E-RequestControl

Специфікація вимог до програмного продукту

Майборода Богдан Олександрович

Кучук Ілля Андрійович

05.06.2025

Зміст

[Вступ 4](#_Toc200895107)

[1.1 Огляд продукту 4](#_Toc200895108)

[1.2 Мета 4](#_Toc200895109)

[1.3 Межі 5](#_Toc200895110)

[1.4 Посилання 6](#_Toc200895111)

[2 Загальний опис 7](#_Toc200895112)

[2.1 Перспективи продукту 7](#_Toc200895113)

[2.2 Функції продукту 7](#_Toc200895114)

[2.3 Характеристика користувачів 8](#_Toc200895115)

[2.4 Загальні обмеження 9](#_Toc200895116)

[2.5 Припущення і залежності 9](#_Toc200895117)

[3 Конкретні вимоги 11](#_Toc200895118)

[3.1 Вимоги до зовнішніх інтерфейсів 11](#_Toc200895119)

[3.1.1 Інтерфейс користувача 11](#_Toc200895120)

[3.1.2 Програмний інтерфейс 15](#_Toc200895121)

[3.1.3 Комунікаційний протокол 15](#_Toc200895122)

[3.1.4 Обмеження пам’яті 15](#_Toc200895123)

[3.1.5 Операції 16](#_Toc200895124)

[3.1.6 Функції продукту 16](#_Toc200895125)

[3.2 Вимоги до зовнішніх інтерфейсів 17](#_Toc200895126)

[3.3 Атрибути програмного продукту 17](#_Toc200895127)

[3.3.1 Надійність 17](#_Toc200895128)

[3.3.2 Безпека 18](#_Toc200895129)

[3.3.3 Супроводжуваність 18](#_Toc200895130)

[3.3.4 Переносимість 18](#_Toc200895131)

[3.3.5 Продуктивність 18](#_Toc200895132)

[3.4 Вимоги до бази даних 19](#_Toc200895133)

[3.5 Інші вимоги 20](#_Toc200895134)

[3.5.1 Журнали подій 20](#_Toc200895135)

[3.5.2 Резервне копіювання 20](#_Toc200895136)

[3.5.3 Валідація даних 20](#_Toc200895137)

[3.5.4 Мови інтерфейсу 21](#_Toc200895138)

[3.5.5 Масштабованість 21](#_Toc200895139)

[4 Додаткові матеріали 22](#_Toc200895140)

[4.1 Схеми баз даних 22](#_Toc200895141)

[4.2 Діаграми прецедентів 23](#_Toc200895142)

# Вступ

## 1.1 Огляд продукту

E-RequestControl — це вебзастосунок для автоматизованого управління електронними заявками всередині організації. Система надає зручний інтерфейс для створення, обробки, сортування та контролю статусів заявок, дозволяючи адміністраторам, співробітникам та іншим користувачам ефективно взаємодіяти з заявками в єдиному середовищі. Архітектура застосунку побудована за клієнт-серверною моделлю: фронтенд-частина реалізована з використанням Vue.js + Vite, серверна частина — на Golang, з використанням REST API, JWT-аутентифікації, та MySQL як системи управління базами даних. Передбачено також можливість резервного копіювання, експорту/імпорту даних та сповіщень для користувачів.

## 1.2 Мета

Метою розробки інформаційної системи E-RequestControl є створення веборієнтованого інструменту для ефективного управління внутрішніми заявками в межах організації. Система покликана автоматизувати процес прийому, обробки, сортування та контролю заявок, що надходять від співробітників або структурних підрозділів. Таким чином, вона дозволяє мінімізувати ручну працю, зменшити кількість помилок, підвищити оперативність реагування та забезпечити прозорість процесів.

Основними завданнями, які ставляться перед продуктом, є:

* реалізація інтуїтивно зрозумілого та зручного інтерфейсу для користувачів різного рівня доступу;
* впровадження гнучкої системи аутентифікації та авторизації користувачів із розмежуванням прав доступу;
* забезпечення збереження, резервного копіювання та захисту даних;
* підтримка основних операцій з обробки заявок (створення, редагування, зміна статусу, видалення, перегляд історії);
* впровадження механізмів сповіщення користувачів про зміну статусу заявки чи інші важливі події;
* створення RESTful API для можливого майбутнього розширення або інтеграції з іншими сервісами;
* надання функцій для експорту й імпорту даних у зручних форматах (XLXS, SQL) для внутрішньої звітності або переносу.

Додатково, в рамках підсистеми автоматизованого розгортання та підтримки інфраструктури, система E-RequestControl вирішує такі завдання:

* використання інфраструктури як коду за допомогою Terraform для автоматизованого створення хмарних ресурсів (EKS, ECR, S3, IAM тощо);
* автоматизація CI/CD процесів за допомогою GitHub Actions — від перевірки коду до деплойменту в Kubernetes-кластер;
* контейнеризація компонентів системи з використанням Docker, що забезпечує портативність і стабільність середовища виконання;
* керування релізами через Helm для зручного оновлення й масштабування сервісів;
* налаштування автоматичного отримання TLS-сертифікатів за допомогою Cert-Manager;
* впровадження моніторингу за допомогою Prometheus і Grafana для збору та візуалізації метрик продуктивності системи;
* централізоване логування з використанням Loki та можливістю пошуку логів через Grafana;
* підвищення надійності інфраструктури завдяки автоматичному масштабуванню, балансуванню навантаження та ізоляції середовищ.

Реалізація цих завдань забезпечує високий рівень автоматизації, стабільності й масштабованості системи, що є критично важливим для її подальшого розвитку та підтримки в умовах реального навантаження.

## Межі

Для забезпечення цілісності та реалізованості продукту в рамках дипломного проєкту було встановлено чіткі межі функціональності системи E-RequestControl, які визначають обсяг розробки, особливості архітектури та технічних рішень.

Функціональні межі:

* система розроблена як вебзастосунок і не має окремого мобільного клієнта. Проте інтерфейс частково адаптивний для перегляду з мобільних пристроїв;
* користувачі системи поділяються на кілька ролей, зокрема: адміністратор, звичайний користувач (співробітник. Кожна роль має обмежений доступ до функціоналу відповідно до рівня прав;
* система забезпечує автентифікацію з використанням JWT-токенів, але не передбачає інтеграцію з OAuth, LDAP або іншими сторонніми сервісами ідентифікації;
* заявки обробляються в межах чітко визначеного життєвого циклу, який не можна динамічно змінювати з інтерфейсу;
* сповіщення реалізовані у вигляді візуальних повідомлень у інтерфейсі (toast), без використання електронної пошти, SMS чи інтеграцій з месенджерами;
* експорт/імпорт доступні лише адміністратору та здійснюються через простий інтерфейс із обмеженим набором параметрів;
* система передбачає резервне копіювання лише вручну через API або кнопку в інтерфейсі (без автоматичного планування на сервері).

Технічні межі:

* серверна частина реалізована мовою програмування Go, із застосуванням REST API. Гарантується взаємодія тільки з визначеним клієнтським застосунком;
* база даних – MySQL; не передбачена підтримка інших СУБД (наприклад, PostgreSQL чи MongoDB);
* інтерфейс розроблений з використанням Vue.js + Vite;
* система працює у середовищі локального хостингу або контейнеризації (Docker), з налаштуванням на публічне хмарне середовище (AWS);
* для перевірки якості коду застосовується SonarCloud
* міжнародна підтримка (багатомовність) у першій версії реалізована.

Організаційні межі:

* продукт створений у рамках дипломної роботи, тому не передбачає комерційного використання, ліцензування або відповідності стандартам якості розробки на виробничому рівні;
* Система орієнтована на обмежену кількість одночасних користувачів (до 50), тому не оптимізована для високих навантажень або масштабування.

## Посилання

1. Посилання на репозиторій з програмною системою:

<https://github.com/IceZyzel/E-RequestControl>

1. Посилання на репозиторій IAC частини

<https://github.com/IceZyzel/e-requestcontrol-iac>

# Загальний опис

## 2.1 Перспективи продукту

Програмне забезпечення E-RequestControl є вебзастосунком для управління електронними заявками, який призначено для автоматизації процесів створення, обробки, перегляду, сортування, імпорту, експорту, бекапу та контролю заявок в організації. Система підтримує багаторівневу рольову модель, що дозволяє розмежувати доступи відповідно до посадових обов’язків працівників.

Проєкт реалізований у вигляді клієнт-серверної архітектури з REST API. Серверна частина побудована на мові програмування Go (Golang) з використанням бібліотеки Gin, база даних реалізована на основі MySQL, а клієнтська частина — на основі Vue 3 з використанням Vite для швидкої розробки та збірки.

Завдяки гнучкій архітектурі та застосуванню DevOps-підходів (Docker, Helm, Terraform, CI/CD через GitHub Actions), програмне забезпечення може бути легко адаптоване до різних інфраструктурних середовищ, зокрема до публічних хмарних платформ, таких як AWS.

У перспективі система може бути доповнена:

* інтеграцією з корпоративною поштою або месенджерами для сповіщень;
* підтримкою мобільної версії інтерфейсу;
* розширеною аналітикою для адміністративного контролю;
* підключенням зовнішніх API для обміну даними.
* використання GitOps-підходу для централізованого управління станом інфраструктури через репозиторій;
* потенційне впровадження Cloudflare як зворотного проксі та CDN для покращення продуктивності, забезпечення DDoS-захисту, WAF та зручного керування TLS/SSL сертифікатами.

## 2.2 Функції продукту

Система E-RequestControl дозволяє створювати, редагувати та керувати електронними заявками в межах організації. Користувачі можуть залишати запити, вказуючи опис та адресуючи певній особі. Кожна заявка отримує статус (нова, оновлена тощо), що дозволяє відстежувати її виконання.

Адміністратори можуть призначати ролі, контролювати всі заявки, формувати резервні копії та експортувати/імпортувати дані у форматі XLSX, SQL. Реалізовані пошук і фільтрація заявок за різними параметрами, а також система повідомлень для інформування користувачів про зміни. Усе це забезпечує ефективне управління внутрішніми запитами та зручну взаємодію між працівниками.

В межах проєкту реалізована підсистема автоматизованого розгортання і підтримки інфраструктури, що дозволяє швидко та безпомилково розгортати сервіс у різних середовищах за допомогою контейнеризації (Docker) та оркестрації (Kubernetes). Використання CI/CD-пайплайнів забезпечує безперервну інтеграцію, тестування та доставку оновлень, підвищуючи стабільність і надійність роботи системи.

## 2.3 Характеристика користувачів

Система E-RequestControl призначена для взаємодії з двома основними категоріями користувачів: звичайними користувачами та адміністраторами. Кожна з ролей має власний набір функціональних можливостей, що відповідає її задачам у системі.

Звичайні користувачі — це працівники або користувачі, які потребують створення та управління електронними заявками. Вони мають доступ до власної панелі, де можуть:

* реєструватися та авторизовуватись у системі;
* створювати нові заявки з зазначенням теми, опису та пріоритету;
* переглядати список своїх заявок, статус їх обробки;
* редагувати або видаляти заявки, які ще не були оброблені;
* переглядати сповіщення щодо змін у заявках або нових повідомлень;
* видаляти неактуальні сповіщення.

Для взаємодії із системою від звичайного користувача не вимагається глибоких технічних знань. Інтерфейс інтуїтивно зрозумілий, простий у використанні.

Адміністратори — це користувачі з розширеними правами, які відповідають за загальне управління системою. Їх функціонал включає:

* створення, редагування та видалення користувачів;
* перегляд списку всіх зареєстрованих акаунтів;
* перегляд усіх заявок, поданих у системі;
* можливість видалення заявок і сповіщень;
* виконання дій із резервним копіюванням бази даних;
* імпорт та експорт даних (наприклад, для перенесення або аналізу заявок).

Адміністратори повинні мати базове розуміння структури даних та принципів роботи інформаційних систем, адже вони несуть відповідальність за підтримку та контроль функціонування системи.

У майбутньому можлива розширена рольова модель, з розподілом прав доступу за категоріями працівників, залежно від їх посадових обов’язків та рівня відповідальності.

## 2.4 Загальні обмеження

Система управління електронними заявками розрахована на роботу на сервері з мінімальними апаратними ресурсами, необхідними для стабільної роботи Golang-сервера та бази даних MySQL. Продуктивність бази даних і кількість одночасних підключень можуть обмежувати швидкість обробки запитів, особливо при великому навантаженні. Архітектура підтримує середній рівень користувачів і заявок, але для масштабування потрібні додаткові рішення.

Безпека системи базується на JWT-аутентифікації та використанні HTTPS, тому важливо дотримуватися належного зберігання ключів та оновлень. Клієнтська частина працює лише в сучасних браузерах із підтримкою JavaScript, а розмір файлів для завантаження обмежений задля уникнення перевантажень. Час відповіді системи становить до 1–3 секунд за нормального навантаження, але може збільшуватися при перевантаженні. Автоматизоване розгортання системи залежить від правильності налаштувань середовища і скриптів.

Автоматизоване розгортання системи реалізоване за допомогою контейнеризації (Docker) та використання оркестрації Kubernetes, що забезпечує гнучкість і швидкість деплойменту у різні середовища. CI/CD пайплайни гарантують безперервну інтеграцію, тестування та доставку оновлень, мінімізуючи людський фактор і підвищуючи стабільність системи. Крім того, передбачено базовий моніторинг стану сервісів та логування для оперативного виявлення і реагування на інциденти.

## 2.5 Припущення і залежності

Припущення:

* П-1. Користувачі системи мають базові навички роботи з веб-браузером та доступ до сучасних браузерів із підтримкою JavaScript;
* П-2. Серверне середовище підтримує стабільне підключення до інтернету і має необхідні ресурси для запуску Golang-сервера та бази даних MySQL;
* П-3. Всі ключові компоненти системи (сервер, база даних, клієнтська частина) будуть підтримуватися та оновлюватися відповідно до вимог безпеки;
* П-4. Автоматизоване розгортання буде виконуватися у контрольованому середовищі з доступом до необхідних скриптів і налаштувань.

Залежності:

* З-1. Система залежить від надійної роботи сервера бази даних MySQL, яка забезпечує збереження та обробку даних заявок.
* З-2. Аутентифікація і авторизація базуються на використанні JWT-токенів, що потребує коректної роботи механізмів генерації та перевірки токенів.
* З-3. Клієнтська частина залежить від сучасних браузерів, які підтримують стандарти HTML5, CSS3 та JavaScript ES6+.
* З-4. Коректність роботи автоматизованого розгортання залежить від стабільності робочого середовища та правильності налаштувань CI/CD процесів.

# 3 Конкретні вимоги

## 3.1 Вимоги до зовнішніх інтерфейсів

## 3.1.1 Інтерфейс користувача

Після запуску клієнтської частини E-RequestControl користувач автоматично потрапляє на сторінку авторизацію, за допомогою якої зможе потрапити в систему після вводу облікового ім’я користувача та паролю (див. рис. 3.1).

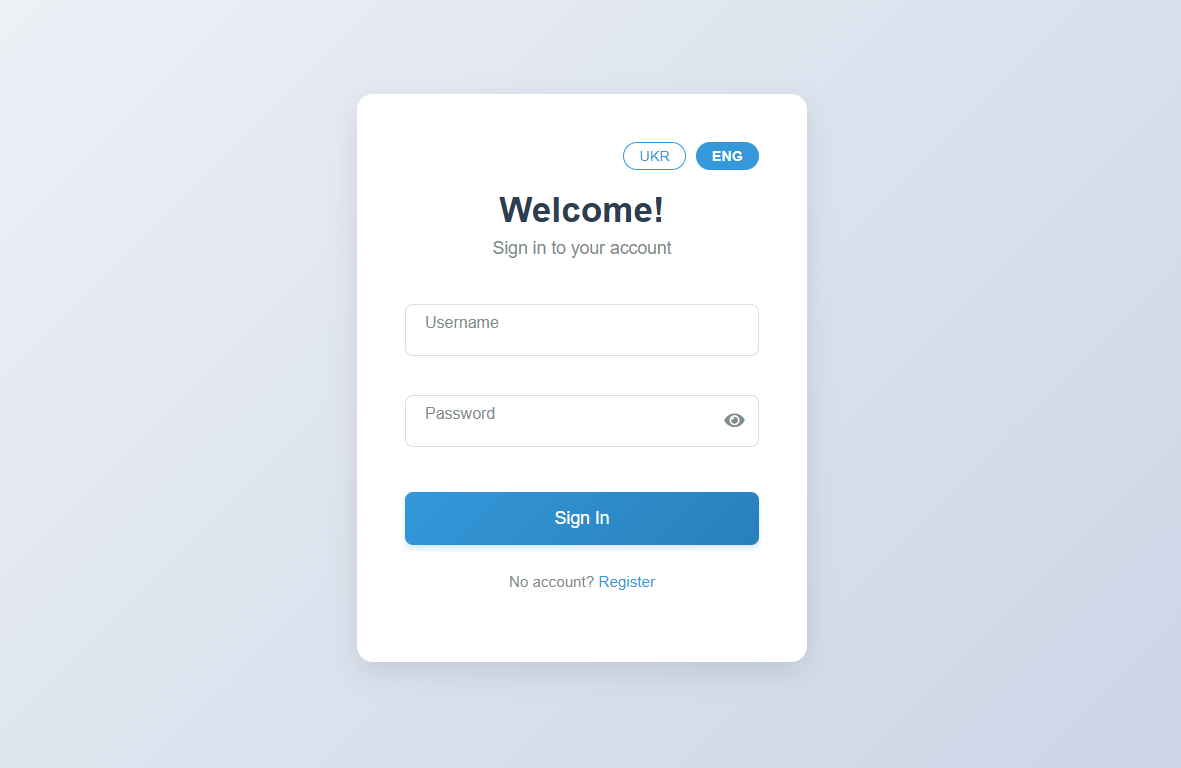


Рисунок 3.1 – Сторінка входу в систему

Після успішного входу в систему, користувач потрапляє на особисту панель управління заявками (див. рис. 3.2), де має можливість здійснювати подальші дії відповідно до своїх повноважень, тобто створювати заявки, адресувавши їх іншому користувачу; редагувати заявки та видаляти їх.

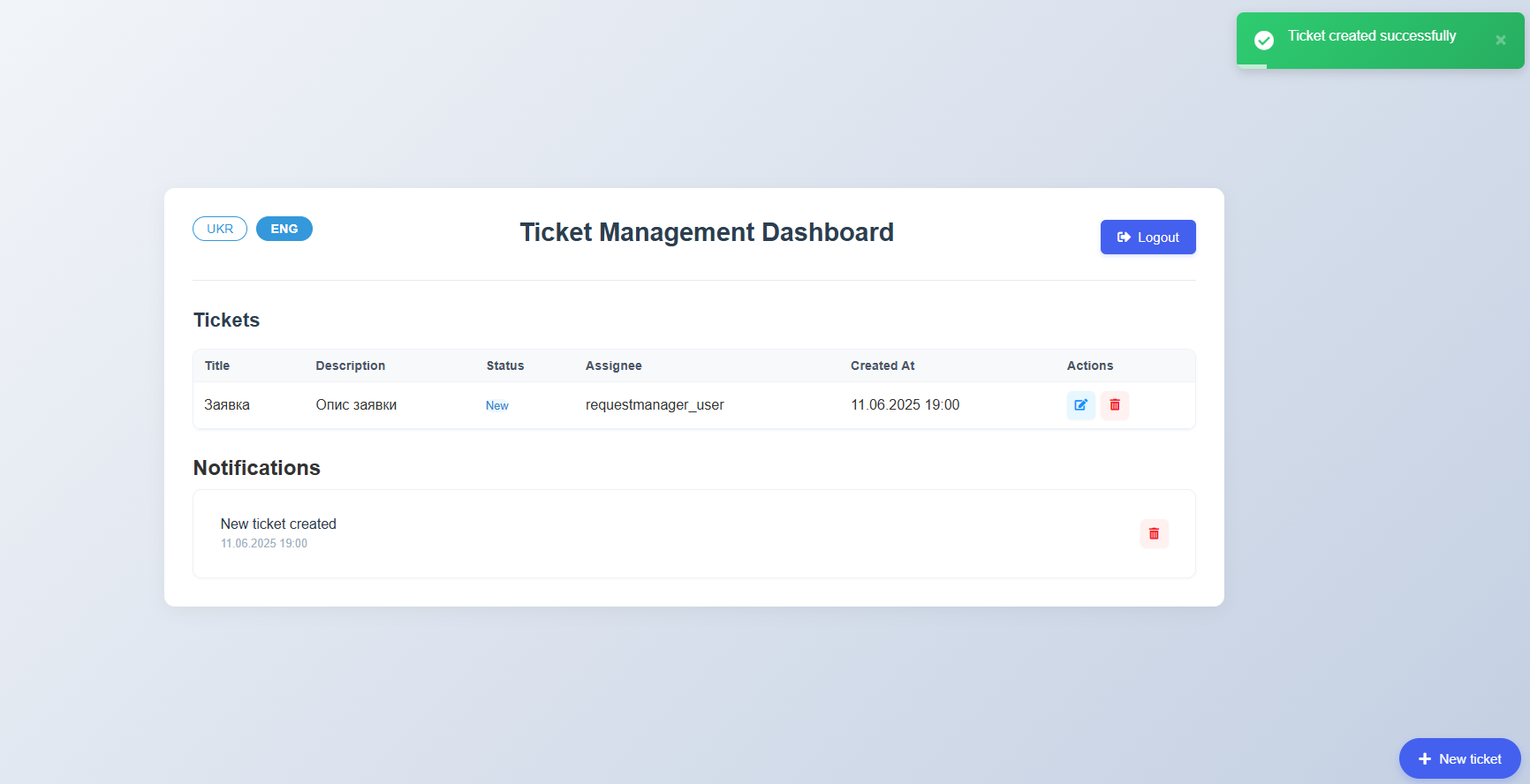


Рисунок 3.2 – Панель управління заявками

При оновленні заявки статус змінюється на “Оновлено”, додається відповідне сповіщення (див. рис. 3.3). Клієнтська частина повністю підтримує локалізацію, враховуючи дані з серверної частини, такий результат досягається завдяки мапінгу.

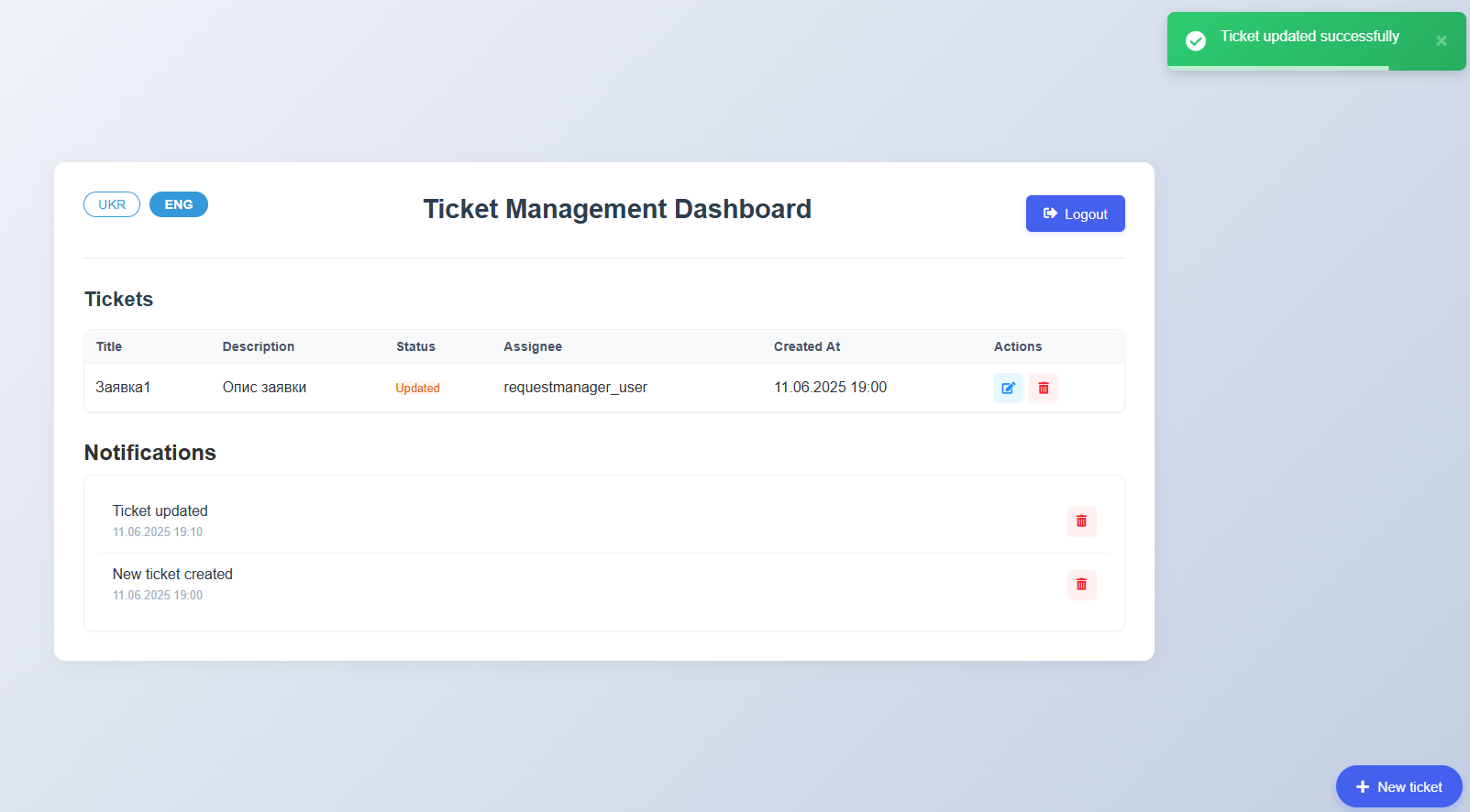


Рисунок 3.3 – Функціонал оновлення заявки

Панель адміністратора надає розширений функціонал (див. рис. 3.4), призначений для повноцінного управління системою та її даними. Зокрема, вона забезпечує доступ до повного списку зареєстрованих користувачів із можливістю перегляду, редагування та видалення облікових записів. Крім того, реалізовано функції керування заявками, зокрема їх перегляд, фільтрацію за ключовими параметрами (статус, відправник, отримувач), а також можливість видалення заявок.

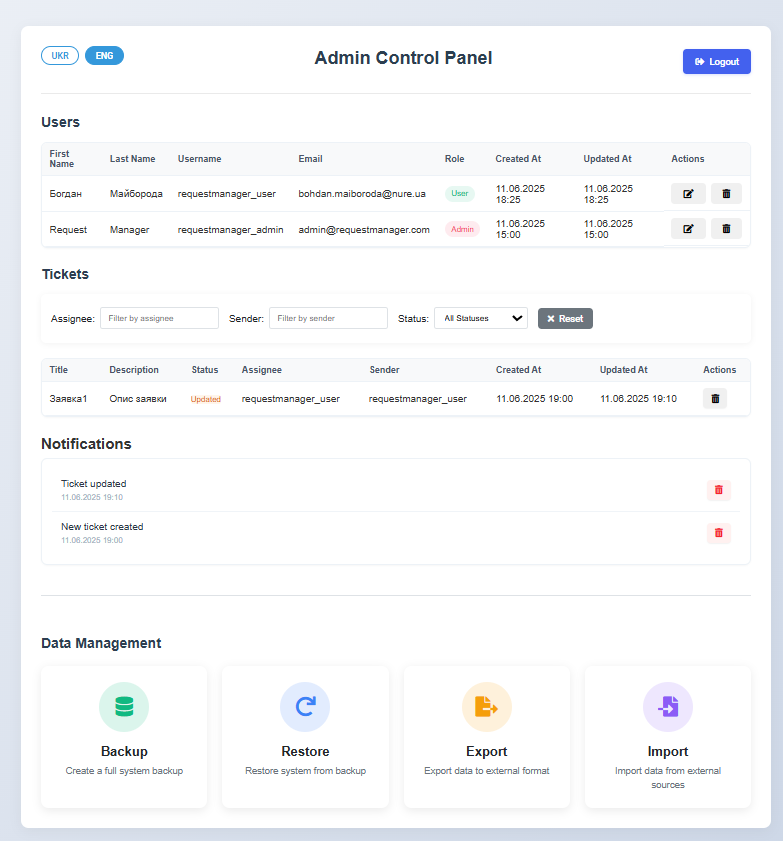


Рисунок 3.4 – Панель адміністратора

Адміністративна панель також передбачає інструменти для роботи з даними: підтримується створення резервної копії структури бази даних (MySQL) та відновлення інформації з попередньо збереженого SQL-файлу. Додатково реалізовано функціонал експорту даних у формат .xlsx, що дає змогу формувати звіти або зберігати інформацію для зовнішньої обробки, а також імпорту даних з файлів .xlsx у систему.

У випадку відсутності облікового запису, користувач має можливість зареєструватись в системі (див. рис. 3.5), при переході на відповідну сторінку. Для забезпечення достатньої захищеності облікового запису, необхідно, щоб пароль відповідав певним вимогам:

* мінімальну довжину 8 символів;
* включати в себе один спец-символ;
* містити одну цифру.

Введене значення повинно відповідати загальноприйнятому формату, що включає локальну частину, символ «@» та доменне ім’я. Крім того, передбачено перевірку на унікальність адреси електронної пошти в системі з метою запобігання дублюванню облікових записів. У разі виявлення некоректного формату або повторного використання вже зареєстрованої адреси, система виводить відповідне повідомлення про помилку, забезпечуючи таким чином коректність і цілісність даних під час реєстрації користувача.

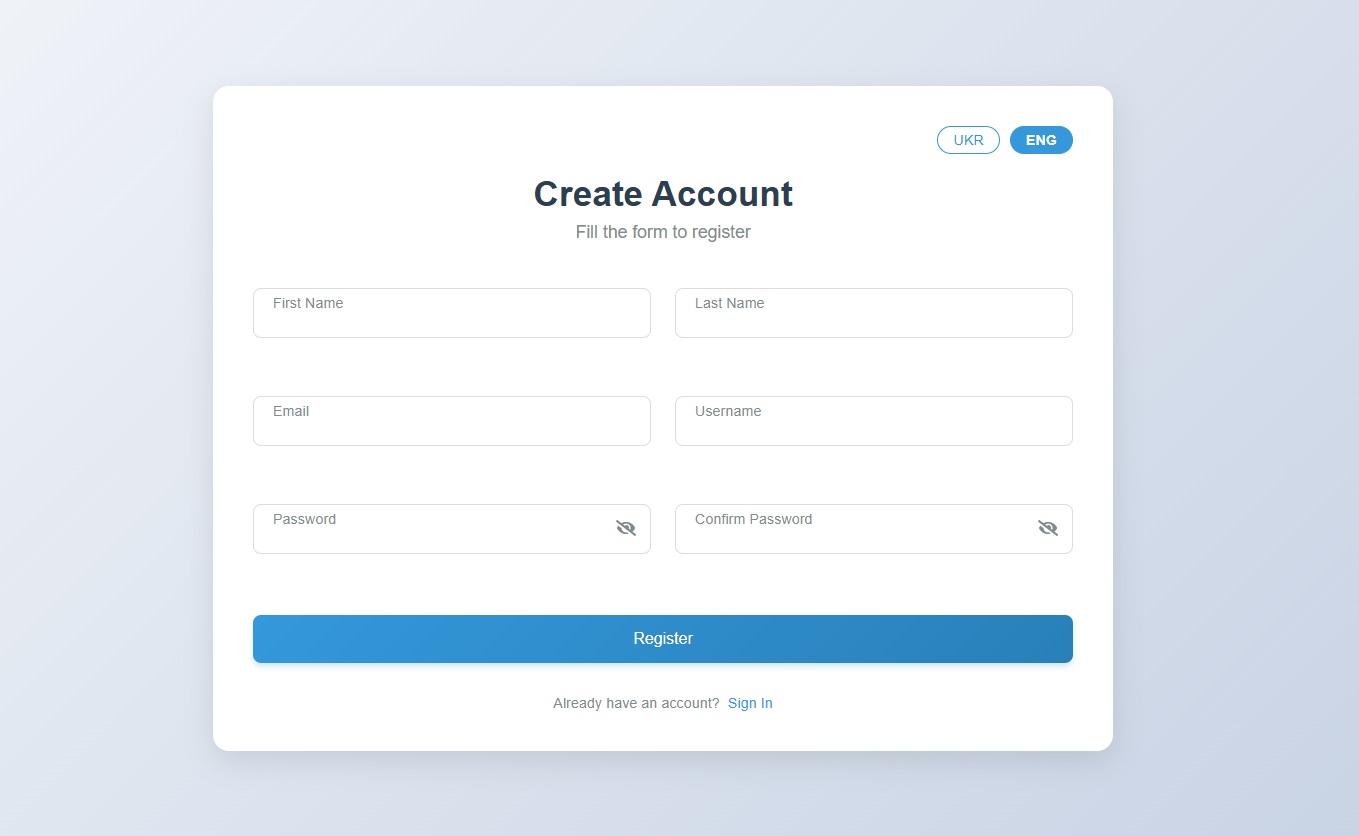


Рисунок 3.5 – Сторінка реєстрації користувача

## 3.1.2 Програмний інтерфейс

Інтерфейс підтримує обробку запитів у форматі JSON. Дані, отримані від клієнта, проходять валідацію, після чого система повертає відповідь також у форматі JSON. Усі запити до захищених маршрутів повинні містити JWT-токен, який перевіряється перед обробкою.

Клієнтська частина, розроблена з використанням фреймворку Vue.js, взаємодіє з API через HTTP-бібліотеку Axios. Інтерфейс побудовано таким чином, щоб за потреби легко додавати інші типи клієнтів — мобільні додатки або зовнішні сервіси.

## 3.1.3 Комунікаційний протокол

Взаємодія між клієнтом і сервером системи відбувається через протокол HTTP/1.1. Передача даних здійснюється у форматі JSON, що забезпечує простоту та уніфікованість обміну інформацією.

Запити до API надсилаються методами GET, POST, PUT та DELETE залежно від необхідної операції. Для захищених операцій використовується JWT-аутентифікація: токен передається в заголовку кожного запиту. У разі відсутності або недійсності токена сервер повертає відповідну помилку (401 Unauthorized).

У системі передбачено підтримку HTTPS-протоколу для забезпечення безпечної передачі даних між клієнтом і сервером.

## 3.1.4 Обмеження пам’яті

Система не вимагає значного обсягу оперативної пам’яті для нормальної роботи. Для запуску серверної частини достатньо 256–512 МБ RAM при невеликому навантаженні. У разі масштабування або збільшення кількості одночасних користувачів, рекомендовано збільшити обсяг оперативної пам’яті до 1–2 ГБ для забезпечення стабільної роботи.

## 3.1.5 Операції

Серверна частина реалізує RESTful API за принципом архітектури клієнт–сервер. API забезпечує доступ до функціоналу через HTTP-запити:

* GET – отримання даних;
* POST – створення нових записів;
* PUT – оновлення існуючих записів;
* DELETE – видалення записів.

Кожен запит обробляється відповідним маршрутом (endpoint), після чого виконується бізнес-логіка сервісного рівня та здійснюється доступ до бази даних через репозиторій.

## 3.1.6 Функції продукту

ГФ-1. Система повинна надавати можливість реєстрації нових користувачів з вибором ролі (користувач або адміністратор).

ГФ-2. Після авторизації користувач повинен мати доступ до персональної панелі з можливістю перегляду, створення та редагування заявок.

ГФ-3. Адміністратор повинен мати доступ до окремої адміністративної панелі з можливістю перегляду всіх заявок, зміни їх статусів, призначення відповідальних осіб тощо.

ГФ-4. Система повинна зберігати всі заявки в базі даних з можливістю фільтрації, сортування та пошуку за різними критеріями (статус, дата створення, виконавець тощо).

ГФ-5. Кожна заявка повинна мати унікальний ідентифікатор, дату створення, автора, опис проблеми, статус і при потребі прикріплені файли.

ГФ-6. Система повинна автоматично створювати резервні копії бази даних згідно заданого розкладу.

ГФ-7. Повинна бути реалізована функція експорту даних заявок у форматі SQL/Excel, а також імпорту з цих форматів.

ГФ-8. Система повинна відображати сповіщення користувачам про зміну статусу їхніх заявок або інші дії, пов’язані з ними.

ГФ-9. Доступ до функцій системи повинен регулюватися ролями користувачів — звичайний користувач не може редагувати заявки інших або бачити адміністративну частину.

ГФ-10. Під час аутентифікації система повинна повертати JWT токен, який використовується для захисту наступних запитів.

## 3.1.7 DevOps-вимоги

* автоматизація процесу розгортання (CI/CD): Забезпечити безперервну інтеграцію і доставку оновлень системи з автоматичним тестуванням та деплойментом у тестове і продуктивне середовища через налаштовані pipeline (GitHub Actions);
* контейнеризація і оркестрація: Усі компоненти системи повинні бути упаковані у Docker-контейнери для забезпечення портативності і відтворюваності середовищ. Оркестрація контейнерів (наприклад, за допомогою Kubernetes) має гарантувати масштабованість і високу доступність;
* моніторинг та логування: Впровадити системи моніторингу (Prometheus, Grafana) і централізованого логування (Loki, ELK), щоб забезпечити прозорість роботи сервісів, своєчасне виявлення збоїв і швидке реагування на інциденти.

# 3.2 Вимоги до зовнішніх інтерфейсів

Система повинна надавати зручний та інтуїтивно зрозумілий графічний інтерфейс користувача (GUI) для двох основних ролей — звичайного користувача та адміністратора. Кожна роль має доступ до окремого функціоналу:

* Користувач після входу бачить персональну панель, де може переглядати, створювати, редагувати або видаляти власні заявки. Також відображаються статуси заявок, повідомлення та можливість експорту даних.
* Адміністратор має доступ до окремої панелі управління, в якій можна бачити всі заявки системи, змінювати їхній статус, призначати відповідальних, керувати користувачами та переглядати системну активність.

Також система передбачає інтеграцію з іншими сервісами за допомогою RESTful API. Зовнішні системи можуть отримувати доступ до основних функцій через HTTP-запити, дотримуючись стандарту обміну даними у форматі JSON та використовуючи JWT для авторизації.

# 3.3 Атрибути програмного продукту

## 3.3.1 Надійність

Система повинна стабільно функціонувати протягом тривалого часу без збоїв. Усі критичні помилки обробляються на серверному рівні та логуються для подальшого аналізу. Реалізовано базові механізми відновлення після збоїв, зокрема резервне копіювання бази даних.

## 3.3.2 Безпека

Захист інформації є критичним аспектом системи. Усі запити до захищених маршрутів вимагають JWT-токен для аутентифікації користувача. Крім того, система використовує HTTPS для шифрування трафіку. Ролі користувачів визначають рівень доступу до функцій. Валідація даних здійснюється як на клієнтській частині, так і на серверній.

## 3.3.3 Супроводжуваність

Система має чітко поділену архітектуру на окремі компоненти: хендлери, сервіси та репозиторії. Це спрощує оновлення, рефакторинг і додавання нового функціоналу. Весь код добре структурований і документований, що дозволяє швидко розібратися новим розробникам.

## 3.3.4 Переносимість

Програмний продукт може бути розгорнутий на будь-якому сервері, де доступні Go, MySQL та Node.js для клієнтської частини. Конфігурації зберігаються у .env-файлах, що дозволяє легко переносити систему між середовищами (локальне, тестове, продуктивне).

## 3.3.5 Продуктивність

Система оптимізована для швидкої обробки HTTP-запитів. Використання легковагового фреймворку Gin забезпечує високу продуктивність серверної частини. Завдяки SPA-архітектурі інтерфейс працює плавно, без затримок. Продуктивність зберігається стабільною навіть при значній кількості запитів завдяки ефективній роботі з базою даних через sqlx.

## 3.3.6 Автоматизація розгортання та інтеграції

Система підтримує безперервну інтеграцію та безперервне розгортання (CI/CD) для швидкого й безпомилкового оновлення коду. Автоматичні пайплайни виконують збірку, тестування та деплой у тестове та продуктивне середовище, що мінімізує ручні операції і ризики помилок.

## 3.3.7 Масштабованість та портативність середовища

Використання контейнеризації (Docker) забезпечує переносимість системи між різними середовищами і спрощує масштабування. Оркестрація контейнерів (наприклад, Kubernetes) дозволяє гнучко регулювати ресурси залежно від навантаження, підтримуючи стабільність роботи при зростанні кількості користувачів.

## 3.3.8 Моніторинг та логування

Інтегровані системи моніторингу (Prometheus, Grafana) та централізованого логування (Loki, ELK) дозволяють відслідковувати стан сервісів у реальному часі, оперативно реагувати на помилки та забезпечують аналіз роботи системи для її подальшого вдосконалення.

# 3.4 Вимоги до бази даних

Для зберігання даних у системі використовується реляційна база даних MySQL. Вона забезпечує надійне збереження заявок, користувачів, ролей, сповіщень, логів змін, а також резервних копій. Доступ до бази даних здійснюється за допомогою бібліотеки sqlx у серверній частині, що дозволяє зручно працювати з SQL-запитами та мапити результати в структури Go.

Основні вимоги до структури БД:

* нормалізація — таблиці побудовані за принципами 2-ї та 3-ї нормальних форм для усунення дублювання даних і забезпечення цілісності;
* зовнішні ключі — використовуються для збереження зв’язків між таблицями: наприклад, заявки пов’язані з користувачами, ролями, статусами;
* таймстемпи (CreatedAt, UpdatedAt) — для відстеження змін у кожному записі;
* резервне копіювання — реалізовано механізм автоматичного створення .sql-дампів за розкладом для відновлення системи у разі збою;
* індекси — створено для основних пошукових полів (наприклад, status, user\_id, created\_at) для підвищення швидкодії при великій кількості записів;
* безпека — обмежено права доступу до бази даних (тільки сервер має доступ), паролі зберігаються у вигляді хешів (bcrypt);

# 3.5 Інші вимоги

## 3.5.1 Журнали подій

Система повинна вести базове логування подій для виявлення помилок, моніторингу активності та забезпечення безпеки. Усі помилки серверної частини (наприклад, невдала авторизація, помилка бази даних, виключення) мають логуватися з позначкою часу, рівнем важливості (info, warning, error) та описом події. Логи зберігаються у текстовому файлі або в окремій таблиці бази даних.

## 3.5.2 Резервне копіювання

Повинна бути реалізована система автоматичного створення резервних копій бази даних. Копії створюються у форматі .sql та зберігаються у визначеній директорії з датою створення в назві файлу. Рекомендується встановити щонайменше щоденне резервне копіювання з можливістю відновлення даних у разі збою.

## 3.5.3 Валідація даних

Усі вхідні дані, що надходять через веб-інтерфейс або API, повинні проходити перевірку на коректність. Наприклад:

* email має відповідати валідному формату;
* поля не повинні бути порожніми (якщо обов’язкові);
* довжина тексту повинна бути обмежена згідно з вимогами до БД.

Це дозволяє зменшити ризики SQL-інʼєкцій, некоректних записів та помилок на стороні клієнта.

## 3.5.4 Мови інтерфейсу

У поточній версії система реалізована двома мовами — українською та англійською мовами. У перспективі можливе розширення списку перекладів.

## 3.5.5 Масштабованість

Система розроблена таким чином, щоб її можна було масштабувати:

* додавати нові модулі без значних змін у структурі коду;
* розгортати на кількох серверах;
* змінювати сховище даних (наприклад, перехід з MySQL на PostgreSQL).

## 3.5.6 Якість коду та аналіз

Впровадження системи автоматичного аналізу якості коду за допомогою SonarCloud, що інтегрована у CI/CD пайплайн.

Результати аналізу доступні розробникам для виправлення виявлених вразливостей, дублікатів коду та покращення загальної якості.

# 4 Додаткові матеріали

## 4.1 Схеми баз даних

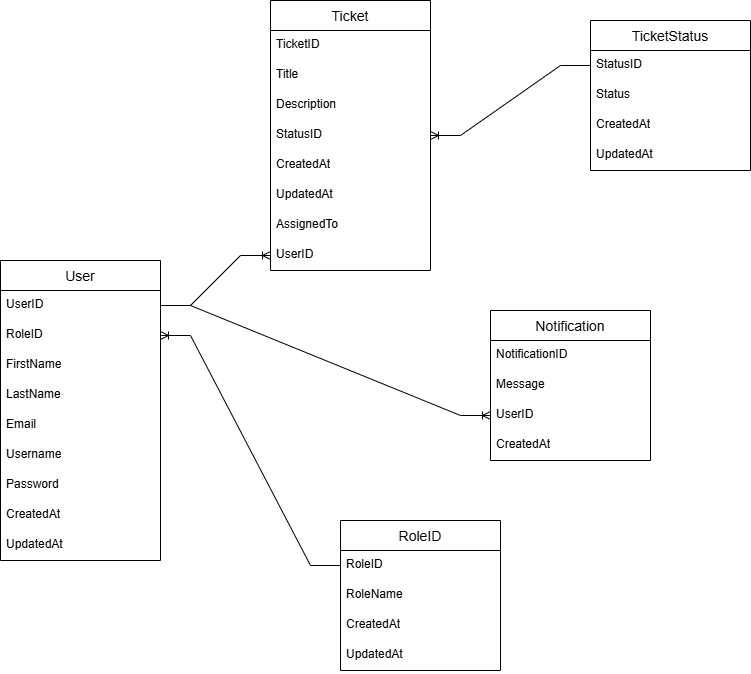


Рисунок 4.1 – Схема бази даних

## 4.2 Діаграми прецедентів

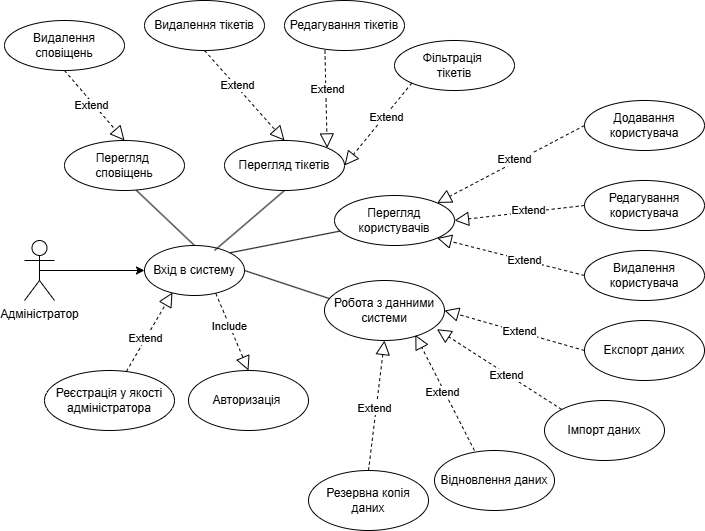


Рисунок 4.2 – Use Case діаграма для ролі Адміністратора

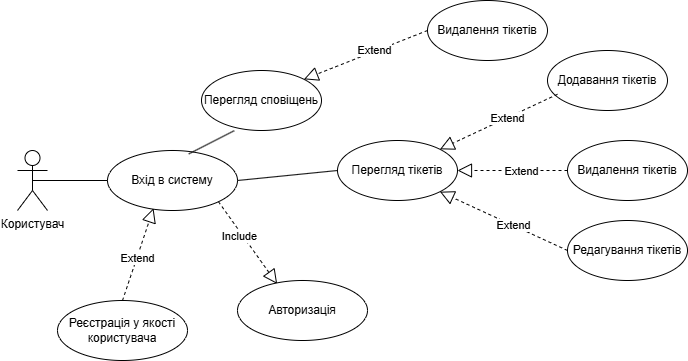


Рисунок 4.3 – Use Case діаграма для ролі Звичайного користувача

## 4.4 Схема DevOps конвеєрів

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4.4 – Pipeline інфраструктурного коду Terraform

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4.5 – Діаграма процесу CI/CD