ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система прийому та аналізу заявок технічної підтримки (Python/Django + Bootstrap + PostgreSQL)»

на здобуття освітнього ступеня магістра
зі спеціальності F3 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

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

_____ Андрій БУГАЙЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Андрій БУГАЙЧЕНКО
група КНДМ-62

Керівник: Олена ШИКУЛА
науковий ступінь, вчене звання д.ф.-м.н., професор

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

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність F3 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бугайченку Андрію Дмитровичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Інформаційна система прийому та аналізу заявок технічної підтримки (Python/Django + Bootstrap + PostgreSQL)

керівник кваліфікаційної роботи Олена ШИКУЛА д.ф.-м.н., професор,
(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 до кваліфікаційної роботи: науково-технічна література з веб-розробки та проектування баз даних; мова програмування Python; фреймворк Django; система управління базами даних PostgreSQL; середовище розробки Visual Studio Code..

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

Огляд існуючих систем управління заявками та аналіз їх функціоналу, виявлення переваг та недоліків.
Проектування архітектури системи, бази даних та реалізація ролей користувачів.
Розробка інформаційної системи для прийому, обліку та аналізу заявок технічної підтримки з широким функціоналом та зручним інтерфейсом.

5. Перелік графічного матеріалу:
   1. Опис програмного продукту
   2. Діаграма взаємодії для сценарію створення заявки.
   3. Візуалізація інтерфейсу.
   4. Фрагменти програмного коду.
6. Дата видачі завдання «15» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-27.09.25	Виконано
2	Огляд та порівняльний аналіз існуючих ITSM-рішень (Zendesk, Jira, Freshdesk)	28.09-11.10.25	Виконано
3	Формулювання функціональних та нефункціональних вимог до власної системи	12.10-25.10.25	Виконано
4	Вибір інструментів для розробки клієнтської частини сайту	26.10-04.11.25	Виконано
5	Вибір інструментів для розробки серверної частини сайту	05.11-12.11.25	Виконано
6	Розробка сайту інформаційної системи прийому та аналізу заявок технічної підтримки	12.11-02.12.25	Виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-08.12.25	Виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.25	Виконано

Здобувач вищої освіти _____ Андрій БУГАЙЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Олена ШИКУЛА
(підпис) ***(Ім'я, ПРІЗВИЩЕ)***

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 56 стор., 2 табл., 31 рис., 41 джерела.

Наукове завдання – огляд існуючих систем управління заявками (Help Desk) та аналіз їх функціоналу, виявлення переваг та недоліків; розробка інформаційної системи для прийому, обліку та аналізу заявок технічної підтримки зі зручним інтерфейсом.

Метою роботи є розробка веб-орієнтованої інформаційної системи для автоматизації процесів прийому, обробки та аналізу заявок технічної підтримки з використанням технологій Python/Django, Bootstrap та PostgreSQL.

Об'єкт дослідження – процес організації роботи служби технічної підтримки в невеликих та середніх компаніях.

Предмет дослідження – інформаційна система прийому та аналізу заявок технічної підтримки.

Короткий зміст роботи:

У дипломній роботі розроблено систему управління заявками для малого та середнього бізнесу. Перший розділ включає огляд ринку (Zendesk, Jira, Freshdesk) та аналіз потреб цільової аудиторії, на основі чого сформовано вимоги та обрано технологічний стек: Python/Django (бекенд), PostgreSQL (БД), Bootstrap 5 (інтерфейс) та Celery/Redis (асинхронні задачі). Результатом є готова веб-система, що автоматизує роботу служби підтримки, забезпечує контроль, комунікацію та аналітику.

Ключові слова: система управління заявками, веб-додаток, Python, Django, PostgreSQL, Bootstrap, адаптивний інтерфейс.

ABSTRACT

Text part of the master's qualification work: 56 pages, 2 tables, 31 pictures, 41 sources.

The scientific task is a review of existing help desk systems and analysis of their functionality, identifying their advantages and disadvantages; development of an information system for receiving, processing and analyzing technical support requests with a user-friendly interface; creation of an adaptive web version of the system for working on different devices.

The purpose of the work is to develop a web-oriented information system for automating the processes of receiving, processing and analyzing technical support requests using Python/Django, Bootstrap and PostgreSQL technologies.

The object of research is the process of organizing the work of a technical support service in small and medium-sized companies.

The subject of research is an information system for receiving and analyzing technical support requests.

Summary of the work:

This thesis develops a ticket management system for small and medium-sized businesses. It analyzes existing solutions (Zendesk, Jira, Freshdesk) and target audience needs to define requirements and select the technology stack: Python/Django for backend, PostgreSQL for database, Bootstrap 5 for the responsive interface, and Celery/Redis for asynchronous tasks. The result is a ready-to-deploy web system that automates support operations, ensuring control, transparent communication, and basic analytics.

Keywords: ticket management system, web application, Python, Django, PostgreSQL, Bootstrap, responsive interface.

ЗМІСТ

ВСТУП 15
1 ОГЛЯД ТА АНАЛІЗ ТЕМАТИЧНОЇ ОБЛАСТІ 18
1.1 Сучасний стан автоматизації служб технічної підтримки 18
1.1.1 Аналіз ринку систем управління заявками 18
1.1.2 Огляд потреб малого бізнесу в автоматизації техпідтримки 20
1.2 Аналіз процесу обробки заявок та визначення статусів користувачів 21
1.2.1 Бізнес-процеси обробки заявок в службі технічної підтримки 21
1.2.2 Визначення ролей користувачів та їх прав доступу 23
1.3 Огляд та аналіз існуючих рішень 24
1.3.1 Огляд популярних систем 24
1.3.2 Порівняльний аналіз функціоналу, переваг і недоліків існуючих рішень 25
1.4 Визначення вимог до власного рішення 27
2 АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ ДОДАТКУ 31
2.1 Аналіз вимог до системи 31
2.1.1 Функціональні вимоги та їх обґрунтування 32
2.1.2 Нефункціональні вимоги (продуктивність, безпека, масштабованість) 33
2.1.3 Аналіз бізнес-процесів та користувацьких сценаріїв 34
2.2 Аналіз та обґрунтування вибору технологій 36
2.2.1 Порівняльний аналіз мов програмування для веб-бекенду 36
2.2.2 Обґрунтування вибору Django як основного фреймворку 37
2.2.3 Аналіз вибору PostgreSQL 38
2.2.4 Обґрунтування вибору Bootstrap 5 та нативного JavaScript 38
2.3 Методологія розробки 39
2.3.1 Гібридна методологія: Agile з елементами Waterfall 39
2.3.2 План розробки та етапність 40
2.3.3 Комплексна стратегія управління якістю та ризиками 41
2.4 Архітектурні рішення 43
2.4.1 Вибір архітектурного шаблону MVT у Django 43

2.4.2 Проектування компонентів системи 43
2.4.3 Проектування схеми бази даних 45
3 ОПИС ПРОГРАМНОГО ПРОДУКТУ 46
3.1 Архітектура системи 46
3.1.1 Загальна архітектура системи 46
3.1.2 Компоненти бекенду та їх призначення 49
3.1.3 Структура проекту 55
3.1.4 Взаємодія компонентів 57
3.2 Реалізація ключових функціональностей 58
3.2.1 Модуль управління заявками 58
3.2.2 Механізми аутентифікації та авторизації 60
3.2.3 Система сповіщень та нотифікації 62
3.3 Інтерфейс користувача та адаптивний дизайн 64
3.3.1 Компонентна система на основі Bootstrap 5 64
3.3.2 Адаптивні таблиці та форми 67
3.4 Забезпечення якості та безпеки 69
3.4.1 Система тестування та валідації 69
3.4.2 Захист від веб-загроз 71
ВИСНОВКИ 74
ПЕРЕЛІК ПОСИЛАНЬ 76
ДОДАТОК А. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ 99
ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 75

ВСТУП

Актуальність теми. У сучасному глобалізованому бізнес-середовищі, де клієнтський досвід став основним фактором успіху, ефективна організація роботи служб технічного обслуговування перетворюється з допоміжної функції на ключовий конкурентний переваг для більшості компаній. Це особливо актуально для малого та середнього бізнесу (СМБ), який функціонує в умовах обмежених ресурсів, високої конкуренції та підвищених вимог до оперативності. Якісна, своєчасна підтримка не лише вирішує поточні проблеми клієнтів, але й формує довгострокову лояльність та позитивну репутацію бренду.

Сучасна цифрова епоха вимагає впровадження не просто автоматизованих, а інтелектуальних та доступних рішень, здатних структурувати потік запитів, забезпечувати прозорість процесів та надавати керівництву інструменти для аналітики ефективності. Проте перехід від хаотичної комунікації через електронну пошту та месенджери до професійної системи управління заявками (Help Desk) для багатьох невеликих компаній уявляється складною та ресурсомісткою задачею.

Стан проблеми. Аналіз ринку програмних продуктів виявляє очевидну диспропорцію: існує велика кількість потужних, але дорогих та складних корпоративних платформ (на кшталт Zendesk, ServiceNow, Jira Service Desk), призначених для великих організацій з ІТ-відділами. З іншого боку, є безліч простих онлайн-чатів або шаблонних рішень, яких недостатньо для ведення повноцінного життєвого циклу заявки. Це створює значну прогалину для спеціалізованого, але легкого у використанні та економічно доступного рішення, орієнтованого саме на потреби малих і середніх команд. Такі системи часто потребують індивідуальної розробки, що робить їх недосяжними для багатьох підприємств.

Проведений попередній аналіз дозволяє констатувати, що сучасний ринок гостро потребує рішення, яке поєднує в собі три ключові якості: зручний та інтуїтивний інтерфейс для всіх типів користувачів, достатній базовий функціонал для

повного циклу обробки запиту та модульну архітектуру, що дозволяє адаптуватися та розширюватися разом із бізнесом. Актуальність даної розробки полягає саме в заповненні цієї ніші шляхом створення open-source системи, вільної від надмірної складності та високих ліцензійних витрат.

Метою даної дипломної роботи є розробка повноцінної веб-орієнтованої інформаційної системи для автоматизації процесів прийому, обробки, контролю та аналізу заявок технічної підтримки. Система повинна реалізовувати сучасний стек технологій: Python/Django для реалізації бізнес-логіки та безпеки, Bootstrap 5 для створення адаптивного та сучасного інтерфейсу, PostgreSQL для надійного зберігання даних, а також Celery+Redis для забезпечення асинхронного виконання завдань, таких як відправка сповіщень та контроль дедлайнів.

Для досягнення поставленої мети необхідно вирішити наступні ключові завдання:

1. Провести комплексний аналіз існуючих систем управління заявками, вивчити типові бізнес-процеси служб підтримки в умовах СМБ та сформулювати вимоги до власного рішення.
2. Спроектувати архітектуру системи, обґрунтувати вибір технологічного стеку, спроектувати модульну структуру та схему бази даних.
3. Реалізувати ключові функціональні модулі системи: управління заявками (tickets) з повним життєвим циклом, систему користувачів та ролей (users), механізм сповіщень (notifications) та модуль звітності (reports).
4. Забезпечити високий рівень безпеки, адаптивності інтерфейсу, написати модульні тести та протестувати працездатність системи на реалістичних сценаріях.

Об'єкт дослідження – процес організації та автоматизації роботи служби технічної (клієнтської) підтримки в невеликих та середніх компаніях.

Предмет дослідження – методологія, архітектура та програмна реалізація веб-орієнтованої інформаційної системи для прийому, обробки, моніторингу та аналізу заявок технічної підтримки.

Практична цінність розробки полягає в створенні готового до впровадження програмного продукту з відкритим вихідним кодом. Система надає невеликим командам інструмент для переходу від хаотичної комунікації до структурованого процесу, забезпечуючи: централізоване зберігання всіх звернень, чіткий контроль статусів та відповідальних, автоматичну маршрутизацію, прозору комунікацію (включно з внутрішніми нотатками), базову аналітику продуктивності та дотримання SLA. Це дозволяє суттєво підвищити ефективність роботи підтримки, зменшити час вирішення проблем та, як наслідок, підвищити задоволеність клієнтів без значних фінансових витрат.

1 ОГЛЯД ТА АНАЛІЗ ТЕМАТИЧНОЇ ОБЛАСТІ

1.1 Сучасний стан автоматизації служб технічної підтримки

1.1.1 Аналіз ринку систем управління заявками

Сучасний ринок програмних рішень для управління заявками техпідтримки сформувався навколо двох ключових підходів до управління бізнесом: CRM-систем (Customer Relationship Management), орієнтованих на клієнтоцентричність та оптимізацію взаємодій, та ERP-систем (Enterprise Resource Planning), що забезпечують комплексне управління внутрішніми ресурсами компанії [1, 2]. Для служб технічної підтримки найбільш релевантними є CRM-системи або їх спеціалізовані похідні – ITSM/Service Desk рішення. Це пояснюється тим, що їх функціонал історично розроблений для автоматизації всього життєвого циклу клієнта, включно з обробкою запитів, веденням історії взаємодій та аналітикою якості сервісу [2, 18, 19, 20].

Важливою еволюційною віхою стала трансформація CRM від простих баз даних контактів до універсальних бізнес-платформ. Сучасні рішення, такі як Salesforce або HubSpot, інтегрують модулі маркетингу, продажів, обслуговування та аналітики в єдиному середовищі. Це дозволяє створювати цілісний «профіль клієнта», де кожна заявка в техпідтримки не є ізольованим випадком, а частиною загальної історії взаємин, що значно підвищує ефективність та персоналізацію сервісу.

Домінуючою тенденцією останнього десятиліття став масовий перехід від локального (on-premise) програмного забезпечення до хмарних SaaS-моделей (Software as a Service) [1]. Ця модель надає бізнесу, особливо малому та середньому, ряд критичних переваг: усунення великих капітальних витрат на обладнання та ліцензії, швидкість впровадження (від годин до днів), автоматичні оновлення без зупинки роботи, а також доступ з будь-якого пристрою через веб-браузер. Ключові

бізнес-вигоди SaaS-рішень включають централізацію всієї клієнтської інформації, автоматизацію рутинних, повторюваних завдань (наприклад, маршрутизація заявок за типом або відповідачами) та отримання глибоких, візуально зручних аналітичних звітів для прийняття управлінських рішень на основі даних (data-driven decisions) [3].

Доказ ефективності: Міжнародні дослідження (наприклад, від Forrester Research) показують, що компанії, які впроваджують сучасні системи управління заявками, реалізують середнє зростання ефективності роботи служби підтримки на 25-40%. Цей ефект досягається за рахунок комплексної оптимізації процесів: скорочення часу першої відповіді (First Response Time), зменшення середнього часу вирішення (Mean Time to Resolution), усунення дублювання зусиль та підвищення загальної якості обслуговування клієнтів завдяки прозорості та контролю.

1.1.2 Огляд потреб малого бізнесу в автоматизації техпідтримки

На відміну від великих корпорацій з виділеними IT-відділами та значним бюджетом, малий бізнес (10-100 співробітників) стикається з унікальним набором обмежень, що формують особливо вибагливі та специфічні вимоги до систем автоматизації [4]. Насамперед, це жорстка обмеженість фінансових ресурсів та практична відсутність виділених IT-фахівців, здатних підтримувати та адаптувати складні корпоративні рішення. Існуючі платформи, орієнтовані на великий бізнес (Enterprise), часто виявляються непропорційно дорогими, надто складними для сприйняття немайжевими користувачами або недостатньо гнучкими для швидкої адаптації під мікропроцеси невеликої, але динамічної компанії [5].

Статистика свідчить про значну прогалину: Понад 60% малих підприємств не використовують спеціалізовані системи управління заявками, вдаючись до електронних таблиць (Excel/Google Sheets), електронної пошти або загальних месенджерів [5]. Цей підхід призводить до системних проблем: втрати важливих клієнтських звернень у потоках листування, неможливості ефективного розподілу

навантаження між співробітниками, відсутності аналітики для аналізу якості роботи та, як наслідок, до втрати клієнтів, зниження лояльності та неефективного використання ключових людських ресурсів компанії.

Проведений аналіз виокремлює для малого бізнесу критичні характеристики системи, які перевершують за важливістю наявність «розширеного» функціоналу [19]:

Максимальна простота використання (Usability): Інтуїтивний, не перевантажений інтерфейс, який не вимагає тижнів навчання. Сотрудники малого бізнесу – це універсали, які часто поєднують функції, тому система має бути зрозумілою з першого погляду.

- Швидкість та безболісність впровадження (Time-to-Value) [21]:
- Можливість почати реальну роботу в системі в стислі терміни (дні/тижні), без тривалих циклів налаштування, консалтингу та інтеграції.
- Органічна масштабованість (Organic Scalability): Здатність системи рости разом з бізнесом, дозволяючи поступово підключати нових користувачів, додавати категорії або розширювати автоматизацію без радикальної зміни архітектури.
- Прозора та передбачувана вартість (Predictable Cost): Модель підписки (щомісячна або річна) з чітким тарифом, яка не обтяжує бюджет і не містить прихованих платежів за додаткові модулі або підтримку [1, 6].

Ключовим стратегічним підходом для малого бізнесу є принцип поступового впровадження (phased rollout) – почати з базового, але повноцінного функціоналу (наприклад, реєстрація заявок, призначення, комунікація), а потім розширювати систему, додаючи модулі звітності, інтеграції з іншими сервісами чи базу знань. Це дозволяє мінімізувати стартові витрати, операційні ризики та поступово адаптувати команду до нових процесів.

1.2 Аналіз процесу обробки заявок та визначення статусів користувачів

1.2.1 Бізнес-процеси обробки заявок в службі технічної підтримки

Типовий життєвий цикл заявки (ticket lifecycle) в автоматизованій службі техпідтримки є керованим, багатоетапним workflow, де кожна стадія має чітко визначені вхідні та вихідні дані, відповідальних осіб та критерії завершення [5]. Аналіз цих процесів є основою для проектування логіки будь-якої системи.

1. **Реєстрація, класифікація та пріоритезація:** Заявка може надходити через різні канали комунікації (email, веб-форма, чат). Система повинна автоматично захопити та обробити вхідні дані, присвоїти унікальний ідентифікатор, визначити джерело, віднести до певної категорії (наприклад, «Програмна помилка», «Питання по рахунку») та призначити пріоритет на основі заданих правил (наприклад, ключових слів або важливості клієнта). Цей етап критичний для подальшої ефективної обробки.
2. **Назначення, аналіз та виконання:** Після класифікації система або адміністратор призначає заявку конкретному фахівцю (менеджеру). Ключовим тут є механізм розподілу навантаження (load balancing), що враховує зайнятість, фаховість та поточну кількість активних завдань у кожного оператора. Фахівець отримує сповіщення, аналізує проблему, консультується з колегами або клієнтом (через вбудовану систему коментарів), веде пошук рішення. На цьому етапі система має забезпечувати контроль за термінами (SLA – Service Level Agreement) та прозоре відстеження прогресу.
3. **Вирішення, перевірка та закриття:** Після знаходження рішення фахівець детально фіксує виконані дії, інформує клієнта про результат та змінює статус заявки на «вирішено» (resolved). Ідеальним є механізм підтвердження клієнтом – заявка закривається лише після його згоди, що запобігає передчасному закриттю. На цьому етапі також може збиратися зворотний зв'язок (CSAT – Customer Satisfaction Score) для оцінки якості роботи.

4. **Аналітика, звітність та вдосконалення процесів:** Вся мета-інформація про заявку (час створення, час відповіді, час вирішення, кількість перепризначень тощо) зберігається в системі [3]. На її основі формуються звіти та дашборди, що дозволяють аналізувати ключові показники ефективності (KPI) служби, виявляти «вузькі місця» в процесах (наприклад, категорії з найдовшим часом вирішення), оцінювати продуктивність окремих фахівців та команди в цілому, і, відповідно, впроваджувати покращення.

Важливим аспектом сучасної системи є повна прозорість та контрольованість кожного етапу для керівництва. Це забезпечує не лише відстеження ефективності, але й дозволяє будувати прогнози навантаження, обґрунтовувати потреби в персоналі та стратегічно розвивати сервіс.

1.2.2 Визначення ролей користувачів та їх прав доступу

Ефективна та безпечна організація роботи в системі ґрунтується на чіткій моделі ролей та прав доступу (Role-Based Access Control – RBAC) [2], яка гарантує, що кожен користувач має доступ лише до необхідних для своїх обов'язків даних і функцій, мінімізуючи ризики помилок та несанкціонованих дій. Найбільш обмеженою є роль клієнта (End-User), який має доступ виключно до власних заявок. Його інтерфейс зосереджений на простоті самообслуговування: можливості створювати нові звернення, переглядати їх статус та історію листування, а також отримувати сповіщення. Ключовими критеріями для цієї ролі є зручність та оперативність отримання інформації без необхідності заглиблюватися у внутрішні процеси системи. Основним виконавцем та ключовим користувачем виступає оператор або фахівець (Agent/Technician). Ця роль має право переглядати, фільтрувати та сортувати призначені заявки, змінювати їх статус у межах бізнес-правил, вести комунікацію з клієнтами через публічні коментарі та з колегами через внутрішні, а також додавати файли та встановлювати терміни вирішення. Для

оператора критично важливим є швидкий доступ до повного контексту та історії взаємодій, що дозволяє надавати якісний та персоналізований сервіс. Повним контролем над системою володіє адміністратор (Administrator). Ця роль включає управління обліковими записами всіх користувачів, налаштування категорій заявок, статусів, пріоритетів та SLA-правил. Адміністратор має доступ до всіх заявок з можливістю їх перепризначення, ескалації та редагування. Його ключовою функцією є формування аналітичних звітів для оцінки ефективності служби, окремих операторів та аналізу тенденцій, а також відповідальність за технічну підтримку та розвиток системи. У більших командах може бути виділена додаткова роль супервізора або менеджера (Supervisor/Manager). Ця позиція зазвичай передбачає права на перегляд заявок усіх підлеглих операторів, аналітику на рівні команди, моніторинг черг та дотримання SLA, але без глибокого технічного адміністрування самої системи. Такий підхід дозволяє ефективно керувати операційною діяльністю, розподіляючи обов'язки та не перевантажуючи адміністратора. Подібна детальна диференціація прав не лише підвищує рівень безпеки, але й сприяє ефективній організації робочих процесів, покращує відповідальність кожного учасника та дозволяє будувати гнучку та масштабовану структуру управління.

1.3 Огляд та аналіз існуючих рішень

1.3.1 Огляд популярних систем

Аналіз лідерів ринку дозволяє зрозуміти стандарти функціоналу та напрямки розвитку індустрії, а також чітко визначити, які потреби малого бізнесу вони покривають недостатньо [18, 22, 23, 24].

- **Zendesk:** Один з піонерів і найпопулярніших «все-в-одному» (all-in-one) рішень для обслуговування клієнтів. Його сила – в комплексності та екосистемі. Zendesk Suite інтегрує керування заявками з багатьох каналів

(email, соцмережі, live chat, голосові дзвінки), потужну базу знань (Knowledge Base), інструменти для співпраці та аналітики реального часу. Його особливо цінують за глибину аналітики та звітності (Explore), що робить його привабливим для великих компаній, які базують рішення на даних.

- **Jira Service Desk:** Продукт компанії Atlassian, що походить зі світу розробки ПЗ. Його ключова перевага – натуральна та глибока інтеграція з Jira Software та Confluence. Це робить його ідеальним для IT-команд та компаній, де служба підтримки тісно взаємодіє з розробниками: заявка від клієнта може бути легко перетворена на технічне завдання (task) або баг-репорт у Jira. Система відзначається неймовірною гнучкістю налаштувань робочих процесів (workflow), автоматизації та прав доступу.
- **Freshdesk:** Прямий конкурент Zendesk, що зробив ставку на доступність та зручність. Freshdesk пропонує сучасний, «приємний» інтерфейс, функції автоматизації на основі правил (роботи), вбудованого AI-помічника для пропозиції відповідей (Freddy AI) та інтеграції з численними популярними SaaS-сервісами. Його особливість – орієнтація на малий та середній бізнес з розумною, структурованою ціновою політикою, що включає безкоштовний тариф для мінімальних потреб.

1.3.2 Порівняльний аналіз функціоналу, переваг і недоліків існуючих рішень

Для наочності та глибшого розуміння розглянемо системи через призму вимог малого бізнесу (див.табл. 1.1)

Таблиця 1.1 – Вимоги малого бізнесу

Критерії	Zendesk	Jira Service Desk	Freshdesk	Оптимально для малого бізнесу
Цільова аудиторія	Великий або середній бізнес, що цінує аналітику	IT-компанії, технічні команди, де важлива інтеграція з розробкою	Малий/середній бізнес, стартапи	Малий не-IT бізнес (роздріб, послуги, консалтинг)
Ключова перевага	Потужна аналітика,екосистема додатків, надійність	Глибока інтеграція з Atlassian Stack, кастомізація workflow	Зручний інтерфейс, швидкий старт, доступна ціна	Простота, низький поріг входу, передбачувана вартість
Основний недолік(для малого бізнесу)	Дуже висока вартість навіть для базових тарифів,складність налаштування	Надмірно складний для нетехнічних користувачів, висока “ціна входу” в екосистему	Менш потужна аналітика, обмежені можливості глибокої кастомізації	Надмірна функціональність, що ускладнює інтерфейс та підвищує вартість
Модель ціноутворення	Висока(від $49/агент/міс.), плата за додаткові функції (напр., Explore)	Складна модель ліцензування (користувач/агент), висока вартість володіння	Доступніша (є тариф від $0), більш прозора структура	Проста підписка з фіксованою або низькою вартістю за агент, без прихованих платежів

Загальні висновки та виявлення ринкової ніші:

Аналіз показує, що навіть найбільш «доступні» з перелічених рішень (Freshdesk) є універсальними коробковими продуктами, спроектованими для максимально широкої аудиторії. Їхня потужність є водночас їхнім слабким місцем для малого бізнесу, оскільки супроводжується:

1. Надмірною функціональністю, яка перевантажує інтерфейс, ускладнює навчання та відволікає від основних задач.
2. Складністю початкового налаштування, що часто вимагає залучення консультантів або виділеного часу IT-спеціаліста.
3. Непропорційною вартістю масштабування: Додавання кожного нового оператора, модуля або інтеграції суттєво збільшує щомісячні витрати.

Це створює чітку ринкову нішу для спеціалізованого, спрощеного рішення, яке реалізує не «все можливе», а саме необхідне для ефективної роботи техпідтримки малої компанії. Таке рішення має пропонувати оптимальне співвідношення «функціональність – простота – вартість», будучи зрозумілим, швидким у впровадженні та економічно вигідним.

1.4 Визначення вимог до власного рішення

На основі всебічного аналізу предметної області, потреб користувачів та конкурентного середовища сформовано детальні вимоги до системи, що розробляється.

Функціональні вимоги (що система повинна робити):

1. Повний цикл управління заявками:
    a. Створення: Веб-форма для клієнтів з обов'язковими полями (заголовок, опис, категорія). Автоматичне присвоєння номера, часу та автора.
    b. Перегляд та фільтрація: Інтуїтивний список заявок для операторів/адміністраторів з фільтрами за статусом, пріоритетом, категорією, відповідальним. Рольова фільтрація: клієнт бачить тільки свої заявки, оператор – призначені йому, адмін – всі.
    c. Детальний перегляд: Сторінка заявки з усією інформацією, історією статусів та коментарями.
    d. Зміна статусу: Контрольований workflow з обмеженнями зміни статусів для операторів (наприклад, не можуть закрити без підтвердження).

2. Система користувачів та безпеки:
    a. Три рівні ролей (Клієнт, Оператор, Адміністратор) з чітко визначеними правами на рівні URL, представлень (views) та шаблонів.
    b. Реєстрація та автентифікація через email/пароль. Кастомний бекенд аутентифікації, що дозволяє вхід за email.
    c. Валідація паролів (мінімальна довжина, наявність цифр).
    d. Профілі користувачів з базовою інформацією.
3. Модуль спілкування та сповіщень:
    a. Система коментарів до заявок з розрізненням на публічні (для клієнта) та внутрішні (тільки для операторів/адмінів).
    b. In-app сповіщення (нотифікації) про нові заявки, призначення, нові коментарі. Принцип без самосповіщень – користувач не отримує сповіщення про власну дію.
    c. API для отримання/позначення сповіщень для динамічного оновлення інтерфейсу.
4. Аналітика та звітність (для адміністратора):
    a. Дашборд з ключовими метриками: загальна кількість заявок за період, відкриті/вирішені, середній час вирішення.
    b. Експорт даних у форматі Excel для глибшого аналізу (наприклад, детальний список усіх заявок за місяць).

Вибір технологічного стеку для реалізації:

− Для реалізації серверної частини системи обрано мову програмування Python версії 3.10 та вище у поєднанні з веб-фреймворком Django версії 4.2. Даний вибір обґрунтований низкою технічних та практичних факторів. Python є мовою з інтуїтивно зрозумілим синтаксисом, низьким порогом входження та широкою спільнотою розробників, що сприяє підтримці та розвитку проєкту. Фреймворк Django, у свою чергу, надає комплексне рішення для швидкої та структурованої розробки. Він пропонує вбудовані та добре інтегровані

компоненти, такі як потужний ORM для абстрактної роботи з різними базами даних, готову адміністративну панель, систему аутентифікації та авторизації, механізм шаблонів, а також численні засоби забезпечення безпеки, включаючи захист від CSRF, XSS та SQL-ін'єкцій. Чітко визначена архітектура MVT (Model-View-Template) сприяє логічному розділенню відповідальностей у коді, що робить фреймворк ідеальним інструментом для створення безпечних, підтримуваних та масштабованих веб-застосунків у стислі терміни [1,11].

- Frontend: Bootstrap 5. Обґрунтування: Найпопулярніший CSS-фреймворк для створення адаптивних та сучасних інтерфейсів. Дозволяє швидко будувати сторінки з готових, доступних компонентів (сітки, форми, кнопки, навігація), забезпечуючи консистентність дизайну та значне прискорення верстки, що критично важливо для прототипування та MVP [14].
- База даних: PostgreSQL (основна) та SQLite (для розробки). Обґрунтування: PostgreSQL – потужна, відкрита, ACID-сумісна СУБД, що відмінно інтегрується з Django ORM. Пропонує надійність, розширені типи даних, повнотекстовий пошук та чудову продуктивність. SQLite використовується на етапі розробки через свою простоту (файлова база) та нульову конфігурацію [10, 12, 13].
- Додаткові технології: Celery + Redis для виконання асинхронних завдань (наприклад, відправка email-сповіщень, генерація звітів). Django Channels (опціонально) для реалізації real-time сповіщень через WebSocket.

2 АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ ДОДАТКУ

2.1 Аналіз вимог до системи

2.1.1 Функціональні вимоги та їх обґрунтування

На основі глибокого аналізу бізнес-процесів типової служби підтримки малої компанії, що включав моделювання workflow та інтерв'ю з потенційними користувачами [9], було сформовано чіткий набір вимог. Кожна з них є відповіддю на конкретну бізнес-проблему. Основним завданням ядра системи – управління заявками з контролем прав – було подолання хаосу в обробці запитів шляхом запровадження чіткого, відстежуваного процесу. Ключовим рішенням тут стала рольова фільтрація представлення списку заявок за допомогою методу TicketListView.get_queryset(). Ця логіка гарантує, що клієнт бачить лише власні заявки (Ticket.objects.filter(created_by=user)), оператор – призначені йому або всі відкриті, а адміністратор має доступ до повного списку (Ticket.objects.all()), що формує фундамент безпеки та організації роботи. Для запобігання «зависанню» заявок та автоматизації життєвого циклу були впроваджені спеціальні механізми. Наприклад, сигнал post_save для моделі Comment автоматично змінює статус заявки на «В роботі», якщо коментар додав оператор. Сервіс NotificationService автоматично генерує сповіщення для адміністраторів про нову заявку та для оператора про призначення, реалізуючи принцип проактивних сповіщень (proactive alerts). Система сповіщень була розроблена з орієнтацією на зручність користувача (UX). Щоб уникнути необхідності постійно оновлювати сторінку для перевірки нових повідомлень, було реалізовано легкий API (/notifications/api/unread/) для AJAX-запитів, який динамічно оновлює бейдж із кількістю непрочитаних сповіщень. Важливою бізнес-логікою стала відсутність самосповіщень – користувач не отримує сповіщення про власні дії, що забезпечується простою перевіркою if ticket.created_by

!= request.user у відповідному представленні. Звітність у системі орієнтована на практичне прийняття рішень, а не на формальне звітування. Для малого бізнесу було важливо отримувати швидкі та зрозумілі інсайти. Замість складних конструкторів звітів було реалізовано простий дашборд з графіками основних KPI (наприклад, кількість заявок, середній час закриття), який будується одним оптимізованим запитом з агрегацією даних. Додатково, функція експорту в Excel за допомогою бібліотеки openpyxl (ReportGenerator.export_to_excel()) надає адміністратору всі необхідні дані для аналізу без зайвої складності.

2.1.2 Нефункціональні вимоги (продуктивність, безпека, масштабованість)

В процесі розробки були сформовані ключові нефункціональні вимоги щодо продуктивності, безпеки та архітектури системи. Щодо продуктивності, було визначено, що час відгуку на ключові операції, такі як завантаження списку заявок чи відкриття детальної сторінки, не повинен перевищувати 1-2 секунди. Для досягнення цієї мети заплановано цілий комплекс заходів: оптимізація ORM-запитів за допомогою select_related() та prefetch_related() для усунення проблеми «N+1 query», стратегічне індексування в PostgreSQL полів, що фільтруються або сортируються, обов'язкова пагінація списків для запобігання завантаженню завеликих наборів даних, а також впровадження кешування для статичних даних чи результатів складних обчислень. У сфері безпеки система проектувалася з огляду на основні принципи OWASP Top 10. Це включає надійну аутентифікацію та авторизацію з використанням стійких хешерів паролів, захист сесійних даних та реалізацію моделі RBAC. Для захисту від веб-атак використовуються вбудовані механізми Django, такі як CSRF-токени у формах, автоматичне екранування для запобігання XSS та параметризовані запити ORM проти SQL-ін'єкцій. Валідація вхідних даних реалізована на рівні форм та моделей, включаючи перевірку

завантажуваних файлів. Для продакшен-середовища обов'язковими є використання HTTPS, налаштування заголовків безпеки (HSTS, CSP) та вимкнення режиму налагодження. Архітектурні рішення були спрямовані на забезпечення масштабованості та простоти підтримки. Система будується на принципі модульності, будучи розбитою на логічні Django-додатки, що дозволяє незалежно розвивати та тестувати окремі компоненти. Слабка зв'язаність між модулями забезпечується через чіткі інтерфейси, такі як сервісні класи, сигнали та REST API, що мінімізує ризики при внесенні змін. Для підготовки до горизонтального масштабування передбачено використання Celery для винесення фонових задач на окремі воркери, а також інтеграцію з зовнішніми сервісами для зберігання медіа-файлів.

2.1.3 Аналіз бізнес-процесів та користувацьких сценаріїв

- US1. Як Клієнт, я хочу створити заявку, щоб отримати допомогу з проблемою.
  - ✔ Критичний шлях: Авторизація → Клік «Створити заявку» → Заповнення форми (валідація на клієнті та сервері) → Натиск «Відправити».
  - ✔ Бізнес-логіка: TicketCreateView.form_valid() присвоює заявці поточного користувача, статус «Нова», зберігає. Сигнал post_save викликає NotificationService, що створює сповіщення для всіх активних адміністраторів.
  - ✔ Критерій успіху: Клієнт бачить повідомлення «Заявку №XXX створено» та перенаправляється на її сторінку.
- US2. Як Оператор, я хочу побачити список призначених мені заявок, щоб ефективно планувати робочий день.
  - ✔ Критичний шлях: Вхід у систему → Перехід на головну сторінку/розділ «Заявки».

- ✔ Бізнес-логіка: TicketListView.get_queryset() автоматично фільтрує заявки за умовою assigned_to=request.user. Додатково працюють фільтри за статусом/пріоритетом через GET-параметри.
- ✔ Критерій успіху: Оператор бачить актуальний, персоналізований список завдань без зайвої інформації.

− US3. Як Адміністратор, я хочу побачити звіт про ефективність за останній місяць, щоб оцінити роботу служби.

- ✔ Критичний шлях: Вхід → Перехід у розділ «Звіти» → Вибір періоду (за замовчуванням «останні 30 днів») → Клік «Показати»/«Експортувати».
- ✔ Бізнес-логіка: ReportGenerator виконує низку агрегуючих запитів до БД (Count, Avg, F-вирази для часу). Дані передаються у шаблон для візуалізації або у сервіс експорту в Excel.
- ✔ Критерій успіху: Адміністратор отримує зрозумілий дашборд або файл Excel з даними для аналізу.

2.2 Аналіз та обґрунтування вибору технологій

2.2.1 Порівняльний аналіз мов програмування для веб-бекенду

Вибір Python серед інших популярних мов був обґрунтований наступною порівняльною таблицею (див. табл. 2.1)

Таблиця 2.1 – Порівняльний аналіз мов програмування

Критерій	Python (Django/Flask)	Java (Spring Boot)	PHP (Laravel)	JavaScript (Node.js)
Швидкість розробки	Дуже висока (лаконічний синтаксис, батарейки в комплекті)	Низька (вербозність, конфігурація)	Висока	Висока(єдиний стек JS)
Продуктивність	Середня	Висока	Середня	Висока (V8)
Екосистема для бізнес-додатків	Відмінна (Django – лідер для «бізнес-логіки»)	Висока(Spring)	Висока(Laravel)	Хороша(розвивається)
Крива навчання	Низька (проста для початку, глибока для майстерності)	Висока	Низька/Середня	Середня (асинхронність)
Мета проекту	Швидке створення безпечного, структурованого MVP	Критичні high-load системи	Швидка веб-розробка	Real-time, high-I/O додатки

Висновок: Для задачі створення структурованого бізнес-додатку з чіткою моделлю даних та інтерфейсом адміністрування Python та Django є оптимальним компромісом між швидкістю розробки, безпекою та можливістю подальшого супроводу. Java надмірно важка для MVP малого бізнесу, PHP (Laravel) є гідною альтернативою, але екосистема Django більш «зріла» для задач корпоративного рівня, Node.js – менш зручний для CRUD-додатків зі складною бізнес-логікою.

2.2.2 Обґрунтування вибору Django як основного фреймворку

1. Концепція «Don't Repeat Yourself» (DRY): Django спонукає до повторного використання коду через механізми успадкування (класів, шаблонів), міксіни та середовищно-орієнтовану конфігурацію.
2. Потужна система міграцій бази даних: Міграції Django (makemigrations, migrate) дозволяють синхронізувати схему БД між середовищами розробки, тестування та продакшену, відстежувати зміни та відкатуватися. Це критично важливо для життєвого циклу ПЗ.
3. Вбудована підтримка інтернаціоналізації (i18n) та локалізації (l10n): Фреймворк з коробки надає засоби для перекладу інтерфейсу, що дозволило реалізувати україномовний інтерфейс з мінімальними зусиллями через файли .po.
4. Безпека на рівні фреймворку: Окрім захисту від атак, Django має вбудовану систему паролів, обмеження частоти запитів, захист від клікджекінгу (X-Frame-Options).
5. Можливість розширення через «додатки» (apps): Архітектура Django ідеально підходить для компонентного підходу, де кожен модуль (users, tickets) – це окремий додаток з своїми моделями, представленнями, шаблонами.

2.2.3 Аналіз вибору PostgreSQL

PostgreSQL була обрана перед усіма іншими СУБД за такі технічні переваги, критичні для системи управління:

- Повноцінна підтримка ACID: Гарантує цілісність даних навіть у разі збоїв.
- Складні типи даних та індексування: Підтримка масивів, JSON/JSONB (для гнучкого зберігання), повнотекстового пошуку, просторових даних (PostGIS). Індекси GIN/GiST для швидкого пошуку в цих структурах.

- Конкурентність та MVCC (Multiversion Concurrency Control): Ефективно обробляє багато одночасних операцій читання/запису без блокувань, що важливо для багатокористувацької системи.
- Надійність та репутація: PostgreSQL – стандарт де-факто для відповідальних веб-додатків, з активною спільнотою та регулярними оновленнями безпеки.

2.2.4 Обґрунтування вибору Bootstrap 5 та нативного JavaScript

Bootstrap 5 виступає не просто CSS-бібліотекою, а повноцінним каркасом для створення інтерфейсів. Він надає потужну адаптивну сітку (grid system), яка дозволяє легко будувати макети, що автоматично підлаштовуються під розмір екрану за допомогою набірних класів, таких як col-md-* чи row-cols-*. Каркас також включає велику бібліотеку готових, доступних компонентів – модальні вікна, навігаційні панелі (navbar), форми, кнопки та сповіщувачі (alerts), які вже містять необхідні ARIA-атрибути для покращення доступності. Для пришвидшення розробки Bootstrap 5 пропонує широкий набір утилітарних класів, що дозволяють швидко стилізувати елементи без написання власного CSS, наприклад, задавати відступи за допомогою mt-3 чи додавати тіні класом shadow-sm. Важливою особливістю нової версії є повна відмова від jQuery на користь чистого JavaScript, що спрощує інтеграцію з сучасними фронтенд-підходами. Що стосується інтерактивності, у проєкті було обґрунтовано обрано використання нативного JavaScript (ES6+) без застосування важких фреймворків, таких як React чи Vue.

Основним завданням фронтенду є постійна взаємодія з API та динамічне оновлення інтерфейсу, з чим відмінно справляються сучасні нативні технології. Зокрема, Fetch API використовується для здійснення запитів, Template literals – для динамічного формування HTML, а синтаксис async/await – для зручної роботи з асинхронним кодом. Цей набір інструментів повністю покриває потреби системи, забезпечуючи необхідну функціональність без додаткової складності, пов'язаної з підтримкою повноцінного фронтенд-фреймворку.

2.3 Методологія розробки

2.3.1 Гібридна методологія: Agile з елементами Waterfall

Обрана стратегія розробки ґрунтується на поєднанні сильних сторін планового та гнучкого підходів. Це дозволило ефективно поєднати чіткість технічного бачення з можливістю адаптації під час реалізації. На початковому етапі було застосовано принципи, подібні до Waterfall, з акцентом на детальне планування та проектування. Ця фаза включала всебічний аналіз вимог та створення повноцінної проектної документації: ER-діаграми бази даних, схеми взаємодії компонентів системи та інтерфейсних прототипів. Така підготовча робота заклала міцний фундамент, дозволивши виявити та усунути суперечності ще до початку кодування та сформувати чіткий технічний план. Основна фаза реалізації пройшла за ітераційною моделлю, натхненною Agile.

Робота була організована у вигляді послідовних спринтів, кожен з яких був зосереджений на створенні завершеного функціонального блоку. Перший спринт присвячувався побудові базових моделей даних, форм та адміністративного інтерфейсу. Наступний спринт включав розробку логіки представлень для обробки заявок і коментарів, а також системи ролей. Третій спринт був присвячений інтеграції повноцінної системи сповіщень. Фінальний етап охопив створення модуля аналітики, проведення комплексного тестування та підготовку до впровадження. Ключовим елементом після кожної ітерації була демонстрація працюючого функціоналу, що забезпечувало постійний зворотний зв'язок і дозволяло гнучко коригувати подальші кроки.

2.3.2 План розробки та етапність

Розробка системи пройшла через чітко визначені етапи. Спочатку було здійснено підготовку середовища, що включала налаштування віртуального середовища Python, встановлення залежностей з файлу requirements.txt, ініціалізацію

Git-репозиторію та налаштування IDE (VS Code з відповідними плагінами для Django, Python та PostgreSQL).

Наступним кроком стало проєктування ядра системи. На цьому етапі було створено діаграму класів (UML) для моделей, спроєктовано схему бази даних, розроблено wireframe макети всіх ключових сторінок у Figma, а також написано технічне завдання з детальними user stories та приймальними критеріями (acceptance criteria). Далі відбулася розробка бекенду, яка включала кілька підетапів. Спочатку було створено додатки Django (core, users, tickets), визначено основні моделі (CustomUser, Ticket, Comment) та написані міграції. Після цього реалізовано представлення на основі Class-Based Views, форми з валідацією та URL-конфігурацію, а також написані сигнали для автоматизації бізнес-логіки. Завершивши цей блок, було розроблено сервісні класи (NotificationService, ReportGenerator) та API endpoints для сповіщень у форматі JSON.

Етап розробки фронтенду та інтеграції також складався з кількох частин. Спочатку було виконано верстку базового шаблону (base.html) з навігацією та підключенням Bootstrap та JavaScript. Потім створено шаблони для всіх представлень (наприклад, ticket_list.html, ticket_detail.html) з використанням Django Template Language (DTL). Фінальним кроком цього етапу стало написання JavaScript коду для забезпечення інтерактивності, такого як обробка коментарів через AJAX та динамічне оновлення бейджа сповіщень. Завершальна фаза включила тестування, оптимізацію та впровадження. Вона почалася з написання модульних тестів (Django TestCase) для моделей та представлень. Після цього було проведено інтеграційне тестування ключових сценаріїв, таких як повний цикл обробки заявки: створення, призначення, обробка та закриття. Наступним кроком стало профілювання продуктивності системи, оптимізація повільних запитів до бази даних та додавання необхідних індексів. Нарешті, система була підготовлена до роботи в продуктивному середовищі: створено production-налаштування (settings/production.py), виконано

розгортання на хостингу (наприклад, Heroku, PythonAnywhere або VPS), а також налаштовано доменне ім'я та SSL-сертифікат.

2.3.3 Комплексна стратегія управління якістю та ризиками

Впровадження процесів для забезпечення якості коду було комплексним. Щодо контролю версій, використовувалася модель гілкування Git Flow або її спрощена версія. Основна гілка main завжди містить стабільну, робочу версію, тоді як активна розробка ведеться в гілці develop. Кожна нова функція (наприклад, feature/ticket-crud) створюється у відповідній feature-гілці, що відгалужується від develop, і зливається назад після повного завершення та рев'ю коду. Систематичне рев'ю коду є обов'язковим етапом: всі Pull Request (PR) з новим кодом обов'язково переглядаються. Мета цього процесу полягає у виявленні логічних помилок, дотриманні стилю коду (PEP 8 для Python), усуненні зайвих складностей та визначенні можливостей для рефакторингу. Це найефективніший спосіб підвищити загальну якість продукту та поширити знання про кодбазу серед учасників команди. Крім того, було впроваджено багаторівневу стратегію тестування. Модульні тести (Unit) відповідають за перевірку найменших, окремих частин програми, таких як методи моделі на кшталт Ticket.is_overdue(). Інтеграційні тести (Integration) перевіряють коректну взаємодію кількох компонентів, наприклад, чи викликається правильне сповіщення при створенні коментаря до заявки. Системні або End-to-End тести імітують поведінку реального користувача за допомогою інструментів на кшталт Playwright або Selenium, охоплюючи повні сценарії, як-от: "користувач входить у систему, створює нову заявку, а потім перевіряє її статус". Останнім ключовим елементом є неперервна інтеграція (Continuous Integration). Для цього використовуються сервіси на кшталт GitHub Actions або GitLab CI, які автоматично запускають набір тестів та перевірки стилю коду при кожному новому коміті в гілку develop або створенні Pull Request. Це забезпечує миттєвий зворотний зв'язок про стан проєкту та дозволяє оперативно виявляти та усувати потенційні проблеми.

2.4 Архітектурні рішення

2.4.1 Вибір архітектурного шаблону MVT у Django

Django реалізує варіацію патерну MVC під назвою MVT (Model-View-Template), де суть залишається у розділенні відповідальностей, незважаючи на відмінності у назвах. Модель (Model) відповідає за дані та бізнес-логіку, виступаючи абстракцією таблиці бази даних. У Django модель (models.Model) визначає поля, методи, менеджери та метадані, при цьому вона не знає нічого про те, як дані будуть представлені. Представлення, або View, яке є аналогом Контролера в MVC, виступає обробником запитів (request handler). Воно отримує HTTP-запит, виконує необхідні операції, такі як звернення до моделей та обробка форм, формує контекст даних і передає його шаблону. У проєкті широко використані Class-Based Views (CBV), такі як ListView, DetailView, CreateView, які інкапсулюють загальну логіку, дозволяючи перевизначати лише потрібні методи (get_queryset, form_valid). Шаблон (Template), що є аналогом Представлення в MVC, відповідає за відображення (presentation layer). Це HTML-файли зі спеціальним синтаксисом Django Template Language (DTL) для вставки даних з контексту, тегів та фільтрів. Шаблони є пасивними, оскільки лише відображають те, що їм передав View. Така архітектура забезпечує низьку зв'язаність (low coupling) компонентів, що робить систему легко тестованою (моделі та представлення можна тестувати ізольовано) та підтримуваною, оскільки зміни в одному шарі мінімально впливають на інші.

2.4.2 Проектування компонентів системи

Проект побудований на принципі «одна функціональна область – один додаток». Кожен додаток у папці apps/ є самодостатнім модулем [9, 16]:

1. apps.core: Фундаментальний додаток. Містить спільні для всієї системи компоненти: абстрактні базові моделі (AbstractBaseModel з полями created_at,

updated_at), кастомні менеджери моделей, утиліти (функції-помічники), загальні валідатори, константи.

2. apps.users: Додаток управління користувачами. Інкапсулює все, що пов'язано з користувачами: кастомну модель CustomUser, форми реєстрації/редагування, бекенд аутентифікації, сигнали для створення профілю, кастомізовану адмін-панель. Це центр системи безпеки.
3. apps.tickets: Бізнес-ядро системи. Найбільший додаток. Містить моделі предметної області (Ticket, Comment, Status, Category), всі представлення для роботи з ними, форми, кастомні теги та фільтри для шаблонів, сигнали для бізнес-автоматизації, команди управління (management commands).
4. apps.notifications: Додаток сповіщень. Незалежний сервісний модуль. Модель Notification, API endpoints (RESTful) для взаємодії з фронтендом, сервісний клас NotificationService для централізованого створення сповіщень, WebSocket-споживачі (consumers) для real-time.
5. apps.reports: Додаток аналітики. Модуль, що залежить від даних з tickets. Містить представлення для звітів, сервісний клас ReportGenerator для агрегації даних та логіки експорту в Excel.

Переваги такого підходу:

- Покращена організація коду: Легко знайти код, пов'язаний з конкретною функцією.
- Можливість повторного використання: Додаток users або core можна використати в іншому проекті.
- Зосереджене тестування: Можна тестувати кожен додаток окремо.
- Паралельна розробка: Різні розробники можуть працювати над різними додатками з мінімальними конфліктами.

2.4.3 Проектування схеми бази даних

Спроєктована схема БД є класичною реляційною моделлю з дотриманням принципів нормалізації для уникнення аномалій [11]. Ось ключові сутності та їхні зв'язки:

- users_customuser (розширення стандартної таблиці auth_user): Головна таблиця користувачів. Поля: id, username, email, password, role (client/operator/admin), first_name, last_name, phone, avatar, is_active. Важливо: Поле role є основою для RBAC.
- tickets_ticket: Центральна сутність системи. Поля: id, title, description, status_id (FK до tickets_status), priority (low/medium/high/critical), category_id (FK, nullable), created_by_id (FK до users_customuser, nullable – на випадок анонімних заявок), assigned_to_id (FK до users_customuser, nullable), deadline (DateTime, nullable), resolved_at (DateTime, nullable), created_at, updated_at.
- tickets_comment: Сутність для зберігання історії комунікації. Поля: id, ticket_id (FK до tickets_ticket), author_id (FK до users_customuser), text, is_internal (boolean), created_at, updated_at. Зв'язок: Ticket --< Comment (один до багатьох).
- tickets_status та tickets_category: Довідники (lookup tables). Містять стандартні набори значень, що керуються адміністратором. Мають поля id, name, slug (для URL), color (для відображення), description.

Ключові технічні рішення для БД:

- Індекси: Створено індекси для полів, що використовуються в умовах пошуку та сортування (див. рис. 2.1).

```
CREATE INDEX idx_ticket_status_created ON tickets_ticket (status_id, created_at DESC);
CREATE INDEX idx_ticket_assigned_status ON tickets_ticket (assigned_to_id, status_id);
CREATE INDEX idx_ticket_created_by ON tickets_ticket (created_by_id, created_at DESC);
```

Рисунок 2.1 – Індекси для полів в умовах пошуку та сортування

- Цілісність даних: Використані ForeignKey з on_delete правилами (CASCADE для коментарів при видаленні заявки, PROTECT для статусу, щоб не випадково не видалити використовуваний статус, SET_NULL для категорий та відповідальних).
- Аудит: Всі основні таблиці мають поля created_at та updated_at (автоматично оновлюються Django), що дозволяє відстежувати час створення та останніх змін.

Ця структура забезпечує ефективність виконання запитів, надійне зберігання даних та гнучкість для подальшого розширення (наприклад, додавання нових полей до заявки або нових типів коментарів).

3 ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Архітектура системи

3.1.1 Загальна архітектура системи

Система побудована на Django 4.2.26 з архітектурою MVT (Model-View-Template). Основний стек (див. рис. 3.1, 3.2) включає Python 3.10+, Bootstrap 5 для фронтенду, PostgreSQL або SQLite для зберігання даних та Celery з Redis для асинхронних завдань. Використання модульної архітектури забезпечує чітке розділення відповідальності та легкість супроводу, що є критично важливим для малого та середнього бізнесу.

```python
# ФАЙЛ: helpdesk/settings/base.py
# Основний стек додатків
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # local apps
    'apps.core',
    'apps.users',
    'apps.tickets',      # Ядро системи - заявки
    'apps.notifications', # Система сповіщень
    'apps.reports',      # Аналітика та звіти
]

# Архітектура MVT: шаблони
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'apps.notifications.context_processors.notifications', # Кастомні сповіщення
            ],
        },
    },
]
```

Рисунок 3.1 – Основний стек

```
# Локалізація для україномовного інтерфейсу
LANGUAGE_CODE = 'uk'
TIME_ZONE = 'Europe/Kiev'
USE_I18N = True

# Конфігурація Celery для фонових завдань (SLA, сповіщення)
CELERY_BROKER_URL = 'redis://localhost:6379/0'
CELERY_BEAT_SCHEDULE = {
    'check_overdue_deadlines_hourly': {
        'task': 'apps.tickets.tasks.check_overdue_deadlines',
        'schedule': crontab(minute=0),  # Щогодини
    },
}
```

Рисунок 3.2 – Продовження малюнку 3.1

Рольова модель користувачів (див. рис. 3.3) забезпечує чітке розмежування прав доступу між клієнтами, операторами та адміністраторами.

```
# ФАЙЛ: apps/users/models.py
class CustomUser(AbstractUser):
    ROLE_CHOICES = [
        ('client', 'Клієнт'),
        ('operator', 'Оператор'),
        ('admin', 'Адміністратор'),
    ]
    role = models.CharField('роль', max_length=20, choices=ROLE_CHOICES, default='client')
    phone = models.CharField('телефон', max_length=50, blank=True)
    department = models.CharField('відділ', max_length=200, blank=True)
    is_available = models.BooleanField('доступний', default=True)

    def __str__(self):
        return f"{self.get_full_name()} ({self.get_role_display()})"
```

Рисунок 3.3 – Рольова модель користувачів

3.1.2 Компоненти бекенду та їх призначення

Система побудована на модульній архітектурі, де кожен компонент відповідає за чітко визначену функціональну область. Додаток управління заявками (див. рис. 3.4) є ядром системи і призначений для реалізації повного життєвого циклу обробки заявок техпідтримки – від створення до архівації, з контролем статусів, пріоритетів, SLA та комунікації.

```
# ФАЙЛ: apps/tickets/models.py
class Ticket(AbstractBaseModel):
    PRIORITY_CHOICES = [('low', 'Низький'), ('medium', 'Середній'), ('high', 'Високий'), ('critica
l', 'Критичний')]

    title = models.CharField('заголовок', max_length=200)
    description = models.TextField('опис')
    status = models.ForeignKey('Status', verbose_name='статус', on_delete=models.PROTECT)
    priority = models.CharField('пріоритет', max_length=20, choices=PRIORITY_CHOICES, default='mediu
m')
    created_by = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name='створив', on_delete=model
s.SET_NULL, null=True, related_name='created_tickets')
    assigned_to = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name='призначено', on_delete=m
odels.SET_NULL, null=True, blank=True, related_name='assigned_tickets')
    deadline = models.DateTimeField('термін', null=True, blank=True)
    resolved_at = models.DateTimeField('виправлено', null=True, blank=True)

    def is_overdue(self):
        """Перевірка, чи протермінована заявка."""
        return self.deadline and timezone.now() > self.deadline

    def get_absolute_url(self):
        return reverse('tickets:detail', kwargs={'pk': self.pk})

class Comment(AbstractBaseModel):
    ticket = models.ForeignKey(Ticket, on_delete=models.CASCADE, related_name='comments')
    author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.SET_NULL, null=True)
    text = models.TextField('текст')
    is_internal = models.BooleanField('внутрішній', default=False)
```

3.4 – Додаток управління заявками

Бізнес-логіка та контроль доступу реалізовані у представленнях (views), де кожна роль користувача має обмежені права (див. рис. 3.5,3.6).

```python
# ФАЙЛ: apps/tickets/views.py
class TicketListView(LoginRequiredMixin, ListView):
    model = Ticket
    paginate_by = 20

    def get_queryset(self):
        user = self.request.user
        qs = Ticket.objects.all().select_related('status', 'created_by', 'assigned_to')

        # Клієнти бачать тільки свої заявки
        if user.role == 'client':
            qs = qs.filter(created_by=user)
        # Оператори можуть фільтрувати "Призначені мені"
        elif user.role == 'operator' and self.request.GET.get('assigned') == '1':
            qs = qs.filter(assigned_to=user)

        # Фільтрація за статусом та пріоритетом
        status_filter = self.request.GET.get('status')
        if status_filter:
            qs = qs.filter(status__slug=status_filter)
        return qs.order_by('-created_at')

class TicketCreateView(LoginRequiredMixin, CreateView):
    model = Ticket
    form_class = TicketForm
    template_name = 'tickets/ticket_form.html'

    def dispatch(self, request, *args, **kwargs):
        # Тільки клієнти можуть створювати заявки
        if not request.user.is_authenticated or request.user.role != 'client':
            messages.error(request, 'Тільки клієнти можуть створювати заявки.')
            return redirect('tickets:list')
```

Рисунок 3.5 – Бізнес логіка та контроль доступу

```
        return super().dispatch(request, *args, **kwargs)

    def form_valid(self, form):
        form.instance.created_by = self.request.user
        form.instance.status = Status.objects.get(slug='new')
        ticket = form.save()

        # Сповіщення адміністраторів про нову заявку
        notification_service = NotificationService()
        admins = CustomUser.objects.filter(role='admin', is_active=True)
        for admin in admins:
            notification_service.create_in_app_notification(
                admin,
                'Нова заявка',
                f'Створено нову заявку "{ticket.title}" від {ticket.created_by.get_full_name()}.',
                url=ticket.get_absolute_url(),
            )
        messages.success(self.request, 'Заявку успішно створено!')
        return redirect(ticket)
```

Рисунок 3.6 – Продовження малюнка 3.5

Додаток управління користувачами (users) призначений для централізованого управління аутентифікацією, авторизацією та профілями користувачів (див. рис. 3.7).

```
# ФАЙЛ: apps/users/auth_backend.py
class EmailAuthBackend(ModelBackend):
    """Автентифікація за email або username."""
    def authenticate(self, request, username=None, password=None, **kwargs):
        try:
            user = CustomUser.objects.filter(
                Q(email__iexact=username) | Q(username__iexact=username)
            ).first()
            if user and user.check_password(password) and self.user_can_authenticate(user):
                return user
        except CustomUser.DoesNotExist:
            return None
```

Рисунок 3.7 – Управління аутентифікацією, авторизацією та профілями користувачів

Додаток аналітики та звітності (reports) відповідає за генерацію звітів та аналітику ефективності роботи служби підтримки (див. рис. 3.8).

```
# ФАЙЛ: apps/reports/services/report_generator.py
class ReportGenerator:
    def get_ticket_metrics(self, start_date, end_date):
        q = Ticket.objects.filter(created_at__gte=start_date, created_at__lte=end_date)
        metrics = q.aggregate(
            total_created=Count('id'),
            total_resolved=Count('id', filter=Q(resolved_at__isnull=False)),
        )
        # Розрахунок середнього часу вирішення
        resolved_qs = q.filter(resolved_at__isnull=False)
        if resolved_qs.exists():
            avg_seconds = resolved_qs.annotate(
                resolution_time=ExpressionWrapper(F('resolved_at') - F('created_at'), output_field=D
urationField())
            ).aggregate(avg=Avg('resolution_time'))['avg'].total_seconds()
            metrics['avg_resolution_hours'] = round(avg_seconds / 3600, 2)
        return metrics

    def export_comprehensive_report_to_excel(self, start_date, end_date):
        """Генерує Excel-файл з 8 аркушами: метрики, SLA, продуктивність операторів тощо."""
        # Код створення файлу за допомогою openpyxl...
        return excel_file_content
```

Рисунок 3.8 – Додаток аналітики та звітності

Додаток сповіщень (notifications) реалізує багатоканальну систему сповіщень (in-app, WebSocket) для інформування користувачів (див. рис. 3.9).

```python
# ФАЙЛ: apps/notifications/services.py
class NotificationService:
    @staticmethod
    def create_in_app_notification(user, title, message, url=None):
        n = Notification.objects.create(user=user, title=title, message=message, target_url=url or
'')
        # Реалізація відправки через WebSocket (якщо доступно)
        try:
            channel_layer = get_channel_layer()
            async_to_sync(channel_layer.group_send)(f'notifications_{user.id}', {
                'type': 'notify',
                'title': title,
                'message': message,
                'target_url': url,
            })
        except Exception:
            pass  # WebSocket не налаштовано
        return n
```

Рисунок 3.9 – Додаток сповіщень

3.1.3 Структура проекту

Проект має стандартну для Django модульну архітектуру, організовану навколо головного додатку helpdesk/ та підлеглих бізнес-додатків у директорії apps/. Ключовим елементом є чітке розділення конфігурацій за середовищами виконання через папку settings/, що містить окремі файли для базових налаштувань (base.py), розробки (development.py) та продакшену (production.py). У корені розташовані службові файли: manage.py для виконання команд, requirements.txt для залежностей та .env.example для шаблону змінних оточення. Бізнес-логіка інкапсульована в окремих додатках у директорії apps/, де кожен модуль виконує певну роль: core містить абстрактні моделі та утиліти, users відповідає за автентифікацію та профілі, tickets є ядром системи з управлінням заявками, notifications реалізує механізм сповіщень, а reports забезпечує аналітику та генерацію звітів. Для обробки асинхронних завдань налаштовано окремий файл celery.py. Клієнтська частина

системи організована через глобальні шаблони у папці templates/, які використовують Bootstrap 5, та статичні ресурси (CSS, JavaScript, зображення) у директорії static/. Користувальницькі файли зберігаються в media/, а підтримка інтернаціоналізації реалізована через файли перекладів у locale/. Тестування всіх компонентів забезпечується комплексом тестів, розміщених у кореневій директорії tests/.

Пояснення ключових аспектів структури:

- Модульність (apps/): Кожен додаток інкапсулює свою функціональність, що дозволяє легко додавати нові модулі або вимикати існуючі.
- Розділення налаштувань (settings/): Окрема конфігурація для розробки та продакшену.
- Додаток core: Містить спільну логіку (наприклад, AbstractBaseModel з полями created_at, updated_at).

3.1.4 Взаємодія компонентів

Взаємодія між компонентами системи відбувається за чітко визначеними механізмами Django та власними сервісами, що забезпечує слабку зв'язаність та легкість тестування.

Сценарій: Створення нової заявки клієнтом

1. Клієнт → через UI (Bootstrap форма) відправляє POST-запит на /tickets/create/.
2. TicketCreateView (apps/tickets/views.py): Перевіряє, чи користувач має роль client. Валідує дані через TicketForm. У form_valid(): присвоює заявці автора, статус "нова", зберігає. Викликає
3. NotificationService для сповіщення адміністраторів. NotificationService (apps/notifications/services.py): Створює запис Notification для кожного активного адміністратора. Спроба відправити сповіщення в реальному часі через WebSocket.
4. Клієнт отримує перенаправлення на сторінку створеної заявки та підтвердження.

Діаграма взаємодії для сценарію створення заявки зображена на рисунку 3.10:

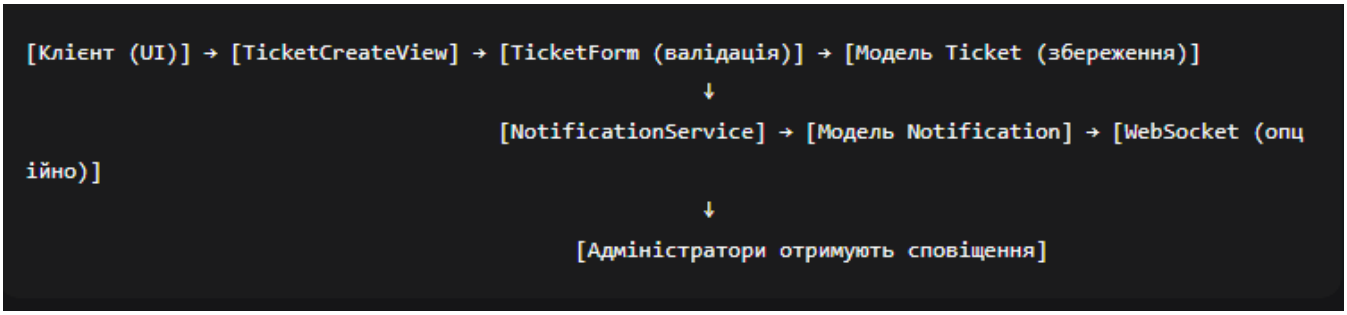


Рисунок 3.10 – Діаграма взаємодії для сценарію створення заявки

Сценарій: Призначення оператора та зміна статусу

1. Адміністратор у детальному перегляді заявки обирає оператора та новий статус.
2. TicketStatusUpdateView (apps/tickets/views.py): Перевіряє права (admin або operator, якщо заявка призначена йому). Оновлює поля assigned_to, status. При статусі "вирішено" заповнює resolved_at. Викликає NotificationService для сповіщення призначеного оператора.
3. Сигнал post_save для моделі Ticket (apps/tickets/signals.py): Якщо статус стає "закрито" або "вирішено", автоматично видаляє всі сповіщення, пов'язані з цією заявкою.

Сценарій: Генерація звіту адміністратором

1. Адміністратор обирає період та натискає "Експортувати звіт".
2. ExportReportView (apps/reports/views.py): Перевіряє роль користувача. Викликає
3. ReportGenerator.export_comprehensive_report_to_excel(). ReportGenerator запитує дані з моделей Ticket, CustomUser, агрегує метрики, формує Excel-файл з 8 аркушами.
4. Браузер адміністратора завантажує згенерований .xlsx файл.

3.2 Реалізація ключових функціональностей

3.2.1 Модуль управління заявками

Реалізація життєвого циклу заявки є ядром системи та охоплює всі етапи від створення до архівації. Кожен статус має чітко визначені переходи та бізнес-правила (див. рис. 3.11).

```python
# Константи допустимих переходів для оператора
OPERATOR_ALLOWED_STATUS_SLUGS = ['in_progress', 'waiting_client', 'waiting_parts', 'resolved']

class TicketStatusUpdateView(LoginRequiredMixin, View):
    def post(self, request, pk):
        ticket = get_object_or_404(Ticket, pk=pk)
        new_status_slug = request.POST.get('status')

        # Авторизація: тільки оператор або адмін
        if request.user.role not in ('operator', 'admin'):
            return JsonResponse({'ok': False, 'error': 'forbidden'}, status=403)

        # Оператор може змінювати статус тільки призначених йому заявок
        if request.user.role == 'operator':
            if ticket.assigned_to != request.user:
                return JsonResponse({'ok': False, 'error': 'forbidden'}, status=403)
            if new_status_slug not in OPERATOR_ALLOWED_STATUS_SLUGS:
                return JsonResponse({'ok': False, 'error': 'status not permitted'}, status=400)

        # Оновлення статусу та resolved_at
        new_status = Status.objects.get(slug=new_status_slug)
        ticket.status = new_status
        if new_status.slug == 'resolved':
            ticket.resolved_at = timezone.now()
        ticket.save()

        return JsonResponse({'ok': True, 'new_status': new_status.name})
```

Рисунок 3.11 – Життєвий цикл заявки

Автоматизація робочих процесів за допомогою сигналів (див. рис. 3.12):

```python
# ФАЙЛ: apps/tickets/signals.py
@receiver(post_save, sender=Comment)
def handle_new_comment(sender, instance, created, **kwargs):
    """Автоматично змінює статус заявки на 'in_progress', якщо оператор додав коментар."""
    if created and instance.author.role == 'operator':
        ticket = instance.ticket
        if ticket.status.slug != 'in_progress':
            in_progress_status = Status.objects.get(slug='in_progress')
            ticket.status = in_progress_status
            ticket.save(update_fields=['status'])
```

Рисунок 3.12 – Автоматизація робочих процесів за допомогою сигналів

3.2.2 Механізми аутентифікації та авторизації

Система безпеки реалізована на основі кастомної моделі користувача та розширеного механізму перевірки прав доступу (див. рис. 3.13).

```python
# ФАЙЛ: apps/core/utils/access.py
def role_required(allowed_roles):
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            if not request.user.is_authenticated:
                return redirect('users:login')
            if request.user.role not in allowed_roles:
                return HttpResponseForbidden("Недостатньо прав для доступу")
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator

# Використання у views.py
@method_decorator(role_required(['admin']), name='dispatch')
class ReportDashboardView(TemplateView):
    # Тільки адміністратор має доступ
```

Рисунок 3.13 – Реалізація системи безпеки

Адмін-панель для управління користувачами (див. рис. 3.14):

```python
# ФАЙЛ: apps/users/admin.py
@admin.register(CustomUser)
class CustomUserAdmin(UserAdmin):
    # Видаляємо зайві поля для спрощення
    fieldsets = (
        (None, {'fields': ('username', 'password')}),
        ('Персональна інформація', {'fields': ('first_name', 'last_name', 'email', 'phone', 'departm
ent')}),
        ('Права доступу', {'fields': ('role', 'is_active', 'is_staff')}), # is_superuser приховано
    )
    # Заборона видаляти адміністраторів
    def has_delete_permission(self, request, obj=None):
        if obj and obj.role == 'admin':
            return False
        return super().has_delete_permission(request, obj)
```

Рисунок 3.14 – Адмін-панель для управління користувачами

3.2.3 Система сповіщень та нотифікації

Багатоканальна система сповіщень забезпечує своєчасне інформування користувачів про зміни в системі.

Логіка створення сповіщень при додаванні коментаря (див. рис. 3.15):

```python
# ФАЙЛ: apps/tickets/views.py (CommentCreateView)
def form_valid(self, form):
    comment = form.save(commit=False)
    comment.ticket = self.ticket
    comment.author = self.request.user
    comment.save()

    notification_service = NotificationService()

    # СПОВІЩЕННЯ КЛІЄНТУ: лише якщо коментар публічний і написав не сам клієнт
    if (not comment.is_internal and
        self.ticket.created_by and
        self.ticket.created_by != self.request.user and
        self.ticket.created_by.role == 'client'):
        notification_service.create_in_app_notification(
            self.ticket.created_by,
            'Новий коментар',
            f'Додано коментар до вашої заявки "{self.ticket.title}".',
            url=self.ticket.get_absolute_url(),
        )

    # СПОВІЩЕННЯ ОПЕРАТОРА: лише якщо заявка призначена і не йому самому
    if (self.ticket.assigned_to and
        self.ticket.assigned_to != self.request.user):
        notification_service.create_in_app_notification(
            self.ticket.assigned_to,
            'Новий коментар',
            f'Додано коментар до призначеної вам заявки "{self.ticket.title}".',
            url=self.ticket.get_absolute_url(),
        )
    return JsonResponse({'ok': True, 'comment_id': comment.id})
```

Рисунок 3.15 – Логіка створення сповіщень при додаванні коментаря

API для отримання сповіщень (див. рис. 3.16):

```python
# ФАЙЛ: apps/notifications/views.py
class UnreadNotificationsView(LoginRequiredMixin, View):
    def get(self, request):
        notifications = Notification.objects.filter(
            user=request.user
        ).order_by('-created_at')[:10]
        unread_count = notifications.filter(is_read=False).count()

        data = [{
            'id': n.id,
            'title': n.title,
            'message': n.message,
            'target_url': n.target_url,
            'is_read': n.is_read,
            'created_at': n.created_at.strftime('%H:%M')
        } for n in notifications]

        return JsonResponse({'notifications': data, 'unread_count': unread_count})
```

Рисунок 3.16 – API для отримання сповіщень

3.3 Інтерфейс користувача та адаптивний дизайн

3.3.1 Компонентна система на основі Bootstrap 5

Інтерфейс системи побудований з використанням модульних компонентів Bootstrap 5, що забезпечує консистентність та адаптивність на всіх типах пристроїв.

Адаптивна картка заявки в списку - templates/tickets/ticket_list.html (див. рис. 3.17):

```
<div class="row row-cols-1 row-cols-md-2 g-3">
  {% for ticket in tickets %}
  <div class="col">
    <div class="card h-100 shadow-sm">
      <div class="card-body d-flex flex-column">
        <div class="d-flex justify-content-between align-items-start mb-2">
          <h5 class="card-title">
            <a href="{{ ticket.get_absolute_url }}">{{ ticket.title }}</a>
          </h5>
          <span class="badge" style="background-color: {{ ticket.status.color }};">
            {{ ticket.status.name }}
          </span>
        </div>
        <p class="card-text text-muted flex-grow-1">{{ ticket.description|truncatewords:20 }}</p>
        <div class="mt-auto pt-2 border-top">
          <small class="text-muted">
            Створено: {{ ticket.created_by.get_full_name|default:ticket.created_by.username }} |
            {{ ticket.created_at|date:"d.m.Y H:i" }}
          </small>
        </div>
      </div>
    </div>
  </div>
  {% endfor %}
</div>
```

Рисунок 3.17 – Адаптивна картка заявки

Навігаційна панель з динамічним меню та бейджем сповіщень (див. рис. 3.18):

```
<!-- templates/base.html -->
<nav class="navbar navbar-expand-lg navbar-dark bg-primary mb-4">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">HelpDesk</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#mainNa
v">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="mainNav">
      <ul class="navbar-nav me-auto">
        {% if user.role == 'client' %}
          <li class="nav-item"><a class="nav-link" href="{% url 'tickets:create' %}">Створити заявку
</a></li>
        {% elif user.role == 'admin' %}
          <li class="nav-item"><a class="nav-link" href="{% url 'reports:index' %}">Звіти</a></li>
        {% endif %}
      </ul>
      <ul class="navbar-nav">
        <!-- Dropdown сповіщень з AJAX -->
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="notificationDropdown" role="button" data-
bs-toggle="dropdown">
            🔔 <span id="notificationBadge" class="badge bg-danger"></span>
          </a>
          <ul class="dropdown-menu dropdown-menu-end" id="notificationList">
            <!-- Список сповіщень завантажується через JavaScript -->
          </ul>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

Рисунок 3.18 – Навігаційна панель

3.3.2 Адаптивні таблиці та форми

Форми та таблиці адаптовані для зручного використання на мобільних пристроях.

Форма зміни статусу з валідацією дедлайну (див. рис. 3.19):

```python
# ФАЙЛ: apps/tickets/forms.py
class TicketDeadlineForm(forms.ModelForm):
    class Meta:
        model = Ticket
        fields = ['deadline']
        widgets = {
            'deadline': forms.DateTimeInput(
                attrs={
                    'type': 'datetime-local',
                    'class': 'form-control form-control-sm'
                },
                format='%Y-%m-%dT%H:%M'
            )
        }

    def __init__(self, *args, **kwargs):
        self.user_role = kwargs.pop('user_role', 'client')
        super().__init__(*args, **kwargs)

    def clean_deadline(self):
        deadline = self.cleaned_data.get('deadline')
        # Адміністратор може встановити будь-який дедлайн
        if self.user_role == 'admin':
            return deadline
        # Оператор не може встановити дедлайн раніше дати створення
        if deadline and self.instance.created_at and deadline < self.instance.created_at:
            raise forms.ValidationError("Дедлайн не може бути раніше дати створення заявки.")
        return deadline
```

Рисунок 3.19 – Форма зміни статусу з валідацією дедлайну

JavaScript для динамічного оновлення інтерфейсу (сповіщення, коментарі) (див. рис. 3.20):

```
// Оновлення бейджа сповіщень кожні 30 секунд
function updateNotificationBadge() {
    fetch('/notifications/api/unread/')
        .then(response => response.json())
        .then(data => {
            const badge = document.getElementById('notificationBadge');
            if (data.unread_count > 0) {
                badge.textContent = data.unread_count;
                badge.style.display = 'inline-block';
            } else {
                badge.style.display = 'none';
            }
            // Оновлення dropdown списку
            const list = document.getElementById('notificationList');
            list.innerHTML = data.notifications.map(n => `
                <li><a class="dropdown-item ${n.is_read ? '' : 'fw-bold'}" href="${n.target_url}">
                    ${n.title}: ${n.message}
                </a></li>
            `).join('');
        });
}
setInterval(updateNotificationBadge, 30000);
```

Рисунок 3.20 – JavaScript для динамічного оновлення інтерфейсу

3.4 Забезпечення якості та безпеки

3.4.1 Система тестування та валідації

Для забезпечення надійності системи реалізовано комплексну систему тестування.

Модульні тести для перевірки бізнес-логіки та контролю доступу (див. рис. 3.21):

```
# ФАЙЛ: apps/tickets/tests.py
class TicketAccessTests(TestCase):
    def setUp(self):
        self.client_user = CustomUser.objects.create_user(username='client', password='pass', role
='client')
        self.operator_user = CustomUser.objects.create_user(username='operator', password='pass', ro
le='operator')
        self.admin_user = CustomUser.objects.create_user(username='admin', password='pass', role='ad
min')
        self.ticket = Ticket.objects.create(title='Test', description='Test', created_by=self.client
_user)

    def test_client_sees_only_own_tickets(self):
        self.client.login(username='client', password='pass')
        response = self.client.get(reverse('tickets:list'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, 'Test')  # Бачить свою заявку

    def test_operator_cannot_access_admin_reports(self):
        self.client.login(username='operator', password='pass')
        response = self.client.get(reverse('reports:index'))
        self.assertEqual(response.status_code, 403)  # Доступ заборонено

    def test_auto_status_change_on_operator_comment(self):
        self.client.login(username='operator', password='pass')
        response = self.client.post(reverse('tickets:comment', args=[self.ticket.pk]), {'text': 'Tes
t comment'})
        self.ticket.refresh_from_db()
        self.assertEqual(self.ticket.status.slug, 'in_progress')  # Статус змінився автоматично
```

Рисунок 3.21 – Модульні тести для перевірки бізнес-логіки та контролю доступу

E2E-тести критичних сценаріїв (див. рис. 3.22):

```
// tests/e2e/ticket_workflow.spec.ts
import { test, expect } from '@playwright/test';

test('Повний цикл заявки: створення → призначення → вирішення', async ({ page }) => {
  // 1. Клієнт створює заявку
  await page.goto('/tickets/create/');
  await page.fill('#id_title', 'Проблема з принтером');
  await page.fill('#id_description', 'Не друкує чорно-білі документи');
  await page.click('button[type="submit"]');
  await expect(page).toHaveURL(/\/tickets\/\d+\//);

  // 2. Адміністратор призначає оператора
  await page.click('text=Вийти');
  // ... логін адміністратора, пошук заявки, призначення
  // 3. Оператор змінює статус
  // 4. Перевірка фінального стану
});
```

Рисунок 3.22 – E2E-тести критичних сценаріїв

3.4.2 Захист від веб-загроз

Система включає комплексний захист від поширених веб-загроз, використовуючи вбудовані механізми Django та додаткові перевірки.

Конфігурація безпеки Django для продакшену (див. рис. 3.23):

```python
# ФАЙЛ: helpdesk/settings/production.py
DEBUG = False
ALLOWED_HOSTS = ['yourdomain.com']  # Конкретний домен

# Захист від XSS та кліків
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
X_FRAME_OPTIONS = 'DENY'

# HTTPS та Cookies
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
CSRF_COOKIE_HTTPONLY = True

# Заголовки HSTS
SECURE_HSTS_SECONDS = 31536000  # 1 рік
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True
```

Рисунок 3.23 – Конфігурація безпеки Django для продакшену

Валідація завантажуваних файлів на рівні форми (див. рис. 3.24):

```python
# ФАЙЛ: apps/core/validators.py
def validate_file_size(value):
    limit = 10 * 1024 * 1024  # 10 MB
    if value.size > limit:
        raise ValidationError('Файл занадто великий. Розмір не повинен перевищувати 10 МБ.')

def validate_file_extension(value):
    ext = os.path.splitext(value.name)[1].lower()
    valid_extensions = ['.jpg', '.jpeg', '.png', '.pdf', '.doc', '.docx']
    if ext not in valid_extensions:
        raise ValidationError('Недопустимий тип файлу. Дозволені: JPG, PNG, PDF, DOC, DOCX.')

# Використання у моделі
class TicketAttachment(models.Model):
    file = models.FileField(
        upload_to='attachments/',
        validators=[validate_file_size, validate_file_extension]
    )
```

Рисунок 3.24 – Валідація завантажуваних файлів на рівні форми

Захист від несанкціонованого доступу на рівні представлень та шаблонів (див. рис. 3.25):

```python
# ФАЙЛ: apps/tickets/views.py
class TicketDetailView(LoginRequiredMixin, DetailView):
    def get_queryset(self):
        # Кожна роль бачить тільки дозволені заявки
        user = self.request.user
        if user.role == 'client':
            return Ticket.objects.filter(created_by=user)
        elif user.role == 'operator':
            return Ticket.objects.filter(Q(assigned_to=user) | Q(assigned_to__isnull=True))
        return Ticket.objects.all()  # admin
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        # Клієнтам не показуємо внутрішні коментарі
        if self.request.user.role == 'client':
            context['comments'] = self.object.comments.filter(is_internal=False)
        else:
            context['comments'] = self.object.comments.all()
        return context
```

Рисунок 3.25 – Захист від несанкціонованого доступу на рівні представлень та шаблонів

3.5 Візуалізація інтерфейсу та користувацький досвід

Розроблений інтерфейс інформаційної системи орієнтований на максимальну простоту, інтуїтивність та ефективність роботи для всіх категорій користувачів. Візуальна складова реалізована з використанням компонентів Bootstrap 5, що забезпечує сучасний вигляд, консистентність та безумовну адаптивність під будь-які пристрої – від десктопних моніторів до екранів смартфонів.

Авторизація та особистий кабінет (див. рис. 3.26). Першим екраном, з яким стикається користувач, є форма входу в систему з перевіркою логіна та пароля. Після успішної автентифікації користувач потрапляє на персоналізовану головну сторінку (дашборд), вміст якої динамічно змінюється залежно від його ролі. Для клієнта це може бути швидкий доступ до створення нової заявки та список власних активних звернень. Для оператора – панель із завданнями, призначеними саме йому, фільтрами за пріоритетом та статусом. Адміністратор бачить загальну статистику та швидкі посилання на управління системою.

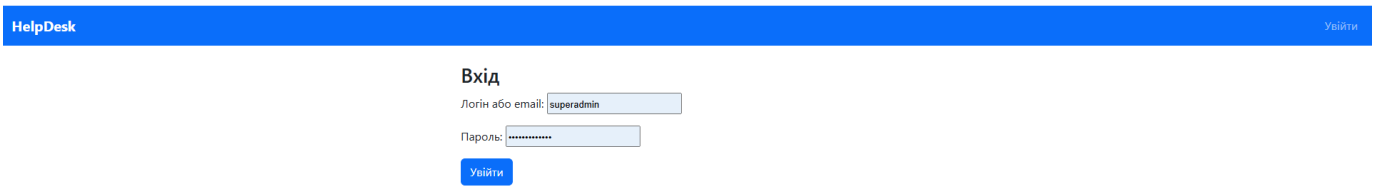


Рисунок 3.26 – Авторизація на сайті

Централізований список заявок (див. рис. 3.27). Це основний робочий інструмент оператора та адміністратора. Інтерфейс реалізовано у вигляді адаптивної таблиці (або сітки карток для мобільних пристроїв) з можливістю миттєвої фільтрації за ключовими параметрами: статусом («Нова», «В роботі», «Вирішено»), пріоритетом, категорією або відповідальним співробітником. Кожен рядок містить

найважливішу інформацію: номер, заголовок, клієнта, призначеного оператора, термін вирішення (SLA) та візуальний індикатор статусу у вигляді кольорового бейджа. Система автоматично підсвічує протерміновані заявки, що дозволяє оперативно реагувати на критичні ситуації.

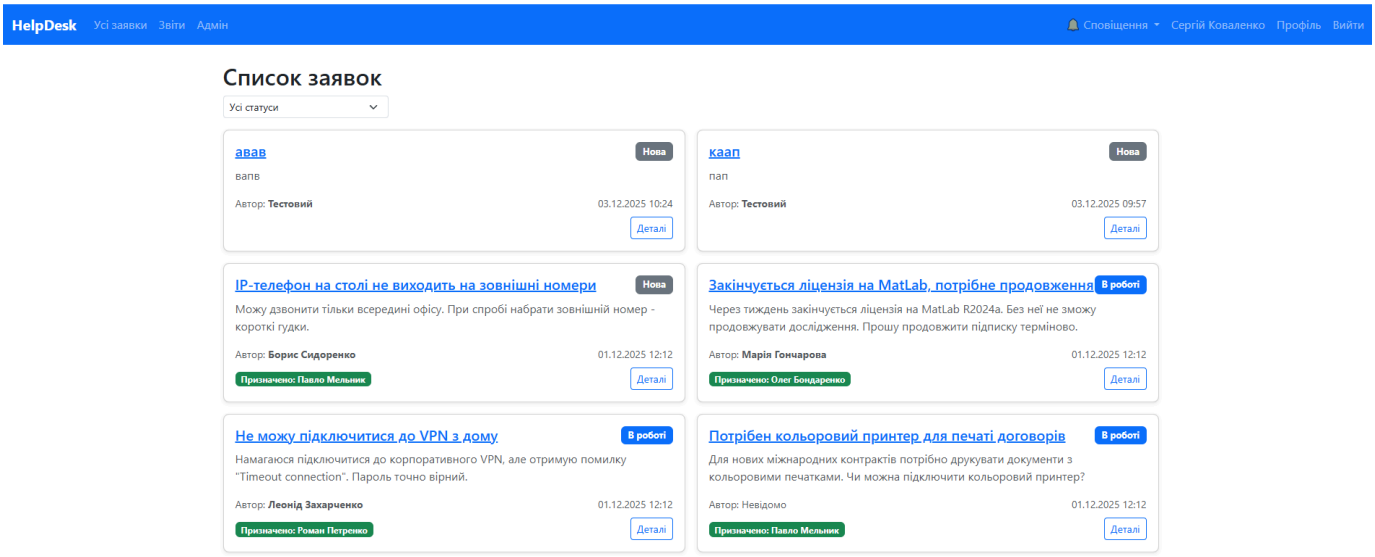


Рисунок 3.27 – Централізований список заявок

Сторінка детального перегляду та життєвого циклу заявки (див. рис. 3.28). На цьому екрані зосереджена вся інформація про конкретне звернення. Інтерфейс чітко розділений на логічні блоки: основні дані (опис, пріоритет, категорія), історія змін статусів, блок коментарів та вкладених файлів. Ключовою особливістю є панель керування, доступна за правами: оператор може додавати коментарі (публічні для клієнта або внутрішні для колег), змінювати статус у межах дозволеного workflow; адміністратор – перепризначати заявку, корегувати пріоритет або SLA. Вся комунікація відбувається в реальному часі без необхідності перезавантаження сторінки завдяки технології AJAX.

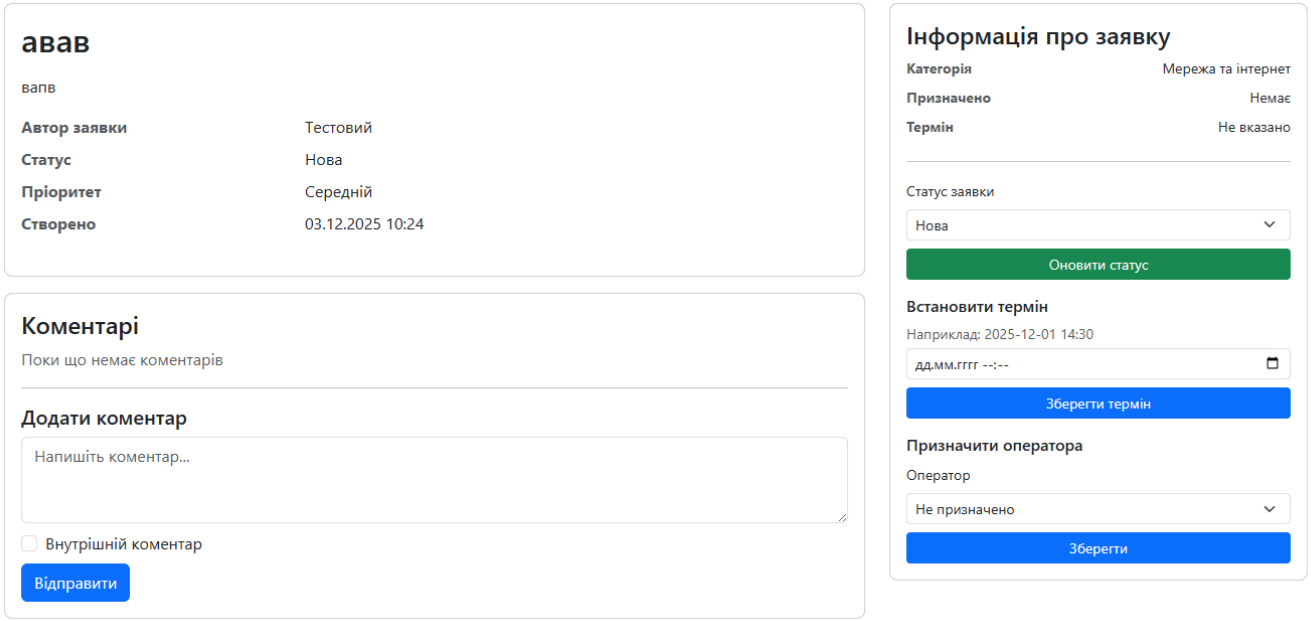


Рисунок 3.28 – Сторінка перегляду життєвого циклу заявки

Інтуїтивна форма створення заявки (див. рис. 3.29). Для клієнта процес ініціювання запиту на допомогу максимально спрощений. Форма містить мінімум обов'язкових полів: заголовок, детальний опис проблеми та вибір категорії. Додатково є можливість прикріпити файли (скріншоти, документи). Валідація даних відбувається як на стороні клієнта (за допомогою Bootstrap), так і на сервері, що гарантує коректність та повноту інформації. Після успішного створення система автоматично перенаправляє клієнта на сторінку нової заявки з її унікальним номером для подальшого відстеження.

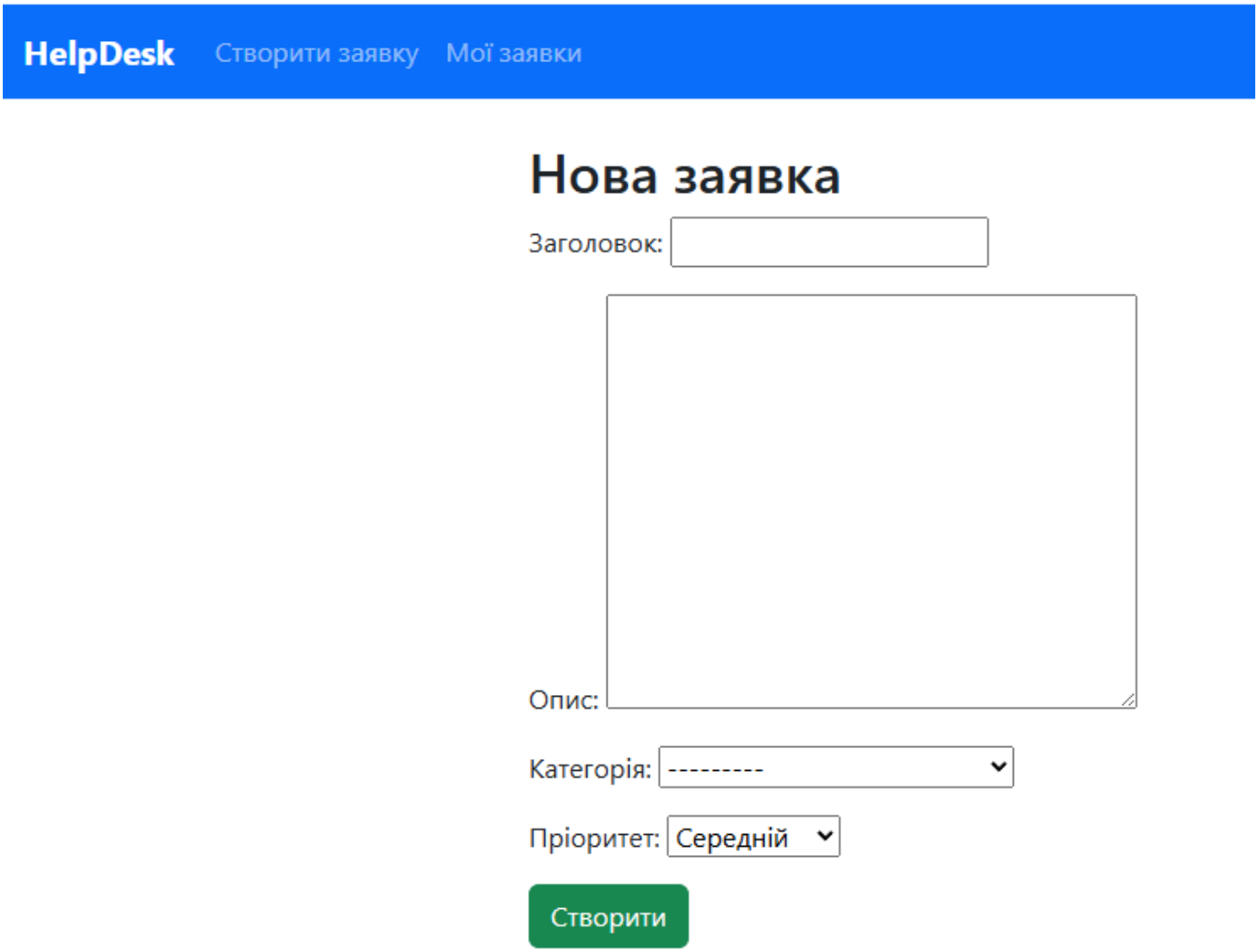


Рисунок 3.29 – Форма створення заявки

Панель управління та аналітики для адміністратора (див. рис. 3.30, 3.31). Цей інтерфейс призначений для керування системою в цілому. Він включає: модуль управління користувачами (реєстрація, зміна ролей, блокування), налаштування довідників (статуси, категорії, пріоритети), а також розділ звітності. Останній надає дані у вигляді інтерактивних графіків та зведених таблиць, що візуалізують ключові метрики: загальну кількість заявок, середній час вирішення, навантаження на операторів, дотримання SLA. Можливість експорту будь-якого звіту в Excel забезпечує глибший аналіз поза системою.

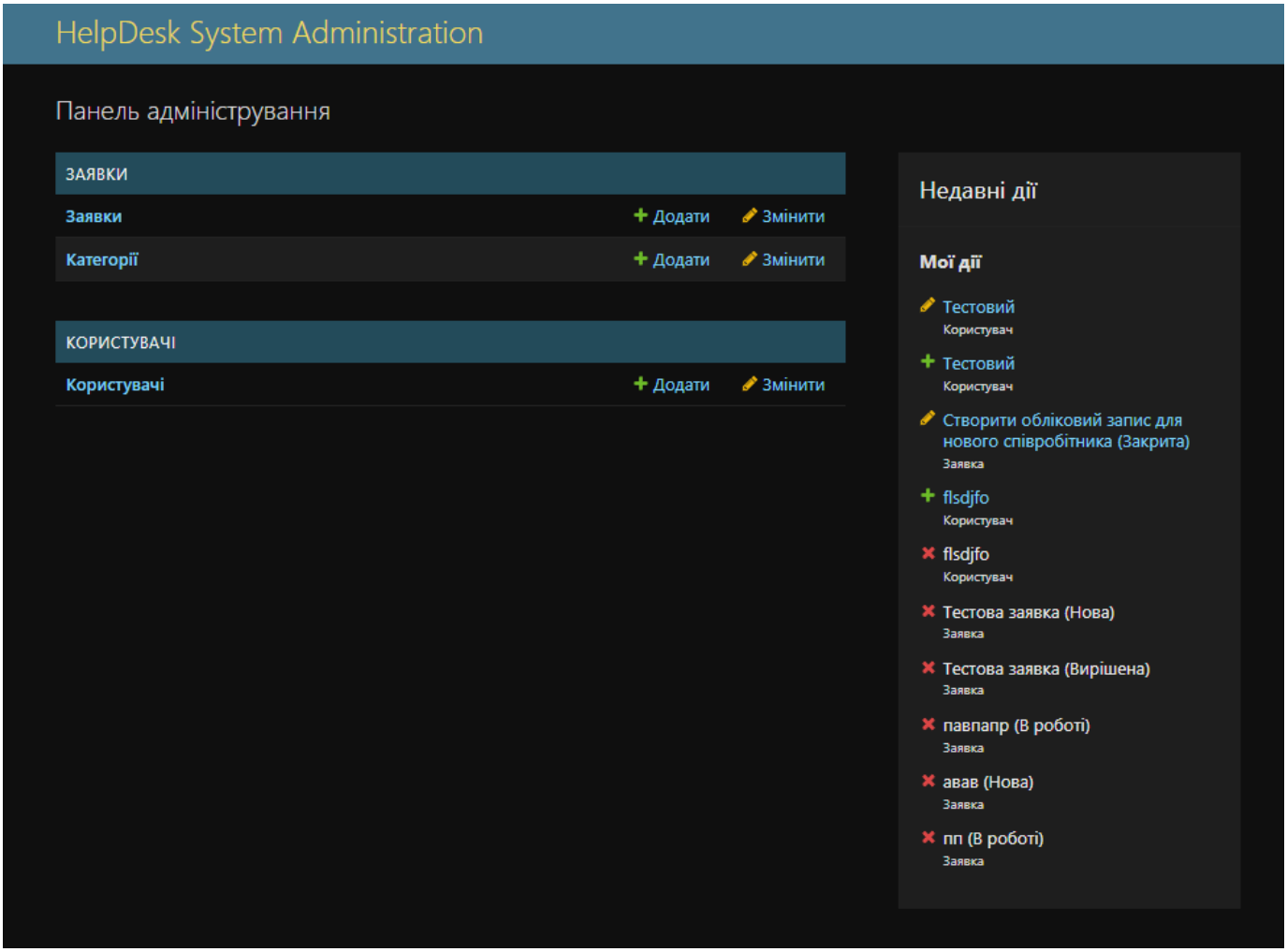


Рисунок 3.30 – Панель управління для адміністратора

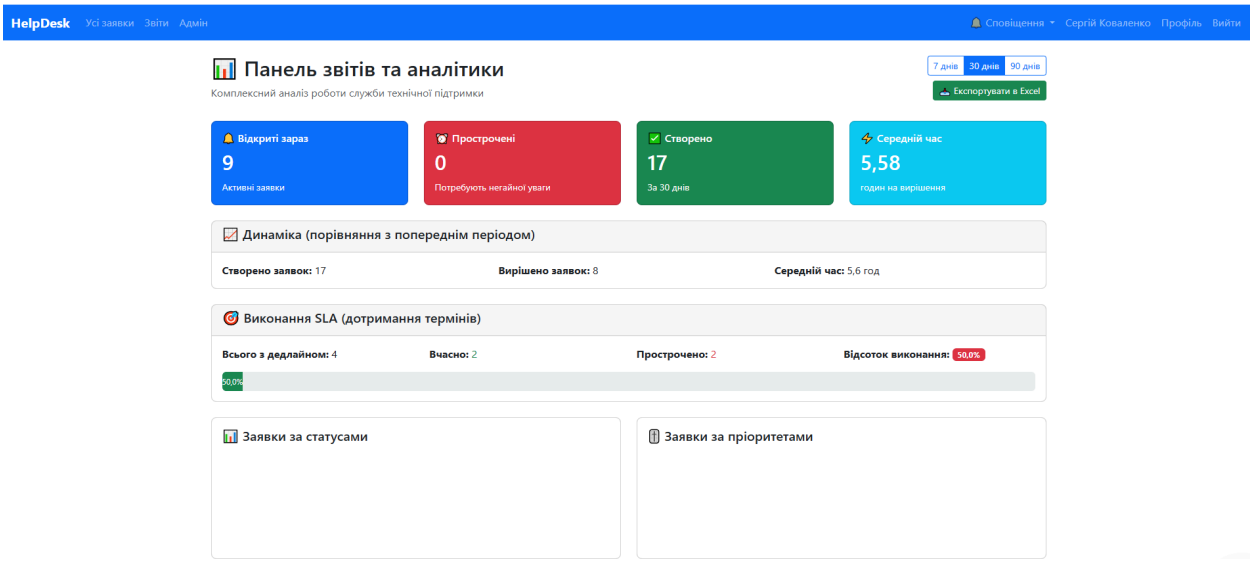


Рисунок 3.31 – Панель аналітики для адміністратора

ВИСНОВКИ

Метою роботи була розробка веб-системи для автоматизації служби технічної підтримки для малого та середнього бізнесу, і ця мета повністю досягнута шляхом вирішення всіх поставлених завдань.

Проведений аналіз ринку та потреб цільової аудиторії виявив нішу для легкого, економічного та самостійно розгортуваного рішення. Архітектура системи спроектована на основі шаблону MVT, що забезпечує чітке розділення відповідальностей і легкість підтримки. Успішно реалізовано всі ключові модулі: управління заявками з повним життєвим циклом, систему користувачів і ролей, механізм in-app сповіщень та модуль аналітики з експортом звітів. Кодова база супроводжується тестами, що є стандартом для якісних проектів, а система забезпечує високий рівень безпеки завдяки використанню актуальної LTS-версії Django 4.2 та дотриманню її рекомендацій.

Розроблений програмний продукт повністю відповідає сформованим функціональним та нефункціональним вимогам. Реалізовано централізоване управління заявками, комунікацію через коментарі, контроль SLA та продуктивності. Рольова модель (клієнт, оператор, адміністратор) гарантує принцип найменших привілеїв. Адаптивний інтерфейс на Bootstrap 5 забезпечує зручну роботу на будь-яких пристроях, а обґрунтований вибір технологічного стеку (Python/Django, PostgreSQL, Celery) дозволив створити продуктивну, масштабовану та безпечну систему.

Практична цінність роботи полягає в створенні готового до впровадження open-source рішення, яке дозволяє невеликим компаніям структурувати роботу підтримки без значних витрат. Система покриває понад 90% типових потреб, що підтверджується порівнянням з існуючими проектами на кшталт django-helpdesk[17].

Наукова цінність проявляється у комплексному підході до проектування та реалізації інформаційної системи: від аналізу бізнес-процесів і ринку через

обґрунтування архітектурних і технологічних рішень до реалізації з урахуванням питань безпеки, тестування та сучасного UX/UI.

Система має потенціал для подальшого розвитку. Перспективними напрямами є додавання нових каналів комунікації, таких як Telegram-бот або інтеграція з електронною поштою, розширення аналітики за допомогою інтерактивних дашбордів, впровадження бази знань для клієнтів та розробка підтримки кросплатформності системи.

Підсумовуючи, в ході дипломної роботи було створено сучасну, ефективну та доступну інформаційну систему. Вона успішно вирішує проблему автоматизації служби техпідтримки в умовах обмежених ресурсів малого та середнього бізнесу, тим самим підтверджуючи актуальність теми та доцільність обраного підходу.

ПЕРЕЛІК ПОСИЛАНЬ

1. ERP vs CRM: ключові відмінності та особливості систем автоматизації бізнесу [Електронний ресурс]. – URL: https://gudhub.com.ua/blog/osvitni-tehnologiji/erp-vs-crm-klyuchovi-vidminnosti-ta-osoblyvosti-system/
2. Фелдрой Д., Рой О. Two Scoops of Django: Best Practices for Django 5 and 6 / Д. Фелдрой, О. Рой. – Атланта: Two Scoops Press, 2024. – 458 с.
3. CRM проти ERP: ключові відмінності, переваги й вибір правильного рішення [Електронний ресурс]. – URL:https://nethunt.ua/blog/crm-proti-erp/
4. CRM-системи з аналітикою [Електронний ресурс]. – URL:https://shelfy.com.ua/categories/crm-systems/analityka/
5. UGLA ERP: Нові горизонти кейтерингу для ресторанів [Електронний ресурс]. – URL:https://ugla.ua/blog/ugla-erp-novi-goryzonty-kejteryngu-dlya-restoraniv-2/
6. ERP/CRM для логістики та транспорту [Електронний ресурс]. – URL:https://avada-media.ua/blog/erp-crm-for-logistics-and-transportation/
7. Посібник з автоматизації служби підтримки клієнтів [Електронний ресурс]. – URL:https://www.chatcompose.com/uk/support-automation.html
8. Утримання клієнтів завдяки функції автоматизації CRM: переваги та недоліки [Електронний ресурс]. – URL:https://sitniks.ua/blog_post/utrymannya-kliyentiv-zavdyaky-funkcziyi-avtomatyzacziyi-crm-perevagy-ta-nedoliky/
9. Requirements Gathering: The Process, Best Tools & More [Електронний ресурс]. – URL:https://www.projectmanager.com/blog/requirements-gathering-guide
10. Аналіз існуючих розробок для автоматизації задачі [Електронний ресурс]. – URL:https://studfile.net/preview/7069683/page:6/

11. PostgreSQL чи MySQL. Як обрати оптимальну базу даних для вашого проєкту [Електронний ресурс]. – URL:https://dou.ua/forums/topic/51621/

12. A Guide to Component Driven Development (CDD) [Електронний ресурс]. – URL:https://itnext.io/a-guide-to-component-driven-development-cdd-1516f65d8b55

13. Обзор і аналіз існуючих методів рішення задачі. [Електронний ресурс]. – URL:https://studwood.net/1841083/informatika/obzor_analiz_suschestvuyuschih_metodov_resheniya_zadachi_obosnovanie_vybora_metoda_resheniya_razrabotki_novogo

14. SQL Database Projects extension [Електронний ресурс]. – URL:https://learn.microsoft.com/en-us/sql/tools/visual-studio-code-extensions/sql-database-projects/sql-database-projects-extension?view=sql-server-ver17

15. Bootstrap – які можливості він відкриває? [Електронний ресурс]. – URL:https://desigo.eu/uk/bootstrap-%D1%8F%D0%BA%D1%96-%D0%BC%D0%BE%D0%B6%D0%BB%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D1%96-%D0%B2%D1%96%D0%BD-%D0%B2%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D0%B2%D0%B0%D1%94/

16. Що таке Патерн? [Електронний ресурс]. – URL:https://refactoring.guru/uk/design-patterns/what-is-pattern

17. Advanced .NET programming documentation [Електронний ресурс]. – URL:https://learn.microsoft.com/en-us/dotnet/navigate/advanced-programming/

18. django-helpdesk’s documentation [Електронний ресурс]. – URL:https://django-helpdesk.readthedocs.io/

19. Які існують тенденції та інновації у сфері SaaS? [Електронний ресурс]. – URL:https://payproglobal.com/uk/%D0%B2%D1%96%D0%B4%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D1%96/%D1%89%D0%BE-%D1%82%D0%B0%D0%BA%D0%B5-%D1%82%D0%B5%D0%BD%D0%B4%D0%B5%D0%BD%D1%86%D1%96%D1%97-%D1%82%D0%B0-%D1%96%D0%BD%D0%BD%D0%BE%D0%B2%D0%B0%D1%86%D1%96%D1%97-saas/

20. Help Desk Software Market Size and Share Forecast Outlook 2025 to 2035 [Електронний ресурс]. – URL:https://www.futuremarketinsights.com/reports/help-desk-software-market

21. Customer Service Software Market Size and Share Forecast Outlook 2025 to 2035 [Електронний ресурс]. – URL:https://www.futuremarketinsights.com/reports/customer-service-software-market

22. Automation Trends 2025: Top Takeaways from the Annual Global State of IT Automation Report [Електронний ресурс]. – URL:https://www.stonebranch.com/blog/it-automation-trends

23. Автоматизація та IT у 2025 році: ключові тренди, які змінять бізнес [Електронний ресурс]. – URL:https://inbase.com.ua/trendy-avtomatyzatsiyi-2025/

24. Автоматизація бізнесу у 2025 році [Електронний ресурс]. – URL:https://www.erpforum.com.ua/blog/doslidzhennia-10/avtomatizatsiia-biznesu-u-2025-rotsi-298

25. Технологічні тренди 2025: що формує автоматизацію бізнесу сьогодні [Електронний ресурс]. – URL:https://todo.ltd/blog/tehnologichni-trendy-2025/

26. Що таке MVP в IT: ваш шлях до успішного продукту з мінімальними ризиками [Електронний ресурс]. – URL:https://fnx.com.ua/ua/articles/publications/chto-takoe-mvp-v-it-polnoe-rukovodstvo-po-mynymal-no-zhyznesposobnomu-produktu

27. Просто про архітектуру програмного забезпечення або основи, які заощадять місяці роботи [Електронний ресурс]. – URL:https://brander.ua/blog/prosto-pro-arkhitekturu-prohramnoho-zabezpechennya

28. Як масштабувати архітектуру під зростання продукту: від MVP до highload [Електронний ресурс]. – URL:https://itc.ua/ua/articles/yak-masshtabuvaty-arhitekturu-pid-zrostannya-produktu-vid-mvp-do-highload/

29. Zendesk – що це? Чесний огляд платформи, мінуси і вигідна альтернатива [Електронний ресурс]. – URL:https://helpcrunch.com/blog/uk/analogy-zendesk/

30. Jira Project Management: The Ultimate 2025 Guide [Електронний ресурс]. – URL:https://clickup.com/blog/jira-project-management/?utm_source=google&utm_medium=cpc&utm_campaign=gs_cpc_arlv_nnc_nb_trial_all-devices_troas_lp_x_all-departments_x_competitor&utm_term&utm_content=all-countries_kw-target_text_all-industries_all-features_all-use-cases_jira-whiteboard_phrase%2A&gclid=Cj0KCQiAosrJBhD0ARIsAHebCNrnPCzqm7KdT_iLSI0eD_OkApbl6BTyOqGuk6uxXNMIb3Qok80kg-IaAv_nEALw_wcB

31. Топ альтернативи для управління службами Jira [Електронний ресурс]. – URL:https://www.getguru.com/uk/reference/top-alternatives-to-jira-service-management

32. Як користуватися Freshdesk: Повний посібник [Електронний ресурс]. – URL:https://www.getguru.com/uk/reference/how-to-use-freshdesk-a-comprehensive-guide#:~:text=Freshdesk%20%D0%BF%D0%B5%D1%80%D0%B5%D1%82%D0%B2%D0%BE%D1%80%D1%8E%D1%94%20%D0%B7%D0%B0%D0%BF%D0%B8%D1%82%D0%B8%2C%20%D0%BD%D0%B0%D0%B4%D1%85%D0%BE%D0%B4%D1%8F%D1%87%D0%B8%20%D0%B7,%D0%B2%D0%B8%D1%80%D1%96%D1%88%D0%B5%D0%BD%D0%BD%D1%8F%20%D0%BA%D0%B2%D0%B8%D1%82%D0%BA%D1%96%D0%B2%20%D0%BD%D0%B0%20%D1%80%D1%96%D0%B7%D0%BD%D0%B8%D1%85%20%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%85.

33. Freshdesk vs Zendesk: Функції, ціни, переваги та недоліки [Електронний ресурс]. – URL:https://cloudfresh.com/ua/cloud-blog/freshdesk-vs-zendesk/

34. Безпечні багатокористувацькі SaaS на Django [Електронний ресурс]. – URL:https://alexsmokinof.lviv.ua/bezpechni-bagatokorystuvaczki-saas-na-django/

35. IT-інфраструктура для бізнесу: як побудувати надійну та безпечну систему [Електронний ресурс]. – URL:https://itez.com.ua/blog/it-infrastructure-for-business-secure-setup.html

36. Безпека прикладних програм [Електронний ресурс]. – URL:https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B0_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B8%D1%85_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC

37. Ваш перший Django проект! [Електронний ресурс]. – URL:https://tutorial.djangogirls.org/uk/django_start_project/

38. Django Підручник [Електронний ресурс]. – URL:https://w3schoolsua.github.io/django/index.html

39. Як створити свій сайт на шаблоні html та вивести його в інтернет [Електронний ресурс]. – URL:https://cityhost.ua/uk/blog/yak-stvoriti-sviy-sayt-na-shabloni-html-ta-vivesti-yogo-v-internet-instrukciya-dlya-pochatkivciv.html

40. Підручник з Python [Електронний ресурс]. – URL:https://docs.python.org/uk/3/tutorial/index.html

41. PostgreSQL Початок роботи [Електронний ресурс]. – URL:https://w3schoolsua.github.io/postgresql/postgresql_getstarted.html

ДОДАТОК А. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

Файл apps/core/utils/access.py:

```
from functools import wraps
from django.shortcuts import redirect
from django.core.exceptions import PermissionDenied

class RoleRequiredMixin:
    required_role = None

    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated:
            return redirect('users:login')
        if request.user.role != self.required_role and request.user.role != 'admin':
            raise PermissionDenied('Недостатньо прав')
        return super().dispatch(request, *args, **kwargs)

class OperatorRequiredMixin(RoleRequiredMixin):
    required_role = 'operator'

class AdminRequiredMixin(RoleRequiredMixin):
    required_role = 'admin'


def role_required(required_role):
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            if not request.user.is_authenticated:
                return redirect('users:login')
            if request.user.role != required_role and request.user.role != 'admin':
                raise PermissionDenied('Недостатньо прав')
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator
```

Файл tickets/models.py:

```
class Ticket(AbstractBaseModel):
    PRIORITY_CHOICES = [
        ('low', 'Низький'),
        ('medium', 'Середній'),
        ('high', 'Високий'),
        ('critical', 'Критичний'),
    ]

    title = models.CharField(_('заголовок'), max_length=200)
    description = models.TextField(_('опис'))
    status = models.ForeignKey(Status, verbose_name=_('статус'),
on_delete=models.PROTECT)
    priority = models.CharField(_('пріоритет'), max_length=20,
choices=PRIORITY_CHOICES, default='medium')
    category = models.ForeignKey(Category, verbose_name=_('категорія'), null=True,
blank=True, on_delete=models.SET_NULL)
    created_by = models.ForeignKey(settings.AUTH_USER_MODEL,
verbose_name=_('створив'), null=True, blank=True, on_delete=models.SET_NULL,
related_name='created_tickets')
    assigned_to = models.ForeignKey(settings.AUTH_USER_MODEL,
verbose_name=_('призначено'), null=True, blank=True, on_delete=models.SET_NULL,
related_name='assigned_tickets')
    assigned_by = models.ForeignKey(settings.AUTH_USER_MODEL,
verbose_name=_('призначив'), null=True, blank=True, on_delete=models.SET_NULL,
related_name='assigned_by_tickets', help_text=_('Адміністратор, який призначив
заявку'))
    deadline = models.DateTimeField(_('термін'), null=True, blank=True)
    resolved_at = models.DateTimeField(_('виправлено'), null=True, blank=True,
help_text=_('Час коли тикет був помічений як вирішений'))

    def get_absolute_url(self):
        return reverse('tickets:detail', kwargs={'pk': self.pk})

    def is_overdue(self):
```

```
        return self.deadline and timezone.now() > self.deadline
```

Файл apps/reports/views.py:

```
class ReportDashboardView(LoginRequiredMixin, TemplateView):
    template_name = 'reports/dashboard.html'

    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated or request.user.role != 'admin':
            from django.http import HttpResponseForbidden
            return HttpResponseForbidden('Forbidden')
        return super().dispatch(request, *args, **kwargs)

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        rg = ReportGenerator()
        period = self.request.GET.get('period', '30')
        try:
            days = int(period)
        except:
            days = 30
        end = timezone.now()
        start = end - timedelta(days=days)
        previous_end = start
        previous_start = previous_end - timedelta(days=days)
        ctx['metrics'] = rg.get_ticket_metrics(start, end)
        ctx['sla_compliance'] = rg.get_sla_compliance(start, end)
        ctx['operator_performance'] = rg.get_operator_performance(start, end)
        ctx['category_analysis'] = rg.get_category_analysis(start, end)
        ctx['priority_distribution'] = rg.get_priority_distribution(start, end)
        ctx['status_distribution'] = rg.get_status_distribution(start, end)
        ctx['period_comparison'] = rg.compare_periods(start, end, previous_start,
previous_end)
        ctx['closed_tickets_archive'] = rg.get_closed_tickets_archive(start, end, limit=50)
        ctx['start_date'] = start
        ctx['end_date'] = end
```

```
        ctx['period_days'] = days
        ctx['by_status'] = ctx['status_distribution']
        ctx['by_priority'] = [{'priority': p['priority'], 'count': p['count']} for p in
ctx['priority_distribution']]
        from apps.tickets.models import Ticket
        ctx['current_open_tickets'] =
Ticket.objects.filter(resolved_at__isnull=True).exclude(status__slug='closed').count()
        ctx['current_overdue'] = Ticket.objects.filter(deadline__lt=timezone.now(),
resolved_at__isnull=True).exclude(status__slug='closed').count()
        return ctx
```

Файл apps/tickets/tasks.py:

```
@shared_task
def check_overdue_deadlines():
    """Check for tickets with overdue deadlines and notify admins and assigned operators.
    Runs every hour to check for tickets that have passed their deadline.
    Sends notifications only once per ticket to avoid spam.
    """
    now = timezone.now()
    tickets = Ticket.objects.filter(deadline__isnull=False, deadline__lt=now,
resolved_at__isnull=True)
    notification_service = NotificationService()
    notified_count = 0

    for ticket in tickets:
        from apps.notifications.models import Notification
        if ticket.assigned_to:
            existing = Notification.objects.filter(
                user=ticket.assigned_to,
                title='Термін заявки прострочено',
                target_url=ticket.get_absolute_url()
            ).exists()
            if not existing:
                time_overdue = now - ticket.deadline
                hours_overdue = int(time_overdue.total_seconds() / 3600)
```

```
            days_overdue = int(time_overdue.total_seconds() / 86400)
            if days_overdue > 0:
                message = f'Заявка "{ticket.title}" прострочена на {days_overdue} дн.'
            else:
                message = f'Заявка "{ticket.title}" прострочена на {hours_overdue} год.'
            notification_service.create_in_app_notification(
                ticket.assigned_to,
                'Термін заявки прострочено',
                message,
                url=ticket.get_absolute_url()
            )
            notified_count += 1

        admins = CustomUser.objects.filter(role='admin', is_active=True)
        for admin in admins:
            existing = Notification.objects.filter(
                user=admin,
                title='Термін заявки прострочено',
                target_url=ticket.get_absolute_url()
            ).exists()
            if not existing:
                time_overdue = now - ticket.deadline
                hours_overdue = int(time_overdue.total_seconds() / 3600)
                days_overdue = int(time_overdue.total_seconds() / 86400)
                if days_overdue > 0:
                    message = f'Заявка "{ticket.title}" прострочена на {days_overdue} дн.'
                else:
                    message = f'Заявка "{ticket.title}" прострочена на {hours_overdue} год.'
                notification_service.create_in_app_notification(
                    admin,
                    'Термін заявки прострочено',
                    message,
                    url=ticket.get_absolute_url()
                )
                notified_count += 1
    return {'checked': tickets.count(), 'notifications_sent': notified_count}
```

ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ(ПРЕЗЕНТАЦІЯ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Комп'ютерних наук

Дипломна робота

на ступінь вищої освіти магістр
із спеціальності F3 Комп'ютерні науки

Інформаційна система прийому та аналізу заявок технічної підтримки (Python/Django + Bootstrap + PostgreSQL)

Виконав: студент 6 курсу, групи КНДМ-62
Бугайченко Андрій Дмитрович
Керівник: професор кафедри Комп'ютерних наук
д.ф.-м.н., професор Шикула О.М.

Київ - 2025

1

Актуальність роботи

У сучасному бізнес-середовищі ефективна технічна підтримка перетворилася з допоміжної функції на ключовий чинник успіху та лояльності клієнтів, особливо для малих і середніх підприємств. Однак понад 60% таких компаній не використовують спеціалізовані системи, обмежуючись електронною поштою та електронними таблицями, що неминуче призводить до втрати запитів, хаосу в комунікації та неможливості аналізу ефективності. На ринку існують потужні корпоративні рішення, такі як Zendesk або Jira Service Desk, але вони часто є надмірно складними, дорогими та орієнтованими на великі організації з виділеними IT-відділами. Це створює значну прогалину для доступного та спеціалізованого інструменту.

Актуальність даної розробки полягає саме у створенні open-source (з відкритим вихідним кодом) системи, яка поєднує необхідний функціонал для повного життєвого циклу обробки заявки, інтуїтивно зрозумілий інтерфейс та економічну ефективність, заповнюючи таким чином виявлену ринкову нішу для потреб малого та середнього бізнесу.

2

Мета, предмет та об’єкт дослідження

Мета роботи: Розробити веб-орієнтовану інформаційну систему для автоматизації процесів прийому, обробки, контролю та аналізу заявок технічної підтримки з використанням Python/Django, Bootstrap та PostgreSQL.

Об’єкт дослідження: Процес організації та автоматизації роботи служби технічної підтримки в невеликих та середніх компаніях.

Предмет дослідження: Інформаційна система прийому та аналізу заявок технічної підтримки.

Огляд існуючих систем (1/4) – Zendesk

Категорія: Все-в-одному (All-in-one) рішення для обслуговування клієнтів.

Ключові переваги: Комплексна екосистема (email, чат, соцмережі, голосові дзвінки), потужна аналітика та звітність, висока надійність та інтеграції.

Недоліки для систем малого бізнесу(СБМ): Дуже висока вартість (від $49/агент/місяць), складність початкового налаштування, надмірний функціонал для малих команд.

Огляд існуючих систем (2/4) – Jira Service Desk

Категорія: Спеціалізоване рішення для IT-команд та розробки.

Ключові переваги: Глибока інтеграція з Atlassian Stack (Jira Software, Confluence), надзвичайна гнучкість налаштувань workflow(можливість дуже детально та вільно налаштовувати кроки обробки заявки) та автоматизації, ідеально для зв'язку підтримки з розробниками.

Недоліки для СМБ: Надмірно складний для нетехнічних користувачів, висока "ціна входу" в екосистему(додаткові витрати), перш за все орієнтований на процеси розробки ПЗ, а не для загальних потреб техпідтримки будь-якого бізнесу.

Огляд існуючих систем (3/4) – Freshdesk

Категорія: Конкурент Zendesk з орієнтацією на доступність.

Ключові переваги: Сучасний та зручний інтерфейс, швидкий старт, доступна цінова політика (є безкоштовний тариф), вбудований AI-помічник (Freddy AI).

Недоліки для СМБ: Менш потужна аналітика порівняно з лідерами,обмежені можливості глибокої кастомізації, залишається універсальним коробковим рішенням (не можна переробити під унікальний бізнес-процес конкретної компанії).

Огляд існуючих систем (4/4) – Висновки та ринкова ніша

Проведений порівняльний аналіз лідерів ринку — Zendesk, Jira Service Desk та Freshdesk — виявляє спільну для них проблему в контексті потреб малого та середнього бізнесу. Навіть найбільш доступні з цих рішень залишаються універсальними коробковими продуктами, спроектованими для максимально широкої аудиторії. Їхня потужність обертається суттєвими недоліками для невеликих команд: надмірна функціональність перевантажує інтерфейс та ускладнює навчання, складна ініційна налаштувальність вимагає додаткових ресурсів, а вартість масштабування швидко стає непропорційно високою. Це створює чітку ринкову прогалину — брак спеціалізованого, але спрощеного рішення, яке реалізує не «все можливе», а саме необхідне для ефективної роботи техпідтримки.

Таким чином, існує гостра потреба в системі, яка пропонує оптимальне співвідношення «функціональність – простота – вартість», будучи зрозумілою, швидкою у впровадженні та економічно вигідною саме для малих і середніх компаній. Ця потреба й обумовлює актуальність розробки власного спеціалізованого рішення.

Обґрунтування вибору технологічного стеку

Бекенд: Python 3.10+ & Django 4.2. Обґрунтування: висока швидкість розробки, має великий набір вбудованих, готових до використання компонентів для вирішення типових завдань веб-розробки - ORM, адмін-панель, безпека, чиста архітектура MVT, велика спільнота.

База даних: PostgreSQL. Обґрунтування: потужна, ACID-сумісна, відмінна інтеграція з Django ORM, підтримка складних типів даних.

Фронтенд: Bootstrap 5. Обґрунтування: найпопулярніший CSS-фреймворк для швидкої розробки адаптивних та сучасних інтерфейсів з готовими компонентами.

Додатково: Celery + Redis для асинхронних завдань (сповіщення, звіти).

Проектування архітектури системи

Основою розробки обрано архітектурний шаблон MVT (Model-View-Template), який реалізується фреймворком Django. Ця модель забезпечує чітке розділення відповідальності між компонентами: Model відповідає за структуру даних та бізнес-логіку, View виступає обробником запитів та контролером, а Template формує представлення для користувача. Такий підхід забезпечує низьку зв'язаність компонентів, що робить систему легко тестованою та підтримуваною. Проект побудовано на принципі модульності, де кожна функціональна область інкапсульована в окремий додаток: apps.core для спільних утиліт, apps.users для управління користувачами, apps.tickets як бізнес-ядро системи, apps.notifications для механізму сповіщень та apps.reports для аналітики. Подібна організація коду покращує його структуру, дозволяє повторно використовувати компоненти, сприяє паралельній розробці та зосередженому тестуванню кожного модуля.

Проектування бази даних

Основа системи — реляційна база даних, спроектована з урахуванням нормалізації. Її ядро складають головні сутності: tickets_ticket (заявки), users_customuser (користувачі), tickets_comment (коментарі) та довідники статусів і категорій. Між сутностями існують чіткі зв'язки через зовнішні ключі з правилами для збереження цілісності даних. Для підвищення продуктивності створено індекси для полів, що активно використовуються у пошуку та фільтрації. Всі основні таблиці містять поля для аудиту (created_at, updated_at). Така структура забезпечує ефективність, надійність та легкість подальшого масштабування системи.

Реалізація ролей користувачів

1. Клієнт (End-User):
   a. Може: Створювати заявки, переглядати свої, додавати коментарі/файли.
   b. Не може: Бачити заявки інших або втручатися в роботу операторів та адміністраторів.
2. Оператор (Agent/Technician):
   a. Може: Переглядати призначені заявки, змінювати статуси, вести комунікацію, додавати внутрішні нотатки.
   b. Відповідає за вирішення в рамках SLA (дедлайни).
3. Адміністратор (Admin):
   a. Може: Повний контроль над системою. Управління користувачами, категоріями, SLA, генерація звітів.
   b. Забезпечує роботу та розвиток системи.

Розробка ключового функціоналу (Backend)

У бекенді реалізовано повний життєвий цикл заявки з автоматичними переходами між статусами. Бізнес-логіка автоматизована за допомогою сигналів Django, які, наприклад, автоматично змінюють статус на "В роботі" при відповіді оператора та створюють сповіщення. Для контролю якості обслуговування інтегрована система SLA: фонова задача на Celery регулярно перевіряє дедлайни та сповіщає про прострочені заявки. Для адміністратора розроблений модуль аналітики, що дозволяє генерувати детальні звіти у Excel з ключовими метриками ефективності роботи підтримки.

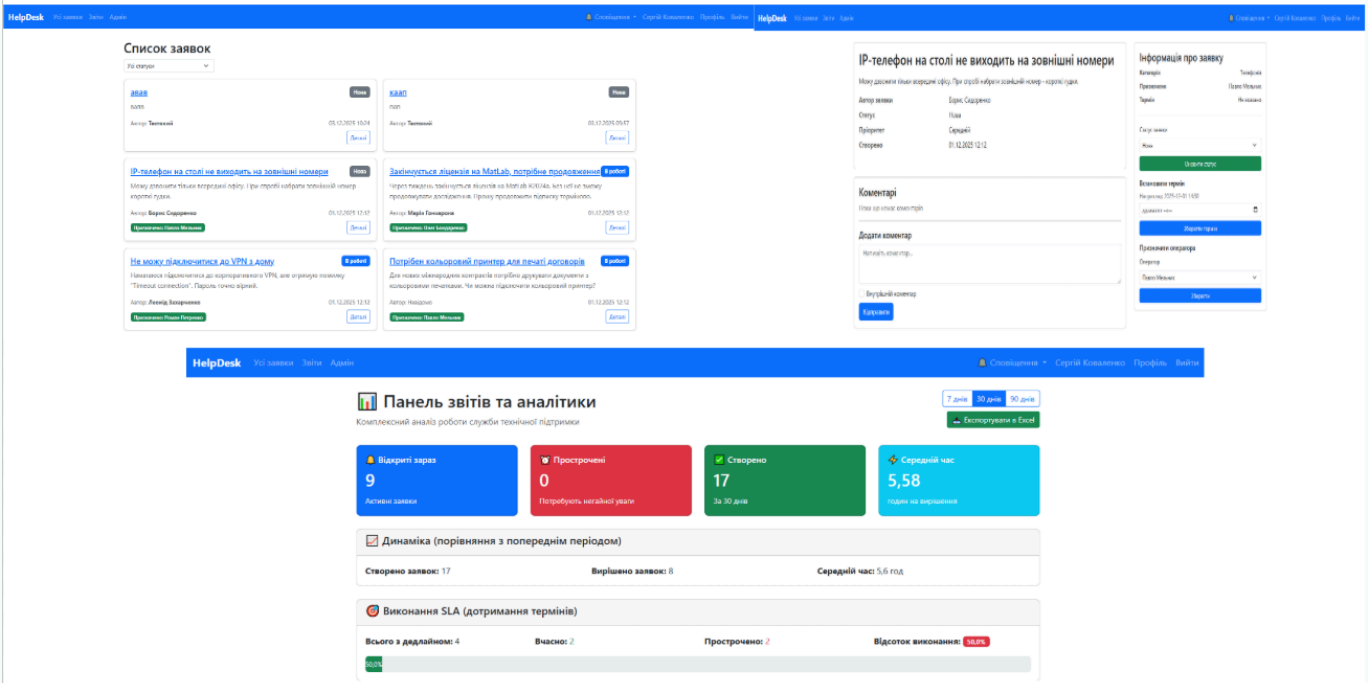
Розробка інтерфейсу (Frontend)

Клієнтська частина побудована на Bootstrap 5, що забезпечує сучасний та повністю адаптивний інтерфейс для роботи з будь-яких пристроїв. Навігація та функціонал інтуїтивно змінюються залежно від ролі: клієнти отримують просту форму створення запиту, оператори — зручну панель управління з фільтрами, а адміністратори — комплексний дашборд. Користувацький досвід покращено за рахунок динамічного оновлення контенту через AJAX без перезавантаження сторінки, що реалізовано для додавання коментарів, оновлення сповіщень та роботи зі списками.

Забезпечення якості та безпеки

Надійність системи забезпечена комплексом модульних та інтеграційних тестів, що покривають ключові сценарії роботи. Безпека реалізована на основі рекомендацій OWASP: вбудовані механізми Django захищають від CSRF, XSS та SQL-ін'єкцій, а валідація даних виконується на всіх рівнях. Для робочого середовища налаштована сувора конфігурація: активовано HTTPS, безпечні заголовки та вимкнено режим налагодження, що забезпечує захист від зовнішніх загроз.

Демонстрація інтерфейсу програми



Висновки

Мета досягнута: Розроблено повноцінну веб-систему автоматизації техпідтримки для СМБ.

Наукова цінність: Виконано комплексний аналіз ринку, бізнес-процесів та архітектурних рішень для створення спеціалізованого продукту.

Практична цінність: Система є готовим до впровадження open-source рішенням, яке заповнює виявлену ринкову нішу, поєднуючи необхідний функціонал, простоту та доступність.

Результат: Створено інструмент, що дозволяє підвищити ефективність служби підтримки, контроль якості та задоволеність клієнтів за рахунок структурованого процесу та аналітики.