Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська кваліфікаційна робота на тему:

WEB-ДОДАТОК ДЛЯ МЕНЕДЖМЕНТУ ЗАДАЧ РІЗНОГО ТИПУ

Виконала: студентка 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Кисилевич А. О.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)
«04» червня 2025 р

Рецензент: к. т. н., доцент каф. АІІТ
Барабан М. В.
(прізвище та ініціали)
«05» червня 2025 р

Допущено до захисту
Зав. кафедри Яровий А. А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д. т. н. Яровий А. А.

"05" травня 2025 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кисилевич Анжеліці Олегівні

1. Тема роботи: WEB-додаток для менеджменту задач різного типу.

Керівник роботи: Ігор АРСЕНЮК, к. т. н., доц..
затверджені наказом вищого навчального закладу від "20" березня 2025 року № 97
2. Термін подання студентом роботи 30 травня 2025 р.
3. Вихідні дані до роботи:
Мінімальна кількість підтримуваних ролей користувачів – 2.
Період відображення аналітики – до 6 місяців.
База даних – реляційна.
Наявність графічного інтерфейсу.
Мова програмування – об'єктно-орієнтована.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки WEB-додатка для менеджменту задач різного типу; проєктування структури та алгоритмів WEB-додатка для менеджменту задач різного типу; обґрунтування вибору інструментів технічної реалізації WEB-додатка для менеджменту задач різного типу; технічна реалізація програмного рішення та тестування механізмів його роботи; висновки; список використаної літератури; додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
– загальна структурна схема компонентів системи;
– діаграми видів діяльності системи;
– діаграма прецедентів з поділом на ролі у системі;

– діаграми класів сервісів;
– ER-діаграми сервісів;
– приклади роботи розробленого програмного забезпечення;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн
1-4	Арсенюк І. Р., к. т. н., доцент		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Приміт
		початок	закінченн	
1	Обґрунтування доцільності розробки WEB-додатка	05.05.25	06.05.25	
2	Проєктування структури та алгоритмів WEB-додатка	07.05.25	09.05.25	
3	Обґрунтування вибору інструментів технічної реалізації	10.05.25	13.05.25	
4	Реалізація програмного рішення	14.05.25	28.05.25	
5	Розробка інструкції з розгортання та використання WEB-додатка	28.05.25	29.05.25	
6	Тестування механізмів роботи WEB-додатка	30.05.25	03.06.25	
7	Оформлення матеріалів до захисту БКР	04.06.25	04.06.25	

Студент _____ Кисилевич А. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Арсенюк І. Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається зі 107 сторінок формату А4, містить 39 рисунків, 1 таблицю та список використаних джерел із 33 найменувань.

У роботі виконано аналіз предметної області та порівняльний огляд сучасних додатків для менеджменту задач. Сформульовано функціональні вимоги до розроблюваного WEB-додатка. Здійснено проєктування структури та логіки роботи додатка шляхом створення структурної схеми, діаграм видів діяльності, прецедентів, UML-діаграм класів і ER-моделей. Обґрунтовано вибір технологічного стеку для реалізації клієнтської та серверної частин. Наведено інструкції щодо розгортання додатку, визначено системні вимоги та виконано успішне тестування основного функціоналу розробленого програмного рішення.

Ключові слова: WEB-додаток, управління задачами, мікросервісна архітектура, Java, Spring, HTML, CSS, Bootstrap.

ABSTRACT

The bachelor's qualification thesis comprises 107 A4-sized pages, includes 39 figures, 1 table and a list of 33 references.

The study presents an analysis of the subject area and a comparative review of modern task management applications. Functional requirements for the developed web application are defined. The structure and logic of the application are designed through the creation of a structural scheme, activity diagrams, use case diagrams, UML class diagrams, and ER models. The choice of technological stack for both the client-side and server-side implementation is justified. Deployment instructions are provided, system requirements are defined, and the core functionality of the developed software solution has been successfully tested.

Keywords: web application, task management, microservice architecture, Java, Spring, HTML, CSS, Bootstrap.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах ефективне управління завданнями набуває особливої важливості. Зі зростанням обсягу проєктів, динамікою бізнес-процесів та поширенням віддаленої роботи, різноманітних задач стає дедалі більше і всі вони потребують більш гнучкого налаштування параметрів та швидкого розподілу. Без належних інструментів планування і контролю існує ризик втрати часу, зниження ефективності та виникнення помилок у роботі.

Розробка WEB-додатка для менеджменту задач різного типу дозволяє підвищити продуктивність праці, спростити планування і контроль виконання завдань, знизити ризики виникнення непорозумінь, а також забезпечити доступ до актуальної інформації в реальному часі. Такі можливості роблять WEB-застосунки затребуваними в командних та індивідуальних проєктах, оскільки вони дозволяють виконувати більше завдань за менший час та автоматизувати процес організації робочого графіка. Таким чином, створення гнучкого WEB-додатка, що оптимізує робочі процеси шляхом керування різними видами задач в єдиній системі, буде розглядатись як актуальне завдання.

Метою дослідження є розширення функціональних можливостей WEB-додатка для менеджменту задач різного типу.

Об’єктом дослідження є процес управління задачами різного типу.

Предметом дослідження є програмні засоби для реалізації управління завданнями різного типу.

Задачі дослідження:

- виконати аналіз предметної області;
- дослідити існуючі програмні рішення;
- сформулювати вимоги до функціональності додатка;
- виконати проєктування структури та алгоритмів додатка;
- обґрунтувати вибір інструментів технічної реалізації;

- реалізувати програмне рішення;
- підготувати інструкції з розгортання та визначити системні вимоги для роботи додатка;
- виконати тестування додатка.

Апробація та публікація. Здійснено апробацію матеріалів бакалаврської кваліфікаційної роботи на міжнародній науково-практичній конференції “Молодь в науці: дослідження, проблеми, перспективи (МН-2025)”. Опубліковано тези доповіді [1]. Подано заявку на реєстрацію авторського права на твір у формі комп'ютерної програми – “WEB-додаток для менеджменту задач різного типу”, номер заявки С202505464 від 10.06.2025.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ WEB-ДОДАТКА ДЛЯ МЕНЕДЖМЕНТУ ЗАДАЧ РІЗНОГО ТИПУ

1.1 Аналіз предметної області

У сучасних умовах цифровізації економіки ефективне управління задачами набуває особливої актуальності. Це питання є ключовим як для IT-команд, що працюють над створенням складних програмних продуктів, так і для команд у сферах освіти, маркетингу, дизайну, наукових досліджень та багатьох інших. Складність організації командної роботи зумовлює потребу в надійних засобах управління, які дозволяють не лише створювати завдання, а й забезпечувати їх повний життєвий цикл: постановка, делегування, контроль, завершення, аналіз. Саме такі системи дозволяють уникнути хаосу в плануванні робочих операцій, знижують ризик втрати інформації та підвищують прозорість і керованість процесів.

Традиційно управління завданнями відбувалося вручну: у таблицях Excel, текстових документах, через email-листування або месенджери. Однак із часом стало зрозуміло, що для масштабованих команд та тривалих проєктів такий підхід є неефективним і надто ресурсоємним. Це зумовило появу спеціалізованих інструментів для менеджменту задач, зокрема WEB-додатків, які забезпечують цілісний підхід до управління проєктами, включаючи можливості комунікації, відстеження прогресу та аналітики даних [2].

Згідно з дослідженням компанії Gartner, понад 70% організацій використовують системи управління задачами у тій чи іншій формі [3]. Найбільш розповсюдженими є Jira, Trello, Asana, ClickUp, Microsoft Planner, Notion тощо. Проте із зростанням вимог до функціональності виникає необхідність створення спеціалізованих рішень, які враховують не лише базові потреби, а й специфічні вимоги, зокрема:

- наявність вбудованої аналітики виконання задач за довгостроковий період;

- зручний механізм взаємодії між членами команди з доступом до контактної інформації;
- можливість пошуку не лише за виконавцями, а й за назвами команд;
- логічний зв’язок між задачами й командами з можливістю посилання на них.

Наявні рішення часто не задовольняють усім цим критеріям одночасно. Наприклад, платформи з розширеною аналітикою, як правило, є складними в налаштуванні та вимагають професійної підготовки. З іншого боку, більш прості інструменти не мають достатніх можливостей для аналізу чи навігації між структурними елементами проєкту. Це створює потребу в новому типі WEB-додатка, орієнтованого на інтуїтивний інтерфейс і водночас функціонально насиченого [4].

Досвід світових практик свідчить про те, що ефективність управління задачами безпосередньо впливає на продуктивність команди та дотримання дедлайнів. Згідно з дослідженнями PMI (Project Management Institute), проєкти, у яких використовуються автоматизовані засоби керування задачами, мають на 28% вищу ймовірність завершення вчасно [5]. Тому інструменти управління задачами вже не є розкішшю, а необхідністю для конкурентоспроможності команди чи компанії.

У цьому контексті доцільним є створення WEB-додатка нового покоління, який поєднує простоту та функціональність, здатність до аналізу і гнучкість використання вбудованих механізмів виконання операцій. Серед ключових функцій – можливість зібрати й переглядати статистику виконаних задач, виявляти перевантажені завданнями команди, оптимізовувати навантаження, формувати якісну звітність для керівництва. Також актуальною є інтеграція з базою контактної інформації співробітників, що полегшує комунікацію без додаткових зовнішніх засобів.

У результаті аналізу предметної області стає зрозумілим, що попит на вдосконалені, порівняно з існуючими, системи управління задачами буде лише зростати. WEB-додаток, який враховує специфіку командної структури,

підтримує розширену аналітику й полегшує комунікацію, має високі перспективи для впровадження в організаціях різного масштабу – від малих стартапів до великих підприємств.

1.2 Огляд та аналіз існуючих аналогів

У сфері управління задачами існує широкий спектр WEB-додатків, які користуються популярністю у різних типах команд. Найбільш поширеними є Jira, Trello, Asana та ClickUp. Кожен із цих інструментів має власну специфіку, функціональність та цільову аудиторію, однак жоден із них не охоплює повністю перелік вимог, необхідних для універсального управління задачами із сучасними очікуваннями користувачів щодо аналітики, комунікації та навігації.

Jira (рис. 1.1) – це потужний інструмент, що розроблений переважно для розробників програмного забезпечення.

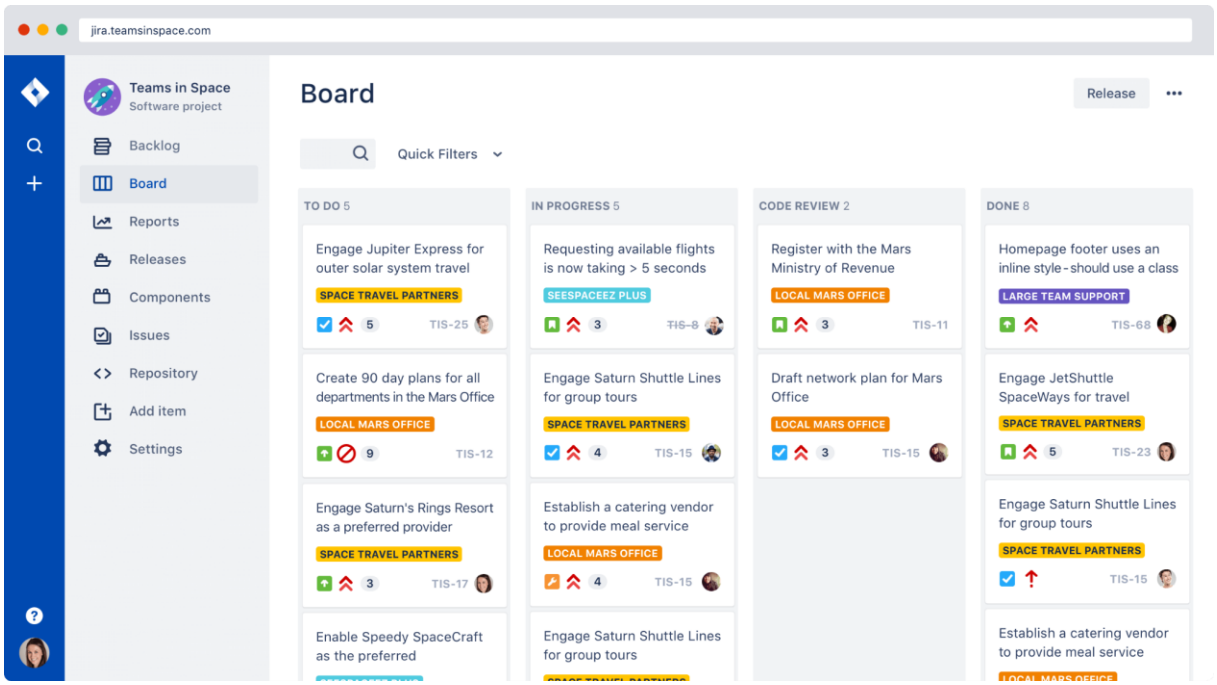

Рисунок 1.1 – Вікно додатка Jira

Головними перевагами Jira є можливість дотримання Agile/Scrum [6] методологій, підтримка спринтів, наявність беклогу та бордів, широкий вибір

інтеграцій з іншими додатками. Однак інтерфейс системи часто вважається надто складним для новачків або нефахівців у сфері IT. Крім того, система не забезпечує зручного доступу до контактної інформації колег – користувач може дізнатись лише ім’я та посаду. Аналітика в Jira обмежена: для перегляду статистики за кількістю задач чи багів за пів року чи менше потрібно встановлювати додаткові модулі або інтегрувати сторонні сервіси. Також відсутній пошук по командах або швидкий перехід на сторінку команди із задачі, що ускладнює управління структурованими командами [7].

Trello (рис. 1.2) – популярний інструмент із візуальним представленням задач у вигляді карток на дошках.

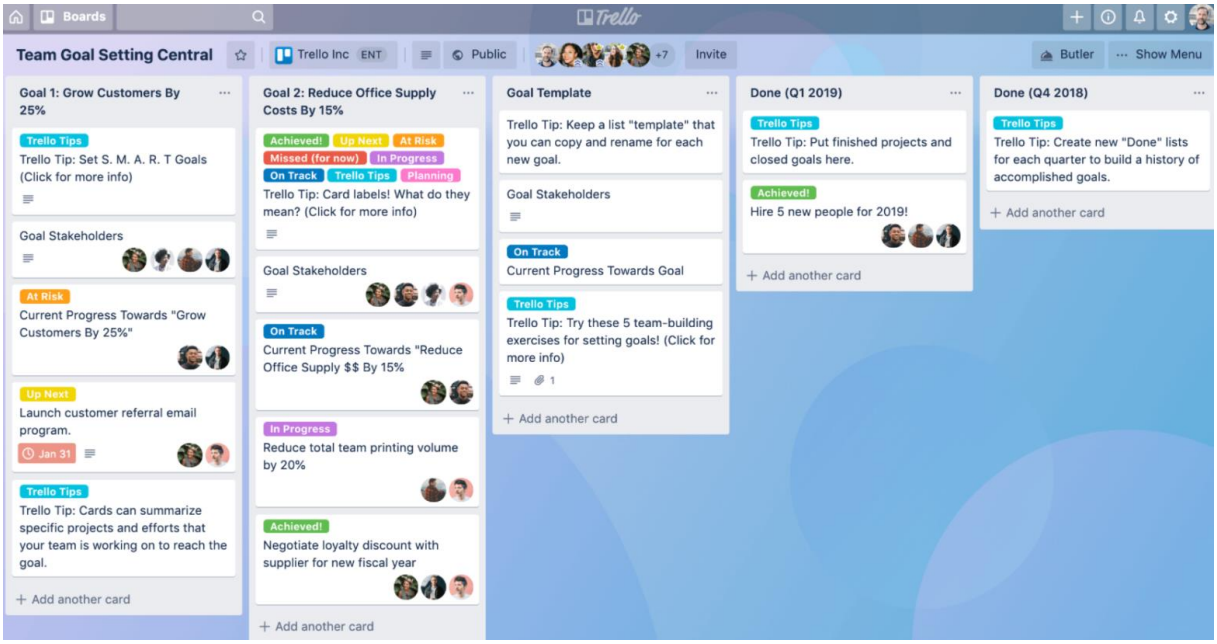

Рисунок 1.2 – Вікно додатка Trello

Переваги Trello – простота використання, інтуїтивно зрозумілий інтерфейс, можливість швидко створювати та переміщувати задачі. Однак функціональність Trello є досить обмеженою, оскільки відсутня вбудована аналітика, яка б дозволяла користувачам бачити статистику виконання задач. Також у Trello немає розширеної підтримки командної структури або можливості переглядати контактні дані членів команди. Пошук реалізований на рівні карток, але немає кластеризації по командам або прямого зв’язку між

задачами й виконавцями з посиланням [8].

Asana (рис. 1.3) – це комплексний інструмент для управління проєктами з широкими можливостями відстеження прогресу, створення залежностей між задачами, подій у календарі.

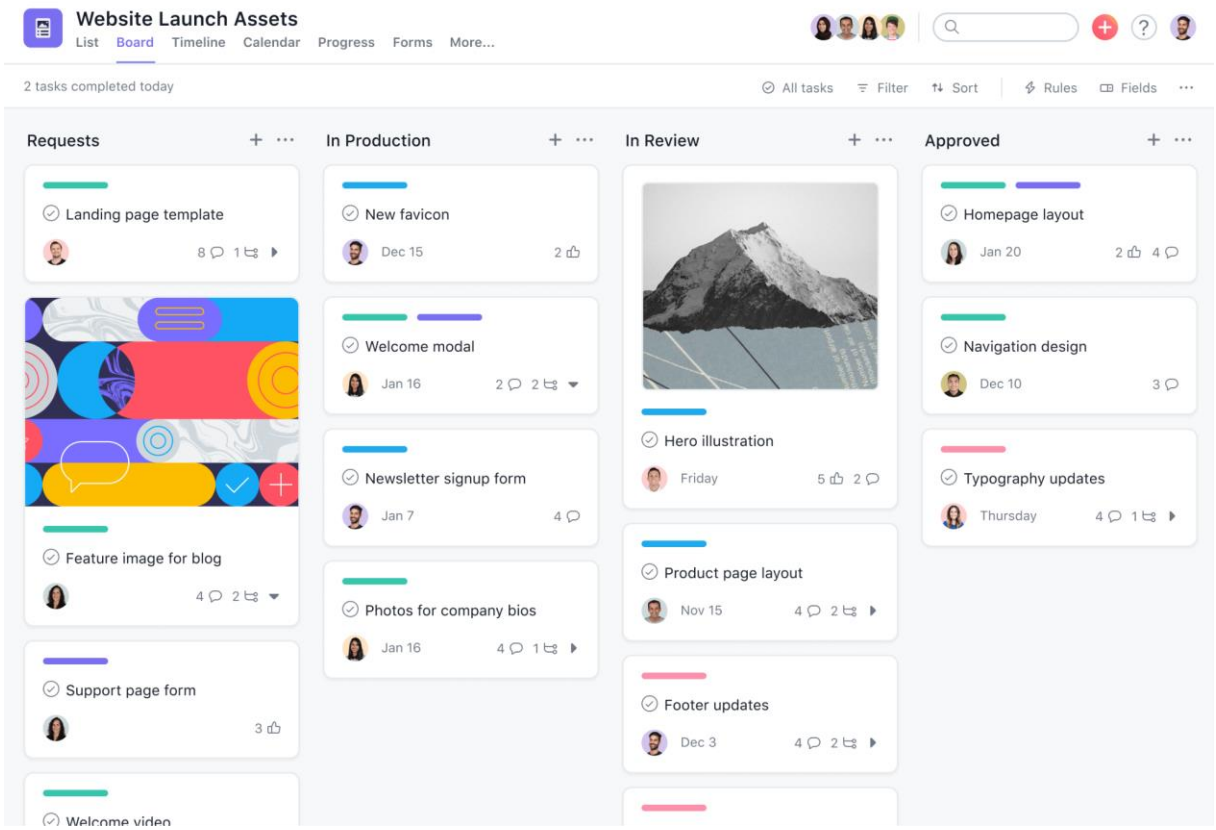

Рисунок 1.3 – Вікно додатка Asana

Проте аналітичні інструменти Asana мають обмежену глибину: немає зведених звітів за період до пів року або окремого відображення багів. Комунікаційні можливості також обмежуються згадками імен у коментарях – контактна інформація учасників не є доступною. Як і в Jira та Trello, пошук зосереджений на задачах та окремих користувачах, а не на командах, що ускладнює навігацію в межах великих організацій [9].

ClickUp (рис. 1.4) – інструмент, що поєднує функції управління задачами, документообігу, календарів і тайм-трекінгу.

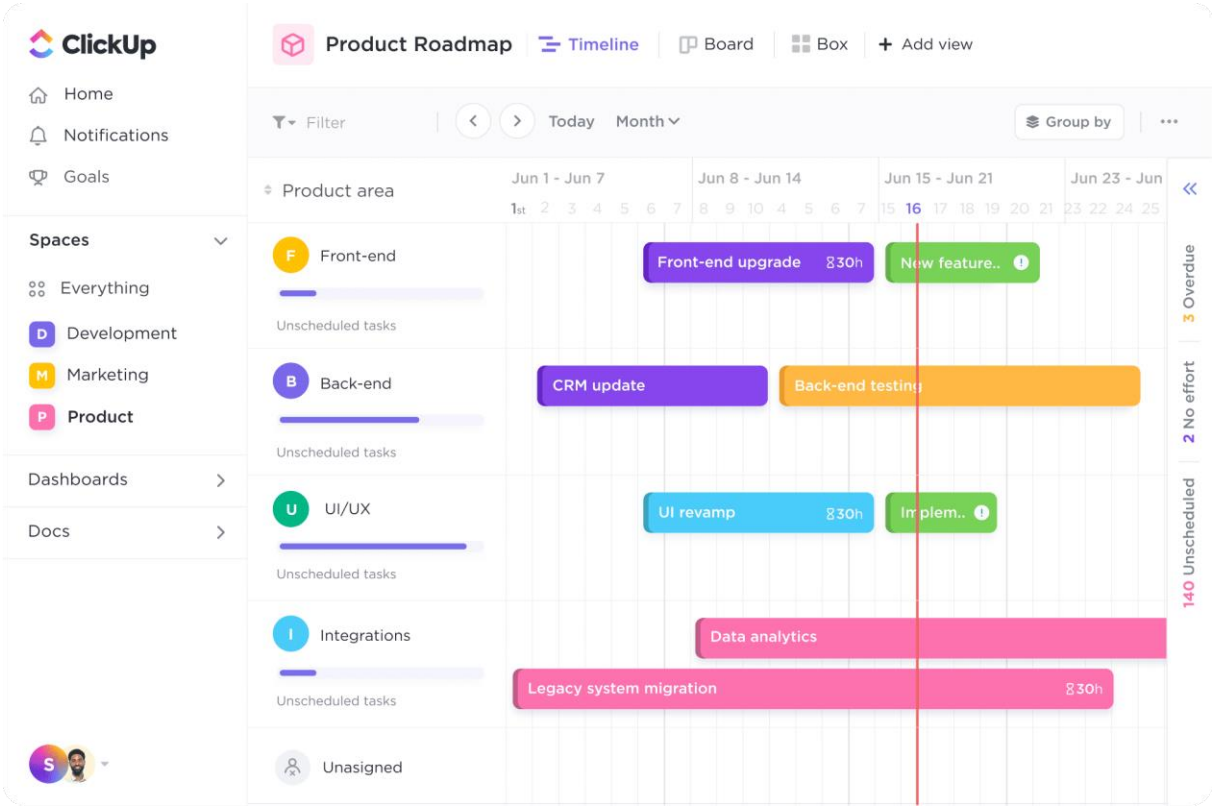

Рисунок 1.4 – Вікно додатка ClickUp

ClickUp має розширений функціонал і підходить для різних сфер, але через перенасичення можливостями може здаватися складним у використанні. Аналітика присутня, проте вона переважно зосереджена на таймінгу виконання задач, а не на глибокій статистиці про динаміку виконаних задач чи кількість багів за певний проміжок часу. У ClickUp немає централізованого профілю команди або можливості переходу до неї з задачі. Також пошук працює за іменами без можливості здійснювати його за назвами команд виконавців [10].

Таким чином, популярні існуючі платформи переважно орієнтовані або на простоту, або на надмірну функціональність, але жодна не забезпечує поєднання таких критичних функцій, як вбудована аналітика, розширена комунікація та навігація між командами й задачами, що залишає простір для нових актуальних рішень.

1.3 Постановка задачі дослідження

Розроблюваний WEB-додаток призначений в першу чергу для учасників команд, а саме: розробників, дизайнерів, маркетологів, керівників, які працюють над спільними проєктами. Водночас він може стати ефективним інструментом для індивідуального користування, наприклад: для ведення особистих справ, щоденних задач або організації побутових процесів. Така універсальність зумовлює необхідність створення інтуїтивно зрозумілого інтерфейсу й адаптивного функціоналу.

Завдання полягає у розробці додатка, який дозволяє створювати, редагувати та контролювати задачі, отримувати статистику їх виконання, шукати інформацію про команди та їх учасників, а також отримувати доступ до розширених контактних даних, що відображаються безпосередньо в інтерфейсі додатка. Застосунок виконуватиме роль універсального цифрового помічника для роботи в команді, надаючи зручні інструменти для щоденного використання.

У розробці додатка передбачено застосування методів об’єктно-орієнтованого проєктування, а також побудова структурної та функціональної моделей системи за допомогою відповідних візуалізацій. Передбачається створення клієнт-серверної архітектури та використання реляційної бази даних для збереження інформації про задачі, команди та користувачів. Реалізація буде здійснюватися з урахуванням вимог до масштабованості, модульності та безпеки.

WEB-додаток має реалізовувати такі функції:

- створення, редагування та перегляд задач;
- фільтрація задач за командами, пріоритетом, статусом виконання;
- статистика виконання задач і кількості багів за певний час;
- доступ до контактної інформації колег (номер телефону, email тощо);
- пошук певних сутностей системи, відповідно до її атрибутів;
- можливість посилання на сторінку команди безпосередньо з

інтерфейсу задачі.

Отже, для виконання поставленого завдання необхідно:

- розробити структуру програмного забезпечення;
- створити діаграми діяльності та діаграми прецедентів;
- створити діаграми класів;
- розробити ER-діаграми;
- описати загальну архітектуру програми та структуру бази даних;
- обґрунтувати вибір технологій розробки;
- реалізувати усі зазначені функції з дотриманням принципів масштабованості, безпеки та адаптивності;
- розробити інструкції з розгортання, перенесення та експлуатації WEB-додатка.

WEB-додаток має передбачати реалізацію адаптивного та інтуїтивно зрозумілого інтерфейсу, модульної структури класів, що дозволяє розширювати функціональність додатка, механізмів автентифікації й авторизації для безпечної роботи користувачів. Застосунок також має бути придатним до масштабування, підтримувати ефективне зберігання та обробку великого обсягу даних.

Завдяки вбудованій аналітиці додаток сприятиме прийняттю менеджерами ефективних управлінських рішень на основі зібраних метрик, а функція переходу між задачами та командами значно підвищить прозорість виконання робіт, покращить навігацію між WEB-сторінками та сприятиме оперативному контролю над командною діяльністю.

1.4 Висновок

У даному розділі було виконано аналіз предметної області. Було виявлено, що традиційні засоби планування, як-от електронні таблиці чи месенджери, не відповідають вимогам сучасного динамічного середовища. На

основі виконаного аналізу виявлено зростаючий попит на функціонально насичені, гнучкі та аналітично орієнтовані системи управління задачами. Зауважено на доцільності створення WEB-додатка, що поєднує зручність використання із розширеною функціональністю.

Виконано огляд і порівняльний аналіз популярних інструментів у сфері менеджменту задач, зокрема Jira, Trello, Asana та ClickUp. Було виявлено, що кожен з розглянутих продуктів має низку переваг, проте не забезпечує повного набору функцій, необхідних для комплексного менеджменту задач, наприклад, вбудована аналітика задач на період до 6 місяців, доступ до контактної інформації колег, можливість пошуку за назвами команд, посилання на сторінку команди безпосередньо з інтерфейсу задачі. Здійснений аналіз дозволив окреслити ключові функціональні прогалини у наявних рішеннях і визначити вектор розвитку майбутнього WEB-додатка.

Сформульовано постановку задачі дослідження, яка містить опис цільової авдиторії, загальну функціональність майбутнього додатка (зокрема ті функції, що були відсутні в аналогах), доцільність створення структурних і функціональних діаграм роботи системи.

2 ПРОЄКТУВАННЯ СТРУКТУРИ ТА АЛГОРИТМІВ ФУНКЦІОНУВАННЯ WEB-ДОДАТКА ДЛЯ МЕНЕДЖМЕНТУ ЗАДАЧ РІЗНОГО ТИПУ

Проєктування структури та логіки функціонування програмного забезпечення є ключовим у процесі створення сучасного WEB-додатка, оскільки слугує фундаментом для подальшої технічної реалізації. Саме на цьому етапі визначаються ключові компоненти, їхні взаємозв'язки, сценарії взаємодії з користувачем та принципи організації даних. У межах даного розділу буде здійснено комплексне моделювання структури та функціонування програмного забезпечення за допомогою загальної структурної схеми, діаграм видів діяльності, діаграми прецедентів, UML-діаграм класів і ER-моделей бази даних.

2.1 Обґрунтування вибору архітектурних рішень розроблюваного WEB-додатка

Насамперед, ефективне функціонування WEB-додатка передбачає узгоджену взаємодію всіх його компонентів. З огляду на це, для реалізації розроблюваної системи було обрано клієнт-серверну архітектуру (рис. 2.1), що вважається надійним та масштабованим підходом і широко застосовується у створенні корпоративних (enterprise) програмних продуктів. У якості єдиного протоколу обміну даними між клієнтом і сервером передбачено використання REST API, що забезпечує уніфікований та ефективний спосіб організації взаємодії між модулями системи.

Потенційне розширення масштабів використання WEB-додатка вимагає підтримки можливостей горизонтального та вертикального масштабування, що дозволяє додавати нові обчислювальні ресурси або розподіляти навантаження між вузлами. Розробка також повинна враховувати потреби у надійності та відмовостійкості системи при зростанні кількості одночасних користувачів.

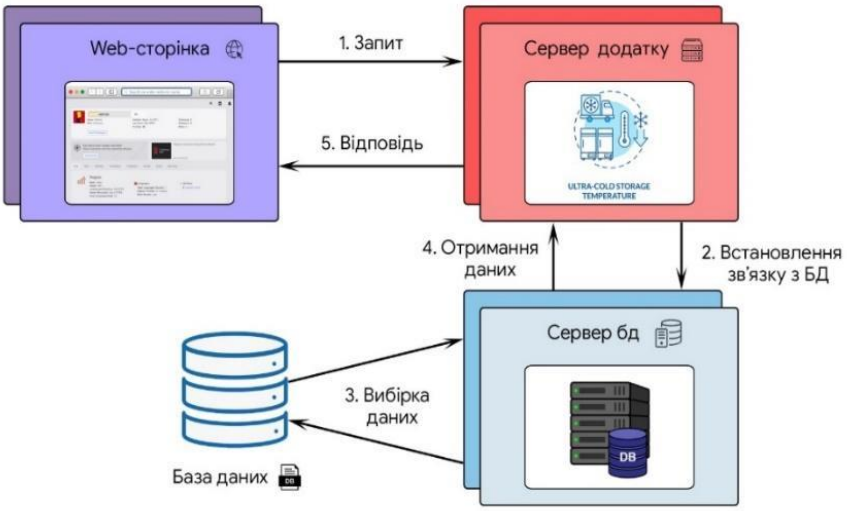

Рисунок 2.1 – Клієнт-серверна архітектура додатка

З огляду на це, прийнято рішення застосувати мікросервісну архітектуру як основнйи підхід у побудові взаємозв’язків між компонентами системи (приклад архітектури мікросервісів наведено на рис. 2.2). Цей підхід дозволяє розподілити функціональність між незалежними сервісами, кожен з яких відповідає за окрему частину бізнес-логіки. Це, у свою чергу, спрощує масштабування, підвищує гнучкість у розробці та забезпечує більшу стійкість до збоїв при високому навантаженні [11].

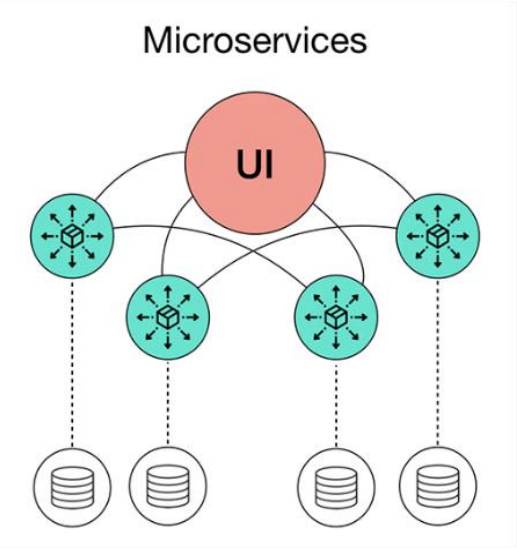

Рисунок 2.2 – Архітектура мікросервісів

2.2 Розробка загальної структурної схеми WEB-додатка

На основі обраної архітектури розроблюваного програмного забезпечення розробимо загальну структурну схему WEB-додатка для менеджменту задач різного типу (рис. 2.3).

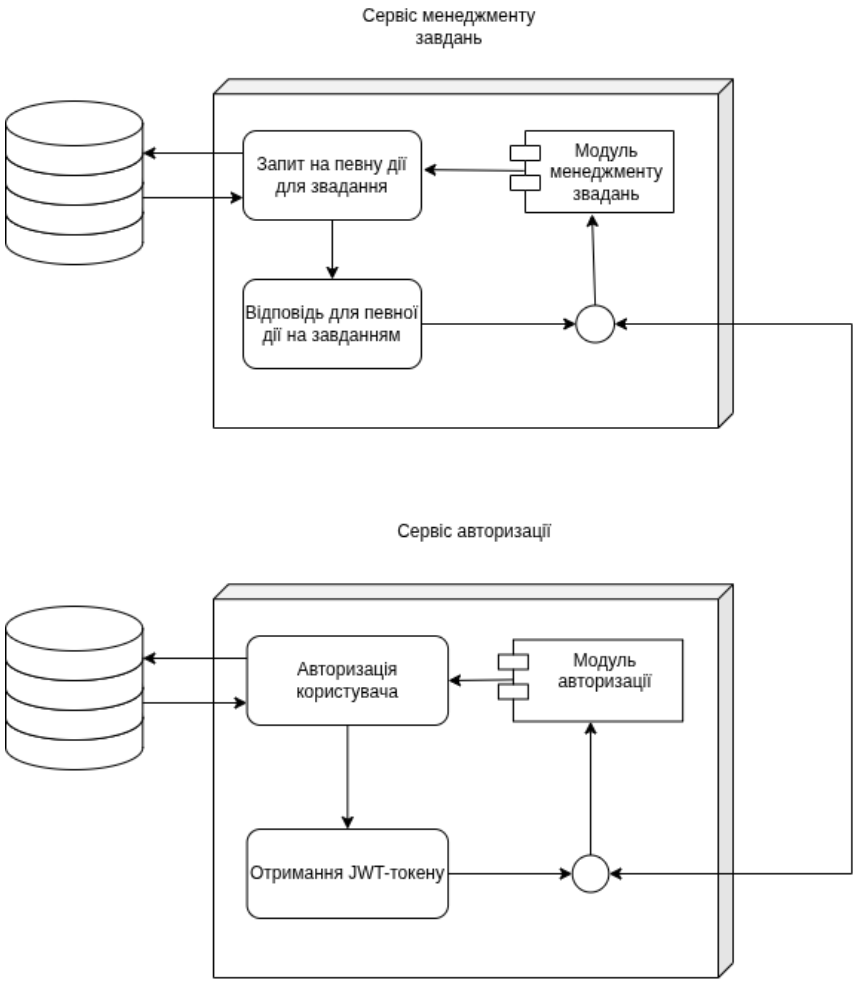

Рисунок 2.3 – Загальна структурна схема WEB-додатка

Дана схема моделює набір мікросервісів, що виконуватимуть окремі функціональні ролі у забезпеченні цілісної та ефективної роботи системи. Передбачено, зокрема, авторизаційний сервіс, відповідальний за обробку процесів реєстрації та автентифікації користувачів, а також сервіс управління задачами, що реалізовуватиме як базовий функціонал створення, редагування, видалення, перегляду та зміни статусу завдань, так і додатковий, покликаний покращити користувацький досвід, до якого належить вбудована аналітика

задач на період до 6 місяців, доступ до контактної інформації колег, можливість пошуку за назвами команд, посилання на сторінку команди безпосередньо з інтерфейсу задачі.

Для підвищення продуктивності та зменшення затримок в обробці запитів у програмній реалізації передбачено застосування технологій кешування на обох сервісах. Комунікація між мікросервісами реалізується шляхом формування REST-запитів, що забезпечує гнучку та зручну взаємодію компонентів у межах обраної архітектури проєкту. Керування змінами в базах даних, які використовуються в авторизаційному сервісі та сервісі управління задачами, передбачає застосування підходів до контрольованого та послідовного оновлення структури даних, що сприяє узгодженості й відстежуваності таких змін.

Окрему увагу приділено реалізації механізмів авторизації, що ґрунтуються на принципах токен-орієнтованого доступу. Після проходження автентифікації користувач отримує токен з обмеженим терміном дії, який містить всю необхідну інформацію для підтвердження доступу при подальших зверненнях до функціоналу системи. Такий підхід дозволяє мінімізувати навантаження на авторизаційний сервіс та підвищити загальну ефективність взаємодії користувача з WEB-додатком.

2.3 Розробка діаграм видів діяльності

Моделювання бізнес-логіки системи за допомогою діаграм виду діяльності (activity diagrams) є важливим інструментом у процесі розробки, оскільки дозволяє формалізувати алгоритмічні аспекти взаємодії користувача з додатком та внутрішні механізми обробки даних. Такі візуалізації сприяють глибшому розумінню функціональних сценаріїв, дозволяють виявити потенційні проблемні місця у процесах і забезпечують ефективне документування динаміки поведінки системи.

На рисунку 2.4 наведено діаграму виду діяльності, яка відображає послідовність дій під час входу користувача в систему.

Розглянуто етапи введення облікових даних, перевірки їхньої коректності, відображення повідомлення про помилку у разі не валідності даних, а також перенаправлення на сторінку технічної підтримки в разі відсутності облікового запису користувача.

Рисунок 2.4 – Діаграма виду діяльності для процесу авторизації

Розглянемо рисунок 2.5, що ілюструє діаграму виду діяльності, яка відображає дії користувача під час створення нового завдання. Візуалізація охоплює повну послідовність операцій: від ініціації створення задачі до збереження інформації у відповідній базі даних.

Також на рис. 2.5 зображено перевірку введених даних, валідацію обов’язкових полів, обробку запиту на рівні сервісу та подальше підтвердження успішного створення завдання. Такий підхід до моделювання дозволяє ефективніше організувати логіку взаємодії користувача з

інтерфейсом і забезпечити надійність створення нових об'єктів у системі.

На рисунку 2.6 представлено діаграму виду діяльності, що відображає загальний процес пошуку сутностей у системі, зокрема таких як команди, співробітники та завдання. Діаграма демонструє основні етапи, які проходить користувач під час здійснення запиту на пошук: введення ключових слів або фільтрів, запуск пошуку, обробка запиту системою, звернення до релевантних сервісів і подальше відображення результатів.

Рисунок 2.5 – Діаграма виду діяльності для створення завдання

Процес пошуку охоплює внутрішню логіку фільтрації та формування відповіді, що забезпечує швидкий доступ до необхідної інформації та підвищує зручність користування додатком.

Розглянемо рисунок 2.7. Дана візуалізація ілюструє послідовність дій користувача під час переходу до робочої дошки із завданнями та зміни статусу одного з них. Діаграма демонструє логіку навігації від стартової точки до

взаємодії з окремим завданням на відповідній дошці, зокрема зміну його статусу зі “Backlog” на “In Progress” або “Completed”.

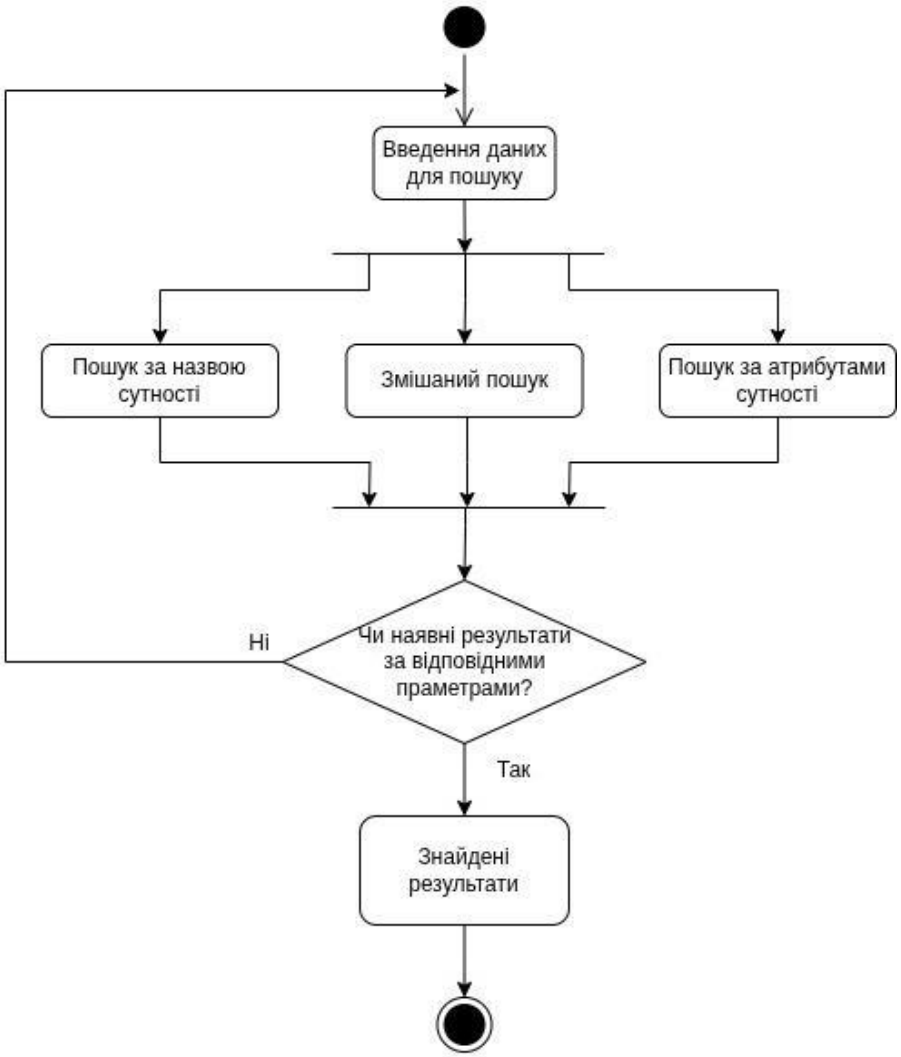

Рисунок 2.6 – Діаграма виду діяльності для пошуку сутності

Зміна статусу не передбачає додаткових перевірок із боку системи, а контроль достовірності таких змін покладається безпосередньо на користувача. Такий підхід дозволяє реалізувати зручний механізм візуального управління поточними задачами в межах командної роботи.

Рисунок 2.8 демонструє загальний процес редагування персональних даних користувача. Діаграма охоплює етапи оновлення імені, контактної інформації, а також елементів, пов’язаних із командною належністю.

Рисунок 2.7 – Діаграма виду діяльності для навігації та зміни статусу завдання

Слід зауважити, що дані, які стосуються командних атрибутів користувача, таких як його роль у команді або її назва, не можуть бути змінені автоматично. Для забезпечення цілісності та контролю доступу, ці зміни вимагають підтвердження з боку адміністратора системи. Такий підхід гарантує, що всі оновлення, які можуть вплинути на організаційну структуру, відбуватимуться лише після відповідного їх затвердження.

Рисунок 2.8 – Діаграма виду діяльності для зміни персональних даних користувача

2.4 Розробка діаграми прецедентів

З метою формалізації функціональних можливостей програмної системи та визначення взаємодії користувачів з її інтерфейсом доцільним є застосування діаграми прецедентів. Такий тип візуалізації дозволяє окреслити основні сценарії використання додатка, описати взаємозв'язки між ролями користувачів і функціоналом системи, а також визначити межі відповідальності кожного з акторів. Створення діаграми прецедентів є важливим етапом моделювання, який слугує підґрунтям для проєктування логіки реалізації програмного забезпечення та подальшої розробки компонентів системи відповідно до потреб кінцевих користувачів.

На рисунку 2.9 наведено діаграму прецедентів WEB-додатка для менеджменту задач різного типу, що ілюструє функціональні можливості системи менеджменту задач у розрізі ролей користувачів. Дана діаграма охоплює два основні типи акторів, а саме "Звичайний користувач" та "Адміністратор", та дозволяє чітко окреслити доступний для кожної ролі функціонал.

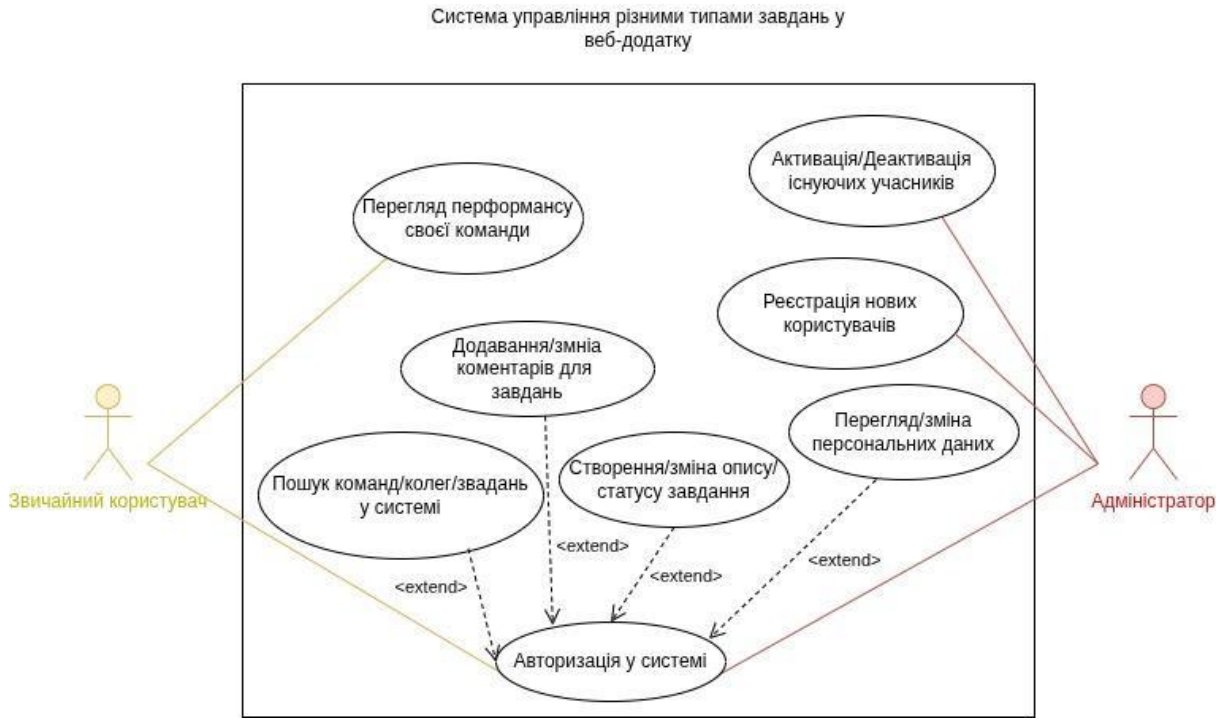

Рисунок 2.9 – Діаграма прецедентів

Роль “Звичайний користувач” має доступ як до базових функцій, так і до персоналізованих можливостей системи, зокрема: автентифікації, створення, редагування та перегляду завдань, додавання та редагування коментарів, навігації по дошці завдань, перегляду аналітики продуктивності команди, а також редагування особистого профілю та здійснення пошуку співробітників, команд і задач.

Натомість “Адміністратор” володіє повним набором базового функціоналу користувача, а також має розширені можливості, серед яких реєстрація нових користувачів, активація та деактивація облікових записів, що дозволяє здійснювати адміністративне управління системою та контролювати доступ до її ресурсів. Діаграма прецедентів таким чином виконує функцію чіткого розмежування обов’язків і повноважень, забезпечуючи прозорість та логічність організації користувацьких сценаріїв взаємодії з системою.

2.5 Розробка структури класів сервісів

У процесі проєктування складових програмного забезпечення важливим етапом є побудова UML-діаграм класів, які відображають структуру основних сервісів системи, їхню внутрішню організацію, взаємозв’язки між класами, а також призначення атрибутів і методів. Такі діаграми дають змогу формалізувати на концептуальному рівні складові підсистем, визначити залежності між компонентами та забезпечити цілісне бачення структури програмної реалізації. У даному підрозділі розглянуто UML-моделі двох ключових сервісів WEB-додатка, а саме сервісу авторизації та сервісу менеджменту завдань.

На рисунку 2.10 наведено UML-діаграму сервісу авторизації, яка ілюструє основні класи, їх атрибути, методи та зв’язки між ними в межах цього підсистемного компонента.

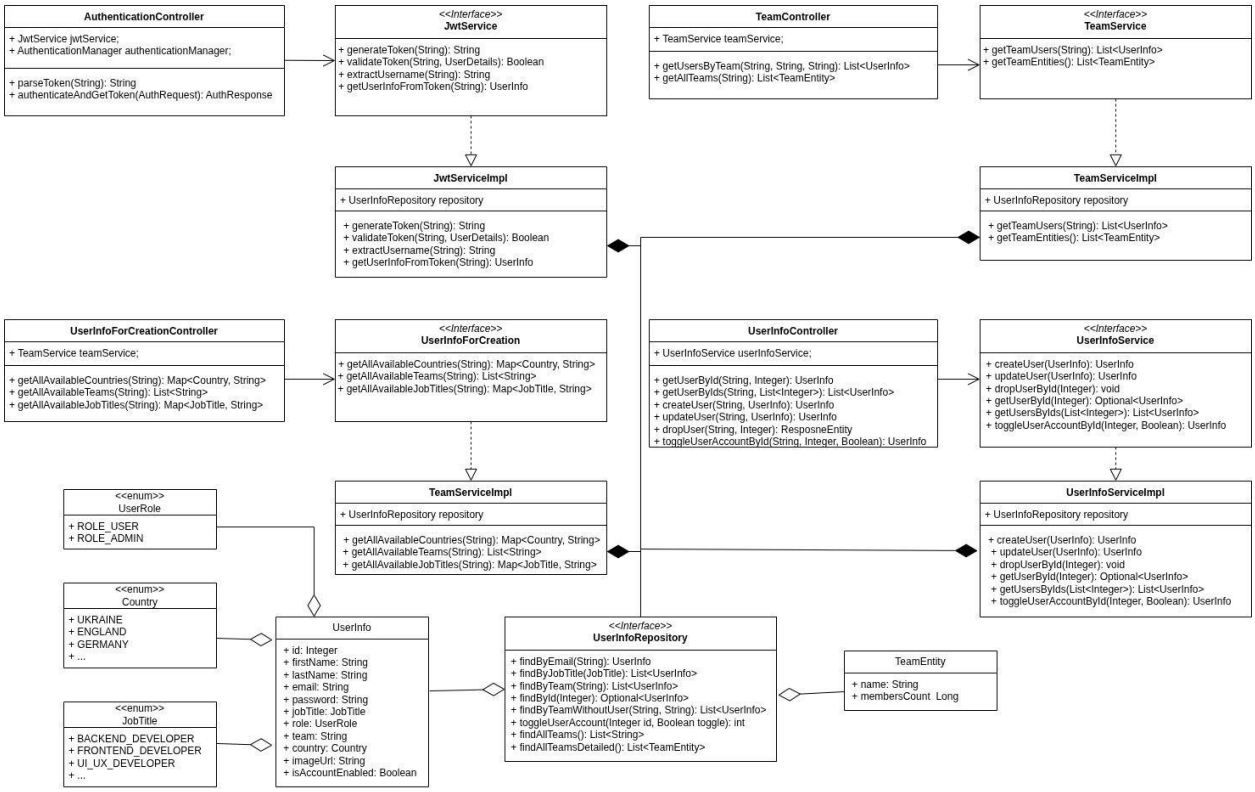

Рисунок 2.10 – UML-діаграма класів сервісу авторизації

На рисунку 2.10 представлено структурну модель сервісу авторизації, яка відображає загальний механізм обробки запитів, пов’язаних із автентифікацією користувачів та керуванням обліковими даними. Усі вхідні запити обробляються відповідними контролерами: UserInfoForCreationController, UserInfoController, TeamController та AuthenticationController. Кожен із них виконує функцію маршрутизації запитів до відповідних сервісів, ініціюючи дії відповідно до логіки бізнес-процесів. Зокрема, компонент JwtService відповідає за генерацію авторизаційних токенів, що гарантують безпечний доступ до функціоналу системи. TeamService забезпечує пошук учасників у межах команди, тоді як UserInfoService реалізує базові CRUD-операції над користувачами. Сервіс UserInfoForCreationService відповідальний за підготовку вхідних даних для створення нових користувачів.

Усі згадані сервіси взаємодіють із шаром доступу до даних – UserInfoRepository, який відповідає за зв’язок з базою даних. Центральним

елементом моделі є сутність UserInfo, що зберігає основну інформацію про користувачів системи. Додаткові об’єкти, такі як Role, Country та JobTitle, реалізовані у вигляді перерахувань (enum) і містять наперед визначені набори значень, які використовуються для класифікації користувачів за роллю, країною та посадою відповідно. Представлена діаграма забезпечує цілісне уявлення про структуру сервісу авторизації та його роль у процесах обробки запитів, керування доступом і зберігання користувацьких даних у системі.

Перейдемо до структури сервісу управління завданнями, що відповідає за весь спектр операцій, пов’язаних із задачами. До його функціоналу належать створення, редагування, видалення та перегляд завдань, додавання коментарів, робота з пов’язаними задачами, а також перегляд задач на рівні команд. Коректне проєктування моделі цього сервісу є критично важливим для забезпечення стабільної взаємодії користувачів із системою,а також для збереження цілісності даних і правильного розмежування доступу до функціональності.

На рисунку 2.11 зображено UML-діаграму класів сервісу менеджменту завдань, що демонструє структуру ключових компонентів, їх атрибути, методи та взаємозв’язки.

Центральним елементом підсистеми є TicketController, який обробляє вхідні запити клієнта та передає їх до відповідного сервісного шару. Основна бізнес-логіка зосереджена в TaskEntityService, який забезпечує координацію дій між пов’язаними компонентами, зокрема, сервісами оцінювання, коментарів, пов’язаних задач та користувачів.

Клас TaskEntityServiceImpl реалізує логіку взаємодії із завданнями, залучаючи відповідні сервіси: LinkedIssueEntityService – для обробки пов’язаних задач, CommentEntityService – для управління коментарями, EstimateEntityService – для оцінювання задач, а також зовнішній клієнт UserClient, який надає доступ до інформації про користувачів. Така організація взаємодії між компонентами дозволяє ефективно обробляти запити будь-якої складності.

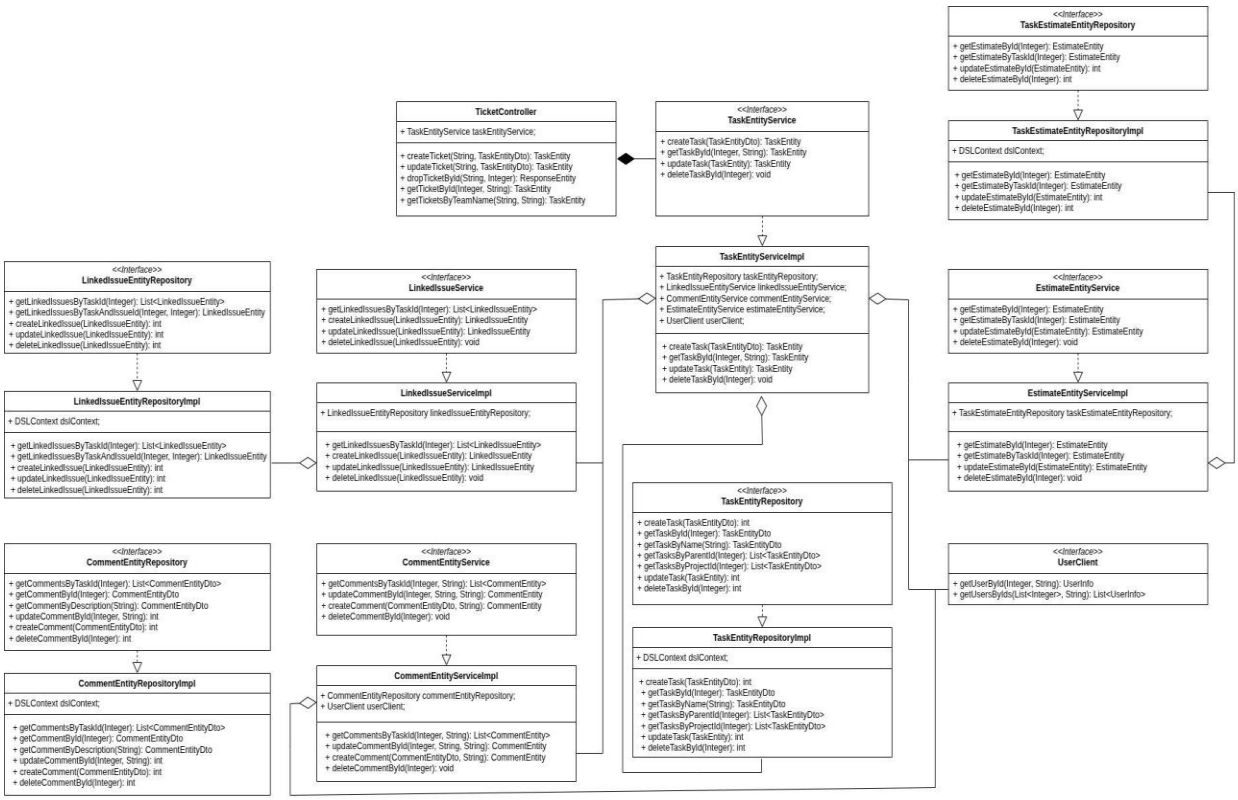

Рисунок 2.11 – UML-діаграма класів сервісу менеджменту завдань

Кожен тип сутності має власний репозиторій і відповідну реалізацію, що використовує DSLContext з бібліотеки JOOQ [27] для формування типізованих SQL-запитів. Наприклад, TaskEntityRepositoryImpl виконує операції над задачами, включно з фільтрацією за ідентифікатором, назвою чи проєктом.

Окрему увагу слід приділити підсистемам пов’язаних задач (LinkedIssueEntity) та коментарів (CommentEntity), реалізованих як окремі сервіси з чітко визначеними інтерфейсами. Це дозволяє фіксувати історію взаємопов’язаних процесів і комунікацій між користувачами. Компонент EstimateEntity виконує функції, пов’язані з розрахунком часу виконання завдань, що є ключовим у плануванні дедлайнів.

Завдяки інтеграції сервісів через інтерфейси та суворому розподілу обов’язків, дана система є гнучкою, масштабованою та зручною для подальшого розвитку. UML-діаграма демонструє високий рівень модульності, що відповідає принципам чистої архітектури та мікросервісного підходу.

2.6 Розробка ER-моделей

ER-діаграма (Entity-Relationship діаграма) – це графічне представлення логічної структури бази даних, що відображає сутності, їхні атрибути та взаємозв'язки між ними. У межах даного проєкту ER-діаграми необхідні для формалізації моделі даних, яка ляже в основу створення реляційної бази даних системи. Вони допомагають структурувати інформацію, визначати взаємозалежності між основними елементами, забезпечувати узгодженість даних і підтримувати логіку функціонування ключових сервісів програмного застосунку.

Розглянемо рисунок 2.12. На ньому зображено ER-діаграму для сервісу авторизації користувачів системи.

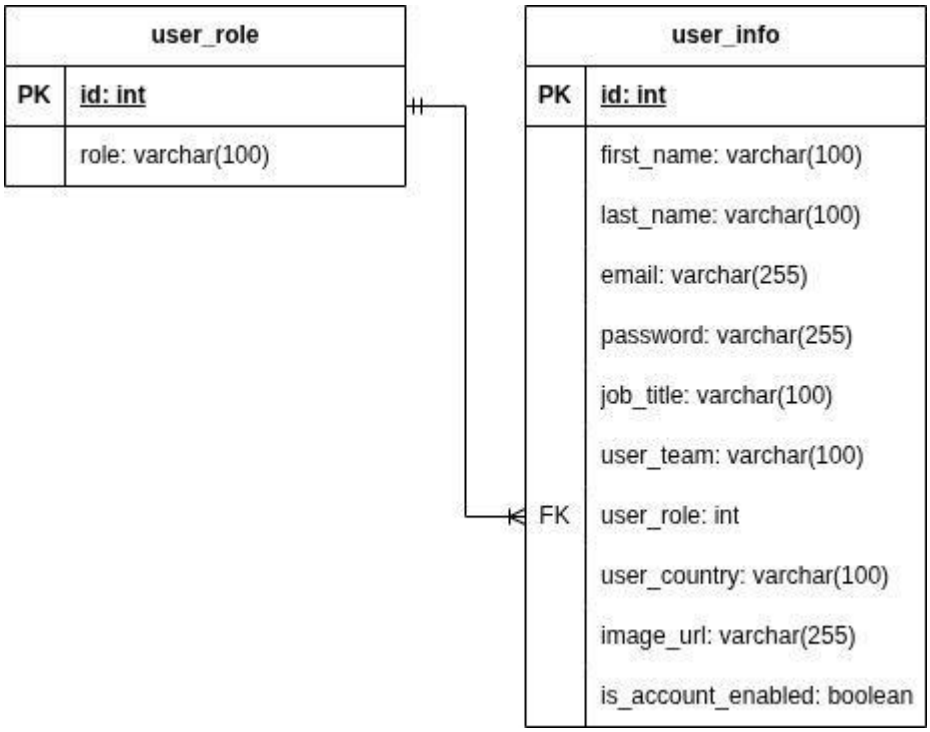

Рисунок 2.12 – ER-діаграма для сервісу авторизації

Представлена ER-діаграма моделює взаємозв'язки між двома основними сутностями системи: user_info та user_role, які мають центральне значення у забезпеченні ідентифікації користувачів та управлінні їхніми правами доступу до функціональності додатка.

Сутність user_info є основною таблицею, що зберігає персональні та облікові дані користувачів. Вона включає такі атрибути, як: first_name та last_name – ім’я та прізвище користувача, email і password – облікові дані для авторизації, job_title, user_team, user_country – службова та організаційна інформація, image_url – посилання на зображення профілю користувача, is_account_enabled – булевий атрибут, що вказує на активність облікового запису.

Первинним ключем виступає поле id, яке гарантує унікальну ідентифікацію кожного запису. Атрибут user_role реалізований як зовнішній ключ, що пов’язує запис із відповідною роллю у таблиці user_role.

Сутність user_role містить перелік доступних ролей користувачів у системі. Вона складається з двох атрибутів: id – первинний ключ, який однозначно ідентифікує кожну роль, та role – текстовий опис ролі ("ADMIN", "USER").

Між таблицями user_info та user_role встановлено зв’язок типу багато до одного (many-to-one): багато користувачів можуть мати одну й ту саму роль, але кожен користувач може бути асоційований лише з однією конкретною роллю.

Перейдемо до рисунку 2.13, де наведено ER-діаграму сервісу менеджменту задач, яка ілюструє логічну структуру даних, що використовуються для зберігання, обробки та взаємодії між ключовими елементами системи. Ця діаграма відображає взаємозв’язки між основними сутностями, такими як завдання, оцінки часу, коментарі та пов’язані проблеми, що дозволяє краще зрозуміти логіку реалізації функціональності на рівні бази даних сервісу.

Дана діаграма охоплює чотири основні таблиці, що формують структуру бази даних відповідного сервісу. Головною таблицею є project_tasks, у якій зберігаються атрибути, що характеризують завдання: назва, опис, автор, виконавець, тип, статус, команда, пріоритет, а також дати початку й завершення виконання. Зовнішній ключ до таблиці task_estimates дозволяє

зберігати інформацію про планові й фактичні витрати часу у тижнях, днях і годинах, реалізуючи таким чином контроль над дотриманням термінів.

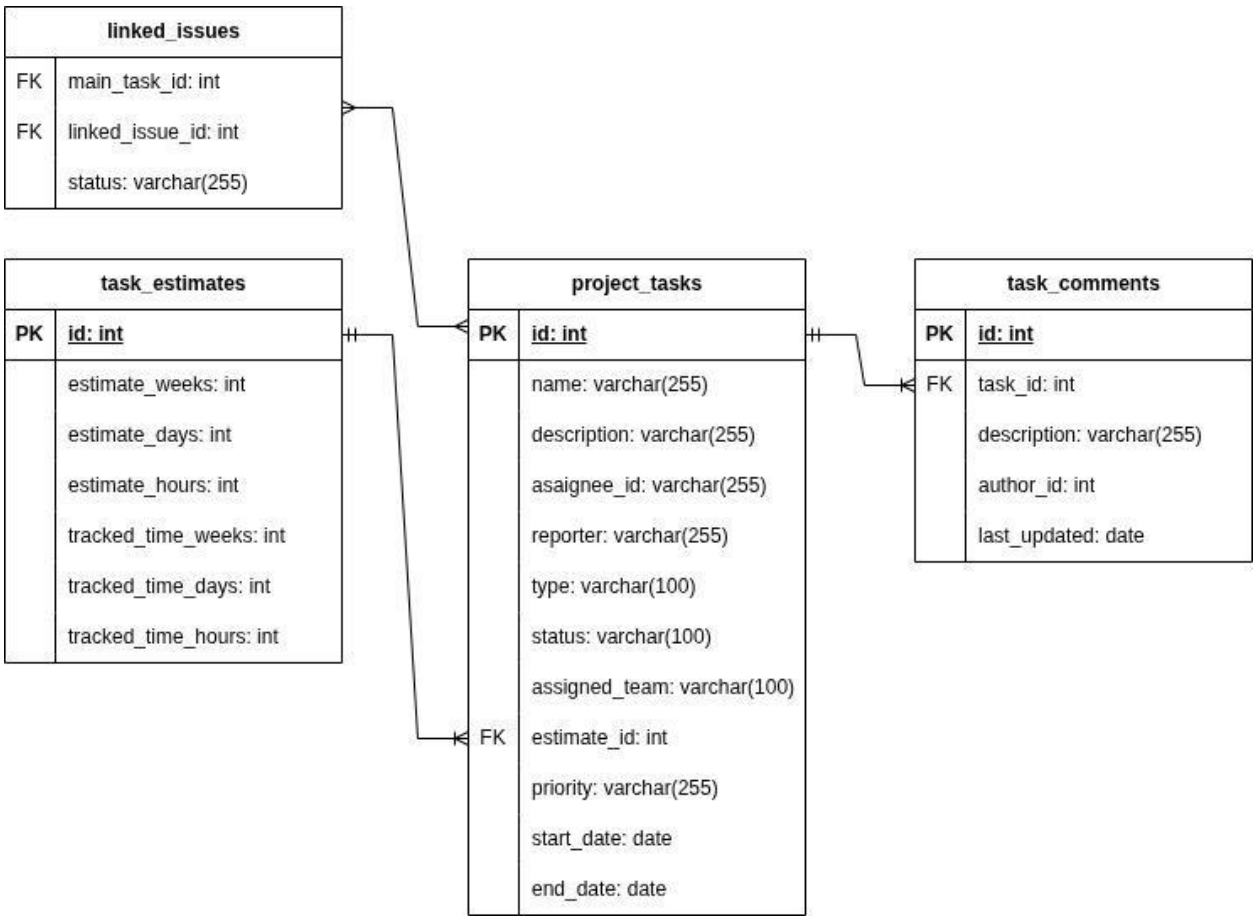

Рисунок 2.13 – ER-діаграма сервісу менеджменту завдань

Таблиця task_comments реалізує зберігання коментарів, залишених до завдань. Вона містить поля з даними автора, часу останнього оновлення та зовнішній ключ до відповідного завдання.

Ще однією важливою частиною моделі є таблиця linked_issues, яка дозволяє встановлювати зв’язки між завданнями через зв’язок типу “багато до багатьох” між завданнями в межах таблиці project_tasks. Поля main_task_id та linked_issue_id описують ці зв’язки, а додатковий атрибут status дозволяє фіксувати тип відношення (наприклад, “пов’язано” чи “блокує”).

Усі ці таблиці разом утворюють гнучку модель даних, здатну забезпечити детальне управління завданнями в рамках розроблюваного проєкту.

2.7 Висновок

У даному розділі обґрунтовано вибір архітектурних рішень для реалізації WEB-додатка. Визначено доцільність використання клієнт-серверної архітектури з REST API, а також мікросервісного підходу, що забезпечує гнучкість, масштабованість і стабільність при зростанні навантаження.

Розроблено загальну структурну схему, яка охоплює ключові компоненти програмного забезпечення та їхню взаємодію. Блок управління задачами містить у собі, окрім базового функціоналу, також додатковий (аналітику, доступ до контактної інформації, пошук за назвами команд та переходи до команд із задач), покликаний покращити користувацький досвід. Передбачено використання механізмів кешування, токен-орієнтованої авторизації та підходів до контрольованого оновлення даних, що сприяє ефективній і безпечній роботі застосунку.

Створено діаграми видів діяльності для основних користувацьких сценаріїв: автентифікації, створення завдання, пошуку за атрибутом, зміни статусу задачі, редагування особистих даних. Візуалізації дозволили формалізувати послідовність дій і внутрішню логіку обробки запитів.

Розроблено діаграму прецедентів, яка відображає функціональність WEB-додатка у розрізі ролей користувачів. Було окреслено межі доступу для звичайного користувача та адміністратора, що дозволяє впровадити чітке розмежування повноважень.

Побудовано UML-діаграми класів для сервісів авторизації та управління задачами. Моделі відображають структуру класів, їхні методи, атрибути та взаємозв’язки, що забезпечує цілісне бачення внутрішньої логіки реалізації.

Розроблено ER-моделі для двох сервісів, що охоплюють основні сутності, атрибути та типи зв’язків між таблицями. Моделювання структури бази даних створює підґрунтя для ефективного зберігання, обробки та взаємодії з інформацією.

3 ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ ТЕХНІЧНОЇ РЕАЛІЗАЦІЇ WEB-ДОДАТКА ДЛЯ МЕНЕДЖМЕНТУ ЗАДАЧ РІЗНОГО ТИПУ

Розробка WEB-додатка для менеджменту задач різного типу потребує виваженого підходу до вибору технологій, що забезпечують ефективну реалізацію як серверної, так і клієнтської частин застосунку. Обґрунтований вибір мов програмування, фреймворків, бібліотек, систем управління базами даних, засобів міжсервісної взаємодії, інструментів безпеки, кешування, тестування та середовищ розробки є важливою передумовою досягнення належного рівня продуктивності, надійності та масштабованості.

У межах цього розділу буде обґрунтовано вибір конкретного технологічного стеку, що забезпечить повноцінне функціонування як бекенд-, так і фронтенд-частин WEB-додатка, що найбільш повно відповідатиме потребам і цілям даного проєкту.

3.1 Обґрунтування вибору мови програмування розробки серверної частини WEB-додатка

Розробка серверної частини є основоположним кроком у створенні цілісної програмної системи, оскільки цей етап забезпечує розробку підґрунтя, що з'єднуватиме всі елементи WEB-додатка та забезпечуватиме їх комплементарну роботу. Виконаємо порівняльний аналіз чотирьох мов програмування, що є найбільш часто використовуваними у вирішенні задач бекенд-розробки через свої сильні сторони, а саме: Java, Python, PHP та C#. Кожну з них буде розглянуто з точки зору відповідності архітектурним вимогам проєкту, зокрема підтримки мікросервісної архітектури, обробки великої кількості запитів, реалізації надійних механізмів безпеки та масштабованості.

Однією з найпоширеніших мов програмування, що активно застосовується для створення серверної логіки у системах корпоративного

рівня, є Java. Ця мова підтримує об'єктно-орієнтовану парадигму програмування, що сприяє модульності, повторному використанню коду та легкості його супроводу. Кросплатформені можливості Java забезпечуються завдяки використанню віртуальної машини Java (JVM), яка виконує байт-код незалежно від конкретної операційної системи. Такий підхід формує технічні передумови для гнучкого та універсального розгортання програмних рішень у різноманітних обчислювальних середовищах.

Суттєвою перевагою Java є підтримка багатопотоковості, що дає змогу ефективно обробляти велику кількість паралельних запитів, знижуючи затримки у відповідях серверної частини та підвищуючи загальну продуктивність системи. Сувора статична типізація, властива цій мові, забезпечує виявлення помилок на етапі компіляції, що, у свою чергу, підвищує надійність програмного коду та сприяє зниженню кількості потенційних вразливостей на рівні логіки. Розвинена екосистема Java включає великий набір бібліотек та фреймворків, серед яких можна виділити Spring. Дана платформа орієнтована на спрощення створення WEB-застосунків, що відповідають вимогам мікросервісної архітектури. Усе вищезазначене свідчить про високу відповідність мови Java поставленим вимогам до проєкту, зокрема в аспектах масштабованості, безпеки та підтримки сучасних архітектурних рішень [12].

Іншим популярним засобом для реалізації серверної логіки є мова Python. Вона вирізняється лаконічним синтаксисом та низьким порогом входу, що забезпечує високу швидкість розробки початкових версій програмних рішень. Наявність фреймворків, таких як Django та Flask, дозволяє швидко створювати WEB-сервери, виконувати маршрутизацію запитів і реалізовувати базову логіку взаємодії з клієнтською частиною. Однак у випадку складних і високонавантажених систем, до прикладу тих, що базуються на мікросервісному підході, Python демонструє низку технічних обмежень. Зокрема, глобальне блокування інтерпретатора (Global Interpreter Lock, GIL) ускладнює реалізацію ефективної багатопоточності, що обмежує можливості

масштабування системи у разі зростання навантаження. Крім того, реалізація захищених механізмів автентифікації та обробки даних нерідко вимагає використання сторонніх бібліотек, стабільність та підтримка яких може не забезпечувати повної відповідності вимогам до продуктивності, надійності та архітектурної гнучкості, які висуваються до розроблюваної серверної частини сучасних додатків [13].

PHP, як класична мова для WEB-розробки, довгий час залишалась одним із основних засобів створення серверних застосунків. Вона широко підтримується хостинг-провайдерами, має значну кількість навчальних матеріалів та активно використовується у проєктах невисокої складності. З-поміж сучасних фреймворків для PHP варто відзначити Laravel, який забезпечує зручні засоби для організації маршрутизації, валідації даних, підключення до баз даних та реалізації базових механізмів безпеки. Утім, у контексті побудови високонавантаженого WEB-додатка, орієнтованого на мікросервісну архітектуру, PHP демонструє низьку ефективність. Обмежена підтримка багатопоточності, відсутність суворої типізації та менш гнучка система управління залежностями ускладнюють реалізацію стабільної, масштабованої серверної логіки [14].

C# є сучасною мовою програмування, яка активно використовується у розробці корпоративних рішень у середовищі платформи .NET. У поєднанні з фреймворком ASP.NET Core ця мова забезпечує створення високопродуктивних WEB-додатків із підтримкою RESTful-сервісів, розвиненої системи авторизації, інтеграції з базами даних через ORM-рішення (зокрема Entity Framework) та підтримки шаблонів проєктування. Статична типізація, об'єктно-орієнтована модель, багатопотокова обробка запитів робить C# потужним інструментом для створення серверної логіки. Попри згадані переваги, ключовим обмеженням є історична орієнтованість екосистеми .NET на інфраструктуру Windows, що, незважаючи на появу кросплатформенного .NET Core, досі створює бар’єри при розгортанні застосунків у середовищах Linux. Окрім того, низка інструментів і сервісів

потребує ліцензування, що може збільшити вартість розробки й обслуговування у довгостроковій перспективі [15].

У таблиці 3.1 представлено порівняльний аналіз мов програмування, які розглядаються для розробки серверної частини WEB-додатка. Символ “+” використано для позначення переваги, тим часом як “–” свідчить про суттєвий недолік, а “+/–” вказує на середній рівень або компромісний варіант.

Таблиця 3.1 – Порівняльний аналіз мов програмування, які розглядаються для розробки серверної частини WEB-додатка

Критерій	Java	Python	PHP	C#
Підтримка мікросервісної архітектури	+	+/–	–	+
Підтримка багатопоточності	+	–	–	+
Статична типізація	+	–	–	+
Кросплатформність	+	+	+	+/–
Розвинена екосистема та інструменти	+	+	+	+
Стабільність та підтримка	+	+/–	+	+
Вартість розробки та обслуговування	+	+	+	–
Простота синтаксису	–	+	+	+/–

З огляду на проведений аналіз можна зробити висновок, що найбільш відповідною мовою програмування для реалізації серверної частини розроблюваного WEB-додатка є Java. Її переваги, зокрема висока продуктивність, підтримка мікросервісної архітектури, багатий інструментальний стек, стабільність, активна спільнота та широка документаційна база, забезпечують ефективне виконання усіх технічних

вимог до програмного рішення. Незначні недоліки, пов’язані з відносною складністю синтаксису, є прийнятними з огляду на масштаб та довготривалу перспективу розвитку проєкту, що робить вибір Java цілком обґрунтованим.

3.2 Обґрунтування вибору технологій розробки серверної частини WEB-додатка

У межах реалізації серверної логіки оберемо фреймворк, який забезпечить відповідність вимогам, поставленим в межах розроблюваного проєкту. Враховуючи обраний технологічний стек на основі Java, доцільним є застосування фреймворку Spring – одного з найпотужніших та найбільш гнучких фреймворків для створення WEB-застосунків на Java. Його структура дозволяє гнучко компонувати окремі функціональні блоки системи, створювати незалежні модулі бізнес-логіки та ефективно розподіляти відповідальність між компонентами.

Spring забезпечує вбудовану підтримку побудови RESTful API, реалізації шаблонів MVC, конфігурації доступу до баз даних, керування транзакціями, кешування та безпеки. Завдяки модулю Spring Security реалізується гнучка система автентифікації та авторизації, а компоненти Spring Data дозволяють ефективно взаємодіяти з реляційними й нереляційними СУБД. Однією з особливостей Spring є можливість програмної конфігурації, що сприяє адаптації до різних проєктних сценаріїв – від монолітних архітектур до мікросервісних систем. [16].

Організація зберігання та доступу до даних є не менш важливим аспектом проєктування серверної частини, тож розглянемо цю складову технічного стеку бекенд-розробки. У WEB-додатка для менеджменту задач різного типу як систему управління базами даних доцільно обрати MySQL – реляційну СУБД із відкритим вихідним кодом, що відзначається стабільністю, високою продуктивністю та широкою підтримкою в Java-екосистемі. MySQL забезпечує підтримку транзакцій, складних запитів, індексування, а також

легко інтегрується з ORM-фреймворками та бібліотеками доступу до даних [17]. Простота адміністрування, доступна документація та сумісність із Spring Data сприяють ефективній роботі з MySQL у межах розробки корпоративного WEB-застосунку [18].

Для забезпечення автоматизованого, контрольованого та повторюваного керування схемою бази даних доцільним є використання інструмента Liquibase. Цей засіб дозволяє впроваджувати зміни в структуру бази даних на основі changelog-файлів. Завдяки цьому можлива безпечна міграція між версіями структури даних, відстеження змін і уникнення конфліктів при колективній розробці. Підтримка інтеграції із Spring та CI/CD-середовищами дозволяє ефективно впроваджувати Liquibase у процес розробки та супроводу додатка [19].

Оскільки робота WEB-додатка передбачає частий доступ до ідентичних або комплексних для обробки даних, застосування механізмів кешування є необхідною умовою оптимізації продуктивності. Для L2 кешування на рівні окремих мікросервісів використовується бібліотека Caffeine Cache. Вона забезпечує високошвидкісне кешування в оперативній пам’яті з можливістю гнучкого налаштування терміну життя об’єктів, а також підтримки асинхронного оновлення. Її інтеграція із Spring дозволяє легко реалізувати кешування результатів методів, запитів до бази даних та API-викликів, що суттєво знижує навантаження на ресурси [20].

У випадках, коли необхідне спільне кешування між кількома мікросервісами або збереження сесій, застосовується високопродуктивне сховище ключ-значення Redis, що функціонує в оперативній пам’яті. Redis забезпечує надзвичайно швидкий доступ до даних, підтримує структури типу список, множина, хеш, черга, а також має вбудовані механізми TTL (time to live). Його використання доцільне для реалізації авторизаційних токенів, лімітів запитів, кешування API-відповідей або зберігання проміжних результатів між компонентами системи. У зв’язку з цим Redis виступає ключовим компонентом у забезпеченні масштабованості та швидкодії

мікросервісного середовища [21].

Одним із критично важливих аспектів безпеки в сучасних WEB-додатках є реалізація механізмів автентифікації та авторизації. Для цього використовується підхід на основі JWT (JSON Web Token). У рамках обраної архітектури застосовується відповідна бібліотека для генерації та перевірки токенів, що передаються між клієнтом і сервером. Використання JWT забезпечує розмежування доступу до ресурсів, захист від несанкціонованих дій, а також зручну масштабованість у розподіленому середовищі без необхідності зберігати сесії на сервері. У поєднанні із Spring Security реалізація JWT дозволяє гнучко керувати ролями користувачів та дозволами на рівні мікросервісів [22].

Доступ до бази даних у Spring-застосунках традиційно здійснюється через об'єктно-реляційне відображення. У межах даного проєкту розглядаються два поширені підходи, а саме Hibernate та JOOQ. Hibernate як складова Spring Data забезпечує повноцінну реалізацію ORM, з автоматичним формуванням SQL-запитів на основі анотованих класів, керуванням транзакціями та життєвим циклом об'єктів. Це полегшує роботу з базою даних, скорочує обсяг коду та зменшує ймовірність помилок при формуванні запитів [23]. У свою чергу, JOOQ (Java Object Oriented Querying) надає функціонал для написання SQL-запитів, дозволяючи при цьому зберігати повний контроль над структурою запитів та їх оптимізацією [24]. Обидва інструменти можуть бути використані залежно від складності бізнес-логіки, а саме Hibernate актуальний для швидкої розробки типових CRUD-операцій, а JOOQ – у випадках, що вимагають підвищеної гнучкості та оптимізації запитів.

У мікросервісній архітектурі важливу роль відіграє внутрішня міжсервісна комунікація. Для її реалізації доцільним є використання клієнта HTTP-запитів Feign, який дозволяє спростити взаємодію між сервісами, шляхом застосування REST-викликів. Його застосування дозволяє досягти високої читабельності коду, мінімізувати кількість шаблонних реалізацій, а також покращити контроль над міжсервісною взаємодією без додаткових

залежностей [25].

З метою забезпечення стабільного функціонування програмної системи необхідним є впровадження як мануального, так і автоматизованого тестування, яке охоплює перевірку працездатності REST API, валідацію вхідних даних, моделювання навантажень та інші критично важливі аспекти, що впливають на надійність взаємодії між компонентами.

Серед найбільш ефективних інструментів для виконання поставлених задач варто виділити Postman (рис. 3.1).

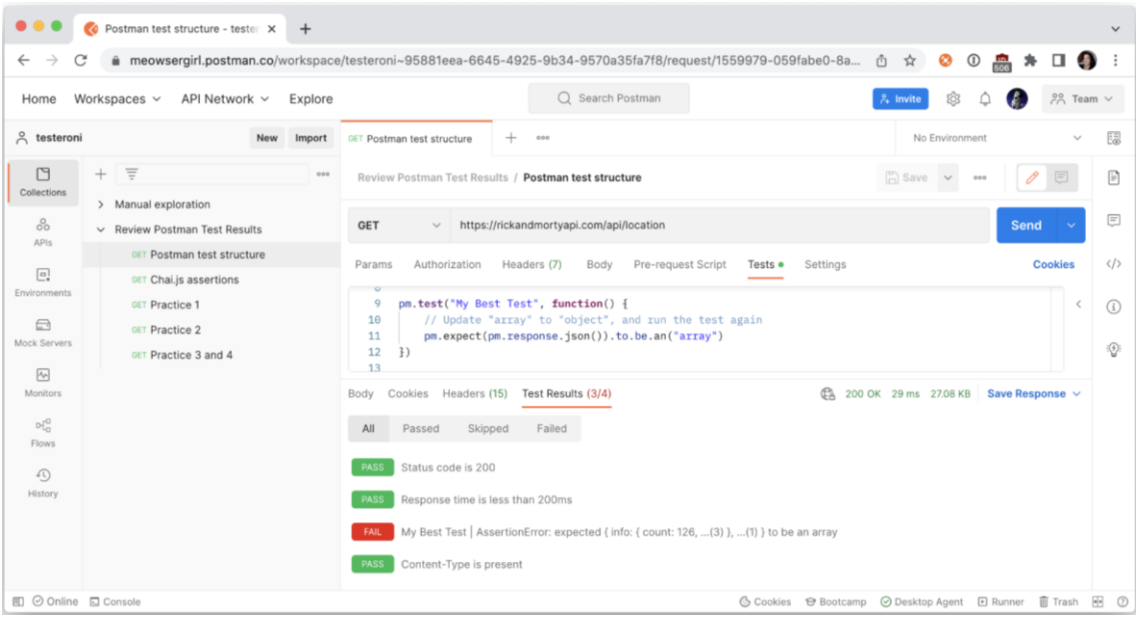

Рисунок 3.1 – Вікно додатка Postman, у процесі створення тесту

Цей програмний засіб забезпечує створення колекцій HTTP-запитів, фіксацію відповідей сервера, перевірку коректності реалізованого API, налаштування змінних середовища, заголовків та авторизаційних токенів, а також дозволяє моделювати типові сценарії взаємодії між клієнтом і сервером. Крім того, Postman підтримує автоматизацію тестування, що дає змогу забезпечити повторюваність тестових кейсів і виявляти помилки на ранніх етапах розробки. Інтеграція з процесами CI/CD дозволяє використовувати Postman як частину конвеєра безперервної перевірки функціональності API після кожного оновлення системи [26].

Для реалізації серверної частини у рамках визначеного технологічного стеку для реалізації системи було обрано інтегроване середовище розробки IntelliJ IDEA (рис. 3.2). Це середовище надає потужний набір інструментів для роботи з Java, включно з автоматичним завершенням коду, рефакторингом, перевіркою помилок у режимі реального часу, підтримкою роботи з системами контролю версій, контейнерами, зовнішніми сервісами, а також фреймворком Spring. IntelliJ IDEA оптимізує процес навігації між класами, конфігураціями, шаблонами запитів і ресурсами, що суттєво підвищує продуктивність команди розробників. Завдяки широкій функціональності та стабільності роботи, це середовище є логічним вибором для реалізації складної та масштабованої серверної інфраструктури [27].

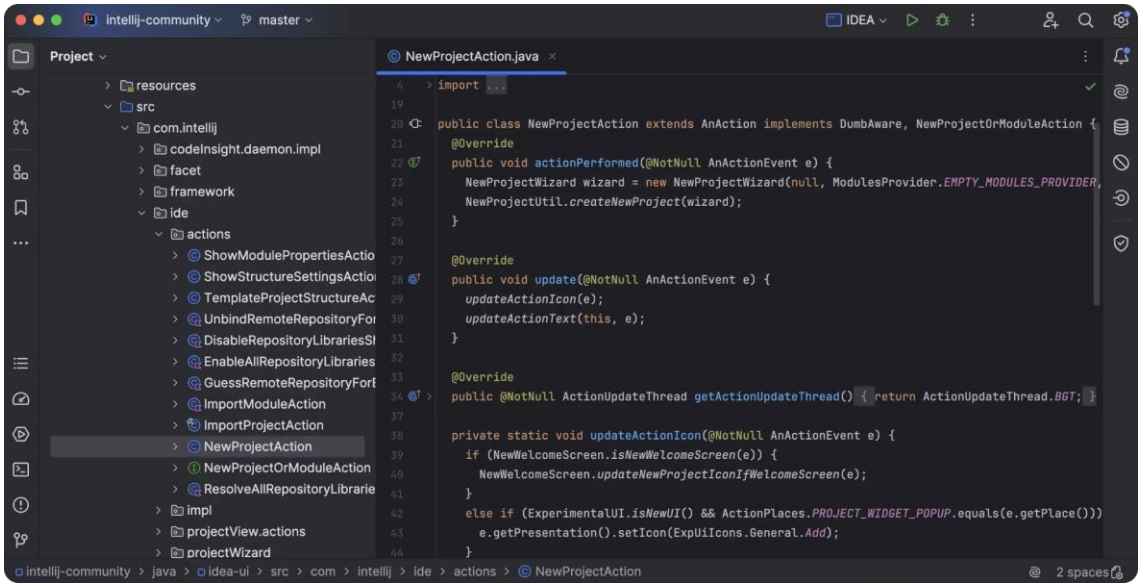

Рисунок 3.2 – Вікно інтегрованого середовища розробки IntelliJ IDEA

3.3 Обґрунтування вибору технологій розробки клієнтської частини WEB-додатка

Проєктування клієнтської частини WEB-додатка є важливим етапом у створенні цілісної програмної системи, оскільки саме цей компонент визначає, яким чином користувач взаємодіє з функціональністю додатка. Основними вимогами до інтерфейсу системи є зручність, адаптивність до різних

пристроїв, мінімальна затримка при завантаженні, зрозумілий візуальний стиль, а також відповідність принципам UX/UI. Вибір технологій реалізації клієнтської частини повинен враховувати масштаб проєкту, складність функціональності, можливості розширення та простоту підтримки.

Серед сучасних підходів до розробки клієнтських частин WEB-застосунків домінують такі два напрямки: використання сукупності технологій HTML, CSS, JavaScript разом із бібліотеками для верстки, або застосування фреймворків для створення односторінкових застосунків (SPA), найпопулярнішим серед яких є Angular. Кожен з підходів має свої переваги та обмеження, які необхідно враховувати у контексті конкретного проєкту.

Angular – це потужний фреймворк, розроблений Google, який орієнтований на створення динамічних WEB-застосунків (рис. 3.3).

Рисунок 3.3 – Логотип фреймворку Angular

Він реалізує архітектуру MVC, має вбудовану підтримку компонентного підходу, маршрутизації, форм, двостороннього зв’язування даних, а також модулі для керування залежностями та роботи з REST API. Angular дозволяє реалізовувати масштабні інтерфейси зі складною логікою, забезпечуючи високу структурованість коду [28]. Проте, попри переваги, цей фреймворк має низку недоліків: складність початкового налаштування, потреба у навичках роботи з TypeScript, більший обсяг завантажуваного коду, що впливає на швидкість старту застосунку. Angular доцільно використовувати у великих системах з великою кількістю динамічних компонентів і взаємозалежностей, однак для проєктів середнього масштабу його використання може бути

надмірним.

Альтернативою цьому є застосування WEB-стека, що складається з HTML (мова розмітки), CSS (стилізація) і JavaScript (реалізація динамічної взаємодії), доповненого фреймворком Bootstrap (рис. 3.4) для створення адаптивної верстки.

HTML дозволяє структуровано описати вміст сторінки, застосування CSS має на меті визначити зовнішній вигляд елементів, тим часом як JavaScript виконує обробку подій, зміну DOM та асинхронну взаємодію з сервером. Такий підхід забезпечує простоту в реалізації, високу сумісність із браузерами та мінімальні вимоги до ресурсів [29].

Рисинок 3.4 – Логотип фреймворку Bootstrap

В свою чергу, Bootstrap покликаний для прискорення процесу розробки інтерфейсу завдяки готовим елементам, що є простими в імплементації. Даний фреймворк дозволяє досягти адаптивного дизайну без перевантаження великою кількістю CSS-рішень, що робить його поєднання з іншими інструментами ефективним рішенням [30].

Отже, з урахуванням поставленого завдання – створити інтуїтивно зрозумілий, адаптивний і легкий в реалізації інтерфейс без надмірної складності – було обрано саме комбінацію HTML, CSS, JavaScript і Bootstrap. Такий вибір обумовлений низкою переваг порівняно з Angular, а саме нижчий рівень складності, краща швидкодія при початковому завантаженні,

відсутність потреби у додаткових компіляторах або складному налаштуванні середовища розробки. Обраний стек дозволяє швидко створювати прототипи, зручно працювати з REST API, підтримувати адаптивність та масштабованість при збереженні простоти.

3.4 Висновок

У даному розділі виконано порівняльний аналіз найбільш поширених мов програмування для реалізації серверної частини. Аргументовано доцільність вибору Java, яка забезпечує високу продуктивність, підтримку мікросервісної архітектури, стабільність, масштабованість і відповідність технічним вимогам проєкту.

Обґрунтовано вибір технологій серверної частини: Spring як гнучкого фреймворку для створення REST API, MySQL як надійної реляційної СУБД, а також інструментів для роботи з кешем, базами даних, безпекою та міжсервісною взаємодією. Для забезпечення надійного тестування API обрано Postman, а інтегроване середовище розробки IntelliJ IDEA покликане для ефективної реалізації та супроводу системи. Такий стек забезпечує узгоджену роботу бекенду в умовах високих навантажень.

Проаналізовано підходи до реалізації клієнтської частини та обрано поєднання HTML, CSS, JavaScript і Bootstrap як раціональне рішення. Такий вибір дозволяє створити простий в обслуговуванні, адаптивний і швидкодійний інтерфейс, що забезпечує комфортну взаємодію користувача з функціоналом застосунку.

4 ТЕХНІЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО РІШЕННЯ ТА ТЕСТУВАННЯ МЕХАНІЗМІВ ЙОГО РОБОТИ

4.1 Технічна реалізація WEB-додатка

Функціональність розроблюваного WEB-додатка для менеджменту задач різного типу охоплює повний цикл роботи із задачами: створення, редагування, перегляд, видалення та зміна статусу. Реалізовано фільтрацію задач за командами, пріоритетом і статусом виконання, а також пошук за атрибутами, наприклад, назвою задачі, ім’ям користувача. Основна логіка реалізована у вигляді окремих мікросервісів – сервісу управління задачами та сервісу автентифікації, що дозволяє масштабувати систему відповідно до навантаження.

Окрім реалізації базового функціоналу, у межах дослідження запропоновано низку покращень, що розширюють функціональні можливості додатка порівняно з існуючими аналогами. Розглянемо одне з таких рішень – модуль вбудованої аналітики. Він забезпечує автоматизований збір статистичних показників щодо кількості виконаних задач і виявлених багів. Дані агрегуються за визначені часові інтервали: шість місяців, три місяці, один місяць, два тижні та один тиждень. Цей механізм дозволяє керівникам команд оцінювати динаміку продуктивності, своєчасно виявляти проблемні ділянки та приймати обґрунтовані рішення щодо розподілу ресурсів. На рисунку 4.1 та 4.2 наведено програмний код, який реалізує відображення статистичних даних команди.

```
private TeamTasksMetricsEntity getTeamTasksMetrics(String teamName) {
    return metricsTasksService.getTasksMetricsByTeamName(teamName);
}
```

Рисуглк 4.1 – Отримання даних для відображення статистики певної команди

```
public Result<Record> fetchBugCompletedTasksWithPeriod(String teamName) {
    LocalDate now = LocalDate.now();
    Record5<String, Field<?>, LocalDate, Integer, Integer>[] periods = new Record5[] {
            row("1_week", TABLE.end_date, now.minusDays(7), 1, 1),
            row("2_weeks", TABLE.end_date, now.minusDays(14), 2, 2),
            row("1_month", TABLE.end_date, now.minusMonths(1), 3, 3),
            row("3_months", TABLE.end_date, now.minusMonths(3), 4, 4),
            row("6_months", TABLE.end_date, now.minusMonths(6), 5, 5)
    };
    SelectSelectStep<Record> unioned = null;

    for (Record5<String, Field<?>, LocalDate, Integer, Integer> period : periods) {
        SelectConditionStep<Record> subquery = dsl
                .select(inline(period.value1()).as("period"),TABLE.asterisk())
                .from(TABLE)
                .where(TABLE.type.eq(TaskType.BUG))
                .and(TABLE.status.eq(TaskStatus.COMPLETED))
                .and(TABLE.end_date.greaterOrEqual(period.value3()))
                .and(TABLE.assigned_team.eq(teamName));

        if (unioned == null) {
            unioned = subquery;
        } else {
            unioned = ((SelectConditionStep<Record>) unioned).unionAll(subquery);
        }
    }
    Table<?> sub = unioned.asTable("sub");
    CaseConditionStep<Integer> periodOrder = DSL
            .when(field("period").eq("1_week"), 1)
            .when(field("period").eq("2_weeks"), 2)
            .when(field("period").eq("1_month"), 3)
            .when(field("period").eq("3_months"), 4)
            .when(field("period").eq("6_months"), 5)
            .otherwise(99);

    return dsl
            .selectDistinct(field("id"), field("period"), sub.fields())
            .from(sub)
            .orderBy(field("id"), periodOrder)
            .fetch();
}
```

Рисунок 4.2 – Запит до бази даних, що отримує дані для аналітики

Ще одним функціональним покращенням є реалізація централізованого профілю користувача з доступом до його контактної інформації, такої як номер телефону та електронна пошта. Це дозволяє учасникам команди налагоджувати безпосередню комунікацію, уникаючи додаткових сторонніх інструментів. Такий підхід підвищує прозорість взаємодії, економить час на пошук потрібних контактів та підтримує організаційну цілісність проєкту. Частину коду, що відповідає за цей функціонал, наведено на рисунку 4.3.

```java
public class UserInfoServiceImpl implements UserInfoService {

    private final UserInfoRepository repository;
    private final PasswordEncoder encoder;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        final Optional<UserInfo> userInfo = Optional.of(repository.findByEmail(username));
        return userInfo.map(UserInfoDetails::new)
                .orElseThrow(() -> new UsernameNotFoundException("User not found: " + username));
    }

    @Override
    public UserInfo createUser(UserInfo userInfo) {
        final String password = encoder.encode(userInfo.getPassword());
        userInfo.setPassword(password);
        return repository.save(userInfo);
    }

    @Override
    @CachePut(value = "UserCache", key = "#userInfo.id")
    public UserInfo updateUser(UserInfo userInfo) {
        final UserInfo userFromDb = repository.findByEmail(userInfo.getEmail());
        userInfo.setPassword(userFromDb.getPassword());
        return repository.save(userInfo);
    }

    @Override
    @CacheEvict(value = "UserCache", key = "#id")
    public void dropUserById(Integer id) {
        repository.deleteById(id);
    }

    @Override
    @Cacheable(value = "UserCache", key = "#id")
    public Optional<UserInfo> getUserById(Integer id) {
        System.out.println("get from db id = " + id);
        return repository.findById(id);
    }

    @Override
    public List<UserInfo> getUsersByIds(List<Integer> userIds) {
        return repository.findAllById(userIds);
    }

    @Override
    @Transactional
    public UserInfo toggleUserAccountById(Integer id, Boolean toggle) {
        if (repository.toggleUserAccount(id, toggle) > 0) {
            return getUserById(id).get();
        }
        throw new IllegalArgumentException("Id = " + id + ", toggle = " + toggle);
    }
}
```

Рисунок 4.3 – Набір методів, притаманних для сутності користувача

Для полегшення навігації системою реалізовано пошук не лише за задачами та виконавцями, а й за назвами команд. Ця функція є корисною в умовах великої кількості учасників і задач, дозволяє оперативно визначати

відповідальних осіб та структуру проєктних груп. Перелік методів для роботи зі структурою команди представлено на рисунку 4.4.

```java
package ua.edu.vntu.authservicedna.service.team;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import ua.edu.vntu.authservicedna.entity.TeamEntity;
import ua.edu.vntu.authservicedna.entity.user.UserInfo;
import ua.edu.vntu.authservicedna.repository.jpa.UserInfoRepository;

import java.util.List;

@Service("teamService")
@Slf4j
@RequiredArgsConstructor
public class TeamServiceImpl implements TeamService {

    private final UserInfoRepository repository;

    @Override
    public List<UserInfo> getTeamUsers(String name) {
        return repository.findByTeam(name);
    }

    @Override
    public List<TeamEntity> getTeamEntities() {
        return repository.findAllTeamsDetailed();
    }

    @Override
    public List<UserInfo> getTeamUsersWithoutUser(String email, String teamName) {
        return repository.findByTeamWithoutUser(email, teamName);
    }
}
```

Рисунок 4.4 – Набір методів, притаманних для сутності команди

Також впроваджено логіку гіперпосилань між задачами та командами: в описі задачі, окрім решти відповідних їй деталей, є активне посилання на сторінку команди, до якої ця задача належить, що сприяє кращому розумінню бізнес-процесів. На рисунку 4.5 подано фрагмент коду, що реалізує посилання на команду з інтерфейсу задачі.

```
<div class="row">
   <div class="col-sm-3">
       <p class="mb-0 fw-bold align-items-center text-center" style="margin-left:
-105px">Assignee</p>
   </div>
   <div class="col-sm-9">
       <a href="@{${assignee.link}}" style="text-decoration: none; display: flex; align-items:
center; gap: 8px;">
           <img src="@{/assets/img/{img}(img=${assignee.image})}" style="width: 30px; height:
auto; border-radius: 50%;">
           <p class="text-muted mb-0" style="margin: 0;" th:text="${assignee.name}"></p>
       </a>
   </div>
</div>
<br>
<div class="row">
   <div class="col-sm-3">
       <p class="mb-0 fw-bold align-items-center text-center" style="margin-left:
-105px">Reporter</p>
   </div>
   <div class="col-sm-9">
       <a href="@{${reporter.link}}" style="text-decoration: none; display: flex; align-items:
center; gap: 8px;">
           <img src="@{/assets/img/{img}(img=${reporter.image})}" style="width: 30px; height:
auto; border-radius: 50%;">
           <p class="text-muted mb-0" style="margin: 0;" th:text="${reporter.name}"></p>
       </a>
   </div>
</div>
<br>
<div class="row">
   <div class="col-sm-3">
       <p class="mb-0 fw-bold align-items-center text-center" style="margin-left:
-115px">Team</p>
   </div>
   <div class="col-sm-9">
       <a href="@{${team.link}}" style="text-decoration: none; display: flex; align-items:
center; gap: 8px;">
           <p class="text-muted mb-0" style="margin: 0;" th:text="${team.name}"></p>
       </a>
   </div>
</div>
```

Рисунок 4.5 – Реалізація посилання на сторінку команди з інтерфейсу задачі

Реалізовані технічні рішення спрямовані на покращення зручності користування та підвищення ефективності процесів всередині команд. Сукупність запропонованих механізмів дозволяє позиціонувати створений WEB-додаток як ефективний інструмент управління задачами, здатний адаптуватися до потреб команд різного масштабу та структури.

4.2 Інструкції з розгортання та використання WEB-додатка

Сьогодні найбільш поширеними платформами для хмарного розгортання є Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, IBM Cloud та Oracle Cloud [31]. У таких середовищах активно застосовуються практики безперервної інтеграції та розгортання (CI/CD), що значно пришвидшують цикл поставки програмних продуктів.

Розгортанням систем у хмарі зазвичай займаються DevOps-фахівці, які володіють спеціалізованими знаннями та досвідом роботи з такими інструментами. Враховуючи обрані архітектурні підходи, для реалізації хмарного розгортання у межах даного проєкту було обрано платформу AWS, яка вирізняється серед аналогічних широкою функціональністю, високим рівнем надійності, масштабованістю та розвиненою екосистемою сервісів для розгортання мікросервісної архітектури .

Отже, щоб розпочати розгортання, спочатку необхідно увійти до облікового запису AWS та перейти до розділу Amazon EC2, де буде створено інстанс для розміщення сервісів. На цьому інстансі потрібно встановити Java та MySQL, які забезпечать базову інфраструктуру застосунку. Далі необхідно завантажити сервер для запуску Java-додатків Apache Tomcat із офіційного сайту та розгорнути його на EC2. Оскільки завантаження файлів напряму на EC2 може бути незручним, рекомендується скористатися Amazon S3 – масштабованим об'єктним сховищем, яке забезпечує високу доступність, безпеку та гнучкість у керуванні даними.

Проєкт компілюється за допомогою команди gradle build, яка створює .war-файл для подальшого розгортання. Цей файл разом із Tomcat можна завантажити у S3, а згодом розгорнути на EC2, перемістивши .war-файл до теки WEBapps. Для автоматизації цього процесу доцільно використовувати Jenkins – популярний інструмент з відкритим кодом для CI/CD. Він дозволяє автоматизувати всі етапи розгортання, від збирання коду до завантаження та запуску на сервері, з використанням відповідних плагінів.

Щоб почати роботу над проєктом локально, розробнику слід встановити IntelliJ IDEA – інтегроване середовище розробки від JetBrains. Далі необхідно інсталювати систему керування версіями Git, яка дозволяє зберігати історію змін та співпрацювати над кодом. Наступним кроком варто створити обліковий запис на GitHub та підключити його до організації, щоб отримати доступ до репозиторіїв.

Наступним важливим інструментом є AWS Toolkit, який інтегрується у IntelliJ IDEA та спрощує взаємодію з сервісами AWS. Для запуску додатка локально або створення середовища для тестування необхідно також встановити Docker, що є платформою для розгортання програми в ізольованих контейнерах разом із усіма залежностями.

З боку кінцевих користувачів, для використання WEB-додатка для менеджменту задач різного типу необхідна наявність електронної пошти та стабільного інтернет-з'єднання. Оскільки застосунок є кросплатформним, ним можна користуватися як на персональних комп’ютерах, так і на мобільних пристроях.

4.3 Системні вимоги до запуску WEB-додатка

Для забезпечення стабільної та коректної роботи WEB-додатка для менеджменту задач різного типу необхідно дотримуватись певних системних вимог. Системні вимоги визначають мінімальні та рекомендовані характеристики апаратного й програмного забезпечення, які дозволяють програмі функціонувати без збоїв на відповідних пристроях. Ці вимоги охоплюють операційні системи, типи браузерів, а також технічні параметри пристроїв, зокрема обсяг оперативної пам’яті, процесорну потужність тощо.
Для персональних комп’ютерів (Windows/Linux):

- WEB-додаток підтримує роботу в сучасних браузерах, включаючи Microsoft Edge, Google Chrome (версія 3.0.195 і вище), а також Internet Explorer 6.x і новіші;

- операційна система повинна бути не нижче Windows XP, рекомендується Windows 10 або новіші версії. Також повністю підтримується Linux із будь-якими сучасними дистрибутивами (Ubuntu, Fedora, Debian тощо);
- рекомендована конфігурація ПК: процесор AMD Ryzen або Intel i3 з тактовою частотою від 2,4 ГГц, не менше 4 ГБ оперативної пам’яті, стабільне підключення до мережі Інтернет.

Для комп’ютерів Apple (macOS):

- підтримуються браузери Safari (версія 3.0 і вище) та Microsoft Internet Explorer (версія 5.0 і вище, якщо використовується через емулятор або стороннє ПЗ);
- операційна система – macOS версії 10.4 або новіші.

Для мобільних пристроїв на Android:

- Мінімальною версією операційної системи є Android 6.0 (Marshmallow). Для коректної роботи застосунку рекомендується використовувати браузер Google Chrome версії 3.0.195 або вище. Застосунок адаптовано до широкого спектру екранів та пристроїв.

Для мобільних пристроїв на iOS:

- Необхідна операційна система iOS версії 5.0 або новіша. Рекомендується використовувати стандартний браузер Safari версії 3.0 або вище. Інтерфейс застосунку оптимізований для роботи як на iPhone, так і на iPad.

4.4 Автоматизоване тестування та підготовка сценаріїв мануального тестування WEB-додатка

Важливим є етап тестування WEB-застосунку. Це систематичний і

детальний процес перевірки функціональних можливостей, продуктивності, сумісності, безпеки та інших технічних аспектів WEB-додатка з метою виявлення помилок, затримки даних, дефектів або невідповідностей очікуваній поведінці. Такий підхід застосовується з метою запобігання серйозним проблемам ще до впровадження застосунку публічно, а також удосконалювати вже наявні компоненти системи.

У процесі перевірки функціональності та надійності WEB-додатка було обрано два основні підходи: автоматизоване та мануальне тестування. Така комбінація є доцільною, оскільки дозволяє комплексно оцінити якість розробленої системи: автоматизоване тестування забезпечує перевірку стабільності технічних компонентів і відповідність API-логіки очікуваній поведінці, тоді як мануальне дозволяє оцінити коректність роботи інтерфейсу та зручність взаємодії з додатком з позиції кінцевого користувача [32].

Для забезпечення високої якості коду під час розробки було застосовано інструменти статичного аналізу. Зокрема, використовувався плагін SonarLint, інтегрований у середовище розробки IntelliJ IDEA. Цей інструмент виявляє потенційні помилки, порушення правил безпеки, стилістичні недоліки та інші вразливості безпосередньо під час написання коду, що сприяє ранньому усуненню дефектів і зменшенню вартості подальшого тестування.

Для проведення автоматизованого тестування використовувався інструмент Postman, що дозволив здійснювати серії HTTP-запитів до серверної частини додатка. Зокрема, перевірялася коректність взаємодії з базою даних, відповідність форматів даних при виконанні CRUD-операцій, обробка параметрів фільтрації та сортування. Особливу увагу було приділено перевірці коректної конвертації вхідних і вихідних даних у JSON-форматі, а також стабільності відповідей у разі валідних і невалідних запитів.

Завдяки модульному тестуванню вдалося симулювати різні сценарії використання системи, перевірити граничні випадки та забезпечити очікувану поведінку окремих компонентів. На рисунку 4.6 наведено приклад простого тесту, що демонструє перевірку роботи одного з елементів бізнес-логіки WEB-

застосунку.

```
    @Test
    public void givenCorrectId_WhenGetDAORequest_ThenReturnUser() {
    private final UserInfo userInfo = userInfoSerivce.getUserInfoById(correctId);
    assertEquals(expectedUser, userInfo);
}
```

Рисунок 4.6 – Приклад сценарію автоматизованого тестування додатку

Мануальне (ручне) тестування програмного забезпечення є одним із найпростіших, але водночас важливих способів перевірки роботи застосунку. Воно полягає у послідовному виконанні дій користувача з метою виявлення функціональних помилок, логічних невідповідностей або збоїв в інтерфейсі. Попри свою базовість, цей метод дозволяє швидко виявити критичні проблеми, які могли залишитися непоміченими під час автоматизованого тестування.

З метою переконання у відповідності реалізованого функціоналу вимогам технічної документації було розроблено низку тестових сценаріїв, які охоплювали ключові аспекти роботи системи. Розглянемо ці сценарії.

Сценарій 1

Назва: перевірка повідомлення про помилку при некоректному форматі електронної адреси.

Передумови: користувач знаходиться на сторінці авторизації.

Кроки виконання:

- у поле електронної пошти вводиться значення, що не відповідає встановленому формату (наприклад, без символу @);
- у поле пароля вводиться довільне значення;
- натискається кнопка "Login".

Очікуваний результат: система генерує повідомлення про некоректний формат електронної адреси.

Сценарій 2

Назва: вхід без заповнення пароля.

Передумови: користувач перебуває на сторінці входу в систему.

Кроки виконання:

- вводиться коректна електронна адреса;
- поле пароля залишається порожнім;
- натискається кнопка "Login".

Очікуваний результат: виводиться повідомлення про необхідність введення пароля.

Сценарій 3

Назва: спроба входу незареєстрованого користувача.

Передумови: користувач знаходиться на сторінці входу.

Кроки виконання:

- вводиться електронна адреса, яка не зареєстрована в системі;
- вводиться будь-який пароль;
- натискається кнопка "Login".

Очікуваний результат: система інформує про відсутність облікового запису.

Сценарій 4

Назва: введення неправильного пароля для зареєстрованого користувача.

Передумови: користувач має зареєстрований обліковий запис.

Кроки виконання:

- вводиться правильна електронна адреса;
- вводиться неправильний пароль;
- натискається кнопка "Login".

Очікуваний результат: з’являється повідомлення про невірні облікові дані.

Сценарій 5

Назва: перегляд інформації у персональному кабінеті.

Передумови: Користувач успішно авторизований.

Кроки виконання:

- відкривається сторінка персонального кабінету.

Очікуваний результат: відображаються актуальні персональні дані, контактна інформація, а також кнопки “Update info” і “Log out”.

Сценарій 6

Назва: зміна персональних даних користувача.

Передумови: користувач перебуває у власному профілі.

Кроки виконання:

- натискається кнопка “Update info”;
- вводяться зміни до відповідних полів.
- натискається кнопка “Save”.

Очікуваний результат: дані оновлюються та зберігаються; зміни відображаються у профілі.

Сценарій 7

Назва: пошук задачі за назвою.

Передумови: користувач перебуває на сторінці пошуку задач.

Кроки виконання:

- вводиться точна назва задачі в пошукове поле;
- виконується пошук.

Очікуваний результат: задача з відповідною назвою з’являється в результатах.

Сценарій 8

Назва: перехід до задачі зі списку результатів пошуку.

Передумови: задача відображена у результатах пошуку.

Кроки виконання:

- користувач натискає на назву задачі у колонці “Ticket name”.

Очікуваний результат: відкривається сторінка з деталізованою інформацією про задачу.

Сценарій 9

Назва: перехід на сторінку команди з інтерфейсу задачі.

Передумови: користувач переглядає деталі задачі.

Кроки виконання:

- натискається назва команди, яка пов’язана з поточною задачею.

Очікуваний результат: відкривається сторінка команди з інформацією про її склад та контактні дані.

Сценарій 10

Назва: редагування даних задачі.

Передумови: користувач перебуває на сторінці задачі.

Кроки виконання:

- натискається кнопка “Update info”;
- вносяться необхідні зміни в поля форми.

Натискається кнопка “Save”.

Очікуваний результат: дані задачі оновлюються та коректно відображаються після збереження.

Сценарій 11

Назва: додавання коментаря до задачі.

Передумови: користувач знаходиться у секції коментарів до задачі.

Кроки виконання:

- вводиться текст у поле коментаря;
- натискається кнопка “Submit”.

Очікуваний результат: коментар з’являється у відповідному розділі інтерфейсу задачі.

Сценарій 12

Назва: перегляд інформації про команду.

Передумови: користувач знаходиться на сторінці команди.

Кроки виконання:

- переглядається список учасників та ролі;
- ознайомлення з контактною інформацією.

Очікуваний результат: відображається повна інформація про склад команди.

Сценарій 13

Назва: перегляд статистики виконаних задач і багів.

Передумови: користувач знаходиться у розділі статистики.

Кроки виконання:

- обирається часовий інтервал (6 міс., 3 міс., 1 міс., 2 тижні, 1 тиждень);
- ознайомлення з діаграмами та графіками.

Очікуваний результат: Відображається релевантна аналітична інформація відповідно до обраного фільтра.

4.5 Мануальне тестування WEB-додатка

У ході тестування було здійснено перевірку роботи сторінки “Вхід” та процесу “Створення завдання”. Тестування сторінки “Вхід” передбачає перевірку коректності відображення форм і правильності їх функціонування при введенні даних. Перевірка процесу створення завдання включає послідовне заповнення даних, а також відповідність відображення введеної інформації.

Для входу в систему користувач має ввести логін (електронну адресу) і пароль. У разі некоректного введення електронної адреси система виведе відповідне повідомлен ня про помилку (рис. 4.7).

У випадку, якщо під час спроби автентифікації користувач не введе пароль, система автоматично згенерує повідомлення про помилку. Це

повідомлення інформує про необхідність заповнення відповідного поля для продовження процесу входу в систему. Приклад відображення такої помилки наведено на рисунку 4.8.

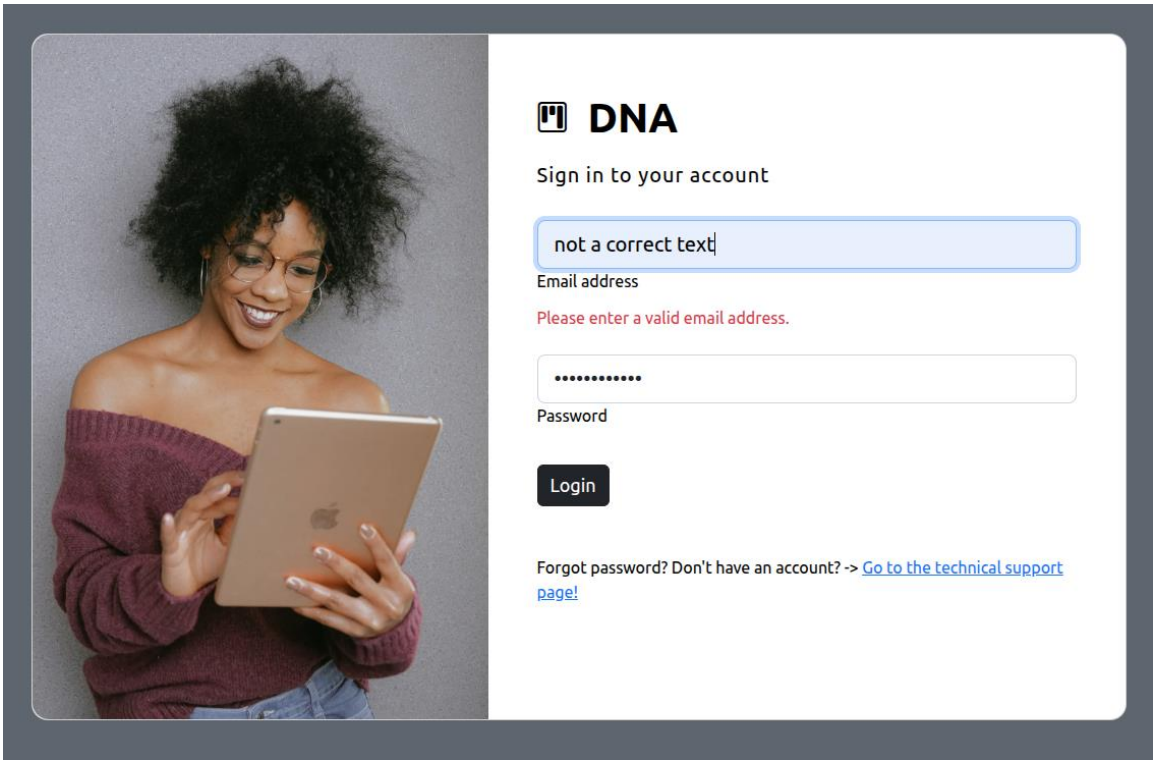

Рисунок 4.7 – Помилка при некоректному введенні електронної адреси

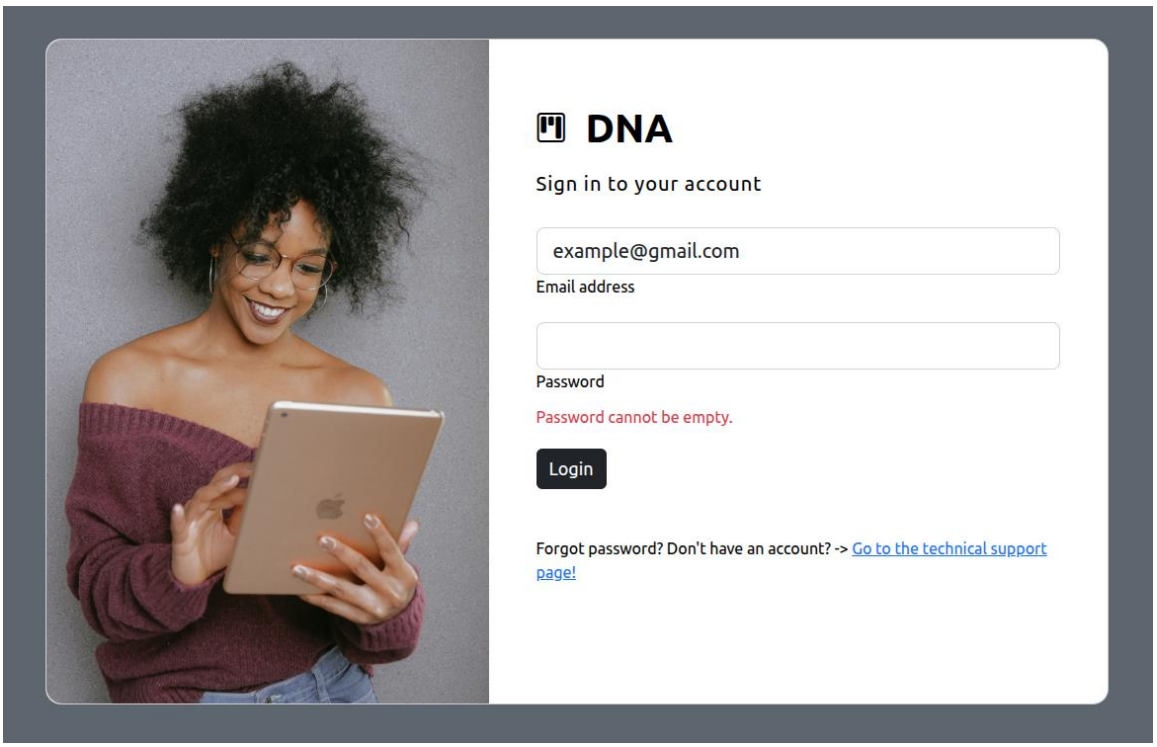

Рисунок 4.8 – Повідомлення про помилку при відсутності пароля

Також необхідно перевірити, що у випадку спроби авторизації користувачем, який не зареєстрований у системі, буде коректно відображено відповідне повідомлення про помилку. Це повідомлення має чітко інформувати користувача про відсутність облікового запису в системі, щоб уникнути непорозумінь і забезпечити зрозумілу комунікацію. Таке повідомлення сприяє покращенню користувацького досвіду, допомагаючи користувачу вчасно виявити помилку та здійснити необхідні дії, наприклад, перевірити правильність введення даних або перейти до реєстрації нового акаунту. Приклад відображення такого повідомлення наведено на рисунку 4.9.

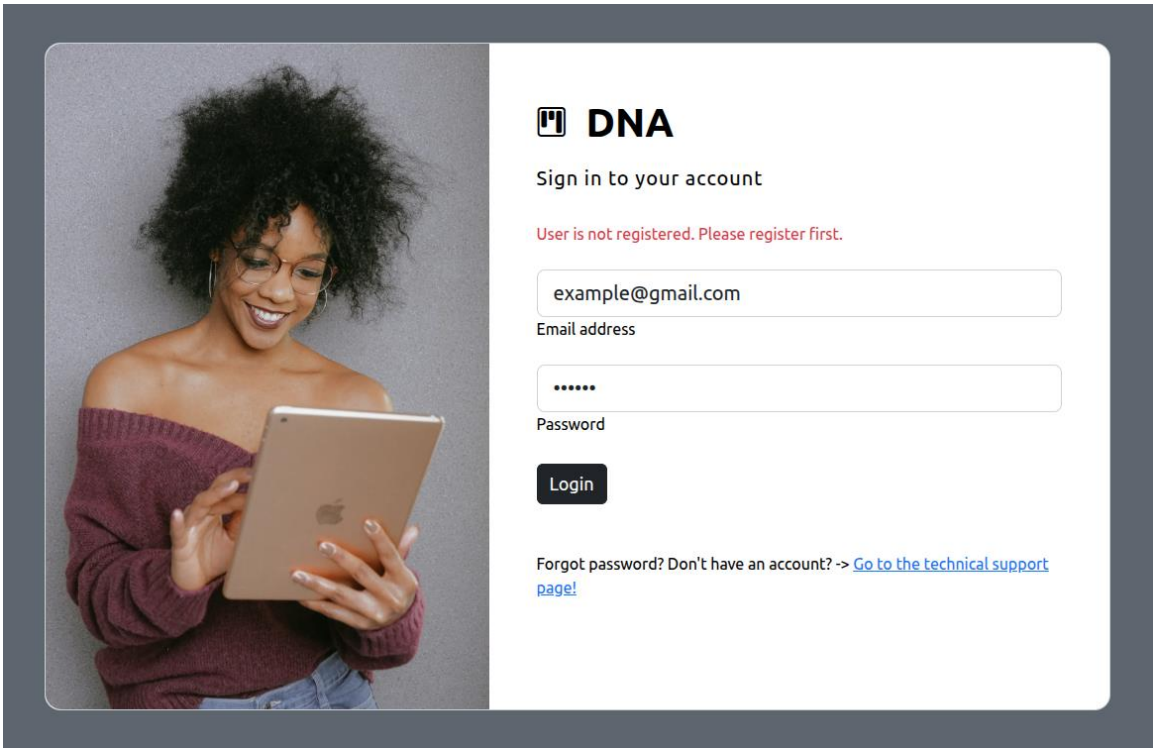

Рисунок 4.9 – Повідомлення про помилку при відсутності користувача у системі

Важливо переконатися, що у разі введення неправильного пароля для вже зареєстрованого користувача система коректно відображає відповідне повідомлення про помилку. Це повідомлення має чітко інформувати користувача про невідповідність введених даних та допомагати уникнути подальших помилок при авторизації. Завдяки цьому користувач може

повторно ввести пароль без зайвих труднощів або скористатися функцією відновлення пароля, якщо він забув свої облікові дані. Приклад відображення такого повідомлення наведено на рис. 4.10.

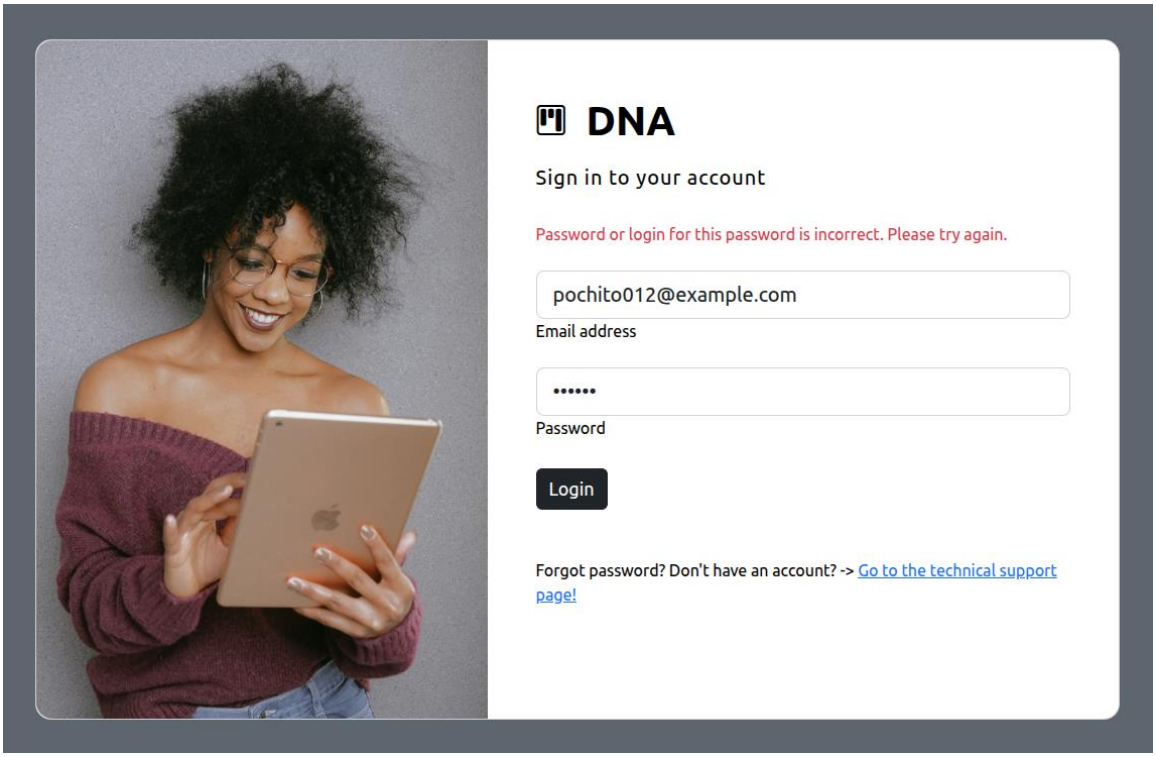

Рисунок 4.10 – Повідомлення про помилку при введених некоректності даних

Після успішної авторизації в системі користувач отримує доступ до сторінки особистого кабінету, де має можливість переглянути свої персональні дані. Це дозволяє переконатися в актуальності та правильності збереженої інформації. Приклад відображення особистого кабінету наведено на рисунку 4.11. Варто зауважити, що оприлюднена в профілі користувача інформація, зокрема контактні дані, є доступною членам його команди, що користувачеві варто враховувати під час роботи в додатку.

На даному зображенні можна побачити заагальну інформацію про користувача, а також кнопки “Update info” та “Log out”. Перша відповідає за зміну персональних даних користувача, а друга дозволяє вийти із системи і повернутись до сторінки входу у систему.

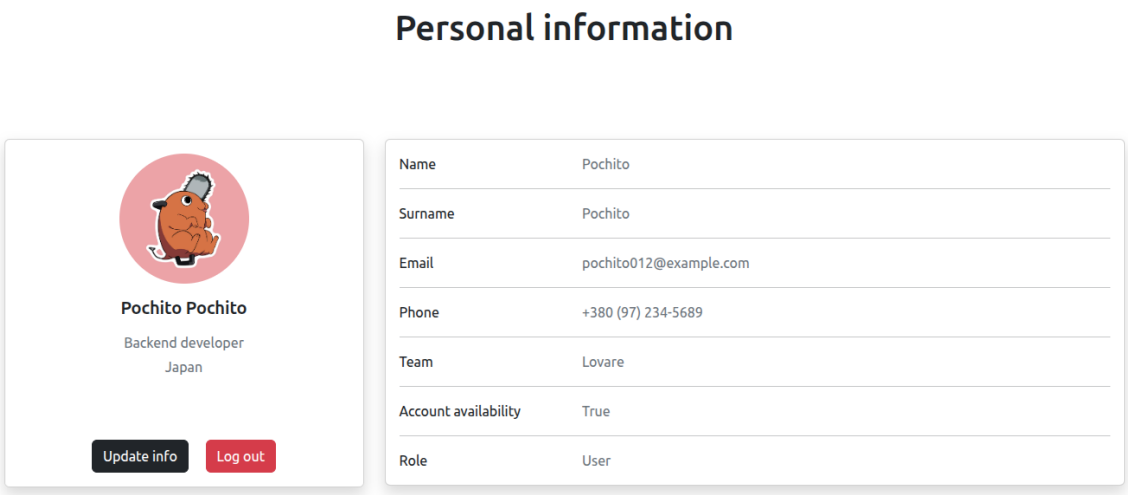

Рисунок 4.11 – Персональний кабінет користувача

У випадку, якщо користувач помічає, що деякі дані є неточними або застарілими, він може скористатися функцією редагування, натиснувши кнопку “Update info”. Після цього відкриється спеціальна форма, яка дозволяє змінити персональні відомості користувача Ілюстрацію функціоналу редагування наведено на рисунку 4.12.

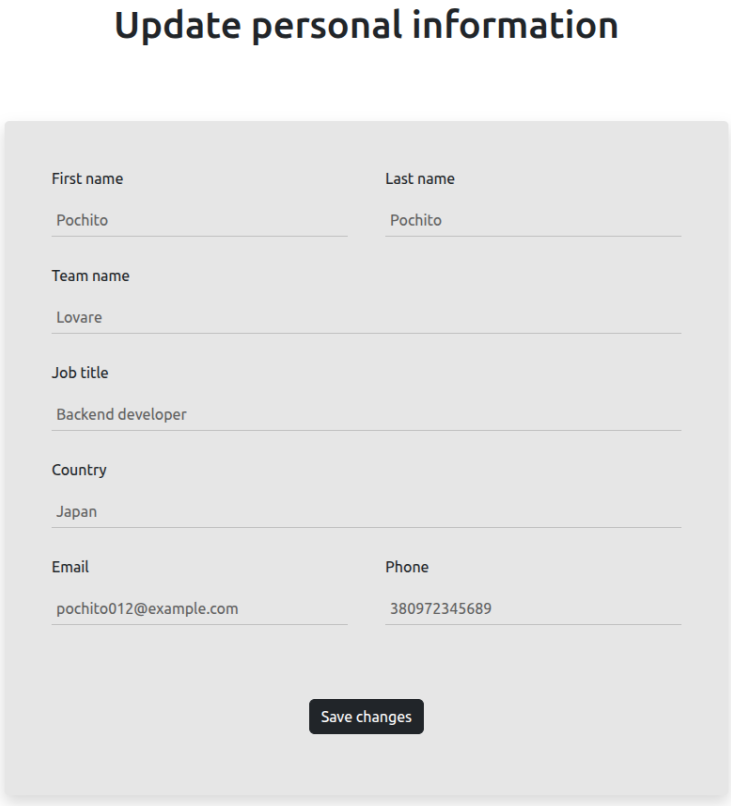

Рисунок 4.12 – Зміна персональних даних користувача

Перейдемо до роботи із задачами. На рис. 4.13 зображена форма для пошуку завдання. Знайти потрібну нам інформацію допоможе пошукове поле, у яке ми вводимо назву завдання для його подальшого опрацювання. Подібним чином виконується пошук за іншими атрибутами, серед яких пошук за ім'ям користувача та нововведення порівняно з аналогами, а саме функція пошуку за назвою команди.

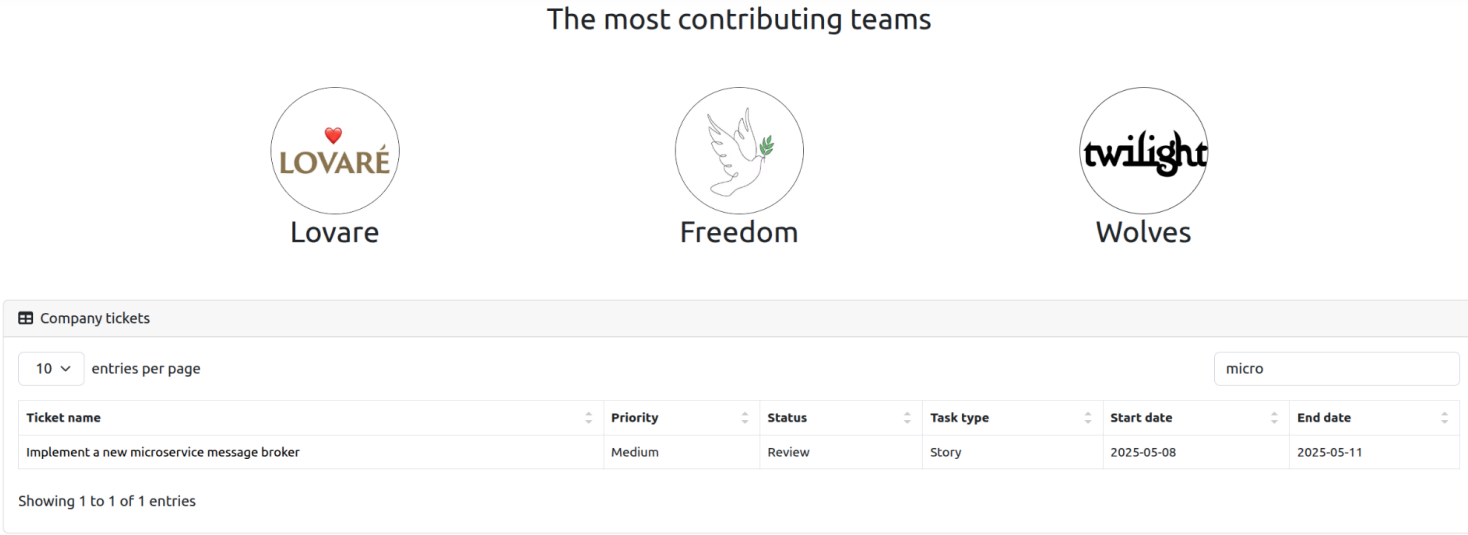

Рисунок 4.13 – Сторінка пошуку завдань за назвою

Після процедури пошуку потрібного завдання ми можемо перейти до нього через посилання в атрибуті “Ticket name”. Інтерфейс відображення завдання зображений на рис. 4.14.

Як основні атрибути, що описують дані для опрацювання, були визначені: опис, дата початку та завершення, пріоритет, поточний статус, тип, виконавець, автор заповнення даних, а також пов’язані задачі, що стосуються відповідної області роботи. Додатково, в деталі задачі було додано функціональне покращення порівняно з аналогами – зв'язок між командою та задачею, що за нею закріплена. Даний зв'язок реалізований у вигляді гіперпосилання на команду з інтерфейсу задачі, що дозволяє чітко визначити зону відповідальності в компанії, а також створює простір для можливостей подальшого дослідження даної команди, як от її складу, інших задач тощо.

Якщо виникає ситуація, коли користувач побачив неточність у викладеній інформації, чи потрібно змінити статус завдання, можна скористатись функцією зміни деталей завдання за допомогою кнопки “Update info”. На рисунку 4.15 зображено те, яким чином буде відбуватись процес зміни даних для завдання.

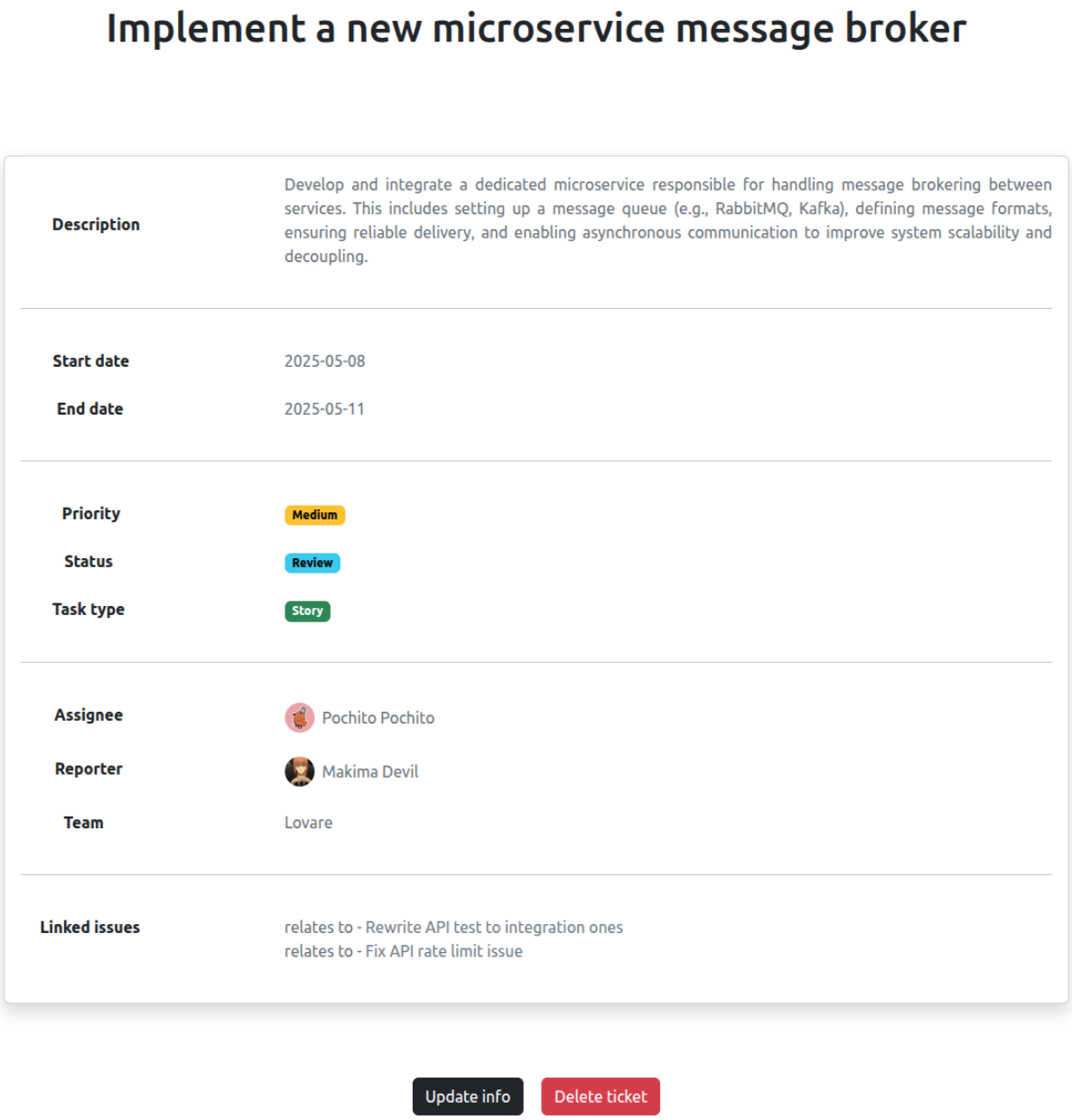

Рисунок 4.14 – Сторінка із інформацією про завдання

Розглянемо роботу секції коментарів. Користувачі можуть обговорювати ту чи іншу проблемну область в коментарі до певної задачі чи

ділитись додатковою інформацією, що може стати у нагоді під процесу виконання. Також користувач має можливість редагувати чи видаляти коментар, який було залишено під завданням. Зображення секції коментарів наведено на рисунку 4.16.

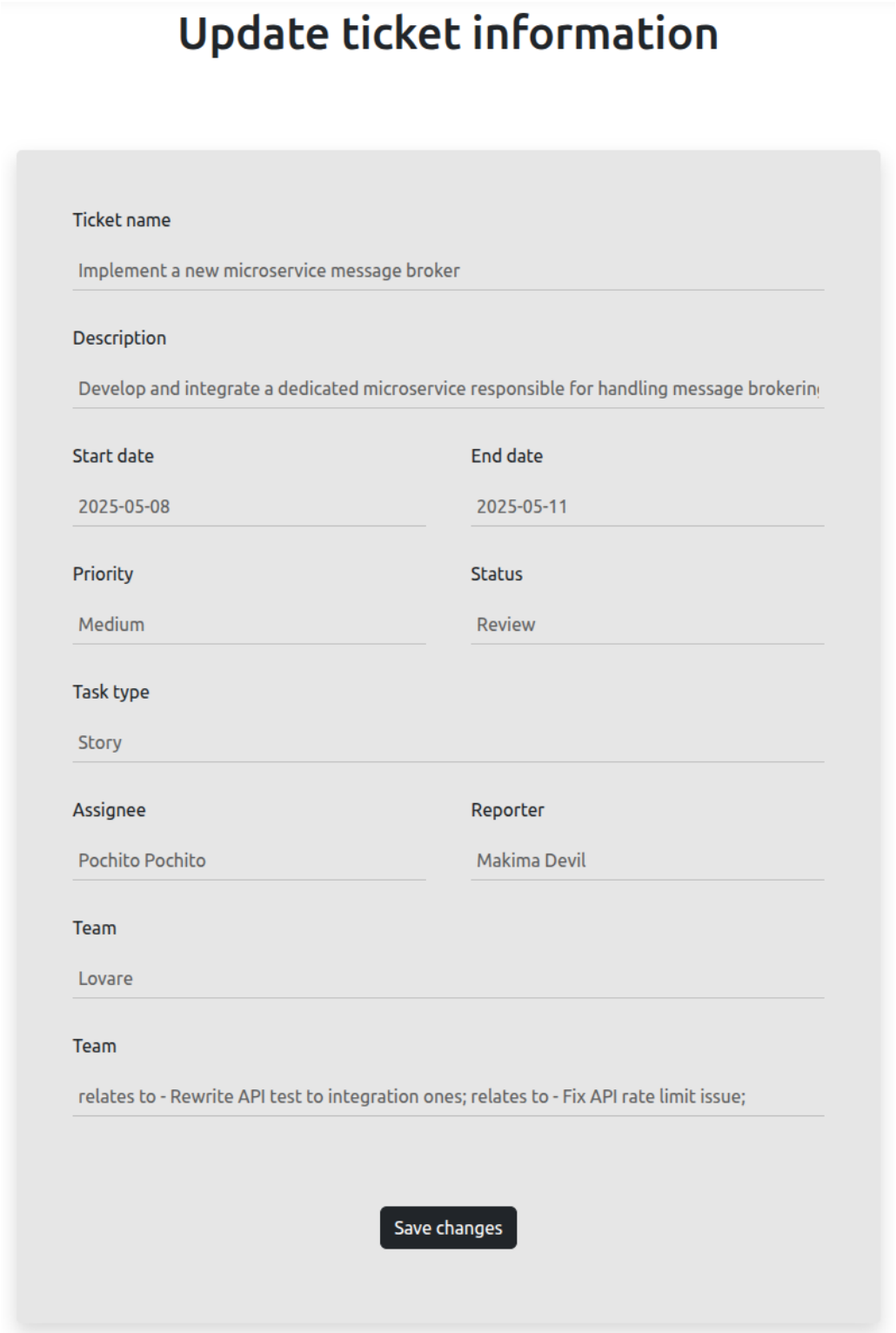

Рисунок 4.15 – Сторінка із формою зміни даних для завдання

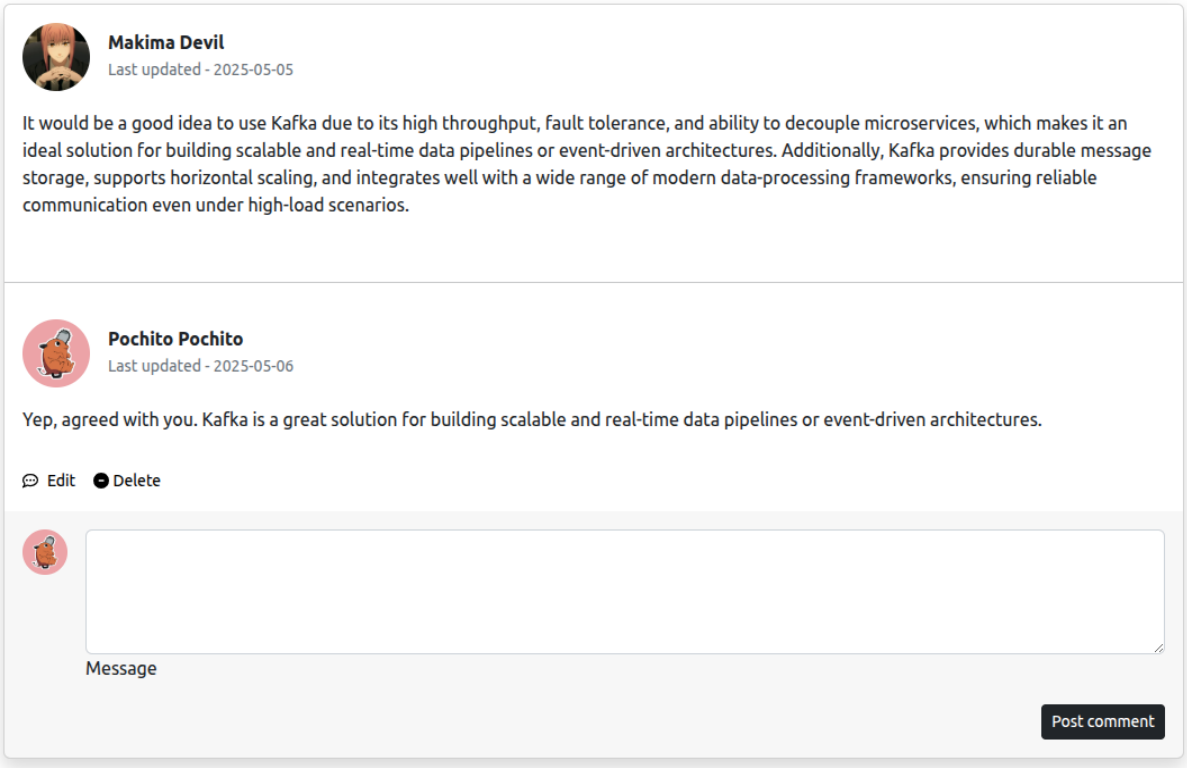

Рисунок 4.16 – Секція із коментарями до завдання

Розглянемо вкладку команди. Тут відображається список її учасників із зазначенням ролей (наприклад, менеджер, розробник, дизайнер), а також електронна пошта для подальшої комунікації. Інтерфейс сторінки команди наведено на рисунку 4.17.

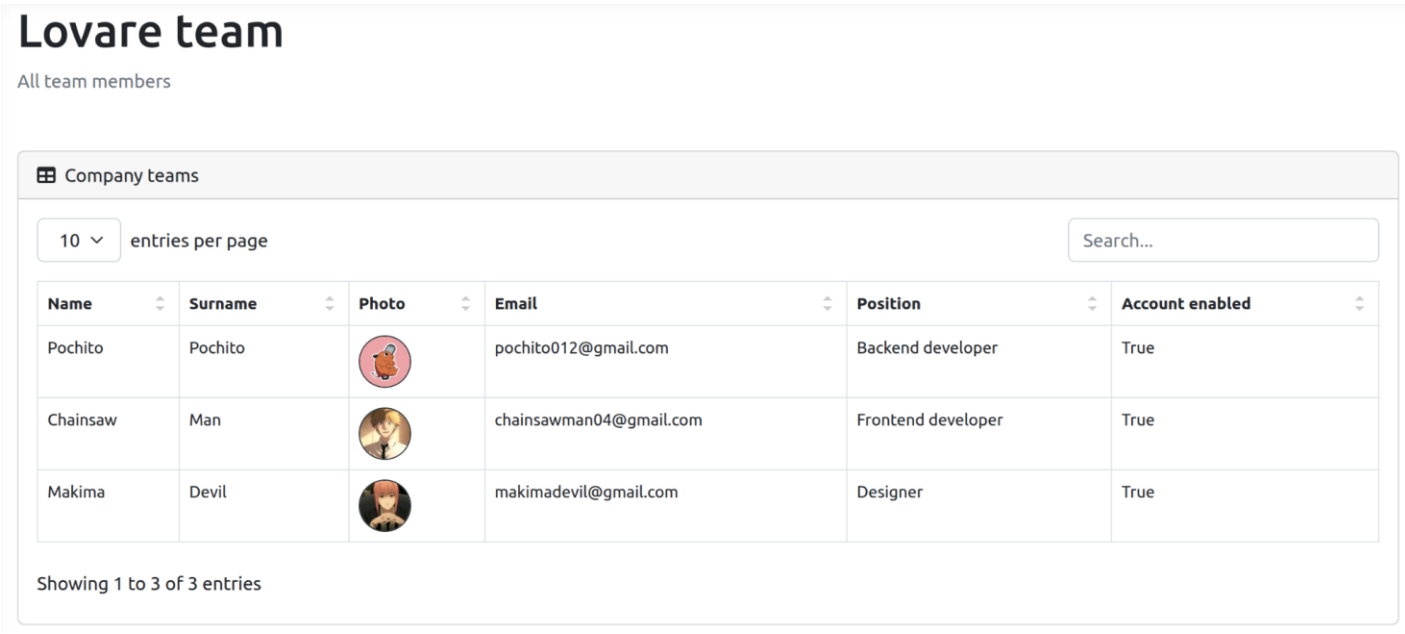

Рисунок 4.17 – Інтерфейс вкладки певної команди

Також у системі реалізовано функціональність перегляду статистики виконаних задач та зареєстрованих багів, що надає важливу аналітичну інформацію про динаміку виконання роботи. Завдяки цьому компоненту можливо оперативно оцінити прогрес проєкту, ефективність окремих команд чи виконавців, а також виявити потенційні проблемні зони в процесі розробки чи менеджменту.

Можна спостерігати ключові метрики, які надаються для перегляду: загальну кількість виконаних задач та кількість багів за обраний період (6 місяців, 3 місяці, 1 місяць, 2 тижні, 1 тиждень). Дана візуалізація надається у вигляді графіків та діаграм, що спрощує сприйняття інформації. Дані можуть фільтруватися за командами, окремими учасниками або типами задач, що дозволяє дослідити ситуацію з різних точок зору. Приклад зображення аналітичних даних представлене на рисунку 4.18.

У результаті проведеного тестування встановлено відповідність реалізованого функціоналу технічним вимогам. Основні компоненти WEB-додатка продемонстрували стабільну роботу, а перевірка ключових сценаріїв підтвердила коректність їх реалізації.

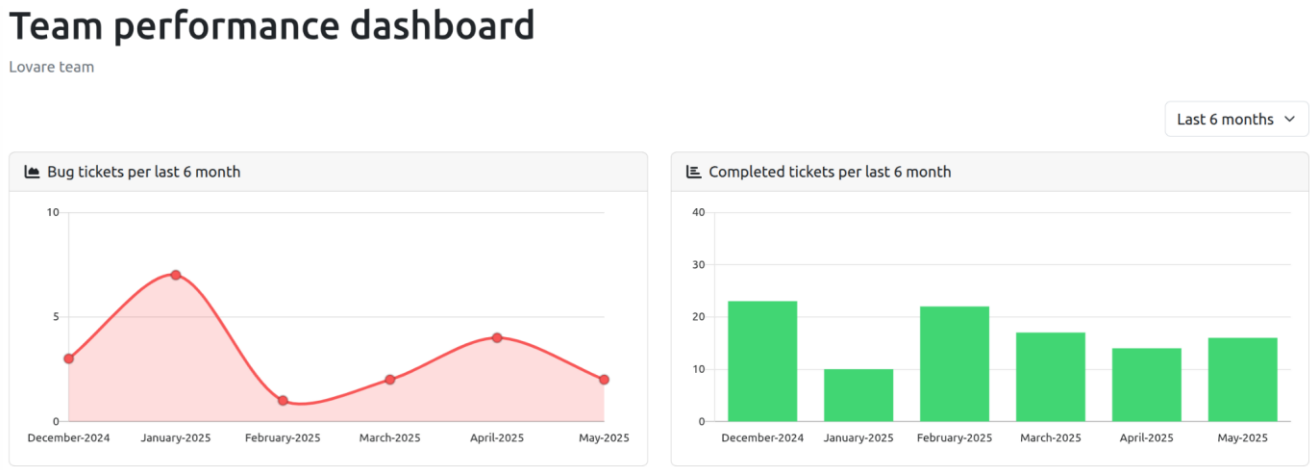

Рисунок 4.18 – Аналітичні дані певної команди за кількістю закритих задач та виявлених серед них багів

4.6 Висновок

У межах розділу реалізовано функціональну частину WEB-додатка з урахуванням повного циклу роботи із задачами, включно зі створенням, редагуванням, переглядом, фільтрацією, пошуком та аналітичним обліком. Система побудована на архітектурі мікросервісів, що забезпечує гнучке масштабування та ефективну взаємодію компонентів. Окремо впроваджено розширення функціональності, зокрема аналітику задач, пошук за назвою команд, централізовані профілі користувачів, де міститься контактна інформація, та логічні зв’язки між командами та задачами.

Наведено інструкції щодо локального та хмарного розгортання WEB-додатка, а також налаштування середовища розробки. Визначено послідовність дій для запуску серверної частини, підключення до сховища, компіляції проєкту та підготовки до роботи кінцевого користувача.

Визначено системні вимоги для запуску WEB-додатка на різних пристроях. Встановлено перелік підтримуваних операційних систем і браузерів, а також мінімальні характеристики апаратного забезпечення, необхідні для стабільної роботи додатка.

Виконано автоматизовану перевірку функціональних елементів системи за допомогою інструментів Postman та SonarLint. Здійснено верифікацію стабільності API, правильності обробки запитів та відповідей, а також відповідності даних форматам обміну. Статичний аналіз коду дозволив мінімізувати вразливості на ранніх етапах розробки. Також сформовано набір сценаріїв для мануального тестування, що охоплюють ключові аспекти взаємодії користувача з інтерфейсом додатка.

Виконано ручну перевірку основних сценаріїв взаємодії користувача з WEB-додатком, зокрема автентифікації, зміни персональних даних, пошуку задач і команд, створення та редагування задач, а також роботи з коментарями та статистикою. Перевірено коректність відображення повідомлень про помилки, реакцію системи на некоректні дії та логіку гіперпосилань між

сутностями.

У результаті виконано повне функціональне тестування WEB-додатка, що включало автоматизовану перевірку бізнес-логіки та модулів API, а також мануальну оцінку інтерфейсу й користувацьких сценаріїв. Усі протестовані компоненти продемонстрували стабільну роботу, а реалізовані функціональні покращення підтвердили свою ефективність, що свідчить про досягнення мети розробки програмного рішення.

ВИСНОВКИ

Поставлені перед бакалаврською кваліфікаційною роботою задачі, а саме – виконати аналіз предметної області; дослідити існуючі програмні рішення; сформулювати вимоги до функціональності додатка; виконати проєктування структури та алгоритмів додатка; обґрунтувати вибір інструментів технічної реалізації; реалізувати програмне рішення; підготувати інструкції з розгортання та визначити системні вимоги для роботи додатка; виконати тестування – виконано в повному обсязі.

Було виконано аналіз предметної області та виявлено обмеження традиційних засобів управління задачами, таких як електронні таблиці та месенджери, в умовах сучасного динамічного середовища. Проведено дослідження популярних програмних рішень, таких як Jira, Trello, Asana та ClickUp, визначено їхні сильні та слабкі сторони. На основі цього сформульовано функціональні вимоги до WEB-додатка, орієнтованого на зручність використання, аналітичність та гнучкість.

У межах роботи спроєктовано структуру та логіку функціонування додатка. Розроблено загальну структурну схему, діаграми видів діяльності, діаграму прецедентів, UML-діаграми класів і ER-моделі. Усі ці візуалізації дозволили формалізувати функціональні та структурні компоненти додатку, визначити ролі користувачів, сценарії взаємодії та логіку обробки запитів.

Обґрунтовано вибір інструментів технічної реалізації. Для серверної частини обрано Java у поєднанні з фреймворком Spring та СУБД MySQL. Також визначено засоби для реалізації кешування, міжсервісної взаємодії, безпеки та роботи з даними. Для забезпечення надійного тестування API застосовано інструмент Postman, для розробки обрано інтегроване середовище IntelliJ IDEA. Клієнтська частина реалізується на базі HTML, CSS, JavaScript і Bootstrap, що забезпечує адаптивний та простий в обслуговуванні користувацький інтерфейс.

У межах технічної реалізації розроблено WEB-додаток, що охоплює

повний цикл роботи із задачами: створення, редагування, перегляд, видалення та зміна статусу. Реалізовано фільтрацію задач за командами, пріоритетами та статусами, а також пошук за різними атрибутами. Архітектура побудована за принципом мікросервісів, що включає окремий сервіс для управління задачами та інший для автентифікації. На відміну від типових аналогів, додаток містить такі розширені функції: вбудована аналітика задач на період до 6 місяців, доступ до контактної інформації колег, можливість пошуку за назвами команд, посилання на сторінку команди безпосередньо з інтерфейсу задачі. Такі покращення підвищують прозорість командної взаємодії, полегшують навігацію й дозволяють ефективніше керувати задачами в межах проєктної діяльності.

Описано інструкції щодо розгортання додатку, зокрема локального запуску, підключення до сховища, підготовки середовища розробки та запуску серверної частини. Визначено системні вимоги для роботи додатку на різних пристроях, включно з мінімальними характеристиками апаратного та програмного забезпечення.

Здійснено комплексне тестування функціональних можливостей WEB-додатка з використанням автоматизованого та мануального підходів. Автоматизоване тестування забезпечило перевірку коректності взаємодії між компонентами системи, обробки запитів до API, а також валідності форматів даних при CRUD-операціях. Мануальне тестування дало змогу оцінити поведінку інтерфейсу з позиції кінцевого користувача, перевірити стабільність роботи основних сценаріїв – автентифікації, редагування профілю, пошуку задач, перегляду аналітики тощо. Проведена перевірка підтвердила відповідність реалізованого функціоналу технічним вимогам та забезпечила високий рівень надійності системи.

Мета дослідження – розширення функціональних можливостей WEB-додатка для менеджменту задач різного типу – досягнута за допомогою реалізації таких нових функцій порівняно з аналогами, як перегляд аналітики задач на період до 6 місяців, забезпечення доступу до контактної інформації

членів команди для подальшої комунікації, пошук за назвами команд, а також зв'язок задач та команд через наявність посилання між ними. Результатом виконаної роботи став WEB-додаток, що поєднує зручність використання з високою продуктивністю, надійністю, масштабованістю та здатністю до адаптації під потреби сучасних команд.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Кисилевич А. О. Обґрунтування доцільності розробки WEB-додатка для менеджменту задач різного типу / А. О. Кисилевич , І. Р. Арсенюк // Молодь в науці: дослідження, проблеми, перспективи (МН-2025), 15.10.24 – 14.06.25 : збірник матеріалів. – Вінниця: ВНТУ, 2025. – URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/25360/20964 (дата звернення 08.06.2025)
2. Грицюк Ю. І., Жабич М. Р. Управління ризиками реалізації програмних проєктів. *Науковий вісник НЛТУ України*. 2018. Т. 28. № 1. С. 150–162. URL: *https://www.researchgate.net/publication/375536748_UPRAVLINNA_PROEKTAMI_V_IT-GALUZI_METODIKI_INSTRUMENTI_TA_KERUVANNA_RIZIKAMI* (дата звернення 05.05.2025)
3. Gartner Magic Quadrant for Collaborative Work Management. URL: https://www.gartner.com/en/documents/5975471 (дата звернення: 05.05.2025)
4. Top 20 Best Task Management Software for Work in 2023. URL: https://friday.app/p/task-management-software (дата звернення: 05.05.2025)
5. Pulse of the Profession® 2024, 15th Edition. URL: https://www.pmi.org/learning/thought-leadership/pulse/future-of-project-work (дата звернення: 05.05.2025).
6. Машницький Є. П., Арсенюк І. Р. Обгрунтування доцільності Розробки веб-додатка для менеджменту стартапів: Тези доповідей L науково-технічної конференції факультету інформаційних технологій та комп`ютерної інженерії. Вінниця : ВНТУ, 2021.
7. Jira: Issue & Project Tracking Software. URL: https://www.atlassian.com/software/jira (дата звернення 06.05.2025)

8. Trello: Capture, organize, and tackle your to-dos from anywhere. URL: https://trello.com/ (дата звернення 06.05.2025)
9. Asana: Manage your team’s work, projects, & tasks online. URL: https://asana.com/ (дата звернення: 06.05.2025)
10. ClickUp: The everything app for work. URL: https://clickup.com/ (дата звернення: 06.05.2025)
11. Architectural Styles and the Design of Network-based Software Architectures. URL: https://www.academia.edu/22384476/Architectural_Styles_and_the_Design_of_Network_based_Software_Architectures#loswp-work-container (дата звернення: 06.05.2025)
12. Java. URL: https://www.java.com/en/ (дата звернення: 07.05.2025)
13. Python Software Foundation. URL: https://www.python.org/psf-landing/ (дата звернення: 07.05.2025)
14. Usage statistics and market share of PHP. URL: https://w3techs.com/technologies/details/pl-php (дата звернення: 07.05.2025)
15. C# language documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 07.05.2025)
16. Spring Documentation. URL: https://spring.io/projects/spring-boot (дата звернення: 07.05.2025)
17. Михайленко Є. О., Арсенюк І. Р. Обґрунтування вибору інструментів для створення бази даних клієнтів для бізнесу: Матеріали LI науково-технічної конференції підрозділів ВНТУ. Вінниця : ВНТУ, 2022.
18. MySQL Documentation. URL: https://dev.mysql.com/doc (дата звернення: 07.05.2025)
19. Liquibase Documentation. URL: https://www.liquibase.org/documentation (дата звернення: 07.05.2025)
20. Caffeine Cache GitHub Repository. URL: https://github.com/ben-manes/caffeine (дата звернення: 07.05.2025)

21. Redis Documentation. URL: https://redis.io/docs (дата звернення: 07.05.2025)
22. JWT Introduction. URL: https://jwt.io/introduction (дата звернення: 07.05.2025)
23. Hibernate ORM Documentation. URL: https://hibernate.org/orm/documentation (дата звернення: 07.05.2025)
24. JOOQ Documentation. URL: https://www.jooq.org/doc (дата звернення: 07.05.2025)
25. OpenFeign Documentation. URL: https://github.com/OpenFeign/feign (дата звернення: 07.05.2025)
26. Postman Docs. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення: 07.05.2025)
27. IntelliJ IDEA Overview. URL: https://www.jetbrains.com/idea (дата звернення: 07.05.2025)
28. Introduction to Angular. URL: https://angular.io/docs (дата звернення: 08.05.2025)
29. Overview of HTML, CSS, and JavaScript. URL: https://dev.to/sudhakar_v_c404997aeec839/overview-of-html-css-and-javascript-1jja (дата звернення: 08.05.2025)
30. Bootstrap Documentation. URL: https://getbootstrap.com/ (дата звернення: 08.05.2025)
31. Battle of the Clouds: AWS vs. Azure vs. Google vs. IBM. URL: https://zesty.co/blog/battle-of-clouds-aws-vs-azure/ (дата звернення: 03.06.2025)
32. *Богач І. В., Слободян Р. В.* Автоматизоване тестування веб-додатків шляхом взаємодії з користувацьким інтерфейсом. Вінниця : ВНТУ, 2017.
33. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний

ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: WEB-додаток для менеджменту задач різного типу

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,10 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✔ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Арсенюк І.Р.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Кисилевич А.О.
(підпис) (прізвище, ініціали)

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА WEB-ДОДАТКА ДЛЯ МЕНЕДЖМЕНТУ ЗАДАЧ РІЗНОГО ТИПУ

Щоб увійти в систему, користувач повинен бути попередньо зареєстрований адміністратором. Якщо цього не зроблено, слід перейти на сторінку технічної підтримки (рисунок Б.1).

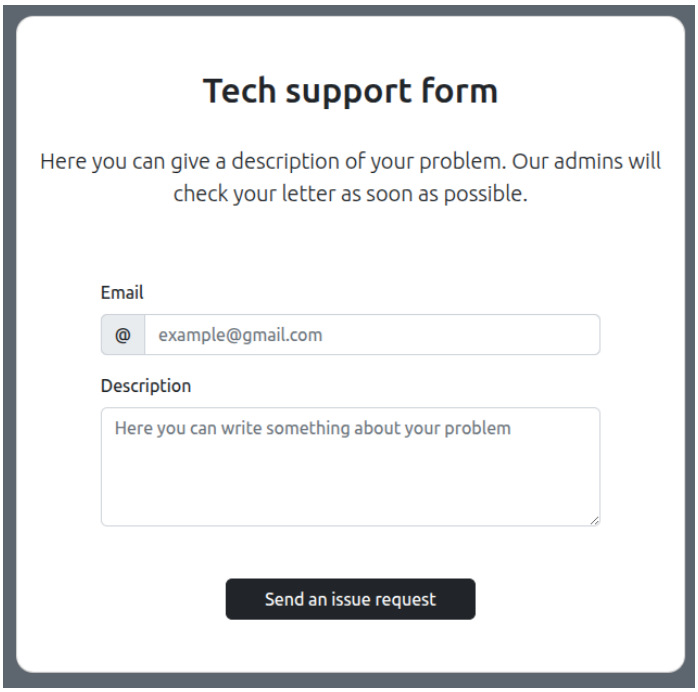

Рисунок Б.1 – WEB-сторінка технічної підтримки

Зареєстровані користувачі вводять електронну адресу та пароль на сторінці входу (рисунок Б.2) й натискають кнопку “Login”.

Після успішної авторизації система автоматично перенаправляє користувача на сторінку з аналітичними даними команди, де можна переглядати кількість виконаних задач і виявлених багів за обраний період: 6 місяців, 3 місяці, 1 місяць, 2 тижні або 1 тиждень. Для зміни інтервалу достатньо скористатися випадаючим списком (рисунок Б.3).

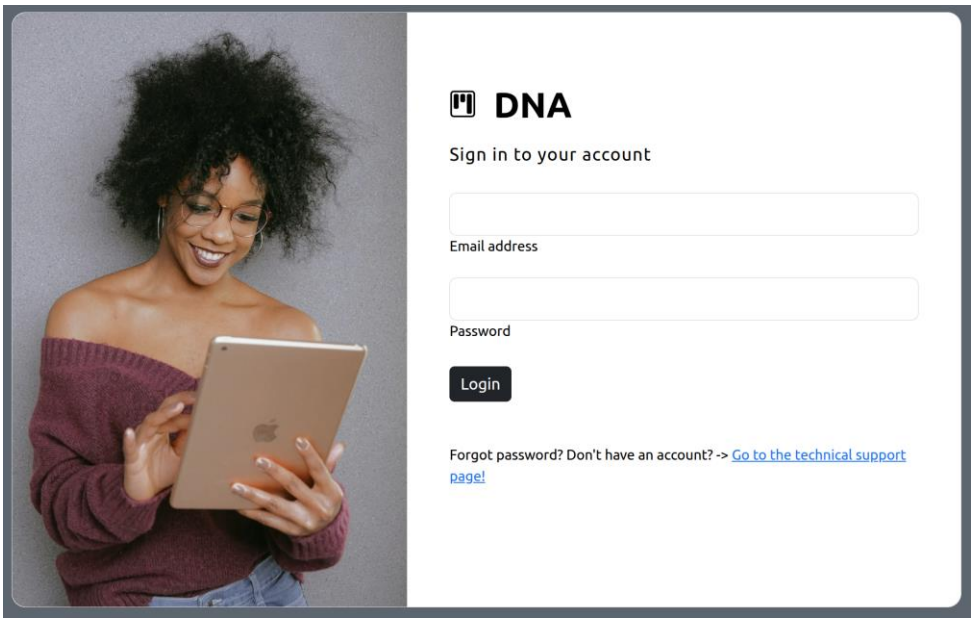

Рисунок Б.2 – WEB-сторінка входу у систему

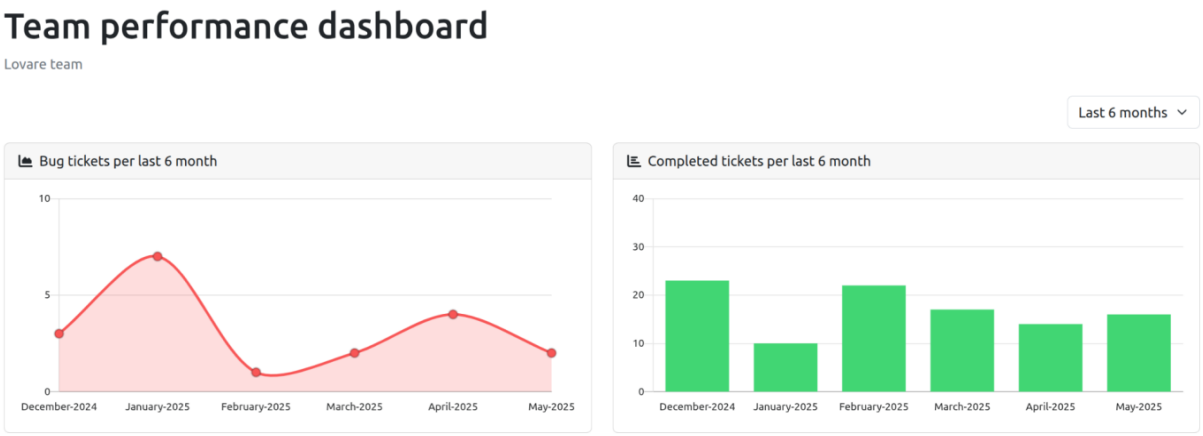

Рисунок Б.3 – WEB-сторінка аналітичних даних команди

Для зручної навігації передбачено бокове меню з посиланнями на основні розділи системи: “Tickets” – для перегляду та створення задач, “People” – для перегляду профілів колег, “Teams” – для роботи з командами, “About us” – для загальної інформації про систему, “Settings” – для доступу до особистого кабінету (рисунок Б.4).

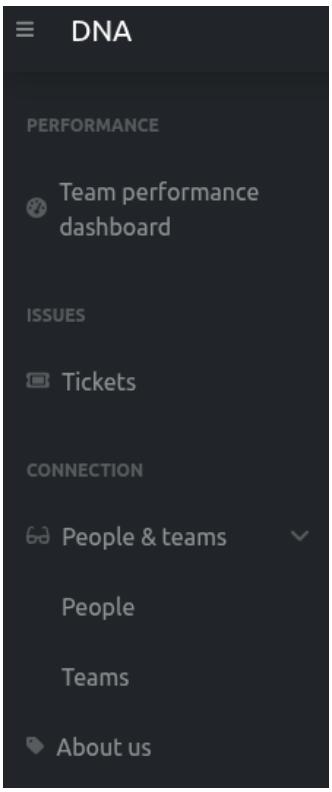

Рисунок Б.4 – Боковий список посилань на WEB-сторінки системи

Щоб переглянути задачу, користувач переходить у розділ “Tickets” і вводить її назву у відповідне поле пошуку. Також тут відображаються всі наявні задачі та команди з найбільшою кількістю виконаних завдань (рисунок Б.5).

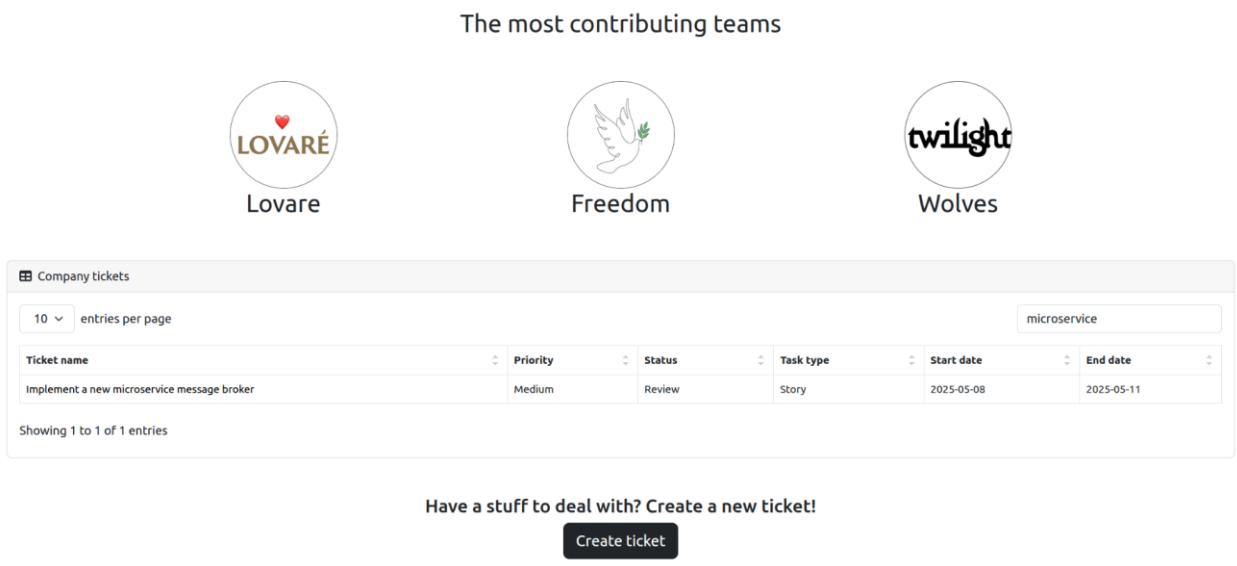

Рисунок Б.5 – WEB-сторінка для пошуку задач

Для перегляду деталей потрібно натиснути на назву задачі у колонці “Ticket name”, після чого відкриється її сторінка з описом, пріоритетом, статусом, автором, виконавцем, пов’язаними задачами та командою (рисунок Б.6).

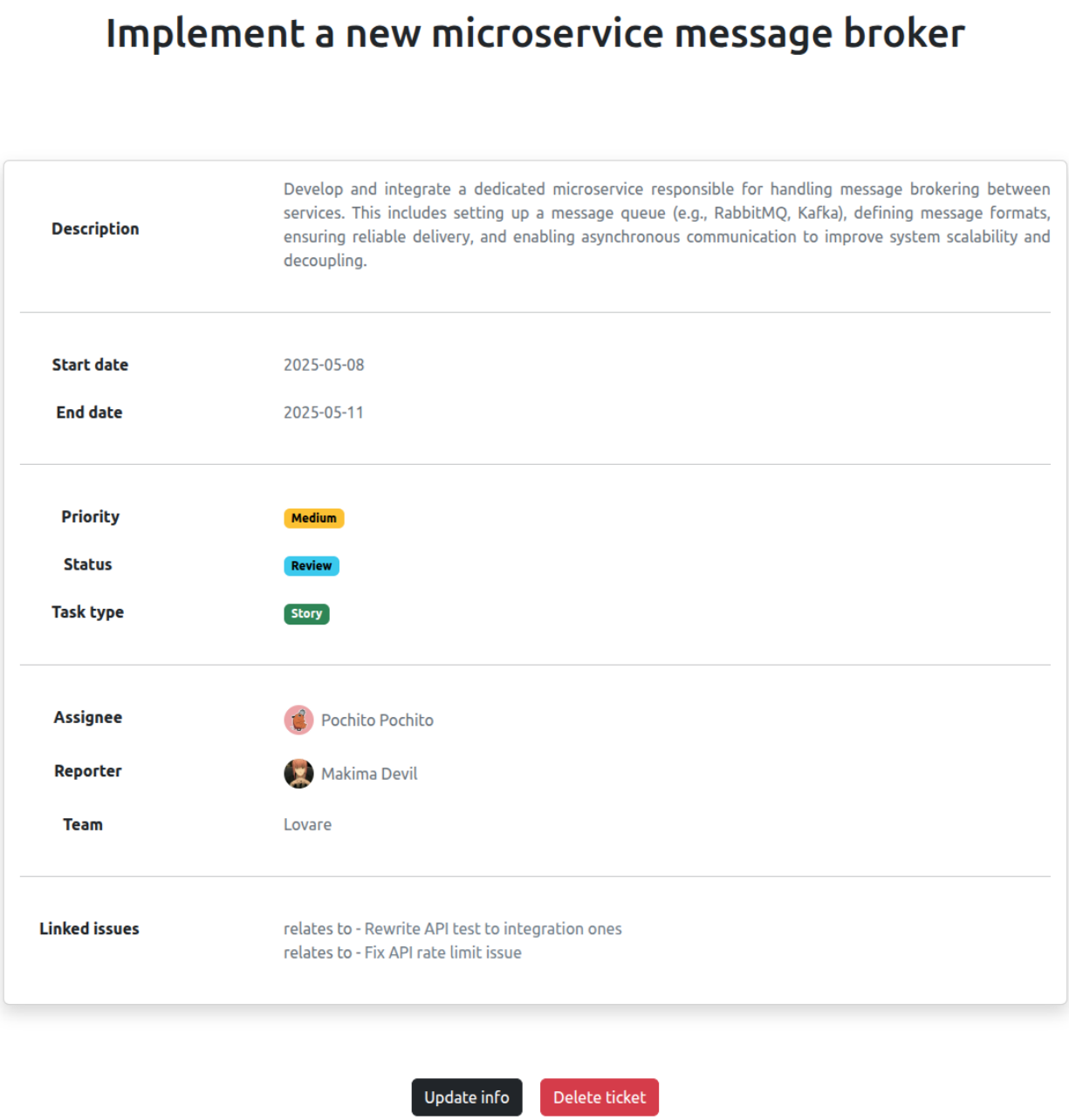

Рисунок Б.6 – Детальна інформація про задачу

За необхідності дані задачі можна змінити, натиснувши кнопку “Update info”, що відкриє форму редагування (рисунок Б.7).

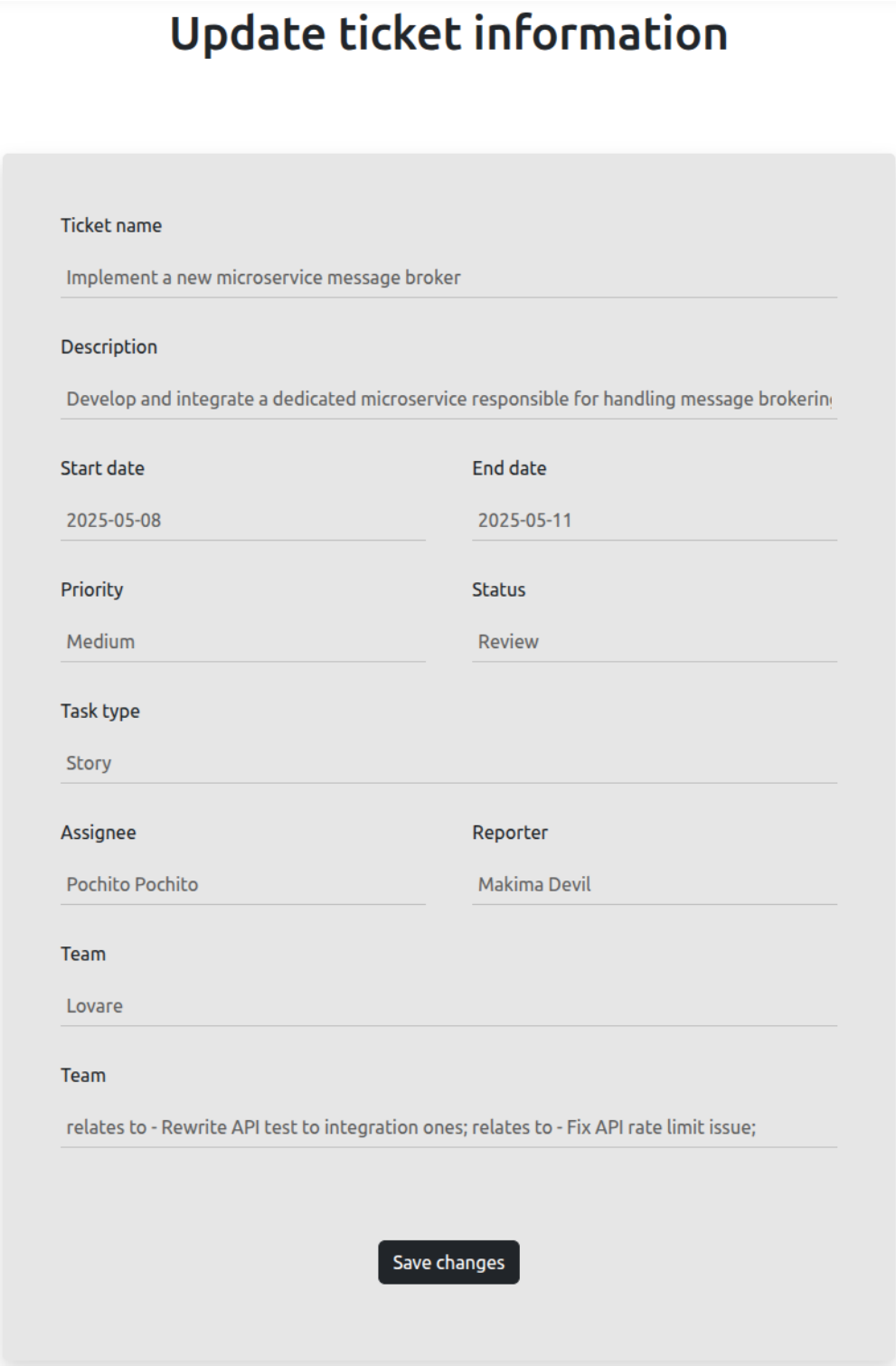

Рисунок Б.7 – Форма для зміни деталей завдання

Крім того, зі сторінки задачі доступний перехід до профілю виконавця або автора задачі (через поля “Assignee” та “Reporter”) або до сторінки відповідальної команди (через поле “Team”). Відповідні сторінки зображено на рисунках Б.8 і Б.9.

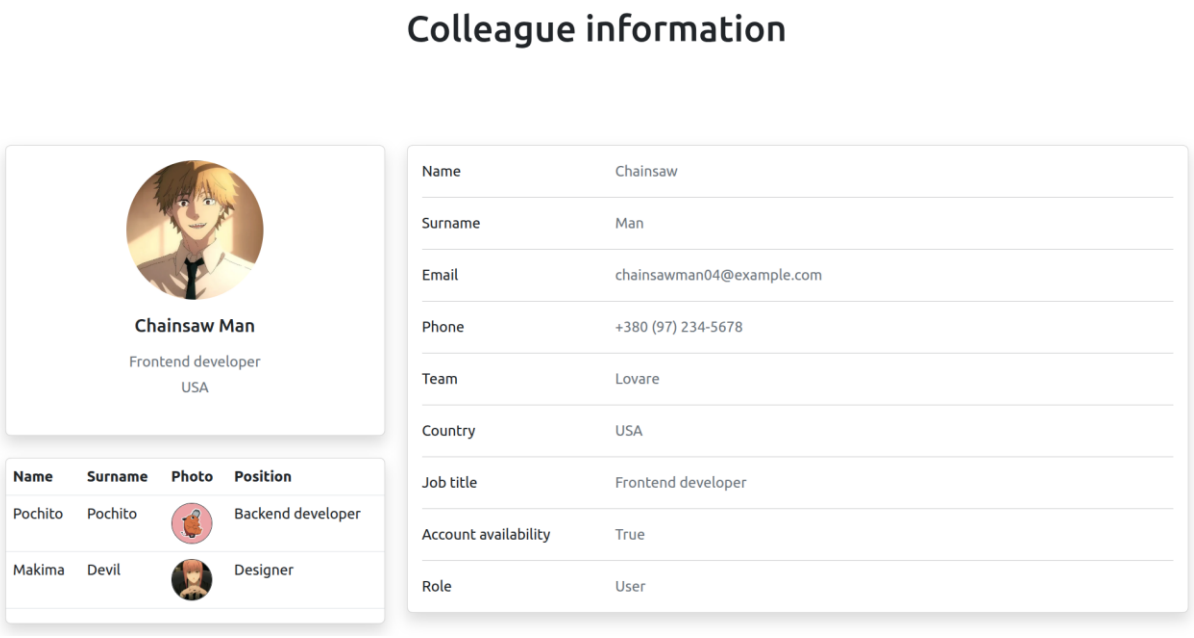

Рисунок Б.8 – WEB-сторінка інформації про колегу

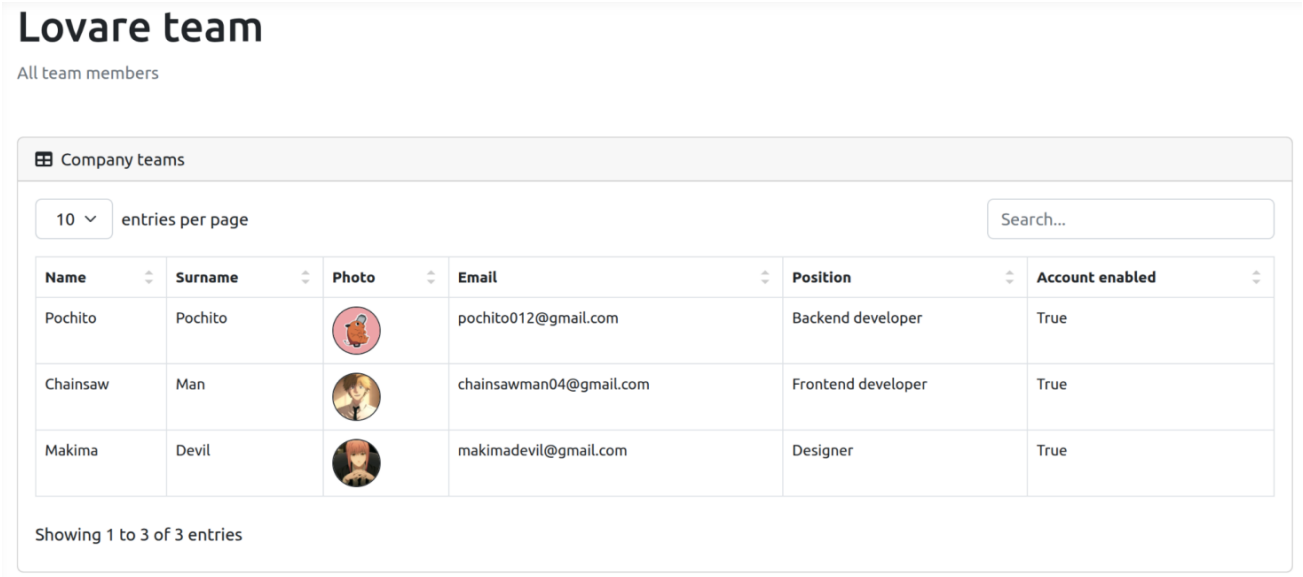

Рисунок Б.9 – WEB-сторінка інформації про команду

Якщо потрібно знайти конкретну команду, у боковому меню необхідно вибрати пункт “Teams”. Це перенаправить користувача до сторінки пошуку команд, де відображено повний список команд і поле для фільтрації за назвою (рисунок Б.10).

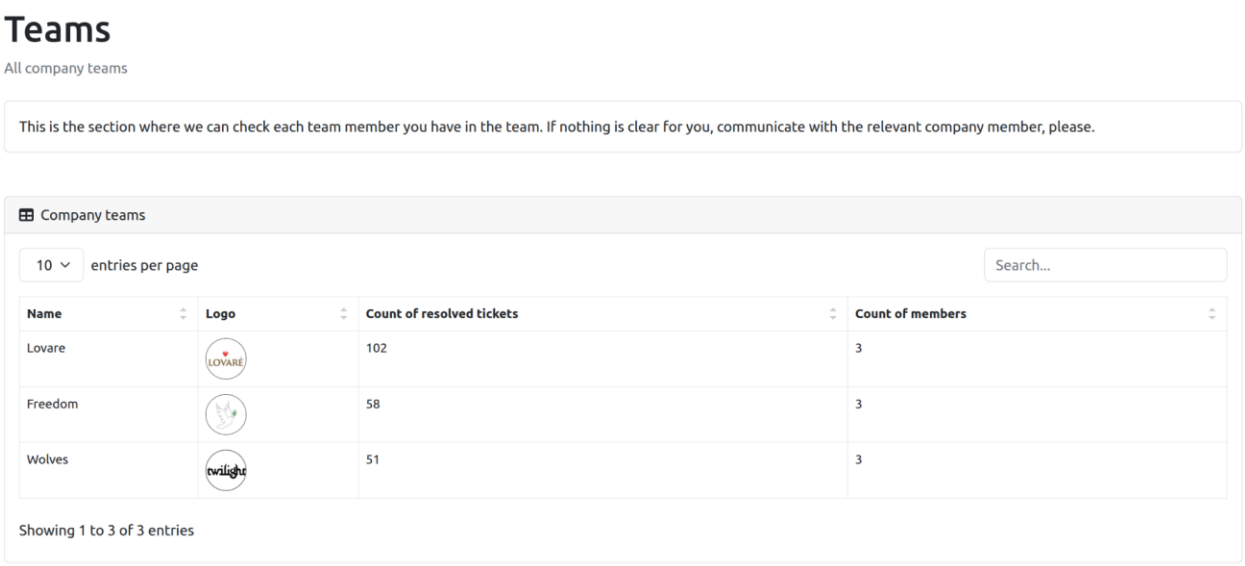

Рисунок Б.10 – WEB-сторінка пошуку команд у системі

Для перегляду або редагування власних персональних даних слід перейти до розділу “Settings” (рисунок Б.11), де відображається актуальна інформація користувача, яка може бути доступною іншим членам команди для ефективної взаємодії (рисунок Б.12).

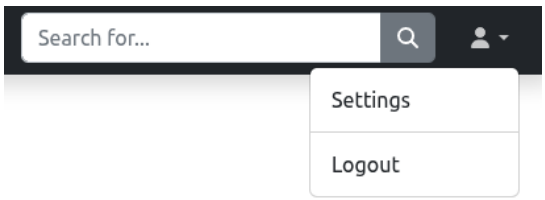

Рисунок Б.11 – Опції для переходу до персонального кабінету

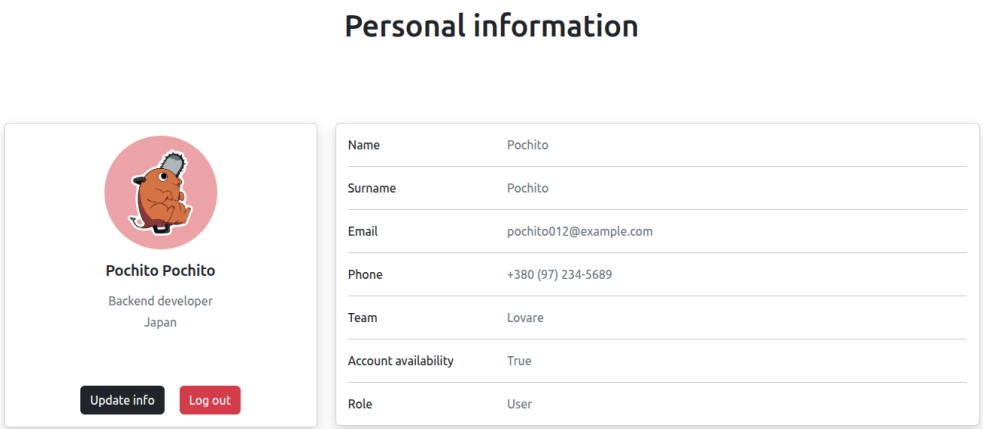

Рисунок Б.12 – WEB-сторінка персонального кабінету користувача

За потреби оновлення даних натискається кнопка “Update info”, що відкриває форму редагування (рисунок Б.13).

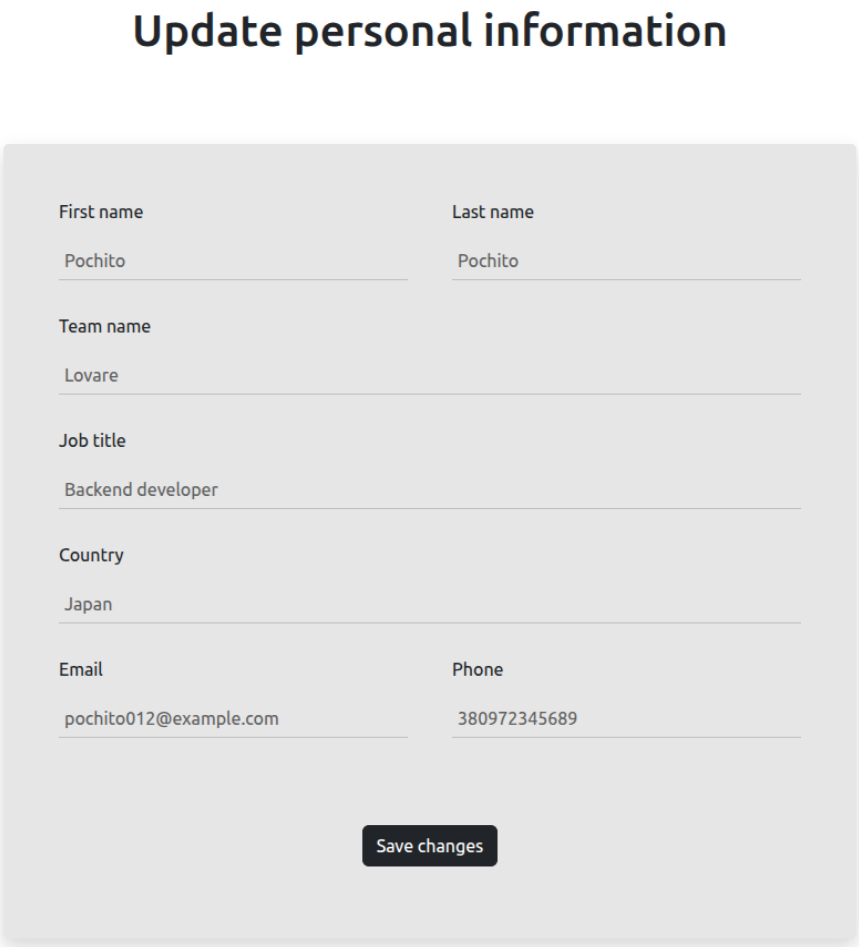

Рисунок Б.13 – WEB-сторінка редагування персональних даних користувача

Якщо жодна з наявних задач не відповідає поточній діяльності, користувач може створити нову. Для цього потрібно перейти у розділ “Tickets” і натиснути кнопку “Create a ticket” (рисунок Б.5).

Після заповнення усіх необхідних полів форма підтверджується повторним натисканням кнопки “Create a ticket” (рисунок Б.14).

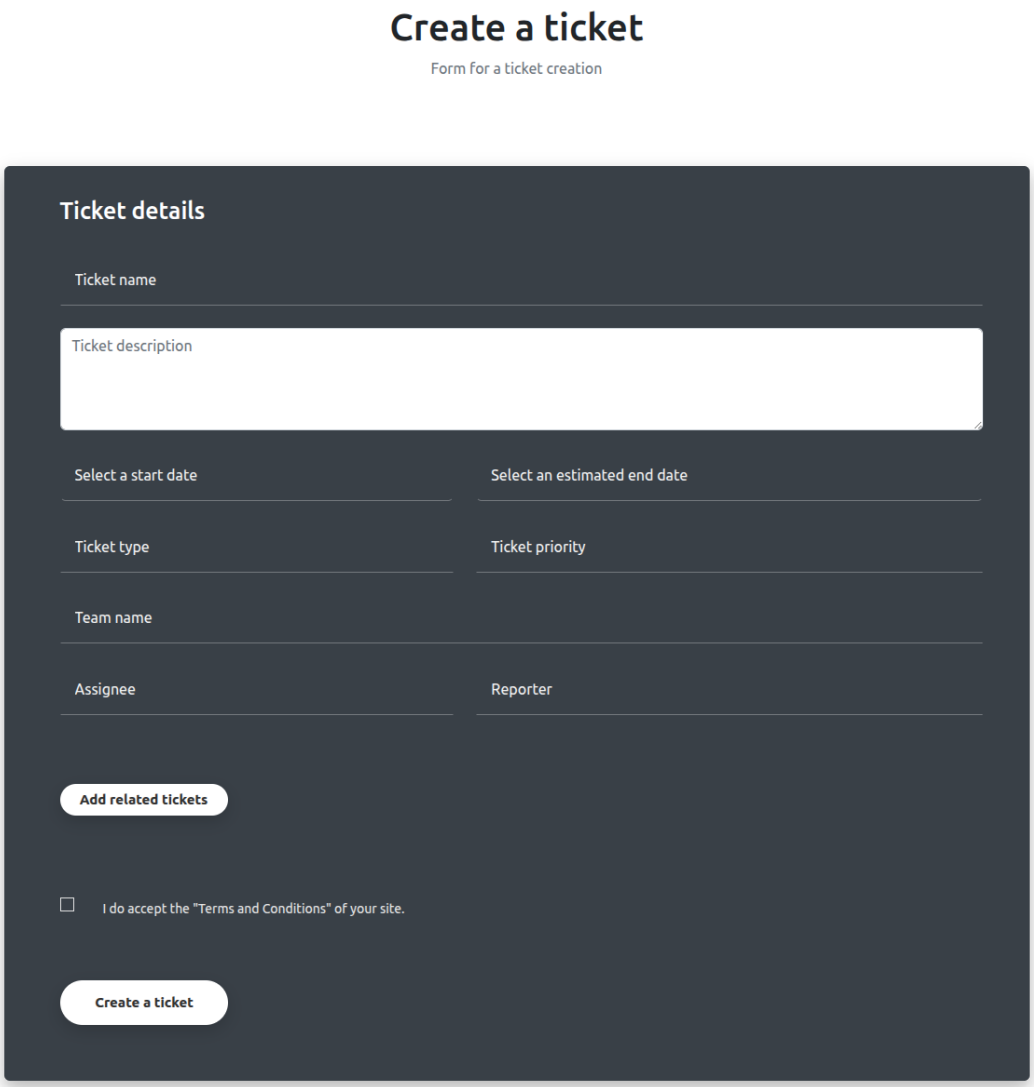

Рисунок Б.14 – WEB-сторінка для створення задачі

ДОДАТОК В
ЛІСТИНГ ПРОГРАМИ

Клас UserInfoController:

```
package ua.edu.vntu.authservicedna.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatusCode;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import ua.edu.vntu.authservicedna.entity.user.UserInfo;
import ua.edu.vntu.authservicedna.service.user.UserInfoService;
import ua.edu.vntu.authservicedna.utils.TokenValidationUtils;

import java.util.List;
import java.util.Optional;

@RestController
@RequiredArgsConstructor
@CrossOrigin(origins = "*")
@RequestMapping("/user")
public class UserInfoController {

    private final UserInfoService userInfoService;

    @GetMapping("/{id}")
    @PreAuthorize("hasAuthority('ROLE_ADMIN') || hasAuthority('ROLE_USER')")
    public      UserInfo      getUserById(@RequestHeader("Authorization")      String
authorizationHeader,
                                @PathVariable("id") Integer id) {
        TokenValidationUtils.validateToken(authorizationHeader);
        final Optional<UserInfo> userInfo = userInfoService.getUserById(id);
```

```
        return userInfo.orElseThrow(() -> new UsernameNotFoundException("User not found
by id: " + id));
    }

    @GetMapping
    @PreAuthorize("hasAuthority('ROLE_ADMIN') || hasAuthority('ROLE_USER')")
    public    List<UserInfo>    getUserById(@RequestHeader("Authorization")    String
authorizationHeader,
                                      @RequestParam List<Integer> userIds) {
        TokenValidationUtils.validateToken(authorizationHeader);
        return userInfoService.getUsersByIds(userIds);
    }

    @PostMapping
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public      UserInfo      createUser(@RequestHeader("Authorization")      String
authorizationHeader,
                               @RequestBody UserInfo userInfo) {
        TokenValidationUtils.validateToken(authorizationHeader);
        return userInfoService.createUser(userInfo);
    }

    @PutMapping
    @PreAuthorize("hasAuthority('ROLE_USER')")
    public      UserInfo      updateUser(@RequestHeader("Authorization")      String
authorizationHeader,
                               @RequestBody UserInfo userInfo) {
        TokenValidationUtils.validateToken(authorizationHeader);
        return userInfoService.updateUser(userInfo);
    }

    @DeleteMapping("/{id}")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public    ResponseEntity<?>    dropUser(@RequestHeader("Authorization")    String
authorizationHeader,
                                     @PathVariable("id") Integer id) {
        TokenValidationUtils.validateToken(authorizationHeader);
        userInfoService.dropUserById(id);
        return ResponseEntity.status(HttpStatusCode.valueOf(204)).build();
    }

    @PutMapping("/{id}/toggle")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public   UserInfo   toggleUserAccountById(@RequestHeader("Authorization")   String
authorizationHeader,
                                          @PathVariable("id") Integer id,
                                          @RequestParam("toggle") Boolean toggle) {
```

```
        TokenValidationUtils.validateToken(authorizationHeader);
        return userInfoService.toggleUserAccountById(id, toggle);
    }
}
```

Клас UserInfoServiceImpl:

```
package ua.edu.vntu.authservicedna.service.user;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import ua.edu.vntu.authservicedna.entity.user.UserInfo;
import ua.edu.vntu.authservicedna.entity.user.UserInfoDetails;
import ua.edu.vntu.authservicedna.repository.jpa.UserInfoRepository;

import java.util.List;
import java.util.Optional;

@RequiredArgsConstructor
public class UserInfoServiceImpl implements UserInfoService {

    private final UserInfoRepository repository;
    private final PasswordEncoder encoder;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        final Optional<UserInfo> userInfo =
Optional.of(repository.findByEmail(username));
        return userInfo.map(UserInfoDetails::new)
                .orElseThrow(() -> new UsernameNotFoundException("User not found: " +
username));
    }

    @Override
    public UserInfo createUser(UserInfo userInfo) {
        final String password = encoder.encode(userInfo.getPassword());
        userInfo.setPassword(password);
        return repository.save(userInfo);
    }
```

```
    @Override
    @CachePut(value = "UserCache", key = "#userInfo.id")
    public UserInfo updateUser(UserInfo userInfo) {
        final UserInfo userFromDb = repository.findByEmail(userInfo.getEmail());
        userInfo.setPassword(userFromDb.getPassword());
        return repository.save(userInfo);
    }

    @Override
    @CacheEvict(value = "UserCache", key = "#id")
    public void dropUserById(Integer id) {
        repository.deleteById(id);
    }

    @Override
    @Cacheable(value = "UserCache", key = "#id")
    public Optional<UserInfo> getUserById(Integer id) {
        System.out.println("get from db id = " + id);
        return repository.findById(id);
    }

    @Override
    public List<UserInfo> getUsersByIds(List<Integer> userIds) {
        return repository.findAllById(userIds);
    }

    @Override
    @Transactional
    public UserInfo toggleUserAccountById(Integer id, Boolean toggle) {
        if (repository.toggleUserAccount(id, toggle) > 0) {
            return getUserById(id).get();
        }
        throw new IllegalArgumentException("Id = " + id + ", toggle = " + toggle);
    }
}
```

Клас UserInfoRepository:

```
package ua.edu.vntu.authservicedna.repository.jpa;

import jakarta.transaction.Transactional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import ua.edu.vntu.authservicedna.entity.JobTitle;
import ua.edu.vntu.authservicedna.entity.TeamEntity;
```

```
import ua.edu.vntu.authservicedna.entity.user.UserInfo;

import java.util.List;
import java.util.Optional;

@Repository
public interface UserInfoRepository extends JpaRepository<UserInfo, Integer> {
    UserInfo findByEmail(String email);
    List<UserInfo> findByJobTitle(JobTitle jobTitle);
    List<UserInfo> findByTeam(String team);
    Optional<UserInfo> findById(Integer id);

    @Query("SELECT u FROM UserInfo u WHERE u.email NOT IN (?1) AND u.team LIKE ?2")
    List<UserInfo> findByTeamWithoutUser(String email, String team);

    @Modifying
    @Transactional
    @Query("UPDATE UserInfo u SET u.isAccountEnabled = :toggle WHERE u.id = :id ")
    int toggleUserAccount(Integer id, Boolean toggle);

    @Query("SELECT DISTINCT u.team FROM UserInfo u WHERE u.team NOT LIKE '%UNKNOWN%'")
    List<String> findAllTeams();

    @Query("SELECT new ua.edu.vntu.authservicedna.entity.TeamEntity(t.team, COUNT(t))
FROM UserInfo t WHERE t.team != 'UNKNOWN' GROUP BY t.team")
    List<TeamEntity> findAllTeamsDetailed();

    @Query("SELECT u FROM UserInfo u " +
        "WHERE (:isEquals = FALSE AND ((LOWER(u.firstName) NOT LIKE LOWER(CONCAT('%',
:firstName, '%'))) " +
        "AND (LOWER(u.lastName) NOT LIKE LOWER(CONCAT('%', :lastName, '%'))))) " +
        "OR (:isEquals = TRUE AND ((LOWER(u.firstName) LIKE LOWER(CONCAT('%',
:firstName, '%'))) " +
        "OR (LOWER(u.lastName) LIKE LOWER(CONCAT('%', :lastName, '%')))))")
    List<UserInfo> findByFirstNameAndLastName(String firstName, String lastName,
boolean isEquals);

    @Query("SELECT u FROM UserInfo u " +
            "WHERE (:isEquals = FALSE AND ((LOWER(u.firstName) NOT LIKE
LOWER(CONCAT('%', :name, '%'))) " +
            "AND (LOWER(u.lastName) NOT LIKE LOWER(CONCAT('%', :name, '%'))))) " +
            "OR (:isEquals = TRUE AND ((LOWER(u.firstName) LIKE LOWER(CONCAT('%', :name,
'%'))) " +
            "OR (LOWER(u.lastName) LIKE LOWER(CONCAT('%', :name, '%')))))")
    List<UserInfo> findByAnyName(String name, boolean isEquals);
}
```

Клас TaskEntityServiceImpl:

```
package ua.edu.vntu.taskservice.service.task;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import ua.edu.vntu.taskservice.dao.repository.task.TaskEntityRepository;
import ua.edu.vntu.taskservice.entity.task.CommentEntity;
import ua.edu.vntu.taskservice.entity.task.EstimateEntity;
import ua.edu.vntu.taskservice.entity.task.LinkedIssueEntity;
import ua.edu.vntu.taskservice.entity.task.SprintEntity;
import ua.edu.vntu.taskservice.entity.task.TaskEntity;
import ua.edu.vntu.taskservice.entity.task.dto.TaskEntityDto;
import ua.edu.vntu.taskservice.entity.user.UserInfo;
import ua.edu.vntu.taskservice.service.comment.CommentEntityService;
import ua.edu.vntu.taskservice.service.estimate.EstimateEntityService;
import ua.edu.vntu.taskservice.service.feign.UserClient;
import ua.edu.vntu.taskservice.service.linkedissue.LinkedIssueEntityService;
import ua.edu.vntu.taskservice.service.sprint.SprintEntityService;

import java.util.List;

@Service("taskEntityService")
@RequiredArgsConstructor
public class TaskEntityServiceImpl implements TaskEntityService {

    private final TaskEntityRepository taskEntityRepository;
    private final LinkedIssueEntityService linkedIssueEntityService;
    private final CommentEntityService commentEntityService;
    private final EstimateEntityService estimateEntityService;
    private final SprintEntityService sprintEntityService;
    private final UserClient userClient;

    @Override
    public TaskEntity createTask(TaskEntityDto task) {
        final String name = task.getName();
        final int result = taskEntityRepository.createTask(task);
        if (result > 0) {
            return null;
        }
        throw new IllegalStateException("We can't create task with name = " + name);
    }

    @Override
    public TaskEntity getTaskById(Integer id, String auth) {
        final TaskEntityDto taskEntityDto = taskEntityRepository.getTaskById(id);
```

```
        return buildEntityFromDto(taskEntityDto, auth);
    }

    @Override
    public List<TaskEntity> getTasksByParentId(Integer parentId, String auth) {
        final             List<TaskEntityDto>             taskEntityDtos             =
taskEntityRepository.getTasksByParentId(parentId);
        return taskEntityDtos.stream()
                .map(ent -> buildEntityFromDto(ent, auth))
                .toList();
    }

    @Override
    public List<TaskEntity> getTasksByProjectId(Integer projectId, String auth) {
        final             List<TaskEntityDto>             taskEntityDtos             =
taskEntityRepository.getTasksByProjectId(projectId);
        return taskEntityDtos.stream()
                .map(ent -> buildEntityFromDto(ent, auth))
                .toList();
    }

    @Override
    public TaskEntity updateTask(TaskEntity task) {
        final Integer taskId = task.getId();
        final int result = taskEntityRepository.updateTask(task);
        if (result > 0) {
//            return getTaskById(taskId);
            return null;
        }
        throw new IllegalStateException("We can't update task with id = " + taskId);
    }

    @Override
    public void deleteTaskById(Integer id) {
        final int result = taskEntityRepository.deleteTaskById(id);
        if (result <= 0) {
            throw new IllegalStateException("We can't delete task with id = " + id);
        }
    }

    private   TaskEntity   buildEntityFromDto(TaskEntityDto   taskEntityDto,   String
authorizationHeader) {
        final Integer taskId = taskEntityDto.getId();
        final          List<LinkedIssueEntity>          linkedIssueEntities          =
linkedIssueEntityService.getLinkedIssuesByTaskId(taskId);
        final               List<CommentEntity>               comments               =
commentEntityService.getCommentsByTaskId(taskId, authorizationHeader);
```

```
        final                EstimateEntity                estimate                =
estimateEntityService.getEstimateById(taskEntityDto.getEstimateId());
        final                 SprintEntity                 sprint                 =
sprintEntityService.getSprintById(taskEntityDto.getSprintId());
        final UserInfo assignee = userClient.getUserById(taskEntityDto.getAssigneeId(),
authorizationHeader);
        final UserInfo reporter = userClient.getUserById(taskEntityDto.getReporterId(),
authorizationHeader);

        return TaskEntity.builder()
                .id(taskId)
                .parentId(taskEntityDto.getParentId())
                .projectId(taskEntityDto.getProjectId())
                .name(taskEntityDto.getName())
                .description(taskEntityDto.getDescription())
                .assignee(assignee)
                .reporter(reporter)
                .type(taskEntityDto.getType())
                .status(taskEntityDto.getStatus())
                .linkedIssues(linkedIssueEntities)
                .comments(comments)
                .assignedTeam(taskEntityDto.getAssignedTeam())
                .estimate(estimate)
                .sprint(sprint)
                .priority(taskEntityDto.getPriority())
                .startDate(taskEntityDto.getStartDate())
                .endDate(taskEntityDto.getEndDate())
                .build();
    }
}
```

Клас TaskEntityRepositoryImpl:

```
package ua.edu.vntu.taskservice.dao.repository.task;

import lombok.RequiredArgsConstructor;
import org.jooq.DSLContext;
import org.jooq.Record;
import org.jooq.RecordMapper;
import org.springframework.stereotype.Repository;
import ua.edu.vntu.taskservice.dao.table.ProjectTasksTable;
import ua.edu.vntu.taskservice.entity.task.TaskEntity;
import ua.edu.vntu.taskservice.entity.task.dto.TaskEntityDto;

import java.util.List;
```

```
@Repository("taskEntityRepository")
@RequiredArgsConstructor
public class TaskEntityRepositoryImpl implements TaskEntityRepository {

    private static final RecordMapper<Record, TaskEntityDto> ROW_MAPPER = record ->
            TaskEntityDto.builder()
                    .id(record.getValue(ProjectTasksTable.TABLE.id))
                    .parentId(record.getValue(ProjectTasksTable.TABLE.parent_id))
                    .projectId(record.getValue(ProjectTasksTable.TABLE.project_id))
                    .name(record.getValue(ProjectTasksTable.TABLE.name))
                    .description(record.getValue(ProjectTasksTable.TABLE.description))
                    .assigneeId(record.getValue(ProjectTasksTable.TABLE.assignee_id))
                    .reporterId(record.getValue(ProjectTasksTable.TABLE.reporter_id))
                    .type(record.getValue(ProjectTasksTable.TABLE.type))
                    .status(record.getValue(ProjectTasksTable.TABLE.status))

.assignedTeam(record.getValue(ProjectTasksTable.TABLE.assigned_team))
                    .estimateId(record.getValue(ProjectTasksTable.TABLE.estimate_id))
                    .sprintId(record.getValue(ProjectTasksTable.TABLE.sprint_id))
                    .priority(record.getValue(ProjectTasksTable.TABLE.priority))

.startDate(record.getValue(ProjectTasksTable.TABLE.start_date).toLocalDate())

.endDate(record.getValue(ProjectTasksTable.TABLE.end_date).toLocalDate())
                    .build();

    private final DSLContext dslContext;

    @Override
    public int createTask(TaskEntityDto task) {
        return dslContext.insertInto(ProjectTasksTable.TABLE)
                .set(ProjectTasksTable.TABLE.parent_id, task.getParentId())
                .set(ProjectTasksTable.TABLE.project_id, task.getProjectId())
                .set(ProjectTasksTable.TABLE.name, task.getName())
                .set(ProjectTasksTable.TABLE.description, task.getDescription())
                .set(ProjectTasksTable.TABLE.assignee_id, task.getAssigneeId())
                .set(ProjectTasksTable.TABLE.reporter_id, task.getReporterId())
                .set(ProjectTasksTable.TABLE.type, task.getType())
                .set(ProjectTasksTable.TABLE.status, task.getStatus())
                .set(ProjectTasksTable.TABLE.assigned_team, task.getAssignedTeam())
                .set(ProjectTasksTable.TABLE.estimate_id, task.getEstimateId())
                .set(ProjectTasksTable.TABLE.sprint_id, task.getSprintId())
                .set(ProjectTasksTable.TABLE.priority, task.getPriority())
                .set(ProjectTasksTable.TABLE.start_date,
java.sql.Date.valueOf(task.getStartDate()))
                .set(ProjectTasksTable.TABLE.end_date,
java.sql.Date.valueOf(task.getEndDate()))
```

```
            .execute();
}

@Override
public TaskEntityDto getTaskById(Integer id) {
    return dslContext.selectFrom(ProjectTasksTable.TABLE)
            .where(ProjectTasksTable.TABLE.id.eq(id))
            .fetchOne(ROW_MAPPER);
}

@Override
public TaskEntityDto getTaskByName(String name) {
    return dslContext.selectFrom(ProjectTasksTable.TABLE)
            .where(ProjectTasksTable.TABLE.name.eq(name))
            .fetchOne(ROW_MAPPER);
}

@Override
public List<TaskEntityDto> getTasksByParentId(Integer parentId) {
    return dslContext.selectFrom(ProjectTasksTable.TABLE)
            .where(ProjectTasksTable.TABLE.parent_id.eq(parentId))
            .fetch(ROW_MAPPER);
}

@Override
public List<TaskEntityDto> getTasksByProjectId(Integer projectId) {
    return dslContext.selectFrom(ProjectTasksTable.TABLE)
            .where(ProjectTasksTable.TABLE.project_id.eq(projectId))
            .fetch(ROW_MAPPER);
}

@Override
public int updateTask(TaskEntity task) {
    return dslContext.update(ProjectTasksTable.TABLE)
            .set(ProjectTasksTable.TABLE.parent_id, task.getParentId())
            .set(ProjectTasksTable.TABLE.project_id, task.getProjectId())
            .set(ProjectTasksTable.TABLE.name, task.getName())
            .set(ProjectTasksTable.TABLE.description, task.getDescription())
            .set(ProjectTasksTable.TABLE.assignee_id, task.getAssignee().getId())
            .set(ProjectTasksTable.TABLE.reporter_id, task.getReporter().getId())
            .set(ProjectTasksTable.TABLE.type, task.getType())
            .set(ProjectTasksTable.TABLE.status, task.getStatus())
            .set(ProjectTasksTable.TABLE.assigned_team, task.getAssignedTeam())
            .set(ProjectTasksTable.TABLE.estimate_id, task.getEstimate().getId())
            .set(ProjectTasksTable.TABLE.sprint_id, task.getSprint().getId())
            .set(ProjectTasksTable.TABLE.priority, task.getPriority())
```

```
                .set(ProjectTasksTable.TABLE.start_date,
java.sql.Date.valueOf(task.getStartDate()))
                .set(ProjectTasksTable.TABLE.end_date,
java.sql.Date.valueOf(task.getEndDate()))
                .where(ProjectTasksTable.TABLE.id.eq(task.getId()))
                .execute();
    }

    @Override
    public int deleteTaskById(Integer id) {
        return dslContext.deleteFrom(ProjectTasksTable.TABLE)
                .where(ProjectTasksTable.TABLE.id.eq(id))
                .execute();
    }
}
```

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

«WEB-ДОДАТОК ДЛЯ МЕНЕДЖМЕНТУ ЗАДАЧ РІЗНОГО ТИПУ»

Виконала: студентка 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Кисилевич А. О.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)

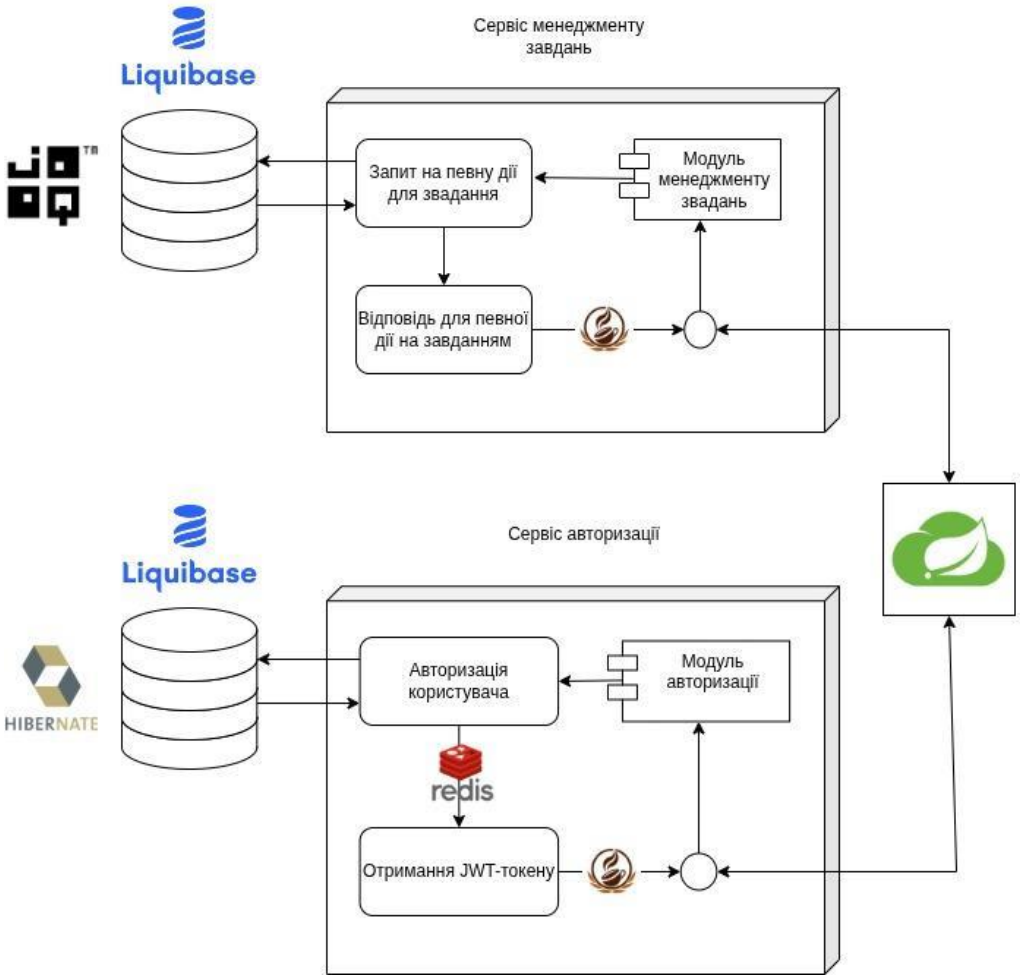

Рисунок Г.1 – Загальна структура системи

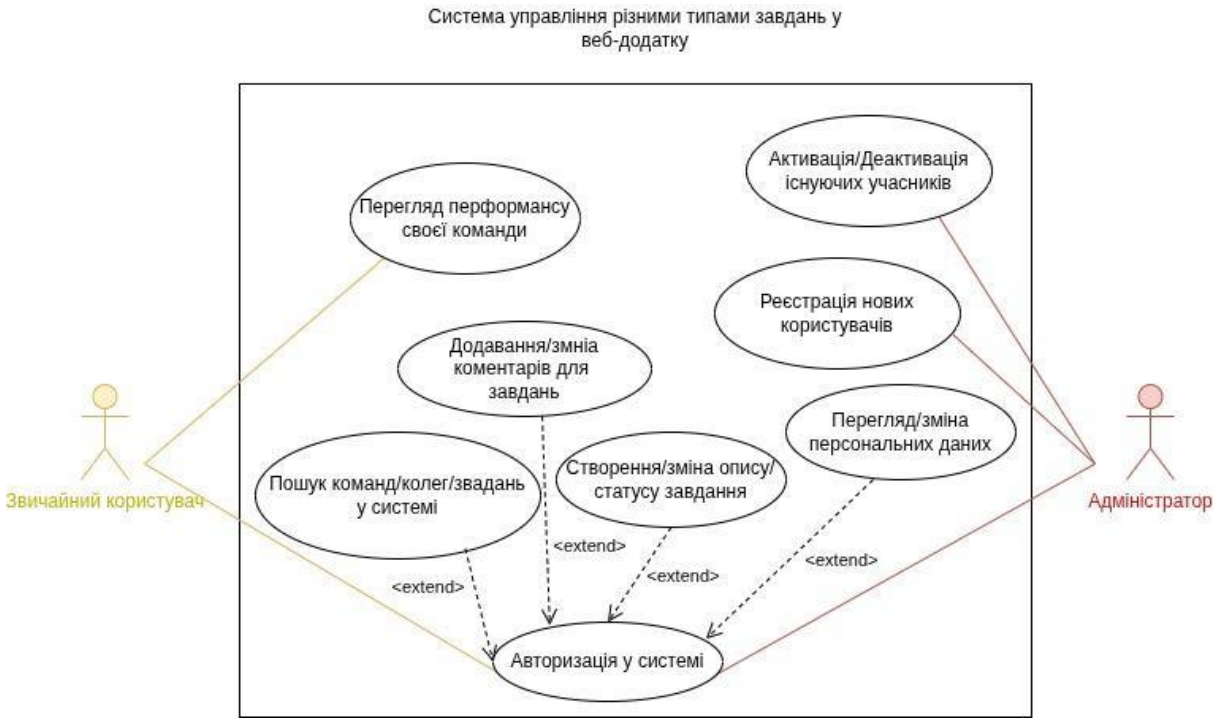

Рисунок Г.2 – Діаграма прецедентів

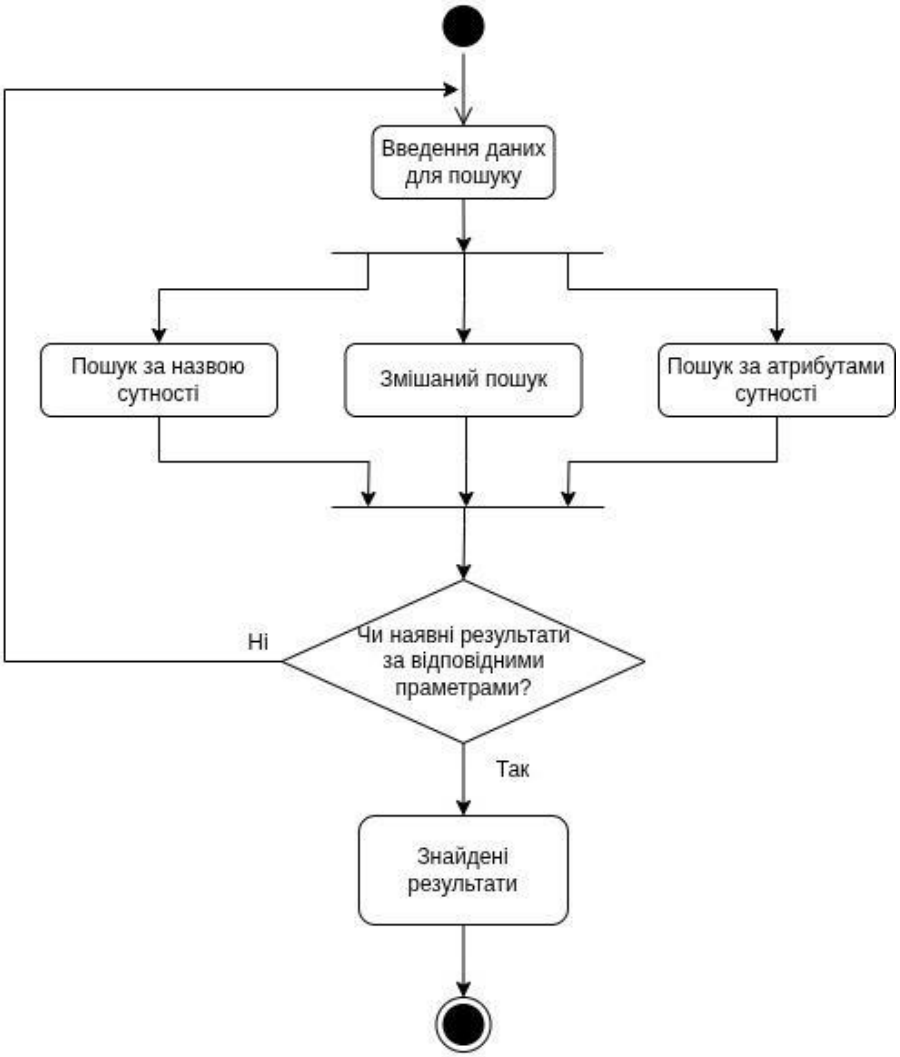

Рисунок Г.3 – Діаграма виду діяльності для пошуку сутності

Рисунок Г.4 – Діаграма виду діяльності для зміни персональних даних користувача

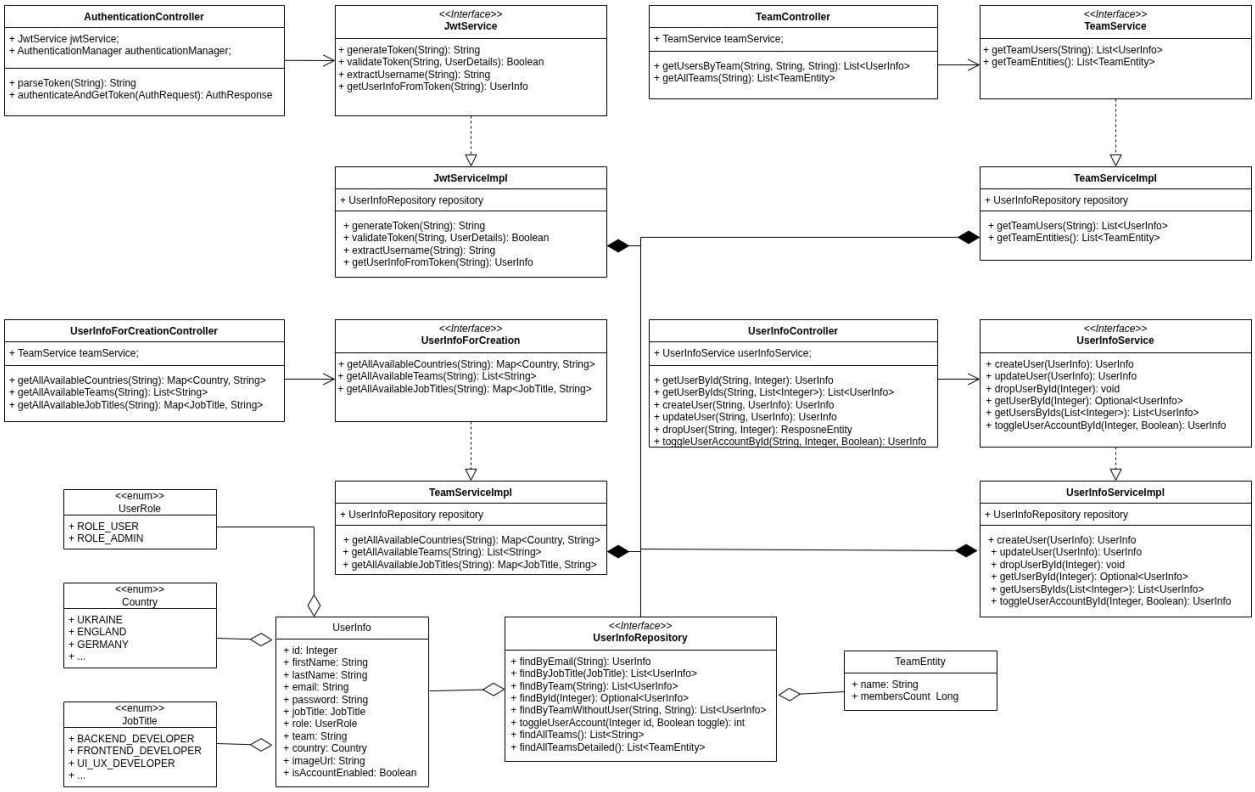

Рисунок Г.5 – UML-діаграма класів сервісу авторизації

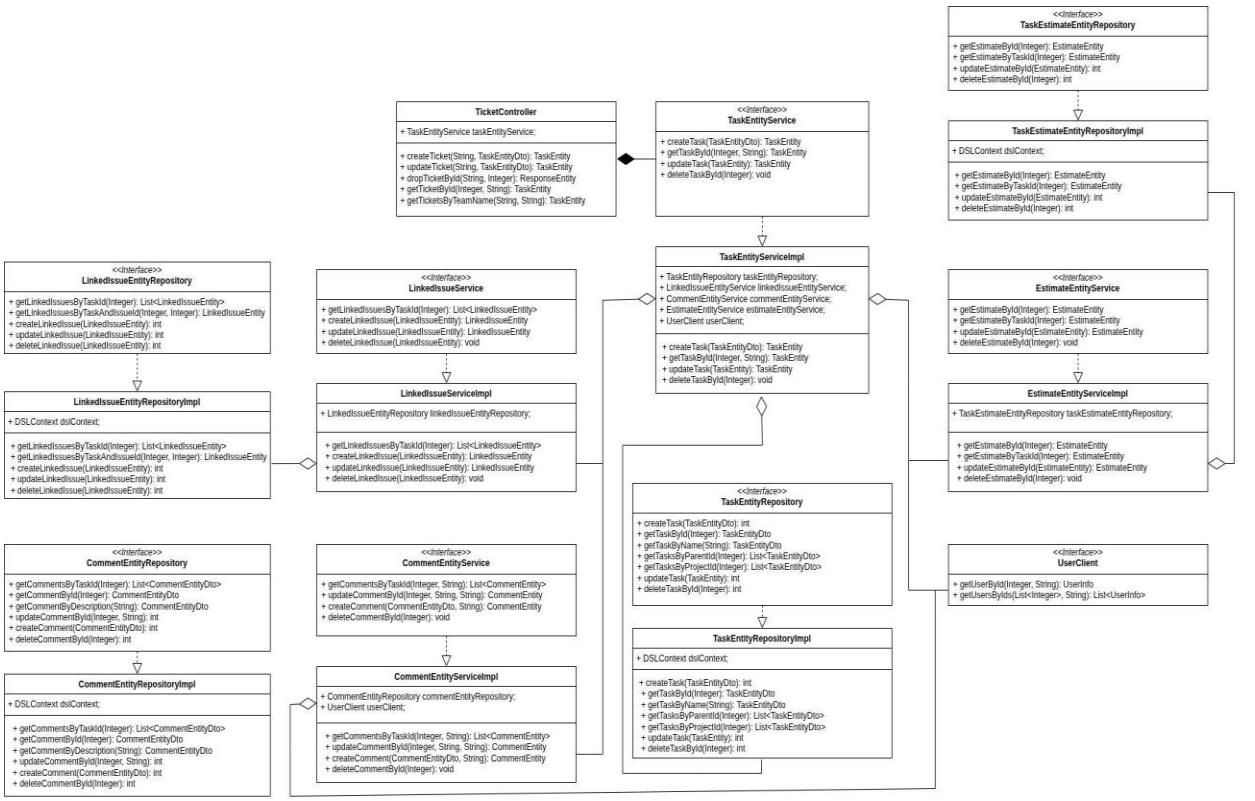

Рисунок Г.6 – UML-діаграма класів сервісу менеджменту завдань

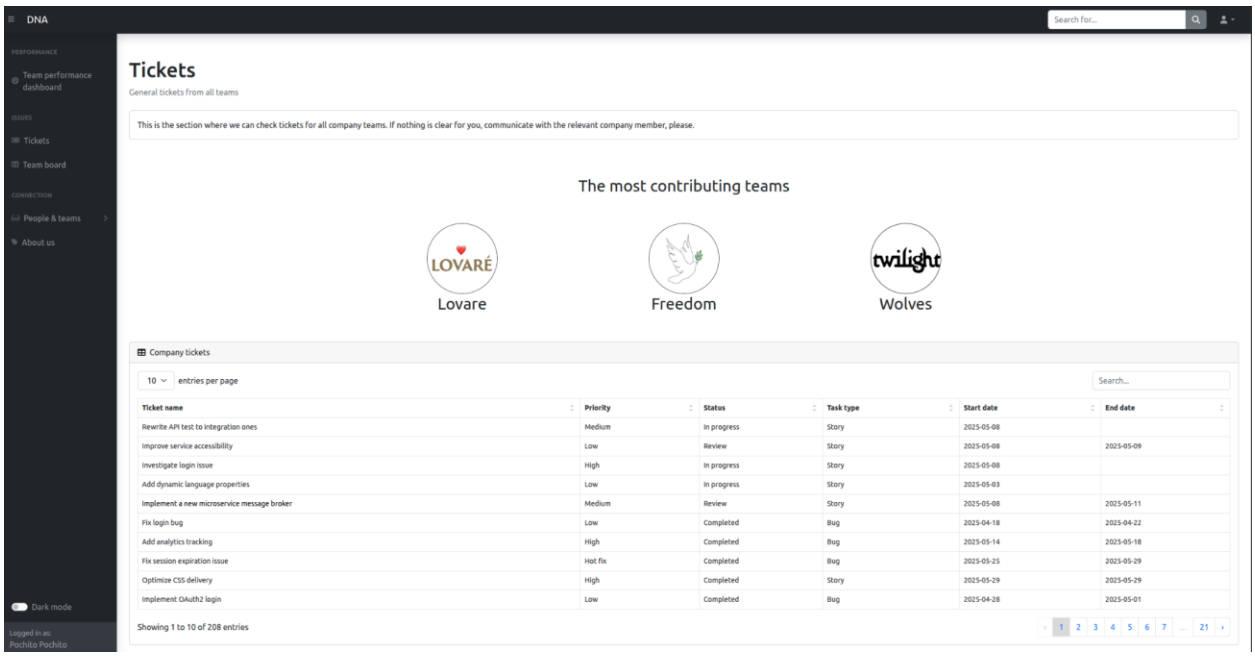

Рисунок Г.7 – WEB-сторінка перегляду задач проєкту