НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра інформаційних технологій в телекомунікаціях

До захисту допущено:
В.о. завідувача кафедри
_____ Валерій ПРАВИЛО
«_____»_____2026 р

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційно-комунікаційні технології»
спеціальності 172 Електронні комунікації та радіотехніка

на тему: Модифікована система управління проєктами в Jira з використанням технологій штучного інтелекту

Виконав: здобувач вищої освіти 4 курсу, групи ТІ-22
Сабіров Олег Едуардович _____

Науковий керівник: професор кафедри ІТТ НН ІТС, доктор технічних наук, доцент Астраханцев Андрій Анатолійович _____

Рецензент: доцент кафедри фізико-технічних засобів захисту інформації, доктор технічних наук Прогонов Д.О. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут телекомунікаційних систем

Кафедра інформаційних технологій в телекомунікаціях

Рівень вищої освіти – перший (бакалаврський)
Освітньо-професійна програма «Інформаційно-комунікаційні технології»
спеціальності 172 Електронні комунікації та радіотехніка

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Валерій ПРАВИЛО
«_____» _____2026 р

ЗАВДАННЯ

на дипломну роботу студенту

Сабірову Олегу Едуардовичу

1. Тема дипломної роботи: Модифікована система управління проєктами в Jira з використанням технологій штучного інтелекту, науковий керівник роботи професор кафедри ІТТ НН ІТС, доктор технічних наук, доцент Астраханцев Андрій Анатолійович, затверджені наказом по університету від 26.05.2026 р. № 1912-с
2. Строк подання студентом дипломної роботи 09.06.2026 р.
3. Об’єкт дослідження: Процеси обробки користувацьких звернень та управління сервісними заявками в системах класу IT Service Management, зокрема в Jira, що охоплюють створення, супровід, зміну станів, пошук, аналіз та автоматизацію роботи із заявками на основі використання великих мовних моделей.

4. Вихідні дані до роботи: Документація Jira REST API, документація Spring AI, матеріали щодо використання великих мовних моделей та механізму виклику інструментів (tool calling), специфікація протоколу контексту моделі (MCP), документація локальної великої мовної моделі Qwen 2.5 та платформи Ollama, методи обробки природної мови, підходи до реалізації контекстної пам’яті та AI-агентів, методи виявлення конфіденційної інформації в тексті, тестові дані сервісних заявок, а також програмні засоби розробки серверного застосунку на платформі Java із використанням Spring Boot.
5. Перелік завдань, які потрібно розробити: Провести аналіз сучасних підходів до автоматизації сервісних процесів та використання великих мовних моделей у системах підтримки управління заявками; дослідити архітектурні особливості Jira та механізми взаємодії через REST API; проаналізувати принципи використання протоколу контексту моделі (MCP), виклику інструментів та контекстної пам’яті у складі AI-агентів; виконати обґрунтування вибору локальної великої мовної моделі Qwen 2.5 та програмного каркасу Spring AI; розробити архітектуру AI-системи підтримки управління сервісними заявками; реалізувати інтеграцію великої мовної моделі з Jira; реалізувати механізми створення заявок, отримання даних, фільтрації, аналітики та зміни статусів через природномовний інтерфейс; розробити модуль перевірки вхідних даних на наявність конфіденційної інформації; реалізувати механізми контекстної пам’яті та MCP-орієнтованої взаємодії через STDIO і виклик інструментів; провести тестування функціональних можливостей системи та оцінити ефективність розробленого рішення.
7. Дата видачі завдання: 10.09.2025

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Узгодження організаційних питань з науковим керівником	10.09.25-25.09.25	Виконано
2	Узгодження теми роботи та початок роботи над першим розділом	26.09.25-15.10.25	Виконано
3	Робота над першим, другим та третім розділами	16.10.25-10.03.26	Виконано
4	Робота над четвертим розділом. Планування розробки власного рішення	11.03.26-15.05.26	Виконано
5	Оформлення дипломної роботи	16.05.26-01.06.26	Виконано
6	Підготовка презентації до захисту	02.06.26-05.06.26	Виконано

Здобувач вищої освіти _____ Олег САБІРОВ

Науковий керівник роботи _____ Андрій АСТРАХАНЦЕВ

РЕФЕРАТ

Дипломна робота містить 88 сторінок, 27 рисунків, 2 таблиці та 22 використаних джерел.

У сучасних організаціях значна частина процесів управління проєктами та сервісними зверненнями реалізується за допомогою спеціалізованих інформаційних систем. Однією з найбільш поширених платформ такого класу є Jira яка забезпечує управління задачами, сервісними заявками, процесами виконання та підтримку взаємодії між учасниками проєкту. Водночас збільшення кількості звернень, обсягів текстової інформації та складності бізнес-процесів призводить до зростання навантаження на користувачів системи та ускладнює виконання типових операцій.

Одним із перспективних напрямів підвищення ефективності роботи з корпоративними інформаційними системами є використання великих мовних моделей. Сучасні LLM здатні аналізувати природномовні запити, визначати намір користувача, формувати структуровані параметри та виконувати взаємодію із зовнішніми системами через механізм виклику інструментів. Це створює передумови для побудови інтелектуальних агентів, здатних автоматизувати роботу із задачами та сервісними зверненнями без необхідності безпосередньої взаємодії користувача з графічним інтерфейсом інформаційної системи.

Метою роботи є підвищення ефективності системи управління сервісними процесами шляхом розробки системи підтримки на основі штучного інтелекту.

У межах роботи проведено аналіз сучасних підходів до інтеграції великих мовних моделей з інформаційними системами, досліджено

механізми роботи Jira та розроблено програмну систему на основі Java, Spring Boot і Spring AI. Реалізована архітектура забезпечує інтеграцію локальної великої мовної моделі Qwen 2.5 з Jira через REST API та підтримує механізм виклику інструментів для створення, пошуку, фільтрації та оновлення заявок.

У розробленій системі реалізовано механізми аналізу намірів користувача, контекстної пам’яті діалогу, підтримки багатокрокових сценаріїв взаємодії, аналітичної обробки даних сервісних звернень, а також модуль перевірки чутливої інформації перед передачею даних до Jira. Додатково реалізовано підтримку протоколу контексту моделі (MCP), що забезпечує інтеграцію AI-асистента із зовнішніми клієнтськими застосунками [9].

Об’єктом дослідження є процеси управління проєктами та сервісними зверненнями в інформаційних системах підтримки проєктної діяльності.

Предметом дослідження є методи та засоби інтеграції великих мовних моделей із системами управління проєктами, механізми виклику інструментів, обробки природної мови та автоматизації роботи із задачами Jira.

Наукова новизна роботи полягає у розробці архітектури AI-асистента для Jira, що поєднує використання локальної великої мовної моделі, механізму виклику інструментів, контекстної пам’яті та контролю процесів виконання заявок. Запропонований підхід забезпечує автоматизовану взаємодію з Jira через природну мову без необхідності прямої роботи користувача з REST API або графічним інтерфейсом системи.

Практичне значення роботи полягає у створенні програмної системи, яка автоматизує виконання типових операцій із сервісними заявками, спрощує взаємодію користувачів з Jira, скорочує час обробки звернень та підвищує ефективність управління сервісними процесами.

Ключові слова: JIRA, ВЕЛИКІ МОВНІ МОДЕЛІ, LLM, QWEN, SPRING AI, ШТУЧНИЙ ІНТЕЛЕКТ, ВИКЛИК ІНСТРУМЕНТІВ, ПРОТОКОЛ КОНТЕКСТУ МОДЕЛІ, MCP, REST API, УПРАВЛІННЯ ПРОЄКТАМИ, SERVICE DESK, ОБРОБКА ПРИРОДНОЇ МОВИ, AI-АСИСТЕНТ, ІНФОРМАЦІЙНА БЕЗПЕКА.

ABSTRACT

This thesis contains 88 pages, 27 figures, 2 tables and 22 references.

Modern organizations increasingly rely on information systems to support project management and service operations. Jira is one of the most widely used platforms for managing service requests, workflows and operational processes. However, the growing volume of service tickets, textual information and business process complexity increases the workload on users and complicates the execution of routine operations.

One of the most promising approaches to improving human-system interaction is the integration of Large Language Models (LLMs) with enterprise information systems. Modern LLMs are capable of understanding natural language requests, identifying user intent, extracting structured parameters and interacting with external systems through tool calling mechanisms. This enables the development of intelligent assistants that automate routine operations and simplify access to system functionality.

The aim of this work is to improve the efficiency of service process management systems through the development of an artificial intelligence-based support system.

The study includes the analysis of modern approaches to integrating large language models with enterprise information systems, investigation of Jira architecture and development of a software solution based on Java, Spring Boot and Spring AI. The implemented architecture integrates the local Qwen 2.5 large language model with Jira through REST API and supports tool calling for creating, searching, filtering, updating and analysing service requests.

The developed system implements user intent recognition, conversational memory, multi-step interaction scenarios, workflow transition

management and analytical processing of service desk data. In addition, a security module has been implemented to detect sensitive information before transmitting data to Jira. Support for the Model Context Protocol (MCP) is also provided to enable integration with external AI clients.

The object of the research is project management and service request processing within enterprise information systems.

The subject of the research is methods and technologies for integrating large language models with project management systems, including natural language processing, tool calling, conversational memory and workflow automation in Jira.

The scientific novelty of the work lies in the development of an AI assistant architecture for Jira that combines a local large language model, tool calling mechanisms, conversational memory and workflow-aware task processing. The proposed approach enables natural language interaction with Jira without requiring direct interaction with REST APIs or graphical user interfaces.

The practical significance of the work lies in the development of a software system that automates service request management, simplifies user interaction with Jira, reduces manual workload, improves operational efficiency and provides an additional layer of information security.

Keywords: JIRA, ARTIFICIAL INTELLIGENCE, LARGE LANGUAGE MODELS, LLM, QWEN, SPRING AI, MODEL CONTEXT PROTOCOL, MCP, TOOL CALLING, NATURAL LANGUAGE PROCESSING, REST API, SERVICE DESK, PROJECT MANAGEMENT, WORKFLOW AUTOMATION, INFORMATION SECURITY.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AI – Artificial Intelligence

PMIS – Project Management Information System

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

LLM – Large Language Model

MCP – Model Context Protocol

NLP – Natural Language Processing

OAuth – Open Authorization

REST – Representational State Transfer

JQL — Jira Query Language

SQL – Structured Query Language

UI – User Interface

JPA — Java Persistence API

OWASP — Open Web Application Security Project

PMIS — Project Management Information System

RAG — Retrieval-Augmented Generationє

TTL — Time To Live

URL — Uniform Resource Locator

UUID — Universally Unique Identifier

ВСТУП

У сучасних організаціях значна частина процесів управління проєктами, сервісними зверненнями та внутрішніми бізнес-процесами реалізується за допомогою спеціалізованих інформаційних систем. Однією з найбільш поширених платформ такого класу є Jira, яка забезпечує управління задачами, сервісними заявками, процесами виконання та взаємодію між учасниками проєкту. Водночас зі збільшенням кількості звернень, складності процесів та обсягів текстової інформації зростає навантаження на користувачів системи, що ускладнює виконання рутинних операцій і знижує ефективність роботи.

Стрімкий розвиток технологій штучного інтелекту, зокрема великих мовних моделей (LLM), відкриває нові можливості для автоматизації взаємодії користувача з корпоративними інформаційними системами. Сучасні мовні моделі здатні аналізувати природномовні запити, визначати намір користувача, використовувати контекст попередньої взаємодії та виконувати прикладні операції через механізм виклику інструментів. Це дозволяє створювати інтелектуальних асистентів, які спрощують роботу з інформаційними системами та зменшують потребу у безпосередній взаємодії з графічним інтерфейсом.

Актуальність роботи обумовлена необхідністю підвищення ефективності управління проєктами та сервісними процесами шляхом інтеграції великих мовних моделей із Jira. Використання природної мови як основного інтерфейсу взаємодії дозволяє автоматизувати створення, пошук, оновлення та аналіз заявок, скоротити час виконання типових операцій та підвищити зручність роботи користувачів.

Метою роботи є підвищення ефективності системи управління сервісними процесами шляхом розробки системи підтримки на основі штучного інтелекту.

Об’єктом дослідження є процеси управління проєктами та сервісними зверненнями в інформаційних системах підтримки проєктної діяльності.

Предметом дослідження є методи та засоби інтеграції великих мовних моделей із системами управління проєктами, механізми обробки природної мови, виклику інструментів та автоматизації роботи із задачами Jira.

Наукова новизна роботи полягає у розробці архітектури AI-асистента для Jira, що поєднує використання локальної великої мовної моделі, механізму виклику інструментів, контекстної пам’яті та контролю процесів виконання заявок. Запропонований підхід забезпечує виконання операцій над задачами за допомогою природномовних запитів без необхідності прямої взаємодії користувача з REST API або графічним інтерфейсом системи.

Практичне значення роботи полягає у створенні програмної системи, яка автоматизує роботу із сервісними заявками, спрощує взаємодію користувачів з Jira, скорочує час виконання рутинних операцій та підвищує ефективність управління сервісними процесами.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ПРОЄКТАМИ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Принципи управління проєктами

Методології управління проєктами зазнали значної еволюції протягом останнього століття під впливом змін бізнес-середовища, розвитку технологій та зростання складності проєктів. Ефективне управління проєктами є критично важливим для успіху будь-якої діяльності, оскільки забезпечує виконання проєктів у встановлені строки, в межах бюджету та відповідно до визначених стандартів якості. У прагненні підвищити ефективність, адаптивність і задоволеність зацікавлених сторін, вибір відповідної методології управління проєктами стає ключовим управлінським рішенням. Двома найбільш поширеними методологіями сучасного управління проєктами є Waterfall та Agile.

Методологія Waterfall, одна з найстаріших моделей управління проєктами, являє собою лінійний і послідовний процес (рис 1.1). Вперше вона була запропонована у 1970 році Вінстоном Ройсом, який описав поетапний підхід до розробки програмного забезпечення, де кожен етап має бути завершений перед переходом до наступного (Royce, 1970). Waterfall характеризується акцентом на попередньому плануванні, детальній документації та жорстко визначених фазах, де результати одного етапу стають основою для наступного. Її часто називають прогнозною моделлю, оскільки вона передбачає наявність чітко визначених вимог на початку проєкту, які залишаються стабільними протягом усього життєвого циклу.

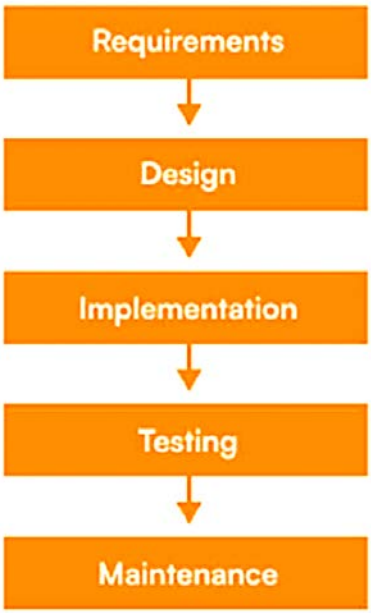


Рисунок 1.1. - Waterfall методологія

У зв’язку з цим виникла потреба у більш гнучкому підході, що призвело до появи Agile-методології на початку 2000-х років. Agile сформувався як відповідь на обмеження традиційних лінійних моделей, зокрема у сфері розробки програмного забезпечення. Agile Manifesto, опублікований у 2001 році, підкреслює важливість гнучкості, співпраці та орієнтації на потреби замовника [10]. На відміну від Waterfall, Agile базується на ітеративному та інкрементному підході, де проєкт поділяється на короткі цикли (спринти), тривалістю зазвичай 2-4 тижні (рис.1.2)

У кожній ітерації створюється, тестується та оцінюється функціональний результат. Agile передбачає постійний зворотний зв’язок із зацікавленими сторонами, що дозволяє швидко реагувати на зміни вимог або умов ринку. Методології Agile, зокрема Scrum та Kanban, набули широкого застосування не лише в IT, але й у фінансах, медицині, маркетингу та дизайні продуктів.

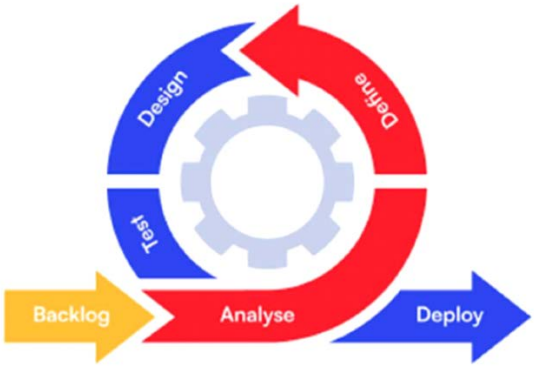


Рисунок 1.2. - Agile методологія

1.2 Agile-методології управління проєктами

Agile-методології включають різні методології та практики, серед яких найбільш поширеними є Scrum, Kanban.

Методологія Scrum є одним із найбільш формалізованих підходів Agile-управління проєктами, що орієнтований на реалізацію складних та динамічних систем [11]. В основі Scrum лежить ітераційно-інкрементальний підхід, відповідно до якого процес розробки поділяється на короткі часові інтервали – спринти. Кожен спринт завершується створенням функціонально завершеного інкременту продукту, який може бути оцінений зацікавленими сторонами (рис.1.3).

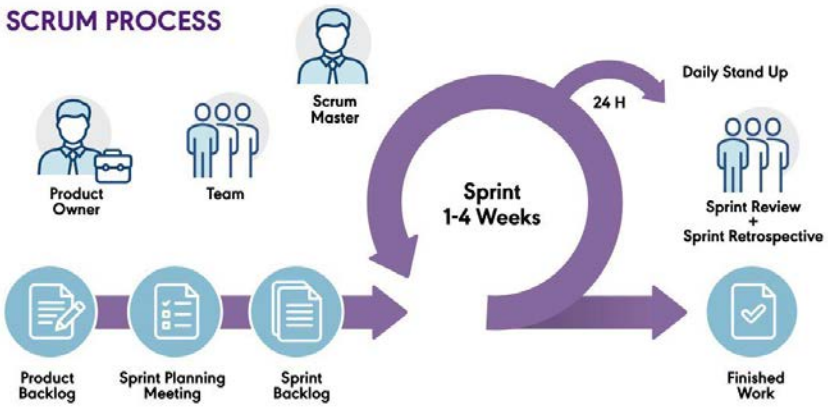


Рисунок 1.3. - Схема процесу Scrum-методології управління проєктами

Методологія Kanban, на відміну від Scrum, реалізує потоковий підхід до управління завданнями та не передбачає фіксованих ітерацій (рис1.4). Основною метою Kanban є оптимізація потоку виконання робіт шляхом візуалізації процесів і обмеження кількості задач у роботі (Work In Progress, WIP).

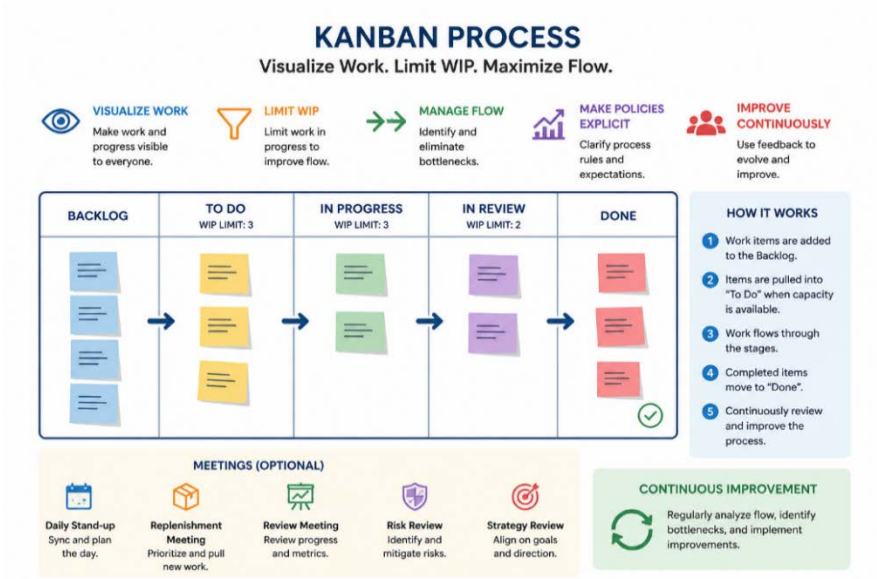


Рисунок 1.4. - Схема процесу Kanban-методології управління проєктами

У межах Scrum усі задачі виконуються в рамках спринтів тривалістю від 2 до 4 тижнів. Протягом кожного спринту проводяться основні події: планування (Sprint Planning), щоденні зустрічі (Daily Scrum), огляд результатів (Sprint Review) та ретроспектива (Sprint Retrospective). Як правило, такі зустрічі є короткими та не перевищують 15 хвилин.

Вибір конкретної Agile-методології значною мірою залежить від особливостей бізнес-процесів організації та необюхідних параметрів (табл. 1.1). Для команд розробки програмного забезпечення частіше використовується Scrum, який забезпечує контроль виконання робіт у межах спринтів. Натомість у системах управління сервісними зверненнями та технічною підтримкою переважає Kanban-підхід, оскільки він дозволяє ефективно працювати з безперервним потоком заявок та оперативно реагувати на нові інциденти.

Таблиця 1.1 - Порівняння параметрів методологій Scrum та Kanban.

№	Параметр	Scrum	Kanban
1	Наявність журналу задач	Так	ні
2	Великі проєкти	Ні	так
3	Пріоритет – люди та взаємодія	Так	ні
4	Пріоритет – процеси та інструменти	Ні	так
5	Ітерації фіксованої тривалості (спринти)	Так	ні
6	Безперервний процес	Ні	так
7	Випуск після кожного спринту	Так	ні
8	Безперервний потік задач	Ні	так
9	Чітко визначені ролі (Scrum Master, PO, команда)	Так	ні
10	Гнучка структура ролей	Ні	так
11	Основний показник – швидкість команди	Так	ні
12	Основний показник – час виконання	Ні	так
13	Зміни під час ітерації небажані	Так	ні
14	Зміни можливі в будь-який момент	Ні	так

1.3 Системи управління проєктами

PMIS є ключовим елементом успішної реалізації проєктів, оскільки забезпечує підтримку прийняття управлінських рішень, планування, організацію та контроль виконання проєктних робіт.

Серед великої кількості програмних рішень особливе місце займає Jira, яка є однією з найбільш поширених систем управління проєктами у сфері інформаційних технологій.

Результати дослідження використання систем управління проєктами серед IT-фахівців показують, що саме Jira займає провідні позиції за рівнем використання (рис.1.5). Це підтверджується тим, що значна частина респондентів обирає саме цей інструмент для щоденної роботи, що свідчить про його високу адаптованість до потреб сучасних команд.

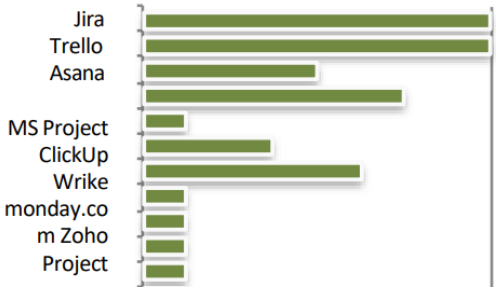


Рисунок 1.5. Розподіл використання систем управління проєктами

Таким чином, Jira є однією з найбільш ефективних і широко використовуваних систем управління проєктами, що забезпечує гнучкість, масштабованість та можливості інтеграції, необхідні для сучасних IT-проєктів. Це робить її оптимальною платформою для впровадження інтелектуальних систем управління, зокрема AI-асистентів.

1.4 Дані та процеси в системах управління проєктами

Процес роботи з даними в PMIS починається зі збору та введення даних. Система отримує інформацію про задачі (терміни виконання, залежності та відповідальних осіб), ресурси (бюджети, обладнання та доступність персоналу) та прогрес виконання (звіти про статус або облік часу). Це дозволяє сформувати єдине інформаційне середовище, що відображає поточний стан проєкту.

Після збору дані організовуються та зберігаються у структурованому вигляді, наприклад у вигляді панелей управління, діаграм та звітів. Таке представлення дозволяє отримати як загальну картину проєкту, так і детальний аналіз окремих його складових. Наприклад, діаграми Ганта відображають виконання задач, а фінансові панелі дозволяють контролювати використання бюджету.

Важливою складовою PMIS є аналіз даних і формування звітності. Система обробляє зібрані дані та генерує інформацію, необхідну для прийняття управлінських рішень. Типові результати включають звіти про прогрес, аналіз розподілу ресурсів і оцінку ризиків. Це дозволяє керівникам проєктів своєчасно виявляти проблеми та оптимізувати виконання робіт.

Окрім роботи з даними, PMIS підтримує процеси управління через механізми співпраці та комунікації. Учасники проєкту можуть обмінюватися інформацією, документами та коментарями безпосередньо в системі, що забезпечує узгодженість дій і покращує координацію на всіх етапах життєвого циклу проєкту.

1.5 Висновок

У цьому розділі висвітлено теоретичні основи управління проєктами та функціонування інформаційних систем, що забезпечують організацію проєктної діяльності. Виділено ключові принципи управління проєктами, а також розглянуто сучасні підходи, зокрема Agile-методології.

Проаналізовано системи управління проєктами як програмні інструменти, що підтримують планування, виконання та контроль задач, а також забезпечують координацію роботи команд. Окрему увагу приділено даним і процесам, які формують основу функціонування таких систем, зокрема збору, обробці та аналізу інформації, а також реалізації процесів виконання робіт. Встановлено, що інтеграція даних і процесів у межах сучасних систем, таких як Jira, дозволяє підвищити ефективність управління проєктами та забезпечити прозорість виконання робіт.

РОЗДІЛ 2

ТЕХНОЛОГІЇ ШТУЧНОГО ІНТЕЛЕКТУ В УПРАВЛІННІ ПРОЄКТАМИ

2.1 Великі мовні моделі (LLM)

Великі мовні моделі (LLM) є обчислювальними моделями на основі штучного інтелекту (AI), призначеними для розуміння та генерації тексту, подібного до людського. Завдяки мільярдам параметрів навчання, LLM демонструють високу ефективність у виявленні складних мовних закономірностей, що забезпечує їхню значну результативність у широкому спектрі завдань обробки природної мови (NLP) [15].

Після впровадження архітектури трансформерів (рис 2.1) великі мовні моделі почали суттєво впливати на індустрію завдяки своїм можливостям генерації тексту. Ранні моделі, такі як T5 і mT5, використовували трансферне навчання, тоді як GPT-3 продемонструвала можливість виконання завдань у режимі zero-shot без додаткового донавчання [14, 16]. LLM здатні точно відповідати на запити, коли їм надаються відповідні інструкції та приклади, однак їх ефективність значно підвищується після донавчання на інструкціях .

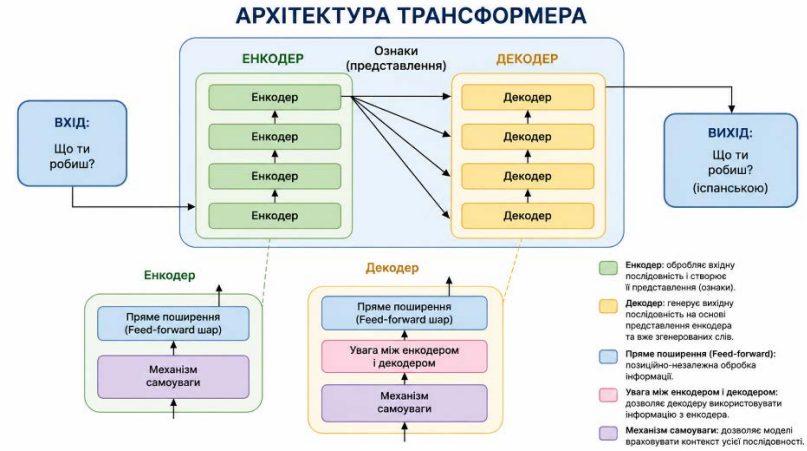


Рисунок 2.1. - Архітектура великої мовної моделі на основі трансформера

Окрім здатності до узагальнення та адаптації, LLM демонструють емерджентні можливості, такі як міркування, планування, прийняття рішень і навчання в контексті. Ці властивості виникають завдяки великому масштабу моделей і дозволяють використовувати їх у складних інтелектуальних задачах.

Завдяки своїм можливостям LLM забезпечують значний прогрес у різних галузях шляхом автоматизації процесів, підвищення точності та надання глибших аналітичних висновків. Водночас їх використання супроводжується певними викликами, зокрема етичними питаннями, упередженістю навчальних даних, значними обчислювальними витратами та високими вимогами до ресурсів.

Таким чином, великі мовні моделі є потужним інструментом сучасного штучного інтелекту, що забезпечує нові можливості для автоматизації, аналізу та підтримки прийняття рішень, одночасно вимагаючи подальших досліджень для вирішення існуючих обмежень і забезпечення ефективного та безпечного використання.

2.2 Інтеграція LLM з інформаційними системами

Інтеграція великих мовних моделей з інформаційними системами базується на використанні програмних інтерфейсів, що забезпечують взаємодію між користувачем і системою через обробку природної мови. У цьому випадку запити користувача, сформульовані у текстовому вигляді, автоматично перетворюються на структуровані команди, які можуть бути виконані в системі управління проєктами, зокрема в Jira. Такий підхід дозволяє значно спростити доступ до функціональних можливостей системи та підвищити рівень автоматизації процесів.

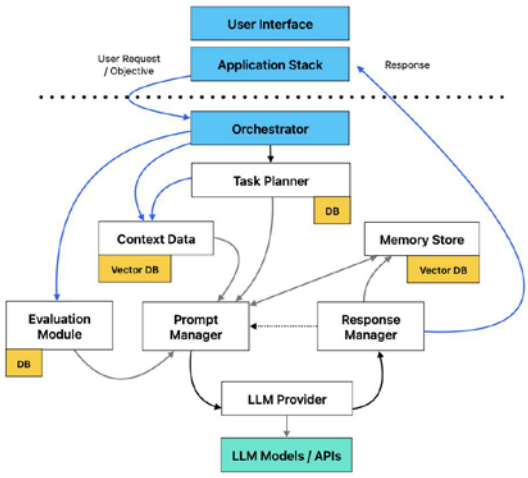


Рисунок 2.2. - Архітектура інтеграції LLM з інформаційною системою управління проєктами

Крім того, мовні моделі можуть виконувати функцію проміжного програмного шару, який поєднує наміри користувача з виконанням конкретних дій у системі. Вони інтерпретують запит, визначають необхідну операцію та ініціюють її виконання, що дозволяє реалізувати більш природну взаємодію з інформаційними системами.

Формально вибір інструменту можна подати як задачу вибору найбільш імовірної дії з множини доступних функцій:

$$T^* = \arg max_{T_i \in T} P(T_i|U, C),$$

де T^* – *обраний інструмент,* T_i *-окремий інструмент із множини доступних інструментів, U - запит користувача, C – контекст взаємодії*

При використанні штучного інтелекту в процесах управління проєктами є низка переваг (табл.2.1) .У випадку інтеграції LLM з Jira основною перевагою є функція виклику інструментів, тобто є можливість підключення великих мовних моделей до актуальних джерел даних. Це особливо важливо для корпоративних систем, у яких інформація постійно змінюється.

Таблиця 2.1 - Переваги використання штучного інтелекту в управлінні

Перевага	**Вплив на галузі управління проєктами**
Підвищення точності оцінки тривалості завдань	Управління інтеграцією, управління графіком
Зменшення ризиків проєкту	Управління інтеграцією, управління ризиками, управління якістю, управління зацікавленими сторонами
Оптимізація розподілу ресурсів	Управління інтеграцією, управління ресурсами
Забезпечення видимості процесів у реальному часі	Управління інтеграцією, управління ресурсами, управління ризиками, управління якістю, управління комунікаціями, управління графіком, управління зацікавленими сторонами
Покращення комунікації між учасниками проєкту	Управління інтеграцією, управління комунікаціями, управління зацікавленими сторонами
Автоматизація адміністративних процесів	Управління інтеграцією, управління витратами, управління ресурсами, управління комунікаціями, управління графіком
Підвищення гнучкості проєкту	Управління інтеграцією, управління обсягом, управління ресурсами, управління комунікаціями, управління графіком, управління зацікавленими сторонами
Підтримка ініціації нових проєктів	Управління інтеграцією, управління обсягом, управління ризиками, управління графіком, управління зацікавленими сторонами

На відміну від статичних відповідей моделі, виклик зовнішньої функції дозволяє отримати поточні дані безпосередньо з прикладної системи [17]. Крім того, такий підхід зменшує потребу у великій кількості жорстко заданих правил, оскільки вибір дії виконується на основі змісту запиту, а не лише через наперед прописані сценарії.

Для вибору великої мовної моделі в межах нашого проєкту було проведено порівняльний аналіз декількох сучасних локальних LLM, які підтримують запуск через платформу Ollama та можуть використовуватися у задачах виклику інструментів і взаємодії із зовнішніми API. Оцінювання виконувалося за критеріями якості виконання інструкцій, підтримки багатомовності, ефективності виклику інструментів, швидкодії локального

виконання, вимог до апаратних ресурсів та придатності для побудови AI-агентів. За результатами аналізу найбільшу сумарну оцінку отримала модель Qwen 2.5, яка продемонструвала найкращі результати у задачах визначення наміру користувача, формування параметрів виклику інструментів та роботи зі складними сценаріями робочих процесів. Саме тому дана модель була обрана як основа AI-рівня розробленої системи

2.3 Протокол контексту моделі (MCP)

Розвиток великих мовних моделей та AI-агентів призвів до необхідності створення уніфікованих механізмів взаємодії між моделлю, зовнішніми інструментами та інформаційними системами. У класичних підходах інтеграція великої мовної моделі з прикладним середовищем зазвичай реалізовується через окремі програмні модулі, власні формати передачі даних та індивідуальні механізми виклику функцій. Така архітектура ускладнює масштабування системи, збільшує кількість залежностей між компонентами та ускладнює підтримку AI-рішень у корпоративному середовищі.

У сучасних AI-архітектурах для розв’язання цієї проблеми використовується протокол контексту моделі (MCP), який забезпечує стандартизований механізм передачі контексту між великою мовною моделлю, інструментами та зовнішніми джерелами даних [9]. Основна ідея MCP полягає у відокремленні AI-рівня від конкретної реалізації зовнішніх сервісів. У такій архітектурі модель працює не з безпосередньою реалізацією API, а зі структурованим контекстом взаємодії, який містить опис доступних функцій, параметрів виклику, результатів попередніх операцій та історії взаємодії (рис.2.3).

Подібний підхід використовується у сучасних системах, де велика мовна модель повинна враховувати не лише поточний запит користувача, а й попередній стан системи, результати викликів функцій та доступний набір інструментів. Контекст у такому випадку стає окремим елементом архітектури, який забезпечує зв’язність багатокрокової взаємодії та підтримку складних сценаріїв роботи AI-систем. Саме тому у сучасних дослідженнях системи оркестрації ШІ контекст розглядається як одна з ключових складових побудови інтегрованих AI-платформ.

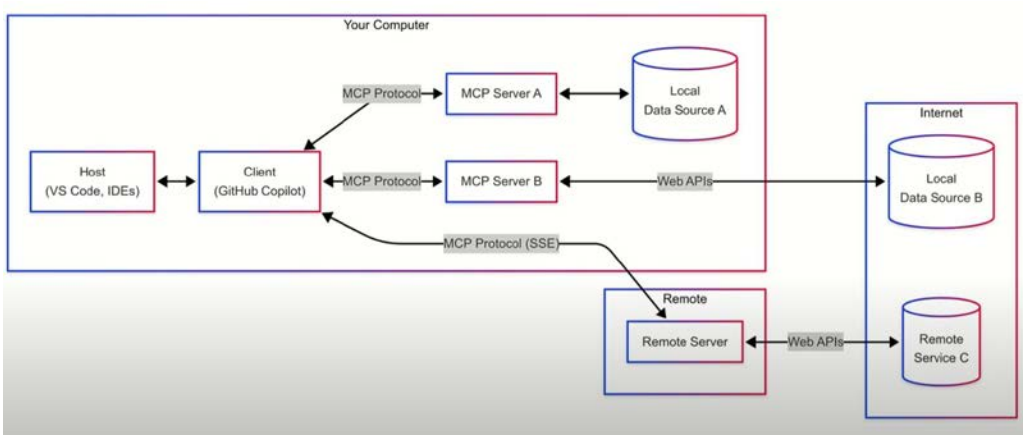


Рисунок 2.3. - Узагальнена схема взаємодії через MCP

Такий підхід дозволяє системі враховувати попередні дії під час формування наступного кроку взаємодії. У результаті велика мовна модель працює не ізольовано, а як компонент багаторівневої інформаційної системи.

Однією з ключових переваг використання MCP є зменшення зв’язності між AI-рівнем та зовнішнім програмним середовищем. Модель не залежить від конкретної реалізації API або внутрішньої архітектури прикладної системи. У разі зміни зовнішнього сервісу достатньо оновити опис доступних функцій без зміни логіки роботи AI-моделі. Це особливо важливо для корпоративних систем, де інтеграційна інфраструктура постійно змінюється.

Саме тому сучасні AI-системи, побудовані з використанням MCP, зазвичай поєднують механізми передачі контексту із засобами перевірки

параметрів, журналювання дій, контролю доступних операцій та валідації зовнішніх викликів.

2.4 Питання інформаційної безпеки при використанні AI

Використання штучного інтелекту в інформаційних системах супроводжується появою нових викликів у сфері інформаційної безпеки, що обумовлено складністю, динамічністю та залежністю таких систем від великих обсягів даних (рис.2.4). На відміну від традиційних IT-рішень, AI-системи мають більш складну структуру та можуть функціонувати на основі зовнішніх моделей, API та навчальних наборів даних, що підвищує рівень ризику.

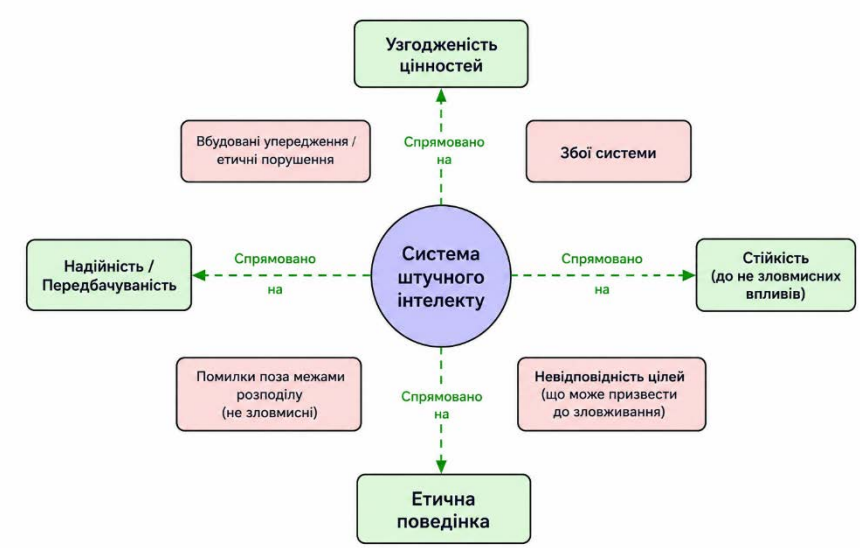


Рисунок 2.4. - Основні аспекти безпеки та ризики систем ШІ

Узагальнення сучасних досліджень дозволяє виділити основні групи загроз інформаційної безпеки, які виникають при використанні штучного інтелекту [19]:

- Викриття навчальних даних.
- Атаки інверсії моделі.

- Незахищені точки доступу.
- Тіньове використання штучного інтелекту.
- Надмірні права доступу.
- Інтеграція сторонніх сервісів.

У контексті управління ризиками доцільно також розмежовувати поняття Безпека ШІ та Інформаційна безпека ШІ. Перше спрямоване на запобігання ненавмисним помилкам та збоям системи, тоді як друге орієнтоване на протидію навмисним атакам та захист даних. Такий підхід дозволяє більш ефективно формувати стратегії захисту залежно від характеру загроз.

У межах AI захищеності основна увага приділяється запобіганню ненавмисній шкоді, що може виникати внаслідок внутрішніх недоліків системи або некоректного формулювання цілей. Основною метою є забезпечення того, щоб система діяла відповідно до очікувань користувача та не створювала небажаних наслідків. Важливими аспектами цього напряму є узгодженість із людськими цінностями, що передбачає відповідність поведінки AI намірам користувача, а також стійкість до змін середовища, які можуть впливати на якість результатів. Окрему роль відіграє забезпечення прозорості роботи моделей, що дозволяє підвищити рівень довіри до них та спростити процес виявлення помилок.

Крім того, значна увага приділяється зменшенню етичних ризиків, пов’язаних із упередженістю моделей, що може призводити до дискримінаційних або некоректних рішень. Для мінімізації таких ризиків застосовуються підходи, спрямовані на контроль якості даних, а також впровадження механізмів перевірки результатів. Важливим елементом є наявність механізмів аварійного зупинення або обмеження роботи системи, що дозволяє запобігти поширенню помилкових результатів у разі виникнення критичних ситуацій.

У свою чергу, AI захист орієнтований на захист системи від навмисних атак та зловмисного використання. Основними загрозами у цьому випадку є адверсарні атаки, спрямовані на зміну поведінки моделі, отруєння навчальних даних, що призводить до викривлення результатів, а також крадіжка моделей або доступ до конфіденційної інформації. У зв’язку з цим застосовуються методи захисту, що включають перевірку даних, шифрування, контроль доступу та постійний моніторинг активності.

2.5 Висновок

У цьому розділі розглянуто технології штучного інтелекту для автоматизації управління проєктами. Досліджено принципи роботи великих мовних моделей на основі архітектури трансформера та підходи до їх інтеграції з інформаційними системами, зокрема з Jira. Встановлено, що ключовим механізмом інтеграції є виклик інструментів, який дозволяє моделі звертатися до актуальних даних і виконувати операції на основі змісту запиту. За результатами порівняльного аналізу локальних моделей обрано Qwen 2.5, а для стандартизованої передачі контексту між моделлю та зовнішніми сервісами розглянуто протокол контексту моделі (MCP).

Окрему увагу приділено інформаційній безпеці AI-систем: виділено основні групи загроз та визначено обмеження, пов'язані з контролем доступу до зовнішніх функцій і ризиком виконання небажаних операцій. Таким чином, розглянуті технології створюють теоретичне підґрунтя для постановки задачі та проєктування архітектури AI-системи підтримки управління проєктами, що розглядається в наступному розділі.

РОЗДІЛ 3
АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

3.1 Постановка задачі та послідовність дослідження

3.1.1 Постановка задачі для дослідження

Основною метою даної роботи є підвищення ефективності системи управління сервісними процесами шляхом розробки та дослідження можливостей інтелектуального AI-асистента для підтримки діяльності керівника проєкту, який інтегрується з системою управління проєктами Jira та використовує великі мовні моделі для автоматизації процесів обробки інформації, аналізу задач і формування управлінських рішень.

Завдання бакалаврської роботи полягає у створенні підходу до використання технологій штучного інтелекту для підвищення ефективності управління проєктами, зокрема шляхом автоматизації рутинних операцій, аналізу текстових даних, генерації звітності та підтримки прийняття рішень у межах робочих процесів. Особлива увага приділяється інтеграції AI-рішень у існуючі інформаційні системи, що дозволяє забезпечити їх практичне застосування в умовах реального середовища.

Для досягнення поставленої мети необхідно вирішити ряд взаємопов’язаних задач, що включають аналіз функціональних можливостей системи Jira та її інтеграційного потенціалу, дослідження сучасних підходів до використання великих мовних моделей у інформаційних системах, визначення вимог до AI-асистента з урахуванням специфіки управління проєктами, а також розробку архітектури системи, яка забезпечує взаємодію між користувачем, AI-моделлю та платформою Jira.

Крім того, важливою складовою дослідження є визначення механізмів обробки запитів користувачів, реалізація автоматизації робочих процесів, а також врахування аспектів інформаційної безпеки та контролю доступу до даних. Це дозволяє забезпечити не лише функціональність системи, але й її надійність та безпечне використання у корпоративному середовищі.

3.1.2. Послідовність виконання дослідження

Для досягнення поставленої мети дослідження було визначено послідовність дій, спрямованих на розробку та реалізацію AI-асистента для підтримки управління проєктами з інтеграцією в систему Jira та використанням великих мовних моделей

1. Етап аналізу первинної інформації:
   - Дослідження принципів управління проєктами та особливостей використання системи Jira
   - Аналіз сучасних AI-рішень та можливостей великих мовних моделей (LLM)
   - Визначення основних задач, які можуть бути автоматизовані за допомогою AI
2. Етап дослідження інтеграції AI з інформаційними системами:
   - Аналіз підходів до інтеграції LLM через API та інші механізми взаємодії
   - Дослідження можливостей обробки запитів користувачів у природній мові
   - Визначення способів взаємодії між AI-асистентом та системою Jira
3. Етап формування вимог до системи:
   - Визначення функціональних можливостей AI-асистента

- Формування нефункціональних вимог, зокрема щодо безпеки та продуктивності
- Визначення обмежень системи та умов її використання

4. Етап реалізації функціональних можливостей:
   - Розробка загальної архітектури AI-системи
   - Визначення взаємодії між компонентами: користувачем, LLM та Jira
   - Проєктування механізмів обробки запитів та автоматизації
5. Етап проєктування архітектури системи:
   - Реалізація створення задач у Jira за допомогою AI
   - Реалізація оновлення статусів задач
   - Отримання та обробка списку задач
   - Реалізація аналітики та генерації звітності
6. Етап реалізації механізмів безпеки:
   - Забезпечення контролю доступу до системи
   - Реалізація захисту даних при взаємодії з AI
   - Врахування ризиків, пов’язаних із використанням LLM
7. Етап тестування системи:
   - Проведення тестування основних сценаріїв використання
   - Аналіз коректності роботи AI-асистента
   - Оцінка ефективності автоматизації процесів
8. Етап аналізу результатів:
   - Оцінка досягнутих результатів
   - Визначення переваг та обмежень розробленої системи
   - Формування рекомендацій щодо подальшого розвитку

3.1.3 Обмеження та припущення системи

Розглянемо основні обмеження та припущення, що визначають умови функціонування AI-асистента та впливають на ефективність його використання у середовищі управління проєктами.

Технологічні обмеження

Великі мовні моделі (LLM) – використовуються як основний інструмент обробки запитів, проте функціонують на основі ймовірнісних алгоритмів, що може призводити до генерації неточних або неповних результатів.

Продуктивність системи – обробка запитів потребує значних обчислювальних ресурсів, що може впливати на швидкість відповіді та стабільність роботи в режимі реального часу.

Інтеграційні обмеження

Інтеграція через API – взаємодія з Jira та AI-сервісами здійснюється через зовнішні інтерфейси, що створює залежність від їх доступності та стабільності.

Доступ до даних Jira – визначається ролями та правами користувачів, що обмежує обсяг інформації, доступної для обробки.

Обмеження безпеки

Інформаційна безпека – передача даних до AI-сервісів може створювати ризики витоку конфіденційної інформації.

Контроль доступу – недостатньо налаштовані права доступу можуть призводити до несанкціонованого використання системи або витоку даних.

Функціональні обмеження

Залежність від запитів користувача – якість роботи системи безпосередньо залежить від коректності та повноти сформульованих запитів.

Обмеженість контексту – система працює в межах наданих даних і не завжди може враховувати всі фактори проєктного середовища.

Припущення щодо системи

Користувач системи – передбачається, що користувач має базові навички роботи з Jira та розуміє логіку формування запитів.

Дані системи – вважається, що дані, отримані з Jira, є коректними, актуальними та придатними для обробки.

Середовище функціонування – система працює у стабільному мережевому середовищі з постійним доступом до зовнішніх сервісів.

3.2 Аналіз архітектури Jira

3.2.1. Організація процес виконання задач

Оновлення задач у Jira пов’язане з життєвим циклом заявки та виконується відповідно до заздалегідь налаштованої моделі процесу виконання (workflow). На відміну від звичайного редагування полів задачі, зміна статусу відбувається лише через дозволені переходи між станами, що забезпечує контроль послідовності виконання процесів. На рисунку 3.1 наведено процес виконання-моделі заявки в Jira.

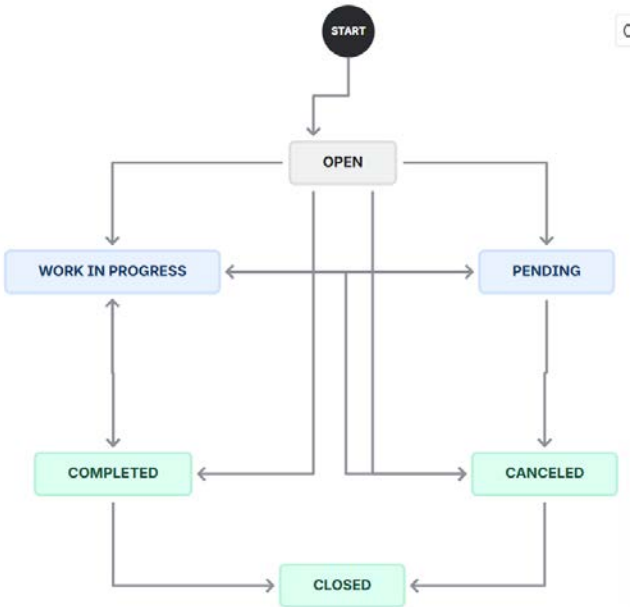


Рисунок 3.1. - Модель життєвого циклу заявки в Jira

Модель починається зі стану Open, який означає створення заявки та готовність до її обробки. У процесі виконання заявка може переходити до статусу Work in Progress, що відображає активне виконання робіт, або до статусу Pending, якщо для продовження роботи необхідно отримати додаткову інформацію чи виконати зовнішні дії. Після завершення робіт заявка переводиться у статус Completed, а остаточне закриття фіксується статусом Closed. Для випадків втрати актуальності або помилкового створення заявки використовується статус Canceled.

З технічної точки зору така модель може бути подана як орієнтований граф:

$$Workflow = (S, T),$$

$$\text{де } S - \text{множина можливих статусів,}$$

$$\text{T} - \text{множина допустимих переходів}$$

У межах процес виконання кожна задача перебуває у певному стані, наприклад Open, In Progress або Done. Для кожного статусу система визначає перелік доступних переходів між статусами, які можуть бути виконані користувачем або зовнішньою системою. Таким чином, процес виконання забезпечує контроль послідовності виконання процесів та запобігає некоректній зміні стану задачі.

У документації Atlassian зазначається, що зміна статусу задачі виконується через кінцеве точку `/rest/api/3/issue/{issueKey}/transitions`, який використовується для отримання списку доступних переходів та подальшого виконання переходу між статусами. Такий підхід забезпечує узгодженість процес виконання та підтримує бізнес-логіку системи [3].

У JSON-відповіді система повертає перелік переходів, доступних для поточного стану задачі відповідно до конфігурації процес виконання. Формування списку переходів виконується динамічно та залежить від поточного статусу задачі, ролі користувача, прав доступу та правил процес виконання-моделі.

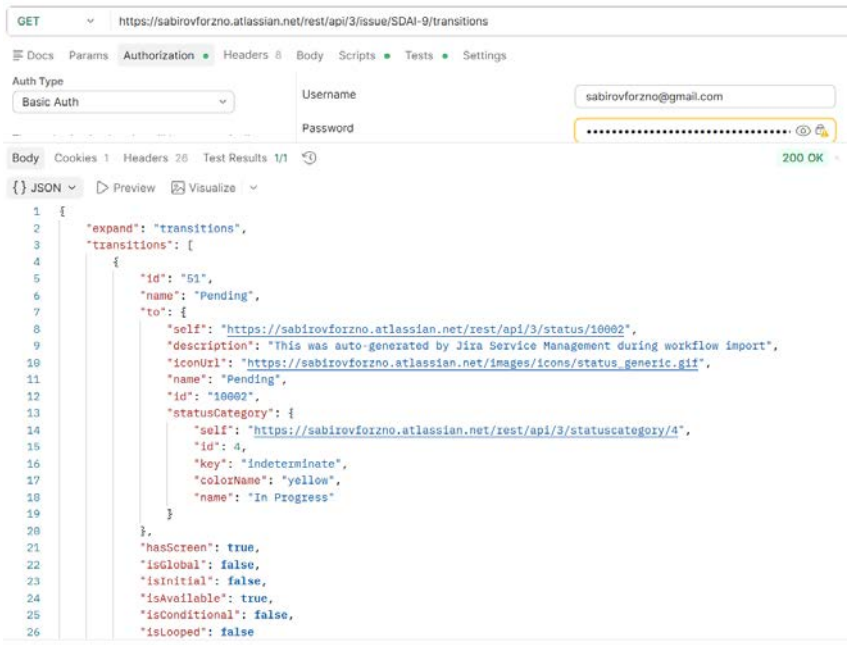


Рисунок 3.2. - Отримання доступних переходів задачі через REST API Jira у середовищі Postman

Після отримання списку всіх переходів зміна статусу задачі виконується через HTTP POST - запит, який передає ідентифікатор необхідного переходу у структурі JSON-запиту. У результаті виконання запиту Jira виконує перевірку допустимості переходу та здійснює зміну стану задачі відповідно до поточної процес виконання-моделі.

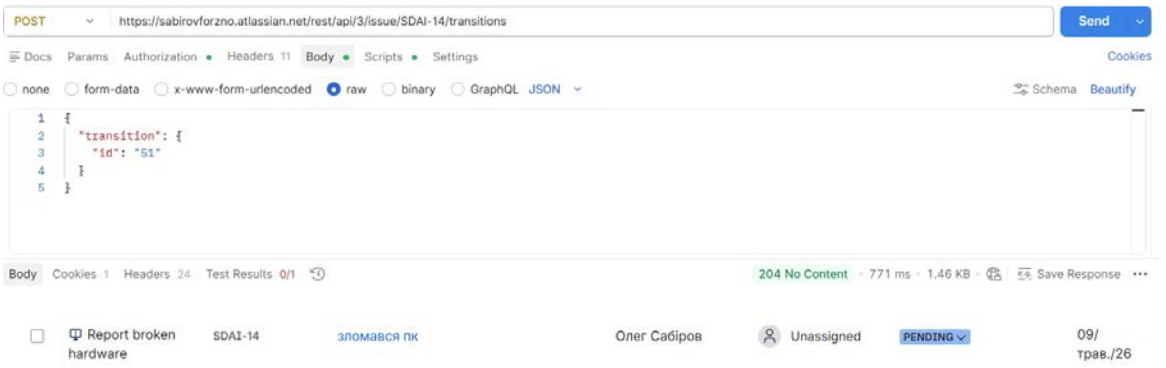


Рисунок 3.3. - Виконання переходу для зміни статусу задачі через REST

У процесі виконання запиту система аналізує поточний стан задачі та перевіряє відповідність переходу правилами процесу виконання. У разі успішного виконання запиту Jira оновлює статус задачі та виконує пов'язані процес виконання-операції, зокрема оновлення метаданих, запуск механізмів автоматизації та обробку службових подій системи.

Використання переходів забезпечує контрольовану зміну станів задачі та підтримує цілісність процесів виконання моделі. На відміну від прямого оновлення поля `status`, механізм переходів дозволяє реалізувати формалізований процес переходу між станами задачі з урахуванням бізнес-логіки системи, прав доступу та конфігурації процесів обробки сервісних заявок.

Jira реалізує не просту зміну значення поля `status`, а контрольований механізм переходів. Це дозволяє уникнути некоректних сценаріїв, наприклад прямого переходу із закритої заявки назад у випадковий проміжний стан без відповідного правила процес виконання.

3.2.2. Прикладний програмний інтерфейс взаємодії через HTTP (REST API) в Jira

REST API у Jira використовується для інтеграції зовнішніх інформаційних систем із функціональними можливостями Jira. Через REST API зовнішній

застосунок може створювати заявки, отримувати інформацію про задачі, виконувати переходи між статусами задачі, а також взаємодіяти з моделлю життєвого циклу задачі.

В офіційній документації Atlassian REST API визначається як механізм інтеграції Jira з іншими програмними системами. Взаємодія здійснюється через HTTP-запити точок доступу, а результати повертаються у форматі JSON. Такий підхід забезпечує стандартизовану взаємодію між зовнішнім застосунком та серверною частиною Jira [3].

Архітектурно REST API виступає проміжним рівнем між клієнтським застосунком та внутрішньою логікою Jira (рис.3.4) . Зовнішня система не працює напряму з базою даних Jira, а надсилає HTTP-запити до відповідних кінцеву точку. У відповідь система повертає структуровані JSON-документи, які містять інформацію про задачі, переходи або результати виконаних операцій.

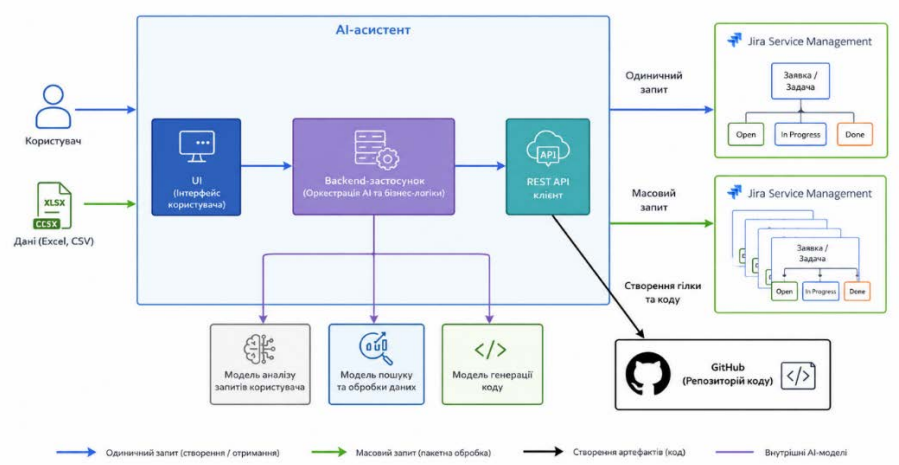


Рисунок 3.4. - Взаємодія AI-системи з Jira через REST API

У Jira REST API основними HTTP-методами є GET та POST. Метод GET використовується для отримання інформації із системи (рис.3.5), тоді як POST застосовується для створення нових об’єктів або виконання операцій над задачами.

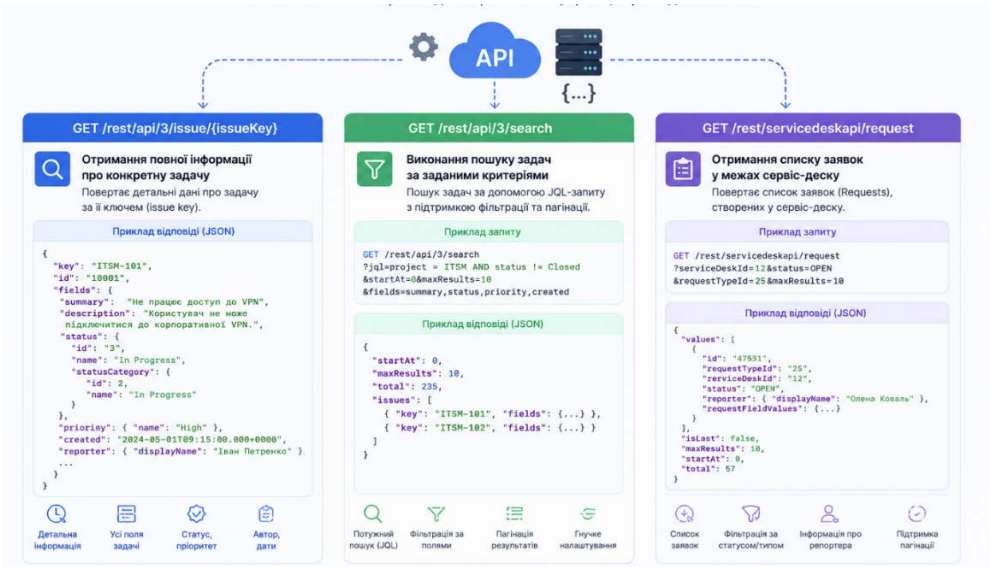


Рисунок 3.5. - Використання REST API Jira для отримання інформації про задачі та заявки

На рисунку 3.5 наведено приклади використання REST API Jira для роботи із задачами та сервісними заявками. Зокрема, кінцева точка `GET /rest/api/3/issue/{issueKey}` використовується для отримання повної інформації про конкретну задачу за її унікальним ключем [3]. У JSON-відповіді містяться основні параметри задачі, зокрема `summary` (короткий опис), `description` (опис проблеми), `status` (поточний статус задачі), `priority` (пріоритет), а також службова інформація про автора та дату створення.

Для виконання пошуку задач у системі використовується кінцева точка `GET /rest/api/3/search`, який підтримує роботу з JQL (Jira Query Language —мова пошуку Jira) [3]. Використання даного кінцева точка дозволяє здійснювати фільтрацію задач за статусом, проєктом, пріоритетом або іншими параметрами. Крім того, REST API підтримує пагінацію результатів та сортування даних, що є важливим при роботі з великою кількістю задач.

У межах Jira окремо використовується кінцева точка `GET /rest/servicedeskapi/request`, призначений для отримання списку

сервісних заявок [4]. Через параметри HTTP-запиту система дозволяє виконувати фільтрацію заявок за статусом, типом запиту або сервісним проєктом. У JSON-відповіді повертається масив заявок та службова інформація щодо кількості результатів і параметрів пагінації.

Важливою особливістю REST API Jira є підтримка роботи з `transitions` (перехід між статусами задачі). Зміна статусу задачі виконується не через пряме оновлення поля `status`, а через окремий `transition`, що забезпечує контроль процес виконання та підтримує логіку бізнес-процесів системи. Для отримання списку доступних переходів використовується кінцева точка `/rest/api/3/issue /{issueKey}/transitions`, після чого система може виконати HTTP POST-запит для зміни стану задачі [3].

REST API Jira використовується як основний механізм автоматизації процес виконання-процесів та інтеграції зовнішніх систем із платформою Jira. Через REST API реалізується інтеграція AI-асистента, автоматизація процесів технічної допомоги, отримання списків задач, виконання переведення статусів та аналітична обробка інформації про заявки.

3.2.3 Отримання та обробка даних

Взаємодія із системою реалізується через REST API, що надає уніфікований механізм доступу до даних як окремих задач, так і вибірок задач за визначеними критеріями.

Кожна задача в Jira може бути представлена як інформаційний об’єкт, що містить унікальний ідентифікатор, набір полів, поточний статус та службові метадані. Така модель забезпечує єдиний підхід до роботи з різними типами задач і створює основу для подальшої обробки інформації.

Важливою можливістю системи є фільтрація даних за статусом, часовими параметрами, текстовими характеристиками та значеннями користувацьких

полів. Формально процес відбору задач можна описати як задачу побудови підмножини елементів, що задовольняють заданим критеріям пошуку. Нехай множина всіх доступних задач системи визначається як (Issues), тоді результат виконання запиту формується відповідно до виразу:

$$Result = \{Issue_i \in Issues \mid C(Issue_i) = true\},$$

де C – предикат відповідності умовам відбору.

У наведеному виразі множина Result формується шляхом застосування предиката C до кожного елемента множини Issues. Предикат виконує роль функції відбору та визначає, чи відповідає конкретна задача встановленим критеріям пошуку. Формально операція фільтрації є відображенням множини задач у її підмножину, що містить лише релевантні елементи. Такий підхід дозволяє описати процес вибірки незалежно від конкретного набору параметрів запиту. У практичній реалізації Jira критерії предиката можуть включати перевірку статусу задачі, виконавця, часових характеристик, пріоритету або інших атрибутів, доступних через REST API.

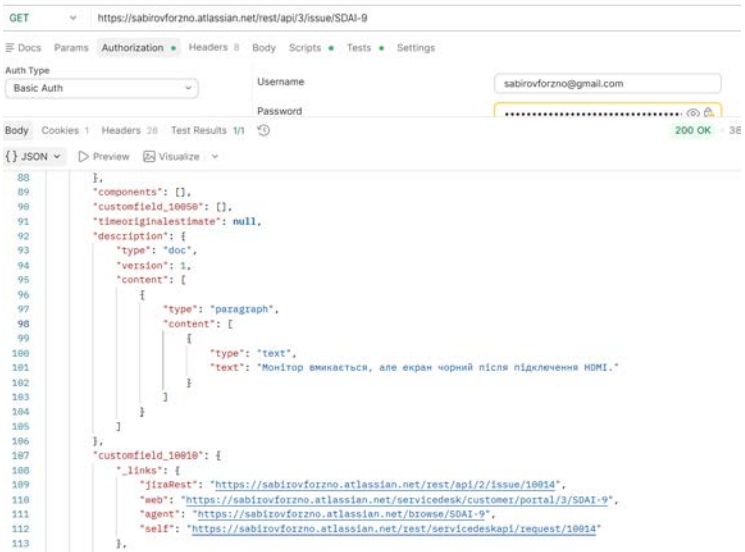


Рисунок 3.6. - Приклад отримання інформації про певну задачу в Jira

Після отримання даних суттєвого значення набуває етап їх обробки, який спрямований на підвищення якості та придатності інформації для подальшого використання. Обробка включає нормалізацію текстових полів, структуризацію неформалізованих описів, усунення неоднозначностей та перетворення даних у формат, зручний для аналізу. Це особливо важливо у випадку роботи з текстовими описами проблем, які можуть містити різні варіації формулювань та потребують уніфікації.

Окрім цього, для отримання узагальнених характеристик системи використовується агрегація даних, яка дозволяє перейти від аналізу окремих задач до оцінки загального стану системи. У процесі агрегації виконується групування задач за певними параметрами, підрахунок їх кількості, визначення частоти виникнення типових проблем та інші статистичні операції. Такий підхід дозволяє виявляти закономірності у функціонуванні системи та формувати аналітичні висновки.

3.2.4 Аналітика та звітність

Аналітика та формування звітності у Jira є завершальним етапом обробки даних, що забезпечує трансформацію первинної інформації про задачі у вигляд, придатний для прийняття управлінських рішень. На відміну від простого зберігання або відображення даних, аналітична підсистема спрямована на виявлення закономірностей, оцінку ефективності процесів та формування узагальнених показників функціонування системи.

Основою аналітики є використання агрегованих даних, отриманих у результаті обробки множини задач. У загальному випадку аналітичну модель можна представити як відображення:

$$Analytics = g(\{Issue1, Issue2, \ldots, Issuen\})$$

де g – функція узагальнення, що виконує обчислення статистичних характеристик на основі набору задач.

Одним із базових аналітичних показників є загальна кількість задач у системі, яка відображає обсяг навантаження на сервіс або проєкт. Разом із цим важливим є розподіл задач за статусами, що дозволяє оцінити поточний стан виконання робіт. Формально це може бути подано як функція підрахунку:

$$f(s) = Count(s) = i = 1 \sum \{status(Issuei) = s\}, \quad \text{де}$$

$$\{status(Issue_i) = s\} = \begin{cases} 1, \text{якщо статус дорівнює параметру} \\ 0, \text{не підходить} \end{cases}$$

Отриманий розподіл задач за статусами дозволяє оцінити поточне завантаження команди та виявити можливі вузькі місця в процесі виконання робіт. Наприклад, надмірна кількість задач у статусі «В роботі» може свідчити про перевантаження виконавців, тоді як велика кількість відкритих задач може вказувати на накопичення нерозглянутих звернень. Для узагальненої оцінки результативності роботи додатково використовується показник частки завершених задач.

На основі цих даних визначається один із ключових показників ефективності – частка завершених задач. Цей показник характеризує рівень виконання робіт та може бути представлений у вигляді відношення:

$$Completion\ Rate = \frac{\mathrm{N_{total}}}{\mathrm{N_{completed}}} \cdot 100\%$$

$$Frequency(term) = |\{Issue_i \mid term \in text(Issue_i)\}|$$

Такий підхід дозволяє виявляти найбільш поширені категорії проблем, типові звернення користувачів та основні напрями навантаження сервісної служби.

Важливим аспектом аналітики є також можливість формування звітів, які відображають стан системи у зручному для сприйняття вигляді. Звіти можуть

включати як числові показники, так і їх візуалізацію у вигляді діаграм або графіків. При цьому джерелом даних виступає REST API, а формування звітів базується на попередньо обробленій та агрегованій інформації.

Рисунок 3.7. - Приклад візуального представлення аналітики в Jira

Крім того, стандартні засоби аналітики орієнтовані переважно на описові показники і не забезпечують автоматичну інтерпретацію отриманих результатів.

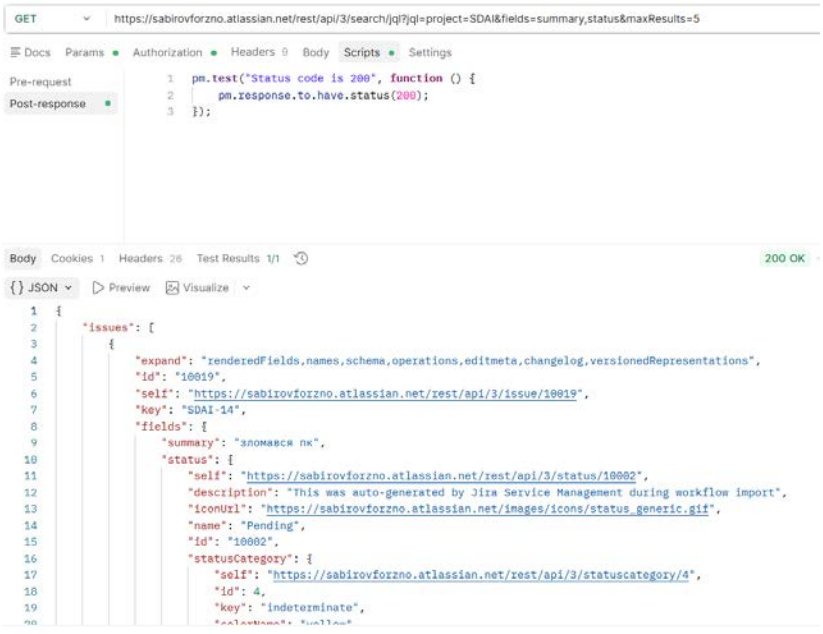


Рисунок 3.8. - Використання JQL для формування аналітичної вибірки задач по статусу Done

Використання JQL та REST API забезпечує можливість формування аналітичних вибірок даних, необхідних для побудови звітності, оцінки навантаження на службу підтримки та реалізації AI-орієнтованих механізмів аналізу задач.

Під час роботи з REST API Jira було враховано особливості актуальної версії API Atlassian. Зокрема, кінцева точка пошуку `/rest/api/3/search` у нових версіях Jira був замінений на `/rest/api/3/search/jql`, що пов’язано з оновленням механізму роботи JQL-запитів та зміною політики підтримки REST API [3, 18].

Такий підхід демонструє необхідність врахування версійності REST API та підтримки сумісності інтеграційних компонентів із поточною версією Jira.

3.3 Аналіз існуючих AI-асистентів для систем управління проєктами

Використання великих мовних моделей дозволяє автоматизувати роботу з текстовою інформацією, зокрема створення описів задач, узагальнення даних, підготовку звітів, аналіз коментарів та підтримку планування проєктної діяльності.

Особливістю проєктних інформаційних систем є значний обсяг неструктурованих даних, представлених у вигляді описів задач, повідомлень користувачів, службових звернень та документації. У таких умовах AI-асистент виступає проміжним інтелектуальним рівнем між користувачем і системою управління проєктами, спрощуючи взаємодію та зменшуючи обсяг ручних операцій.

Типовими функціями сучасних AI-асистентів є створення задач на основі природномовних запитів, автоматичне формування заголовків і описів, пошук інформації, узагальнення стану проєкту та підготовка аналітичних звітів.

Подібні можливості поступово інтегруються у комерційні системи управління проєктами, зокрема в екосистему Atlassian через платформу Rovo.

Разом із тим більшість існуючих рішень орієнтована переважно на генерацію та аналіз тексту. Такі системи не завжди забезпечують повноцінну інтеграцію з моделлю життєвого циклу задач, механізмами переходів та правилами процесу виконання. Це є важливим обмеженням для систем Jira, де коректність виконання операцій визначається налаштованими бізнес-процесами.

Крім того, результати роботи AI-моделей потребують контролю з боку користувача, особливо під час виконання дій, що впливають на стан задач або сервісні показники. Тому перспективним напрямом є використання AI не як автономного засобу прийняття рішень, а як керованого інструмента підтримки роботи користувача.

Отже, аналіз існуючих AI-асистентів показує, що найбільш ефективним є підхід, за якого велика мовна модель поєднується з інструментами інформаційної системи та взаємодіє з ними через прикладні програмні інтерфейси. Саме такий підхід покладено в основу розроблюваної системи підтримки управління проєктами на базі Jira.

3.4 Формування вимог до AI-системи

У випадку AI-асистента для Jira процес формування вимог безпосередньо пов’язаний із особливостями робочих процесів, процес виконання-моделі Jira та використанням великих мовних моделей для обробки природномовних запитів користувача.

Ефективність інтеграції великих мовних моделей у корпоративні інформаційні системи залежить не лише від якості моделі, а й від правильного

визначення функціональних сценаріїв використання, обмежень безпеки, механізмів інтеграції та способів взаємодії з користувачем.

Основною метою розроблюваної AI-системи є автоматизація взаємодії користувача із Jira через природномовний інтерфейс. Система повинна забезпечувати можливість створення задач, отримання списків заявок, зміни статусів, аналітичної обробки інформації та формування відповідей на основі даних середовища служби підтримки.

Однією з базових вимог є підтримка механізму створення сервісних заявок у Jira через природномовний опис проблеми. Система повинна автоматично формувати `summary` (короткий опис задачі), `description` (детальний опис проблеми), а також передавати відповідні параметри до REST API Jira.

Крім створення задач, AI-система повинна підтримувати:

1. отримання списку задач;
2. фільтрацію задач за статусом;
3. пошук задач за ключовими словами;
4. комбіновану фільтрацію;
5. отримання доступних переходів між статусами;
6. зміну статусу задачі;
7. аналітичну обробку інформації;
8. формування статистичних показників.

Важливою функціональною вимогою є підтримка процес виконання-моделі Jira. AI-асистент не повинен виконувати пряме оновлення статусу задачі, а має працювати через механізм переходу відповідно до правил процесів виконання. Для цього система повинна попередньо отримувати перелік доступних переходів через REST API та виконувати лише допустимі переходи між статусами [5, 6, 12].

Система повинна забезпечувати:

1. підтримку REST-орієнтованої архітектури;
2. обробку JSON-відповідей Jira;
3. стабільну роботу HTTP кінцева точка;
4. підтримку механізму виклику інструментів;
5. контроль коректності процес виконання-переходів;
6. підтримку аналітичної обробки даних.

Модуль безпеки повинен:

1. виконувати аналіз тексту користувача;
2. виявляти `password, token, api_key` та інші чутливі дані;
3. блокувати передачу секретної інформації у Jira;
4. забезпечувати контроль взаємодії із зовнішнім API.

У контексті архітектури система реалізується як серверна частина-застосунок на основі Java та Spring Boot із використанням Spring AI для інтеграції великої мовної моделі. Взаємодія із Jira виконується через REST API, а обмін даними реалізується у форматі JSON через HTTP-запити.

Таким чином, формування вимог до AI-системи базується на поєднанні функціональних можливостей Jira, механізмів інтеграції REST API, процес виконання-архітектури та використання великих мовних моделей для природномовної взаємодії з користувачем. Визначені вимоги створюють основу для подальшого проєктування архітектури AI-асистента та реалізації механізмів автоматизації сервісних процесів.

3.5 Висновок

У межах третього розділу проведено аналіз архітектури Jira, механізмів роботи із задачами, процесів виконання та REST API системи. Встановлено, що

Jira реалізує процес виконання-орієнтовану модель управління задачами, у якій зміна станів виконується через механізм переходів між станами , що забезпечує контроль допустимих переходів та підтримку бізнес-процесів.

Дослідження REST API підтвердило можливість отримання, пошуку, оновлення та аналітичної обробки задач за допомогою HTTP-запитів, JQL та структурованих JSON-відповідей. Визначено, що ієрархічна структура даних Jira дозволяє реалізувати фільтрацію, агрегацію та подальшу аналітичну обробку інформації.

Проведений аналіз показав, що стандартні засоби аналітики Jira забезпечують формування звітності та статистичних вибірок, проте мають обмежені можливості щодо автоматизованої інтерпретації результатів. Водночас дослідження сучасних AI-асистентів засвідчило ефективність використання великих мовних моделей для роботи з текстовими даними, створення задач, пошуку інформації та підготовки узагальнень, але також виявило недостатню інтеграцію більшості рішень із процесами виконання та сервісними процесами.

На основі проведеного аналізу сформовано вимоги до AI-системи підтримки управління проєктами, що передбачає використання великої мовної моделі, механізму виклику інструментів, інтеграції через REST API, контролю переходів між статусами та перевірки інформаційної безпеки. Отримані результати створюють основу для проєктування архітектури та реалізації AI-асистента, інтегрованого з Jira.

РОЗДІЛ 4
РОЗРОБКА AI-СИСТЕМИ ПІДТРИМКИ УПРАВЛІННЯ ПРОЄКТАМИ

4.1 Постановка задачі та аналіз вимог

4.1.1 Постановка задачі та аналіз вимог

У зв’язку з цим постає задача розробки AI-системи підтримки управління проєктами, інтегрованої з Jira та здатної виконувати операції над задачами через REST API з використанням природномовних запитів користувача. Система повинна забезпечувати автоматизоване створення задач, отримання та фільтрацію списків заявок, зміну статусів через процес виконання переходу між , аналітичну обробку інформації та формування звітності без необхідності прямої взаємодії користувача з графічним інтерфейсом Jira.

Ключовою особливістю системи має стати використання великої мовної моделі як інтелектуального рівня інтерпретації користувацьких запитів. Модель повинна визначати намір користувача, обирати відповідний інструмент взаємодії з REST API Jira та формувати параметри виклику для подальшого виконання операції серверний застосуноком.

4.1.2 Функціональні вимоги

Розроблювана AI-система повинна забезпечувати автоматизовану взаємодію користувача із Jira через запити природною мовою. Основою її функціонування є використання великої мовної моделі для аналізу запиту, визначення наміру користувача та формування параметрів взаємодії з REST API Jira.

Однією з основних функцій системи є створення задач на основі природномовного опису. Під час створення заявки система автоматично формує необхідні параметри та враховує вимоги до заповнення полів відповідно до конфігурації сервіс-деску.

Система також повинна підтримувати оновлення статусів задач через механізм переходу між статусами Для цього виконується отримання доступних переходів та вибір коректного переходу відповідно до поточного стану задачі й правил процесу виконання.

Окремим функціональним напрямом є отримання та фільтрація даних. Система повинна забезпечувати пошук задач, фільтрацію за статусами та іншими параметрами, а також відображення основної інформації про заявки у структурованому вигляді.

Для підтримки прийняття рішень передбачено базові аналітичні можливості, зокрема підрахунок кількості задач, аналіз їх розподілу за статусами та визначення найбільш поширених категорій звернень.

Взаємодія між великою мовною моделлю та функціональними модулями реалізується за допомогою механізму виклику інструментів, який дозволяє моделі обирати необхідну операцію та виконувати її через REST API Jira. Завдяки цьому мовна модель виступає не лише засобом генерації тексту, а й компонентом координації прикладних операцій.

Додатково система повинна забезпечувати перевірку текстових даних на наявність конфіденційної інформації, зокрема паролів, токенів доступу та API-ключів, перед передачею даних до Jira.

Таким чином, функціональні можливості системи спрямовані на автоматизацію роботи із задачами та сервісними зверненнями шляхом поєднання великих мовних моделей, механізму виклику інструментів і REST API Jira.

4.1.3 Нефункціональні вимоги

Нефункціональні вимоги до розроблюваної AI-системи підтримки управління проєктами охоплюють продуктивність, безпеку, надійність та масштабованість.

Система повинна забезпечувати швидку обробку користувацьких запитів і стабільну взаємодію з REST API Jira, а також підтримувати одночасне виконання декількох операцій без суттєвого зниження продуктивності.

Важливою вимогою є забезпечення безпеки даних. Передача інформації повинна здійснюватися через захищені канали зв’язку із використанням механізмів автентифікації та авторизації. Конфігураційні параметри та ключі доступу мають зберігатися поза програмним кодом. Додатково система повинна виконувати перевірку текстових даних на наявність конфіденційної інформації перед їх передачею до Jira.

Надійність системи забезпечується коректною обробкою помилок REST API, контролем результатів роботи великої мовної моделі та збереженням працездатності у випадку недоступності зовнішніх сервісів.

Масштабованість досягається завдяки модульній архітектурі, що дозволяє розширювати функціональність шляхом додавання нових інструментів і сервісів без зміни базової структури застосунку. Використання Spring Boot забезпечує гнучкість розгортання та подальшого розвитку системи.

4.2 Архітектура та технологічний набір

4.2.1 Обгрунтування вибору технологій

Вибір технологічного набору для реалізації AI-системи підтримки управління проєктами здійснювався з урахуванням вимог до інтеграції великих мовних моделей, підтримки REST-орієнтованої взаємодії та забезпечення модульної архітектури застосунку.

Як базову платформу розробки обрано Java, яка є одним із найбільш поширених інструментів створення корпоративних інформаційних систем завдяки високій надійності, платформонезалежності та розвиненій екосистемі програмних компонентівх [12]. Реалізація серверної частини виконана із використанням Spring Boot, що забезпечує побудову REST-сервісів, керування залежностями та модульну організацію бізнес-логіки [6].

Інтеграція великої мовної моделі реалізована за допомогою Spring AI, який забезпечує взаємодію між AI-рівнем та функціональними модулями системи. Використання даного програмного каркасу дозволяє реалізувати аналіз природномовних запитів, визначення наміру користувача та механізм виклику інструментів для виконання прикладних операцій [5].

Взаємодія із Jira здійснюється через REST API, що забезпечує стандартизований доступ до задач, процесів виконання, переходів між статусами та аналітичних даних [3]. Для реалізації HTTP-взаємодії використовується `HttpClient`, який забезпечує формування та обробку REST-запитів між серверною частиною системи та Jira.

Тестування інтеграційної взаємодії виконувалося із застосуванням Postman, що дозволило перевірити коректність роботи REST API, структуру JSON-відповідей та механізми зміни статусів.

Особливістю реалізованої архітектури є використання прямої інтеграції між великою мовною моделлю та функціональними модулями через механізм виклику інструментів без застосування протоколу контексту моделі (MCP). Такий підхід забезпечує зменшення архітектурної складності, спрощує керування бізнес-логікою та надає повний контроль над виконанням операцій у Jira.

Таким чином, обраний технологічний набір забезпечує реалізацію масштабованої AI-системи, здатної виконувати автоматизовану обробку природномовних запитів, взаємодіяти з Jira через REST API та підтримувати процеси управління проєктами в єдиному інформаційному середовищі.

4.2.2 Загальна архітектура AI-системи

Архітектура розроблюваної AI-системи підтримки управління проєктами побудована відповідно до принципів модульної REST-орієнтованої архітектури та передбачає розділення системи на окремі функціональні рівні: рівень взаємодії з користувачем, серверна частина-рівень обробки запитів, AI-рівень інтеграції великої мовної моделі та рівень взаємодії з Jira через REST API.

У межах розробленої системи користувач взаємодіє з AI-асистентом через природномовні текстові запити. Взаємодія реалізована через HTTP кінцева точка `/api/chat`, а також через консольний інтерфейс, інтегрований через протокол контексту моделі (MCP). Такий підхід дозволяє використовувати систему як у форматі REST API-сервісу, так і через інтерактивну взаємодію з великою мовною моделлю у консольному режимі.

Після отримання запиту серверна частина-застосунок, реалізований на основі Spring Boot, передає текст користувача до AI-рівня через Spring AI. Велика мовна модель виконує аналіз тексту, визначає намір користувача та формує рішення щодо подальшої операції над задачами Jira.

Серверний рівень виконує функції координації компонентів системи, маршрутизації запитів, обробки JSON-відповідей, контролю зміни статусу та обробки помилок. Окремим компонентом є модуль перевірки безпеки, який аналізує текст запиту на наявність чутливих даних, зокрема паролів, токенів доступу та API-ключів, перед передачею інформації до Jira.

Для взаємодії із Jira використовується REST API Jira. Через REST API система виконує створення сервісних заявок, отримання списків задач, роботу з процес виконання переходів між статусами, виконання пошуку через JQL, отримання необхідних полів та формування аналітичних вибірок даних. Обмін інформацією між серверний застосуноком та Jira виконується через HTTP REST-запити з використанням JSON-формату.

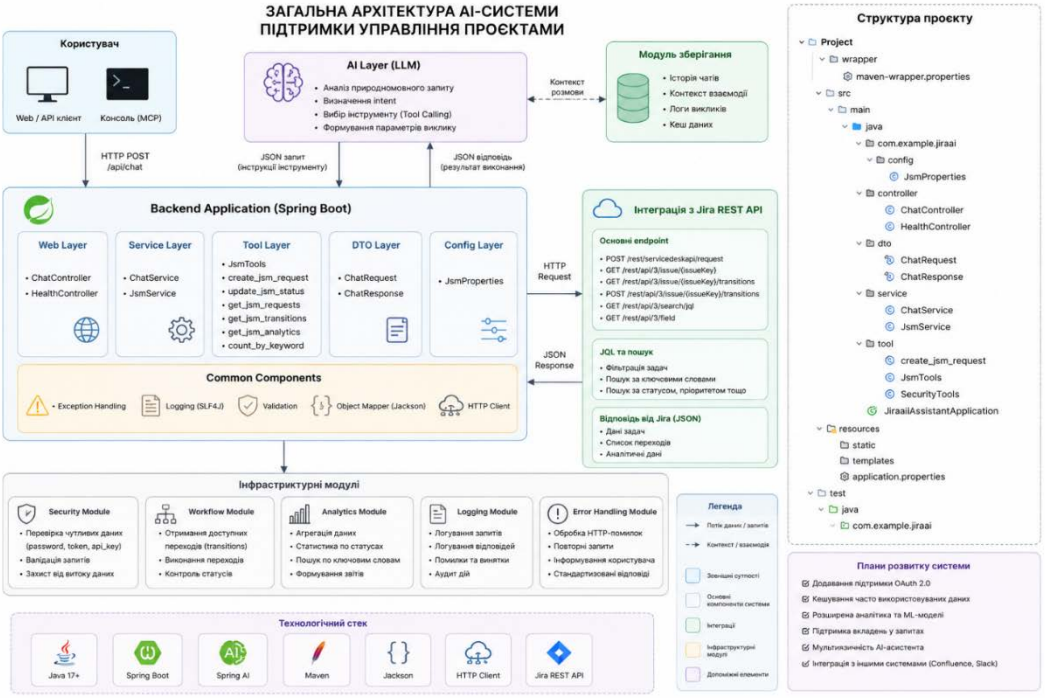


Рисунок 4.1. - Загальна архітектура AI-системи підтримки управління проєктами

У загальному вигляді взаємодія компонентів системи реалізується як послідовна передача природномовного запиту від користувача до серверний застосуноку, подальша обробка запиту великою мовною моделлю, визначення необхідного інструментк та виконання REST API-взаємодії з Jira. У результаті

система отримує JSON-відповідь від REST API, виконує обробку результатів та формує природномовну відповідь користувачу.

4.2.3 Архітектура взаємодії з Jira REST API

Архітектура взаємодії розробленої AI-системи з Jira REST API реалізована через окремий інтеграційний рівень серверний застосуноку. Основна задача цього рівня полягає у забезпеченні уніфікованого механізму обміну даними між AI-рівнем системи та зовнішньою платформою Jira.

У межах реалізації за взаємодію з Jira відповідає клас `JsmService`, який інкапсулює логіку формування HTTP-запитів, створення заголовків авторизації, передачі JSON-структур, отримання відповідей REST API та їх первинної обробки. Такий підхід дозволяє відокремити інтеграційну логіку від контролерів, AI-рівня та інструментів, що викликаються великою мовною моделлю.

Контролер реалізований максимально тонким шаром — він лише приймає запит та передає його до ChatService:

```
@PostMapping
public ChatResponse chat(@RequestBody ChatRequest request) {
    String response = chatService.ask(request.message());
    return new ChatResponse(response);
}
```

Лістинг 4.1 - ChatController: точка входу REST API

AI-рівень реалізовано у класі `ChatService`. У цьому класі формується запит до великої мовної моделі, передається повідомлення користувача та підключаються доступні інструменти.

Клас `JsmTools` виконує роль проміжного шару між великою мовною моделлю та сервісом інтеграції з Jira. Методи цього класу позначені анотацією `@Tool`, що дозволяє Spring AI передавати їх моделі як доступні інструменти. У

межах архітектури цей шар не містить HTTP-логіки, а лише маршрутизує виклик до відповідного методу `JsmService`.

Приклад оголошення інструменту для взаємодії з Jira:

```
@Tool(description = "Get Jira requests filtered by status,
issue key, summary keyword, or other text")
public String get_jsm_requests_by_filter(String filter) {
    return jsmService.getRequestsByFilter(filter);
}
```

Лістинг 4.2 - JsmTools: делегування виклику до сервісного шару

Безпосередня REST API-взаємодія з Jira реалізована у класі `JsmService`. У цьому класі формується базова структура запиту: URL до Jira, заголовок авторизації, HTTP-заголовки та об'єкт запиту. Для виконання HTTP-взаємодії використовується `RestTemplate`, а для обробки JSON-відповідей – `ObjectMapper`.

Приклад формування заголовків авторизації для Jira REST API:

```
String auth = props.getEmail() + ":" + props.getApiToken();
String encodedAuth =
Base64.getEncoder().encodeToString(auth.getBytes());
HttpHeaders headers = new HttpHeaders();
headers.set("Authorization", "Basic " + encodedAuth);
headers.setAccept(MediaType.parseMediaTypes("application/json"
));
```

Лістинг 4.3 - Формування заголовка Basic Authentication

Після формування заголовків серверна частина-застосунок виконує HTTP-запит до Jira REST API. Відповідь зовнішньої системи повертається у вигляді JSON-рядка та надалі перетворюється у структурований об'єкт для отримання необхідних полів.

4.2.4 Використання протоколу контексту моделі (MCP)

Протокол контексту моделі (MCP) у розробленій системі реалізує стандартизований механізм передачі контексту між моделлю Qwen 2.5,

програмний каркасом Spring AI та серверний компонентами. Основне завдання MCP полягає не лише в організації консольної взаємодії, а у формуванні єдиного контекстного рівня, через який модель отримує описи доступних інструментів, результати їх виконання та історію взаємодії з користувачем. Завдяки цьому модель функціонує не як ізольований текстовий генератор, а як AI-агент оркестрації, здатний ініціювати серверну частина-операції у Jira через механізм виклику інструментів (виклик інструментів).

```
@Service
public class ChatService {
    private final ChatClient chatClient;
    private final SecurityTools securityTools;
    private final JsmTools jsmTools;
    private final ChatMemory chatMemory = new
InMemoryChatMemory();
    public ChatService(
            ChatClient.Builder builder,
            SecurityTools securityTools,
            JsmTools jsmTools
    ) {
        this.securityTools = securityTools;
        this.jsmTools = jsmTools;

        this.chatClient = builder
                .defaultAdvisors(
 MessageChatMemoryAdvisor.builder(chatMemory).build()
                )
                .defaultTools(securityTools, jsmTools)
                .build();
    }
```

Лістинг 4.4 - Ініціалізація ChatService: реєстрація інструментів та контекстної пам’яті

Spring AI автоматично генерує JSON Schema для кожного `@Tool`-методу на основі Java-рефлексії та передає її моделі як маніфест інструментів [5] — саме це є ключовим MCP-елементом, що визначає, які операції доступні моделі та які параметри вони приймають.

Архітектурне відокремлення опису інструменту від виконавчої логіки забезпечує незалежність LLM-оркестратора від сервісного шару, дозволяє розширювати набір інструментів без модифікації оркестраційного шару та реалізує перевірку безпеки як окремий MCP-компонент у загальному pipeline виклику інструментів.

4.3 Реалізація функціональних модулів

4.3.1 Реалізація створення задач

Після отримання повідомлення користувача мовна модель аналізує його зміст, визначає намір створення заявки та формує необхідні параметри для подальшого виклику інструменту `create_jsm_request`. Перед створенням заявки виконується перевірка введених даних за допомогою інструменту безпеки, який аналізує текст на наявність потенційно чутливої інформації. Лише після успішного проходження перевірки система переходить до створення заявки в Jira (рис.4.1).

Основна логіка створення заявки реалізована в методі `createRequest()`, фрагмент якого наведено нижче.

```
public String createRequest(String summary, String description) {
    String url = props.getBaseUrl() +
"/rest/servicedeskapi/request";

    String auth = props.getEmail() + ":" + props.getApiToken();
    String encodedAuth =
Base64.getEncoder().encodeToString(auth.getBytes());

    HttpHeaders headers = new HttpHeaders();
    headers.set("Authorization", "Basic " + encodedAuth);
    headers.setContentType(MediaType.APPLICATION_JSON);

headers.setAccept(MediaType.parseMediaTypes("application/json"));

    Map<String, Object> body = Map.of(
            "serviceDeskId", props.getServiceDeskId(),
            "requestTypeId", props.getRequestTypeId(),
```

```
            "requestFieldValues", Map.of(
                    "summary", summary,
                    "description", description
            )
    );

    HttpEntity<Map<String, Object>> entity = new HttpEntity<>(body,
headers);

    ResponseEntity<String> response =
            restTemplate.exchange(url, HttpMethod.POST, entity,
String.class);

    JsonNode root = objectMapper.readTree(response.getBody());

    String issueKey = root.path("issueKey").asText();
    String status =
root.path("currentStatus").path("status").asText();
    String webLink = root.path("_links").path("web").asText();

    return """
    Заявку створено успішно.
    Номер: %s
    Статус: %s
    Посилання: %s
    """.formatted(issueKey, status, webLink);
}
```

Лістинг 4.5 - Реалізація методу створення заявки в Jira

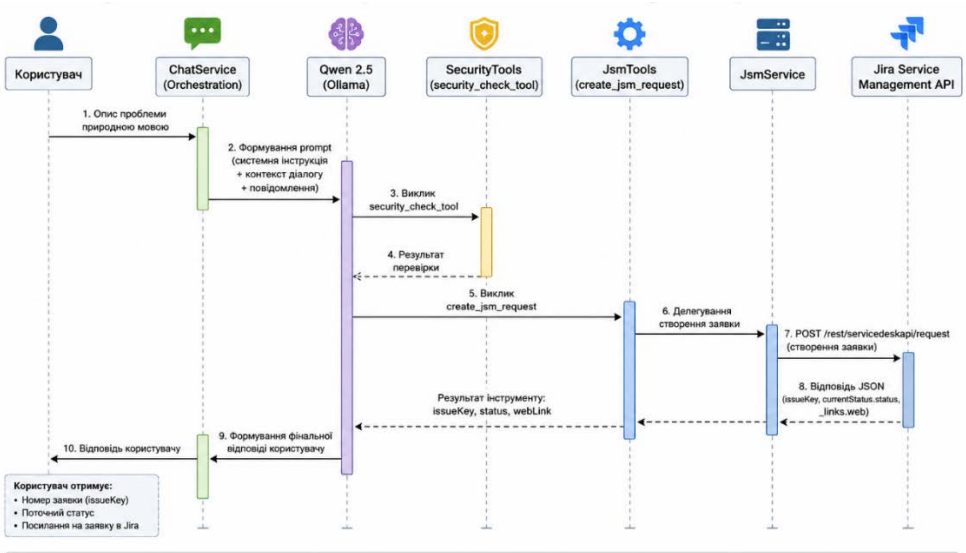


Рисунок 4.2. - Послідовність створення заявки в Jira через AI-асистента

Технічно взаємодія з Jira реалізована через REST API платформи. Для створення заявки використовується HTTP POST-запит до кінцевої точки `/rest/servicedeskapi/request` [4]. Авторизація здійснюється за допомогою механізму `Basic Authentication`, де для формування заголовка авторизації використовується пара значень: електронна адреса – токен API. Параметри підключення зберігаються у конфігураційних властивостях застосунку та автоматично завантажуються під час запуску системи.

4.3.2 Реалізація оновлення статусів задач

Оновлення статусу заявки в Jira реалізується через механізм доступних переходів робочого процесу, оскільки зміна стану задачі визначається правилами процесу виконання і не може здійснюватися шляхом прямого редагування поля статусу [3].

На першому етапі система отримує перелік доступних переходів для заявки через Jira REST API. Після цього виконується пошук переходу, який відповідає статусу, визначеному великою мовною моделлю.

```
for (JsonNode transition : transitions) {
    String id = transition.path("id").asText();
    String name = transition.path("name").asText();
    String toStatus =
transition.path("to").path("name").asText();

    if (
            name.equalsIgnoreCase(targetStatus) ||
                    toStatus.equalsIgnoreCase(targetStatus)
    ) {
        selectedTransitionId = id;
        selectedTransitionName = name + " → " + toStatus;
        break;
    }
```

Лістинг 4.6 - Пошук `transition` за назвою або цільовим статусом

Після визначення ідентифікатора переходу система формує запит до Jira та виконує зміну статусу.

Після цього модель викликає інструмент `update_jsm_request`, який передає параметри до сервісного шару системи. Якщо для поточного стану заявки потрібний перехід доступний, система виконує зміну статусу через Jira REST API та повертає повідомлення про успішне оновлення. Якщо відповідний перехід відсутній, користувач отримує повідомлення про неможливість виконання операції в межах поточного процес виконання-стану.

4.3.3 Реалізація отримання та фільтрації задач

Отримання заявок із Jira реалізовано через звернення до Jira REST API. Для цього використовується GET-запит до кінцевої точки `/rest/servicedeskapi/request` із передаванням параметра `serviceDeskId`, який визначає сервісний проєкт, з якого необхідно отримати заявки [4]. Такий підхід дозволяє обмежити вибірку лише тими зверненнями, що належать до конкретної проблеми. Фільтрація заявок у системі реалізована двома способами. Перший спосіб передбачає відбір заявок за статусом і реалізований у методі `getRequestsByStatus()`.

```
(JsonNode item : values) {
    String issueKey = item.path("issueKey").asText();
    String summary = item.path("summary").asText();
    String status =
item.path("currentStatus").path("status").asText();
    String link = item.path("_links").path("web").asText();

    if (status.equalsIgnoreCase(statusFilter)) {
        result.append("- ")
                .append(issueKey)
                .append(" | ")
                .append(status)
                .append(" | ")
                .append(summary)
                .append("\n")
                .append("  ")
                .append(link)
```

```
                .append("\n");
        }
    }
```

Лістинг 4.7 - Клієнтська фільтрація заявок за статусом

У цьому випадку система спочатку отримує повний список заявок із Jira, після чого виконує клієнтську фільтрацію за значенням поля `currentStatus.status` (рис.4.3) Порівняння виконується без урахування регістру, що дозволяє коректно обробляти запити користувача незалежно від способу написання статусу.

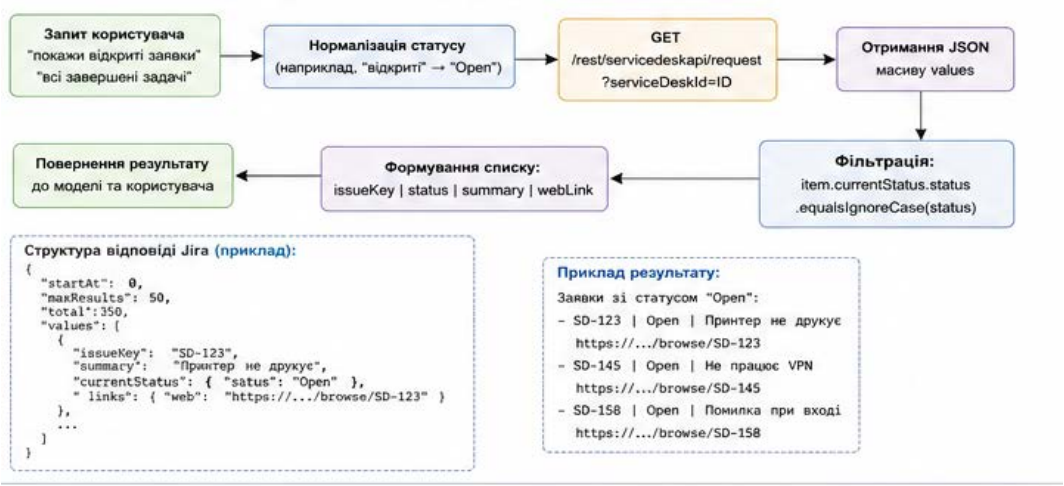


Рисунок 4.3. - Фільтрація задач за статусом

Другий спосіб фільтрації реалізований у методі `getRequestsByFilter()` і призначений для пошуку заявок за ключовим словом або частиною опису. На відміну від фільтрації за статусом, цей підхід не прив'язаний до одного конкретного поля. Система формує рядкове представлення заявки та виконує пошук за допомогою `String.contains()` у нижньому регістрі. Завдяки цьому користувач може шукати заявки одночасно за ключем, назвою, статусом або фрагментом тексту з короткого опису (рис4.4).

```
public String getRequestsByFilter(String filter) {
    String allRequests = getRequests();
    String lowerFilter = filter.toLowerCase();
    StringBuilder result = new StringBuilder();

    for (String line : allRequests.split("\\n")) {
```

```
            if (line.toLowerCase().contains(lowerFilter)) {
                result.append(line).append("\n");
            }
        }
        return result.isEmpty()
                ? "Заявок за фільтром \"" + filter + "\" не
знайдено."
                : result.toString();
    }
```

Лістинг 4.8 - getRequestsByFilter(): пошук по рядковому представленню

Рисунок 4.4. - Фільтрація задач за статусом

4.3.4 Реалізація аналітичного модуля

Для забезпечення можливості отримання узагальненої інформації про стан сервісного обслуговування в системі реалізовано аналітичний модуль, який виконує автоматичну агрегацію показників на основі даних Jira. Основна логіка модуля реалізована в методі `JsmService.getAnalytics()`.

Під час виконання методу система отримує повний перелік заявок сервісного проєкту через Jira REST API. Для цього використовується GET-запит до кінцевої точки `/rest/servicedeskapi/request [4]`. Отримана JSON-відповідь містить набір заявок, які надалі використовуються для обчислення статистичних показників.

На першому етапі виконується підрахунок загальної кількості заявок та їх групування за статусами. Для кожної заявки аналізується поле currentStatus.status, на основі якого формується розподіл між категоріями Open, In Progress та Completed.

```
String status =
item.path("currentStatus").path("status").asText().toLowerCase
();
String summary = item.path("summary").asText().toLowerCase();

if (status.contains("open")) open++;
else if (status.contains("progress")) inProgress++;
else if (status.contains("complete") ||
status.contains("done")) completed++;
```

Лістинг 4.9 - Підрахунок загальної кількості заявок

Другим етапом є аналіз змісту заявок на основі поля `summary`. Текст приводиться до нижнього регістру, розбивається на окремі слова та використовується для визначення найбільш поширених категорій проблем.

```
for (String word : summary.split("[^a-zA-Za-яA-ЯiIїЇєЄ]+")) {
    if (word.length() < 4) continue;

    word = word.toLowerCase();

    wordFrequency.put(word, wordFrequency.getOrDefault(word,
0) + 1);
}
```

Лістинг 4.10 - Аналіз змісту заявок

Після завершення аналізу виконується сортування отриманих результатів і відбір найбільш поширених ключових слів.

```
List<Map.Entry<String, Integer>> topWords =
wordFrequency.entrySet()
        .stream()
        .sorted((a, b) -> b.getValue() - a.getValue())
        .limit(3)
        .toList();
```

Лістинг 4.9 - Відбір найбільш поширених ключових слів

Додатково обчислюється відсоток завершених заявок відносно загальної кількості звернень, що дозволяє оцінити ефективність обробки запитів користувачів.

Результатом виконання методу є структурований аналітичний звіт, що містить:

- загальну кількість зареєстрованих заявок;
- кількість заявок у кожному статусі;
- відсоток завершених звернень;
- перелік найпоширеніших проблемних категорій.

Сформований звіт повертається до мовної моделі через інструмент `get_jsm_analytics`, після чого відображається користувачеві у вигляді природномовної відповіді. Завдяки цьому користувач може отримати узагальнену інформацію про роботу сервісного підрозділу за допомогою одного запиту без необхідності використання стандартних засобів звітності Jira.

4.4 Реалізація AI-оркестрації запитів

4.4.1 Реалізація механізму виклику інструментів

У розробленій системі механізм виклику інструментів реалізовано засобами Spring AI та великої мовної моделі Qwen 2.5. На відміну від традиційного підходу, де вибір функції та формування параметрів виконуються окремими алгоритмами, у запропонованій архітектурі ці задачі покладаються безпосередньо на мовну модель [5, 7].

Після отримання повідомлення користувача модель аналізує його зміст разом з описами доступних інструментів, сформованими на основі анотацій `@Tool` та `@ToolParam`. На основі цього аналізу визначається необхідна операція та автоматично формуються параметри її виклику [5].

Важливою складовою механізму є нормалізація параметрів відповідно до правил, визначених у системній інструкції. Це дозволяє використовувати природні формулювання користувача без необхідності точного введення назв статусів, ідентифікаторів або інших службових значень.

Додатково під час формування параметрів використовується контекстна пам’ять, що дає змогу відновлювати необхідні дані з попередніх етапів діалогу. Завдяки цьому підтримуються багатокрокові сценарії взаємодії, у яких користувач може уточнювати або продовжувати виконання операції без повторного введення всіх параметрів.

Приклад визначення інструменту наведено нижче.

```
@Tool(description = "Count how many Jira requests contain a
specific keyword in their summary. Use when user asks 'how
many', 'count', 'скільки'.")
public String count_jsm_requests_by_keyword(
        @ToolParam(description = "Keyword to search for in
request summaries")
        String keyword
) {
```

Лістинг 4.12 - Приклад мінімального інструмента

Запропонований підхід забезпечує слабке зв’язування між мовною моделлю та бізнес-логікою застосунку. Для додавання нової функціональності достатньо створити новий метод із анотацією `@Tool`, після чого він автоматично стає доступним для моделі без внесення змін до механізму оркестрації..

4.4.2 Реалізація системної інструкції AI-агента

Системна інструкція є центральним компонентом керування поведінкою AI-агента та визначає правила роботи великої мовної моделі під час взаємодії з Jira. Вона передається до моделі під час кожного виклику та забезпечує стабільність її поведінки незалежно від змісту поточного запиту користувача.

Основними функціями системної інструкції є визначення ролі AI-агента, маршрутизація запитів до відповідних інструментів, використання контекстної пам’яті, нормалізація параметрів та формування відповідей користувачеві. Зокрема, інструкція задає правила визначення наміру користувача, відповідність між категоріями запитів та інструментами системи, а також механізми роботи зі статусами Jira.

Крім цього, системна інструкція містить правила використання контекстної пам’яті, які дозволяють продовжувати багатокрокові сценарії без повторного введення всіх параметрів, а також визначає формат фінальних відповідей, забороняючи виведення службової інформації та внутрішніх структур даних.

Ключовим елементом є блок `TOOL SELECTION RULES` – матриця відповідності між описом запиту та інструментом. Замість складної логіки класифікації правила задані декларативно у природній мові, і сама модель виконує розпізнавання наміру:

```
TOOL SELECTION — apply the FIRST matching rule:
1. Message contains "покажи", "покажіть", "показати", "список",
"заявки", "тікети", "show", "list", "get", "display"
   AND does NOT describe a problem → call get_jsm_requests
2. Message contains a status word (open, відкриті, completed,
завершені, pending, in progress, canceled) + "заявки"
   → call get_jsm_requests_by_status with that status
3. Message contains "скільки", "порахуй", "how many", "count"
   → call count_jsm_requests_by_keyword
4. Message contains "аналітика", "статистика", "dashboard",
"analytics", "метрики"
   → call get_jsm_analytics
5. Message contains an issue key (e.g. SDAI-21, IT-5) AND a change
word (закрий, оновити, змінити, update, close, resolve, move)
   → call update_jsm_request_status
6. Message contains an issue key (e.g. SDAI-21) only
   → call get_jsm_request_transitions
7. Message contains a search term or keyword but no action verb
   → call get_jsm_requests_by_filter
8. Message describes a technical problem: broken hardware, software
crash, no access, BSOD, синій екран, не працює, зламався, впав,
```

```
вилітає, зависає, мережа, VPN, пошта
   → call security_check_tool THEN call create_jsm_request
immediately after
```

Лістинг 4.13 - Фрагмент системної інструкції: правила маршрутизації та маппінг статусів

Окремим правилом визначено поведінку при підтвердженні: якщо попереднє повідомлення містило пропозицію дії, а поточне – 'так', 'ok', 'виконуй', модель продовжує розпочату операцію без повторного з'ясування параметрів. Це реалізує природній двокроковий сценарій взаємодії.

4.4.3 Обробка помилок та механізм відновлення виконання

У розробленому рішенні механізми обробки помилок реалізовані на двох рівнях: рівні сервісного шару взаємодії з Jira та рівні оркестраційної логіки мовної моделі.

На рівні сервісного шару всі операції взаємодії з Jira REST API виконуються всередині конструкцій `try-catch`. Це дозволяє перехоплювати помилки, пов’язані з мережевими збоями, недоступністю сервера, помилками автентифікації або некоректними відповідями API. У випадку виникнення виняткової ситуації система фіксує інформацію про помилку в журналі виконання та формує текстове повідомлення, яке повертається як результат роботи відповідного інструменту.

Фрагмент реалізації механізму обробки помилок наведено нижче.

```
try {
    String response = chatService.ask(input);

    System.out.println("\nAssistant:");
    System.out.println(response);

} catch (Exception e) {
    System.out.println("Error while processing request:");
    System.out.println(e.getMessage());
}
```

Лістинг 4.14 - Обробка помилок: повернення тексту замість виключення

Другий рівень обробки помилок реалізований на рівні оркестраційної логіки AI-агента. Системна інструкція містить правила поведінки моделі у випадках, коли виконання інструменту завершилося помилкою або коли жоден із доступних інструментів не відповідає наміру користувача. У таких ситуаціях модель переходить до режиму генерації текстової відповіді та намагається надати користувачеві корисну інформацію без виконання операцій у зовнішній системі.

Такий підхід реалізує механізм керованого відновлення виконання (guided error recovery), за якого система допомагає користувачеві знайти коректний шлях продовження роботи замість простого повідомлення про помилку.

4.5 Реалізація модуля безпеки

Основною функцією інструменту є виявлення потенційно чутливих даних у тексті повідомлення. Для цього вхідний текст приводиться до нижнього регістру, після чого виконується послідовна перевірка на наявність характерних маркерів, що можуть свідчити про передачу конфіденційної інформації. Поточна реалізація підтримує перевірку чотирьох категорій даних:

- паролі (`password, passwd, pwd`);
- API-ключі (`api_key, apikey, api-key`);
- ключі автентифікації (`token, access_token`);
- секретні ключі (`secret, private_key`).

Фрагмент реалізації інструменту наведено нижче:

```
@ToolParam(description = "The full user message text to scan for
sensitive data")
        String text
) {
```

```
    System.out.println("=== SECURITY CHECK === input=" + text);

    if (text == null || text.isBlank()) {
        return "No text provided";
    }

    List<String> issues = new ArrayList<>();
    String lower = text.toLowerCase();

    if (lower.contains("password") || lower.contains("passwd") ||
lower.contains("pwd"))
        issues.add("password");
    if (lower.contains("api_key") || lower.contains("apikey") ||
lower.contains("api-key"))
        issues.add("API key");
    if (lower.contains("token") || lower.contains("access_token"))
        issues.add("token");
    if (lower.contains("secret") || lower.contains("private_key"))
        issues.add("secret/private key");

    if (issues.isEmpty()) return "CLEAR";

    return "WARNING: sensitive data detected — " + String.join(",
", issues) + ". Sanitize before creating the request.";
}
```

Лістинг 4.15 - Детекція чутливих даних

Особливістю реалізації є інтеграція механізму перевірки безпеки не на рівні програмного коду сервісного шару, а на рівні системної інструкції мовної моделі. У системній інструкції визначено правило, згідно з яким перед виконанням будь-якої операції створення заявки обов’язково повинен бути викликаний інструмент `security_check_tool.` Таким чином перевірка стає невід’ємною частиною процесу створення заявки незалежно від формулювання запиту користувача.

Загальний алгоритм роботи модуля безпеки полягає в наступному. Після отримання повідомлення користувача модель визначає намір створення заявки та передає повний текст повідомлення до інструменту перевірки безпеки. На основі результату перевірки приймається рішення щодо подальших дій. Якщо потенційно небезпечних даних не виявлено, система продовжує виконання

операції створення заявки. Якщо ж виявлено ознаки передачі конфіденційної інформації, користувач отримує попередження, а операція створення заявки не виконується.

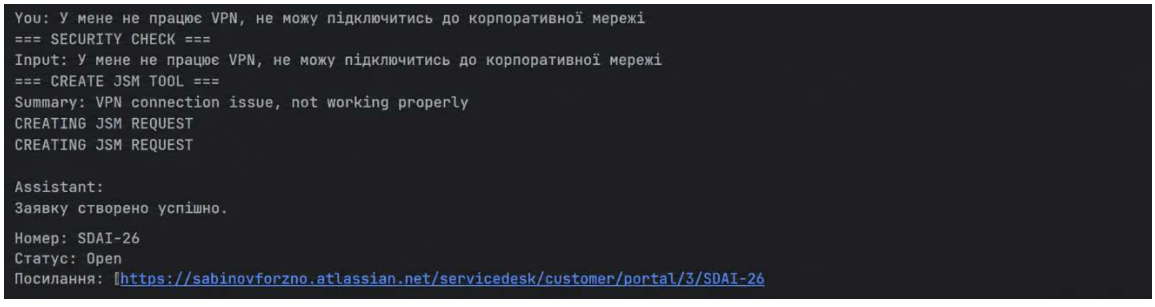
Додатково системна інструкція забороняє включати виявлені чутливі дані до полів `summary` або `description` заявки навіть у зміненому чи скороченому вигляді. Це забезпечує захист від випадкового потрапляння конфіденційної інформації до Jira.

Практичним результатом роботи модуля є зменшення ризику витоку облікових даних через природномовний інтерфейс системи. Разом із тим поточна реалізація базується на пошуку ключових слів і має певні обмеження. Вона ефективно виявляє лише ті випадки, коли чутливі дані супроводжуються характерними маркерами. Наприклад, система може не виявити токен доступу або пароль, якщо вони передані без відповідних ключових слів.

4.6 Тестування та валідація системи

4.6.1 Функціональне тестування

Функціональне тестування виконувалося з метою перевірки коректності реалізації основних можливостей AI-асистента та його взаємодії з Jira. Під час тестування перевірялися сценарії створення заявок, отримання списку заявок, фільтрації за статусом, зміни статусів, отримання аналітичної інформації та роботи механізму безпеки.



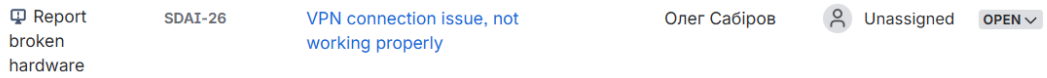


Рисунок 4.5. - Тестування створення заявки

На рисунку 4.5 наведено результат тестування операції створення заявки на основі природномовного опису проблеми. Система коректно визначила намір користувача, сформувала параметри заявки та створила запис у Jira через REST API.

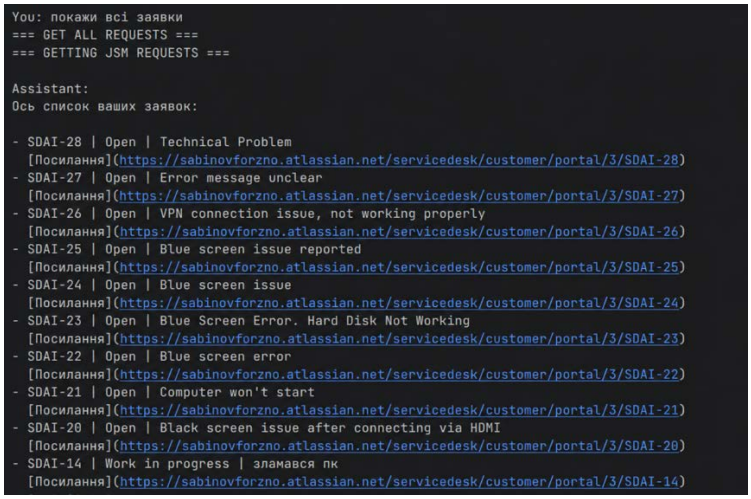


Рисунок 4.6 - Отримання повного списку заявок

На рисунку 4.6 показано результат отримання повного списку заявок. Після виконання запиту система повернула перелік зареєстрованих звернень із зазначенням номера заявки, поточного статусу та посилання для перегляду.

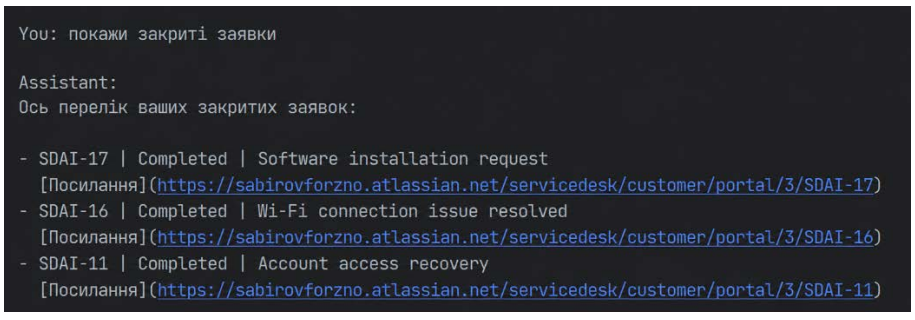


Рисунок 4.7 - Операція отримання списку заявок зі статусом «Completed»

На рисунку 4.7 наведено результат фільтрації заявок за статусом «Completed». Система успішно виконала відбір лише відкритих заявок та відобразила результати у структурованому вигляді.

4.6.2 Тестування механізму виклику інструментів та контекстної пам’яті

Тестування було спрямоване на перевірку коректності роботи механізму виклику інструментів та використання контекстної пам’яті під час багатокрокової взаємодії з користувачем.

Під час виконання сценарія система повинна була автоматично визначати необхідний інструмент на основі наміру користувача та формувати параметри його виклику. Для перевірки використовувалися запити на створення заявок, зміну статусів та отримання доступних переходів процесу виконання.

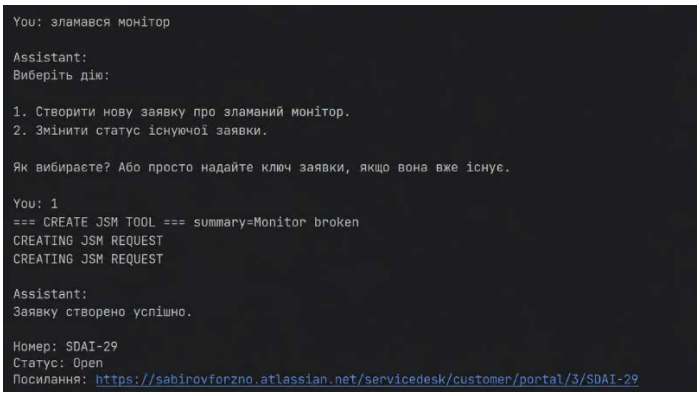


Рисунок 4.8 - Перевірка механізму контекстної пам’яті

4.6.3 Тестування модуля безпеки

Тестування модуля безпеки виконувалося з метою перевірки виявлення чутливих даних у повідомленнях користувача перед створенням заявок у Jira.

Для перевірки використовувалися тестові повідомлення, що містили паролі, токени доступу та інші потенційно конфіденційні дані. Під час

тестування система аналізувала текст повідомлення та виконувала перевірку за допомогою інструменту `security_check_tool`.

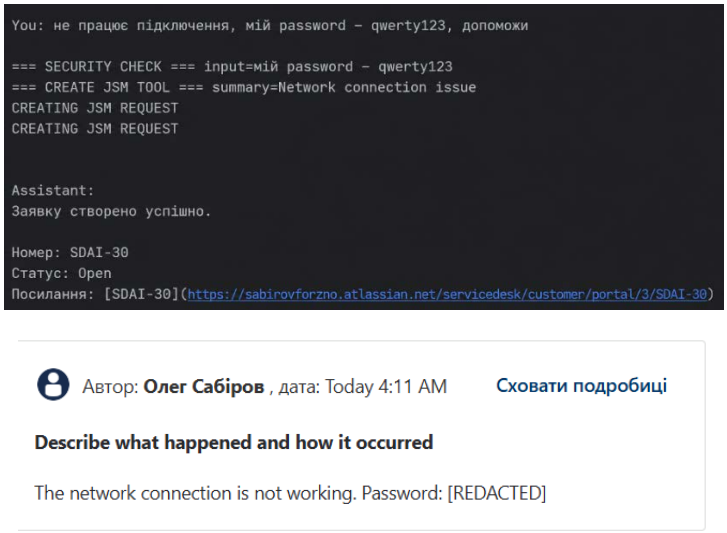


Рисунок 4.9. - Перевірка виявлення чутливих даних

4.6.4 Тестування аналітичного модуля

Тестування аналітичного модуля виконувалося з метою перевірки можливості отримання узагальненої інформації про заявки в Jira Для цього користувач надсилав природномовний запит щодо найчастіших проблем у сервісних зверненнях.

У результаті система виконала аналіз наявних заявок, визначила найбільш повторювані ключові слова у заголовках та сформувала відповідь у структурованому вигляді. На рис. 4.10 наведено приклад роботи аналітичного модуля, де система визначила найчастіші проблеми за заявками.

Отримані результати підтвердили, що аналітичний модуль здатний виконувати базову агрегацію даних, аналізувати частотність проблем та надавати користувачеві узагальнену інформацію без необхідності ручної обробки заявок у Jira.

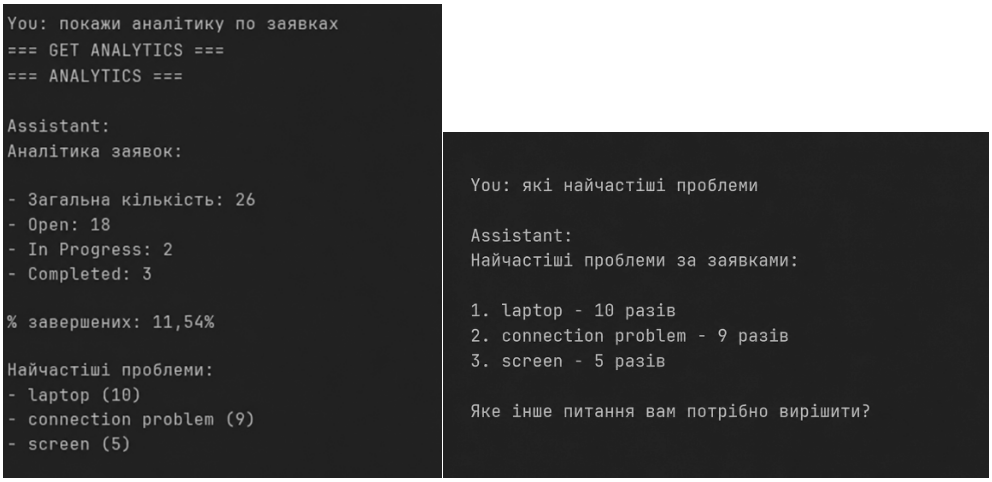


Рисунок 4.10. Перевірка аналітики по заявкам

4.6.5 Аналіз результатів тестування

Результати тестування підтвердили функціональну коректність усіх основних компонентів системи в межах визначених сценаріїв. Механізм виклик інструментів через Spring AI забезпечив стабільну передачу параметрів між моделлю та Java-методами у всіх перевірених сценаріях. маршрутизація запитів ШІ продемонстрував стійке визначення наміру для восьми категорій запитів при варіативності формулювань та мови введення.

Виявлені обмеження стосуються трьох аспектів. По-перше, детекція чутливих даних не охоплює токени та ключі без явних маркерних слів – токен Bearer без слова token у повідомленні не ідентифікується як чутливий. По-друге, витягування `issueKey` з контекстної пам'яті при неявних посиланнях на заявку є нестабільним при тривалих сеансах. По-третє, відсутність автоматизованих тестів унеможливлює регресійну верифікацію при модифікації компонентів — кожна зміна вимагає повторного ручного тестування базових сценаріїв.

4.7 Розширення функціональності

Поточний набір інструментів JsmTools охоплює базові CRUD-операції над заявками та обмежену аналітику. Першим напрямком розширення є підтримка коментарів до заявок через кінцева точка `/rest/api/3/issue/{issueKey}/comment` [3] . Додавання інструменту `add_jsm_comment` дозволить користувачам оновлювати заявки додатковою інформацією без переходу до веб-інтерфейсу Jira, що є типовим сценарієм при уточненні деталей інциденту в процесі його обробки.

Другим напрямком є підтримка призначення заявок на конкретного виконавця через поле `assignee` у `/rest/api/3/issue/{issueKey}` методом PUT [3] . Це дозволить реалізувати сценарій автоматичного маршрутизацію інцидентів до відповідального спеціаліста на основі типу проблеми, визначеного моделлю з тексту заявки. Логіка маршрутизації може бути інкапсульована безпосередньо у системній інструкції у вигляді маппінгу категорій проблем до ідентифікаторів виконавців.

Третім напрямком є розширення аналітичного модуля: додавання агрегації за часовими діапазонами (заявки за останній тиждень/місяць), розрахунок середнього часу вирішення інцидентів на основі порівняння `createdDate` та дати переходу в статус Completed, а також побудова розподілу заявок за типами обладнання на основі частотного аналізу summary. Ці метрики можуть бути реалізовані як окремі `@Tool`-методи без модифікації існуючої архітектури [5].

4.7.1 Оптимізація та масштабування

Поточна реалізація JsmService виконує окремий HTTP-запит для кожної операції без кешування результатів. При збільшенні кількості заявок у службі підтримки, це призводить до зростання латентності відповіді системи, оскільки

методи `getRequestsByStatus`, `getRequestsByFilter` та `getAnalytics` щоразу завантажують повний список заявок. Першим кроком оптимізації є введення короткочасного кешування результату `getRequests()` з TTL порядку 30–60 секунд через `Spring Cache` або простий `ConcurrentHashMap` із верифікацією. Це суттєво знизить кількість HTTP-запитів до Jira REST API при послідовних аналітичних запитах у рамках однієї сеансу.

Масштабування системи на рівні LLM передбачає перехід від локально розгорнутої моделі Qwen 2.5 через Ollama до хмарного постачальника LLM при зростанні навантаження [7, 8]. Spring AI забезпечує незалежний від постачальника інтерфейс – заміна моделі потребує лише зміни конфігурації `ChatClient.Builder` без модифікації рівня оркестрації або інструментів [5]. Альтернативою є горизонтальне масштабування через розгортання кількох екземплярів Ollama за балансувальником навантаження із спільним JsmService як компонентом без стану.

На рівні постійного зберігання даних поточна реалізація `InMemoryChatMemory` є сховищем, що унеможливлює збереження контексту між перезапусками застосунку. Перехід до PostgreSQL-backend сховища контекстної пам'яті через реалізацію інтерфейсу ChatMemory із JPA-репозиторієм дозволить зберігати контекст між сеансами та підтримувати персоналізовану взаємодію для кожного користувача системи [5].

4.7.2 Інтеграція RAG-механізмів

Retrieval-Augmented Generation (RAG) є перспективним напрямком розширення системи, що дозволяє доповнити контекст запиту до LLM релевантними документами з корпоративної бази знань без збільшення розміру

системної інструкції. У контексті системи управління інцидентами RAG-механізм може бути використаний для автоматичного пошуку рішень із попередніх заявок при створенні нового інциденту: система знаходить семантично схожі завершені заявки та включає їх опис і статус вирішення у контекст запиту до моделі.

Технічна реалізація RAG у поточній архітектурі передбачає введення трьох компонентів. Перший – векторне сховище (PostgreSQL із розширенням `pgvector` або окремий векторний індекс) для зберігання векторних представлень заявок Jira. Другий – модель векторизації для перетворення тексту заявок та запитів користувача у векторний простір; Spring AI підтримує інтеграцію моделей векторизації через `EmbeddingClient`. Третій – агент першої допомоги , що виконує пошук за схожістю у векторному сховищі та ін’єктує результати до системної інструкції перед відправленням до LLM. Spring AI надає `QuestionAnswerAdvisor` як готовий компонент для реалізації цього патерну [5].

Практичний ефект інтеграції RAG полягає у переході системи від реактивної обробки інцидентів до проактивної: при створенні нової заявки модель може автоматично запропонувати рішення на основі досвіду попередніх інцидентів аналогічного типу, скорочуючи час вирішення типових проблем без залучення технічного спеціаліста.

4.7.3 Розвиток multi-agent архітектури

Поточна архітектура системи реалізує одноагентну модель: один ChatService із одним набором інструментів обробляє всі типи запитів. При розширенні функціональності та збільшенні кількості інструментів цей підхід має обмеження – зростання розміру контекстного вікна через велику кількість

описів інструментів знижує точність визначення наміру та збільшує латентність відповіді моделі.

Мультиагентна архітектура передбачає декомпозицію системи на спеціалізованих агентів із чітко визначеною відповідальністю. Агент оркестрації отримує запит користувача, визначає його тип та делегує виконання до відповідного спеціалізованого агента: Агент інцидентів для створення та управління інцидентами, Аналітичний агент для агрегації метрик та звітності, Агент для управління переходами та статусами. Кожен агент має власний мінімальний набір інструментів та системну інструкцію, оптимізовану під конкретну предметну область.

У рамках Spring AI мультиагент взаємодія може бути реалізована через ланцюжок викликів ChatClient із передачею результату одного агента як контексту наступного, або через спільне сховище результатів інструментів на основі PostgreSQL, до якого мають доступ усі агенти. Такий підхід дозволяє масштабувати систему горизонтально – кожен агент може бути розгорнутий як окрема мікрослужба із власним LLM-кінцевою точкою [20].

4.7.4 Підвищення рівня безпеки

Поточна реалізація модуля безпеки базується на тексті введеному користувачем тому детекції чутливих даних має суттєве обмеження: токени, ключі та паролі без явних маркерних слів залишаються невиявленими. Першим напрямком підвищення рівня безпеки є розширення детекції через регулярні вирази для розпізнавання характерних форматів: токен Bearer (`Bearer [A-Za-z0-9\-._~+/]+=*`), JWT (`[A-Za-z0-9_-]{2,}(?:\.[A-Za-z0-9_-]{2,}){2}`), API-ключі у форматі UUID, а також хеші у форматах MD5, SHA-1 та SHA-256. Впровадження з використанням регулярних виразів детекції у `SecurityTools` не потребує змін в архітектурі системи – лише розширення методу `security_check_tool` [19].

Другим напрямком є введення рівнів критичності знахідок: замість бінарного результату `sensitive data detected / not found` метод повертає структурований об'єкт із переліком знахідок та їх категоріями. Модель на основі рівня критичності приймає рішення: при низькому рівні – попереджає користувача та продовжує операцію, при високому – блокує створення заявки та вимагає підтвердження з явним визнанням ризику.

Третім напрямком є аутентифікація користувачів системи. Поточна реалізація не розрізняє користувачів – всі запити виконуються від імені одного сервісного акаунту Jira. Введення JWT-аутентифікації на рівні `ChatController` та маппінгу користувачів системи до їх Jira-акаунтів дозволить виконувати операції від імені конкретного користувача, забезпечуючи коректне відображення автора заявки та аудит-трейл операцій у Jira [19].

4.8 Рекомендації щодо безпечної взаємодії з AI-системами

На основі виконаного аналізу архітектури реалізованої системи та результатів її тестування можна сформулювати низку практичних рекомендацій щодо безпечної експлуатації AI-асистентів, інтегрованих із корпоративними системами управління проєктами:

Контроль чутливих даних у природномовних запитах. Користувачам слід уникати включення паролів, API-ключів, токенів автентифікації та інших конфіденційних даних у повідомлення до AI-асистента. Незважаючи на наявність модуля безпеки, детекція не гарантує виявлення всіх форматів чутливих даних, тому первинна відповідальність за нерозголошення конфіденційної інформації залишається на стороні користувача.

Верифікація виконаних операцій. Оскільки AI-агент виконує реальні операції у зовнішній системі – створення заявок, зміну статусів, виконання переходу – користувачам рекомендується верифікувати результат кожної

операції через веб-інтерфейс Jira. Підтвердження з боку системи є індикатором успішного виклику інструменту, однак не замінює перевірку фактичного стану даних у зовнішній системі.

Принцип мінімальних привілеїв сервісного акаунту. При налаштуванні інтеграції сервісний акаунт Jira, токен API якого використовується системою, має отримувати виключно ті права, що необхідні для реалізованих операцій: створення заявок, читання заявок у межах визначеного сервісного проєкту та виконання зміни певного значення. Надання адміністративних прав сервісному акаунту суттєво розширює поверхню можливого збитку у разі компрометації системи.

Захист конфігураційних даних. токен API, email та ідентифікатори сервісних проєктів зберігаються у конфігураційних параметрах системи.. Ці дані не повинні потрапляти до систем контролю версій у відкритому вигляді. Рекомендується використання змінних середовища або систем управління секретами (Vault, AWS Secrets Manager) для зберігання та ін’єкції конфігураційних даних під час виконання.

Обізнаність щодо ін’єкції в інструкцію. Користувачі, що мають доступ до AI-асистента, повинні розуміти, що система виконує операції на основі природномовних інструкцій без додаткового підтвердження. Навмисне або ненавмисне формулювання запиту у спосіб, що імітує системну інструкцію, може призвести до небажаної поведінки моделі. У корпоративному середовищі рекомендується обмежити доступ до системи колом авторизованих користувачів.

Аудит операцій AI-агента. В середовищі кожна операція, виконана AI-агентом – виклик інструменту, параметри запиту та результат – має фіксуватись у журналі подій. Це забезпечує можливість аналізу дій системи, виявлення

аномалій та відповідність вимогам корпоративного комплаєнсу щодо аудиту автоматизованих операцій над даними.

Дотримання наведених рекомендацій дозволить мінімізувати операційні ризики при експлуатації AI-асистента в корпоративному середовищі та забезпечити відповідність системи базовим вимогам інформаційної безпеки підприємства.

Для реалізованих операцій створення та перегляду заявок у межах визначеного сервісного проєкту та виконання механізму переходів. Надання адміністративних прав сервісному акаунту суттєво розширює поверхню можливого збитку у разі компрометації системи.

Дотримання наведених рекомендацій дозволить мінімізувати операційні ризики при експлуатації AI-асистента в корпоративному середовищі та забезпечити відповідність системи базовим вимогам інформаційної безпеки підприємства.

ЗАГАЛЬНІ ВИСНОВКИ

У даній дипломній роботі було розроблено AI-асистента для підтримки управління проєктами, інтегрованого з Jira та побудованого на основі великих мовних моделей. Розроблене програмне забезпечення забезпечує виконання операцій над сервісними заявками за допомогою природномовних запитів користувача без необхідності безпосередньої роботи з графічним інтерфейсом Jira або REST API.

Архітектура системи реалізована відповідно до принципів багаторівневої модульної побудови та включає:

- AI-рівень на базі великої мовної моделі Qwen 2.5 для аналізу природномовних запитів користувача;
- серверний рівень, реалізований із використанням Java, Spring Boot та Spring AI;
- механізм виклику інструментів для взаємодії між мовною моделлю та функціональними модулями системи;
- інтеграційний рівень, який забезпечує взаємодію з Jira через REST API.

У межах роботи реалізовано функціональні модулі створення сервісних заявок, отримання та фільтрації задач, оновлення статусів через механізм отримання доступних переходів, аналітичної обробки звернень та роботи з контекстною пам’яттю діалогу. Особливу увагу приділено механізму виклику інструментів, який дозволяє великій мовній моделі автоматично визначати намір користувача, обирати необхідну операцію та формувати параметри її виконання.

Для підвищення рівня інформаційної безпеки реалізовано окремий модуль перевірки чутливих даних, який виконує аналіз тексту звернення та запобігає передачі до Jira паролів, токенів доступу та інших конфіденційних

даних. Додатково реалізовано підтримку протоколу контексту моделі (MCP), що забезпечує інтеграцію AI-асистента із зовнішніми клієнтськими застосунками.

Значну увагу було приділено практичній перевірці працездатності системи. У розділі 4 наведено результати тестування основних функціональних можливостей розробленого рішення. Під час тестування підтверджено коректну роботу механізмів створення заявок, зміни статусів, пошуку та фільтрації даних, аналітичної обробки інформації, роботи контекстної пам’яті та модуля безпеки. Результати тестування засвідчили стабільність роботи системи та коректність інтеграції великої мовної моделі з Jira.

Розроблена система дозволяє автоматизувати виконання типових операцій із сервісними заявками, скоротити час взаємодії користувачів із Jira та підвищити ефективність управління сервісними процесами за рахунок використання природномовного інтерфейсу.

Подальший розвиток роботи може бути спрямований на розширення набору доступних інструментів, удосконалення механізмів контекстної пам’яті, використання більш складних методів аналізу сервісних даних, а також інтеграцію з іншими корпоративними інформаційними системами та сервісами управління проєктами.

СПИСОК ЛІТЕРАТУРИ

1. Atlassian. Jira Software Documentation [Електронний ресурс]. – Режим доступу: https://support.atlassian.com/jira-software-cloud/
2. Atlassian. Jira Service Management Documentation [Електронний ресурс]. – Режим доступу: https://support.atlassian.com/jira-service-management-cloud/
3. Atlassian. Jira REST API Documentation [Електронний ресурс]. – Режим доступу: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/
4. Atlassian. Jira Service Management REST API [Електронний ресурс]. – Режим доступу: https://developer.atlassian.com/cloud/jira/service-desk/rest/intro/
5. Spring AI Documentation [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-ai/reference/
6. Spring Boot Documentation [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/html/
7. Alibaba Cloud. Qwen2.5 Technical Report [Електронний ресурс]. – Режим доступу: https://qwenlm.github.io/blog/qwen2.5/
8. Ollama. Ollama Documentation [Електронний ресурс]. – Режим доступу: https://ollama.com/library
9. Anthropic. Model Context Protocol Specification [Електронний ресурс]. – Режим доступу: https://modelcontextprotocol.io/
10. Agile Alliance. Agile Manifesto [Електронний ресурс]. – Режим доступу: https://agilemanifesto.org/
11. Schwaber K., Sutherland J. The Scrum Guide. – 2020. – Режим доступу: https://scrumguides.org/

12. Oracle. Java SE 17 Documentation [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/en/java/javase/17/

13. Varanasi B., Belida S. Introducing Spring Framework. – Apress, 2013.

14. Brown T. et al. Language Models are Few-Shot Learners // Advances in Neural Information Processing Systems. – 2020.

15. Zhao W. X. et al. A Survey of Large Language Models // arXiv preprint arXiv:2303.18223. – 2023.

16. Vaswani A. et al. Attention Is All You Need // Advances in Neural Information Processing Systems. – 2017.

17. Qin Y. et al. Tool Learning with Foundation Models // arXiv preprint arXiv:2304.08354. – 2023.

18. Atlassian. Jira Query Language (JQL) Reference [Електронний ресурс]. – Режим доступу: https://support.atlassian.com/jira-service-management-cloud/docs/use-advanced-search-with-jira-query-language-jql/

19. OWASP Foundation. OWASP Top 10 Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/

20. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. – O'Reilly Media, 2021.

21. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002.

22. Swagger/OpenAPI Documentation [Електронний ресурс]. – Режим доступу: https://swagger.io/docs/