю випадковим. Потім, у зворотному процесі (reverse), нейронна мережа навчається покроково видаляти цей шум, відновлюючи з нього осмислене зображення. Важливою особливістю є кондиціонування (наприклад, текстовим описом), яке дозволяє точно керувати тим, яке саме зображення буде відновлено з шуму.

Сфера їх застосування у проєктуванні ІС включає генерацію UI/UX макетів (mockups) за текстовим описом (наприклад, «створи екран входу в мобільному додатку в стилі мінімалізм»), а також створення іконок, ілюстрацій для додатків або маркетингових матеріалів.

Мультимодальні системи – це системи, здатні обробляти та генерувати інформацію у кількох модальностях (типах даних) одночасно, наприклад: текст, зображення та аудіо.

Вони функціонують завдяки об'єднанню різних типів нейронних мереж (таких як LLM для тексту та Vision Transformer для зображень) в єдиному «просторі уявлень» (embedding space). Така архітектура дозволяє моделі знаходити глибокі зв'язки між текстовим описом та візуальними елементами [13].

Це відкриває широкі можливості для проєктування ІС, зокрема для перетворення намальованого від руки ескізу (wireframe) на готовий HTML/CSS код, читання та аналізу UML – діаграм у вигляді зображень з можливістю пояснення їхньої логіки або генерації коду за ними, а також для створення діаграм на основі текстового опису бізнес – процесу.

Фундаментальний прорив, що уможливив появу сучасних LLM, пов'язаний з публікацією у 2017 році статті «Attention Is All You Need», яка представила архітектуру Transformer. До цього домінуючі моделі (RNN,

LSTM) обробляли текст послідовно, слово за словом. Такий підхід призводив до втрати контексту на довгих реченнях і унеможливлював ефективне паралельне навчання. Архітектура Transformer вирішила ці проблеми завдяки двом ключовим інноваціям.

Першою інновацією є механізм уваги (Self–Attention). Замість послідовної обробки, він дозволяє моделі одночасно «дивитися» на всі слова в реченні (або навіть у всьому документі). Для кожного слова модель обчислює «бали уваги», які показують, наскільки це слово пов'язане з кожним іншим словом у контексті. Наприклад, у реченні «Система зберігає дані користувача у базі даних» механізм уваги пов'яже слово «вона» (якщо воно буде далі) зі словом «система», а слово «зберігає» – з «даними» та «базою даних».

Саме цей механізм має вирішальне значення для проєктування, оскільки він дозволяє LLM розуміти складні логічні зв'язки у програмному коді. Модель може пов'язати оголошення змінної на початку файлу з її використанням через сотні рядків коду, розуміти, який метод викликається, і як параметри одного класу співвідносяться з іншим.

Другою ключовою інновацією є архітектура кодера – декодера (Encoder – Decoder). Кодер (Encoder) читає вхідну послідовність (наприклад, текстовий опис вимоги) і будує її внутрішнє «розуміння» у вигляді векторного представлення, насиченого контекстом завдяки механізму уваги. У свою чергу, Декодер (Decoder) бере це представлення і генерує вихідну послідовність (наприклад, програмний код), також використовуючи механізм уваги, щоб співвідносити кожне згенероване слово (токен) як з вхідним запитом, так і з уже згенерованою частиною коду.

Саме здатність Transformer ефективно обробляти довгострокові залежності та паралелізувати обчислення дозволила масштабувати моделі до мільярдів параметрів. Це, в свою чергу, призвело до емерджентної здатності «розуміти» логіку, структуру та семантику не лише людської мови, але й мов програмування.

Базові LLM, такі як GPT-4 та LLaMA 3, отримують свої знання із загальних даних Інтернету. Попри їхні вражаючі здібності «з коробки», для ефективного застосування у специфічних умовах проєктування ІС, наприклад, у конкретній компанії, їх необхідно адаптувати. Для такої адаптації існують два основні методи.

Перший метод – це fine-tuning. Його суть полягає у продовженні навчання попередньо навченої моделі, але вже на вузькому, специфічному наборі даних. Наприклад, модель можна донавчити на всій кодовій базі компанії, її технічній документації та внутрішніх стандартах кодування (code style). Внаслідок цього модель «вбирає» в себе специфічний стиль коду, починає знати внутрішні бібліотеки та API компанії і, як наслідок, генерує код, який краще відповідає існуючим практикам. Однак, цей підхід має суттєві обмеження: це дорогий процес, що вимагає значних обчислювальних ресурсів (GPU) та ретельно підготовленого набору даних.

Другим, альтернативним методом є Retrieval – Augmented Generation (RAG), або Генерація з доповненим пошуком. Фундаментально цей підхід не змінює саму модель, а натомість надає їй «зовнішні знання» безпосередньо під час генерації відповіді (в момент запиту). Процес складається з двох ключових етапів. Перший етап – Пошук (Retrieval): коли користувач ставить запит (наприклад, «Як мені додати функцію оплати у відповідності до наших стандартів?»), система спочатку шукає у векторній базі даних (де зберігається вся документація проєкту, вимоги, стандарти) найбільш релевантні фрагменти тексту. Другий етап – Генерація (Generation): ці знайдені фрагменти автоматично додаються до початкового запиту користувача як «контекст». Модель отримує промпт вигляду: «Використовуючи ці документи [фрагмент_1, фрагмент_2], дай відповідь на запит: Як мені додати функцію оплати»

Внаслідок такого підходу, модель генерує відповідь, що базується на актуальній та фактичній інформації з документації проєкту. Це дозволяє уникнути галюцинацій та надавати відповіді, специфічні для конкретної ІС.

Такий метод має значні переваги: це значно дешевше за fine-tuning, а база знань може оновлюватися в реальному часі (просто додаванням нових документів). Саме RAG вважається найбільш перспективним методом для створення методик застосування ШІ в проєктуванні [14].

Окрім RAG та fine-tuning, існують також гібридні підходи. Вони поєднують сильні сторони обох методів. Спочатку модель проходить fine-tuning, щоб «вивчити» специфічний стиль коду, термінологію та унікальну архітектуру компанії. Потім ця, вже адаптована, модель використовує RAG, щоб отримати доступ до актуальних даних проєкту (поточних завдань, нової документації). Таке поєднання дозволяє досягти як глибокої спеціалізації моделі, так і високої точності й актуальності її відповідей (табл. 1.1).

Таблиця 1.1 – Порівняння методів адаптації генеративних моделей

Критерій	Fine-tuning	RAG
Принцип	Процес продовження навчання моделі на вузькому, специфічному наборі даних.	Надання моделі «зовнішніх знань» (контексту) під час запиту, не змінюючи саму модель.
Процес	Вимагає підготовки великого набору даних та значних обчислювальних ресурсів (GPU) для fine-tuning	2-етапний: 1) Пошук (Retrieval) релевантних документів у векторній БД; 2) Генерація (Generation) відповіді на основі запиту + знайденого контексту .
Оновлення знань	Потребує повного або часткового *перенавчання* моделі, що є дорогим та повільним процесом.	Відбувається в *реальному часі* простим додаванням нових документів у векторну базу даних.
Ризик «галюцинацій»	Вищий. Модель все ще може Генерувати відповіді на основі своїх «внутрішніх» знань, не підтверджених фактами.	Низький. Відповідь «заземлена» (grounded) на знайдених фактах з документації, що запобігає галюцинаціям.
Основна перевага	Глибока адаптація стилю та знання неявних патернів (напр., стиль коду).	Актуальність даних, контроль над джерелами, нижча вартість впровадження та підтримки.

Розуміння цих відмінностей є фундаментальним для побудови ефективної методики використання ШІ, оскільки дозволяє розробникам свідомо обирати інструменти залежно від етапу життєвого циклу системи.

1.3 Аналіз можливостей сучасних генеративних моделей у контексті автоматизації проєктування

Поява потужних генеративних моделей відкрила нові горизонти для автоматизації інтелектуальних завдань у проєктуванні ІС, які раніше вважалися виключною прерогативою людини. Розглянемо ключових гравців ринку та їхні можливості у табл. 1.2.

Таблиця 1.2 – Порівняння генеративних моделей

Модель (Сімейство)	Розробник	Ключові особливості у контексті проєктування ІС
GPT (Generative Pre-trained Transformer)	OpenAI	GPT-4/GPT-4o: Висока якість генерації коду, відмінні здібності до міркування та дотримання складних інструкцій. Мультимодальність (GPT-4o, GPT-4V) дозволяє аналізувати скріншоти, діаграми та ескізи. Широка інтеграція (GitHub Copilot).
Gemini	Google	Gemini 1.5 Pro/Flash: «Нативно» мультимодальна архітектура, розроблена з нуля для роботи з текстом, кодом, зображеннями та відео. Має дуже велике контекстне вікно (до 1 млн токенів), що дозволяє аналізувати цілі кодові бази або об'ємну документацію за один запит.
Claude	Anthropic	Claude 3 (Opus, Sonnet): Висока продуктивність, сильні аналітичні здібності. Також має велике контекстне вікно. Часто демонструє кращі результати у завданнях, що вимагають глибокого розуміння контексту та генерації довгих, структурованих документів (наприклад, технічних специфікацій).
LLaMA (Large Language Model Meta AI)	Meta	LLaMA 3**:** Найпотужніша відкрита (open-source) модель на сьогодні. Дозволяє локальне розгортання, що є критичним для компаній, які не можуть надсилати свій код або дані на сторонні API через політики безпеки.
Mistral	Mistral AI	Mistral Large / Mixtral**:** Європейський конкурент з високою продуктивністю. Моделі Mixtral використовують архітектуру «суміші експертів» (MoE), що робить їх швидшими та дешевшими у використанні при збереженні високої якості.

При аналізі вимог усі сучасні моделі, такі як Gemini, GPT-4 та Claude 3, чудово справляються з обробкою неструктурованих вхідних даних. Вони здатні виявляти двозначності, групувати вимоги за категоріями, пропонувати відсутні сценарії (edge cases) та генерувати формалізовані артефакти, такі як

User Stories або Use Cases. У цьому завданні перевагу мають моделі з великим контекстним вікном, зокрема Gemini 1.5 Pro та Claude 3, оскільки вони можуть «прочитати» та проаналізувати весь об'ємний документ з вимогами за один запит [15].

Генерація коду є однією з найсильніших сторін сучасного GenAI. Моделі можуть писати код на десятках мов програмування, автоматично генерувати «boilerplate» код, реалізовувати складні алгоритми за текстовим описом, писати юніт – тести, а також оптимізувати та проводити рефакторинг існуючого коду. Де стандартом у цій галузі є GPT-4, значною мірою завдяки його глибокій інтеграції в GitHub Copilot. Спеціалізовані моделі, як CodeLlama від Meta, також демонструють високі результати.

У сфері проєктування архітектури та моделей даних моделі можуть виступати цінними асистентами. Вони здатні пропонувати архітектурні патерни (наприклад, мікросервіси чи моноліт) на основі опису проєкту, генерувати ER-діаграми у текстовому форматі (наприклад, Mermaid або DDL-скрипти) або пропонувати готові схеми API у форматі OpenAPI/Swagger. Однак, тут існують значні обмеження: якість пропозицій сильно залежить від досвіду користувача та його навичок у промпт-інжинірингу. Наразі GenAI слід розглядати як асистента архітектора, а не його повну заміну.

Щодо генерації UI/UX та діаграм, мультимодальні моделі, такі як Gemini та GPT-4o, демонструють здатність перетворювати намальовані від руки ескізи на готовий HTML/CSS код, тоді як дифузійні моделі можуть генерувати візуальні макети. Проте, ця сфера також має свої обмеження. Генерація складних, логічно зв'язаних UML-діаграм (наприклад, Sequence або Activity) все ще залишається складним завданням. Водночас, генерація коду для діаграм у форматах PlantUML чи Mermaid є цілком робочою та ефективною функцією.

Одним із ключових ризиків є «галюцинації» та фактична некоректність. Оскільки LLM є імовірнісними моделями, вони можуть генерувати відповіді,

які виглядають правдоподібно, але є фактично неправильними. У контексті проєктування це може проявлятися як генерація коду, що використовує неіснуючі функції або бібліотеки, запропонування логічно хибних або неоптимальних алгоритмів, чи написання тест – кейсів, які не покривають важливі граничні умови.

Наступною серйозною проблемою є ризики безпеки (Security Risks). Моделі, навчені на величезних масивах коду з Інтернету (включно з GitHub), могли засвоїти і небезпечні патерни кодування. Внаслідок цього GenAI може генерувати код, що містить поширені вразливості, такі як SQL-ін'єкції, Cross-Site Scripting (XSS), або неправильну обробку аутентифікації, особливо якщо розробник не надав чітких вказівок щодо безпеки [16, 17].

Значним бар'єром для впровадження GenAI в корпоративному середовищі є питання конфіденційності та інтелектуальної власності (IP). При використанні публічних хмарних сервісів, як ChatGPT або Claude API, весь вхідний код, вимоги та бізнес-логіка компанії надсилаються на сервери третіх осіб (OpenAI, Anthropic). Існує ризик, що ці дані можуть бути використані для навчання майбутніх моделей, що є неприйнятним витоком пропрієтарної інформації.

Тісно пов'язані з цим проблеми авторського права та ліцензування. Навчальні дані LLM включають мільйони рядків коду з відкритих репозиторіїв, що мають різні ліцензії (GPL, MIT, Apache тощо). Наразі залишається невирішеним юридичне питання: чи є код, згенерований ШІ, «похідним твором» від коду, на якому він навчався. Існує ризик, що згенерований фрагмент коду може несвідомо порушувати умови ліцензії (наприклад, GPL), що створює серйозні юридичні проблеми для комерційного продукту.

Нарешті, існує стратегічний ризик надмірної залежності та «деквалiфікації» розробників. Це особливо актуально для молодших фахівців, які можуть перестати глибоко вивчати основи, архітектуру системи та алгоритми, покладаючись на ШІ як на «чорну скриньку». У

довгостроковій перспективі це може призвести до ситуації, коли команда втрачає здатність вирішувати складні, нетипові проблеми або здійснювати глибокий рефакторинг системи без допомоги ШІ. Успішна методика застосування GenAI повинна мати вбудовані механізми для адресації кожного з цих ризиків, зокрема через процедури верифікації, використання RAG – систем на приватних даних та чіткі політики безпеки (табл. 1.3).

Таблиця 1.3 – Систематизація ризиків застосування генеративного ШІ в проєктуванні ІС

Категорія ризику	Напрям застосування у проєктуванні ІС
«Галюцинації» / Фактична некоректність	Генерація коду, що використовує неіснуючі функції або бібліотеки. Пропозиція логічно хибних або неоптимальних алгоритмів.
Безпека (Security)	Генерація коду, що містить поширені вразливості, такі як SQL-ін'єкції або XSS, через навчання на небезпечних патернах.
Конфіденційність та IP	Витік пропрієтарної бізнес-логіки та коду компанії при надсиланні запитів на публічні сервери третіх осіб (наприклад, OpenAI).
Авторське право та ліцензування	Ризик того, що згенерований ШІ код є «похідним твором» від коду з обмежувальними ліцензіями (напр., GPL), що створює юридичні проблеми.
Надмірна залежність / «Декваліфікація»	Втрата командою (особливо молодшими розробниками) здатності глибоко розуміти основи та вирішувати складні проблеми без ШІ-асистента.

Отже, попри високий потенціал технології, ігнорування наведених у таблиці аспектів безпеки та якості може призвести до критичних вразливостей створюваних систем.

1.4 Формулювання вимог до методики застосування генеративного штучного інтелекту

Проведений аналіз показує, що традиційні засоби автоматизації проєктування, такі як CASE, MDA та LCNC, вирішують завдання на рівні формальних перетворень та візуальної композиції . Вони автоматизують те,

що вже чітко визначено, залишаючи поза увагою «концептуальну прогалину».

GenAI, натомість, пропонує парадигму інтелектуального партнерства. Він здатен працювати з нечіткими вхідними даними (природна мова, ескізи) і генерувати нові артефакти (код, текст, моделі), які раніше вимагали креативної та аналітичної роботи людини. GenAI може виступати в ролі асистента – програміста, бізнес – аналітика, тестувальника та технічного письменника одночасно.

Однак, стихійне використання GenAI несе значні ризики, зокрема галюцинації, проблеми безпеки та конфіденційності. Тому, для ефективного та безпечного впровадження генеративного ШІ у процес створення ІС, необхідна чітка методика, яка б регламентувала його застосування.

На основі аналізу, можна сформулювати наступні ключові вимоги до такої методики.

Вона має забезпечувати глибоку інтеграцію в життєвий цикл (SDLC). Методика повинна чітко визначати, на яких етапах SDLC (від аналізу вимог до тестування та супроводу) і для яких конкретних завдань доцільно використовувати GenAI. Важливо, щоб вона була сумісною з гнучкими (Agile) підходами, де GenAI може використовуватися для швидкої генерації коду прототипів у спринтах.

Фундаментальним принципом має стати верифікація та «Людина – в – циклі» (Human-in-the-Loop, HITL). Методика повинна базуватися на тому, що GenAI є асистентом, а не повноцінним виконавцем. Це, в свою чергу, вимагає розробки обов'язкових процедур валідації та верифікації згенерованих артефактів (через код – рев'ю, перевірку специфікацій, тестування) з боку кваліфікованих фахівців.

Методика також повинна address управління якістю та ризиками. Вона має включати детальні гайдлайни з «промпт – інжинірингу» (Prompt Engineering), адаптовані до завдань проєктування ІС (наприклад, шаблони промптів для генерації API чи аналізу вимог). Разом з тим, необхідно

визначити чіткі метрики для оцінки якості згенерованого контенту, такі як точність, повнота та безпечність коду.

Критично важливим є аспект безпеки та конфіденційності (Data Governance). Методика повинна враховувати ризики витоку комерційної таємниці. Для цього вимоги мають включати чіткі рекомендації щодо вибору моделей (наприклад, публічні API, локально розгорнуті open-source моделі, як LLaMA 3) та технологічних підходів (зокрема, RAG на внутрішніх даних) для забезпечення повної конфіденційності [18].

Нарешті, методика має формалізувати визначення нових ролей та компетенцій. Необхідно описати, як саме змінюються традиційні ролі в команді (наприклад, поява «аналітика з підтримкою ШІ» або «розробника-валідатора») та які нові компетенції, зокрема – інжиніринг та навички валідації ШІ-рішень, стають необхідними для фахівців.

Теоретичний аналіз підтвердив, що традиційні підходи до автоматизації проєктування ІС та новітні технології генеративного ШІ не суперечать, а доповнюють один одного. Традиційні засоби ефективні у формалізації, тоді як GenAI бере на себе когнітивні та креативні завдання, закриваючи «концептуальну прогалину». Таким чином, існує обґрунтована теоретична можливість використання генеративного ШІ для кардинального підвищення продуктивності, якості та швидкості створення інформаційних систем. Однак ця можливість може бути реалізована лише за наявності чіткої методики, що регламентує процеси, забезпечує верифікацію та управляє ризиками, вимоги до якої були сформульовані вище.

Висновки до розділу 1

У першому розділі проаналізовано теоретичні засади застосування генеративного штучного інтелекту в проєктуванні інформаційних систем, що дозволило виявити суттєву обмеженість традиційних засобів автоматизації.

Існуючі інструменти, такі як CASE, MDA та Low-code/No-code платформи, ефективно вирішують рутинні та формалізовані задачі, проте залишають поза увагою «концептуальну прогалину», яка виникає на початкових етапах трансформації нечітких вимог у структуровані специфікації. Генеративний ШІ, зокрема великі мовні моделі на базі архітектури Transformer, здатен закрити цю прогалину завдяки механізмам уваги, які дозволяють розуміти контекст і семантику. Порівняльний аналіз методів адаптації показав, що для задач проєктування підхід RAG (Retrieval-Augmented Generation) є більш доцільним, ніж Fine-Tuning, оскільки він забезпечує актуальність даних, мінімізує галюцинації та дозволяє працювати з корпоративною документацією без ризику витоку даних. Водночас впровадження цих технологій несе значні ризики, серед яких ключовими є фактична некоректність відповідей («галюцинації»), проблеми безпеки згенерованого коду, питання захисту інтелектуальної власності та загроза зниження кваліфікації розробників через надмірну довіру до асистента. З огляду на це, успішне застосування генеративного ШІ можливе лише в межах чіткої методики, інтегрованої в життєвий цикл розробки (SDLC), яка базується на принципі (Human-in-the-Loop) і передбачає обов'язкову верифікацію результатів та адаптацію ролей у команді.

РОЗДІЛ 2
РОЗРОБКА МЕТОДИКИ ЗАСТОСУВАННЯ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ПРОЄКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

2.1 Концепція методики

В основі пропонованої методики лежить не повна автоматизація, а парадигма інтелектуального партнерства або «другого пілота» (Copilot). Методика базується на кількох ключових принципах. Вона вимагає Human – in – the – Loop (HITL), де людина (аналітик, архітектор) залишається центральним елементом процесу. Вона не є пасивним спостерігачем, а виступає в ролі валідатора, коректора та джерела експертного контексту, що є прямою вимогою, сформульованою згідно п. 1.4.

Також методика спирається на ітеративне уточнення. Оскільки проєктування не є лінійним процесом, вона побудована на коротких циклах «Запит > Генерація > Верифікація > Корекція», що дозволяє поступово «звужувати» нечіткі вимоги до рівня конкретних артефактів. Це було продемонстровано на практичному кейсі.

Крім того, методика вимагає контекстуальної обізнаності: якість генерації прямо залежить від якості вхідних даних. Тому передбачається активне «заземлення» (grounding) моделі за допомогою релевантного контексту (стандарти компанії, існуючий код, документація), що відповідає підходу RAG.

Концептуальну модель методики, яку можна назвати GAID (GenAI Assisted Iterative Design), можна зобразити у вигляді циклічної діаграми, що повторюється на кожному етапі життєвого циклу ІС [19]. Сам цикл GAID починається з формулювання завдання аналітиком (рис. 2.1).

Експерт визначає завдання (наприклад, «описати Use Cases для модуля X») та збирає необхідний контекст. Далі відбувається генерація

артефакту (GenAI), де модель генерує первинний результат (драфт) на основі промпту та контексту. Наступним кроком є верифікація (Аналітик / HITL), під час якої експерт оцінює результат на точність, повноту, безпечність та відповідність вимогам.

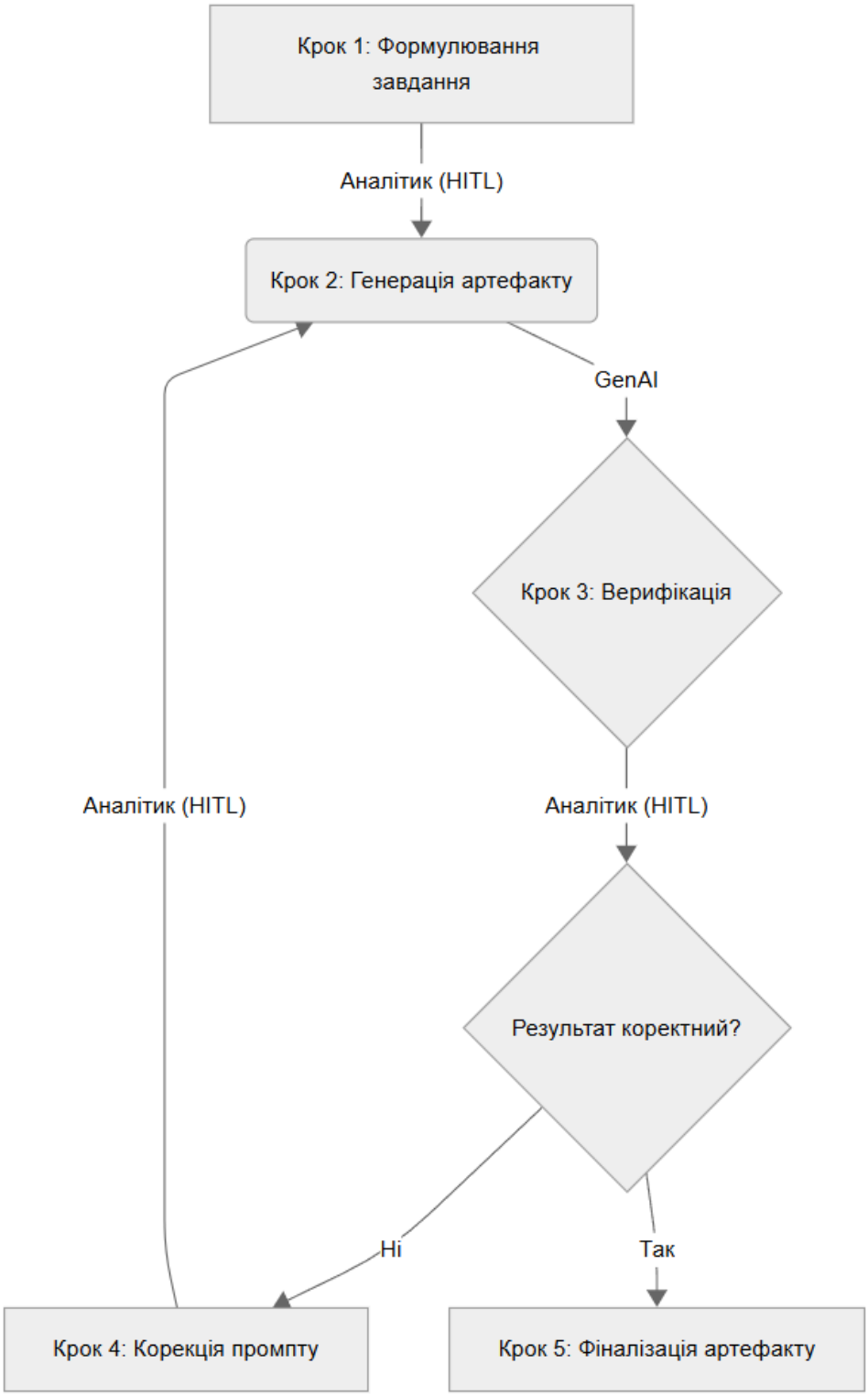


Рисунок 2.1 – Концептуальна модель методики GAID

Завершується цикл етапом корекції або прийняття (Аналітик). Якщо результат неточний, аналітик формулює коригувальний промпт (наприклад, «Це вірно, але додай обробку помилок») і цикл повертається до генерації. Якщо ж результат прийнято, артефакт фіналізується і стає основою для наступного завдання.

Впровадження методики GAID безпосередньо впливає на традиційні ролі в команді, зміщуючи акцент з рутинного створення на стратегічну верифікацію.

Бізнес-аналітик перетворюється з «письменника» ТЗ на «редактора» та «валідатора». Замість того, щоб витрачати дні на написання User Stories з нуля, аналітик використовує GenAI для генерації десятків варіантів за хвилини. Після цього він фокусується на їх пріоритизації, виявленні суперечностей та валідації з замовником.

Розробник (Junior/Middle) звільняється від написання «boilerplate» коду, такого як шаблони, конфігурації чи прості CRUD-операції. Його роль зміщується до інтегратора та валідатора згенерованого коду. Це дозволяє йому фокусуватися на складній, унікальній бізнес-логіці та архітектурних рішеннях.

Архітектор системи отримує можливість швидше прототипувати та порівнювати альтернативні архітектурні рішення. Замість одного детально пропрацьованого варіанту, він може за допомогою GenAI згенерувати та оцінити декілька (наприклад, моноліт vs. мікросервіси) та обрати оптимальний [20, 21].

2.2 Взаємодія генеративної моделі з користувачем

Концептуальна модель GAID застосовується на різних етапах життєвого циклу ІС, трансформуючи роль аналітика та розробника. На етапі аналізу вимог методика вирішує традиційну проблему, яка полягає в тому,

що вимоги замовника часто неповні, двозначні та викладені природною мовою. Застосування методики, де GenAI виступає як «Бізнес – аналітик», дозволяє проводити інтерпретацію та структурування даних. Аналітик «згодовує» моделі неструктуровані нотатки з зустрічей або листи, а GenAI допомагає структурувати їх, виявити суперечності та згрупувати за функціональними блоками. Це також включає генерацію артефактів: модель генерує формалізовані документи, такі як User Stories (за шаблоном «Як [роль], я хочу [дія], щоб [ціль»), Use Cases (детальний текстовий опис) та первинне Технічне Завдання (ТЗ), включаючи функціональні та нефункціональні вимоги (наприклад, вимоги до безпеки, як у кейсі). Крім того, стає можливим виявлення «сліпих зон»: аналітик може поставити запитання на кшталт «Які важливі сценарії (edge cases) ми пропустили в цих вимогах?».

Переходячи до етапу створення архітектури (проєктування), методика допомагає подолати традиційну проблему, пов'язану з тим, що ручне створення діаграм є трудомістким, а швидко оцінити альтернативні архітектурні підходи складно. Тут GenAI застосовується як «Архітектор-асистент». Ключовою можливістю стає генерація коду діаграм: замість «малювання» аналітик текстово описує логіку, а GenAI генерує код для інструментів візуалізації, таких як PlantUML або Mermaid, що значно прискорює процес.

Методика також дозволяє пропозицію архітектурних патернів: на основі ТЗ аналітик може запитати: «Яка архітектура (моноліт чи мікросервіси) краще підходить для цього проєкту? Обґрунтуй». Окрім цього, модель допомагає у декомпозиції (розбитті високорівневих блоків на дрібніші компоненти, як це було продемонстровано в кейсі з декомпозицією «Бекенду») та у генерації моделей даних, створюючи DDL-скрипти (SQL) або ER-діаграми на основі опису сутностей [22].

Нарешті, на етапі генерації технічної документації та коду методика вирішує проблему написання «шаблонного» коду (boilerplate) та ведення

документації вручну, що займає багато часу. У ролі «Програміста-асистента» GenAI забезпечує генерацію коду окремих функцій, класів, реалізацію алгоритмів за текстовим описом та створення юніт-тестів. Він також здатний проводити рефакторинг, аналізуючи існуючий код та пропонуючи варіанти його оптимізації або покращення читабельності. Окрім того, він автоматизує генерацію документації, наприклад, пишучи пояснення до складних ділянок коду або створюючи API-документацію у форматі OpenAPI/Swagger на основі коду. Це процес автоматизованого формування специфікації API шляхом аналізу вихідного коду програмної системи. Такий підхід базується на використанні спеціальних анотацій, коментарів або структур даних у коді, з яких інструменти генерації (наприклад, Swagger, Swashbuckle, SpringDoc, FastAPI-генератори тощо) отримують інформацію про доступні end-point, параметри запитів, моделі даних, формати відповідей та коди помилок. Формат OpenAPI/Swagger є стандартизованою мовою опису REST-інтерфейсів, яка дозволяє представити API у вигляді машиночитного документа (зазвичай у форматі JSON або YAML). Під час генерації «на основі коду» цей документ знаходиться у синхронному зв’язку з фактичною реалізацією серверної логіки. Завдяки цьому уникають розриву між документацією та реалізацією. Тобто, опис API автоматично оновлюється при зміні коду. Також підвищується точність та повнота документації, оскільки вона формується за реальними структурами та методами. Крім того, спрощується інтеграція систем, оскільки розробники отримують формалізований опис end-point та можуть автоматично генерувати клієнтські бібліотеки. Нарешті, забезпечується підтримка сучасних DevOps-процесів, включно з CI/CD, де документація генерується при кожній збірці. Таким чином, створення OpenAPI/Swagger-документації на основі коду – це методика, що інтегрує процес документування API у сам цикл розробки програмного забезпечення, підвищує його прозорість, прискорює розробку та покращує якість інформаційних систем.

2.3 Опис компонентів методики

Для практичної реалізації моделі GAID необхідні декілька ключових компонентів (Додаток А). Важливу роль відіграють вхідні дані. Вони поділяються на первинні (Запит), тобто неструктуровані текстові запити, ескізи (для мультимодальних моделей) або голосові нотатки, та керуючі (промпт), що являють собою чіткі інструкції від експерта (HITL), які керують генерацією.

Найважливішим типом вхідних даних є контекстні (RAG) – рис. 2.2 та 2.3. Це специфічна для проєкту інформація (технічна документація, стандарти кодування, існуюча кодова база, внутрішня база знань Wiki), яка «заземлює» модель.

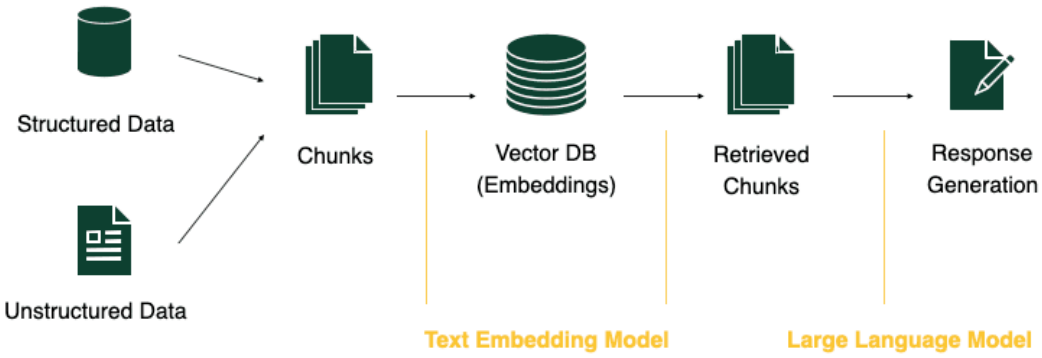


Рисунок 2.2 – Проста схема RAG

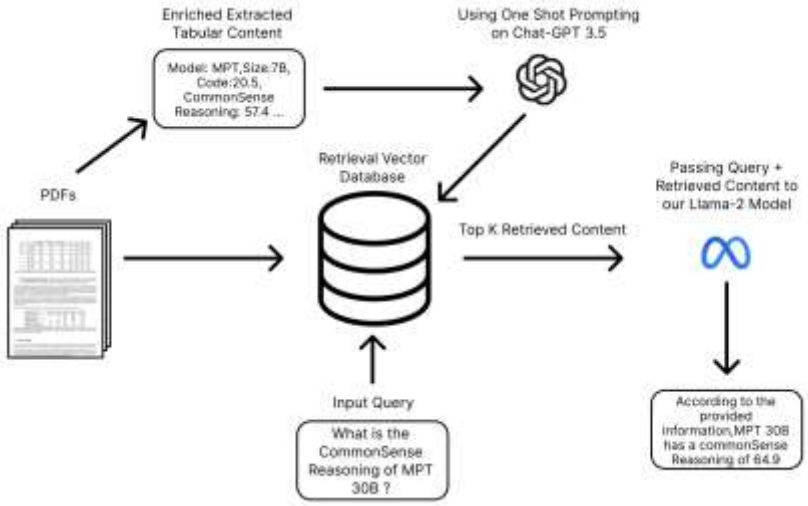


Рисунок 2.3 – Збагачення контексту при використанні RAG

Цей підхід мінімізує «галюцинації» та адаптує GenAI до реалій конкретного проєкту.

Така архітектура дозволяє системі оперувати інформацією, яка з'явилася вже після моменту тренування моделі, забезпечуючи актуальність генерації в реальному часі. Це також вирішує проблему статичності знань, оскільки оновлення інформаційної бази відбувається динамічно і не потребує ресурсномісткого перенавчання (fine-tuning) самої нейронної мережі. Таким чином, модель виступає механізмом логічного виведення, а зовнішня база – надійним джерелом фактів. Деталізований потік даних та взаємодію цих компонентів візуалізовано на рис. 2.4.

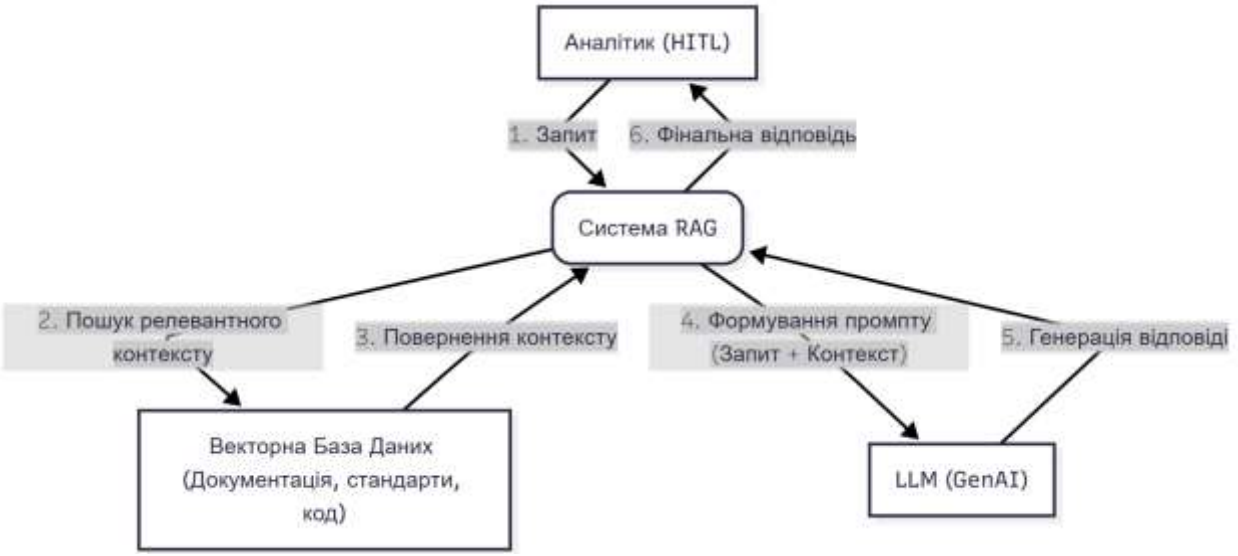


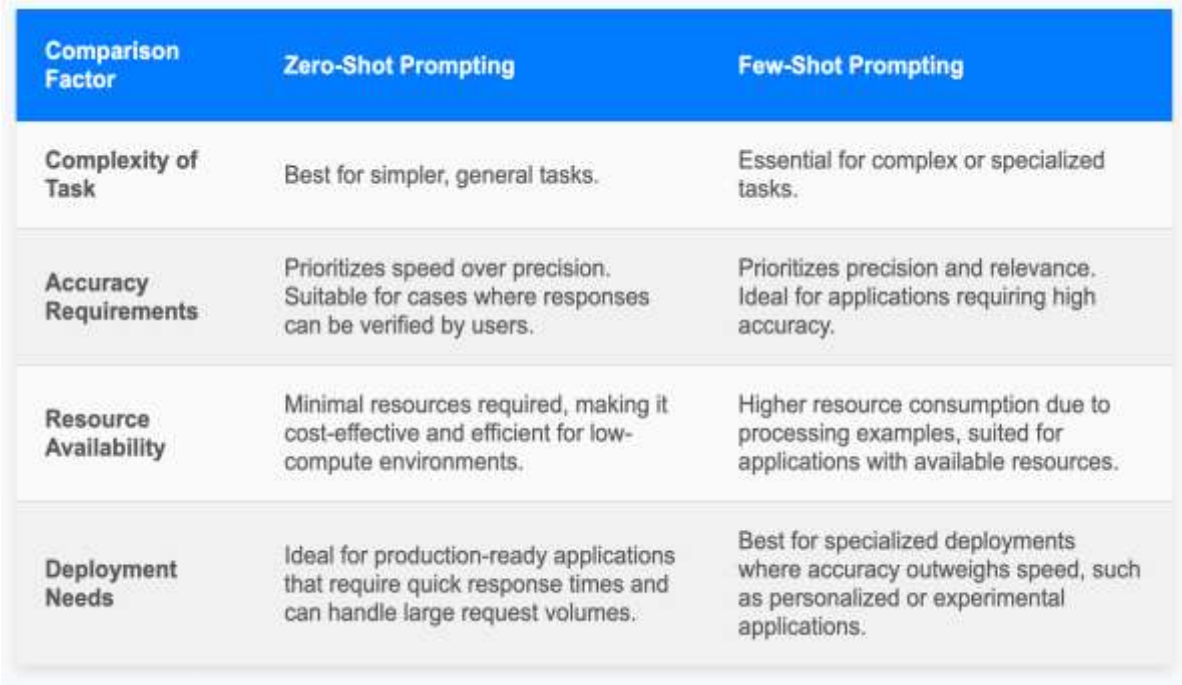
Рисунок 2.4 – Адаптована схема RAG

Також необхідний відповідний інтерфейс взаємодії. Методика вимагає діалогового інтерфейсу, а не статичного «запит-відповідь», оскільки це необхідно для ітеративного уточнення (крок 4 моделі GAID). Це може бути реалізовано як чат–бот (наприклад, ChatGPT, Claude), плагін в IDE (наприклад, GitHub Copilot) або спеціалізований внутрішній веб-портал.

Наступним компонентом є вибір моделі, який залежить від конкретного завдання та ризиків. Великі публічні моделі (GPT-4o, Gemini 1.5 Pro, Claude 3 Opus) використовуються для складних завдань, що вимагають міркування,

аналізу та мультимодальності (аналіз діаграм, ескізів). Швидкі/легкі моделі (Gemini Flash, Claude 3 Sonnet) призначені для рутинних завдань, де важливіша швидкість, наприклад, генерація boilerplate-коду. Водночас, локальні/Open-Source моделі (LLaMA 3, Mistral) є обов'язковим вибором для компаній, де політики безпеки забороняють надсилати пропрієтарний код на сторонні API, що напряму вирішує ризики конфіденційності.

Робота методики спирається на алгоритми генерації результатів. Це може бути «Zero-shot» (Прямий запит) для простих, універсальних завдань («Напиши SQL-запит»), або «Few-shot» (Запит з прикладами) для завдань, що вимагають специфічного форматування («Проаналізуй цей лог... ось 3 приклади, як це робити») – рис. 2.5.



Comparison Factor	Zero-Shot Prompting	Few-Shot Prompting
Complexity of Task	Best for simpler, general tasks.	Essential for complex or specialized tasks.
Accuracy Requirements	Prioritizes speed over precision. Suitable for cases where responses can be verified by users.	Prioritizes precision and relevance. Ideal for applications requiring high accuracy.
Resource Availability	Minimal resources required, making it cost-effective and efficient for low-compute environments.	Higher resource consumption due to processing examples, suited for applications with available resources.
Deployment Needs	Ideal for production-ready applications that require quick response times and can handle large request volumes.	Best for specialized deployments where accuracy outweighs speed, such as personalized or experimental applications.

Рисунок 2.5 – Порівняння Zero-shot та Few-shot

Ключовим алгоритмом для методики є RAG. Цей процес складається з двох етапів: (1) Запит користувача → (2) Система знаходить релевантні документи з векторної бази даних (контекст) → (3) Система «склеює» запит та знайдені документи в один великий промпт → (4) GenAI генерує відповідь, базуючись тільки на наданому контексті [23, 24].

2.4 Інструментарій для застосування методики

Методика GAID не повинна змушувати розробників змінювати звичне середовище. Вона має інтегруватися в існуючі інструменти (workflow) через декілька ключових механізмів.

Найтіснішою є інтеграція з IDE (VS Code, IntelliJ IDEA), що реалізується через плагіни (наприклад, GitHub Copilot, Axenix NeuroCode) або кастомні розширення. Такий механізм дозволяє застосовувати методику безпосередньо в коді для генерації, рефакторингу, написання тестів та документування.

Для інтеграції з інструментами для реалізації CASE/UML (наприклад, Enterprise Architect, PlantUML – рис. 2.6) використовується непрямий механізм. GenAI не «малює» в CASE-інструменті, а генерує текстовий опис діаграми (наприклад, код PlantUML/Mermaid).

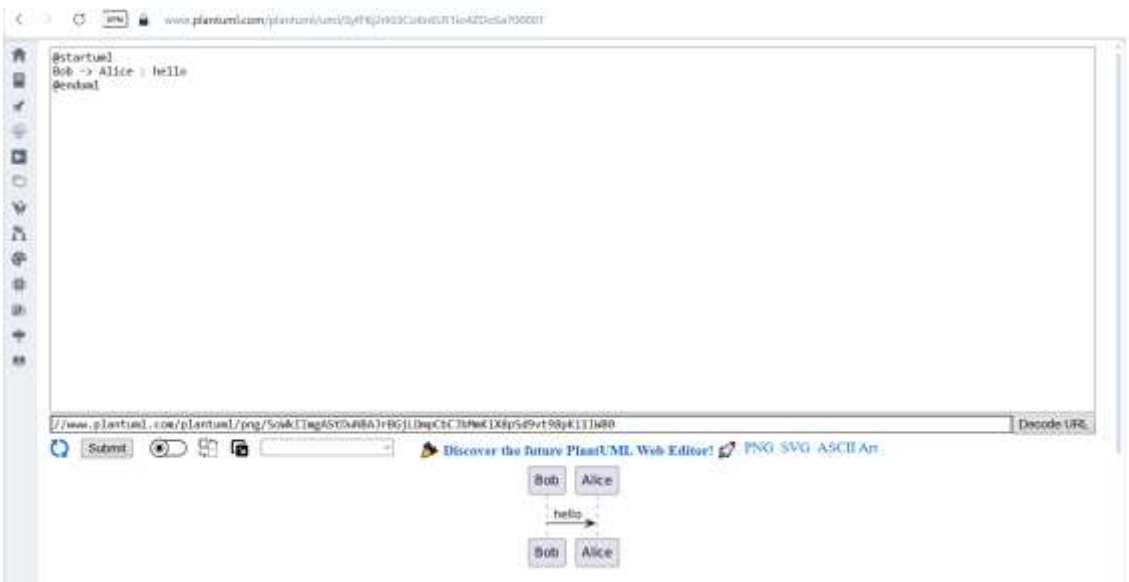


Рисунок 2.6 – Інтерфейс on-line версії редактору PlantUML

Цей код потім копіюється в інструмент для візуалізації, що дозволяє швидко прототипувати архітектуру та діаграми [25]. Також можлива інтеграція через API (ChatGPT API, Google AI Studio), що полягає у створенні власних інструментів, які викликають API моделі. Цей механізм

застосовується для створення кастомних «асистентів» (наприклад, «Асистент з аналізу ТЗ»), які підключені до внутрішньої бази знань (RAG) [26, 27].

Нарешті, інтеграція через платформи автоматизації (n8n, Make.com, Zapier) реалізується через побудову ланцюжків автоматизації («workflows»), що поєднують різні сервіси. Це дозволяє вбудовувати методику у бізнес-процеси. Наприклад, (1) аналітик створює нове завдання в Jira, (2) n8n автоматично «ловить» цю подію, (3) надсилає деталі завдання в ChatGPT API з промптом «Згенеруй 5 User Stories для цього завдання», та (4) публікує результат у Slack для верифікації (HITL).

На основі вищеописаних компонентів та етапів, формулюємо узагальнений алгоритм застосування методики GAID у відповідному процесі проєктному процесі (рис. 2.7).

Процес починається з Кроку 1: Ініціалізація та збір контексту. На цьому етапі аналітик (HITL) чітко визначає мету завдання (наприклад, «Створити API для сервісу користувачів»). Далі визначається необхідний контекст, такий як стандарти компанії щодо API, існуючі моделі даних та вимоги безпеки. У випадку використання RAG, контекстні дані завантажуються у векторну базу або готуються для промпту.

Наступним є Крок 2: Формулювання початкового промпту (HITL). Аналітик створює детальний промпт, що обов'язково включає: Роль («Ти – досвідчений Senior-розробник»), Завдання («Згенеруй специфікацію OpenAPI v3»), Контекст («для сервісу користувачів, який має CRUD – операції. Врахуй наші стандарти безпеки OAuth 2.0») та Формат («Результат очікую у форматі YAML.»). Після цього на Кроці 3: Генерація первинного артефакту обрана модель (наприклад, GPT-4) генерує першу версію (драфт) специфікації [28].

Крок 4: Верифікація, валідація та управління ризиками (HITL) є найбільш відповідальним. Аналітик критично перевіряє згенерований артефакт. Ця верифікація проводиться за метриками якості, що включають: Повноту (чи всі вимоги з промпту враховані?), Коректність (чи немає

логічних помилок або «галюцинацій», наприклад, посилань на неіснуючі бібліотеки?) та Відповідність (чи відповідає артефакт стандартам компанії?).

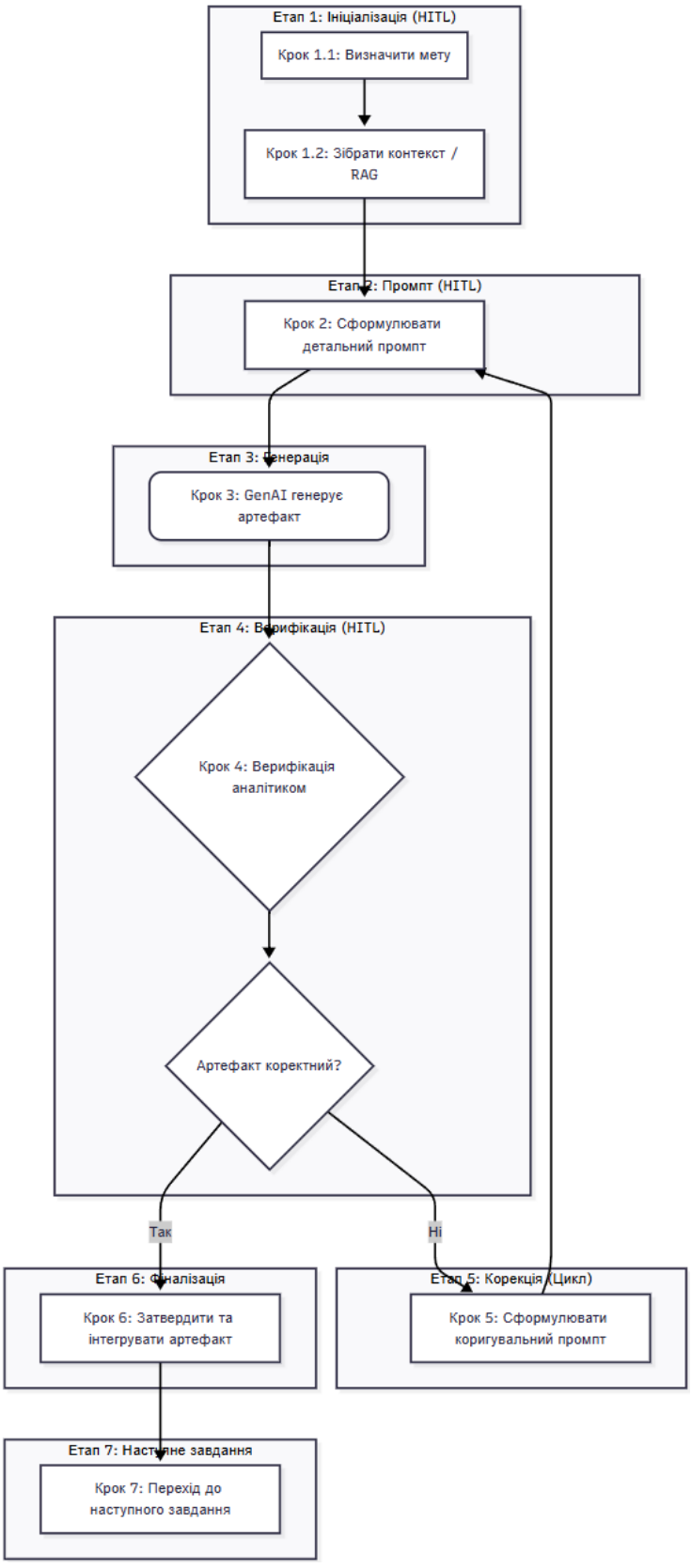


Рисунок 2.7 – Блок-схема методики GAID

Окрім цього, цей крок є обов'язковим для управління ризиками: експерт зобов'язаний перевірити згенерований код на наявність ризиків безпеки (поширених вразливостей, як SQL-ін'єкції чи XSS, оскільки модель могла навчатися на небезпечних патернах) та ризиків IP й ліцензування (переконатися, що код не порушує ліцензійних обмежень, наприклад GPL).

Далі слідує Крок 5: Ітеративна корекція (Цикл). Якщо артефакт визнано некоректним, аналітик не виправляє його вручну, а пише коригувальний промпт (наприклад: «Дякую. Це майже ідеально, але ти забув додати ендпоінт для зміни пароля. Додай його»). Після цього система повертається на Крок 3, генеруючи нову версію. Цей цикл (Кроки 3–4–5) повторюється, доки аналітик не буде повністю задоволений результатом.

На Кроці 6: Фіналізація та інтеграція, аналітик (HITL) остаточно затверджує верифікований артефакт. Після цього артефакт (YAML-файл, код, діаграма) коментується в репозиторій, імпортується в інструменти розробки і стає частиною проєкту [29, 30].

Завершує процес Крок 7: Перехід до наступного завдання. Алгоритм повертається на Крок 1 для ініціалізації наступного завдання (наприклад, «Напиши код контролера на Python, використовуючи цю OpenAPI специфікацію»).

2.5 Промпт-інжиніринг для завдань проєктування

Ефективність усієї методики GAID критично залежить від якості промпту на Кроці 2. Промпт-інжиніринг – це набір практик для формулювання таких запитів, які максимізують точність та релевантність відповіді ШІ. Практичні експерименти показують, що «некоректно складений промпт може призвести до генерації незадовільного результату».

На основі проведеного аналізу сформульовано наступні принципи. Необхідно дотримуватися чіткості та лаконічності, уникаючи двозначності,

довгих формулювань та подвійних заперечень. Важливим є визначення ролі (Role Prompting): завжди варто починати з надання моделі ролі («Уяви себе досвідченим DevOps-інженером»), оскільки це налаштовує модель на правильний стиль та рівень деталізації.

Також ключовим є надання контексту: не слід змушувати модель «вгадувати», адже чим більше релевантної інформації (контексту) надано, тим кращий результат (це основа RAG). Слід застосовувати ітеративність і не намагатися вирішити складну задачу одним промптом, а розбивати її на частини (декомпозиція). Нарешті, потрібно вказувати чіткий формат виводу, завжди зазначаючи, в якому форматі очікується відповідь («Надай відповідь у форматі JSON», «Згенеруй код для PlantUML»).

Системне дотримання цих правил дозволяє перейти від ситуативних запитів до відтворюваних інженерних патернів, що є критичним для забезпечення якості в командній розробці. Це гарантує, що отримані результати будуть консистентними та придатними для інтеграції, незалежно від того, який саме спеціаліст ініціював взаємодію з моделлю.

Такий підхід значно знижує когнітивне навантаження на команду, дозволяючи використовувати вже готові, перевірені конструкції замість вигадування нових запитів з нуля. Для наочної ілюстрації того, як ці теоретичні правила трансформуються в робочі інструкції, було розроблено реєстр еталонних промптів. Ці приклади демонструють правильне поєднання рольової моделі, контексту та обмежень, слугуючи готовою базою для швидкого старту роботи над завданнями. Узагальнені шаблони та конкретні приклади запитів для різних стадій проєктування систематизовано в табл. 2.1.

У цьому розділі була розроблена та формалізована методика застосування генеративного ШІ для підтримки процесів проєктування інформаційних систем (GAID). На відміну від стихійного використання, запропонована методика є структурованим процесом, що базується на принципах Human–in–the–Loop, ітеративного уточнення та контекстуальної обізнаності (RAG).

Таблиця 2.1 – Шаблони промпт-інжинірингу для завдань проєктування

Етап методики	Шаблон промпту	Приклад
Аналіз вимог	Ти – досвідчений бізнес-аналітик. Проаналізуй наступний текст [Текст_вимог] та виконай такі завдання: 1. Вияви суперечності. 2. Виділи ключові функціональні вимоги. 3. Згенеруй 5 User Stories у форматі «Як [роль]»	Проаналізуй цей email від клієнта
Створення архітектури	Ти – архітектор систем. На основі цих вимог [Вимоги] запропонуй архітектуру [Тип_архітектури] у форматі [PlantUML/Mermaid]. Опиши ключові компоненти: [Компонент_1], [Компонент_2].	Згенеруй діаграму компонентів у форматі Mermaid для системи бронювання
Генерація коду	Ти – Senior [Мова] розробник. Напиши [Функцію/Клас] для [Завдання]. Код має відповідати стандартам [Стандарт, напр. PEP8]. Врахуй ці граничні умови [умови]. Додай юніт-тести.	Напиши функцію на Python для валідації email
Корекція (Ітерація)	[Похвала/Констатація]. Твій попередній результат [Попередній_результат] був добрим, але [Проблема]. Модифікуй його, щоб [Зміна].	Ти забув про Сервіс клієнтських даних. Додай його до діаграми

Було визначено концептуальну модель, її вплив на ролі в команді, а також формалізовано застосування методики на ключових етапах SDLC (аналіз, проєктування, кодування). Описано компоненти методики, механізми її інтеграції в існуючі інструменти розробки [31, 32].

Фінальним результатом розділу є узагальнений покроковий алгоритм, який включає механізми верифікації та управління ризиками. Як ключовий елемент успішного застосування алгоритму, було розроблено принципи та шаблони промпт-інжинірингу.

2.6 Прикладні аспекти проєктування за допомогою генеративного штучного інтелекту

Проєктування інформаційної системи бронювання квитків здійснюється з використанням технологій генеративного штучного інтелекту (GenAI), що дозволяє автоматизувати частину етапів аналізу та проєктної

діяльності. Такий підхід відповідає сучасній концепції AI-Augmented Software Engineering (AIASE), де штучний інтелект виступає не просто як генератор коду, а як повноцінний «ко-пілот» архітектора, беручи на себе рутинні когнітивні навантаження.

Основною метою даного етапу є формування архітектурних рішень, визначення функціональних вимог та структуризація компонентів майбутньої системи. При цьому інтеграція GenAI у процес проєктування змінює саму парадигму розробки: відбувається перехід від імперативного створення моделей (створення діаграм вручну) до декларативного (опис намірів та обмежень природною мовою).

У межах дослідження розглядається використання GenAI як інструмента для підтримки інженерних рішень. Методологія використання інструменту базується на принципах ітеративного промпт-інжинірингу. Замість одноразових запитів («zero-shot prompting»), застосовується підхід «ланцюжка думок» (Chain-of-Thought), коли складна задача декомпозується на послідовні логічні кроки. Це дозволяє нівелювати ризик виникнення структурних логічних помилок на ранніх стадіях [33].

Генеративна модель застосовується для формування технічного завдання, уточнення вимог, структурування бізнес-процесів, побудови діаграм та розробки архітектури. Окрім скорочення трудовитрат, цей метод дозволяє виявити приховані залежності та «сліпі зони» у вимогах (edge cases), які аналітик може пропустити через людський фактор (наприклад, специфічні сценарії обробки відмов платіжних систем або колізії при одночасному бронюванні).

Для демонстрації можливостей системи було сформовано вхідні текстові запити (промпти), які базувалися на принципах контекстно-залежного моделювання. Відповіді GenAI використовувалися як вхідні дані для подальшого етапу проєктування. Важливим аспектом методології є верифікація згенерованих артефактів: оскільки LLM (великі мовні моделі) схильні до «галюцинацій» (генерації правдоподібних, але технічно

некоректних тверджень), кожен згенерований блок проходив експертну валідацію на відповідність принципам ACID (для транзакцій) та патернам мікросервісної архітектури. Нижче наведено фрагмент відповіді GenAI, який ілюструє можливості асистента щодо участі у процесі проєктування (рис. 2.8).

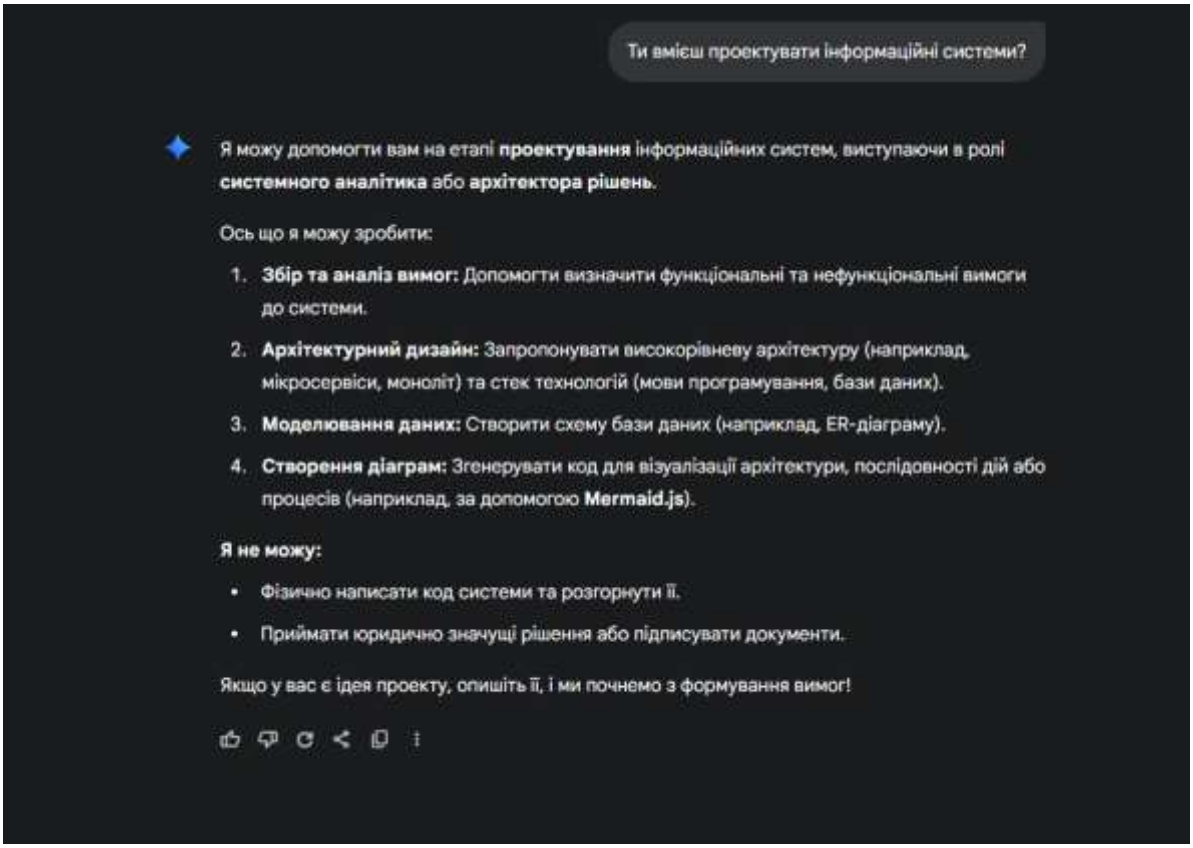


Рисунок 2.8 – Фрагмент відповіді GenAI у процесі проєктування

Отримані результати свідчать, що GenAI здатний ефективно підтримувати процеси збору та структуризації вимог, розроблення архітектури та моделювання. Тому доцільним є використання генеративного підходу як допоміжного інструмента на ранніх етапах життєвого циклу програмного забезпечення.

Крім того, за допомогою GenAI було визначено ключові напрями його застосування у проєктуванні системи:

– аналіз вимог користувачів;

– моделювання бізнес-процесів;

– проєктування структури компонентів;

– створення ER – діаграм;

– формування концепції інтерфейсу користувача.

Для проведення експериментальної частини роботи було обрано середовище Google Gemini, інтегроване у програмну інфраструктуру середовища розробки. Подальші результати, отримані в межах даного етапу, використовуються у наступних підрозділах.

Розроблення технічного завдання (ТЗ) є ключовим етапом проєктування інформаційної системи, оскільки саме на цьому етапі формуються вимоги до функціональності, безпеки, експлуатаційних характеристик та обмежень майбутнього програмного продукту. У межах дослідження було використано можливості генеративного штучного інтелекту для автоматизованого формування структури технічного завдання відповідно до заданих критеріїв та нормативних вимог [34].

З метою визначення базових вимог до системи було сформовано вхідний запит, у якому зазначалося необхідність створення ТЗ на інформаційну систему бронювання квитків із дотриманням вимог безпеки. Генеративна модель сформувала документ, що містив структуровані функціональні, технічні та безпекові вимоги, а також опис етапів розроблення і відповідних термінів виконання.

Фрагмент ТЗ, сформованого GenAI наведено у Додатку А. З огляду на сформоване ТЗ була виконана оцінка необхідного бюджету та трудовитрат. GenAI здійснив попереднє визначення вартості виконання робіт, розподіливши її за фазами розроблення. Оцінювання включало аналіз складності програмного продукту, необхідних ресурсів, вибраних технологій, а також структури команди.

Для наочності в Додатку А наведено блок результатів, сформованих генеративною моделлю.

GenAI надав результати з розбивкою кожного етапу, оцінив необхідний ресурс часу. GenAI не врахував у своїй оцінці трудовитрати на проходження

випробувань. Попросив скоригувати оцінку наступним промптом. “Додай проходження приймально-здавальних випробувань.

В результаті, GenAI перегенерує попередню пропозицію, додавши до неї пункт «Прийомні випробування», а «Впровадження та підтримка» зміщується на фазу 5 (рис. 2.9).

Рисунок 2.9 – Результати автоматизованої оцінки вартості та термінів розроблення ІС

У результаті автоматизованої оцінки було сформовано детальну розбивку вартості за етапами: аналіз, проєктування, розроблення, тестування, впровадження та приймально-здавальні випробування. Остаточний діапазон бюджету становив від 643700 до 1287400 грн, а загальний термін реалізації – від 6 до 10 місяців.

Таким чином, застосування генеративного підходу дозволило значно прискорити процес підготовки технічної документації, а також сформувати обґрунтовану оцінку вартості розроблення системи. Згенеровані матеріали можуть стати базою для наступних етапів проєктування [35].

Висновки до розділу 2

У цьому розділі розроблено методику GAID, яка регламентує застосування генеративного ШІ як інтелектуального партнера в процесі проєктування на основі принципів Human-in-the-Loop та ітеративного уточнення. Запропонований підхід трансформує традиційні ролі в команді, зміщуючи акцент з рутинного створення артефактів на їх стратегічну верифікацію та архітектурне управління.

Ключовим технічним елементом методики визначено механізм RAG, який забезпечує контекстуальну обізнаність моделі та мінімізує ризики «галюцинацій» шляхом використання актуальної проєктної документації.

Практичним результатом розділу стала побудова узагальненого алгоритму, що інтегрує етапи генерації, перевірки безпеки та корекції в єдиний цикл, сумісний з існуючими інструментами розробки.

Для забезпечення стабільної якості результатів було систематизовано принципи промпт-інжинірингу та розроблено набір шаблонів, що дозволяє стандартизувати взаємодію з моделями та перетворити ситуативні запити на відтворювані інженерні патерни.

РОЗДІЛ 3
РЕКОМЕНДАЦІЇ ЩОДО ІНТЕГРАЦІЇ ГЕНЕРАТИВНОГО ШІ У ПРАКТИКУ ПРОЄКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

3.1 Моделювання сценаріїв використання (Use Case)

На етапі структурування вимог було здійснено аналіз функціональних можливостей інформаційної системи бронювання квитків та визначено основні взаємодії між користувачами й системою. Для підтримки цього процесу застосовувався генеративний штучний інтелект (GenAI), який виконував функції допоміжного інструмента для уточнення вимог, формування переліку прецедентів використання та створення текстових моделей.

Із застосуванням промптів було ініційовано автоматичне формування описів основних, додаткових, адміністративних та технічних сценаріїв взаємодії. Це дозволило прискорити процес моделювання та забезпечити чітке структурування вимог згідно з методологією аналізу use case. Взаємодія з GenAI також дала змогу отримати текстову UML-модель у форматі PlantUML, яка відображає перелік акторів, прецедентів та їхні зв’язки. Застосування підходу «Diagrams as Code» дозволило не лише автоматизувати візуалізацію, а й забезпечити легку модифікацію моделей на етапі узгодження вимог без необхідності ручного перемальовування схем. Нижче наведено фрагмент відповіді GenAI на рис. 3.1.

Наступним етапом було отримання переліку use case для системи бронювання квитків за запитом: «Я хочу створити систему Бронювання квитків. Вкажи use cases для даної системи». Цей підхід дозволив не лише згенерувати список функцій, а й автоматично класифікувати їх за ролями та рівнями абстракції, що забезпечує повноту покриття вимог.

Застосування генеративного моделювання на цьому етапі дозволило реалізувати стратегію «від загального до конкретного» (Top-Down Approach).

Первинна генерація забезпечила створення так званих «Kite-level» сценаріїв (рівень огляду бізнесу), які описують стратегічні цілі системи. Надалі, шляхом декомпозиції, ці сценарії були деталізовані до рівня «Sea-level» (цілі користувача) та «Fish-level» (підфункції), що відповідає класичній методології Коберна А. Крім того, ШІ продемонстрував високу ефективність у виявленні альтернативних потоків та виняткових ситуацій (Rainy Day Scenarios) – таких як обрив з'єднання під час транзакції або конфлікт бронювання, – які часто упускаються аналітиками при ручному формуванні вимог.

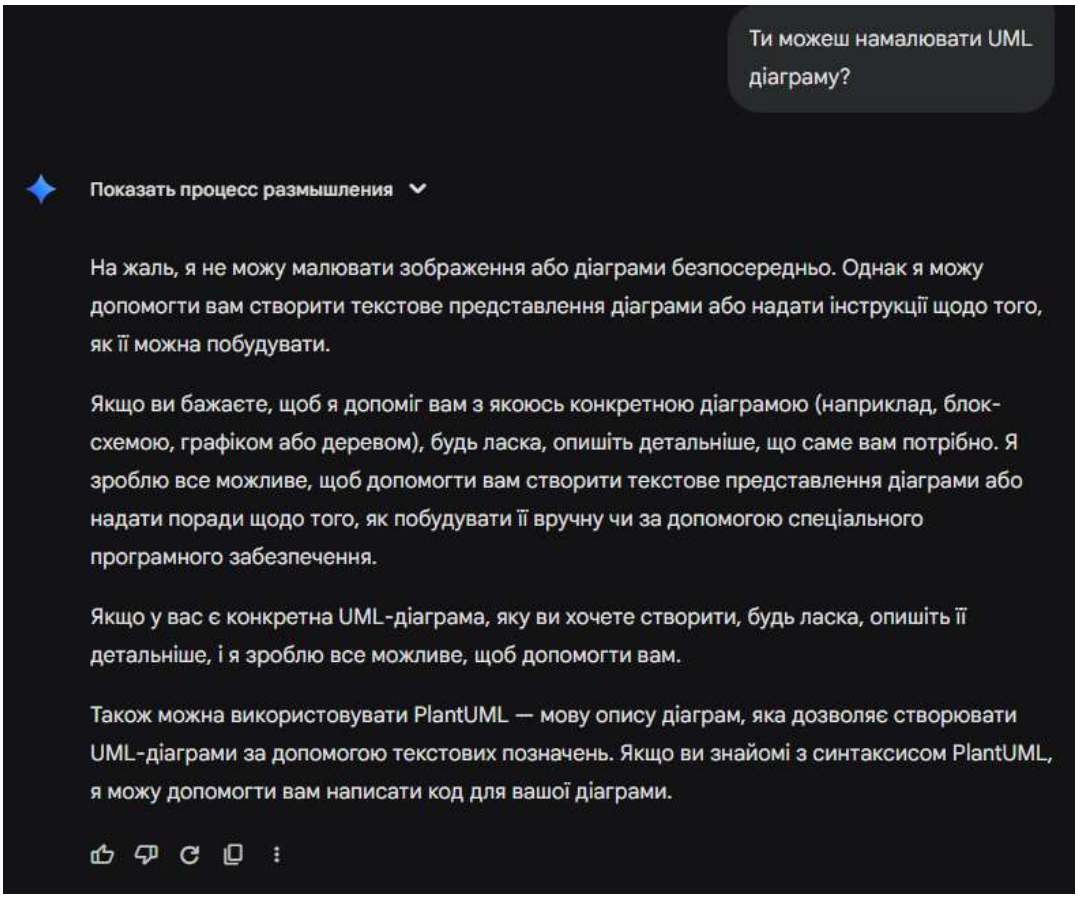


Рисунок 3.1 – Фрагмент генерації сценаріїв використання (Use Cases) засобами GenAI

Основні use cases:

- реєстрація користувача: користувач реєструється в системі, вказуючи свої особисті дані та контактну інформацію;

– авторизація користувача: зареєстрований користувач авторизується в системі, використовуючи логін та пароль;

– пошук квитків: користувач може шукати доступні квитки на певні дати, час, місця відправлення та прибуття;

– бронювання квитка: користувач може забронювати обраний квиток, вказуючи кількість місць та інші необхідні дані;

– оплата квитка: користувач оплачує заброньований квиток, використовуючи один із доступних методів оплати (наприклад, кредитна картка, електронний гаманець);

– отримання квитка: користувач отримує підтвердження про бронювання і може роздрукувати або завантажити заброньований квиток.

Додаткові use cases:

– скасування бронювання: користувач може скасувати заброньований квиток, якщо це дозволено правилами системи;

– зміна даних бронювання: користувач може змінити дані про заброньований квиток (наприклад, дату, час, кількість місць);

– перевірка статусу бронювання: користувач може перевірити статус свого заброньованого квитка (наприклад, підтверджений, скасований, очікує оплати);

– управління профілем користувача: користувач може редагувати свої особисті дані та контактну інформацію у профілі;

– історія бронювань: користувач може переглянути історію своїх минулих бронювань.

Адміністративні use cases:

– управління розкладом рейсів: адміністратор може додавати, редагувати та видаляти розклад рейсів;

– управління цінами на квитки: адміністратор може встановлювати та змінювати ціни на квитки;

– управління доступністю місць: адміністратор може керувати доступністю місць на рейсах;

– моніторинг бронювань: адміністратор може відстежувати поточні бронювання та їхні статуси;

– звітність: адміністратор може генерувати звіти про бронювання, продажі та інші показники.

Технічні use cases:

– інтеграція з платіжними системами: система повинна інтегруватися з різними платіжними системами для обробки оплат;

– інтеграція з базою даних: система повинна взаємодіяти з базою даних для зберігання та вилучення даних про бронювання, користувачів та рейси;

– обробка помилок та виключень: система повинна обробляти помилки та виключення, що виникають під час роботи (наприклад, помилки оплати, проблеми з доступністю місць).

Для візуального подання функціональних вимог було застосовано текстовий формат опису UML-діаграм за допомогою PlantUML. Використання цього інструменту дозволяє автоматично трансформувати текстові сценарії у графічні моделі, що забезпечує синхронізацію між описом вимог та їх візуалізацією. Такий підхід відповідає сучасній парадигмі «Docs as Code», що дозволяє зберігати архітектурні схеми у системах контролю версій (наприклад, Git) нарівні з програмним кодом. Це вирішує проблему підтримки актуальності документації: діаграми можуть автоматично оновлюватися та перегенеруватися в рамках процесів CI/CD, що унеможливлює розбіжності між реальною архітектурою системи та її задокументованим станом. На основі сформованих сценаріїв GenAI згенерував код діаграми прецедентів описаному в Додатку Б (рис. 3.4).

У результаті, було виокремлено ключові групи сценаріїв використання:

– для кінцевого користувача – реєстрація, авторизація, пошук, бронювання, оплата та отримання квитка;

– для адміністратора – управління розкладом, цінами, доступністю місць та моніторинг процесів;

– для технічного рівня – інтеграція з базами даних та платіжними сервісами.

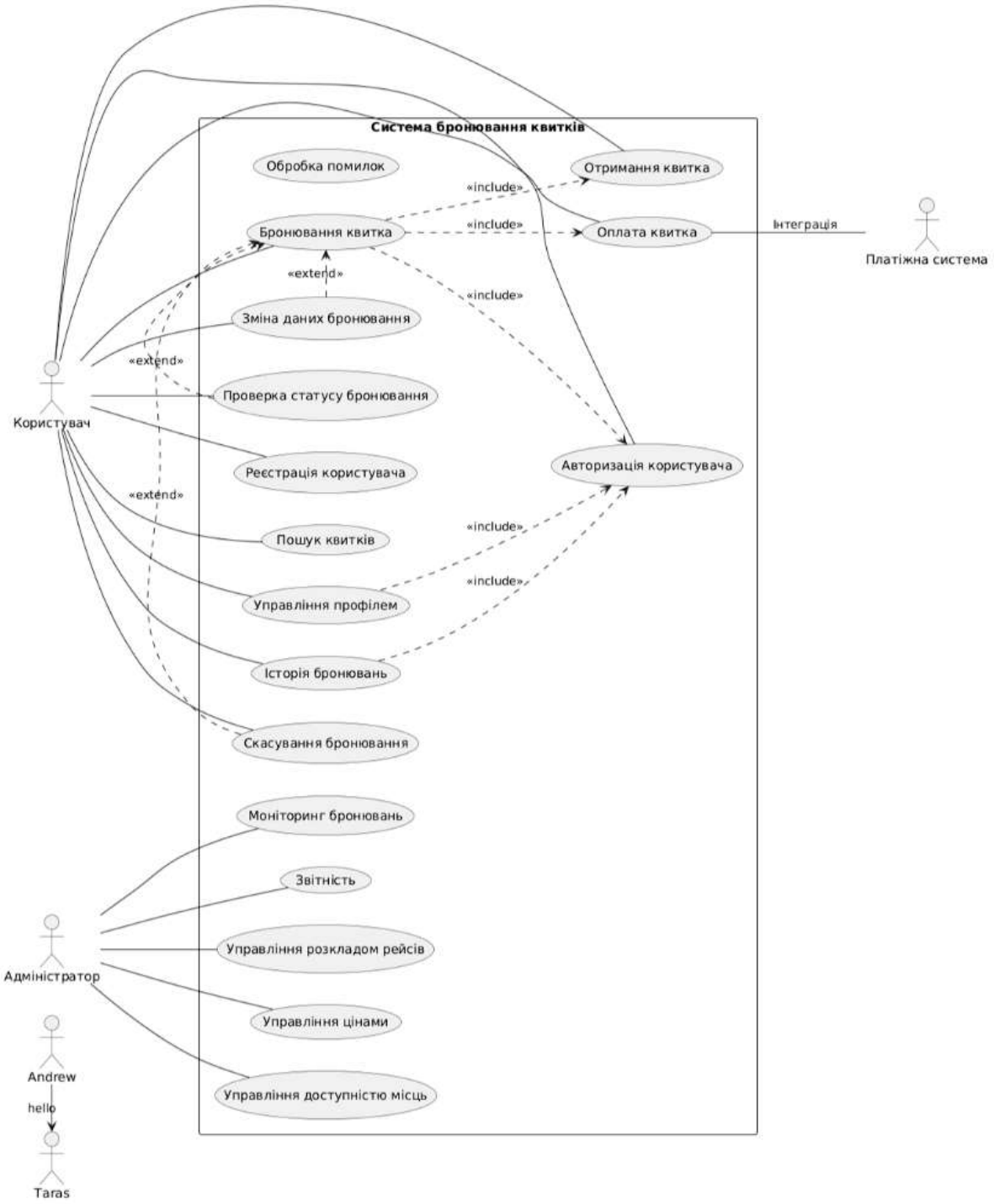


Рисунок 3.2 – Діаграма прецедентів

Згенерований код PlantUML дозволив чітко відобразити взаємозв’язки між акторами та прецедентами, а також використання відносин типу include та extend, що забезпечує формалізоване подання логіки роботи системи.

3.2 Проєктування архітектури та компонентна декомпозиція

Наступним етапом проєктування стало формування компонентної архітектури інформаційної системи. З цією метою за допомогою GenAI було згенеровано діаграму компонентів у форматі Mermaid, яка відображає логічну декомпозицію системи на окремі функціональні сервіси. Використання генеративного підходу дозволило автоматизувати перехід від текстових функціональних вимог до формалізованої структурної моделі, мінімізуючи ризик архітектурних помилок на ранніх стадіях проєктування.

У результаті автоматизованого аналізу система була представлена у вигляді сукупності таких основних компонентів:

- сервіс авторизації та автентифікації користувачів;
- сервіс пошуку квитків;
- сервіс бронювання;
- сервіс оплати;
- підсистема адміністрування.

Запропонована структура базується на принципах слабкої зв’язності (Low Coupling), що є критично важливим для забезпечення незалежного розгортання та горизонтального масштабування кожного окремого модуля в майбутньому. Водночас архітектура враховує принцип високої згуртованості (High Cohesion), де кожен сервіс інкапсулює виключно свою предметну логіку. Такий розподіл забезпечує можливість асиметричного масштабування (наприклад, виділення більших обчислювальних ресурсів для Read-heavy сервісу пошуку порівняно з сервісом оплати) та підвищує загальну відмовостійкість системи: збій в одному компоненті не призводить до каскадної зупинки всієї платформи.

На основі заданого промпту було сформовано код діаграми компонентів у середовищі Mermaid, описаний в додатку Б (рис. 3.3). Отримана схема наочно демонструє взаємодію між користувачами, адміністраторами, сервісами обробки даних та платіжними шлюзами.

Застосування генеративного підходу дозволило оперативно отримати логічну модель компонентної структури, що надалі використовується для побудови детальніших діаграм взаємодії.

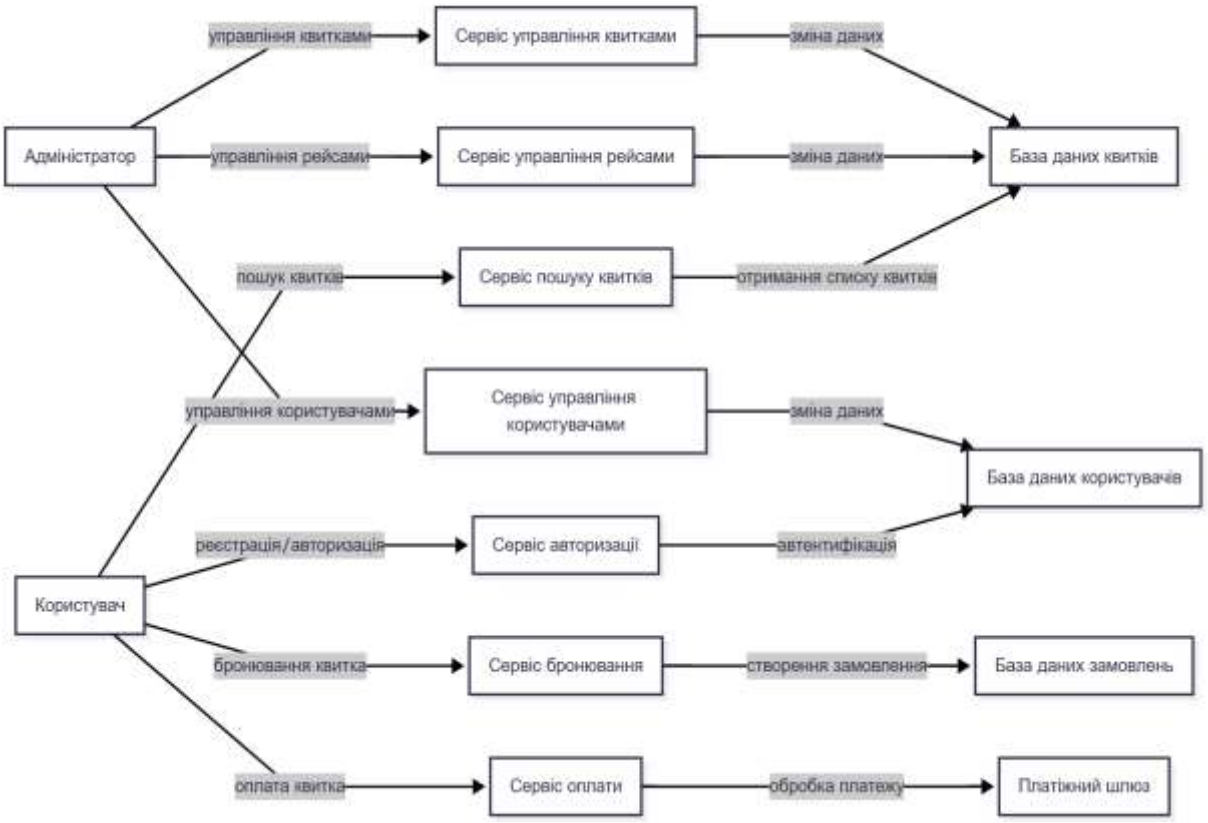


Рисунок 3.3 – Діаграма компонентів, візуалізована через Mermaid

Це забезпечило плавний перехід від статичного опису архітектури до динамічного моделювання поведінки системи, дозволяючи верифікувати коректність обміну даними між модулями. Вхідний промпт містив вимоги до перевірки реєстрації користувача, валідації даних, вибору способу оплати та обробки альтернативних сценаріїв (рис 3.4).

Розроби діаграму послідовності у форматі mermaid для процесу бронювання та купівлі квитка.

Користувач заходить у систему бронювання квитків.

Система перевіряє, чи зареєстрований користувач:

- якщо так, переходить до кроку 5;
- якщо ні, пропонує зареєструватися.

Користувач заповнює форму реєстрації:

- вводить особисті дані;
- створює пароль;
- підтверджує email.

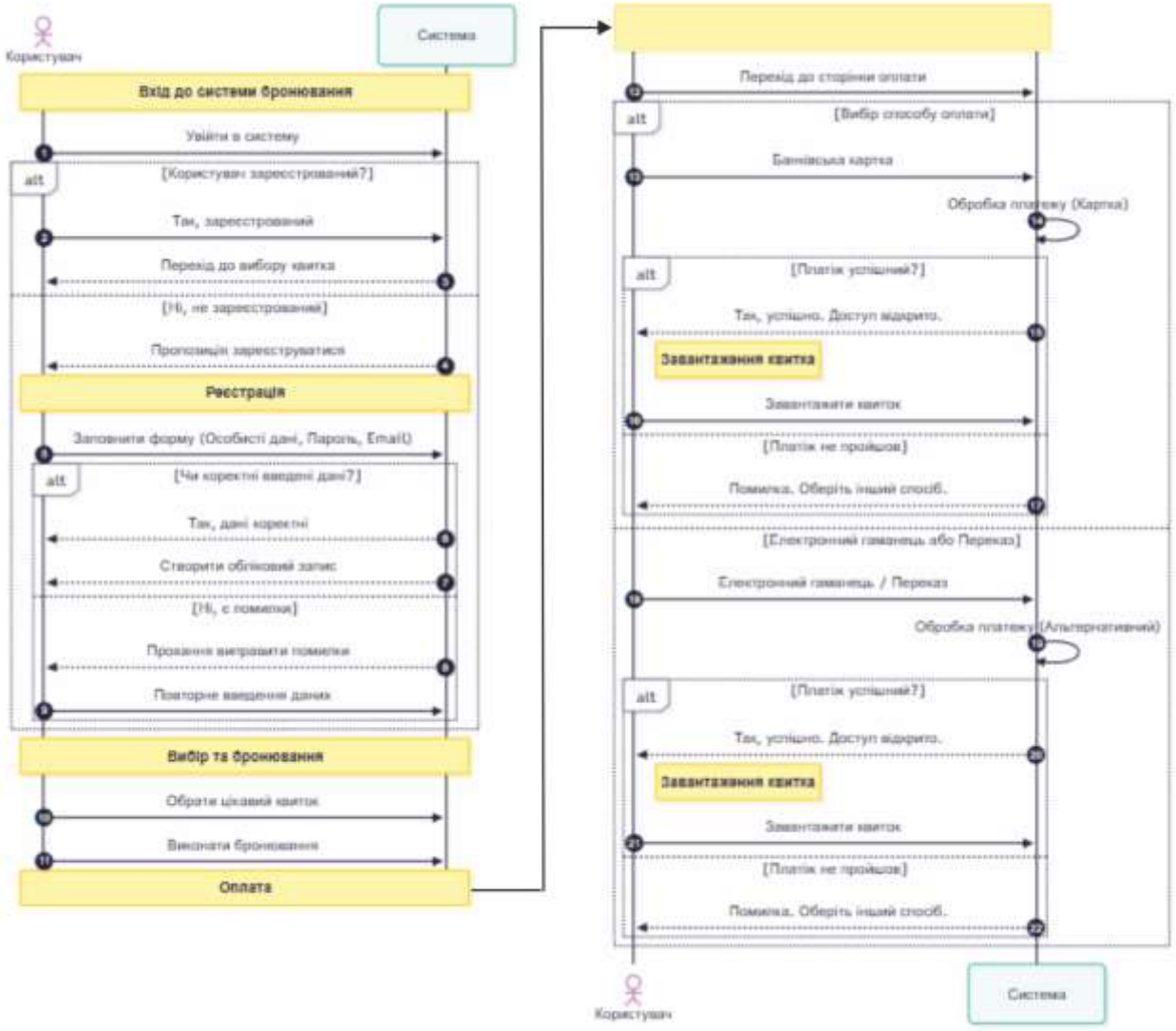


Рисунок 3.4 – Діаграма послідовності процесу бронювання та купівлі квитка

Система перевіряє введені дані:

- якщо дані коректні, створює обліковий запис;
- якщо є помилки, просить користувача виправити їх.

Користувач вибирає цікавий для нього квиток.

Користувач виконує бронювання квитка.

Користувач переходить до сторінки оплати квитка.

Користувач вибирає спосіб оплати:

- банківська картка;
- електронний гаманець;
- банківський переказ.

Система обробляє платіж:

- якщо платіж успішний, відкриває доступ до курсу;
- якщо платіж не пройшов, пропонує вибрати інший спосіб оплати;
- користувач отримує можливість завантажити квиток.

Діаграма повинна відображати всі можливі шляхи виконання процесу, включаючи альтернативні сценарії, точки прийняття рішень та паралельні дії. Використовуй відповідні елементи для відображення розгалужень, злиття, переходів та кінцевих станів. Фрагмент можна переглянути в додатку Б.

Отримана діаграма відображає всі основні та альтернативні гілки виконання процесу, включаючи точки прийняття рішень, помилки платежів та успішну генерацію квитка.

3.3 Декомпозиція бекенду та мікросервісна архітектура

Перед початком безпосередньої реалізації програмного забезпечення було виконано попередній аналіз архітектури інформаційної системи. Цей крок, відомий у сучасній інженерії як «Design Verification», спрямований на мінімізацію розриву між високорівневими вимогами та програмним кодом. На даному етапі здійснювалася декомпозиція задачі з метою подальшого формування діаграми послідовності. Важливо було не лише розділити систему на модулі, а й чітко визначити контракти взаємодії (API Contracts) та потоки даних між ними, щоб уникнути інтеграційних конфліктів у майбутньому. Для цього було залучено генеративний штучний інтелект як інструмент автоматизованого аналізу та формалізації логіки взаємодії компонентів системи [36].

Використання LLM дозволило трансформувати неструктуровані описи бізнес-правил у формалізовані алгоритмічні ланцюжки, автоматично перевіряючи їх на логічну цілісність та виявляючи приховані залежності ще до початку написання коду. Спроєктуй систему бронювання авіаквитків.

Система бронювання авіаквитків повинна включати наступні компоненти:

- фронтенд: веб-інтерфейс або мобільний додаток, через який користувачі можуть шукати та бронювати авіаквитки;
- бекенд: серверна частина системи, яка обробляє запити від фронтенду, взаємодіє з базою даних та забезпечує логіку бізнес-процесів;
- база даних: сховище даних, яке містить інформацію про рейси, авіакомпанії, аеропорти, пасажирів та бронювання;
- сервіси оплати: інтеграція з платіжними системами для обробки оплати авіаквитків.

Функціональні вимоги.

- пошук рейсів: користувач може шукати рейси за різними критеріями (відправлення, прибуття, дата, час, авіакомпанія тощо);
- бронювання квитків: користувач може бронювати авіаквитки онлайн, вказуючи необхідну інформацію (ім'я, прізвище, паспортні дані тощо);
- оплата квитків: користувач може оплачувати авіаквитки через різні платіжні системи;
- управління бронюваннями: користувач може скасувати або змінити своє бронювання, якщо це дозволено авіакомпанією;
- звітність: система повинна надавати звіти про продажі, бронювання та інші бізнес-метрики.

Технічні вимоги.

- мова програмування: Java, Python або інша мова, що підходить для розробки веб-додатків;
- фреймворк: Spring Boot, Django або інший фреймворк, який забезпечує швидку та ефективну розробку;

– база даних: MySQL, PostgreSQL або інша реляційна база даних;

– сервіси оплати: інтеграція з платіжними системами, такими як PayPal, Stripe або інші.

Архітектура системи.

– клієнт-серверна архітектура: фронтенд і бекенд розділені, що дозволяє легко масштабувати систему;

– мікросервісна архітектура: система складається з декількох мікросервісів, кожен з яких відповідає за конкретну функціональність (пошук рейсів, бронювання квитків, оплата тощо).

Безпека.

– шифрування даних: усі дані повинні бути зашифровані під час передачі та зберігання;

– автентифікація та авторизація: користувачі повинні пройти автентифікацію та авторизацію для доступу до системи;

– захист від SQL-ін'єкцій: система повинна бути захищена від SQL-ін'єкцій, щоб запобігти несанкціонованому доступу до бази даних.

Тестування.

– юніт-тести: тестування окремих компонентів системи;

– інтеграційні тести: тестування взаємодії між компонентами системи;

– функціональні тести: тестування функціональності системи в цілому.

Розгортання.

– хмарне розгортання: система повинна бути розгорнута на хмарній платформі (AWS, Google Cloud або Azure);

– контейнеризація: система повинна бути контейнеризована за допомогою Docker;

– оркестрація: система повинна бути оркестрована за допомогою Kubernetes.

Ця система бронювання авіаквитків повинна забезпечити зручний і безпечний спосіб для користувачів шукати та бронювати авіаквитки онлайн. У результаті обробки запиту було отримано узагальнений високорівневий

опис функціонування системи. Такий результат є типовим для першої ітерації проєктування, коли генеративна модель формує концептуальний каркас (Conceptual Framework), абстрагуючись від низькорівневих деталей реалізації. Це дозволяє сфокусуватися на перевірці коректності основних бізнес-потоків без ризику «загрузнути» в технічних нюансах на старті [37, 38]. З метою наочного подання сформованої логіки було виконано її візуалізацію у вигляді діаграми послідовності за допомогою відповідного скрипту Mermaid відображеному в додатку Б (рис. 3.5).

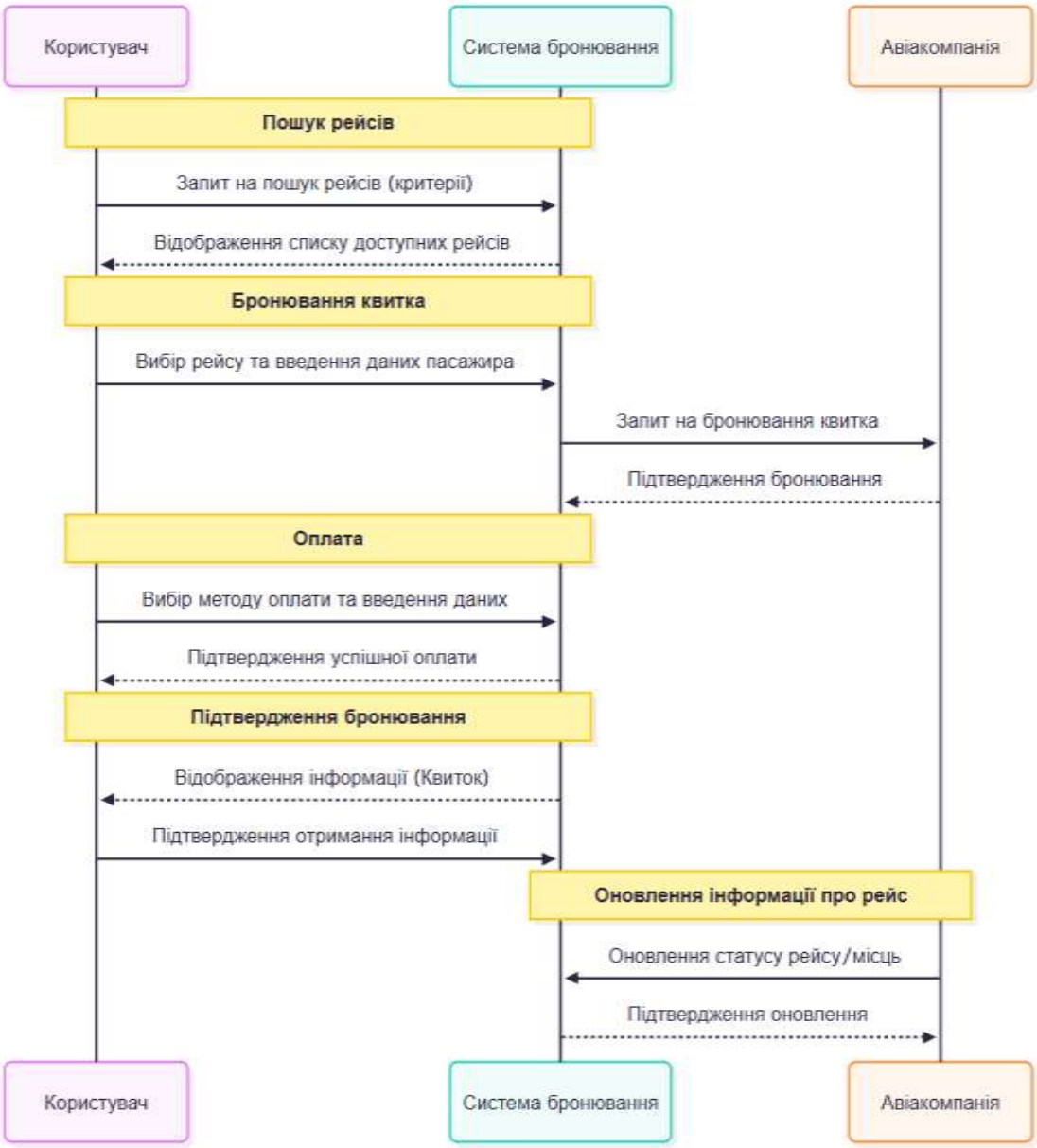


Рисунок 3.5 – Алгоритм обміну даними між системою бронювання та зовнішнім сервісом авіакомпанії

Вибір інструмента Mermaid обумовлений його здатністю рендерити діаграми безпосередньо з текстового опису, що забезпечує швидку ітеративність: будь-яка зміна в логіці системи миттєво відображається на схемі шляхом простого редагування коду скрипту, а не ручного перемальовування графічних елементів.

На початковому етапі вся серверна частина була представлена на схемі у вигляді одного узагальненого елемента, що унеможливлювало детальний аналіз взаємодії внутрішніх компонентів.

Таке представлення фактично трактує бекенд як «чорну скриньку» (Black Box): ми бачимо входи та виходи, але процеси обробки даних залишаються прихованими. Це створює ризик пропустити потенційні «вузькі місця» продуктивності (bottlenecks) та точки відмови ще на етапі проєктування [39]. У зв’язку з цим було виконано його декомпозицію за допомогою відповідного запиту до GenAI відображеному в додатку Б. Дана діаграма послідовності у форматі Mermaid, яка відображає взаємодію цих декомпозованих компонентів (рис. 3.6).

Декомпозируй Бэкенд. Бекенд можна декомпозувати на наступні підсистеми (мікросервіси), кожен з яких відповідатиме за свою зону відповідальності:

– api gateway: відповідає за прийом та обробку вхідних запитів від фронтенду, а також за маршрутизацію цих запитів до відповідних мікросервісів;

– сервіс аутентифікації та Авторизації (Auth Service): відповідає за перевірку особи користувачів та надання доступу до системи на основі їхніх ролей та прав;

– мікросервіс пошуку (Search Service): відповідає за обробку запитів на пошук (квитків/рейсів), включаючи фільтрацію, сортування та пагінацію результатів;

– мікросервіс бронювання (Booking Service): відповідає за створення, зміну та скасування бронювань, а також за оновлення інформації про місця;

– мікросервіс оплати (Payment Service): відповідає за обробку платежів, включаючи перевірку доступності коштів, проведення транзакцій та оновлення статусу оплати;

– сервіс управління даними: відповідає за зберігання, оновлення та надання даних про події (рейси), місця, пасажирів та бронювання;

– сервіс логування та аудиту: відповідає за запис та аналіз журналу подій системи, включаючи помилки, зміни та доступ до даних;

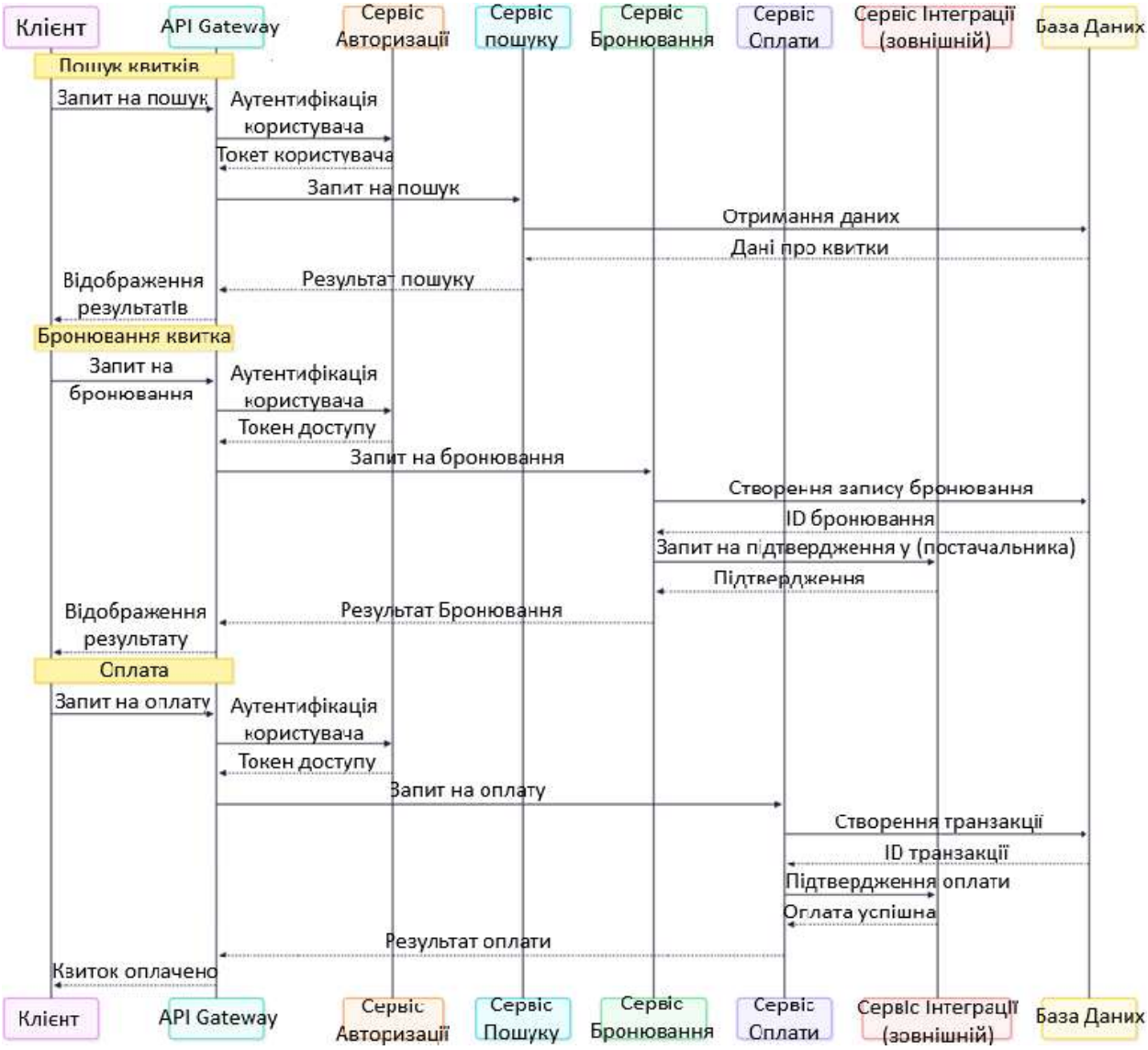


Рисунок 3.6 – Діаграма послідовності взаємодії мікросервісів бекенду

GenAI коректно виконав поставлене завдання, запропонувавши варіант декомпозиції бекенду на окремі сервіси та відобразивши їх взаємодію у

діаграмі послідовності. Водночас у сформованій схемі всі сервіси здійснювали доступ до спільної бази даних, що не відповідає принципам мікросервісної архітектури. Використання спільного сховища є класичним антипатерном, оскільки воно порушує принцип інкапсуляції даних: будь-яка зміна схеми бази даних одним сервісом може призвести до відмови в роботі інших сервісів, блокуючи можливість їх незалежного розгортання (Independent Deployment). У зв’язку з цим спільну базу даних було вилучено зі структури [40, 41].

Генеративний асистент виконав поставлене завдання відповідно до заданих вимог, у результаті чого було сформовано оновлену архітектурну схему такого вигляду (рис. 3.7).

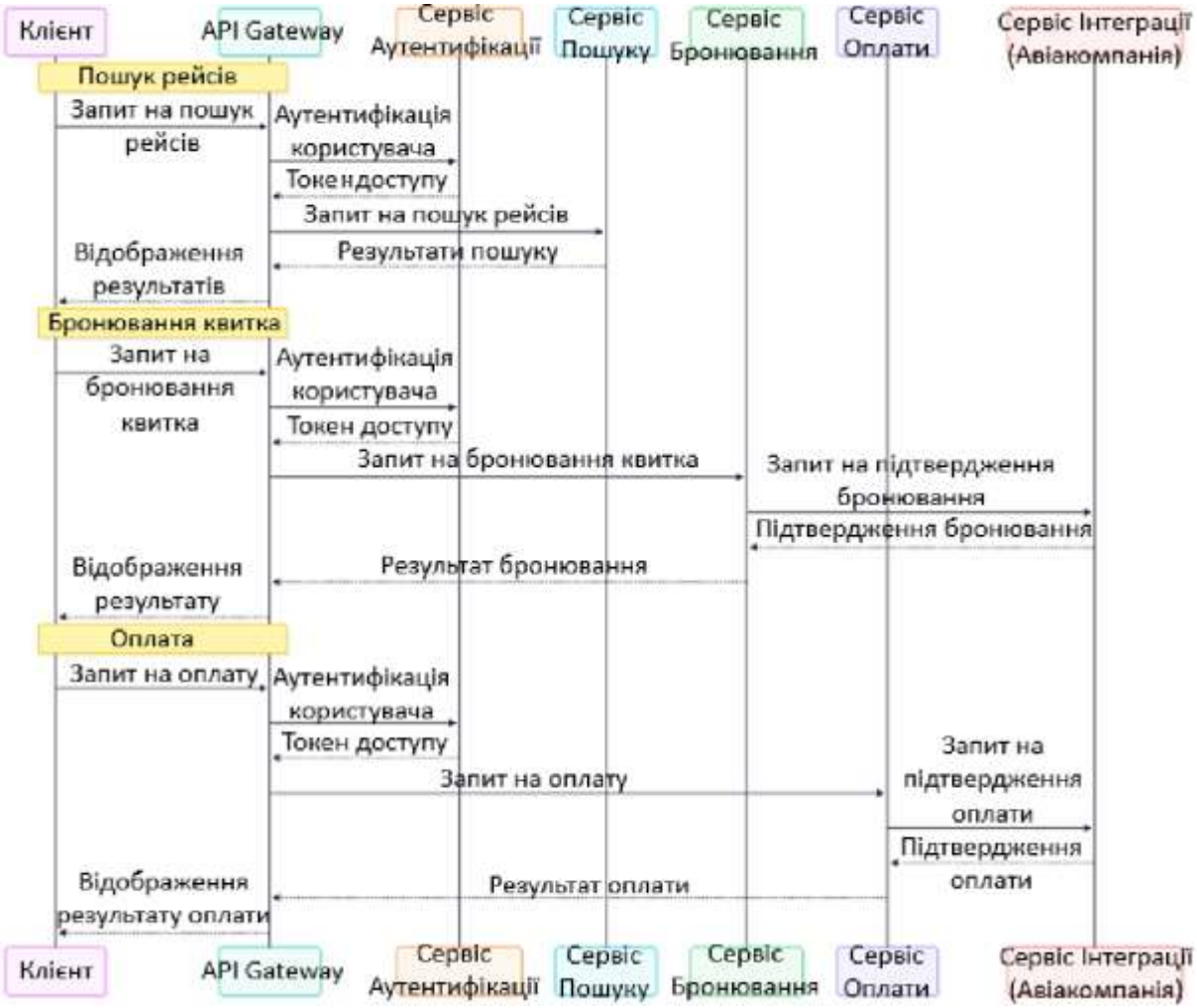


Рисунок 3.7 – Алгоритм обміну даними між внутрішніми сервісами та зовнішньою системою авіакомпанії

Отримана схема є логічно узгодженою, однак для повнішого відображення функціональної структури системи виникла потреба у додатковому сервісі, відповідальному за обробку даних клієнтів. Це рішення продиктоване необхідністю відокремлення транзакційних даних (бронювання) від персональних даних користувачів (PII – Personally Identifiable Information).

Такий підхід дозволяє централізувати управління профілями, реалізуючи патерн «Single Source of Truth», та значно спрощує забезпечення відповідності регламентам захисту даних (таким як GDPR), оскільки вся чутлива інформація зберігається в ізольованому контурі.

У зв’язку з цим архітектурна модель потребує подальшого коригування, яке може бути виконане як шляхом ручного редагування скрипту, так і за допомогою повторного звернення до генеративного асистента. «Додай Сервіс клієнтських даних» (рис. 3.8).

Генеративні моделі демонструють високу ефективність у роботі з готовими архітектурними рішеннями, зокрема при необхідності їх графічного відображення. За умови наявності деталізованого опису структури ІС, GenAI забезпечує швидку генерацію необхідних схем, виступаючи сполучною ланкою між текстовою документацією та візуальним моделюванням (рис. 3.9).

«Спроєктуй систему бронювання авіаквитків. Додай послуги: Служба управління бронюванням: обробляє всі операції, пов'язані з бронюванням, включаючи створення, оновлення та скасування бронювань. Служба управління запасами: управляє доступністю квитків, цінами та спеціальними пропозиціями. Служба керування користувачами: керує обліковими записами користувачів, профілями, автентифікацією та авторизацією. Служба обробки платежів: контролює безпечну обробку транзакцій та інтеграцію з платіжними шлюзами. Служба сповіщень: надсилає підтвердження та сповіщення електронною поштою клієнтам. Напиши скрипт Mermaid, що зображує структуру модулів».

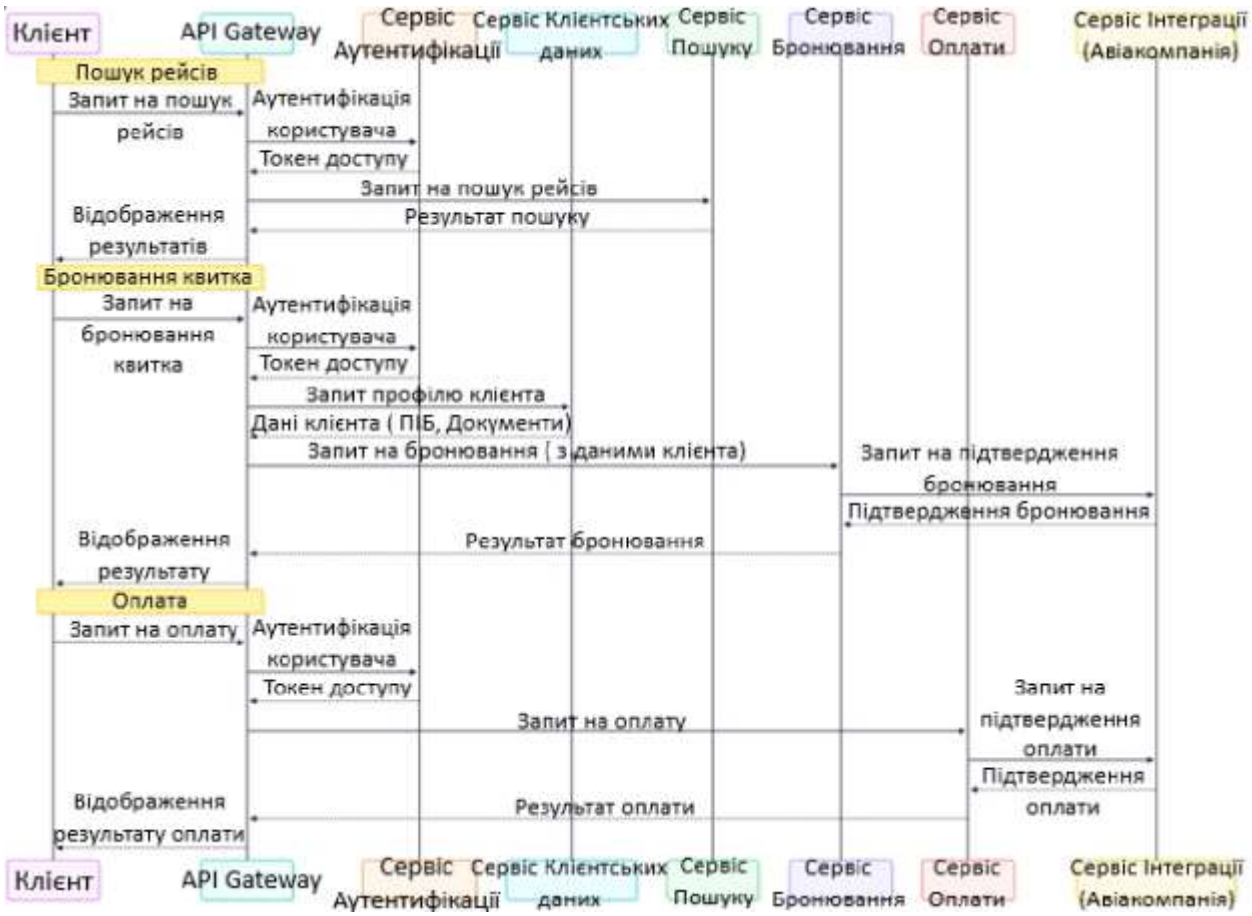


Рисунок 3.8 – Схема взаємодії сервісів бекенду з інтеграцією сервісу клієнтських даних

Для переходу від логічного до фізичного рівня архітектури було виконано декомпозицію бекенду. Спочатку система була представлена у верхньорівневому вигляді: Фронтенд, Бекенд, База даних, Сервіси оплати.

Далі, за допомогою ітеративних запитів, монолітний «Бекенд» було розділено на мікросервіси:

– api gateway: маршрутизація запитів;
– authentication service: безпека та доступ;
– flight search & booking services: пошук та бронювання;
– payment service: обробка транзакцій.

У ході уточнення архітектури було виявлено недолік: усі сервіси зверталися до спільної бази даних, що є антипатерном для мікросервісів. Після команди «Видали Database» та «Додай Сервис клієнтських даних» схема була оптимізована. Фінальна архітектура включає спеціалізовані

служби: управління бронюванням, запасами (inventory), користувачами, платежами та повідомленнями (notifications).

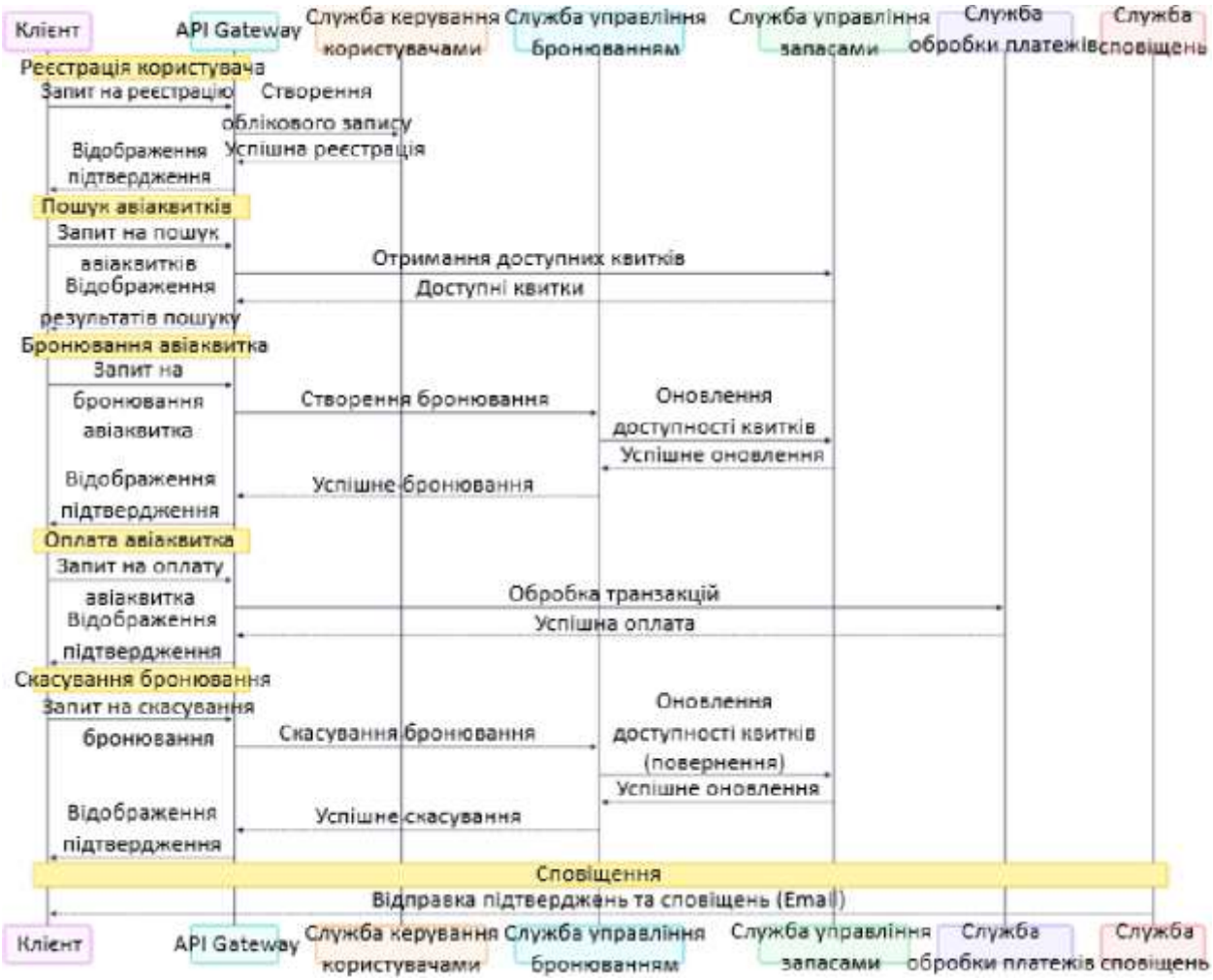


Рисунок 3.9 – Фінальна діаграма послідовності мікросервісної архітектури

Проведений експеримент продемонстрував високу ефективність використання GenAI при проєктуванні інформаційних систем. Штучний інтелект успішно впорався з генерацією технічного завдання, оцінкою бюджету та створенням архітектурних схем у форматах PlantUML та Mermaid.

Ключовим фактором успіху є якість промпт – інжинірингу. На основі отриманих результатів сформульовано рекомендації для ефективної роботи з GenAI:

– використовувати чіткі директиви («напиши», «склади»);

– уникати подвійних заперечень та довгих неоднозначних формулювань;

– застосовувати ітеративний підхід, не намагаючись вирішити всю задачу одним запитом.

Використання GenAI дозволяє значно скоротити час на розробку документації та візуалізацію архітектури, забезпечуючи при цьому відповідність стандартам та можливість швидкого внесення змін.

3.4 Оцінка економічної ефективності прийнятих рішень

Даний підрозділ присвячений проведенню комплексного кількісного порівняльного аналізу та економічного обґрунтування впровадження розробленої методики GAID. Метою дослідження є демонстрація того, як технічна оптимізація процесів проєктування трансформується у вимірювані фінансові показники порівняно з традиційними, усталеними підходами до розробки інформаційних систем. В умовах сучасного ринку, де швидкість виведення продукту є критичним фактором, оцінка ефективності нових інструментів виходить за межі суто технічних метрик, тому основними завданнями цього етапу є аналіз трудовитрат на виконання типових проєктних задач, розрахунок прямих та непрямих витрат, визначення ключових економічних показників, зокрема коефіцієнта повернення інвестицій та терміну окупності, а також формування обґрунтованого висновку про доцільність інтеграції GenAI як основного інструментарію архітектора та бізнес-аналітика.

Для забезпечення об’єктивності аналізу порівнюються два принципово відмінні підходи до проєктування інформаційної системи, розглянуті на прикладі розробленої у цьому розділі ІС бронювання квитків. Перший з них, визначений як «Підхід А» (Традиційний), базується на класичних методологіях, що передбачають ручне виконання більшості когнітивних та

рутинних операцій. Натомість альтернативний «Підхід Б» (Оптимізований/GAID) впроваджує парадигму інтелектуального партнерства, де генеративний ШІ виступає в ролі «ко-пілота» архітектора. Цей сценарій передбачає використання великих мовних моделей для автоматизованого створення чернеток документації, генерацію коду діаграм за принципом «Diagrams as Code» з використанням інструментів PlantUML або Mermaid на основі текстових промптів, а також використання технології RAG для контекстуалізації вимог. Впровадження такого підходу дозволяє змістити роль людини з рутинного створення контенту на стратегічну верифікацію та управління архітектурою, суттєво скорочуючи час ітерацій проєктування. Ключові технічні та організаційні відмінності між цими підходами, що мають безпосередній вплив на вартість та швидкість розробки, систематизовані та представлені у табл. 3.1.

Таблиця 3.1 – Порівняння характеристик підходів до проєктування

Характеристика	Підхід А (Традиційний)	Підхід Б (Оптимізований / GAID)
Створення вимог (SRS)	Ручне написання, тривалий брейнштормінг	Генерація структури та наповнення за шаблонами
Моделювання (UML)	Візуальне малювання (Drag-and-drop)	Генерація коду (Diagrams as Code)
Архітектура БД	Ручне створення ER-діаграм та DDL	Автоматична генерація DDL на основі вимог
Внесення змін	Трудомістке перемальовування схем	Швидка регенерація коду діаграм

Економічна оцінка ефективності запропонованої методики базується на розрахунку прямих витрат на оплату праці кваліфікованого персоналу, оскільки в інтелектуаломістких процесах, таких як проєктування інформаційних систем, саме людський ресурс є основною статтею бюджету. Вхідними параметрами для проведення розрахунків прийнято середньоринкову годинну ставку фахівця рівня Middle/Senior (P_{hour}) яка, за результатами аналізу ринку праці в IT-секторі, становить 1261,8 грн/год. Ця сума враховує не лише чисту заробітну плату, а й супутні податкові та

адміністративні навантаження. Додатковою складовою витрат для «Підходу Б» є вартість використання спеціалізованого інструментарію (C_{infra}) а саме – щомісячна підписка на сервіс генеративного штучного інтелекту (рівня ChatGPT Plus або Gemini Advanced) у розмірі 841,2 грн/міс, що є необхідною умовою для доступу до найбільш потужних мовних моделей. Детальний порівняльний розрахунок часу, фактично витраченого на виконання ідентичних етапів проєктування за обома підходами, систематизовано та наведено в табл. 3.2.

Таблиця 3.2 – Розрахунок трудовитрат та вартості етапів проєктування

Етап проєктування	Підхід А (годин)	Підхід Б (годин)
Аналіз предметної області та формування ТЗ	12	2,5
Розробка Use Cases та сценаріїв користувача	8	1,5
Побудова діаграм (Use Case, Sequence, Component)	10	2
Проєктування схеми БД та API	6	1
Верифікація та виправлення помилок	4	3
Разом годин (T)	40	10
Вартість праці ($C = 30T$), грн	50472	12618

На основі отриманих даних розрахуємо загальні витрати на виконання одного проєкту (C_{total}). Для традиційного підходу (C_A) вони складають:

$$C_A = 40 \cdot 1261{,}8 = 50472 \text{ грн.}$$

Для оптимізованого підходу (C_B), з урахуванням вартості підписки, витрати становлять:

$$C_B = (10 \cdot P_{hour}) + C_{infra}. \quad (3.1)$$

Ця стаття витрат складає ≈ 13460 грн.

Економія на реалізації одного проєкту визначається за виразом:

$$E_{proj} = C_A - C_B. \quad (3.1)$$

Відповідно, вона дорівнює 37012,8 грн, що свідчить про скорочення витрат у 3,75 рази.

Для визначення ефективності інвестицій розрахуємо термін окупності (Payback Period) та ROI. Початкові інвестиції (I_0) на впровадження методики (розробка шаблонів промптів, налаштування середовища, навчання) оцінено

у 20 робочих годин, що становить 25236 грн. Термін окупності інвестицій становить:

$$PP = \frac{I_0}{E_{proj}} = \frac{25236}{37012,8} \approx 0,68.$$

Це означає, що витрати на впровадження методики окупаються ще до завершення першого пілотного проєкту.

Коефіцієнт повернення інвестицій (ROI) для одного проєкту складає:

$$ROI = \frac{880 - 600}{600} \cdot 100\% \approx 46,6\%.$$

При річному горизонті (наприклад, виконання 10 проєктів) ROI зростає до 1366%, оскільки початкові інвестиції є разовими.

Окрім прямої фінансової вигоди, впровадження методики забезпечує низку якісних переваг. По-перше, мінімізується «Concept Gap» завдяки швидкій візуалізації вимог, що знижує ризик дорогих переробок. По-друге, використання підходу «Diagrams as Code» спрощує підтримку документації, адже оновлення схем відбувається через редагування тексту, а не перемальовування графіки. По-третє, знижується поріг входу для фахівців рівня Junior/Middle, які можуть створювати якісні артефакти під наглядом, оптимізуючи структуру команди.

Проведений розрахунок підтвердив економічну доцільність впровадження методики. Автоматизація рутинних операцій дозволила скоротити трудовитрати на 75 %, а високі показники ROI свідчать про значний економічний потенціал запропонованого підходу в реальних умовах розробки.

Висновки до розділу 3

У третьому розділі було проведено детальне проєктування інформаційної системи бронювання квитків із залученням технологій

генеративного штучного інтелекту, що дозволило на практиці реалізувати концепцію AI-Augmented Software Engineering. Використання GenAI як інструмента підтримки дозволило автоматизувати підготовку технічної документації, зокрема формування технічного завдання та розрахунок бюджету проєкту, а застосування методології «ланцюжка думок» сприяло глибокому аналізу вимог та уникненню логічних помилок ще на початкових етапах.

Процес моделювання системи базувався на стратегії «від загального до конкретного», що дало змогу не лише визначити основні сценарії використання, а й врахувати специфічні ситуації, такі як обробка помилок під час транзакцій. Впровадження підходу «Diagrams as Code» із використанням інструментарію PlantUML та Mermaid забезпечило автоматичну візуалізацію вимог та архітектури, дозволяючи зберігати діаграми як код та легко їх актуалізувати без ручного перемальовування.

Ключовим досягненням етапу проєктування стала розробка та оптимізація мікросервісної архітектури системи. Додавання ізольованого сервісу для роботи з клієнтськими даними дозволило підвищити безпеку та відповідність стандартам захисту інформації. Отримані результати засвідчили, що інтеграція GenAI значно прискорює процес проєктування, проте вимагає ретельної експертної верифікації кожного згенерованого артефакту.

ВИСНОВКИ

У рамках даної роботи було проведено комплексне дослідження можливостей підвищення ефективності проєктування інформаційних систем шляхом впровадження технологій генеративного штучного інтелекту. Аналіз теоретичних основ автоматизації проєктування дозволив виявити суттєві обмеження традиційних підходів, таких як CASE – технології, MDA та Low – code/No – code платформи. Встановлено, що ці інструменти ефективно працюють з уже формалізованими даними, але не здатні допомагати на ранніх, творчих етапах розробки, коли відбувається інтерпретація нечітких вимог замовника. Цей феномен у роботі було визначено як «концептуальну прогалину» в наявних засобах автоматизації.

Дослідження показало, що генеративний штучний інтелект, зокрема великі мовні моделі на архітектурі Transformer, здатен закрити цю прогалину завдяки своїй здатності розуміти природну мову та генерувати новий контент, включаючи програмний код і технічну документацію. Завдяки механізму уваги (Self-Attention) такі моделі можуть аналізувати складні залежності у вимогах та коді, що недоступно для попередніх поколінь автоматизованих систем. Однак стихійне використання цих технологій несе серйозні ризики, серед яких найбільш критичними є «галюцинації» або фактична некоректність відповідей, генерація вразливого коду та потенційний витік конфіденційної інформації при використанні публічних хмарних сервісів.

Для вирішення цих проблем та систематизації використання ШІ у роботі було розроблено методику GAID (GenAI-Assisted Iterative Design). В основі цієї методики лежить парадигма інтелектуального партнерства, де штучний інтелект виступає не як заміна людині, а як асистент або «другий пілот». Фундаментальним принципом методики є обов’язкова участь людини в циклі розробки (Human-in-the-Loop). Експерт-проєктувальник залишається центральним елементом процесу, виконуючи функції валідації, корекції та

прийняття остаточних рішень, що дозволяє нівелювати ризики помилок генерації.

Запропонована методика базується на ітеративному підході, що складається з циклів запиту, генерації, верифікації та корекції артефактів. Важливим елементом розробленого підходу є використання технології генерації з доповненим пошуком (RAG), яка передбачає надання моделі доступу до актуальної документації та стандартів проєкту. Це дозволяє «заземлювати» відповіді моделі на реальних фактах, значно підвищуючи їх точність та релевантність конкретному контексту розробки. Впровадження методики призводить до трансформації ролей у команді, де бізнес – аналітики та архітектори зміщують фокус з рутинного створення документів на їх редагування та перевірку, що дозволяє приділяти більше часу вирішенню складних архітектурних завдань.

Практична апробація розробленої методики була проведена на прикладі проєктування інформаційної системи бронювання квитків. Експеримент охопив усі ключові етапи життєвого циклу створення системи, починаючи від аналізу вимог і завершуючи деталізацією архітектури. На етапі ініціалізації проєкту за допомогою генеративного ШІ було автоматично створено детальне технічне завдання, яке враховувало не лише функціональні потреби, а й специфічні вимоги до безпеки згідно з ГОСТ Р 56939 – 2024 та налаштування міжмережевих екранів.

Далі було продемонстровано ефективність ШІ у ресурсному плануванні. Система змогла миттєво згенерувати детальний кошторис проєкту з розбивкою по етапах аналізу, розробки, тестування та впровадження, а також розрахувати орієнтовні терміни реалізації. Це підтверджує, що GenAI може бути потужним інструментом для попередньої оцінки проєктів, значно прискорюючи процеси бюджетування.

Особливу увагу в практичній частині було приділено візуальному моделюванню. Використовуючи підхід генерації коду для діаграм, вдалося створити комплекс моделей у форматах PlantUML та Mermaid. Було успішно

згенеровано діаграми прецедентів, які відображали ролі користувачів та адміністраторів, діаграми компонентів для відображення структури системи, а також діаграми послідовності для деталізації процесів бронювання та оплати. Такий підхід дозволив уникнути трудомісткого ручного малювання схем, замінивши його швидкою генерацією та редагуванням текстових описів.

На етапі архітектурного проєктування методика довела свою гнучкість. Початковий варіант системи був автоматично декомпозований з монолітної структури на набір мікросервісів, що відповідають за пошук, бронювання та оплату. У процесі ітеративного уточнення архітектури за допомогою коригувальних промптів було виявлено та виправлено архітектурні недоліки, зокрема видалено спільну базу даних для мікросервісів та додано окремий сервіс для роботи з клієнтськими даними.

Результати роботи дозволяють стверджувати, що застосування генеративного штучного інтелекту в проєктуванні інформаційних систем є ефективним способом підвищення продуктивності розробників. Запропонована методика GAID забезпечує структурований, безпечний та контрольований процес використання можливостей ШІ, перетворюючи його на надійного асистента, здатного брати на себе рутинні завдання. Це дозволяє команді проєкту зосередитися на стратегічно важливих аспектах створення якісних програмних продуктів.

Таким чином, результатами роботи є методика застосування генеративного ШІ для підтримки процесів проєктування інформаційних систем; рекомендації щодо інтеграції генеративного ШІ у практику проєктування інформаційних систем. Вони можуть бути використані для автоматизації рутинних етапів розробки, підвищення ефективності роботи команд розробників та подальших досліджень за даною тематикою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems (NeurIPS)*. 2020. Vol. 33. URL: https://arxiv.org/abs/2005.11401 (дата звернення: 01.09.2025).

2. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol: O'Reilly Media, 2020. 432 p.

3. Грицай В. І., Корченко О. Г. Захист інформації в мережах передачі даних: підручник. Київ: КПІ ім. Ігоря Сікорського, 2020. 432 с.

4. Литвин В. В., Висоцька В. А. Інтелектуальні системи: підручник. Львів: "Новий Світ-2000", 2020. 406 с.

5. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd Edition. Sebastopol: O'Reilly Media, 2021. 614 p.

6. Ford N., Richards M., Sadalage P. Software Architecture: The Hard Parts. Sebastopol: O'Reilly Media, 2021. 468 p.

7. Wei J., Wang X., Schuurmans D. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*. 2022. Vol. 35. URL: https://arxiv.org/abs/2201.11903 (дата звернення: 12.09.2025).

8. Ouyang L., Wu J., Jiang X. et al. Training language models to follow instructions with human feedback (RLHF). *Advances in Neural Information Processing Systems*. 2022. Vol. 35. URL: https://arxiv.org/abs/2203.02155 (дата звернення: 14.09.2025).

9. NIST AI 100-1. Artificial Intelligence Risk Management Framework (AI RMF 1.0). *Gaithersburg: National Institute of Standards and Technology*, 2023. URL: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf (дата звернення: 16.09.2025).

10. OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*. 2023. URL: https://arxiv.org/abs/2303.08774 (дата звернення: 18.09.2025).

11. White J., Fu Q., Hays S. et al. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *arXiv preprint arXiv:2302.11382*. 2023. URL: https://arxiv.org/abs/2302.11382 (дата звернення: 20.09.2025).

12. Touvron H., Martin L., Stone K. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*. 2023. URL: https://arxiv.org/abs/2307.09288 (дата звернення: 21.09.2025).

13. ISO/IEC 42001:2023. Information technology – Artificial intelligence – Management system. ISO, 2023. URL: https://www.iso.org/standard/81230.html (дата звернення: 23.09.2025).

14. OWASP. OWASP Top 10 for Large Language Model Applications. Version 1.1. 2023. URL: https://owasp.org/www-project-top-10-for-large-language-model-applications/ (дата звернення: 25.09.2025).

15. Shappee M. The Impact of Generative AI on Software Development Lifecycle. *Forbes Technology Council*. 2023. URL: https://www.forbes.com/sites/forbestechcouncil/2023/06/05/the-impact-of-generative-ai-on-software-development-lifecycle/ (дата звернення: 27.09.2025).

16. Liu P., Yuan W., Fu J. et al. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Computing Surveys*. 2023. Vol. 55, Issue 9. URL: https://dl.acm.org/doi/10.1145/3560815 (дата звернення: 29.09.2025).

17. Google. Gemini: A Family of Highly Capable Multimodal Models. Google DeepMind Technical Report. 2024. URL: https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf (дата звернення: 01.10.2025).

18. Anthropic. The Claude 3 Model Family: Opus, Sonnet, Haiku. *Anthropic Model Card*. 2024. URL: https://www-cdn.anthropic.com/files/4b39b56/Model-Card-Claude-3.pdf (дата звернення: 03.10.2025).

19. Meta AI. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*. 2024. URL: https://arxiv.org/abs/2407.21783 (дата звернення: 05.10.2025).

20. Zhang Y., Jin L., Chen Z. A Comprehensive Survey on Generative AI for Software Engineering. *arXiv preprint arXiv:2402.04353*. 2024. URL: https://arxiv.org/abs/2402.04353 (дата звернення: 07.10.2025).

21. Hou X., Zhao Y., Liu Y. et al. Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv preprint arXiv:2308.10620*. 2024. URL: https://arxiv.org/abs/2308.10620 (дата звернення: 09.10.2025).

22. Regulation (EU) 2024/1689. Laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). *Official Journal of the European Union. 2024*. URL: https://eur-lex.europa.eu/eli/reg/2024/1689/oj (дата звернення: 11.10.2025).

23. Корченко О.Г., Іванченко В.О. Методи та засоби захисту інформаційних ресурсів в умовах використання штучного інтелекту. *Кібербезпека: освіта, наука, техніка*. 2023. № 4 (20). С. 45-58.

24. Vasylenko V., Lytvyn V. Architecture of decision support systems based on Large Language Models. *Proceedings of the 4th International Workshop on Intelligent Information Technologies & Systems of Information Security* (IntelITSIS). 2023. URL: https://ceur-ws.org/Vol-3373/ (дата звернення: 15.10.2025).

25. PlantUML Language Reference Guide. PlantUML Official Documentation. Version 1.2025. URL: https://plantuml.com/guide (дата звернення: 18.10.2025).

26. Mermaid User Guide. Mermaid.js Documentation. Version 11.0. 2025. URL: https://mermaid.js.org/intro/ (дата звернення: 20.10.2025).

27. OpenAPI Specification (OAS). Version 3.1.0. The Linux Foundation. URL: https://spec.openapis.org/oas/v3.1.0 (дата звернення: 22.10.2025).

28. Spring Boot Reference Documentation. Version 3.3. VMware, Inc. 2025. URL: https://docs.spring.io/spring-boot/index.html (дата звернення: 24.10.2025).

29. Docker Documentation. Docker Inc. 2025. URL: https://docs.docker.com/ (дата звернення: 25.10.2025).

30. Kubernetes Documentation. The Linux Foundation. 2025. URL: https://kubernetes.io/docs/home/ (дата звернення: 27.10.2025).

31. PostgreSQL 17 Documentation. The PostgreSQL Global Development Group. 2025. URL: https://www.postgresql.org/docs/current/index.html (дата звернення: 29.10.2025).

32. React Documentation. Meta Platforms. 2025. URL: https://react.dev/reference/react (дата звернення: 30.10.2025).

33. Мулько А. Формування вимог засобами штучного інтелекту в процесі проєктування інформаційної системи. *Сучасні аспекти та перспективні напрямки розвитку науки: матеріали X Міжнародної студентської наукової конференції* (жовтень 2025 р. м. Луцьк), 2025. С. 242, 243.

34. Мулько А. Вплив сучасних інтелектуальних технологій на ефективність проєктування інформаційних систем. *Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXII щорічного міждисциплінарного семінару* (листопад 2025 р., м. Полтава), 2025. С. 79, 80.

35. GitHub Copilot Documentation. *GitHub, Inc*. 2025. URL: https://docs.github.com/en/copilot (дата звернення: 05.11.2025).

36. Mistral AI. *Mistral Large Model Card*. 2024. URL: https://mistral.ai/news/mistral-large/ (дата звернення: 08.11.2025).

37. Stack Overflow Developer Survey 2024. AI Tools and Technologies. URL: https://survey.stackoverflow.co/2024/ (дата звернення: 10.11.2025).

38. Google Cloud. *Generative AI on Google Cloud Documentation*. 2025. URL: https://cloud.google.com/ai/generative-ai (дата звернення: 12.11.2025).

39. Microsoft. Azure OpenAI Service Documentation. 2025. URL: https://learn.microsoft.com/en-us/azure/ai-services/openai/ (дата звернення: 15.11.2025).

40. Prompt Engineering Guide. *DAIR.AI*. 2025. URL: https://www.promptingguide.ai/ (дата звернення: 18.11.2025).

41. Bozic J. Testing Artificial Intelligence Systems: Models and Algorithms. *IEEE Transactions on Software Engineering*. 2024. Vol. 50, Issue 2. URL: https://ieeexplore.ieee.org/document/ (дата звернення: 20.11.2025).