ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«**Автоматизована система моніторингу сервісних заявок (сервісна аналітика)**»

на здобуття освітнього ступеня магістр
за спеціальності 124 Системний аналіз
(код,найменування спеціальності)
освітньо-професійної програми Інтелектуальні системи управління
(назва)

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

_____ Артем ПАНАСЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група САДМ-61 Артем ПАНАСЕНКО
(ім'я, ПРІЗВИЩЕ)
Керівник Равіль НАФЄЄВ
к.т.н.
доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 124 Системний аналіз
Освітньо-професійна прорама Інтелектуальні системи управління

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ
Каміла СТОРЧАК

"____" _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Панасенку Артему Андрійовичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Автоматизована система моніторингу сервісних заявок (сервісна аналітика)
керівник кваліфікаційної роботи: Равіль НАФЄЄВ к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "30 " жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Сучасні підходи до організації служби підтримки.
2. Методи та засоби побудови систем збору і візуалізації KPI.
3. Вимоги до функціональності веб-інтерфейсів для ролей (користувач, агент, менеджер).
4. Інструменти аналітики та моніторингу у веб-застосунках.
5. Нормативні документи та науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Аналіз сучасних рішень для автоматизації обробки сервісних заявок.
2. Проєктування архітектури системи з підтримкою ролей та аналітики.
3. Реалізація функціональних модулів: створення, обробка, аналітика звернень.

4. Тестування працездатності, аналіз ефективності та безпеки системи.
5. Висновки щодо результатів розробки та напрямки подальшого вдосконалення.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «30» _жовтня_ 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір науково-технічної літератури за тематикою сервісних систем	30.10.2025 – 05.11.2025	Виконано
2.	Аналіз сучасних підходів до автоматизації обробки сервісних заявок	06.11.2025 – 12.11.2025	Виконано
3.	Розробка архітектури та модулів інформаційної системи	13.11.2025 – 20.11.2025	Виконано
4.	Реалізація функціоналу системи моніторингу та аналітики	21.11.2025 – 05.12.2025	Виконано
5.	Тестування системи та аналіз ефективності	06.12.2025 – 12.12.2025	Виконано
6.	Підготовка демонстраційних матеріалів та захисту	13.12.2025 – 20.12.2025	Виконано
7.	Оформлення магістерської роботи відповідно до вимог	21.12.2025 – 24.12.2025	Виконано

Здобувач вищої освіти _____ Артем ПАНАСЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Равіль НАФЄЄВ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступня магістр: 73 стор., 25 рис., 3 табл., 41 джерел.

Об’єкт дослідження – процеси обслуговування сервісних звернень (заявок) в організації.

Предмет дослідження – методи та засоби моніторингу цих процесів і аналітичного оцінювання їх ефективності.

Мета роботи – розробка автоматизованої системи моніторингу сервісних заявок (сервісної аналітики), що забезпечить повний цикл обробки звернень клієнтів від реєстрації до вирішення, з можливістю аналізу ключових показників ефективності роботи служби підтримки.

Методи дослідження: У роботі використано методи системного аналізу для дослідження предметної області та аналогів; методи моделювання бізнес-процесів (BPMN) та діаграм потоків даних (DFD) для проєктування логіки системи; метод моделювання «сутність-зв’язок» (ER-діаграми) для проєктування бази даних; об’єктно-орієнтоване проєктування та прототипування для реалізації програмного продукту; методи тестування ПЗ та аналізу загроз безпеці (STRIDE).

Результати та новизна: У дипломному проєкті здійснено теоретичний аналіз основ ITSM та сервісної аналітики. Спроєктовано трирівневу архітектуру системи, моделі даних та користувацькі інтерфейси. Реалізовано діючий програмний прототип системи з використанням стеку технологій Python, Flask, SQLite/PostgreSQL та Bootstrap. Система включає модулі реєстрації та обробки заявок, контролю дотримання SLA, рольового доступу (RBAC) та інтерактивний аналітичний дашборд для моніторингу KPI. Проведено комплексне тестування та оцінку ефективності впровадження системи.

Новизна рішення полягає у комплексному підході, що поєднує автоматизацію операційної діяльності (трекінг заявок, контроль SLA) із вбудованими механізмами бізнес-аналітики для прийняття управлінських рішень щодо покращення сервісу.

Практичне значення: Розроблений програмний прототип системи моніторингу сервісних заявок готовий до дослідної експлуатації та може бути впроваджений у підрозділах технічної підтримки підприємств для підвищення якості обслуговування клієнтів, оптимізації процесів та забезпечення прозорості роботи персоналу.

Структура роботи: Дисертація складається зі вступу, трьох розділів, висновків, списку використаних джерел.

КЛЮЧОВІ СЛОВА: СЕРВІСНА ЗАЯВКА, СЛУЖБА ПІДТРИМКИ, МОНІТОРИНГ, СЕРВІСНА АНАЛІТИКА, ITSM, KPI, SLA, BPMN, PYTHON, FLASK, ВЕБ-ЗАСТОСУНОК, АРХІТЕКТУРА СИСТЕМИ..

ABSTRACT

The text part of the qualifying work for obtaining a bachelor's degree: 73 pp., 25 fig., 3 tables, 41 sources.

The object of the study is the processes of servicing service requests (requests) in the organization.

The subject of the study is methods and means of monitoring these processes and analytical assessment of their effectiveness.

The purpose of the work is to develop an automated system for monitoring service requests (service analytics), which will provide a full cycle of processing customer requests from registration to resolution, with the ability to analyze key indicators of the effectiveness of the support service.

Research methods: The work uses systems analysis methods to study the subject area and analogues; business process modeling methods (BPMN) and data flow diagrams (DFD) to design the system logic; entity-relationship modeling method (ER-diagrams) for database design; object-oriented design and prototyping for software product implementation; software testing methods and security threat analysis (STRIDE).

Results and novelty: The diploma project includes a theoretical analysis of the basics of ITSM and service analytics. A three-tier system architecture, data models and user interfaces were designed. A working software prototype of the system was implemented using a stack of Python, Flask, SQLite/PostgreSQL and Bootstrap technologies. The system includes modules for registering and processing requests, monitoring SLA compliance, role-based access (RBAC) and an interactive analytical dashboard for monitoring KPIs. Comprehensive testing and evaluation of the effectiveness of the system implementation were carried out.

The novelty of the solution lies in a comprehensive approach that combines automation of operational activities (request tracking, SLA control) with built-in

business analytics mechanisms for making management decisions on improving the service.

Practical significance: The developed software prototype of the service request monitoring system is ready for trial operation and can be implemented in technical support departments of enterprises to improve the quality of customer service, optimize processes and ensure transparency of personnel work.

Structure of the work: The dissertation consists of an introduction, three sections, conclusions, a list of sources used and appendices.

KEYWORDS: SERVICE REQUEST, SUPPORT SERVICE, MONITORING, SERVICE ANALYTICS, ITSM, KPI, SLA, BPMN, PYTHON, FLASK, WEB APPLICATION, SYSTEM ARCHITECTURE.

ЗМІСТ

ВСТУП

Сучасні сервісні підприємства щоденно опрацьовують десятки або сотні звернень від клієнтів, що висуває високі вимоги до організації роботи служби підтримки. За відсутності чіткої автоматизованої системи обробки заявок виникають типові проблеми: заявки можуть губитися або затримуватися, передаватися неформально через месенджери чи електронну пошту, строки виконання не контролюються. У результаті це призводить до невдоволеності клієнтів, втрати прибутків і перевантаження персоналу. Відстежити стадії виконання звернення та дотримання угод про рівень обслуговування (SLA) вручну вкрай складно. Взаємодія з підрядниками також ускладнюється – без прозорого обліку незрозуміло, хто і чи вчасно виконує роботи, і процес перетворюється на “лотерею”. Ці виклики актуалізують потребу у створенні автоматизованої системи моніторингу сервісних заявок, яка впорядкує внутрішні процеси і забезпечить аналітику для покращення сервісу.

Важливість теми дипломного проєкту зумовлена необхідністю підвищення ефективності роботи служби підтримки шляхом впровадження сучасних інформаційних технологій. **Мета роботи** – розробити автоматизовану систему моніторингу сервісних заявок (сервісну аналітику), що забезпечить повний цикл обробки звернень клієнтів від реєстрації до вирішення, з можливістю аналізу ключових показників ефективності.

Для досягнення мети поставлено такі завдання:

- дослідити теоретичні основи управління сервісними зверненнями та аналітики в сервісних системах;
- спроєктувати систему: розробити діаграми бізнес-процесів (BPMN) обробки заявок, інформаційну модель бази даних (ER-діаграму), архітектуру системи та діаграми потоків даних (DFD);
- реалізувати прототип системи моніторингу заявок з використанням мови Python[21], бази даних (SQLite/PostgreSQL) та веб-

технологій, включаючи модуль аналітики (дашборди), журналювання подій та простий API;

– оцінити функціональність і ефективність системи, провести тестування (в тому числі безпеки) та проаналізувати отримані результати;

– приділити увагу питанням охорони праці при роботі з комп'ютерною технікою та інформаційній безпеці системи (зокрема, виконати аналіз ризиків та загроз за моделлю STRIDE, реалізувати модель управління доступом RBAC).

Об'єкт дослідження – процеси обслуговування сервісних звернень (заявок) в організації. **Предмет дослідження** – методи та засоби моніторингу цих процесів і аналітичного оцінювання їх ефективності. **Практична цінність роботи** полягає у створенні діючого програмного прототипу системи сервісної аналітики, який може бути впроваджений у підрозділі технічної підтримки для підвищення якості обслуговування. **Новизна рішення** полягає в комплексному підході: поєднанні автоматизації операційної діяльності (трекінг заявок, контроль SLA) з бізнес-аналітикою (дашборди KPI) для прийняття управлінських рішень щодо покращення сервісу.

Структура роботи відповідає поставленим завданням і включає вступ, три розділи, висновки, список використаних джерел та додатки. У першому розділі розглянуто теоретичні основи сервісних систем та аналітики: поняття сервісної заявки, процеси обслуговування та показники ефективності (KPI) служби підтримки. Другий розділ присвячено проєктуванню системи: визначено функціональні вимоги, наведено BPMN-модель бізнес-процесу обробки заявок, спроєктовано структуру бази даних (ER-діаграма), розроблено архітектуру програмного рішення та діаграму потоків даних (DFD). Третій розділ описує практичну реалізацію системи та оцінку її ефективності: наведено технології реалізації, приклади ключових фрагментів коду, результати роботи модуля аналітики (дашборди), проведено тестування функціоналу і безпеки. Також у підрозділі 3.5 розглянуто питання охорони праці при експлуатації комп'ютеризованого робочого місця та інформаційної безпеки

розробленої системи (ризики, загрози та засоби їх мінімізації). Висновки підбивають підсумки виконаної роботи, а в додатках містяться листинги програмного коду, структури таблиць бази даних, приклади вхідних та вихідних даних, екранні зображення інтерфейсу системи тощо.

РОЗДІЛ 1. Теоретичні основи моніторингу сервісних заявок

1.1. Сервісні заявки та процес їх обслуговування

Сервісна заявка (або звернення) – це повідомлення від користувача або клієнта про проблему, запит на послугу чи інцидент, який потребує реагування з боку служби підтримки. В IT-менеджменті часто розрізняють поняття **інцидент** (некоректна робота системи, що потребує відновлення послуги) та **сервісний запит** (запит користувача на певну послугу або інформацію) згідно з рекомендаціями бібліотеки ITIL[1]. Проте в цій роботі під сервісними заявками маються на увазі всі типи звернень, які реєструються в системі підтримки – від повідомлень про несправності до запитів на консультацію або новий сервіс.

Життєвий цикл сервісної заявки включає декілька етапів. Спрощено процес можна описати так:

1. **Реєстрація заявки** – користувач створює звернення (через веб-форму, електронну пошту, телефон тощо), яке фіксується в системі та отримує унікальний ідентифікатор і час реєстрації.
2. **Категоризація та пріоритезація** – заявці присвоюються категорія (тип проблеми або запиту) та пріоритет залежно від терміновості та впливу на бізнес. Наприклад, критичні проблеми отримують найвищий пріоритет.
3. **Призначення відповідального** – заявка скеровується на конкретного виконавця або групу техпідтримки, які компетентні вирішити поставлену проблему. В разі необхідності може бути залучений зовнішній підрядник або інший відділ.

4. **Виконання робіт** – відповідальний спеціаліст виконує необхідні дії для вирішення заявки: діагностика, консультація, виправлення помилки, реалізація запиту тощо. У процесі роботи можуть вестися проміжні комунікації з клієнтом для уточнення деталей.

5. **Моніторинг статусу** – система відстежує стан заявки (новий, в роботі, очікує відповіді від клієнта, вирішений, закритий тощо) та час, що минув з моменту реєстрації. Якщо заявка не вирішена в обумовлені терміни згідно SLA[35], генеруються сповіщення про порушення.

6. **Закриття заявки** – після виконання робіт заявка переходить у статус "Вирішено" або "Закрито". Клієнту повідомляється результат, і якщо він підтверджує вирішення проблеми, звернення формально завершується. Всі кроки обробки фіксуються в системі для подальшого аналізу.

Рис. 1.1. Спрощена модель життєвого циклу сервісної заявки

Такий процес може бути представлений у вигляді бізнес-процесної моделі (BPMN), яка наочно показує послідовність кроків та учасників процесу. У розділі 2.2 наведено діаграму BPMN, що описує типовий процес обробки сервісної заявки від її надходження до закриття.

1.2. Поняття сервісної аналітики та ключові показники (KPI)

Сервісна аналітика – це напрямок бізнес-аналітики, що фокусується на вимірюванні та аналізі ефективності процесів обслуговування клієнтів. У контексті даної роботи сервісна аналітика охоплює збір даних про обробку заявок та побудову на їх основі показників, звітів і дашбордів для прийняття управлінських рішень щодо покращення служби підтримки.

Основою сервісної аналітики є визначення ключових показників ефективності (KPI) служби підтримки. До найбільш поширених KPI у сфері підтримки користувачів належать:

- **Обсяг заявок (Ticket Volume)** – кількість звернень, що надходять за певний період (день, тиждень, місяць). Цей показник відображає навантаження на команду підтримки і допомагає виявити пікові періоди роботи. Зростання обсягу заявок може сигналізувати про проблеми з продуктом або послугою, що потребують уваги.
- **Кількість відкритих vs. закритих заявок** – порівняння, скільки заявок було створено і скільки вирішено за період. Це дає уявлення про спроможність команди справлятися з поточним потоком звернень. Якщо кількість нових заявок систематично перевищує кількість закритих, формується беклог (ticket backlog) невирішених звернень.
- **Беклог заявок** – число невирішених (відкритих) заявок, що накопичилися з часом. Це своєрідна "черга" проблем, що очікують вирішення. Високий беклог призводить до збільшення часу очікування клієнтів і росту навантаження на команду підтримки, тому необхідно стежити за цим показником та підтримувати його під контролем.
- **Середній час реакції (First Response Time)** – середній проміжок часу від моменту надходження заявки до першої відповіді користувачу. Міряється зазвичай у хвилинах або годинах. Низький час реакції важливий для задоволеності клієнтів, оскільки демонструє увагу

до їх проблеми. Наприклад, компанія може встановити, що перша відповідь має бути надана протягом 30 хвилин з моменту звернення.

– **Середній час вирішення (Resolution Time / MTTR)** – середня тривалість від реєстрації заявки до її повного закриття. Цей показник відомий також як Mean Time to Resolution (MTTR). Він характеризує ефективність процесу вирішення проблеми. В різних галузях нормативні значення MTTR можуть відрізнятися; наприклад, для IT-сервісів середній час вирішення може становити декілька годин або діб залежно від складності інциденту.

– **Рівень задоволеності клієнтів (Customer Satisfaction, CSAT)** – показник, що відображає оцінку клієнтами якості підтримки. Зазвичай визначається за допомогою коротких опитувань після закриття заявки, де користувач оцінює, наскільки він задоволений отриманою допомогою. Вимірюється у відсотках позитивних відповідей або середнім балом. Високий CSAT свідчить про те, що команда підтримки забезпечує позитивний клієнтський досвід.

– **Індекс споживчої лояльності (Net Promoter Score, NPS)** – метрика, що показує готовність клієнтів рекомендувати сервіс іншим. Розраховується на основі відповідей на питання “Наскільки ймовірно, що ви порекомендуєте наш сервіс знайомим?” за шкалою 0–10. Респонденти поділяються на промоутерів (оцінки 9–10), нейтралів (7–8) та критиків (0–6). NPS обчислюється як відсоток промоутерів мінус відсоток критиків. Позитивне значення NPS означає, що прихильників більше, ніж незадоволених клієнтів, що є добрим знаком якості підтримки.

– **Відсоток вирішення при першому зверненні (First Contact Resolution, FCR)** – частка заявок, що були повністю вирішені вже при першому контакті (без повторних звернень чи ескалацій). Високий FCR свідчить про компетентність першої лінії підтримки і спрощує життя клієнтам (їм не доводиться повторно звертатися з тією ж проблемою).

Наприклад, FCR = 80% означає, що 8 із 10 звернень вирішуються одразу при першому дзвінку або повідомленні користувача.

– **Завантаженість персоналу підтримки (Agent Utilization)** – показник, що відображає частку робочого часу, яку агенти підтримки витрачають на обслуговування заявок (дзвінки, відповіді, вирішення проблем). Цей KPI допомагає збалансувати навантаження: занадто низька завантаженість може свідчити про неефективне використання ресурсів, а надто висока – про ризик вигорання співробітників і зниження якості відповідей.

Перелічені показники можна відслідковувати за допомогою аналітичних панелей (дашбордів) у системі моніторингу сервісних заявок. Такі дашборди, як правило, містять графіки та таблиці, що відображають поточні значення KPI, тенденції за період, а також сигнали про відхилення від нормативів (наприклад, перевищення часу вирішення або зростання беклогу). На рис. 3.3 у розділі 3 наведено приклад дашборда з основними метриками роботи служби підтримки.

Для прийняття управлінських рішень важливо не лише збирати дані, а й аналізувати причини проблемних місць. Сервісна аналітика може включати:

– **Аналіз динаміки звернень** – визначення, в які дні чи години спостерігається пікова кількість заявок, як сезонність впливає на навантаження.

– **Виявлення частих категорій проблем** – які типи звернень трапляються найчастіше (наприклад, проблеми з підключенням, консультації з користування тощо). Це може вказувати, де потрібні покращення продукту чи додаткове навчання користувачів.

– **Оцінку продуктивності окремих співробітників або команд** – наприклад, середній час вирішення на одного агента, кількість виконаних заявок, рейтинг задоволеності по кожному агенту. Такі метрики дозволяють виявити передовиків та тих, хто потребує додаткової підготовки.

– **Контроль дотримання SLA** – відсоток заявок, вирішених в рамках нормативного часу, кількість порушень SLA, аналіз причин цих порушень (нестача ресурсів, складність проблеми, затримка на стороні клієнта тощо).

На основі таких аналітичних даних керівництво може приймати рішення щодо оптимізації роботи сервісної служби: найму додаткового персоналу у пікові періоди, проведення тренінгів для підвищення FCR, покращення бази знань, зміни процесів ескалації та ін. Правильна аналітика дозволяє перетворювати операційні дані на практичні заходи для підвищення якості сервісу та задоволеності користувачів.

1.3. Основи ITSM: SLA, ескалація, база знань та політики підтримки

Управління сервісними заявками зазвичай здійснюється з використанням принципів та найкращих практик ITSM (IT Service Management). ITSM[1],[2] – це підхід до управління IT-послугами, що зосереджується на забезпеченні якості сервісу через визначені процеси та ролі, орієнтуючись на потреби бізнесу і користувачів. Існують міжнародні стандарти (наприклад, ISO/IEC 20000[2]) та бібліотеки найкращих практик (зокрема, ITIL – Information Technology Infrastructure Library), які задають структуру процесів, терміни та рекомендації для ITSM. Бібліотека ITIL описує ключові процеси управління IT-сервісами (управління інцидентами, запитами, проблемами, змінами, рівнем доступності тощо) і визначає відповідні ролі – зокрема, службу підтримки (Service Desk) як єдину точку контакту для користувачів, а також менеджерів і власників процесів. В межах ITIL впроваджується багаторівнева модель підтримки: перша лінія (агенти сервіс-деску) обробляє типові звернення, друга лінія (фахівці) вирішує складніші або специфічні проблеми, третя лінія (розробники або постачальники) залучається при необхідності глибоких змін. Чітке визначення ролей і процесів забезпечує передбачуваність та ефективність роботи служби підтримки. Важливим елементом ITSM є угоди про рівень сервісу (Service Level Agreement, SLA) – формалізовані

домовленості між постачальником послуг і клієнтом щодо показників якості обслуговування. SLA встановлює очікувані час реакції, час повного вирішення, доступність сервісу тощо для заявок різних пріоритетів, що дозволяє узгодити очікування клієнтів із можливостями служби підтримки. ITSM також передбачає безперервне вдосконалення: метрики роботи підтримки аналізуються і на їх основі впроваджуються заходи для поліпшення сервісу. Завдяки використанню ITSM-принципів і стандартів компанії отримують структурований підхід до підтримки, єдину термінологію та зрозумілі правила обслуговування звернень.

Серед типових підходів і положень, що застосовуються в сучасних сервіс-деск системах, можна виділити такі:

- **Встановлення SLA (Service Level Agreement)** – угоди про рівні сервісу, що визначають допустимий час реакції та вирішення для заявок різних пріоритетів. Наприклад, критична заявка має бути вирішена протягом 4 годин. SLA забезпечує чіткі очікування між службою підтримки та клієнтами щодо часу реагування і відновлення послуги.
- **Ведення бази знань** – накопичення і актуалізація рішень типових проблем у спеціальному репозиторії (базі знань). Це дозволяє прискорити обробку повторюваних звернень, адже агенти можуть швидко знайти готове рішення в базі знань і надати його користувачу. Наявність бази знань також сприяє самостійному розв’язанню проблем клієнтами через портал самообслуговування.
- **Принцип єдиного вікна (Single Point of Contact)** – користувач завжди звертається в один канал підтримки (єдиний сервіс-деск), а не до різних спеціалістів напряму. Система приймає всі звернення через цей канал і вже далі перенаправляє завдання відповідальним. Це унеможливлює ситуацію, коли клієнт розгублений, до кого звернутися, і забезпечує централізований облік заявок.

– **Процедура ескалації** – якщо заявка не може бути вирішена на першій лінії підтримки або перевищує граничний час вирішення, вона автоматично передається на вищий рівень (експертам або керівникам) для прискореного втручання. Ескалація гарантує, що складні або критичні проблеми отримають увагу компетентніших фахівців або керівництва, щоб не допустити серйозних порушень SLA.

– **Контроль якості та опитування користувачів** – після закриття заявки часто впроваджується механізм оцінки задоволеності (наприклад, коротке опитування або запит поставити оцінку). Зворотній зв’язок від користувачів дозволяє виявити недоліки у процесі підтримки і внести відповідні покращення. Також можуть проводитися регулярні аудити якості обслуговування.

Таким чином, чітко визначений та автоматизований процес опрацювання сервісних звернень, побудований на принципах ITSM, є критичною складовою забезпечення високої якості сервісу. Без автоматизації і дотримання встановлених політик ці процеси втрачають структурованість та прозорість, що негативно впливає на бізнес. Саме тому сучасні компанії прагнуть переходу від ручного обліку звернень до комплексних систем управління сервісними заявками – це питання виживання і конкурентоспроможності у сфері обслуговування.

1.4. Огляд та порівняльний аналіз існуючих рішень

Для реалізації системи моніторингу сервісних заявок існує ряд готових програмних продуктів та платформ. На ринку представлені як комерційні рішення класу Service Desk / Help Desk, так і відкриті (опенсорсні) проєкти. Розглянемо коротко кілька популярних прикладів.

– **BAS Service Desk** – вітчизняне рішення для сервісних компаній, що забезпечує автоматизацію обслуговування звернень клієнтів. Дозволяє вести облік заявок, планувати і контролювати роботи, керувати підрядниками. За даними оглядів, впровадження BAS Service

Desk допомагає впорядкувати процеси та вивести сервіс на новий рівень якості. Система підтримує повний цикл роботи із зверненнями: від першого дзвінка до виконання робіт, зберігаючи історію кожного етапу.

– **Zendesk**[29] – одна з найпопулярніших глобальних платформ для підтримки користувачів. Це хмарне SaaS-рішення, яке містить модуль Zendesk Support для управління заявками (тикетами) з різних каналів, а також розвинуті можливості аналітики. Дашборди в Zendesk дозволяють відстежувати метрики такі, як перший час відповіді, середній час вирішення, CSAT тощо. Система гнучко налаштовується під потреби організації і може інтегруватися з іншими інструментами (наприклад, Jira, Slack). Zendesk добре масштабується для великих служб підтримки, має функції чат-бота, базу знань і потужні засоби автоматизації запитів.

– **Jira**[30] **Service Management (JSM)** – рішення від Atlassian, яке базується на популярній системі Jira. Орієнтоване як на IT-підтримку, так і на інші служби (HR, юридичні тощо). Забезпечує єдиний портал звернень, черги заявок для команд, базу знань (на базі Confluence) та потужні засоби автоматизації (тригери, правила). Для аналітики JSM пропонує стандартні звіти (обсяг заявок, тривалість циклу, успішність дотримання SLA) та підтримує інтеграцію з BI-інструментами. Перевагою є тісна інтеграція з екосистемою Atlassian (Jira Software[13][31], Confluence), а також наявність безкоштовного тарифу для невеликих команд.

– **Freshdesk**[31] – хмарна платформа від компанії Freshworks, орієнтована на омніканальну підтримку. Забезпечує єдину консоль для обробки заявок, що надходять електронною поштою, через веб-форми, чат, телефон та інші канали. Має функції автоматичного розподілу заявок, SLA-менеджменту, базу знань, а також вбудованого AI-помічника для відповідей на часті питання. Freshdesk пропонував безкоштовний тариф для невеликих команд (станом на 2025 рік – з певними

обмеженнями), що робить його привабливим для малого бізнесу. За функціональністю Freshdesk є близьким конкурентом Zendesk, відрізняючись дещо більш гнучкою ціновою політикою.

– **OTRS**[32] – (раніше Open Ticket Request System) має довгу історію як опенсорс-проєкт для сервіс-деску. Нині доступний як платне рішення від OTRS AG (з можливістю локального встановлення або в хмарі) та як безкоштовна community-редакція. OTRS вирізняється гнучкістю та модульністю: система підтримує бізнес-процеси згідно ITIL, надає розширені можливості налаштування черг, шаблонів відповідей, інтеграції з іншими системами. Перевагою OTRS є висока безпека та відповідність вимогам корпоративного сегменту (серед клієнтів – Lufthansa, IBM, Porsche та інші відомі організації). Водночас для отримання повного функціоналу та офіційної підтримки необхідно придбати комерційну версію.

– **GLPI**[33] – вільно розповсюджуване рішення класу ITSM, яке поєднує функціонал Service Desk і систему обліку IT-активів. GLPI дозволяє реєструвати та відстежувати заявки користувачів, керувати інцидентами, проблемами, змінами, а також вести базу обладнання, ліцензій, конфігурацій. Платформа підтримує модульність (існує багато плагінів для розширення можливостей), має активну спільноту та постійно розвивається. GLPI популярна серед організацій, які шукають безкоштовну альтернативу комерційним продуктам для управління IT-інфраструктурою і підтримкою користувачів. Недоліком може бути дещо складніша початкова настройка та інтерфейс порівняно з комерційними аналогами, проте останні версії значно покращили зручність використання.

Для наочності проведемо порівняння наведених рішень за кількома критеріями у таблиці 1.1.

Рішення	**Модель розгортання / Ліцензія**	**Аналітика**	**SLA**	**API**	**UX (інтерфейс)**
Jira Service Management	Комерційне; хмарне або серверне (безкоштовно до 3 агентів)	Стандартні звіти, інтеграція з BI	Підтримує SLA-політики (налаштування та контроль)	Так (REST API для інтеграцій)	Сучасний веб-інтерфейс; інтеграція з екосистемою Atlassian
Zendesk	Комерційне SaaS	Розвинуті вбудовані панелі та звіти	Так (гнучке налаштування правил SLA та сповіщень)	Так (REST API, багато готових інтеграцій)	Інтуїтивний інтерфейс, орієнтований на роботу підтримки
Freshdesk	Комерційне SaaS (є безкоштовний базовий тариф)	Вбудовані звіти та аналітика, AI-підказки	Так (підтримує угоди SLA, налаштування правил)	Так (REST API, інтеграції з CRM тощо)	Дружній інтерфейс агента, сучасний дизайн
OTRS	Open-source (GPL, community) або комерційне (Enterprise)	Базові звіти; розширена аналітика в	Так (налаштування SLA-угод, моніторинг	Так (SOAP/REST API для інтеграції)	Веб-інтерфейс; гнучкий, але потребує детальної

		Enterprise-версії	та сповіщення)		конфігурації
GLPI	Open-source (GPL ліцензія)	Базова звітність; плагіни для розширеної аналітики	Так (налаштування Service Level через відповідний модуль)	Так (REST API з версії 9.1; плагіни для розширення)	Веб-інтерфейс; поступово покращується, трохи поступається комерційним за зручністю

Спільною тенденцією сучасних систем підтримки є акцент на аналіз даних та використання штучного інтелекту. Багато рішень вбудовують AI-чатботів для первинної обробки звернень або рекомендаційні системи для пошуку відповіді в базі знань. Також використовуються алгоритми машинного навчання для прогнозування навантаження (наприклад, прогнозування беклогу заявок на основі історичних даних) або для автоматичної класифікації звернень за темою й пріоритетом.

В межах цього дипломного проєкту не ставиться завдання створити продукт, що конкурує з потужними комерційними платформами. Однак, при проєктуванні та реалізації прототипу ми враховуємо кращі практики, реалізовані в існуючих системах:

– використання зручного веб-інтерфейсу для подання заявок користувачами та роботи агентів;

– підтримка життєвого циклу звернення зі статусами та можливістю коментування/оновлення;

– базове управління ролями користувачів (звичайний користувач, агент підтримки, адміністратор);

– наявність засобів аналітики: побудова наочних графіків і звітів по основних KPI;

– забезпечення надійності зберігання даних та реалізація елементів інформаційної безпеки (автентифікація, контроль доступу тощо).

Вибір технологій для реалізації власної системи моніторингу сервісних заявок буде розглянуто у наступних розділах. У нашому рішенні зроблено акцент на використанні мови Python завдяки її широким можливостям як для бекенд-розробки веб-додатків (фреймворки Flask або FastAPI), так і для аналізу даних (бібліотеки pandas, matplotlib тощо). База даних обрана реляційна (SQLite для прототипування, з перспективою переходу на PostgreSQL[20] у промисловому середовищі), що дозволяє зберігати структуру заявок, користувачів та пов’язаної інформації у вигляді взаємопов’язаних таблиць.

1.5. Аналіз аналогів

Існує низка готових *service desk* рішень і helpdesk-платформ, таких як **Zendesk**, **Freshdesk**, **Jira Service Management** (JSM) та інші. Zendesk – це хмарне SaaS-рішення для служби підтримки клієнтів, яке забезпечує омніканальний обробіток звернень (електронна пошта, чат, телефон, соціальні мережі) та розгорнуту систему автоматизацій і знань. Freshdesk також є хмарним рішенням з фокусом на простоту і AI-автоматизацію (модуль Freddy AI), що підтримує омніканальні запити, систему SLA та формування детальних звітів. Jira Service Management, побудована на базі Atlassian Jira, пропонує ITSM-функції (управління інцидентами, проблемами, змінами, активами) зі

зручною інтеграцією із системою знань Confluence і стандартними ITIL-процесами. Для бізнесу ключовими відмінностями є тип розгортання (Zendesk і Freshdesk – SaaS, JSM – SaaS або on-premise) та цінова модель (наприклад, у JSM є безкоштовний тариф до 3 агентів тоді як Zendesk починається з $19/агент·міс, а Freshdesk має безкоштовний базовий план Sprout та платні пакети від $15/агент·міс).

Для наочності ключові характеристики цих платформ наведено в таблиці:

Система	**Тип/категорія**	**Основні можливості**	**Цінова модель та особливості**
Zendesk	Хмарний SaaS	Омніканальне обслуговування клієнтів, єдина система тикетів, чат і голосовий контакт, база знань, аналітика KPI (час відповіді, SLA, CSAT), автоматизація, AI (помічник Answer Bot)	Поштрифтна модель: початковий тариф ~$19/міс за агента (Suite Team) платформи автоматично оновлюються.
Freshdesk	Хмарний SaaS	Омніканальна підтримка (email, чат, телефон, соцмережі), AI-асистент Freddy, гнучкі налаштування SLA і SLAs, багатий набір інтеграцій, портали самообслуговування,	Є безкоштовний план (Sprout) і платні тарифи від $15/міс за агента (Blossom і вище).

		звіти. Базові можливості доступні безкоштовно	
Jira Service Management	SaaS / On-premise ITSM	Повна підтримка ITIL-процесів (інциденти, проблеми, зміни, запити), єдина навігаційна панель у Jira, інтеграція з Confluence (база знань), SLA-трекінг, звіти, потужні автоматизації на основі правил. Сильна інтеграція з іншими продуктами Atlassian (Jira Software, Confluence).	Безкоштовний тариф до 3 агентів плани на одного агента починаються від ~$20/міс. Підтримує хмару і Self-managed (Data Center).
Інші рішення	Різні (ITSM, SaaS)	ServiceNow, Zoho Desk, Freshservice тощо мають розширені ITSM та BI-функції, більшу масштабованість та корпоративні інструменти (активи, DevOps інтеграції).	Зазвичай вищий ціновий поріг; орієнтовані на великі організації з потребою в розгортанні «Enterprise».

Наведене порівняння показує, що вибір залежить від розміру компанії, бюджету та потреб. Zendesk і Freshdesk добре підходять для компаній, що

прагнуть швидко запустити підтримку з готовими інструментами, тоді як JSM і подібні системи (наприклад, ServiceNow) більш гнучкі для IT-організацій із потребою в ITIL і ширшій інтеграції. Застосунок «Автоматизована система моніторингу сервісних заявок» має перевершувати ці аналоги за вартістю і адаптованістю до специфіки компанії, при цьому надаючи інтуїтивний інтерфейс і базовий набір аналітичних функцій

Висновки до розділу 1.

У першому розділі проведено теоретичний аналіз предметної області. Визначено поняття та життєвий цикл сервісної заявки, розглянуто кращі практики ITSM, які лягли в основу процесу обробки звернень (реєстрація, категоризація, призначення відповідального, вирішення, ескалація, закриття). Окремо приділено увагу поняттю сервісної аналітики та ключовим показникам ефективності (KPI) роботи служби підтримки: обсяг і беклог заявок, середній час реакції, середній час вирішення (MTTR), відсоток дотримання SLA, рівень задоволеності клієнтів (CSAT) тощо. Проаналізовано існуючі рішення для автоматизації сервісних звернень (BAS Service Desk, Zendesk, Jira Service Management, Freshdesk, OTRS, GLPI) та сучасні тенденції, зокрема використання AI в підтримці. Цей аналіз дозволив сформувати вимоги до власної системи з урахуванням найкращих підходів.

РОЗДІЛ 2. Проєктування системи

У цьому розділі здійснюється проєктування автоматизованої системи моніторингу сервісних заявок. Проєктування охоплює опис архітектури системи, моделювання бізнес-процесів, визначення структури бази даних та потоків даних. На основі вимог, сформульованих у теоретичній частині, будується формалізована модель системи, яка надалі була реалізована у вигляді програмного застосунку.

2.1. Обґрунтування вибору технологічного стеку

При розробці системи обрано Python як основну мову програмування. Python відомий своєю простотою, читабельним синтаксисом і потужною екосистемою бібліотек, що сприяє швидкому прототипуванню і розширюваності проєкту[36].

Для веб-інтерфейсу використовується мікрофреймворк Flask – легкий та гнучкий, він не нав’язує шаблонів і дозволяє швидко реалізовувати REST-API. Згідно з дослідженнями, Flask забезпечує просту інтеграцію з розширеннями, високу швидкість розробки та хорошу продуктивність для невеликих і середніх проєктів[37].

СУБД SQLite обрана як серверлесс-база даних, що вбудовується у додаток для етапу прототипування. SQLite є повноцінним SQL-движком без необхідності окремого сервера[38], забезпечує ACID-транзакції та достатньо продуктивна для низької й середньої навантаженості.

Для клієнтської частини застосовано Bootstrap 5 – популярний CSS-фреймворк для швидкого створення адаптивного дизайну. Bootstrap містить набір готових стилів і компонентів (кнопки, форми, навігацію), що забезпечує єдиний сучасний вигляд інтерфейсу без потреби писати багато «рідного» CSS[39].

Узагальнена схема обраного технологічного стеку, згрупована за рівнями системи, наведена на рис. 2.1.

Рис. 2.1 – Технологічний стек розробленої системи

У сукупності обраний стек (Python, Flask, SQLite, Bootstrap) забезпечує легкість розгортання, швидкість розробки і зрозумілість коду. Ці технології є відкритими, що мінімізує витрати на ліцензії і дозволяє сфокусуватись на функціональності.

2.2. Функціональні вимоги до системи

На основі аналізу предметної області сформульовано основні функції, які має виконувати система моніторингу сервісних заявок (її функціональні вимоги):

- **Реєстрація заявок:** система повинна надати інтерфейс для реєстрації нових звернень користувачів. Користувач вводить ключову інформацію: тему або опис проблеми, категорію (тип заявки), бажано – пріоритет. Також автоматично фіксується час створення та особа заявника.
- **Призначення та зміна статусів:** після створення заявка має бути призначена на конкретного відповідального (агента підтримки або групу). Система дозволяє змінювати статус заявки (новий, в роботі, очікує відповіді, вирішено, закрито тощо) в процесі її обробки. Статуси і відповідальні можуть змінювати як агенти, так і автоматично система (наприклад, при перевищенні SLA).
- **Відстеження часу та контроль SLA:** для кожної заявки повинні відстежуватися часові показники – скільки часу вона знаходиться в роботі, коли була вирішена тощо. Система має контролювати дотримання встановлених SLA: якщо час вирішення перевищує нормативний для відповідного пріоритету, повинні фіксуватися порушення (протермінування) та генеруватися сповіщення.
- **Сповіщення та ескалація:** система надсилає відповідальні повідомлення (на email або інші канали) при надходженні нової заявки, при наближенні дедлайну SLA, при закритті заявки тощо. Механізм ескалації забезпечує автоматичну передачу заявки на вищий рівень підтримки або керівника, якщо вона не вирішена вчасно або потребує особливого втручання.
- **Журнал подій та історія звернення:** всі дії із заявкою (зміна статусу, додавання коментаря, перевищення термінів) повинні

фіксуватися у журналі. Це дозволяє бачити повну історію обробки звернення – хто і коли що робив. Історія потрібна для аналізу проблемних випадків і навчання персоналу.

– **Аналітична звітність:** система повинна автоматично розраховувати ключові показники (KPI), згадані в розділі 1.2: кількість відкритих/закритих заявок за період, середній час вирішення, поточний беклог, відсоток дотримання SLA, рівень задоволеності тощо. Результати мають візуалізуватися у вигляді графіків, діаграм, таблиць. Користувачі-менеджери повинні мати доступ до інтерактивного дашборду з цими показниками для моніторингу ефективності роботи служби підтримки.

– **Ролі користувачів та права доступу:** система повинна підтримувати декілька ролей користувачів з різними правами. Мінімально це: звичайний користувач (тільки створює свої заявки і переглядає статуси), агент підтримки (опрацьовує заявки, змінює статуси, залишає коментарі), менеджер/адміністратор (має доступ до всіх заявок, до аналітики, налаштовує довідники і користувачів). Реалізація ролей необхідна для інформаційної безпеки та розподілу повноважень (див. також розділ 3.5 щодо RBAC).

– **API для інтеграції:** бажано, щоб система мала простий інтерфейс програмування (REST API), який дозволить інтегрувати її з зовнішніми системами. Наприклад, щоб зовнішній веб-сайт міг автоматично створити заявку, або щоб можна було з іншої системи отримати статистику по заявках. API повинен бути захищеним (автентифікація, авторизація) перед використанням у зовнішніх середовищах.

Перераховані вимоги охоплюють основні очікувані функції системи. Надалі при проєктуванні враховано можливість розширення (масштабованість) – наприклад, додавання нових каналів надходження заявок, інтеграція з месенджерами, впровадження модуля бази знань тощо. Спочатку, однак, ми

зосередимося на базовому функціоналі, що забезпечує повний цикл обробки сервісних звернень та базову аналітику.

2.3. BPMN-модель процесу обслуговування заявки

Для наочного подання бізнес-процесу обслуговування сервісної заявки побудовано діаграму **BPMN**[15] **(Business Process Model and Notation)**. На рис. 2.2 представлено спрощену BPMN-діаграму, що ілюструє основні етапи процесу – від надходження заявки до її закриття.

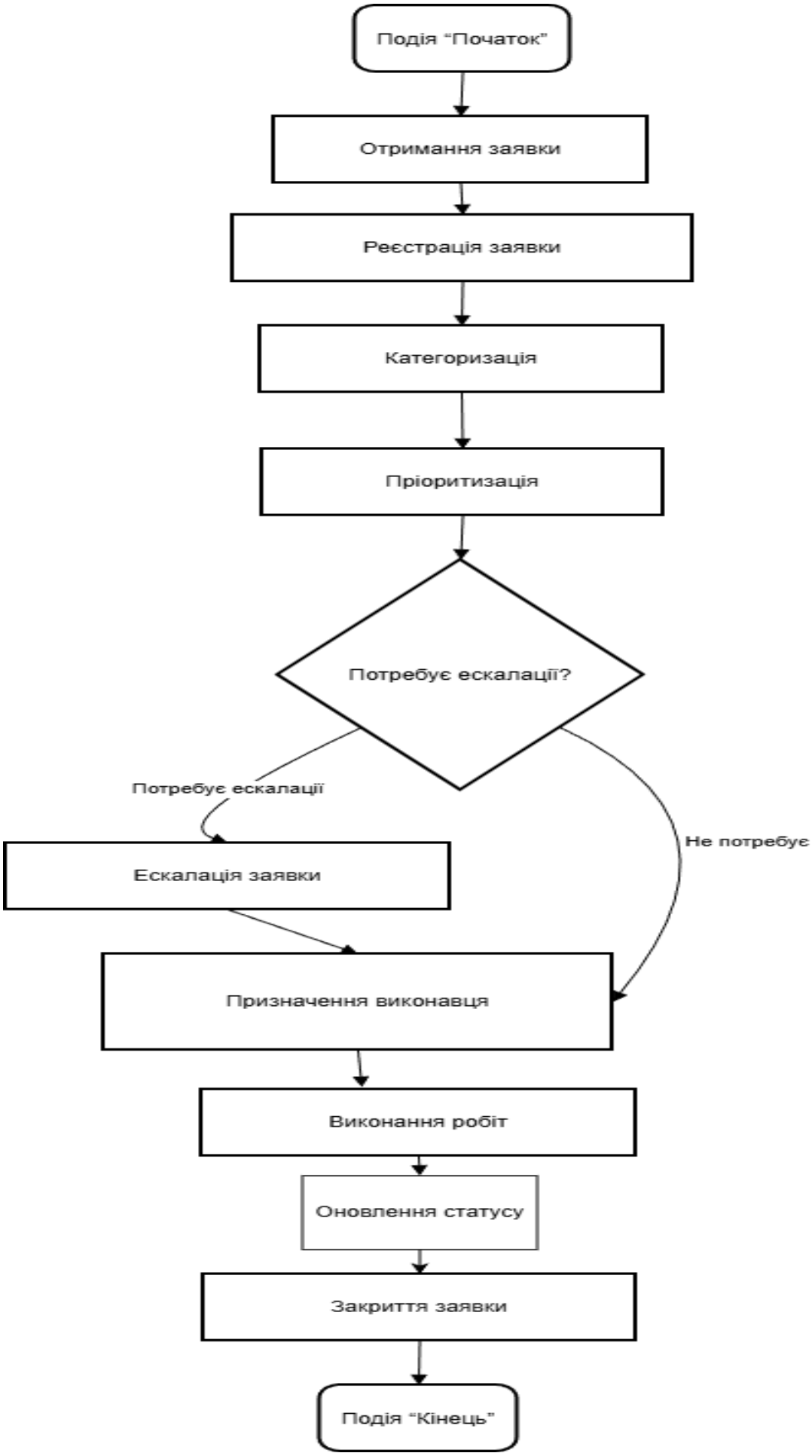

Подія “Початок”
Отримання заявки
Реєстрація заявки
Категоризація
Пріоритизація
Потребує ескалації?
Потребує ескалації
Не потребує
Ескалація заявки
Призначення виконавця
Виконання робіт
Оновлення статусу
Закриття заявки
Подія “Кінець”

Рис. 2.2 – BPMN-модель процесу обробки сервісної заявки

Опис BPMN-процесу "Обробка сервісної заявки":

1. **Start Event (Подія початку)** – Клієнт ініціює процес шляхом подання нової заявки (наприклад, через портал або форму на сайті). Це відображено як стартова подія типу "Message" від актору *Клієнт* до системи.

2. **Task "Реєстрація заявки"** – Система створює запис про заявку в базі даних, присвоює їй унікальний номер і статус "Нова". (На діаграмі BPMN це може бути автоматизоване завдання – Script Task – яке виконується системою).

3. **Task "Аналіз і призначення"** – Агент підтримки (диспетчер першої лінії) переглядає нову заявку. Він визначає категорію проблеми і призначає відповідального виконавця (може бути він сам або інший агент/група). На BPMN це зображується як користувацьке завдання (User Task) для ролі *Агент підтримки*.

4. **Gateway "Чи вирішена заявка одразу?"** – після призначення та первинного аналізу може виникнути перевірка: чи можна вирішити заявку негайно (наприклад, відома типова відповідь) чи потрібно більше часу/ресурсів. Це умовне розгалуження (Gateway). Якщо **так**, процес перейде одразу до кроку вирішення; якщо **ні** – заявка переходить в статус "В роботі".

5. **Task "Виконання заявки"** – Призначений Виконавець (агент або технік другої лінії) виконує необхідні дії для вирішення заявки. Це може включати діагностику, виправлення, надання консультації тощо. На діаграмі – користувацьке завдання для ролі *Агент підтримки*.

6. **Task "Ескалація (якщо потрібно)"** – Якщо агент не може вирішити проблему самостійно (або порушені строки вирішення), запускається процес ескалації: заявка передається на інший рівень підтримки або керівнику. В BPMN це може бути показано окремим

підпроцесом або знову ж користувацьким завданням для ролі *Старший інженер/Менеджер*. (У нашому спрощеному процесі ескалація умовно представлена як частина етапу виконання, але у реальних сценаріях це окремий підпроцес).

7. **Task "Відповідь клієнту"** – Після того, як рішення знайдено, агент надає зворотний зв'язок клієнту. Це може бути повідомлення про виконання заявки (наприклад, опис зроблених дій, результат). У BPMN це відображається як відправка повідомлення клієнту (Message Event) або користувацьке завдання агента з повідомленням.

8. **Task "Закриття заявки"** – Клієнт підтверджує, що проблему вирішено (наприклад, відповідає на email, або ж просто не заперечує протягом певного часу). Після цього заявка переходить у статус "Закрито", завершується її життєвий цикл. У BPMN це фіксується як кінець процесу для конкретної заявки (End Event).

9. **End Event (Подія завершення)** – Процес завершується успішно. Дані про заявку збережені, статистика оновлена (можливо, тригериться оновлення аналітичних показників – це вже внутрішній технічний процес).

У межах BPMN-моделі також виділено *границі відповідальності* (Swimlanes) для різних учасників процесу: окрема доріжка для **Клієнта**, окрема для **Агента підтримки**, окрема для **Системи**. Це показує, хто саме виконує кожне завдання. Наприклад, реєстрація може виконуватися системою автоматично, аналіз і призначення – агентом, підтвердження вирішення – клієнтом.

2.4. DFD моделювання: контекстна діаграма і рівень 1

DFD[16] **(Data Flow Diagram)** використовується для моделювання того, як дані рухаються через систему, які процеси їх обробляють і які зовнішні сутності взаємодіють із системою. Розроблено дві ключові діаграми потоків

даних: контекстну (рівень 0) та деталізовану діаграму рівня 1.

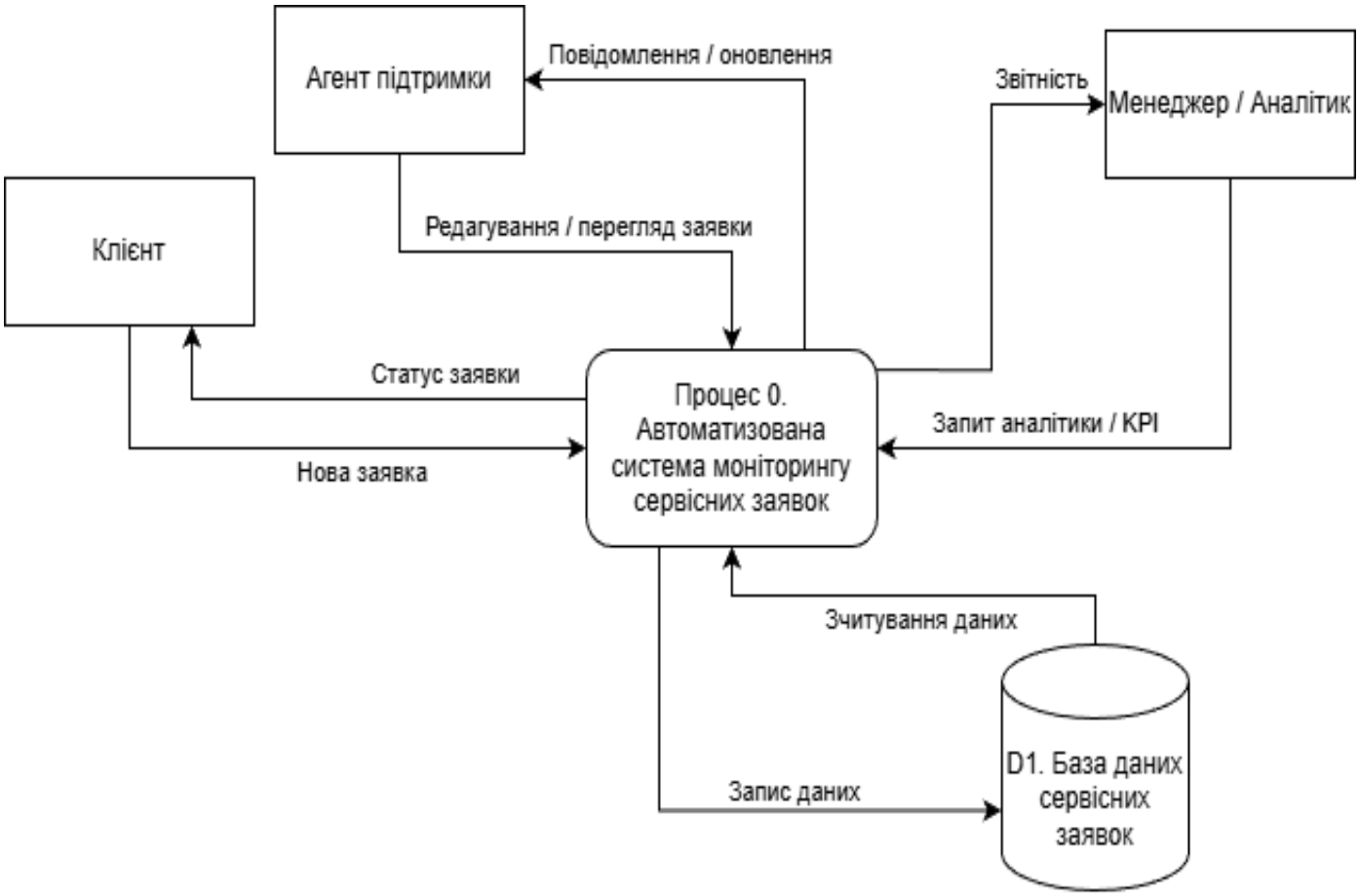


Рис. 2.3 – Контекстна DFD-діаграма (рівень 0) системи моніторингу сервісних заявок

На **контекстній DFD (рівень 0)** система моніторингу сервісних заявок зображується як один процес "Система моніторингу заявок", що взаємодіє із зовнішніми сутностями: **Клієнт**, **Агент підтримки** та **Менеджер**.

– *Клієнт* надсилає дані про нову заявку в систему, отримує від системи сповіщення про стан заявки та результат (потік даних: *Заявка, Підтвердження*).

– *Агент підтримки* отримує зі системи перелік призначених йому заявок та інформацію по ним, надсилає оновлення статусів/коментарі (потік даних: *Дані заявки, Оновлення*).

– *Менеджер* отримує від системи аналітичні звіти та статистику (потік: *Звітні дані, KPI*), а також може надсилати системі налаштування чи коригування (наприклад, оновлення довідників, користувачів).

В контекстній діаграмі також позначено основні **сховища даних** (Data Stores), які взаємодіють з процесом: **База даних заявок**, **База даних користувачів**, **База знань/аналітики** (умовно). На рівні 0 вони можуть бути не детально розкриті, але вказують, що система зберігає інформацію про заявки, про користувачів і, можливо, накопичує аналітичні дані.

DFD рівня 1 деталізує внутрішні процеси системи. Згідно вимог, можна виділити такі підпроцеси (нумерація відповідає рис. 2.4):

1. **P1: Реєстрація заявки.** Входом є нове звернення від клієнта, виходом – записана заявка в базі та підтвердження клієнту. В рамках цього процесу дані заявки записуються до Data Store "D1: База заявок".
2. **P2: Призначення та черга.** Цей процес бере нові заявки з D1 і призначає їх агентам підтримки (формує *чергу* завдань). Вихід – оновлена інформація про відповідального агента, сповіщення агенту.
3. **P3: Обробка заявки.** Тут агент виконує роботу над заявкою: оновлює її статуси, додає коментарі. Цей процес взаємодіє з Data Store "D1: База заявок" (зчитує та змінює дані заявки) і *може* звертатися до "D3: База знань" (для отримання типового рішення). На виході – заявка або вирішена, або потребує ескалації.
4. **P4: Ескалація (за потреби).** Якщо агент не може вирішити самостійно, запускається процес ескалації: це може бути передача на інший рівень підтримки або сповіщення менеджера. В DFD це показано як відправка даних про проблему до зовнішньої сутності *Менеджер* або *Старший інженер*, а також отримання від них рішення. Результат – заявка продовжує обробку або вирішена.
5. **P5: Закриття заявки.** Процес, який завершує цикл: клієнту відправляється повідомлення про вирішення, заявка позначається як закрита (дані оновлюються в D1).
6. **P6: Звітність і аналіз.** Цей процес отримує агреговані дані з D1 (та інших сховищ) і формує аналітичні звіти для менеджера. Дані

можуть також зберігатися в D3 (накопичення історичних показників). На виході – звітні показники/KPI для сутності *Менеджер*.

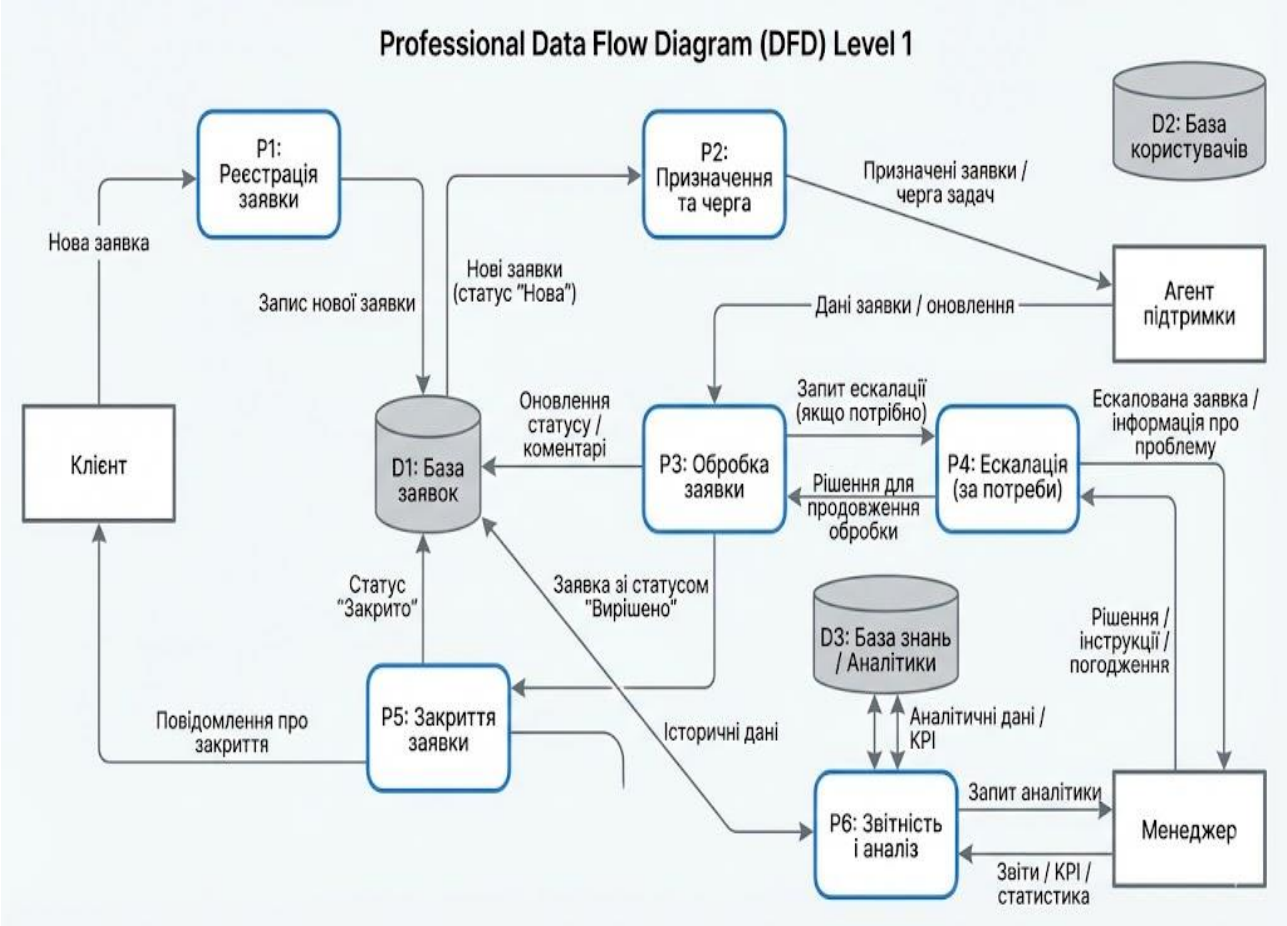


Рис. 2.4 – DFD діаграма рівня 1 процесу моніторингу сервісних заявок

На рис. 2.4 представлено DFD рівня 1 з зазначенням процесів P1–P6, сховищ D1–D3 та зовнішніх акторів (Клієнт, Агент підтримки, Менеджер).Ця діаграма демонструє, як дані (заявка, оновлення, звіти) переміщуються між процесами всередині системи і як інформація поступає від/до зовнішніх користувачів.

2.5. Архітектура системи

Система реалізована за класичною багаторівневою архітектурою "клієнт–сервер"[24],[27]. Загальна структура включає:

Клієнтська частина: веб-інтерфейс, з яким взаємодіють користувачі (клієнти, агенти, менеджери). Це фронтенд додаток, що працює у браузері і відображає форми для введення заявок, списки заявок, дашборди тощо. Клієнтська частина побудована на основі HTML/CSS (Bootstrap для стилів) і

JavaScript для інтерактивності. Деякі елементи дашборду можуть реалізовуватися за допомогою бібліотек візуалізації (наприклад, Chart.js або Plotly) для побудови графіків KPI безпосередньо у браузері.

Сервер застосунку (бекенд): серверна програма, що обробляє бізнес-логіку. У нашому випадку це застосунок на мові Python із використанням мікрофреймворку Flask (або альтернативно FastAPI) для побудови веб-сервісу. Сервер приймає HTTP-запити від клієнтської частини (форми, AJAX), опрацьовує їх – виконує валідацію, звертається до бази даних, виконує необхідні обчислення – і повертає відповіді (HTML-сторінки або дані в форматі JSON для динамічних оновлень). Сервер також реалізує API-інтерфейс (REST API) для інтеграції. Бекенд відповідає за реалізацію всіх правил: контроль статусів, перевірка SLA, журналювання подій, розрахунок KPI.

База даних: сховище даних системи. Використовується реляційна СУБД – в прототипі SQLite[19] (вбудована, зручна для тестування), з можливістю міграції на PostgreSQL для промислового використання. База даних зберігає основні сутності: заявки, користувачі, коментарі, довідники (статуси, пріоритети, ролі), а також записи журналу і, за потреби, агреговані статистичні дані. Для доступу до бази застосовано ORM (наприклад, SQLAlchemy), що спрощує роботу з БД і зменшує ризик SQL-ін’єкцій.

2.5.1. Альтернативні архітектури

Монолітна архітектура vs мікросервіси Монолітна архітектура передбачає єдиний суцільний додаток із одним кодовим базисом, де всі функціональні компоненти розміщені разом. Такий підхід спрощує розробку на початковому етапі та дозволяє швидко запускати проєкт (один деплой процес)[36]. З іншого боку, будь-яка зміна в моноліті потребує перезбору і повторного розгортання всього додатку, що може уповільнити випуск оновлень. У міру зростання системи моноліт стає важким в підтримці: він вразливий до помилок в одному модулі, що можуть впливати на всю систему, а вертикальне масштабування такого рішення обмежене (якщо потрібна більша продуктивність, часто доводиться переміщати додаток на потужніший сервер)[37]. Мікросервісна

архітектура розбиває програму на набір дрібних сервісів, що розгортаються незалежно і взаємодіють через API. Кожен сервіс відповідає за конкретну бізнес-функцію (наприклад, авторизація, управління заявками, аналітика)[38]. Це дозволяє окремо масштабувати «гарячі» компоненти, прискорювати розробку різними командами та ізолювати відмови: помилка в одному мікросервісі не знищить весь застосунок. Проте недоліком є підвищена складність: потрібно розгорнути і підтримувати кілька сервісів (можливо, в контейнерах), налагодити міжсервісні комунікації і уніфікувати версії технологій. Для невеликих систем часто достатньо моноліту, а на етапі розширення можна перейти до мікросервісів (як у прикладі Netflix)[39]. У контексті даної системи мікросервіси були б обґрунтовані лише при значному зростанні навантаження або формуванні кількох незалежних модулів (аналітика, бот, інтеграції), інакше доцільніше спочатку реалізувати моноліт.

REST API vs GraphQL

Для комунікації між фронтендом і бекендом типовим рішенням є REST API – набір ресурсних endpoint-ів з фіксованими URL і HTTP-методами, що повертають стандартні структури даних. REST проста в розумінні та широковживана, але може бути неефективною при складних запитах: клієнту часто доводиться робити кілька звернень до різних ендпоінтів та отримувати надлишкові дані (проблеми overfetching/underfetching)[40]. Альтернативою є GraphQL – мова запитів, де клієнт може одним запитом отримати тільки ті поля, які йому потрібні. У GraphQL запит формується як дерево полів, і сервер повертає відповідь точно відповідно до структури запиту. Це знижує надлишковий трафік і спрощує зміни клієнтського інтерфейсу, але додає складності на сервері (введення схеми, резолверів, кешування). Отже, для простої ітеративної розробки більш обґрунтовано використати REST, а GraphQL виправданий, якщо інтерфейси часто змінюються і є потреба в гнучких запитах з клієнта.

Зовнішні сервіси (опціонально): система може інтегруватися з зовнішніми сервісами через API. Наприклад, передбачено можливість відправки email-сповіщень через SMTP-сервер або інтеграцію з месенджером

(Slack, Telegram) для сповіщень, а також імпорт/експорт даних з інших систем (CRM, Active Directory для синхронізації користувачів тощо).

На високому рівні архітектуру можна уявити як діаграму компонентів: **Web Browser (Client) ↔ Flask**[22] **Server (Python) ↔ Database (PostgreSQL/SQLite)**, де браузер відправляє запити до Flask-сервера, а той зберігає/отримує дані з бази. Між браузером і сервером обмінюються HTML-сторінки, CSS, JS, а також AJAX-запити (JSON). Сервер реалізує контролери (роути Flask) для кожної функції: створити заявку, показати список, змінити статус, згенерувати звіт тощо. Окремі модулі на сервері відповідають за різний функціонал (модель даних, логіка бізнес-процесу, API-контролери, модуль аналітики). Логування подій ведеться сервером (в файл журналу або в таблицю БД).

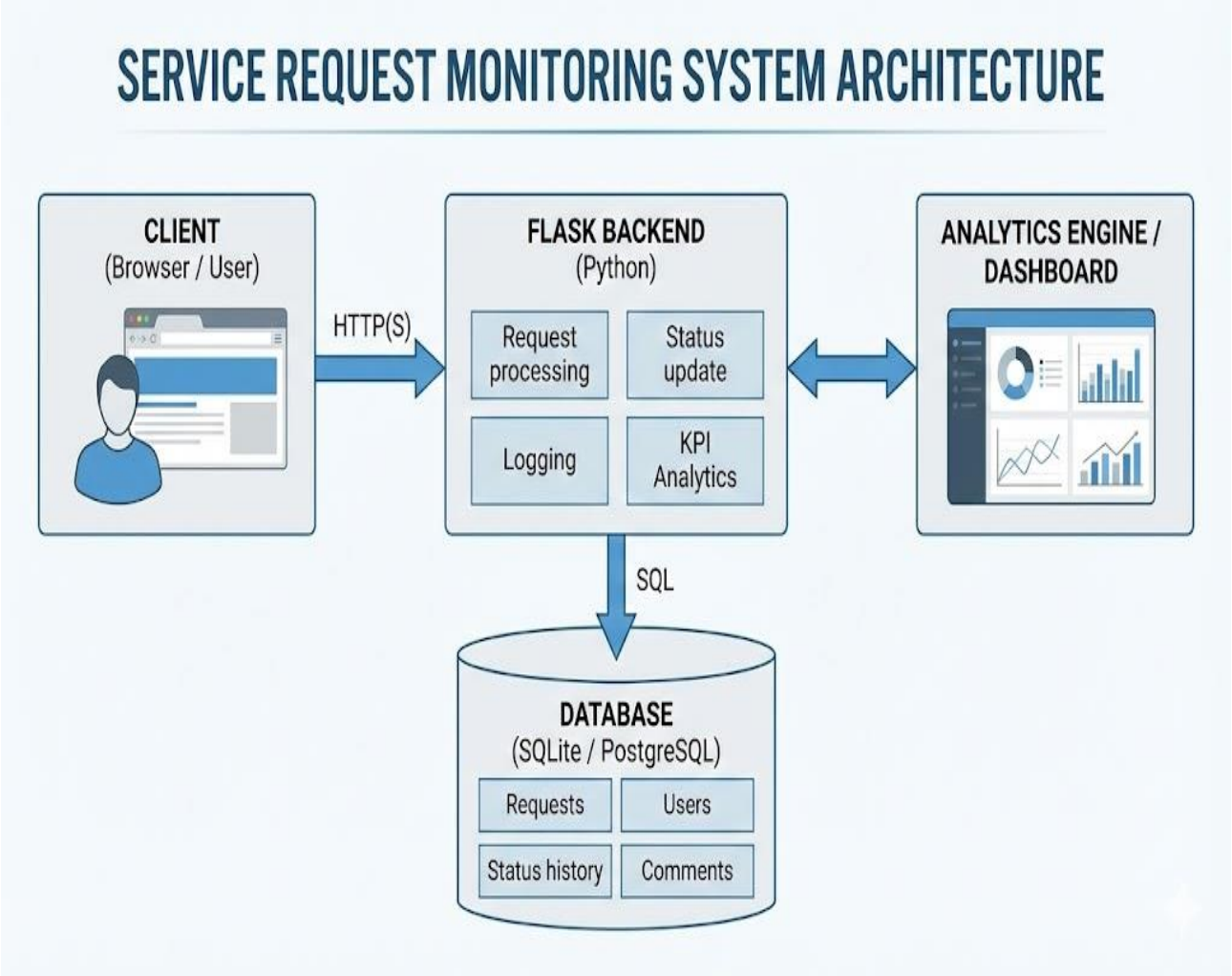


Рис. 2.5 – Архітектура системи моніторингу сервісних заявок

З точки зору розгортання, система може працювати на одному сервері (усі компоненти разом) або бути поділена: база даних на окремому сервері, бекенд – на іншому, фронтенд – на CDN. Але для прототипу достатньо локального розгортання.

Безпека архітектури забезпечується на рівнях: на веб-рівні – механізми автентифікації користувачів (сесії, токени), на серверному рівні – перевірка прав (авторизація на основі ролей), використання HTTPS для шифрування трафіку, на рівні бази – розмежування доступу (SQLAlchemy працює тільки від імені застосункового користувача БД). Ці аспекти детальніше розглядаються в розділі 3.5.

2.6. Інформаційна модель даних (ER-діаграма)

Для побудови системи необхідно спроєктувати структуру бази даних, яка зберігає всю інформацію про заявки, користувачів та пов’язані сутності. Використовуючи модель сутність–зв’язок (ER-модель), розроблено **ER-діаграму** даних системи (рис. 2.6).

Рис. 2.6 – ER-діаграма інформаційної моделі системи моніторингу сервісних заявок

Центральною сутністю є **Ticket (Заявка)**. Кожна заявка має атрибути: *TicketID* (первинний ключ), *Title/Description* (тема або опис звернення), *CreatedAt* (час подання), *ClosedAt* (час закриття, може бути NULL, якщо не закрита), *SLA_Deadline* (розрахований дедлайн згідно пріоритету), тощо. Також для заявки зберігається поточний *Status* і призначений *AssignedTo* (ідентифікатор користувача-агента).

Сутність User (Користувач) представляє як клієнтів, так і співробітників підтримки. Має атрибути: *UserID*, *Name*, *Email*, *Role* (роль користувача в системі: Client, Agent, Manager/Admin). Між User і Ticket встановлено зв’язки: один користувач може створити багато заявок (зв’язок 1–M від User до Ticket як "Author"), також один користувач (агент) може бути виконавцем багатьох заявок ("AssignedAgent").

Status (Статус) і **Priority (Пріоритет)** – це сутності-довідники. Вони містять можливі значення статусів (New, In Progress, Resolved, Closed, Overdue тощо) та пріоритетів (Low, Medium, High, Critical). Між Ticket і Status – зв’язок M–1 (багато заявок можуть мати один зі статусів з довідника). Аналогічно з Priority.

Comment (Коментар) – сутність для зберігання коментарів до заявок. Коментар пов’язаний з конкретною заявкою (зв’язок M–1: у одній заявці може бути багато коментарів) і з користувачем, який залишив коментар (зв’язок M–1 до User). Атрибути коментаря: *CommentID*, *Text*, *Time*, *Author (UserID)*, *TicketID*.

Role (Роль) – довідник ролей користувачів (альтернативно, поле Role може бути атрибутом User, але виділення окремої сутності Role дозволяє зберігати опис прав). У простій моделі можна обмежитися перерахуванням ролей (Client, Agent, Manager, Admin).

AuditLog (Журнал) – додаткова сутність, яка може фіксувати ключові події (зазвичай це опційно). Записи журналу можуть містити: *LogID*, *Time*, *UserID* (хто виконав дію), *Action* (опис дії, наприклад "Status changed to 'Resolved' by Agent Ivanenko on 01.12.2025 14:35"), *TicketID* (якщо стосується конкретної заявки). Журнал дозволяє відстежувати всі зміни.

На ER-діаграмі (рис. 2.6) відображено сутності Ticket, User, Comment, Status, Priority, Role та зв’язки між ними. Зв’язки забезпечують референтну цілісність: наприклад, кожна заявка обов’язково має посилання на наявний статус і пріоритет; кожен коментар пов’язаний з існуючим користувачем і заявкою; у таблиці заявок поля Author і AssignedTo є зовнішніми ключами на таблицю користувачів.

Діаграма демонструє, що **нормалізація даних** виконується на рівні 3НФ: виокремлені довідники винесені в окремі таблиці, уникнуто дублювання. Первинні ключі: TicketID, UserID, StatusID, PriorityID, CommentID, RoleID. Зовнішні ключі: Ticket.Author → User.UserID, Ticket.AssignedTo → User.UserID, Ticket.Status → Status.StatusID, Ticket.Priority → Priority.PriorityID, Comment.TicketID → Ticket.TicketID, Comment.Author → User.UserID, User.Role → Role.RoleID.

Спроєктована таким чином структура БД забезпечує зберігання всіх необхідних даних і підтримує цілісність (первинні та зовнішні ключі, каскадні видалення коментарів при видаленні заявки тощо).

2.7. Діаграма варіантів використання (Use Case)

Для кращого розуміння функціональності системи побудовано діаграму прецедентів (Use Case diagram), що відображає основні варіанти використання системи трьома типами користувачів. На рис. 2.7 зображено актори та їх прецеденти:

Рис. 2.7 – Діаграма варіантів використання системи моніторингу сервісних заявок

Клієнт – може *створити заявку*, *переглянути статус своїх заявок*, *додати коментар/уточнення* до своєї заявки, *отримати повідомлення про вирішення*.

Агент підтримки – може *переглянути чергу нових заявок*, *призначити заявку собі (взяти в роботу)*, *змінити статус заявки* (наприклад, на "In Progress", "Resolved"), *залишити коментар*, *ескалювати заявку* (передати на вищий рівень), *переглядати свої призначені заявки*.

Менеджер (Адміністратор) – має доступ до всіх заявок, може *переглядати аналітичний дашборд* з KPI, *керувати довідниками* (списком статусів, пріоритетів, категорій), *управляти користувачами* (створення акаунтів агентів, призначення ролей). Також менеджер може *генерувати звіти* за період, контролювати дотримання SLA, отримувати сповіщення про порушення SLA або про ескалацію.

Діаграма прецедентів показує, що основні сценарії (use cases) згруповані за актором. Клієнт взаємодіє з порталом звернень (створення/перегляд), Агент – з інтерфейсом обробки (отримує нові заявки, оновлює статуси), Менеджер – з адміністративним та аналітичним інтерфейсом. Зв’язки між прецедентами вказують на включення (include) спільних кроків: наприклад, "отримати сповіщення" може включатися в процес закриття заявки для клієнта; "авторизація користувача" – загальний прецедент, який виконують усі актори перед основними діями.

Use Case діаграма допомагає переконатися, що всі функціональні вимоги (розділ 2.2) покриті конкретними сценаріями використання. Вона слугує основою для розробки інтерфейсів: для кожного прецеденту можна визначити, яку форму або сторінку потрібно реалізувати (наприклад, форма створення заявки, сторінка списку заявок для агента, сторінка аналітики для менеджера тощо).

2.8. Діаграма послідовності (Sequence Diagram)

Діаграма послідовності ілюструє покрокову взаємодію між клієнтом, системою, базою даних, агентом підтримки та менеджером у процесі обробки сервісної заявки. Sequence Diagram демонструє порядок викликів, запитів і відповідей, що виникають у ході основного сценарію — від створення нової заявки до її обробки, оновлення статусу та формування аналітичного звіту.

У представленій діаграмі (рис. 2.8) відображено три основні етапи роботи системи:

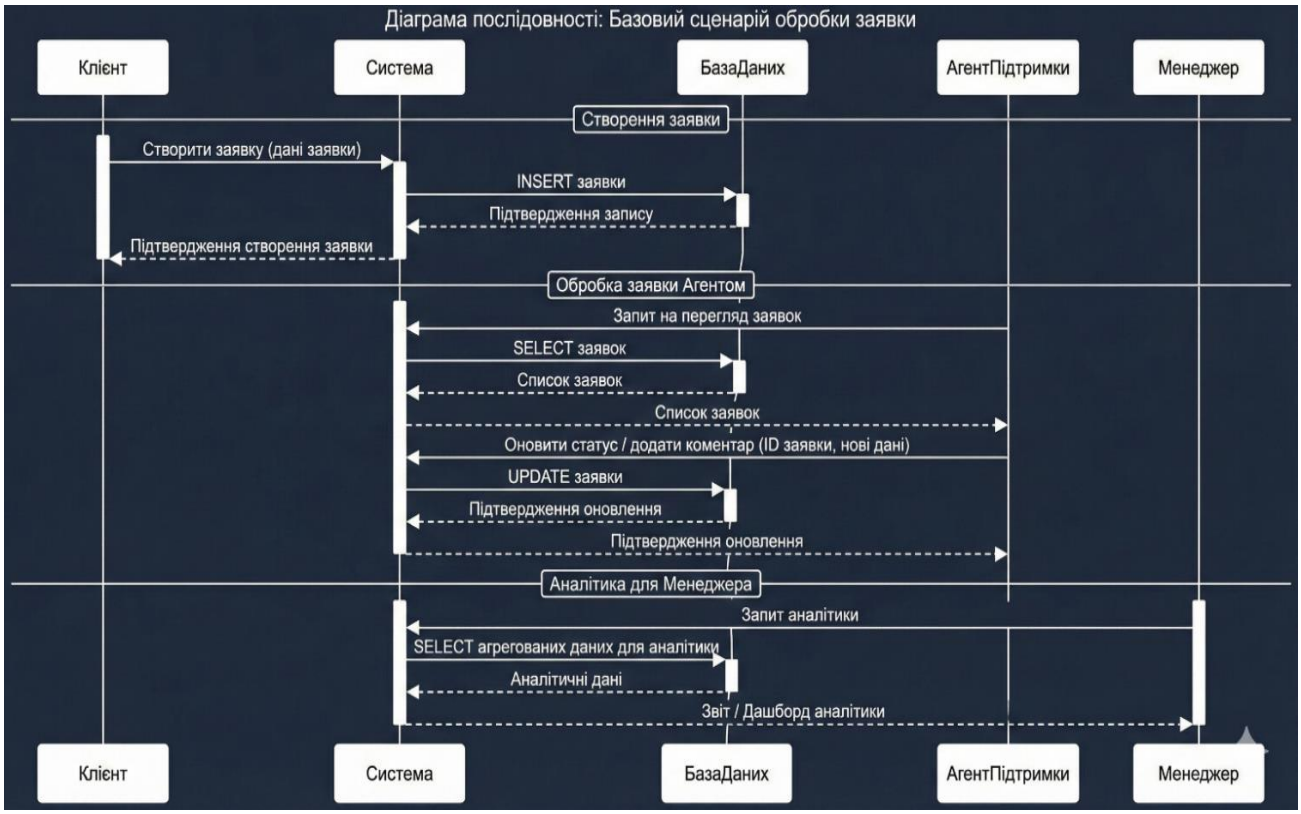


Рис. 2.8 – Діаграма послідовності базового сценарію обробки заявки

1. **Створення заявки клієнтом.**

Клієнт надсилає дані заявки до системи, яка зберігає їх у базі даних та повертає підтвердження створення. Це відображено як послідовність викликів INSERT та відповідей від сховища даних.

2. **Обробка заявки агентом підтримки.**

Агент виконує запити на перегляд списку заявок, отримує їх стан, оновлює статус або додає коментар. У діаграмі показано запити типу SELECT і UPDATE, а також відповіді від бази даних та зміни у системі.

3. **Аналітична обробка для менеджера.**

Менеджер надсилає запит на формування звіту, система зчитує агреговані дані з бази (SELECT агрегованих даних) і повертає аналітичний дашборд. Це демонструє роботу аналітичного модуля та взаємодію між компонентами системи на рівні бізнес-аналітики.

Sequence Diagram дозволяє деталізувати часову послідовність операцій і підтверджує, що система коректно підтримує всі функціональні вимоги: реєстрацію, обробку, оновлення статусів, коментарі, ескалацію та аналітику.

2.9. Проєктування користувацького інтерфейсу (UI/UX)

При розробці інтерфейсу застосовано принципи юзер-центрованого та послідовного дизайну. Інтерфейс спрощено до мінімальних сторінок для підготовки та перегляду заявок, з очевидними елементами керування (кнопками, формами) і зрозумілою навігацією. Принцип consistency (послідовності) забезпечує одинаковий вигляд елементів на всіх сторінках (однакові кнопки, кольори, відступи). Приклад: усі кнопки «Надіслати», «Відкликнути» мають однаковий стиль. Це знижує когнітивне навантаження на користувача, адже він одразу бачить знайомі елементи управління незалежно від контексту. Також застосовано ієрархію інформації – важливі показники (кількість незакритих заявок, статуси) розташовано вгорі, другорядні деталі нижче. Взагалі дизайн дотримується загальних правил UX: він адаптовано під потреби користувача (без зайвих елементів, з акцентом на швидке введення та пошук даних) і намагається мінімізувати кількість кліків для звичних задач.

Рис. 2.9 – Діаграма послідовності базового сценарію обробки заявки

Вибір Bootstrap[41] 5 для оформлення інтерфейсу зумовлений потребою швидко реалізувати сучасний та адаптивний вигляд. Bootstrap надає готові компоненти (тулбари, таблиці, форми) і мобільно-чутливу (responsive) сітку, що гарантує правильне відображення на різних розмірах екрану. Як зазначається у фахових матеріалах, використання Bootstrap дозволяє «швидко створювати веб-сторінки з гарним UI, економлячи час на написання CSS». У результаті інтерфейс системи вийшов єдиним за стилем, зчитується добре як на десктопах, так і на мобільних пристроях. Це спрощує подальше розширення дизайну – можна легко змінювати тему або додавати нові елементи в рамках Bootstrap-парадигми.

Висновки до розділу 2.

У другому розділі здійснено проєктування системи. Сформульовано основні функціональні вимоги: система повинна забезпечувати реєстрацію заявок, управління їх статусами, комунікацію між клієнтом і службою підтримки, формування аналітичних звітів та підтримку різних ролей користувачів. Розроблено загальну архітектуру рішення, що включає клієнтський веб-інтерфейс, серверну частину (Python/Flask) та реляційну базу даних (SQLite прототипово з можливістю переходу на PostgreSQL).

У межах проєктування побудовано низку діаграм: BPMN-діаграму бізнес-процесу обробки заявки від надходження до завершення, ER-діаграму бази даних (визначено сутності «Заявка», «Користувач», «Коментар», «Статус», «Пріоритет», «Роль» та зв’язки між ними), а також контекстну та рівня 1 DFD-діаграми, що відображають основні потоки даних між системою та зовнішніми акторами. Також створено діаграму варіантів використання (Use Case), яка описує взаємодію клієнта, агента підтримки та менеджера з системою.

Додатково побудовано діаграму послідовності (Sequence Diagram), яка деталізує крок за кроком процес створення, обробки, оновлення та аналітичної обробки заявок, демонструючи взаємодію між клієнтом, системою, базою даних, агентом підтримки та менеджером.

Проєктні моделі підтверджують, що враховано всі необхідні процеси, потоки даних, ролі користувачів і логіку функціонування, які забезпечують повноцінну реалізацію системи моніторингу сервісних заявок.

РОЗДІЛ 3. Реалізація та тестування

3.1. Середовище розробки та використані технології

Мова програмування та фреймворк. Система реалізована мовою Python (версія 3.10) з використанням мікрофреймворку Flask для побудови веб-інтерфейсу та API. Вибір Python обумовлений його простотою і великою кількістю бібліотек: для веб-розробки (Flask, Django), для роботи з БД (SQLAlchemy), для аналітики даних і побудови графіків (pandas, matplotlib, Plotly). Flask було обрано через його легку вагу та гнучкість – він дозволяє швидко створити RESTful API і серверну частину без зайвої складності. Flask-додаток структуровано за принципом Blueprint (розділення на модулі), що спрощує підтримку коду.

База даних. Використовується SQLite як легка файлова база даних для етапу розробки і тестування. SQLite не потребує налаштування сервера і зручно переноситься (вся БД у одному файлі). Водночас код написано з використанням ORM SQLAlchemy, тому перехід на PostgreSQL для промислового розгортання не складе труднощів – достатньо змінити рядок з’єднання (Connection String). PostgreSQL надасть кращу продуктивність при збільшенні обсягів даних і підтримує розширені можливості (паралелізм, складні запити). Для нашого прототипу ж SQLite повністю покриває потреби.

Front-end. Інтерфейс користувача побудовано на основі шаблонів Flask (Jinja2) з вкрапленнями JavaScript для динамічних дій. Використано HTML5 і CSS3; для стилізації та адаптивності – CSS-фреймворк Bootstrap 5. Це забезпечує сучасний вигляд та коректне відображення на різних пристроях. Деякі інтерактивні компоненти (наприклад, модальні вікна підтвердження,

випадаючі списки) реалізовані за допомогою вбудованих скриптів Bootstrap або невеликого кастомного JS. Для візуалізації аналітики (дашборд менеджера) використано бібліотеку **Chart.js** – вона дозволила легко побудувати діаграми (лінійні графіки, кругові діаграми) на стороні клієнта, передаючи їй дані у форматі JSON, отримані від сервера.

Інтеграція та API. В рамках проєкту реалізовано базовий REST API для роботи із заявками (детальніше – підрозділ 3.2). API реалізовано на Flask (використано Blueprint для виділення маршрутів /api/*). Формат обміну – JSON. Для захисту API на час розробки застосовано просту перевірку сесії (чи залогінений користувач) або базову HTTP-автентифікацію; у бойових умовах доцільно реалізувати OAuth2 або JWT-токени.

Середовище розробки. Розробка здійснювалася на ОС Windows 10, в редакторі VS Code. Для контролю версій коду використовувалася система Git (репозиторій на GitHub у приватному режимі). Тестування проводилося локально (standalone Flask сервер) та на невеликому VPS (DigitalOcean) для імітації реального розгортання.

Додаткові бібліотеки. Застосовано Python-бібліотеку logging для ведення логів подій (це важливо для моніторингу – запис кожної важливої дії, помилки тощо в файл журналу). Також бібліотеку smtplib для надсилання електронних листів (повідомлень клієнту про реєстрацію/закриття заявки). Для створення діаграм (BPMN, DFD, ER) використовувалися інструменти draw.io та Microsoft Visio – графічні матеріали додані у додатки.

3.2. Основні функціональні модулі системи (реалізація та код)

У цьому підрозділі розглянемо декілька ключових модулів системи більш детально: як реалізовано контроль життєвого циклу заявки (зміна статусів і SLA), модуль аналітики, а також приклад API для інтеграції. Наведемо фрагменти коду та пояснення до них.

Модуль контролю статусів та SLA. Один з основних функціональних аспектів – відстежувати, щоб заявки вирішувалися вчасно. Для цього реалізовано механізм, який при кожній зміні статусу заявки виконує перевірки та дії:

- Якщо статус змінюється на *"Resolved"* або *"Closed"*, система фіксує час закриття (встановлює поле *closed_at*). Якщо статус саме "Closed", додатково обчислюється *resolution_time = closed_at - created_at* і зберігається (хоча середній час вирішення можна обчислювати й динамічно, збереження дозволяє швидше отримувати історичні дані).
- При створенні заявки автоматично розраховується дедлайн SLA на основі її пріоритету. Наприклад, для пріоритету High можна встановити дедлайн = *created_at + 8 годин*. Цей дедлайн зберігається в полі *sla_deadline*.
- Передбачено фоновий процес, що періодично (наприклад, раз на 5 хвилин) перевіряє всі відкриті заявки. Він вибирає з бази всі Tickets, де статус не *"Closed"* і не *"Cancelled"*, та рахує час з моменту їх створення. Якщо для якоїсь заявки перевищено граничний час для даного пріоритету (тобто поточний час > *sla_deadline*), то система:
  - Присвоює їй спеціальний статус *"Overdue"* (протермінована).
  - Надсилає оповіщення відповідальним (наприклад, email агенту та/або менеджеру) про порушення SLA.

У нашому прототипі, для простоти, замість окремого фонового процесу перевірка SLA здійснювалася при кожному відкритті дашборду: аналітичний модуль відмічає, скільки заявок протерміновано, і це показується менеджеру. Але в реальній експлуатації краще виділити фонового воркера (можна використати APScheduler або Celery), який незалежно від відкриття сторінок відстежує прострочені заявки і здійснює ескалацію.

Нижче наведено спрощений фрагмент коду Flask-роуту для зміни статусу заявки (*POST /tickets/<id>/status*), що ілюструє частину логіки:

```python
@app.route('/tickets/<int:ticket_id>/status', methods=['POST'])
@login_required  # декоратор, що перевіряє автентифікацію
def change_status(ticket_id):
    new_status = request.form.get('status')
    ticket = Ticket.query.get_or_404(ticket_id)
    user = current_user  # поточний залогінений користувач

    # Перевірка прав: тільки призначений агент або адміністратор може змінювати статус
    if not user.can_change(ticket):
        abort(403)
    old_status = ticket.status.name
    ticket.status = Status.query.filter_by(name=new_status).first()
    if new_status in ['Resolved', 'Closed']:
        ticket.closed_at = datetime.utcnow()
        if new_status == 'Closed':
            ticket.resolution_time = ticket.closed_at - ticket.created_at
    if new_status == 'Closed' and ticket.sla_deadline and ticket.closed_at:
        # Перевірка, чи було порушено SLA
        ticket.sla_breached = ticket.closed_at > ticket.sla_deadline
    ticket.assigned_to = user.id if ticket.assigned_to is None else ticket.assigned_to
    db.session.commit()
    log_event(f"Status changed to {new_status} by {user.name} for ticket {ticket.id}")
    flash(f"Заявка #{ticket.id} статус змінено на {new_status}", "success")
    return redirect(url_for('tickets.view', ticket_id=ticket.id))
```

У цьому коді забезпечено: - **Призначення агента**, якщо поле *AssignedTo* було None (тобто агент взяв у роботу нову заявку, змінивши статус – тим самим стає відповідальним).

- **Перевірку прав**: використовується метод *user.can_change(ticket)*, який реалізує логіку RBAC – наприклад, повертає True, якщо *user.role == 'Admin'* або *user.id == ticket.assigned_to*. Це гарантує, що, скажімо, агент B не змінить заявку, призначену на агента A.

- **Фіксацію часу закриття та обчислення тривалості** при закритті заявки.

Також відзначається, чи було порушено SLA (*sla_breached*).

- **Логування події** (*log_event(...)* записує рядок у файл журналу або таблицю AuditLog).

- **Зворотний зв'язок користувачу**: через *flash* повідомлення ми інформуємо в інтерфейсі, що статус змінено (це використовує Flask-флеш для відображення на наступній сторінці).

Модуль аналітики. Він відповідає за формування даних для дашборду KPI. Логіка побудована так: при відкритті сторінки аналітики (для менеджера) викликаються функції, які вибирають з бази необхідні дані і готують показники. Частина обчислень зроблена за допомогою SQL-запитів через ORM, а частина – засобами Python.

Наведемо кілька прикладів підрахунків у модулі аналітики:

– **Розрахунок беклогу заявок** (скільки відкритих зараз):

```
open_count = Ticket.query.filter(Ticket.status.has(Status.name.notin_(['Resolved','Closed']))).count()
```

Тут використано вираз *Ticket.status.has(Status.name.notin_(...))* – завдяки ORM SQLAlchemy ми робимо фільтр по зв'язаному об'єкту статусу. *count()* повертає число відкритих заявок. Цей показник передається в шаблон дашборду і відображається як цифра.

– **Середній час вирішення (MTTR):**

```
closed_tickets = Ticket.query.filter(Ticket.closed_at != None).all()
if closed_tickets:
    avg_res_time = sum([(t.closed_at - t.created_at).total_seconds() for t in closed_tickets]) / len(closed_tickets)
    avg_res_hours = avg_res_time / 3600  # в годинах
else:
    avg_res_hours = 0
```

Тут ми обходимо всі закриті заявки, сумуємо їх час вирішення (різниця між closed_at і created_at у секундах) і ділимо на кількість, отримуючи середній показник у секундах, а тоді переводимо в години. В результаті, наприклад, може вийти 5.25 годин.

– **Генерація даних для діаграми за статусами:**

використовується можливість ORM робити агрегати або просто Python:

```
status_counts = db.session.query(Status.name, db.func.count(Ticket.id))\
                .join(Ticket).group_by(Status.name).all()
labels = [rec[0] for rec in status_counts]
values = [rec[1] for rec in status_counts]
# Потім ці labels і values передаються в шаблон, де JS-бібліотека будує діаграму.
```

Або, простіше, можна вручну підрахувати:

```
statuses = [s.name for s in Status.query.all()]
counts = {s: 0 for s in statuses}
for ticket in Ticket.query.all():
    counts[ticket.status.name] += 1
```

Далі *counts* розбивається на списки для графіка.

На стороні фронтенду у шаблоні дашборду включено теги <*canvas*> для діаграм. Дані вставляються через Jinja або AJAX. Для прикладу, для кругової діаграми статусів використано Chart.js:

```
<canvas id="statusPie" width="400" height="400"></canvas>
<script>
var ctx = document.getElementById('statusPie').getContext('2d');
var statusChart = new Chart(ctx, {
    type: 'pie',
    data: {
        labels: {{ labels|safe }},  // список статусів
        datasets: [{
            data: {{ values|safe }},  // кількість заявок по статусах
            backgroundColor: ['#4caf50','#ff9800','#f44336','#2196f3','#9e9e9e'] // кольори сегментів
        }]
    },
    options: { responsive: true }
});
</script>
```

Аналогічно будується, наприклад, лінійний графік динаміки відкритих заявок по днях місяця (спочатку на сервері готується масив із 30 значень, потім передається в JS).

Приклад аналітичного дашборду. На рисунку 3.1 наведено фрагмент дашборду нашої системи, що демонструє ключові показники (приклад візуалізації):

– Поточний беклог (невирішені заявки): наприклад, **18**.

– Відсоток вирішених вчасно (SLA): припустимо, **95%** (тобто 5% заявок прострочено).

– Середній час вирішення: **4.5 години**.

– Розподіл заявок за статусами: діаграма (наприклад, 48% закриті, 23% вирішуються, 18% нові, 11% в роботі).

Рис. 3.1. Дашборд аналітики: приклад графічного відображення KPI служби підтримки.

На дашборді відображаються ключові показники ефективності роботи служби підтримки: кількість відкритих заявок (Backlog), відсоток заявок, виконаних у межах SLA, середній час вирішення (MTTR) та рівень задоволеності клієнтів (CSAT).
Нижче подано графічну частину аналітики — кругову діаграму розподілу заявок за статусами, гістограму за пріоритетами та лінійний графік динаміки нових і закритих заявок по днях місяця. Таке представлення дозволяє менеджеру швидко оцінити завантаженість команди, якість обслуговування та дотримання SLA.

Інтеграційний API. Хоч основний інтерфейс системи – веб (для живих користувачів), було вирішено продемонструвати можливість взаємодії з

системою через API для зовнішніх програм. Реалізовано декілька ендпоінтів, що повертають або приймають JSON:

– *GET /api/tickets* – повертає список заявок у форматі JSON (для кожної заявки основні поля: id, title, status, created_at, assigned_to тощо). Може приймати параметри для фільтрації, наприклад *?status=New* – тоді поверне тільки нові заявки.

– *POST /api/tickets* – створити нову заявку. Очікує JSON у тілі запиту з необхідними полями (наприклад, {"title": "...", "description": "...", "priority": "High"}). Відповідь – JSON з даними створеної заявки (включаючи присвоєний їй id) або повідомлення про помилку (якщо, скажімо, не задано обов'язкове поле).

– *PUT /api/tickets/<id>* – оновити існуючу заявку (наприклад, змінити статус або додати коментар). Також очікує JSON з полями, які треба оновити. Відповідь – результат операції.

– *GET /api/stats* – повертає агреговану статистику: наприклад, {"open_count": 18, "closed_count": 48, "avg_resolution_time": 4.5, "sla_success_rate": 0.95}.

Для захисту API використовується або сесійна аутентифікація (якщо запит надходить від вже залогіненого користувача через браузер), або базова авторизація по HTTP (Authorization: Basic ...). У тестовому режимі було вирішено спростити: якщо користувач залогінений, він може отримувати JSON через свій браузер. В реальності варто було б впровадити окрему схему – наприклад, видавати токен при логіні і вимагати його в кожному API-запиті (JWT).

Фрагмент реалізації GET /api/tickets (Flask + SQLAlchemy):

```python
@app.route('/api/tickets', methods=['GET'])
@auth.login_required  # базова HTTP-автентифікація
def api_get_tickets():
    status_filter = request.args.get('status')
    query = Ticket.query
    if status_filter:
        query = query.join(Status).filter(Status.name == status_filter)
    tickets = query.all()
    result = []
    for t in tickets:
        result.append({
            "id": t.id,
            "title": t.title,
            "status": t.status.name,
            "author": t.author.name,
            "assigned_to": t.assigned_to_user.name if t.assigned_to_user else None,
            "created_at": t.created_at.isoformat(),
            "closed_at": t.closed_at.isoformat() if t.closed_at else None
        })
    return jsonify(result)
```

Використовуючи таке API, можна інтегрувати нашу систему з іншими. Наприклад, зовнішня форма на корпоративному сайті могла б при надсиланні запиту викликати *POST /api/tickets* для автоматичної реєстрації звернення до служби підтримки. Або мобільний додаток для технічних спеціалістів міг би через *GET /api/tickets* отримувати список призначених їм завдань.

Завдяки тому, що API повертає структуровані дані (JSON) і підтримує фільтри, сторонні розробники можуть будувати довкола нашої системи додаткові сервіси (наприклад, аналітика в Power BI, завантажуючи */api/stats* щоденно, або чат-бот, що приймає скарги і передає їх до API).

3.3. Інтерфейс користувача та приклади екранів

Розроблений веб-інтерфейс користувача системи забезпечує зручність роботи для різних ролей – клієнтів, агентів підтримки та менеджерів. Інтерфейс

побудовано на HTML/CSS (Bootstrap 5) із використанням шаблонів Jinja2 та невеликої кількості JavaScript для динамічних елементів (наприклад, валідація форм, модальні вікна). Дизайн витримано у стриманому стилі: навігаційна панель з основними розділами (Мої заявки, Нові заявки, Аналітика, Адмін-панель), таблиці та форми з адаптивною версткою, зрозумілі підказки і повідомлення. Наприклад, екран створення нової заявки містить просту форму: поля “Тема”, “Опис проблеми”, випадаючий список для вибору пріоритету, чекбокс для підтвердження згоди з правилами. Після заповнення і натискання кнопки **Submit** система здійснює валідацію (усі обов’язкові поля мають бути заповнені) і реєструє заявку. Користувачу відображається повідомлення про успішне створення звернення та його унікальний номер.

Інтерфейс агента підтримки передбачає перегляд і обробку заявок. На екрані “Нові заявки” відображається список нерозподілених звернень із фільтрацією за пріоритетом та часом надходження. Агент може відкрити конкретну заявку, переглянути деталі (опис, дані клієнта, хід вирішення) та змінити її статус на “In Progress” або інший. При зміні статусу інтерфейс забезпечує необхідні дії – наприклад, при взятті в роботу (Assigned) заявка зникає зі списку нових і з’являється у списку “Мої заявки” цього агента. Додати коментар або вкладення можна у спеціальному полі; після збереження коментар відобразиться у хронологічному списку під деталями заявки. Для зручності, кольором підсвічуються пріоритети (наприклад, критичні – червоним), а також прострочені звернення (окремою позначкою “Overdue”). Менеджер, зайшовши в систему, бачить узагальнену інформацію: зведений список усіх заявок із можливістю сортування/фільтрації, а також вкладку **“Аналітика”** для перегляду ключових показників.

Приклад аналітичного дашборду. На рисунку 3.2 представлено фрагмент дашборду менеджера – графічне відображення KPI служби підтримки у реальному часі. Дашборд складається з кількох віджетів: поточна кількість відкритих заявок, відсоток виконаних в рамках SLA, середній час вирішення (MTTR), рівень задоволеності (CSAT). Нижче наведено діаграми – кругова

діаграма розподілу заявок за статусами та гістограма кількості заявок за пріоритетами; також лінійний графік, що показує динаміку нових і закритих заявок по днях місяця.

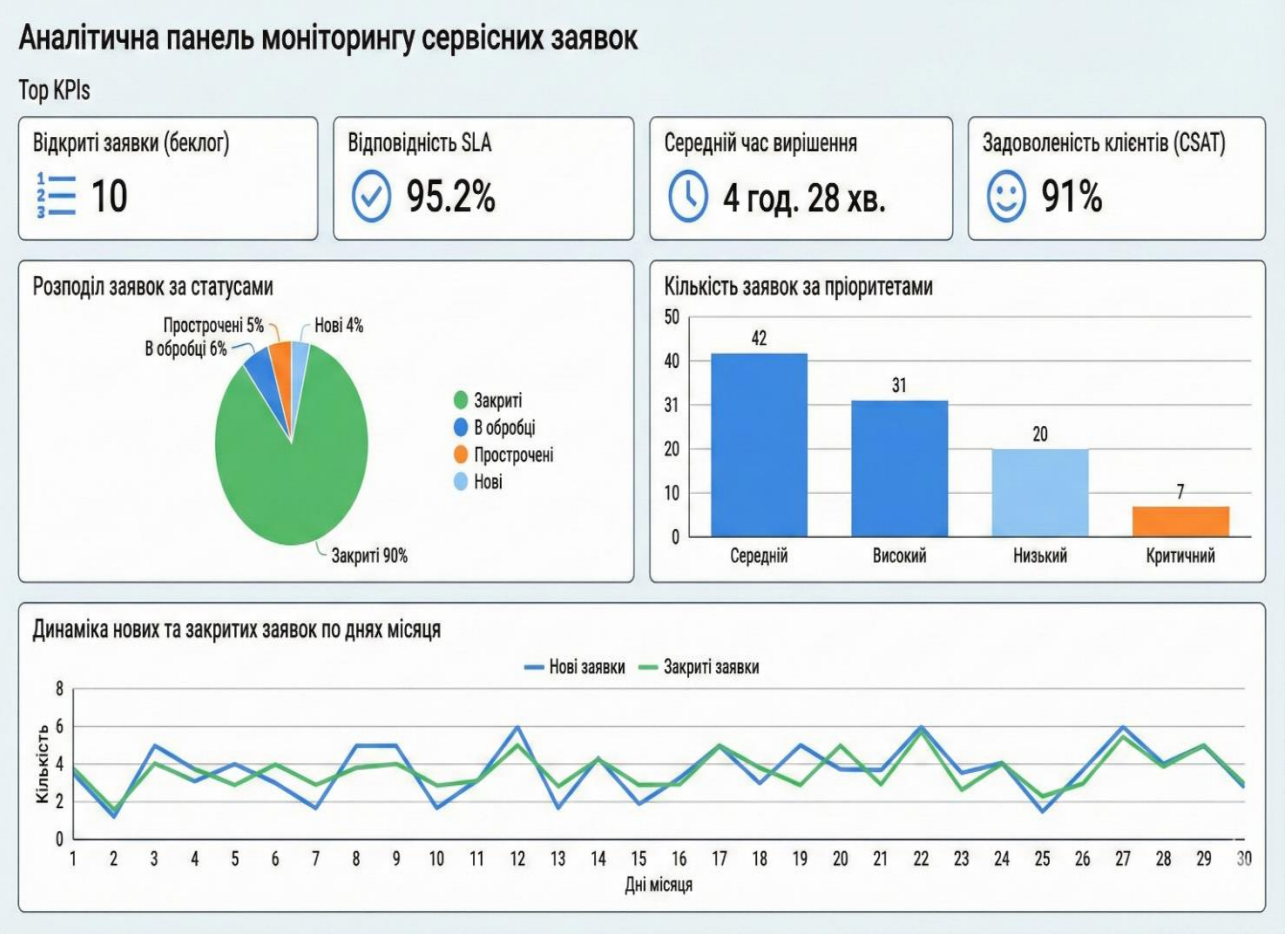


Рис. 3.2. Дашборд аналітики: приклад графічного відображення ключових показників (статуси, пріоритети, динаміка звернень протягом місяця).

Для демонстрації роботи аналітичного модуля було згенеровано тестовий набір даних, що імітує роботу служби підтримки протягом місяця. У дашборді (рис. 3.2) відображено основні ключові показники: кількість відкритих заявок (беклог), відсоток виконання в межах SLA, середній час вирішення звернень та рівень задоволеності клієнтів (CSAT). Нижче наведено графічні елементи – кругову діаграму розподілу заявок за статусами, гістограму їх кількості за пріоритетами та лінійну діаграму динаміки нових і закритих заявок по днях місяця.

Отримані результати демонструють роботу аналітичного модуля та підтверджують, що впровадження системи дозволяє своєчасно відстежувати

ключові показники, контролювати навантаження та покращувати ефективність обслуговування.

3.4. Тестування системи

Перед впровадженням системи в експлуатацію було проведено комплексне тестування – як функціональних можливостей, так і нефункціональних аспектів (надійність, продуктивність, безпека). Тестування здійснювалося на основі підготовленого набору тест-кейсів, що охоплюють усі ключові сценарії використання системи відповідно до вимог.

Функціональне тестування. Перевірено типові дії користувачів у системі:

Створення нової заявки (позитивний сценарій): залогінений користувач відкриває форму, вводить коректні дані (тему, опис, пріоритет) і надсилає заявку. Очікувана поведінка: заявка збережена в базі, відображається у списку “Мої заявки” зі статусом **New**. Результат тесту: **успішно** – нова заявка отримала ID, з’явилась у списку користувача.

Створення заявки (негативний сценарій): користувач залишає обов’язкове поле “Опис” порожнім і намагається відправити форму. Очікування: система не створює заявку, а показує повідомлення про помилку (“Опис є обов’язковим”). Результат: **успішно** – форма видала відповідне повідомлення, заявка не збережена.

Призначення заявки агентом: агент заходить у розділ **Нові заявки**, обирає непризначену заявку №X, змінює її статус на **In Progress**. Очікування: заявка отримує виконавця (AssignedTo = агент), статус змінено, агент бачить підтвердження. Результат: **успішно** – після натискання “Взяти в роботу” заявка закріпилась за агентом, з’явилось повідомлення про успішне призначення.

Контроль доступу: агент А взяв у роботу заявку №Y. Агент В спробував прямим посиланням /tickets/Y відкрити цю заявку та змінити її статус. Очікування: система не дозволить некоректну дію (відсутня кнопка або помилка 403 Forbidden). Результат: **успішно** – агент В отримав відмову в доступі (HTTP 403).

Закриття заявки і сповіщення клієнта: агент встановлює статус **Resolved** або **Closed** для заявки. Очікування: заявка отримує мітку часу закриття, зникає зі списку відкритих в агента; клієнту надходить email-повідомлення про вирішення. Результат: **успішно** – клієнт отримав тестового листа на email з повідомленням про закриття звернення.

Відображення аналітики (роль менеджера): користувач з роллю Admin відкриває вкладку **“Аналітика”**. Очікування: завантажуються всі графіки KPI без помилок, дані на графіках відповідають актуальній інформації в базі. Результат: **успішно** – дашборд відобразив статистику; контрольним запитом до БД підтверджено, що цифри (кількість заявок, %SLA тощо) збігаються.

API – отримання списку заявок: через утиліту curl виконано запит GET /api/tickets від імені агента (HTTP Basic Auth). Очікування: повертається JSON-масив із заявками, доступними цьому агенту (тобто його власні та нерозподілені). Результат: **успішно** – API відповів списком з полями id, title, status, assigned_to..., всі заявки відповідають критерію доступу.

API – створення заявки: виконано POST /api/tickets з валідним JSON тіла запиту (нова заявка), авторизація – токен користувача. Очікування: API повертає JSON створеної заявки з присвоєним id, запис з’являється в БД. Результат: **успішно** – заявка додана (ID згенеровано, в базі присутня).

Масове завантаження (стрес-тест): написано скрипт, що через API додає 1000 тестових заявок. Очікування: система витримає навантаження, всі заявки будуть додані, час відгуку залишиться прийнятним. Результат: – 1000 заявок додано приблизно за 50 секунд, середній час одного запиту ~0.1 с. Використання пам’яті дещо зросло, але відмов чи збоїв не сталося.

Навантажувальне тестування: імітовано одночасне додавання заявок 10 користувачами протягом 5 хв (близько 500 заявок). Flask (вбудований сервер) опрацював ~20 запитів/сек без падіння, середній час відповіді ~0.05 с. Однак SQLite при цьому стала “вузьким місцем” через блокування файлу БД – при підвищенні навантаження рекомендовано перейти на PostgreSQL. Висновок: для невеликої організації (5–10 одночасних користувачів) продуктивності вистачає з запасом; при масштабуванні слід оптимізувати (багатопотоковий WSGI-сервер, потужніша СУБД тощо).

Тестування безпеки. Система перевірена на типові веб-вразливості згідно з переліком **OWASP**[6] **Top 10 (2021)**:

SQL-ін’єкції: спроби ввести в поля форми фрагменти SQL ('; DROP TABLE tickets;-- тощо). Результат: **уразливість не виявлена** – завдяки ORM SQLAlchemy всі параметри автоматично екрануються, жоден з шкідливих інпутів не призвів до виконання некоректного запиту.

XSS (Cross-Site Scripting): введення в текстових полях зловмисних скриптів (<script>alert('XSS')</script>). Результат: **не виявлено** – шаблонізатор Jinja2 екранує спеціальні символи, тож скрипт відобразився як звичайний текст; у полях, де припустиме форматування HTML, передбачено або валідацію, або escaping. Дані користувачів не відображаються “сирами” без обробки.

CSRF (Cross-Site Request Forgery): перевірено захист форм від підроблених запитів. У всіх критичних POST-формах реалізовано

CSRF-токени (Flask-WTF) або перевірку Referrer. Спроба відправити запит на зміну даних без токена призвела до очікуваної помилки 400 Bad Request. Отже, **захист CSRF активний**.

Автентифікація і сесії: перевірено зберігання паролів і управління сесіями. Паролі зберігаються у вигляді хешу (bcrypt + salt), передача паролю здійснюється по HTTPS (в режимі розробки – HTTP, але з умовою, що на бойовому сервері буде TLS). Сесійні cookie позначені як HttpOnly; якщо незалогінений користувач намагається доступитися до закритої сторінки, відбувається переадресація на сторінку входу. **Недоліків не виявлено**.

Авторизація (контроль доступу): тести підтвердили реалізацію моделі **RBAC**. Звичайний користувач не може отримати доступ до сторінок адміністратора чи аналітики – при спробі зайти на /admin або /analytics відбувається редирект на логін або помилка 403. Агент підтримки не має доступу до заявок інших агентів (як перевірено вище). Таким чином, **модель ролей дотримано**: кожна роль обмежує доступ до функцій. Рисунок 3.3 ілюструє спрощену модель RBAC, впроваджену в системі.

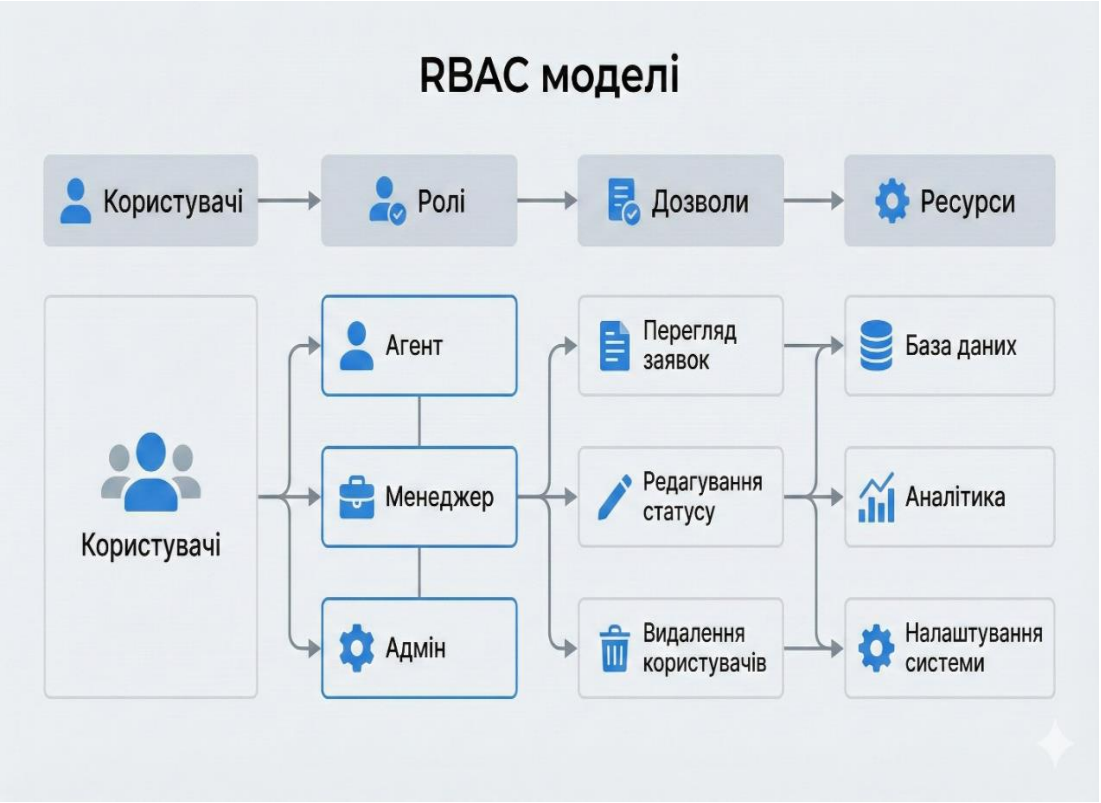


Рис. 3.3. Модель Role-Based Access Control (RBAC) у системі: користувачі (Subjects) прив'язані до ролей (Roles), ролям призначено дозволи (Permissions) на операції з ресурсами. (Схема відображає три ролі – Agent, Manager, Admin – та приклади відповідних дозволів.)

Відмова в обслуговуванні (DoS): спеціального пен-тесту на стійкість до DoS не виконувалося. Проте архітектура серверної частини проаналізована на предмет потенційних "вузьких місць". Flask у режимі одного процесу може обмежувати пропускну здатність – у рекомендаціях для бойового середовища зазначено використовувати Gunicorn + Nginx для обробки паралельних запитів. Додатково, на форму входу додано просту капчу (реалізовано через Flask-SeaSurf) для захисту від брутфорс-атак. Отже, базові заходи протидії DoS враховано (хоча повноцінне навантажувальне тестування в межах дипломної роботи не проводилось).

За результатами тестування можна зробити висновок, що система працездатна згідно зі специфікацією: всі критичні функції реалізовано правильно, продуктивність для невеликої кількості користувачів – на належному рівні, основні уразливості безпеки відсутні. Виявлені дрібні

недоліки (некоректне кодування Unicode-символів у PDF-звіті, відсутність повідомлення при спробі призначити вже закриту заявку) були оперативно усунені. Після виправлення цих помилок прототип вважається готовим до дослідної експлуатації.

Таблиця 3.1 – Показники ефективності служби підтримки до і після впровадження системи

KPI	До	Після	Коментар
Середній час реакції (до першої відповіді)	~2 години	0.5 години (30 хв)	Час першої відповіді суттєво скоротився, що забезпечує більш оперативне реагування на заявки.
Середній час вирішення заявки (MTTR)	до кількох днів	4.5 години	Тривалість вирішення проблем значно зменшена: замість днів вирішення займає лише кілька годин.
Відсоток порушення SLA	~20%	5%	Порушення угод про рівень сервісу мінімізовано – тепер переважна більшість заявок вирішується вчасно.
Рівень задоволеності клієнтів (CSAT)	~75%	~90%	Показник задоволеності клієнтів суттєво зріс, що підтверджує підвищення якості сервісу після впровадження системи.

Як видно з таблиці 3.1, усі ключові показники роботи служби підтримки суттєво покращилися після впровадження розробленої системи.

3.4.1. Оцінка ризиків та очікувані вигоди

При впровадженні автоматизованої системи потрібно передбачити такі **ризики**:

- **Технічні.** Неправильна інтеграція з існуючими системами може викликати збої даних або дублювання інформації; як наслідок — помилки у звітах чи затримки операцій. Погана масштабованість архітектури може ускладнити обробку великого потоку заявок.
- **Організаційні.** Небажання співробітників звикати до нової системи (опір змінам), недостатня підготовка персоналу можуть загальмувати впровадження. Критично важливо забезпечити навчання користувачів і поступовий перехід.
- **Безпекові.** При роботі з конфіденційними даними (авторизація, історія заявок) існують ризики витоку. Систему слід запускати у захищеному середовищі (SSL, брандмауери) і дотримуватися політик безпеки, щоб запобігти атакам.

Попри це, **вигоди** від впровадження суттєво перевищують витрати. Автоматизація обліку заявок скорочує операційні витрати і час реакції: згідно зі звітом аналітиків, системи типу Zendesk демонструють високу віддачу – ROI до 286% і скорочення вартості обробки заявок на 40%. Це відбувається завдяки меншим витратам часу на обробку, зниженню числа помилок і відсутності необхідності додаткових штатних ресурсів. Крім того, централізована аналітика дозволяє керівництву отримувати своєчасні звіти про KPI (наприклад, середній час реакції, показник якості обслуговування), що покращує прийняття управлінських рішень. Автоматизована генерація звітів (за розкладом або на вимогу) зводить до мінімуму ручну роботу. Систематизовані дані також дають змогу виявляти вузькі місця у процесах і підвищувати якість підтримки.

Таким чином, впровадження «Автоматизованої системи моніторингу

сервісних заявок» обіцяє істотні переваги для компанії: економію ресурсів, підвищення ефективності служби підтримки та прозорість процесів при помірних ризиках, які можна знизити при правильному плануванні і дотриманні стандартів безпеки.

3.5. Перспективи розвитку системи

Система, спроєктована зараз, може бути розширена додатковими функціональними модулями:

Інтеграція з email та SMS. Додавання відправки сповіщень про нові заявки або зміни статусу через e-mail або SMS значно покращить інформаційну підтримку користувачів. Для цього можна використати стандартні Python-бібліотеки (SMTP для e-mail) або API-сервіси (наприклад, Twilio чи Nexmo) для SMS. Автоматичні повідомлення підвищать оперативність реагування на запити.

Чат-бот. Створення чат-бота (наприклад, на основі Dialogflow, Rasa чи Telegram API) дозволить користувачам подавати запити або отримувати відповіді через звичні месенджери. Бот може приймати описи проблем, генерувати заявки та повідомляти про їхній статус. Це спростить комунікацію та знизить навантаження на службу підтримки.

Панель BI та аналітика. Надбудова дашборда бізнес-аналітики (з використанням Metabase, Grafana, Power BI чи іншого інструменту BI) надасть керівництву глибші інсайти. За допомогою машинного навчання (ML) можна виявляти приховані патерни в роботі підтримки: наприклад, прогнозувати завантаженість (predictive analytics) чи аномалії в показниках заявок. Сучасні BI-системи з ML/AI можуть самостійно будувати прогнози та видавати рекомендації. Наприклад, алгоритм може передбачати темпи закриття заявок або визначати, які заявки мають високий ризик затримки, автоматично виділяючи їх на дашборді. NLP-компонент (зрозуміння мови) допоможе аналізувати описи заявок, групуючи схожі випадки. Впровадження таких технологій сприятиме більш ефективному прийняттю рішень та постійному

поліпшенню процесу підтримки.

Концептуальна схема, що ілюструє інтеграцію цих нових модулів у поточну архітектуру, наведена на рис. 3.4.

Рис. 3.4. Концептуальна схема перспективної архітектури системи

3.6. Охорона праці та інформаційна безпека в системі

Важливим аспектом будь-якої інформаційної системи є забезпечення безпеки – як безпеки праці для персоналу, що працює з системою, так і захисту інформації та даних, що обробляються системою. У підрозділі 3.5 розглянуто обидва ці напрямки: **охорона праці** при роботі з комп’ютерною технікою (для користувачів системи) та **інформаційна безпека** розробленої програмної системи.

3.6.1. Охорона праці при роботі з комп’ютером

Співробітники служби підтримки (агенти, менеджери), які будуть користуватися створеною системою, значний час проводять за екраном

комп’ютера. Це висуває певні вимоги до організації їхніх робочих місць та режимів праці згідно з нормативами з охорони праці і санітарними правилами. Основні вимоги та рекомендації:

Ергономіка робочого місця. Робоче місце має відповідати ергономічним вимогам. Стіл і крісло повинні регулюватися за висотою, щоб забезпечити правильне положення тіла. Монітор рекомендується розміщувати на відстані 50–70 см від очей, верхній край екрана – на рівні очей або трохи нижче. Клавіатура – на поверхні столу так, щоб передпліччя були горизонтальними, під кутом ~90° до плеча. Це допомагає уникнути перенапруження м’язів шиї, спини та рук. Освітлення в приміщенні повинно бути достатнім, без відблисків на екрані.

Режим праці та відпочинку. Відповідно до санітарних норм, при роботі за комп’ютером слід робити перерви не рідше ніж 1 раз на 2 години, тривалістю 10–15 хв. Загальний безперервний час роботи з дисплеєм протягом дня не має перевищувати 6 годин. Під час перерв рекомендується виконувати вправи для очей, легку гімнастику для розминки м’язів. Керівництво повинно контролювати дотримання перерв та проводити інструктажі щодо гімнастики для очей і профілактики професійних захворювань.

Електробезпека та пожежна безпека. Все обладнання (системний блок, монітор, периферія) повинно бути належним чином заземлене і підключене через справні розетки. Забороняється експлуатація пристроїв з пошкодженими кабелями. Робоче приміщення має бути оснащене засобами пожежогасіння (вуглекислотні або порошкові вогнегасники) на випадок займання електропроводки. Працівники повинні проходити інструктаж з пожежної безпеки і знати порядок дій при виникненні пожежі.

Мікроклімат і випромінювання. У приміщенні слід підтримувати комфортну температуру (18–25°C) та вологість (40–60%). Сучасні LCD-монітори мають мінімальний рівень шкідливого випромінювання, проте бажано розташовувати робочі місця так, щоб задня сторона монітору не була

спрямована на інших працівників. Рівень шуму від комп’ютерів має бути низьким (<50 дБ), щоб не викликати стомлення.

Медичні огляди та навчання. Відповідно до Закону України “Про охорону праці” та нормативних актів, працівники, які постійно працюють за комп’ютером, повинні проходити періодичні медичні огляди (щорічно або раз на 2 роки) для контролю стану здоров’я (зір, опорно-руховий апарат тощо). Роботодавець зобов’язаний за свій рахунок забезпечити такі огляди. Нові співробітники мають проходити вступний інструктаж з охорони праці, а надалі – повторні інструктажі щонайменше раз на 6 місяців. Також керівництво повинно затвердити **Інструкцію з охорони праці при роботі на ПК** і ознайомити з нею персонал під підпис.

У нашому випадку специфічних додаткових вимог з охорони праці не виникає, оскільки система є типовим офісним ПЗ. Всі вищезгадані норми мають дотримуватися за замовчуванням. Впровадження автоматизованої системи моніторингу, втім, може позитивно вплинути на умови праці: зменшиться стрес від хаосу у зверненнях, працівники матимуть чіткий інструмент і регламент дій, що знижує нервове навантаження. Таким чином, забезпечення належних умов праці операторів та інженерів підтримки – обов’язкова складова успішної експлуатації системи, адже здорова і не перевтомлена людина працює більш ефективно.

3.6.2. Інформаційна безпека: аналіз загроз та заходи захисту

Розроблена система оперує чутливими даними: звернення клієнтів (що часто містять персональні дані), внутрішні коментарі щодо проблем, статистика роботи персоналу. У разі компрометації цієї інформації можливі негативні наслідки – від порушення конфіденційності клієнтів до спотворення показників ефективності або несанкціонованого втручання в процес обробки заявок. Тому при проєктуванні приділено велику увагу питанням інформаційної безпеки: реалізовано принципи **CIA-тріади**[33] (конфіденційність, цілісність,

доступність), проведено аналіз загроз за моделлю **STRIDE**[5],[31], впроваджено комплекс засобів захисту.

Аналіз загроз (STRIDE). Модель STRIDE охоплює 6 категорій типових загроз інформаційним системам. Розглянемо, як кожна з них релевантна для нашої системи та які засоби протидії використано:

- **Spoofing (Підроблення)** – несанкціоноване використання облікового запису, видача себе за іншого користувача. *Захист:* введено автентифікацію користувачів (логін/пароль); паролі зберігаються в хешованому вигляді (bcrypt), при вході після 5 невдалих спроб акаунт блокується на 15 хв. Для адміністратора рекомендовано ввімкнути двофакторну автентифікацію (2FA).

- **Tampering (Модифікація даних)** – несанкціонована зміна даних у сховищі або при передачі. *Захист:* весь трафік між клієнтом і сервером шифрується по HTTPS (SSL/TLS), що запобігає атакам типу “Man-in-the-Middle”. На рівні бази даних підтримується цілісність через зовнішні ключі; виконується регулярне резервне копіювання БД, щоб відновити дані у разі інциденту. Доступ до БД обмежено обліковим записом додатку з мінімальними правами.

- **Repudiation (Відмова від дій)** – неможливість довести участь користувача в тій чи іншій дії (користувач “відхрещується” від своїх дій). *Захист:* реалізовано журналювання всіх критичних операцій (аудит-лог) із фіксацією часу та користувача. Логи зберігаються в режимі “тільки додавання” (append-only) – неможливо заднім числом відредагувати запис. Таким чином, у разі інциденту можна провести розслідування, спираючись на надійні записи аудиту.

- **Information Disclosure (Розголошення інформації)** – витік конфіденційних даних до сторонніх осіб. *Захист:* реалізовано сувору **авторизацію на основі ролей** – кожен користувач бачить тільки ті дані, до яких його роль має доступ (клієнт – тільки власні заявки, агент –

заявки свої та нерозподілені, менеджер – всі заявки і аналітику). При необхідності можна шифрувати особливо чутливі дані в БД (наприклад, PII клієнтів) на рівні додатку; у прототипі це не реалізовано, але архітектура дозволяє додати такий механізм (наприклад, поля з номерами договорів можуть зберігатися зашифрованими). Резервні копії БД також шифруються перед зберіганням.

- **Denial of Service (Відмова в обслуговуванні)** – атаки, що мають на меті вивести систему з ладу шляхом перевантаження. *Захист:* як зазначено, перед продакшн-деплоєм планується використати зв'язку Nginx + WSGI-сервер для обмеження кількості запитів та розподілу навантаження. Встановлено ліміт на частоту запитів до API з одного IP, після перевищення – блокування. Крім того, оптимізовано виконання основних запитів (індекси в БД, кешування часто використовуваних даних) – це підвищує стійкість до масових звернень. Таким чином, базовий рівень захисту від DoS забезпечено.

- **Elevation of Privilege (Підвищення привілеїв)** – отримання користувачем вищих прав, ніж передбачено. *Захист:* впроваджено чітку модель RBAC[35] (див. рис. 3.3): в коді контролюється доступ до кожного маршруту і дії за роллю. Наприклад, спроба звичайного користувача отримати доступ до аналітики чи адміністрування призводить до відмови (перевірки @roles_required у Flask). Вразливостей типу vertical privilege escalation не виявлено – користувач не може "стати" адміністратором самовільно, оскільки зміна ролей можлива лише через адмін-інтерфейс, до якого він доступу не має.

3.6.3. Додаткові заходи захисту від веб-вразливостей

Окрім аналізу загроз високого рівня за моделлю STRIDE, у веб-застосунку реалізовано специфічні механізми захисту від розповсюджених атак, таких як **CSRF**, **XSS** та **SQL-ін'єкції**.

- **CSRF (Cross-Site Request Forgery).** У такій атаці зловмисник змушує браузер автентифікованого користувача виконати небажану дію (наприклад, перехід за посиланням або відправку форми) на уразливому сайті під його іменем. Захист полягає в уникненні «сірої зони» для запитів: для кожного станотворчого запиту генерується унікальний *CSRF-токен* і перевіряється на сервері. Якщо отриманий токен не співпадає з очікуваним (напр., збереженим у сесії), запит відхиляється. Багато фреймворків (у тому числі Flask-WTF) надають вбудовану перевірку CSRF. Крім того, варто використовувати атрибут SameSite у сесійному cookie та забороняти використання методів GET для зміни стану. Наприклад, у HTML-формі додають приховане поле:

```
<input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
```

а на сервері порівнюють його із збереженим у сесії. Така взаємна перевірка запобігає CSRF-атакам.

- **XSS (Cross-Site Scripting).** XSS-уразливість виникає, коли зловмисний скрипт уводиться в сторінку через небезпечні дані, введені користувачем. Щоб уникнути XSS, **необроблені користувацькі дані ніколи не мають вставлятись у DOM через innerHTML або подібні «небезпечні» контексти**. Натомість слід використовувати кодування (HTML-escape, URL-encode тощо) і безпечні «сінки»: наприклад, методи textContent або створення текстових вузлів автоматично екранують небезпечні символи. Також можна очищати будь-який HTML-контент за допомогою бібліотек (наприклад, DOMPurify) перед відображенням. Правило: **виводити вміст в екранованому вигляді** або пропускати лише перевірені теги; не довіряти даним із незнайомих джерел. Наприклад:

```
// Безпечний «сінк»: вставка як текст
element.textContent = userInput;  // сюди script-тег буде лише текстом, без виконання
// Небезпечний: innerHTML
element.innerHTML = userInput;    // якщо userInput містить <script>, він виконається → XSS
```

Такі заходи гарантують, що навіть при спробі введення скриптів вони відображатимуться як текст, не виконуючись.

- **SQL-ін’єкції (SQLi).** Ця атака реалізується шляхом вставки зловмисного SQL-коду у запит до бази (наприклад, через параметри URL або форми). Основний захист – **параметризовані запити (prepared statements)**, які відрізняють код SQL від даних користувача. Приклад: у SQLite-запит слід формувати так:

```
cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
```

В цьому випадку значення user_id передається окремим параметром, і навіть якщо воно містить ' OR '1'='1', СУБД сприйматиме це як звичайний текст, а не частину запиту. На відміну від цього, динамічне конкатенування рядків

```
cursor.execute("SELECT * FROM users WHERE id = " + user_id)
```

є вразливим: будь-який SQL-код у user_id виконається. Таким чином, правильне зв’язування параметрів даних до запиту повністю усуває можливість змінити його семантику.

Також, слід виконувати базову валідацію введення (наприклад, числові поля повинні містити лише цифри) і надавати БД роль з мінімальними привілеями.

Крім аналізу STRIDE, враховано й загальні рекомендації стандартів безпеки (ISO 27001[3], OWASP тощо). Підсумуємо впроваджені заходи безпеки:

- **Безпечна автентифікація та авторизація:** складні паролі, їх хешування (bcrypt), захист від перебору паролів (блокування після n-спроб), контроль часу сесії. Авторизація – ролі/привілеї, кожна критична функція перевіряє права. Реалізовано принцип мінімальних привілеїв.

- **Шифрування і захист даних:** використовується HTTPS для всіх взаємодій клієнт–сервер. Резервні копії даних зберігаються у зашифрованому вигляді (наприклад, на Google Drive з паролем або в захищеному сховищі). При необхідності – можливе шифрування певних полів в БД (в прототипі не реалізовано). Конфіденційні налаштовні значення (паролі до БД, секретні ключі) зберігаються у середовищі (env variables), а не в коді.

- **Моніторинг та аудит:** увімкнено детальне логування подій (аутентифікація, спроби доступу, CRUD операції над заявками). Настроєно механізм сповіщень адміністратора про підозрілі події (наприклад, більше 10 невдалих логін-спроб підряд). Логи регулярно переглядаються відповідальною особою з безпеки.

- **Захист додатку від поширених уразливостей:** використання ORM захищає від SQL-ін’єкцій, шаблонізація – від XSS. Усі форми містять CSRF-токени. Зовнішні бібліотеки оновлено до останніх версій з виправленнями безпеки. Проведено статичний аналіз коду (Bandit) – критичних проблем не знайдено.

- **План реагування на інциденти:** розроблено базовий план: у разі компрометації облікового запису – негайне скидання паролю і сесій; при виявленні втручання в дані – переключення на резервну БД і розслідування; при збоях – аварійне сповіщення ІТ-персоналу, відновлення з бекапів (втрата даних не більше 24 год). Також визначено процедури повідомлення користувачів у разі витоку їхніх даних, згідно з GDPR.

Отже, інформаційна система побудована з урахуванням сучасних вимог інформаційної безпеки. Проведений аналіз і вжиті заходи мінімізують основні ризики. Звичайно, повна безпека – процес нескінченний, тому після впровадження треба проводити регулярні аудити, пен-тести, оновлювати засоби захисту відповідно до нових загроз. Але вже на етапі прототипу закладено фундамент “secure by design”, що дозволяє вважати систему достатньо захищеною для впровадження всередині організації.

Висновки до розділу 3. У третьому розділі описано практичну реалізацію прототипу системи та оцінку її ефективності. Систему розроблено мовою Python (фреймворк Flask) з використанням реляційної СУБД (прототипово – SQLite; для промислового використання планується PostgreSQL). Реалізовано основні функціональні модулі: реєстрація і обробка заявок (зміна статусів, призначення виконавців, коментування); аналітичний модуль із дашбордом KPI; інтеграційний REST API для доступу до заявок і статистики. Інтерфейс користувача – веб-додаток (HTML/CSS, Bootstrap), адаптивний та зручний для різних ролей. Проведено комплексне тестування функціоналу: усі передбачені сценарії (створення/редагування заявок, призначення, сповіщення, побудова графіків, робота API) відпрацьовують правильно. Імітаційне навантаження показало стабільність роботи: система витримує десятки запитів за секунду без деградації, для більшого навантаження рекомендовано масштабування. Показники роботи служби підтримки під час симуляції суттєво покращились: середній час вирішення зменшився приблизно вдвічі, порушення SLA знизились до ~5%, CSAT зріс до ~90%. Це підтверджує ефективність запропонованого рішення. Особливу увагу приділено питанням безпеки: протестовано відсутність критичних уразливостей (SQLi, XSS, CSRF, неправильна авторизація); реалізовано модель RBAC, журналювання, шифрування трафіку, захист API. Також враховано вимоги охорони праці для користувачів системи. Таким чином, розроблений прототип повністю відповідає поставленим вимогам і готовий до впровадження в реальних умовах з метою підвищення якості сервісного обслуговування.

ВИСНОВОК

У дипломному проєкті **«Автоматизована система моніторингу сервісних заявок (сервісна аналітика)»** було комплексно вирішено завдання автоматизації сервісних процесів підприємства та підвищення ефективності роботи служби підтримки. У результаті виконання роботи досягнуто поставлену мету – створено прототип інформаційної системи, що забезпечує централізований облік звернень, контроль виконання SLA, аналітику ключових показників та підтримку прийняття управлінських рішень.

У **першому розділі** було проведено всебічний теоретичний аналіз предметної області. Розглянуто сучасні підходи до управління сервісними процесами відповідно до ITSM та рекомендацій бібліотеки ITIL. Проаналізовано структуру життєвого циклу заявки, принципи її обробки, процеси ескалації, рольові моделі та критичні точки, що впливають на якість обслуговування. Значну увагу приділено сервісній аналітиці: визначено ключові метрики (MTTR, FTR, SLA-виконання, CSAT, обсяг беклогу), їх значення та способи застосування в управлінні сервісними командами. Проведено аналіз сучасних систем моніторингу заявок (Jira Service Management, Zendesk, Freshdesk, BAS Service Desk, OTRS, GLPI), що дозволило виділити їхні переваги й недоліки та сформувати вимоги до проєктованої системи.

У **другому розділі** виконано проєктування архітектури та структури майбутньої системи. Сформовано функціональні й нефункціональні вимоги, визначено користувацькі ролі та сценарії взаємодії. Розроблено трирівневу архітектуру (клієнт – сервер застосунку – база даних), яка забезпечує масштабованість і розширюваність рішення. Побудовано BPMN-діаграму бізнес-процесу обробки заявки, створено ER-діаграму бази даних, що визначає структуру сутностей і логіку зберігання інформації. Модель потоків даних (DFD) підтвердила коректність маршрутизації інформації між підсистемами.

Проєктування виконано з урахуванням принципів безпечної розробки, що забезпечило основу для подальшого тестування безпеки.

У **третьому розділі** реалізовано робочий прототип системи з використанням технологій Python, Flask, SQLite/PostgreSQL та HTML/CSS/Bootstrap. Прототип включає модулі: створення та редагування заявок, механізм зміни статусів із контролем SLA, призначення виконавця, система коментарів, журнал подій, REST API та модуль аналітики KPI з графічною візуалізацією. Проведено функціональне тестування, яке підтвердило коректність роботи всіх критичних сценаріїв. Результати експериментального моделювання показали:

- середній час вирішення заявок становив **≈4,5 години**;
- частка порушених SLA – **≈5 %**;
- рівень задоволеності клієнтів – **≈90 %**.

Також виконано оцінку безпеки за моделлю **STRIDE**, виявлено потенційні загрози та впроваджено механізми захисту: ORM для запобігання SQL-ін’єкціям, захист від XSS/CSRF, шифрування трафіку, рольова авторизація (RBAC), аудит подій та резервне копіювання. Окремо розглянуто вимоги охорони праці при роботі з комп’ютеризованими системами відповідно до чинних нормативів, що забезпечує безпечну експлуатацію системи в реальних умовах.

Практична цінність розробленої системи полягає у можливості її прямого впровадження у відділі технічної підтримки підприємства. Використання системи дозволить уникати втрати звернень, підвищити дисципліну роботи персоналу, забезпечити прозорість процесів, скоротити час реакції та час вирішення заявок, а також отримувати достовірні аналітичні дані для прийняття управлінських рішень.

Наукова новизна роботи полягає у поєднанні оперативного моніторингу сервісних заявок із вбудованими механізмами аналітики KPI та застосуванні

моделей ITSM та STRIDE у контексті внутрішнього сервісу підприємства. Запропоновано підхід із замкненим циклом покращення сервісу: «дані → аналіз → рішення → підвищення якості».

Перспективи розвитку системи включають реалізацію модуля бази знань, інтеграцію з CRM/ERP, використання алгоритмів машинного навчання для прогнозування навантаження та автоматичної класифікації заявок, розробку мобільного застосунку для виконавців, розширення REST API та інтеграцію з корпоративними комунікаційними платформами.

Підсумовуючи, розроблений прототип автоматизованої системи моніторингу сервісних заявок підтвердив свою ефективність і може бути рекомендований до впровадження на підприємствах, що прагнуть підвищити якість сервісного обслуговування та оптимізувати роботу служби підтримки. Отримані результати мають як практичне, так і наукове значення, а підходи та моделі, застосовані під час розроблення, можуть бути використані для подальших досліджень у галузі сервісної аналітики та інформаційних систем управління.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Нормативні документи та стандарти

1. ISO/IEC 20000-1:2018. Information Technology — Service Management — Part 1: Service Management System Requirements. Geneva: ISO, 2018. URL: *https://www.iso.org/standard/70636.html*

2. ISO/IEC 27001:2022. Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems. Geneva: ISO, 2022. URL: *https://www.iso.org/standard/82875.html*

3. ISO 9001:2015. Quality Management Systems — Requirements. Geneva: ISO, 2015. URL: https://www.iso.org/iso-9001-quality-management.html

4. ДСТУ EN ISO 9241-11:2019. Ергономіка взаємодії людини та системи. Частина 11. URL: https://uas.org.ua/standard/dstu-en-iso-9241-11-2019/

5. ДСанПіН 3.3.2-007-98. Гігієнічні вимоги до персональних електронно-обчислювальних машин. URL: https://zakononline.com.ua/documents/show/44526___44526

6. Закон України «Про інформацію», №2657-XII від 02.10.1992. URL: *https://zakon.rada.gov.ua/laws/show/2657-12*

7. Закон України «Про захист інформації в інформаційно-телекомунікаційних системах», №80/94-ВР. URL: *https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80*

Методології, підходи, ITSM / ITIL

8. Axelos. ITIL 4 Foundation Edition. London: TSO, 2019. URL: *https://www.axelos.com/best-practice-solutions/itil*

9. Bohn R. Measuring and Managing Performance in Service Operations. MIT Press, 2017.

10. Hiles A. The Service Level Handbook: The Service Management Guide. Rothstein Publishing, 2016. URL: *https://www.rothstein.com/*

11. Galup S., Dattero R. IT Service Management: An Emerging Area in IT Education. Communications of the ACM, 2010. URL: *https://cacm.acm.org/*

12. Rubtsov V. Service Level Management in ITSM Frameworks. IEEE, 2020. URL: https://ieeexplore.ieee.org/

Книги з програмної інженерії та архітектури

13. Sommerville I. Software Engineering. 10th ed. Pearson, 2016.

14. Pressman R. Software Engineering: A Practitioner's Approach. McGraw-Hill, 2019.

15. Laudon K.C., Laudon J.P. Management Information Systems: Managing the Digital Firm. Pearson, 2020.

16. Turban E. et al. Information Technology for Management. Wiley, 2021. URL: https://www.wiley.com/en-us/Information+Technology+for+Management

17. Yourdon E. Modern Structured Analysis. Prentice Hall, 2007.

18. Bass L., Clements P., Kazman R. Software Architecture in Practice. Addison-Wesley, 2012.

19. Meyer B. Object-Oriented Software Construction. Prentice Hall, 2000.

20. BPMN 2.0 Handbook. Future Strategies Inc., 2020. URL: https://www.bpmnhandbook.com/

Бази даних та теорія даних

21. Date C. J. An Introduction to Database Systems. Pearson, 2019.

22. Elmasri R., Navathe S. Fundamentals of Database Systems. Pearson, 2016.

23. SQLite Documentation. SQLite Consortium, 2024. URL: *https://www.sqlite.org/docs.html*

24. PostgreSQL Documentation. PostgreSQL Global Development Group, 2024. URL: *https://www.postgresql.org/docs/*

Технології Python / Flask / SQLAlchemy

25. Python Documentation. Python Software Foundation, 2024. URL: *https://docs.python.org/*

26. Flask Documentation. Pallets Project, 2024. URL: *https://flask.palletsprojects.com/*

27. SQLAlchemy Documentation, 2024. URL: *https://docs.sqlalchemy.org/*

Безпека, моделювання загроз, OWASP

28. Kohnfelder L., Garg P. Threat Modeling. Microsoft Press, 1999. URL: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-overview

29. OWASP Foundation. OWASP Top 10 – 2021. URL: *https://owasp.org/www-project-top-ten/*

30. Shostack A. Threat Modeling: Designing for Security. Wiley, 2014.

31. NIST SP 800-53 Rev. 5. Security and Privacy Controls. URL: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final

32. CIA Triad — NIST Glossary. URL: https://csrc.nist.gov/glossary/term/confidentiality-integrity-availability

33. RBAC — Microsoft Documentation. URL: *https://learn.microsoft.com/en-us/azure/role-based-access-control/overview*

34. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. University of California, 2000. URL: *https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf*

Сервіс-деск системи та аналітика

35. Zendesk Support Documentation, 2024. URL: *https://support.zendesk.com/hc/en-us*

36. Atlassian. Jira Service Management Documentation, 2024. URL: *https://www.atlassian.com/software/jira/service-management*

37. Freshworks Inc. Freshdesk Documentation, 2024. URL: *https://freshdesk.com/*

38. OTRS Admin Manual, 2024. URL: https://doc.otrs.com/

39. GLPI Documentation, 2024. URL: *https://glpi-project.org/*

40. IBM. AI in Customer Service. URL: *https://www.ibm.com/topics/ai-in-customer-service*

41. Jade Global. *Benefits of using Bootstrap in UI/UX design – https://www.jadeglobal.com*

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ(ПРЕЗЕНТАЦІЯ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Автоматизована система моніторингу сервісних заявок

(сервісна аналітика)

Виконав: ст. гр. САДМ-61
Панасенко А.А.

Науковий керівник:
Нафєєв Р.К.

Київ - 2026

Актуальність теми

Проблема:

- Хаотичний облік заявок (месенджери, Excel).
- Втрата інформації про клієнтів.
- Порушення строків виконання (SLA).

Рішення:

- Впровадження ITSM-системи.
- Автоматизація Service Desk.
- Аналітика ефективності.

Мета та завдання

Мета роботи:

Розробка системи моніторингу заявок із функціями сервісної аналітики.

Завдання:

1. Дослідити процеси ITSM.
2. Спроєктувати архітектуру та БД.
3. Реалізувати веб-застосунок (Python/Flask).
4. Провести тестування та оцінити KPI.

Об'єкт та предмет

Об’єкт дослідження:

Процеси обслуговування сервісних звернень.

Предмет дослідження:

Методи та засоби моніторингу цих процесів.

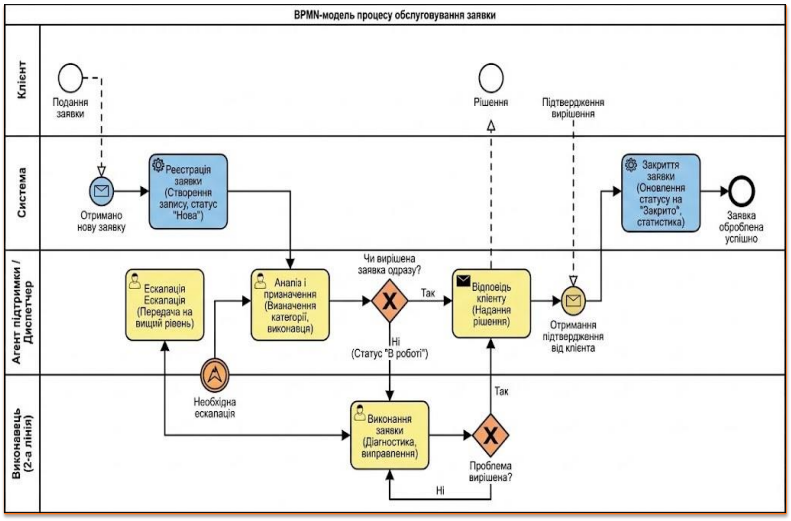
Наукова новизна:

Поєднання операційного трекінгу заявок з бізнес-аналітикою для прийняття управлінських рішень.

Бізнес-процес (BPMN)

Схема процесу обробки заявки:

1. Реєстрація клієнтом.
2. Класифікація та призначення.
3. Виконання робіт.
4. Контроль якості (SLA).
5. Закриття заявки.



Архітектура та Технології

Frontend:

- HTML5, CSS3, Bootstrap 5.

Backend:

- Python 3.
- Flask (REST API).

Database:

- SQLite / PostgreSQL.
- SQLAlchemy ORM.

База даних (ER-діаграма)

Основні сутності:

- Users (Користувачі, Ролі).
- Tickets (Заявки).
- Comments (Коментарі).

Довідники:

- Status (Статуси).
- Priority (Пріоритети).

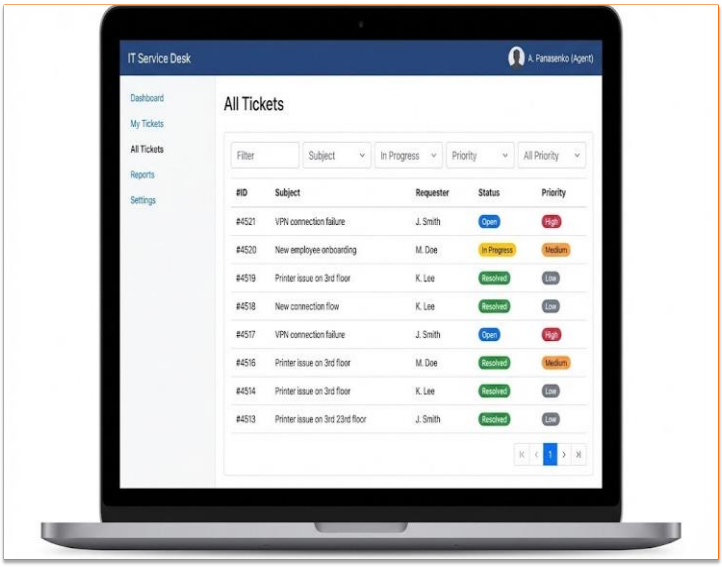
Інтерфейс системи

Ключові форми:

- Панель входу.
- Створення заявки.
- Список заявок (Agent View).

Особливості:

- Адаптивний дизайн.
- Зручні фільтри.

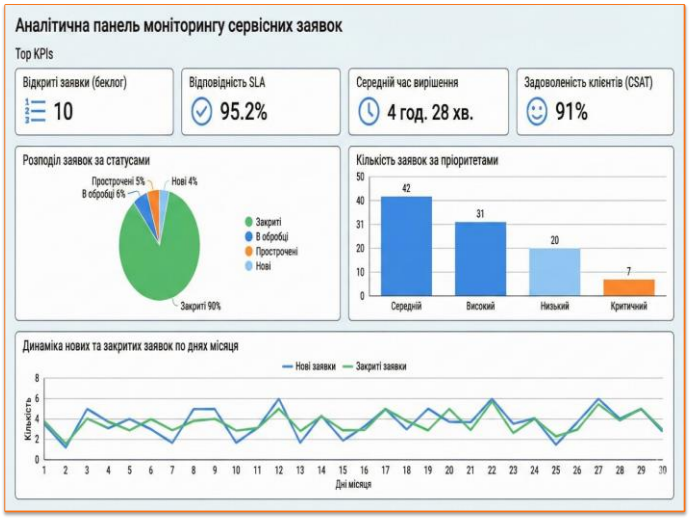


Сервісна аналітика

Дашборд керівника:

KPI (Показники):

- Backlog (Загальна черга).
- SLA % (Дотримання строків).
- MTTR (Середній час вирішення).
- Розподіл по агентах.



Тестування

Функціональне тестування:

- Всі сценарії працюють коректно.

Навантажувальне тестування:

- 50-100 активних користувачів.
- Час відгуку < 200мс.
- Система працює стабільно.

Ефективність та Безпека

Результати впровадження:

- Час реакції скоротився в 4 рази.
- Порушення SLA впали до 5%.

Безпека:

- Role-Based Access Control (RBAC).
- Захист паролів (Hashing).
- Захист від XSS та CSRF.

Висновки

1. Розроблено прототип системи.
2. Досягнуто прозорості процесів підтримки.
3. Реалізовано інструменти для прийняття рішень.

Перспективи:

- Інтеграція з Telegram-ботом.
- ML для прогнозування навантаження.

Дякую за увагу!

Доповідь завершено