Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА КОНСОЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ЗАДАЧ НА MOBI GO

DEVELOPMENT OF A CONSOLE APPLICATION FOR TASK ACCOUNTING IN GO

спеціальність 122 Комп'ютерні науки

освітня програма Комп'ютерні науки

Виконала: здобувачка вищої освіти
групи КНз-41
Бобела Анастасія Сергіївна
_____
(підпис)

Керівник: асистент
Неділько Ольга Володимирівна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2026 р.

Гарант освітньої програми:
д.п.н., професор
Тулашвілі Юрій Йосипович
_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *122 «Комп'ютерні науки»*
Освітня програма: «*Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Бобелі Анастасії Сергіївній
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка консольного застосунку для обліку задач на мові Go»*.
Керівник роботи: асистент Неділько О. В.
затверджені наказом закладу вищої освіти від *«16» січня 2026 р.* *№ 32/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«2» червня 2026 р.*
3. Вихідні дані до роботи: *предметна область управління задачами та організації командної роботи, технології розробки програмного забезпечення (Go, REST API), засоби роботи з базами даних (SQLite, modernc.org/sqlite), вимоги до інформаційних систем та забезпечення інформаційної безпеки.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблем автоматизації управління задачами та організації колаборативної роботи користувачів, постановка завдання, формування вимог до розроблюваної інформаційної системи, обґрунтування вибору технологій та засобів розробки, практична реалізація об'єкта проектування, розробка структури бази даних, обґрунтування принципів захисту інформаційно-комп'ютерної системи на основі JWT-автентифікації та bcrypt-хешування паролів, тестування та налагодження програмного забезпечення.*
5. Перелік графічного матеріалу: *26 рисунків, 19 лістингів, 4 таблиці.*
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Неділько О. В.*		
Теоретичне дослідження	*Неділько О. В.*		
Практична реалізація	*Неділько О. В.*		
Спеціальні розрахунки	*Неділько О. В.*		
Показник запозичень тексту	***___%***		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОП	*Тулашвілі Ю. Й.*		

7. Дата видачі завдання *«16» січня 2026 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*до 17.02.2026 р.*	
2	*Провести аналіз проблеми розробки та впровадження об'єкту проектування*	*до 28.02.2026 р.*	
3	*Розробити функціональну схему роботи програмного продукту*	*до 12.03.2026 р.*	
4	*Описати засоби розробки об'єкта проектування*	*до 26.03.2026 р.*	
5	*Практична реалізація об'єкта проектування*	*до 30.04.2026 р.*	
6	*Провести спеціальні розрахунки*	*до 18.05.2026 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*до 02.06.2026 р.*	

Здобувач вищої освіти _____ Анастасія БОБЕЛА

Керівник роботи _____ Ольга НЕДІЛЬКО

АНОТАЦІЯ

Бобела А. С. Розробка консольного застосунку для обліку задач на мові Go. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків.

У кваліфікаційна робота бакалавра розглянуто процес розробки консольного застосунку для обліку задач мовою програмування Go. Метою роботи є створення програмного продукту для управління задачами з підтримкою авторизації користувачів, REST API та збереженням даних у базі даних SQLite.

У процесі виконання роботи було реалізовано серверну частину застосунку з використанням стандартної бібліотеки net/http, систему автентифікації на основі JWT-токенів, механізм розмежування прав доступу користувачів, а також консольний клієнт для взаємодії із системою. Для збереження даних використано SQLite, а для тестування API – Postman. Розроблений застосунок забезпечує можливість створення, редагування, видалення та спільного використання задач між користувачами. Результатом роботи є функціональний консольний застосунок, який може використовуватися для організації та контролю виконання задач.

Ключові слова: Go, REST API, JWT, SQLite, консольний застосунок, система обліку задач, CLI.

ANNOTATION

Bobela Anastasia. Development of a console application for task accounting in the Go language. Manuscript.

Bachelor’s qualification paper in the study program «Computer Science». Lutsk National Technical University. Lutsk, 2026.

The bachelor’s qualification paper consists of an introduction, four chapters, conclusions, a list of references, and appendices.

The bachelor's qualification work considers the process of developing a console application for task accounting in the Go programming language. The purpose of the work is to create a software product for task management with support for user authorization, REST API, and data storage in the SQLite database.

In the process of performing the work, the server part of the application was implemented using the standard net/http library, an authentication system based on JWT tokens, a mechanism for distinguishing user access rights, and a console client for interacting with the system. SQLite was used to store data, and Postman was used to test the API. The developed application provides the ability to create, edit, delete, and share tasks between users. The result of the work is a functional console application that can be used to organize and control task execution.

Keywords: Go, REST API, JWT, SQLite, console application, task accounting system, CLI.

ЗМІСТ

ВСТУП

Сучасний ритм життя висуває підвищені вимоги до організації особистої та командної продуктивності. Управління завданнями є невід'ємною частиною як індивідуальної роботи фахівців, так і діяльності цілих підприємств. Незважаючи на широкий спектр існуючих рішень у цій галузі, значна їх частина є або надмірно складною для розгортання та підтримки, або не надає відкритих програмних інтерфейсів для інтеграції з іншими системами, або вимагає постійного підключення до хмарних сервісів сторонніх постачальників.

Актуальність теми кваліфікаційної роботи бакалавра зумовлена потребою у легковаговому, самодостатньому та розширюваному програмному продукті для управління задачами, який може бути розгорнутий у будь-якому середовищі без зовнішніх залежностей, забезпечує безпечну автентифікацію користувачів та підтримує колаборативну роботу з задачами в рамках команди.

Метою кваліфікаційної роботи бакалавра є розробка програмного продукту «Система управління задачами з REST API та CLI-інтерфейсом», що реалізує повний цикл управління задачами з підтримкою багатокористувацького режиму, рольової моделі доступу та безпечної автентифікації на основі JWT-токенів.

Для досягнення поставленої мети вирішено такі завдання:

– проведено аналіз існуючих рішень у сфері управління задачами та визначено їх недоліки;

– спроектовано архітектуру програмного продукту з дотриманням принципів розмежування відповідальності між модулями;

– спроектовано схему реляційної бази даних з підтримкою відносин «багато-до-багатьох» між задачами та користувачами;

– реалізовано шар зберігання даних на основі вбудованої СУБД SQLite без залежності від зовнішніх серверів баз даних;

– реалізовано бізнес-логіку управління задачами, включаючи механізм колаборативного доступу та диференційованого видалення;

– реалізовано систему автентифікації та авторизації на основі JWT-токенів із рольовою моделлю доступу;

– розроблено REST HTTP API та інтерактивний CLI-інтерфейс як два незалежні клієнти єдиного бізнес-шару;

– проведено тестування функціональних компонентів системи та усунено виявлені недоліки;

– виконано ергономічну оцінку, розраховано техніко-економічні показники та описано механізми захисту інформації.

Об’єктом дослідження є процеси управління задачами в умовах багатокористувацького середовища.

Предметом дослідження є методи та засоби розробки серверного програмного забезпечення з REST API на мові програмування Go із застосуванням JWT-автентифікації та реляційних баз даних.

Практичне значення отриманих результатів полягає у створенні готового до розгортання програмного продукту, який може використовуватись як самостійне рішення для управління задачами в малих командах, як навчальний приклад реалізації REST API мовою Go, а також як основа для подальшого розширення функціоналу.

Результати дослідження були представлені на міжнародній науково-практичній конференції «Сучасні інформаційні технології: від теорії до практики» (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану проблеми

У сучасних умовах стрімкого розвитку інформаційних технологій та цифровізації суспільства особливого значення набуває ефективна організація процесів управління задачами. Як у професійній діяльності, так і в повсякденному житті користувачі стикаються з необхідністю планування, контролю та координації виконання великої кількості завдань. Це особливо актуально для IT-сфери, де процес розробки програмного забезпечення передбачає постійне управління задачами, їх розподіл між учасниками та контроль виконання.

Традиційні підходи до організації задач, такі як використання паперових записників або простих текстових файлів, є малоефективними в умовах зростання обсягу інформації та необхідності швидкого доступу до неї. Вони не забезпечують належного рівня структурованості, не дозволяють ефективно здійснювати пошук, а також не підтримують механізми спільної роботи між користувачами.

З розвитком інформаційних технологій з’явилася велика кількість програмних засобів для управління задачами, які реалізують різноманітні підходи до організації роботи. Проте, незважаючи на їх різноманітність, існує низка проблем, які залишаються актуальними.

По-перше, значна частина сучасних систем управління задачами є надмірно складною для використання, особливо для невеликих команд або окремих користувачів. Такі системи часто містять широкий набір функцій, які не завжди є необхідними, що ускладнює процес освоєння та знижує продуктивність роботи.

По-друге, багато існуючих рішень орієнтовані на використання графічного інтерфейсу користувача (GUI) та веб-технологій, що може бути незручним для користувачів, які віддають перевагу роботі через командний рядок. Відсутність

CLI-інтерфейсу обмежує можливості автоматизації, інтеграції з іншими інструментами та швидкої роботи з системою.

По-третє, значна кількість систем є комерційними продуктами, що накладає обмеження на використання повного функціоналу без придбання платної підписки. Це може бути суттєвим недоліком для індивідуальних користувачів або невеликих команд.

Ще однією важливою проблемою є обмежені можливості налаштування та інтеграції. Багато існуючих систем не надають відкритого програмного інтерфейсу (API) або мають обмежений функціонал для розширення, що ускладнює їх використання в рамках власних програмних рішень.

Крім того, питання безпеки та розмежування доступу до інформації також є актуальним. У багатьох системах відсутня гнучка система ролей або вона реалізована недостатньо ефективно, що може призводити до несанкціонованого доступу або ускладнення адміністрування.

Таким чином, аналіз сучасного стану проблеми показує, що, незважаючи на наявність великої кількості інструментів для управління задачами, існує потреба у створенні простого, ефективного та гнучкого програмного рішення, яке б поєднувало такі характеристики:

- мінімалістичний та зручний інтерфейс;
- можливість роботи через командний рядок;
- наявність програмного інтерфейсу для інтеграції;
- підтримка багатокористувацького режиму;
- реалізація механізмів автентифікації та розмежування доступу;
- простота розгортання та використання.

Враховуючи наведені недоліки існуючих рішень, доцільним є розроблення консольного застосунку для обліку задач із використанням сучасних технологій програмування, що забезпечить ефективне управління задачами, зручність використання та можливість інтеграції з іншими системами.

1.2 Аналіз аналогічних програм, баз даних, систем

У сучасному середовищі існує значна кількість програмних засобів для управління задачами, які відрізняються функціональністю, складністю використання, підходами до організації роботи та цільовою аудиторією. Для обґрунтування вибору напряму розробки доцільно розглянути найбільш поширені системи, що застосовуються на практиці.

Однією з популярних систем управління задачами є Trello (рис. 1.1). Даний сервіс реалізує підхід канбан-дошок, що дозволяє візуалізувати процес виконання завдань у вигляді колонок та карток. До основних переваг системи належать інтуїтивно зрозумілий інтерфейс, зручність використання та можливість швидкої організації задач. Крім того, система підтримує командну роботу та інтеграцію з іншими сервісами. Водночас недоліками є обмежена функціональність у безкоштовній версії, відсутність повноцінного консольного інтерфейсу, а також залежність від веб-середовища.

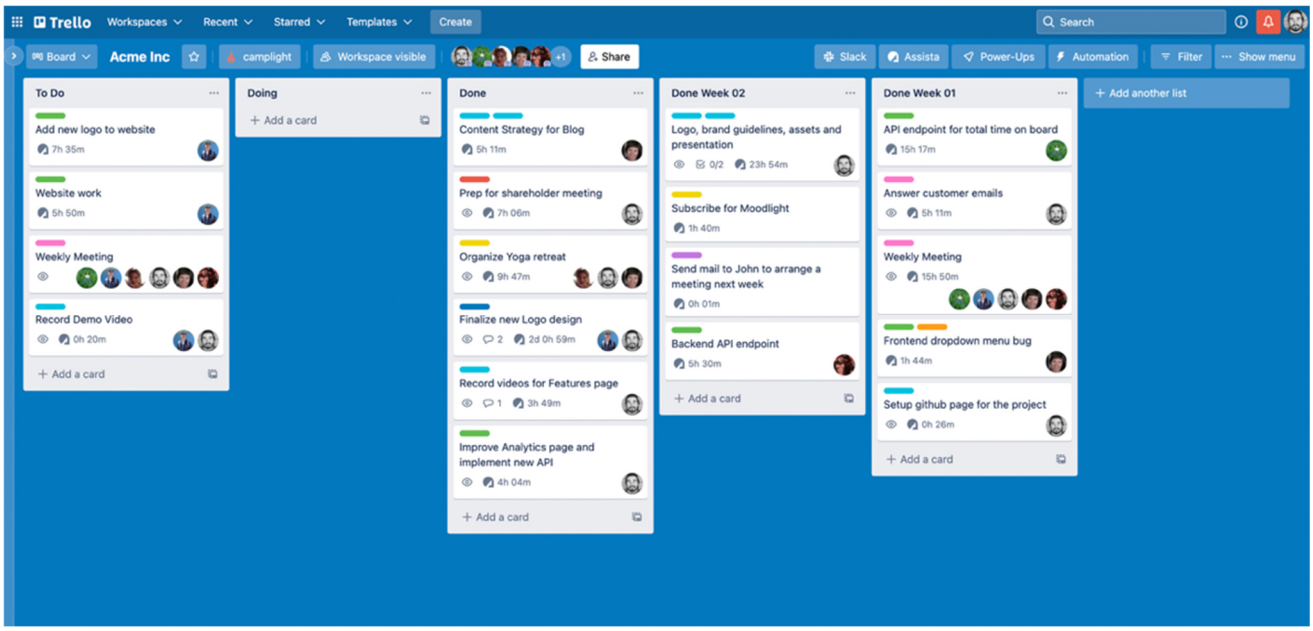

Рисунок 1.1 – Інтерфейс Trello

Іншим широко використовуваним інструментом є Jira (рис. 1.2), який орієнтований переважно на використання в процесі розробки програмного

забезпечення. Система забезпечує розширені можливості управління проєктами, підтримує різні методології розробки (зокрема Agile та Scrum), має гнучку систему налаштувань і розмежування доступу. До переваг можна віднести високу функціональність, масштабованість та інтеграцію з іншими інструментами розробки. Проте система є складною у налаштуванні та використанні, потребує значних ресурсів для розгортання, а також може бути надмірною для невеликих проєктів або індивідуального використання.

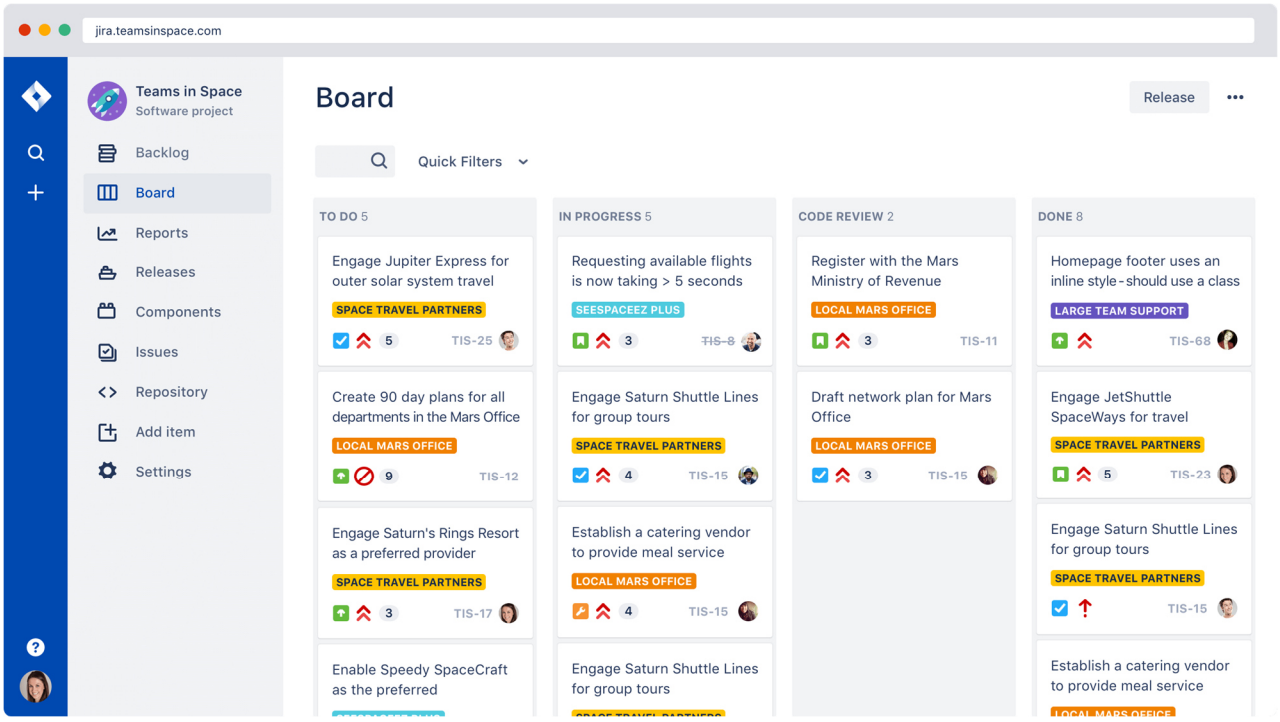

Рисунок 1.2 – Інтерфейс Jira

Ще одним прикладом є Todoist – легковаговий інструмент для персонального управління задачами. Основними перевагами системи є простота інтерфейсу, швидкість роботи та доступність на різних платформах. Сервіс підтримує базові функції планування задач, нагадування та організації списків справ.

Водночас його функціональність є обмеженою у контексті командної роботи, а частина можливостей доступна лише у платній версії. Також відсутній розвинений інтерфейс для інтеграції з іншими системами на рівні власного програмного рішення (рис. 1 3).

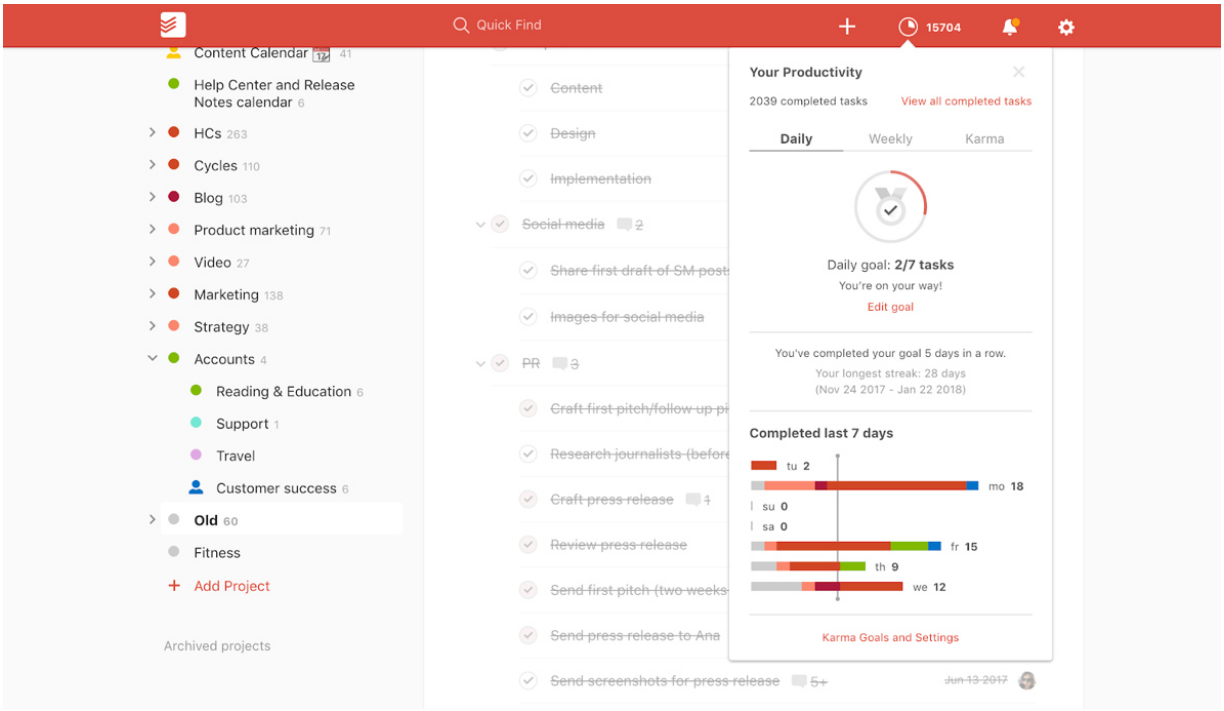

Рисунок 1.3 – Інтерфейс Todoist

Проведений аналіз показує, що існуючі системи управління задачами можна умовно поділити на дві категорії: складні багатофункціональні платформи для командної роботи та прості інструменти для персонального використання. При цьому перші характеризуються високою функціональністю, але складністю використання, тоді як другі є зручними, але мають обмежені можливості.

Жодна з розглянутих систем не поєднує одночасно такі характеристики, як:

– простота використання;
– підтримка консольного інтерфейсу;
– наявність відкритого програмного інтерфейсу (API) для інтеграції;
– можливість розгортання як автономного рішення;
– гнучка система автентифікації та розмежування доступу.

У зв’язку з цим доцільним є розроблення власного програмного рішення, яке буде орієнтоване на поєднання переваг існуючих підходів із мінімізацією їхніх недоліків.

Обраний напрям розробки передбачає створення консольного застосунку з підтримкою REST API, що забезпечує можливість як інтерактивної роботи користувача через командний рядок, так і інтеграції з іншими програмними системами. Такий підхід дозволяє досягти гнучкості у використанні та розширенні функціональності.

Як основні методи реалізації обрано:

– використання клієнт-серверної архітектури;
– застосування REST-підходу для побудови API;
– використання токен-орієнтованої автентифікації (JWT);
– реалізація рольової моделі доступу до ресурсів;
– застосування реляційної бази даних для зберігання інформації.

Таким чином, проведений аналіз аналогічних систем дозволив визначити основні вимоги до розроблюваного програмного забезпечення та обґрунтувати вибір напряму розробки, що полягає у створенні легковагового, розширюваного та зручного інструменту для управління задачами.

1.3 Аналіз і вибір засобів проектування консольного застосунку на Go

Розроблення сучасних інформаційних систем вимагає обґрунтованого вибору засобів проектування, які забезпечують необхідний рівень продуктивності, надійності, масштабованості та зручності супроводу програмного забезпечення. Вибір технологічного стеку має базуватися на особливостях поставленого завдання, вимогах до системи та результатах аналізу існуючих рішень.

У рамках даної роботи розробляється консольний застосунок для обліку задач із підтримкою REST API, що зумовлює необхідність вибору мови програмування, системи управління базами даних, засобів автентифікації, а також архітектурних підходів.

Вибір мови програмування. Як основну мову програмування було обрано Go (рис. 1.4). Даний вибір обумовлений рядом переваг, які роблять цю мову ефективною для розроблення серверних застосунків.

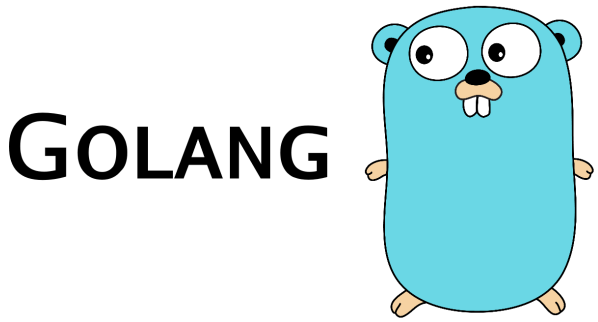

Рисунок 1.4 – Логотип мови програмування Go

По-перше, мова Go забезпечує високу продуктивність, що досягається завдяки компіляції у машинний код та ефективному управлінню пам’яттю. Це дозволяє створювати швидкодіючі застосунки без необхідності використання складних механізмів оптимізації.

По-друге, важливою перевагою є вбудована підтримка конкурентності через механізми горутин та каналів. Це дозволяє ефективно обробляти одночасні запити, що є критично важливим для серверної частини застосунку.

По-третє, Go має простий та зрозумілий синтаксис, що спрощує процес розробки та супроводу коду. Це особливо важливо в умовах створення структурованого проєкту з розподілом на модулі (API, CLI, бізнес-логіка, робота з БД) [1].

Крім того, стандартна бібліотека мови містить необхідні інструменти для роботи з мережею (пакет net/http), що дозволяє реалізувати REST API без використання сторонніх фреймворків.

Таким чином, використання мови Go дозволяє забезпечити високу продуктивність, простоту реалізації та підтримку сучасних архітектурних підходів.

Для зберігання даних у системі було обрано SQLite (рис. 1.5) – вбудовану реляційну систему управління базами даних [2].

Рисунок 1.5 – Логотип SQLite

Основною перевагою SQLite є її легковаговість та автономність. На відміну від серверних СКБД (таких як MySQL або PostgreSQL), SQLite не потребує окремого серверного процесу, що значно спрощує розгортання системи.

Серед інших переваг:

– простота використання та інтеграції;
– зберігання всієї бази даних у одному файлі;
– достатня продуктивність для малих і середніх систем;
– підтримка стандарту SQL [3].

Водночас SQLite має певні обмеження, зокрема обмежену підтримку високонавантажених систем та конкурентного доступу на запис. Проте для задачі розроблення консольного застосунку ці обмеження не є критичними [4].

Враховуючи вимоги до простоти розгортання та використання, SQLite є оптимальним вибором для даного проєкту.

У процесі розроблення застосунку було обрано клієнт-серверну архітектуру з використанням REST-підходу до організації взаємодії між компонентами.

Серверна частина реалізує REST API, що забезпечує обробку запитів, управління бізнес-логікою та взаємодію з базою даних. Клієнтська частина представлена консольним інтерфейсом (CLI), який взаємодіє з сервером через HTTP-запити.

Основними перевагами REST-підходу є:

– простота реалізації;

– використання стандартних HTTP-методів (GET, POST, PUT, DELETE);

– незалежність клієнта і сервера;

– можливість інтеграції з іншими системами [5].

Такий підхід дозволяє створити гнучку систему, яку можна розширювати, наприклад, шляхом додавання графічного інтерфейсу або мобільного клієнта без зміни серверної логіки.

JWT представляє собою компактний токен, який передається у кожному запиті та містить закодовану інформацію про користувача. Основними перевагами даного підходу є:

– відсутність необхідності зберігати сесії на сервері (stateless);

– висока швидкість обробки запитів;

– зручність використання в REST-системах;

– можливість масштабування [6].

У розроблюваній системі JWT використовується для автентифікації користувачів, а також для реалізації рольової моделі доступу (звичайний користувач, адміністратор).

Для захисту облікових даних користувачів використовується алгоритм хешування паролів із застосуванням бібліотеки bcrypt. Це дозволяє уникнути зберігання паролів у відкритому вигляді та підвищує рівень безпеки системи.

Крім того, реалізовано механізми перевірки доступу до ресурсів на основі ролей користувачів, що дозволяє обмежити виконання окремих операцій.

Проєкт побудовано з використанням модульної структури, яка передбачає розділення на окремі компоненти:

– API (обробка HTTP-запитів);

– CLI (інтерфейс користувача);

– бізнес-логіка (робота із задачами та користувачами);

– модуль автентифікації;

– модуль доступу до бази даних.

Такий підхід забезпечує:

– зручність супроводу коду;

– можливість тестування окремих компонентів;

– масштабованість системи [7].

Для функціонування розроблюваної системи достатньо стандартного персонального комп’ютера з наступними характеристиками:

– операційна система: Windows, Linux або macOS;

– встановлене середовище виконання Go (версія 1.21 або вище);

– наявність дискового простору для зберігання бази даних.

Завдяки використанню SQLite система не потребує встановлення додаткових серверів баз даних, що значно спрощує її розгортання та подальше використання в різних середовищах. Такий підхід дозволяє уникнути складної конфігурації серверної інфраструктури та зменшує кількість зовнішніх залежностей програмного продукту. Крім того, це підвищує мобільність системи, оскільки вона може бути запущена практично на будь-якій платформі без додаткових налаштувань бази даних. У результаті спрощується процес інсталяції та знижується поріг входу для кінцевого користувача або розробника [8].

1.4 Постановка завдання на кваліфікаційну роботу бакалавра

На основі проведеного аналізу предметної області (п. 1.1), дослідження існуючих програмних рішень (п. 1.2) та обґрунтування вибору засобів проектування (п. 1.3) сформульовано основну мету та завдання кваліфікаційної роботи.

Метою даної кваліфікаційної роботи є розробка консольного застосунку для обліку задач на мові програмування Go, який забезпечує ефективне управління задачами, підтримку багатокористувацького режиму, можливість інтеграції через REST API та реалізацію механізмів автентифікації і розмежування доступу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1) провести аналіз предметної області та визначити основні проблеми існуючих підходів до управління задачами;

2) дослідити сучасні програмні рішення у сфері управління задачами та виконати їх порівняльний аналіз;

3) обґрунтувати вибір засобів проектування, включаючи мову програмування, систему управління базами даних та архітектурні підходи;

4) розробити архітектуру програмного застосунку, що включає серверну частину (REST API) та клієнтську частину (CLI);

5) реалізувати функціонал управління користувачами, включаючи реєстрацію, автентифікацію та авторизацію;

6) реалізувати функціонал управління задачами, включаючи створення, редагування, видалення та перегляд задач;

7) забезпечити підтримку багатокористувацького режиму та спільної роботи над задачами;

8) реалізувати механізми розмежування доступу на основі ролей користувачів;

9) спроектувати та реалізувати структуру бази даних для зберігання інформації про користувачів і задачі;

10) провести тестування розробленого програмного забезпечення та оцінити його працездатність.

У результаті виконання кваліфікаційної роботи має бути створено програмний продукт, який відповідає сучасним вимогам до систем управління задачами, є зручним у використанні, має гнучку архітектуру та може бути використаний як основа для подальшого розвитку і розширення функціональності.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

Розроблення програмної системи для управління задачами потребує обґрунтованого вибору архітектурних підходів, технологій реалізації та алгоритмів обробки даних. Вибір конкретних рішень здійснюється з урахуванням вимог до системи, визначених у попередньому розділі, а також на основі аналізу існуючих програмних продуктів.

У якості основного підходу до побудови системи обрано клієнт-серверну архітектуру, що передбачає розділення застосунку на серверну частину та клієнтський інтерфейс.

Серверна частина реалізована у вигляді REST API, яке відповідає за обробку запитів, виконання бізнес-логіки та взаємодію з базою даних. Клієнтська частина представлена консольним інтерфейсом (CLI), який взаємодіє з сервером через HTTP-протокол.

Такий підхід має наступні переваги:

– чітке розмежування відповідальності між компонентами системи;
– можливість незалежного розвитку клієнтської та серверної частин;
– спрощення інтеграції з іншими системами;
– можливість масштабування.

Альтернативою могла бути монолітна архітектура без виділеного API або використання графічного інтерфейсу користувача. Проте такі підходи обмежують гнучкість системи та ускладнюють її подальший розвиток.

Для організації взаємодії між клієнтом і сервером обрано архітектурний стиль REST (Representational State Transfer). Даний підхід передбачає використання стандартних HTTP-методів для виконання операцій над ресурсами.

Основними перевагами REST є:

– простота реалізації та розуміння;

– використання стандартних протоколів і форматів (HTTP, JSON);

– незалежність клієнта і сервера;

– підтримка масштабованості та розподілених систем.

Використання REST API дозволяє забезпечити універсальний доступ до функціоналу системи, що дає можливість у майбутньому реалізувати додаткові клієнтські застосунки (наприклад, веб або мобільний інтерфейс).

Для реалізації серверної частини застосунку обрано мову програмування Go.

Основними причинами такого вибору є:

– висока продуктивність завдяки компіляції у машинний код;

– ефективна підтримка конкурентності (горутини);

– наявність розвиненої стандартної бібліотеки;

– простота розгортання (один виконуваний файл);

– зручність роботи з мережевими протоколами.

Альтернативні мови програмування, такі як Python або Java, також можуть бути використані для реалізації подібних систем, проте вони мають певні недоліки. Python поступається за продуктивністю, а Java потребує більш складного налаштування середовища виконання.

Таким чином, мова Go є оптимальним вибором для реалізації серверного застосунку середнього рівня складності.

Для зберігання даних використано SQLite. Основними причинами вибору є:

– відсутність необхідності у встановленні окремого серверу бази даних;

– простота інтеграції у застосунок;

– достатня продуктивність для задач даного типу;

– підтримка стандартної мови SQL;

– компактність і зручність зберігання даних у вигляді одного файлу [9].

Альтернативами могли бути такі СКБД, як MySQL або PostgreSQL, які забезпечують кращу масштабованість та продуктивність у високонавантажених

системах. Проте для консольного застосунку з помірним навантаженням їх використання є надлишковим.

Для реалізації автентифікації користувачів обрано технологію JSON Web Token. JWT-токени дозволяють реалізувати безстанну (stateless) автентифікацію, при якій сервер не зберігає інформацію про сесію користувача. Уся необхідна інформація передається у вигляді токена в кожному запиті. Перевагами такого підходу є:

– зменшення навантаження на сервер;
– спрощення масштабування;
– можливість використання у розподілених системах;
– зручність інтеграції з клієнтськими застосунками.

Альтернативним підходом є сесійна автентифікація, яка потребує зберігання стану на сервері, що ускладнює масштабування.

У системі реалізовано базові алгоритми роботи з даними, що включають:

– створення, читання, оновлення та видалення записів (CRUD-операції);
– фільтрацію та вибірку задач користувача;
– обробку запитів із перевіркою прав доступу;
– управління зв’язками між користувачами та задачами.

Особливістю реалізації є механізм «розумного видалення» задач, який передбачає різну поведінку залежно від ролі користувача:

– власник задачі може повністю видалити її;
– учасник може лише вийти зі списку учасників;
– адміністратор має повний доступ до видалення.

Такий підхід дозволяє забезпечити гнучкість управління даними та відповідає сучасним вимогам до багатокористувацьких систем.

У якості клієнтського інтерфейсу обрано консольний застосунок (CLI), що дозволяє взаємодіяти із системою через командний рядок. Перевагами такого підходу є:

– висока швидкість роботи;
– можливість автоматизації;

– низькі вимоги до ресурсів;

– зручність для технічних користувачів.

Альтернативою є використання графічного інтерфейсу користувача, проте це потребує додаткових ресурсів на розробку та не є критично необхідним у рамках даного проєкту.

2.2 Функціонально-структурна схема роботи об’єкта проектування (блок-схеми алгоритмів, UML діаграми класів, діаграми компонентів)

Розроблювана система являє собою консольний застосунок для управління задачами, що побудований за клієнт-серверною архітектурою з використанням REST API. Основною метою системи є забезпечення ефективного управління задачами, підтримка багатокористувацького режиму та реалізація механізмів автентифікації й авторизації.

Основними цілями розроблюваної системи є:

– забезпечення централізованого зберігання задач;

– надання зручного інтерфейсу для роботи із задачами через CLI;

– реалізація багатокористувацького доступу;

– забезпечення безпеки даних із розмежуванням прав доступу.

Крім цього, система повинна підтримувати можливість інтеграції із зовнішніми сервісами та застосунками через REST API, що дозволяє розширювати функціональні можливості програмного продукту та забезпечує універсальність його використання.

До основних задач системи належать реєстрація та автентифікація користувачів, створення, редагування та видалення задач, перегляд списку задач, управління учасниками задач і контроль доступу до ресурсів залежно від ролі користувача.

Реалізація даного функціоналу дозволяє забезпечити ефективну організацію процесу управління задачами та підтримку багатокористувацького режиму роботи (рис. 2.1).

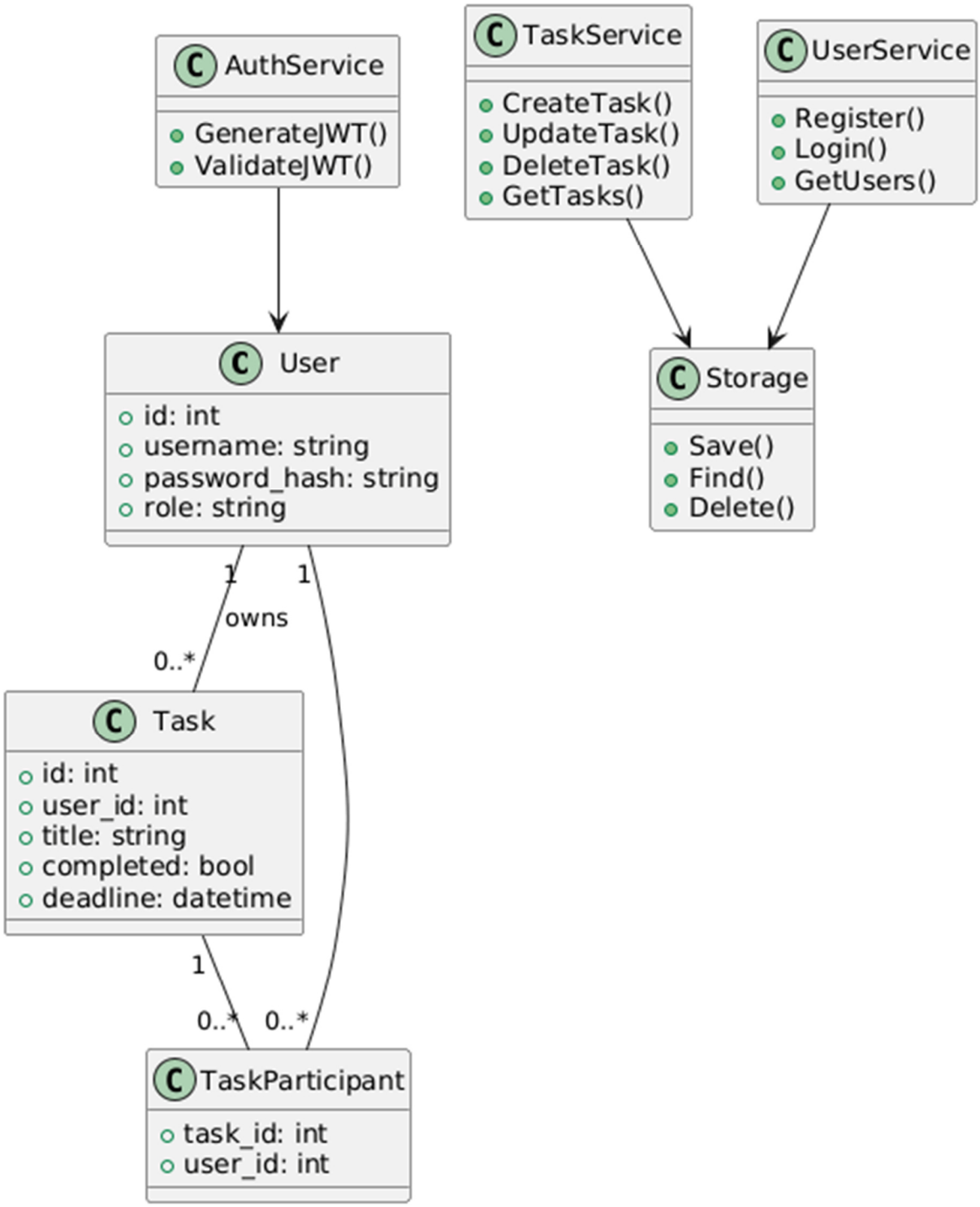

Рисунок 2.1 – UML діаграма класів

Система забезпечує введення нових даних шляхом створення задач, модифікацію існуючих записів, видалення задач із урахуванням прав доступу користувачів, а також пошук та отримання списку задач. Додатково реалізовано механізми контролю доступу до операцій, збереження даних у базі даних та обробку запитів через HTTP API. Такий підхід забезпечує надійність роботи

системи, гнучкість у використанні та можливість подальшого масштабування програмного забезпечення.

Між компонентами системи відбувається обмін інформацією (рис. 2.2):

- CLI → API: HTTP-запити;
- API → CLI: JSON-відповіді;
- API → DB: SQL-запити;
- DB → API: результати запитів.

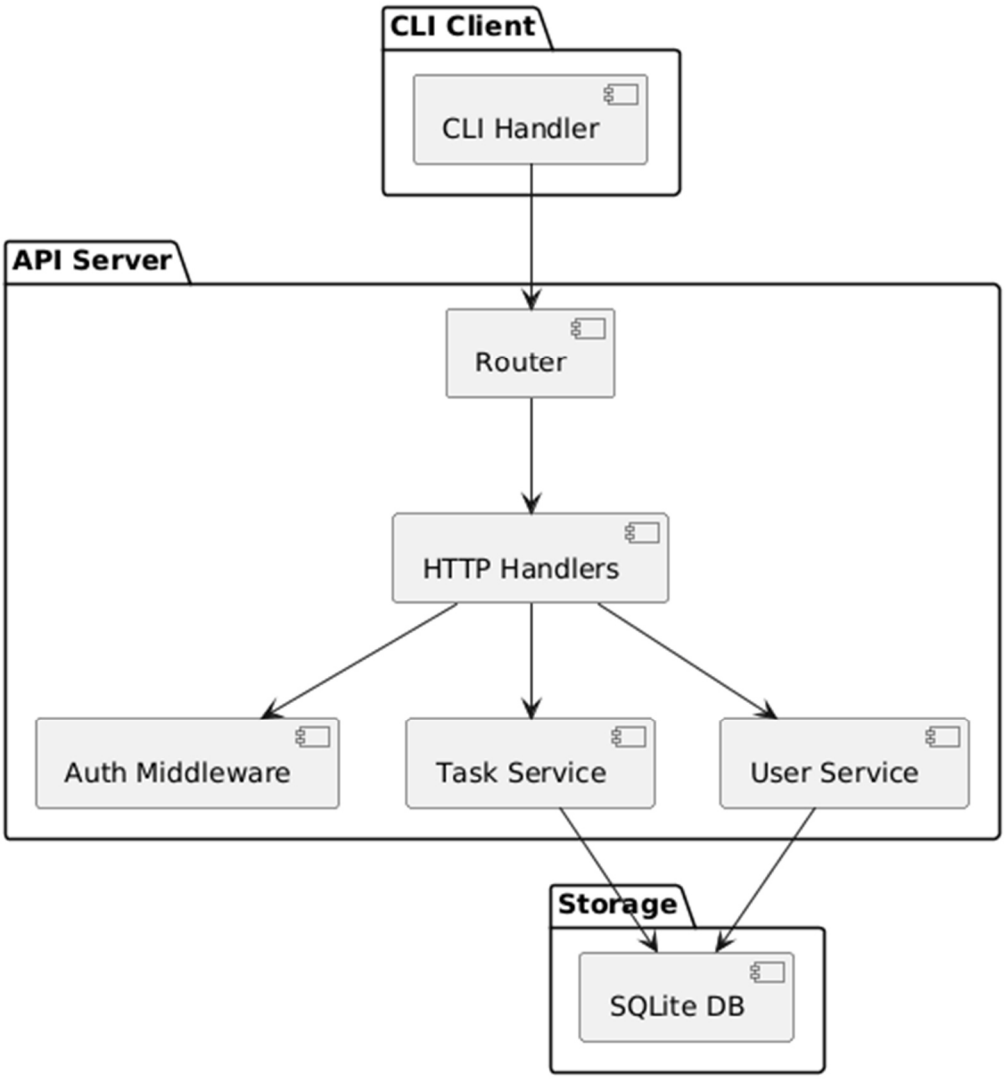

Рисунок 2.2 – UML діаграма компонентів

Основні потоки:

1) авторизація;
2) управління задачами;

3) управління користувачами;

4) розумне видалення задач (рис. 2.3).

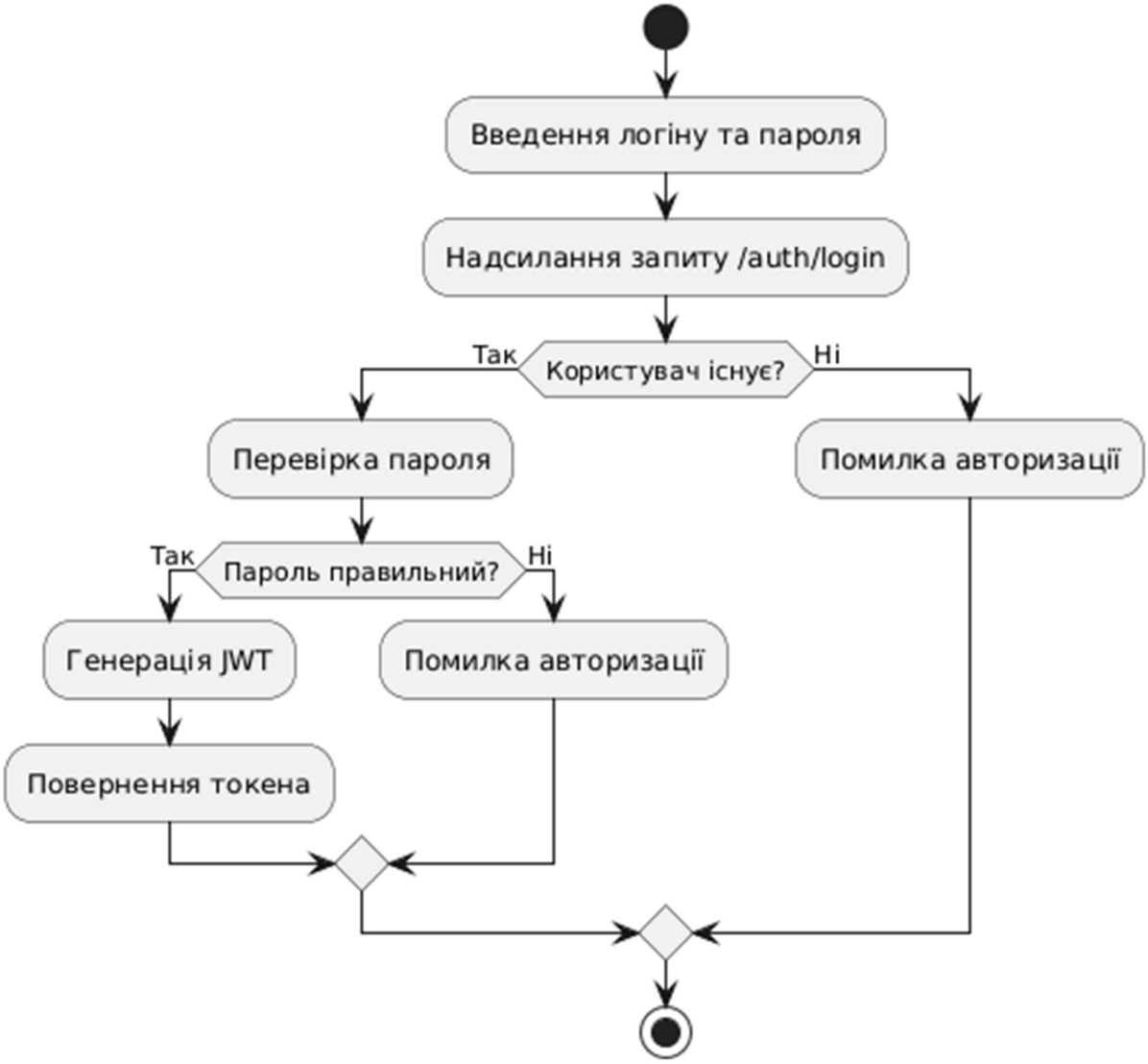

Рисунок 2.3 – Блок-схема: створення задачі

Основними потоками взаємодії в системі є процеси авторизації користувачів, управління задачами та управління користувачами. Під час авторизації здійснюється перевірка облікових даних користувача, визначення його ролі та надання відповідних прав доступу до функціональних можливостей системи.

Модуль управління задачами забезпечує створення, редагування, перегляд, призначення та контроль виконання задач, що дозволяє ефективно

організовувати робочі процеси та координувати діяльність користувачів. Управління користувачами охоплює операції зі створення облікових записів, оновлення персональних даних, зміни ролей, а також блокування або видалення користувачів за потреби.

Реалізована архітектура забезпечує чітке розмежування відповідальності між компонентами системи, завдяки чому кожен модуль виконує окремий набір функцій і взаємодіє з іншими компонентами через визначені інтерфейси. Такий підхід сприяє підвищенню надійності програмного забезпечення, спрощує його супровід і тестування, а також полегшує внесення змін та впровадження нових можливостей без суттєвого впливу на вже існуючий функціонал. Крім того, архітектура створює передумови для подальшого масштабування застосунку, зокрема шляхом розширення бізнес-логіки, інтеграції із зовнішніми сервісами та збільшення кількості одночасних користувачів без втрати продуктивності системи.

2.3 Створення моделі бази даних

Однією з ключових складових розроблюваної системи є база даних, яка забезпечує зберігання, обробку та доступ до інформації про користувачів і задачі. Для ефективної роботи системи було розроблено інфологічну, логічну та фізичну моделі бази даних.

Інфологічна модель описує предметну область на концептуальному рівні та визначає основні сутності системи та зв’язки між ними.

У межах розроблюваної системи виділено такі основні сутності:

– користувач (User) – представляє зареєстрованого користувача системи;

– задача (Task) – описує окреме завдання;

– учасник задачі (TaskParticipant) – зв’язуюча сутність, яка реалізує зв’язок між користувачами та задачами.

Основні зв’язки:

– один користувач може мати багато задач (1:N);

– одна задача може мати багато учасників (M:N);

– користувач може бути учасником багатьох задач (M:N).

Логічна модель бази даних програмного продукту реалізована у вигляді реляційної структури та складається з трьох основних таблиць: users, tasks і task_participants. Використання реляційного підходу дозволяє забезпечити цілісність даних, спростити організацію зв’язків між сутностями системи та підвищити ефективність роботи із даними в процесі виконання операцій створення, редагування, пошуку та видалення інформації. Побудована структура бази даних орієнтована на підтримку багатокористувацького режиму роботи та забезпечує можливість організації колаборативної взаємодії між користувачами системи (рис. 2.4) [10].

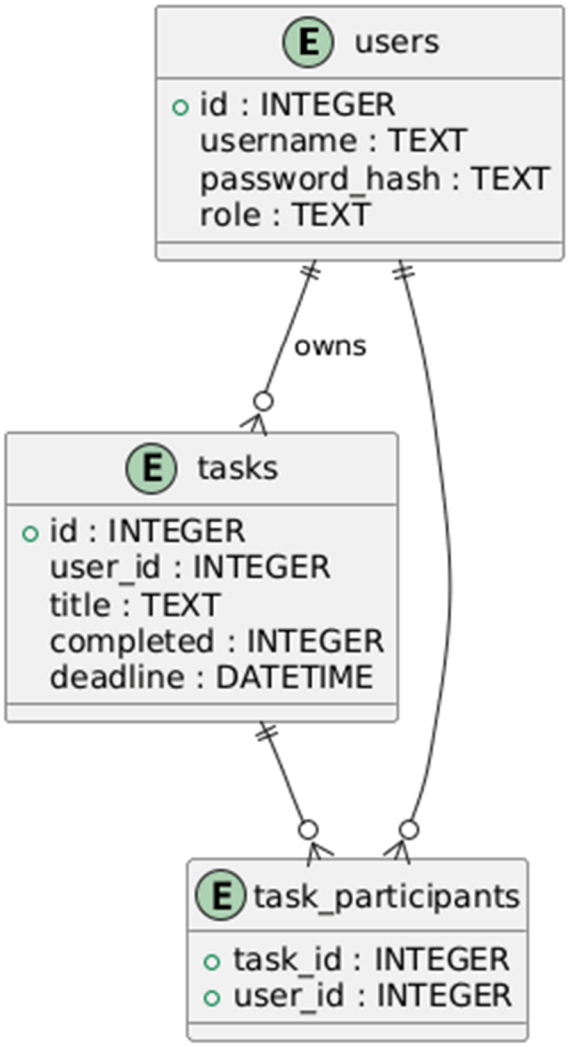

Рисунок 2.4 – ERD (діаграма «сутність-зв’язок»)

Таблиця users містить основну інформацію про зареєстрованих користувачів системи. До її складу входять унікальний ідентифікатор користувача, ім’я користувача, хеш пароля та роль користувача в системі (user або admin). Унікальний ідентифікатор використовується для однозначної ідентифікації кожного користувача в межах системи та встановлення зв’язків з іншими таблицями бази даних. Ім’я користувача застосовується під час автентифікації та взаємодії із системою, а збереження паролів у вигляді хешів забезпечує підвищення рівня безпеки та захист конфіденційних даних користувачів. Поле ролі визначає рівень доступу користувача до функціональних можливостей системи та використовується для реалізації механізму рольової моделі доступу.

Таблиця tasks призначена для збереження інформації про задачі, створені користувачами системи. Вона включає ідентифікатор задачі, ідентифікатор власника задачі, назву задачі, статус виконання та термін виконання. Кожна задача має власника, який відповідає за її створення та може керувати доступом інших користувачів до неї. Поле назви задачі містить короткий опис суті завдання, а статус виконання використовується для відображення поточного стану задачі, наприклад виконана або невиконана. Наявність терміну виконання дозволяє організувати контроль дедлайнів та підвищує ефективність планування роботи користувачів. Структура таблиці tasks забезпечує можливість подальшого розширення функціоналу системи, зокрема додавання пріоритетів, категорій або системи нагадувань.

Для реалізації можливості спільної роботи над задачами використовується таблиця task_participants, яка реалізує зв’язок типу «багато-до-багатьох» між користувачами та задачами. Вона містить ідентифікатор задачі та ідентифікатор користувача, а первинний ключ реалізовано у вигляді складеного ключа (task_id, user_id). Така структура забезпечує гнучкість у керуванні доступом користувачів до задач та підтримує можливість додавання кількох учасників до однієї задачі. Фізична реалізація бази даних здійснюється з використанням SQLite (ліст. 2.1).

Лістинг 2.1 – Скрипт

```
CREATE TABLE users (
id INTEGER PRIMARY KEY AUTOINCREMENT,
username TEXT NOT NULL UNIQUE,
password_hash TEXT NOT NULL,
role TEXT NOT NULL DEFAULT ‘user’
);

CREATE TABLE tasks (
id INTEGER PRIMARY KEY AUTOINCREMENT,
user_id INTEGER NOT NULL,
title TEXT NOT NULL,
completed INTEGER NOT NULL DEFAULT 0,
deadline DATETIME
);

CREATE TABLE task_participants (
task_id INTEGER NOT NULL,
user_id INTEGER NOT NULL,
PRIMARY KEY (task_id, user_id)
);
```

кінець лістингу 2.1

Сучасні системи управління базами даних можна класифікувати за типом моделі даних на реляційні СКБД (MySQL, PostgreSQL, SQLite), NoSQL-рішення (MongoDB, Cassandra), а також ієрархічні та мережеві системи, які сьогодні використовуються значно рідше. У даній роботі використано реляційну модель даних, оскільки вона є найбільш ефективною для роботи зі структурованою інформацією та забезпечує зручне використання SQL-запитів і зв’язків між таблицями [11].

Для реалізації проєкту було обрано SQLite, що обумовлено простотою розгортання без необхідності використання окремого серверного програмного забезпечення, мінімальними вимогами до системних ресурсів та можливістю зберігання всієї бази даних у одному файлі. Крім цього, SQLite забезпечує достатню продуктивність для задач даного типу та повноцінну підтримку SQL. Альтернативні рішення, такі як MySQL або PostgreSQL, є більш складними у налаштуванні та адмініструванні, а використання MongoDB є недоцільним через реляційну структуру даних системи управління задачами.

Доступ до бази даних реалізовано через стандартний пакет мови Go database/sql із використанням драйвера SQLite. Реалізований механізм взаємодії з базою даних забезпечує виконання SQL-запитів, отримання результатів, обробку помилок та підтримку роботи з транзакціями. Такий підхід забезпечує універсальність програмного рішення, незалежність від конкретної системи управління базами даних та зручність подальшого тестування й масштабування застосунку.

Також використання вбудованої бази даних позитивно впливає на швидкість старту системи, оскільки відсутня необхідність ініціалізації окремого серверного процесу. Це робить застосунок більш легким у розгортанні та зручним для тестування й локальної розробки.

РОЗДІЛ 3
РОЗРОБКА КОНСОЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ЗАДАЧ НА МОВІ GO

3.1 Практична реалізація об’єкта проектування. Побудова блок-схем модулів програми

Програмний продукт реалізовано мовою програмування Go з дотриманням принципів чистої архітектури. Система складається з трьох основних шарів: шару зберігання даних (storage), шару бізнес-логіки (task, user) та шару представлення (api, cli). Така структура забезпечує незалежність компонентів та спрощує тестування і подальший супровід.

Обрана архітектура також спрощує процес модифікації та розширення функціональних можливостей програмного продукту. Наприклад, зміна способу зберігання даних або додавання нового інтерфейсу взаємодії з користувачем не потребує повної переробки бізнес-логіки системи. Це забезпечує гнучкість розробки та дозволяє адаптувати систему до нових вимог у майбутньому. Крім того, використання мови програмування Go забезпечує високу продуктивність, простоту розгортання та кросплатформеність програмного продукту [12].

У процесі розробки було використано модульний підхід до організації структури проєкту. Усі компоненти системи згруповано відповідно до їх функціонального призначення, що полегшує навігацію по коду та підвищує зрозумілість структури програмного продукту для інших розробників. Такий підхід відповідає сучасним практикам розробки серверного програмного забезпечення та сприяє покращенню якості програмного коду.

Основні модулі системи (рис. 3.1):

– cmd/api – точка входу HTTP API-сервера;
– cmd/cli – точка входу консольного застосунку;
– internal/api – HTTP-хендлери та маршрутизатор;
– internal/auth – генерація та перевірка JWT-токенів, мідлвари;
– internal/task – бізнес-логіка задач;

– internal/user – бізнес-логіка користувачів;
– internal/storage – взаємодія з базою даних SQLite.

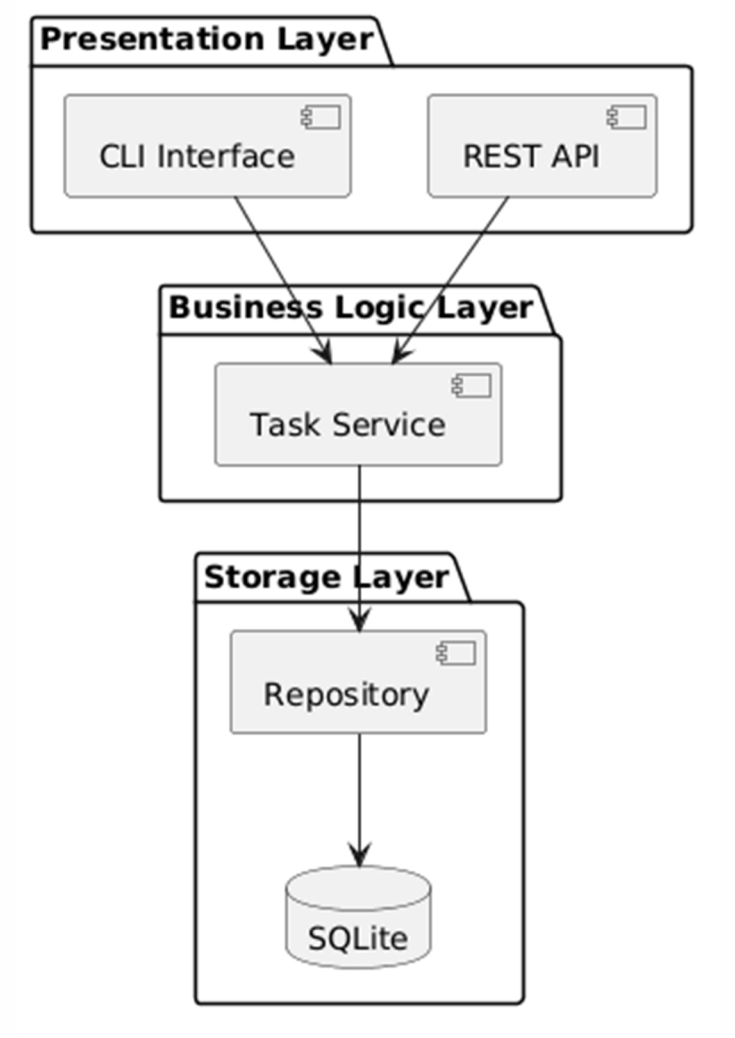

Рисунок 3.1 – Діаграма архітектури застосунку: три шари

При запуску сервера виконується ініціалізація бази даних та створення облікового запису адміністратора за умови його відсутності. Функція storage.Init() відкриває з'єднання з SQLite-базою та виконує DDL-запити для створення таблиць users, tasks і task_participants.

Після ініціалізації користувач може пройти реєстрацію або автентифікацію (рис. 3.2). При реєстрації пароль хешується алгоритмом bcrypt з коефіцієнтом складності 10, що унеможливлює відновлення оригінального пароля з хешу. При вході система генерує JWT-токен, який містить ідентифікатор користувача, його ім'я та роль, та підписується алгоритмом HMAC-SHA256.

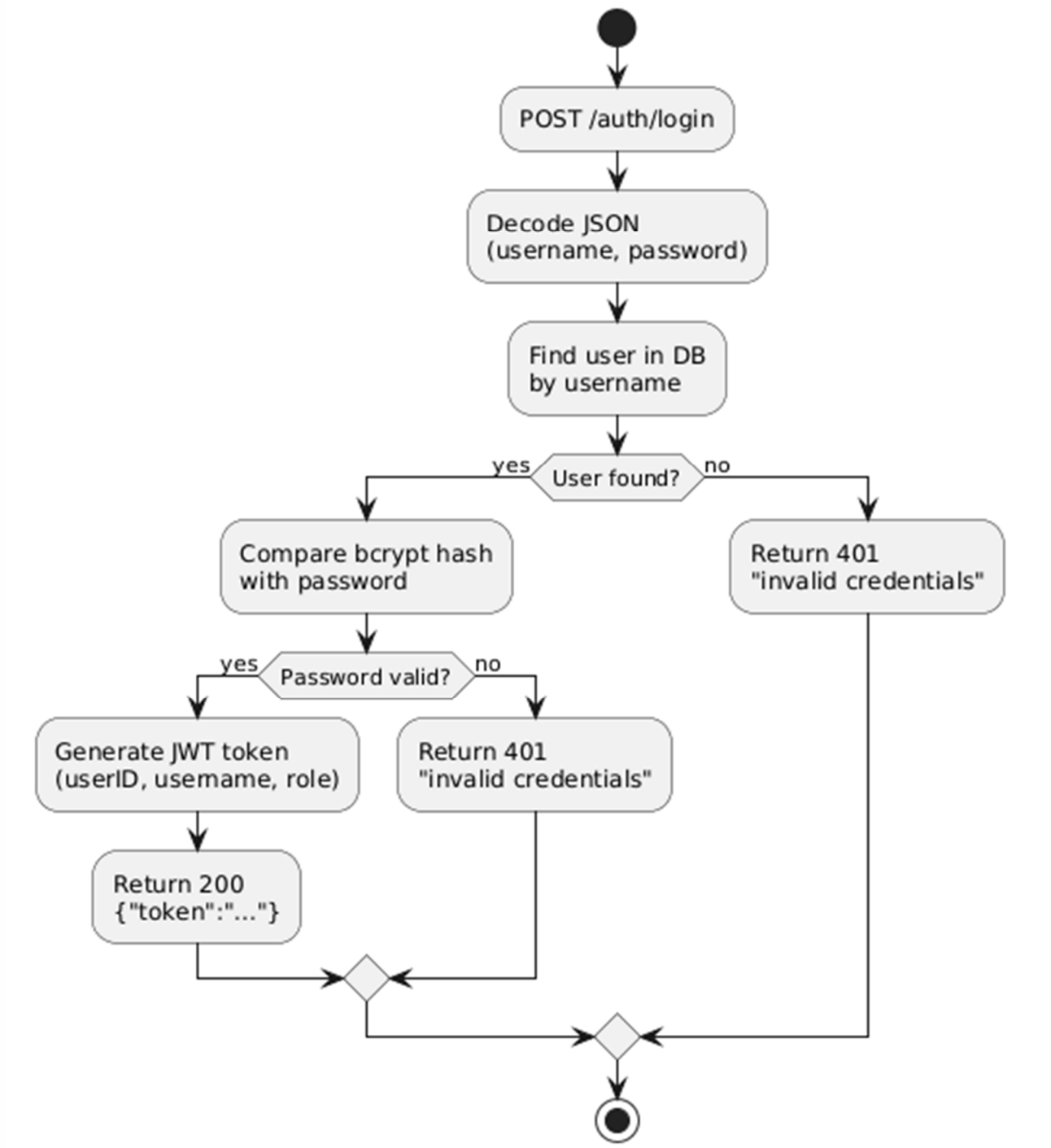

Рисунок 3.2 – Блок-схема процесу автентифікації у графічному вигляді

Модуль задач реалізує повний цикл CRUD-операцій (створення, перегляд, редагування та видалення задач) і містить додаткову бізнес-логіку для управління доступом користувачів та контролю виконання завдань (рис. 3.3). Він забезпечує можливість призначення задач, зміни їх параметрів і статусів, а також

підтримує механізми спільного доступу для командної роботи. Під час виконання операцій система перевіряє права доступу користувачів, що сприяє захисту даних та підтриманню їх цілісності.

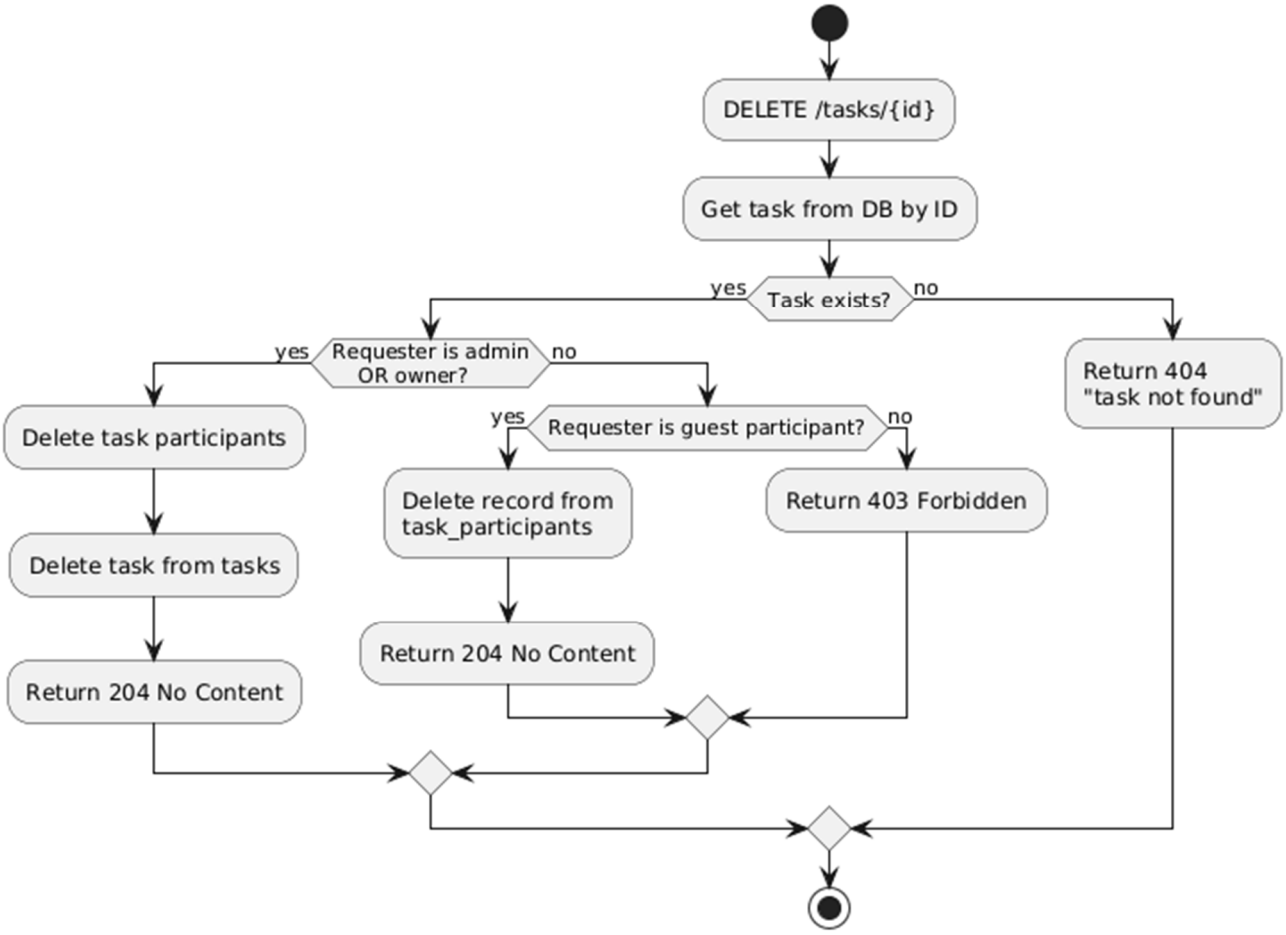

Рисунок 3.3 – Блок-схема «розумного видалення» задачі у графічному вигляді

Кожна задача має власника *(*owner_id*),* який є її основним автором та відповідає за управління нею. Крім власника, до задачі можуть бути додані інші користувачі як учасники, інформація про яких зберігається у таблиці task_participants. Така модель реалізує зв’язок «багато до багатьох» між користувачами та задачами, оскільки один користувач може брати участь у кількох задачах, а одна задача може бути доступною для кількох користувачів одночасно.

Використання окремої таблиці для зберігання учасників забезпечує гнучкість під час налаштування доступу та спрощує управління спільною

роботою. Завдяки цьому користувачі можуть переглядати та редагувати задачі відповідно до наданих їм прав. Такий підхід сприяє ефективній взаємодії між членами команди, підвищує прозорість виконання завдань і забезпечує зручне керування доступом до даних у системі (рис. 3.4).

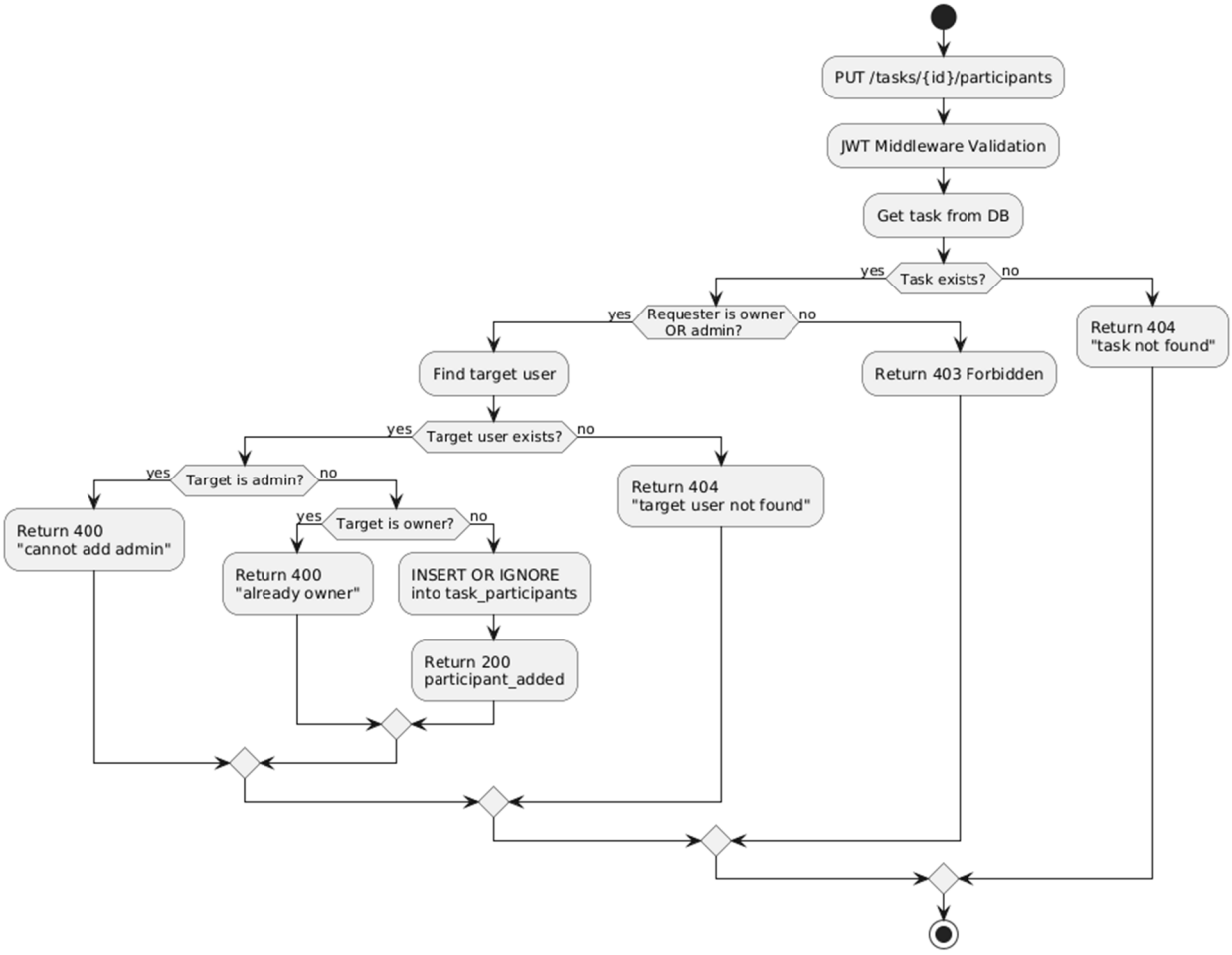

Рисунок 3.4 – Блок-схема додавання учасника до задачі у графічному вигляді

Адміністратор системи автоматично створюється при першому запуску. Модуль адміністрування надає розширені можливості: перегляд усіх користувачів і задач, зміну паролів користувачів, примусове видалення задач та виключення учасників.

Адміністратор має розширені права доступу, зокрема може переглядати всіх користувачів і задачі, змінювати паролі користувачів, а також примусово видаляти задачі та керувати їх учасниками. Доступ до адмін-ендпоінтів

захищений мідлваром AdminMiddleware, який перевіряє не лише наявність валідного токена, а й те, що поле role у Claims дорівнює «admin» (рис. 3.5) [13].

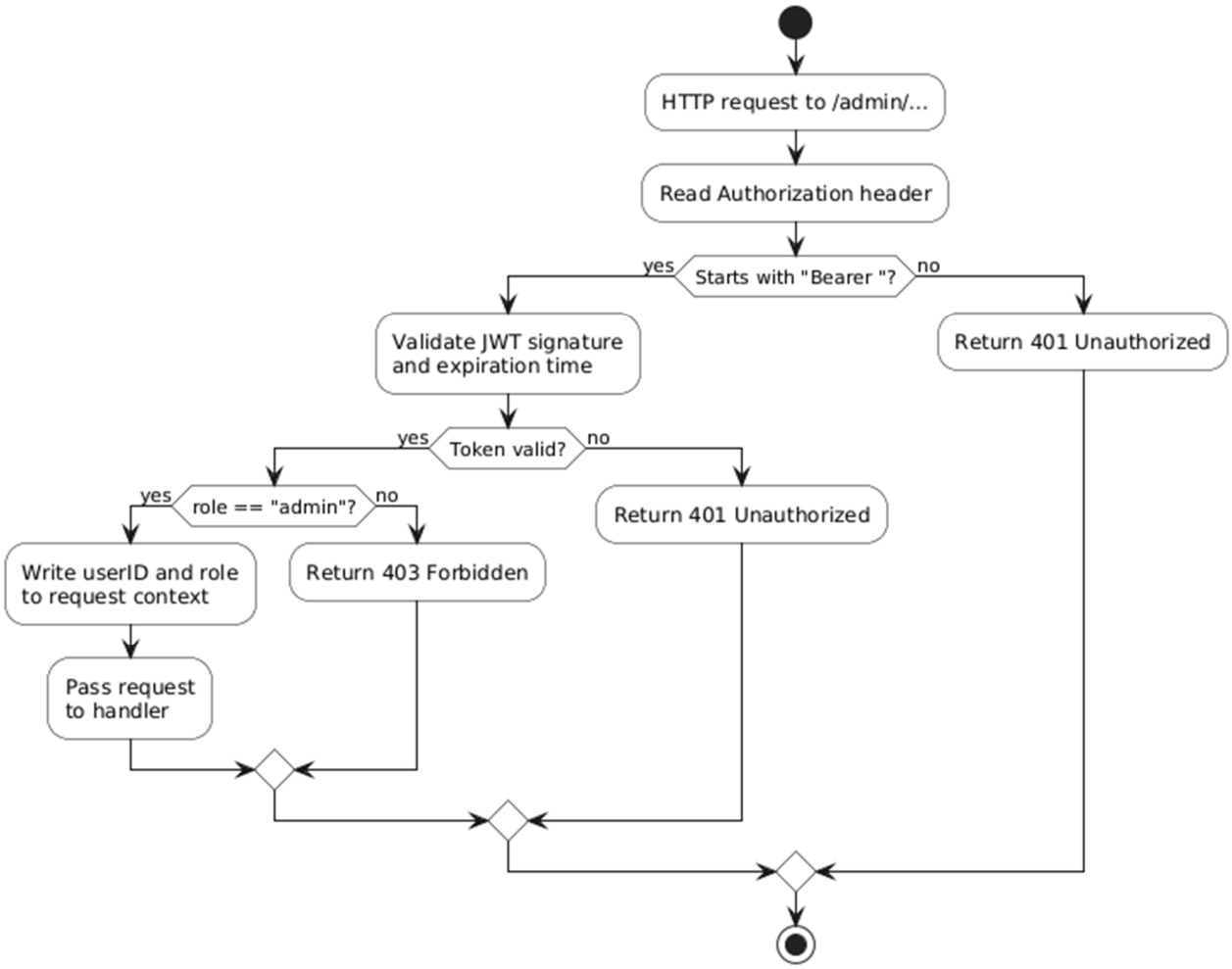

Рисунок 3.5 – Блок-схема мідлвару авторизації у графічному вигляді

База даних складається з трьох таблиць, що реалізують зв’язки один-до-багатьох та багато-до-багатьох:

– users – зберігає облікові записи користувачів із хешованими паролями та ролями;

– tasks – зберігає задачі, де user_id вказує на власника;

– task_participants – проміжна таблиця для зв’язку між задачами та гостьовими учасниками.

Така структура забезпечує гнучку організацію даних і дозволяє ефективно реалізувати модель спільного доступу до задач. Використання проміжної таблиці

також спрощує масштабування системи та подальше розширення функціоналу без необхідності зміни основної логіки зберігання даних (рис. 3.6).

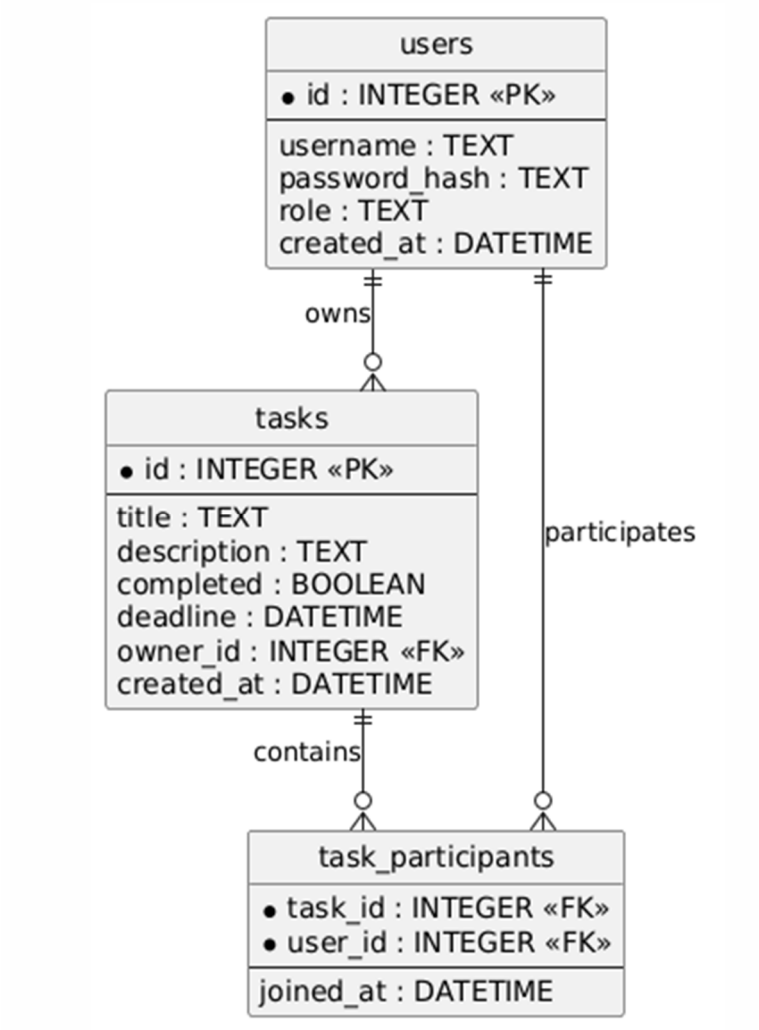

Рисунок 3.6 – ER-діаграма бази даних

Запропонована структура бази даних забезпечує надійне та впорядковане зберігання інформації про користувачів, задачі та зв’язки між ними, що є важливим для коректної роботи системи. Використання проміжної таблиці для реалізації зв’язку «багато до багатьох» дозволяє організувати спільний доступ до задач, спрощує керування правами учасників та створює основу для подальшого розширення функціональних можливостей застосунку без суттєвих змін у структурі даних.

3.2 Тестування та налагодження системи

Тестування системи проводилося на рівні функціонального та інтеграційного тестування. Функціональне тестування перевіряло коректність роботи кожного ендпоінту окремо – правильність HTTP-статусів, структуру JSON-відповідей та обробку некоректних вхідних даних. Інтеграційне тестування перевіряло взаємодію між модулями у типових сценаріях використання.

Для ручного тестування HTTP API використовувався інструмент Postman, що дозволяє надсилати довільні HTTP-запити, формувати та зберігати колекції запитів, а також аналізувати відповіді сервера. Використання цього інструмента значно спростило перевірку коректності роботи API та взаємодії між клієнтською і серверною частинами застосунку.

За допомогою Postman було виконано тестування всіх основних функцій системи, включаючи реєстрацію користувачів, автентифікацію та авторизацію, отримання і використання JWT-токенів для доступу до захищених ресурсів. Також перевірялися операції створення, отримання, редагування та видалення задач, робота механізмів спільного доступу до задач і коректність обробки помилок. Окрему увагу було приділено тестуванню адміністративних ендпоінтів, зокрема управлінню користувачами та перевірці обмежень доступу відповідно до ролей користувачів.

Особлива увага приділялася перевірці механізмів авторизації, зокрема доступу до захищених ресурсів із валідними та невалідними токенами [14].

Тестування модуля автентифікації (ліст. 3.1). Тест 1: Успішна реєстрація, мета якого перевірити коректність створення облікового запису користувача.

Лістинг 3.1 – REST запит на реєстрацію

```
POST /auth/register
Content-Type: application/json
{ "username": "testuser", "password": "pass1234" }
```

кінець лістингу 3.1

Очікуваний результат: статус 201 Created, тіло містить id, username, role, скриншот Postman (рис. 3.7).

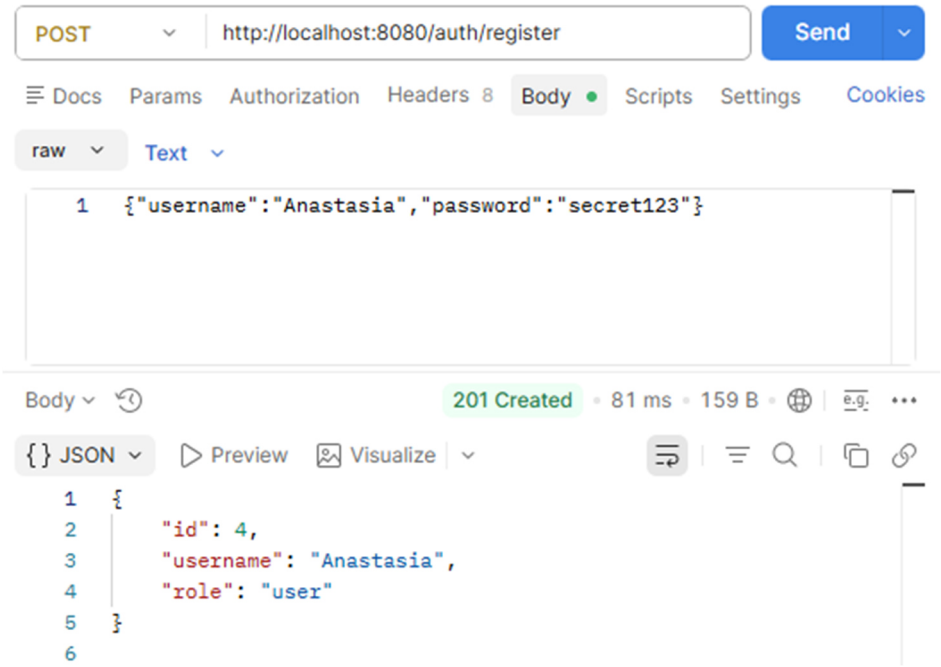

Рисунок 3.7 – Запит POST /auth/register та відповідь 201 з даними користувача

Тест 2: Реєстрація з існуючим іменем. Повторний запит із тим самим username. Очікуваний результат: статус 400 Bad Request, повідомлення «username already taken» (рис. 3.8).

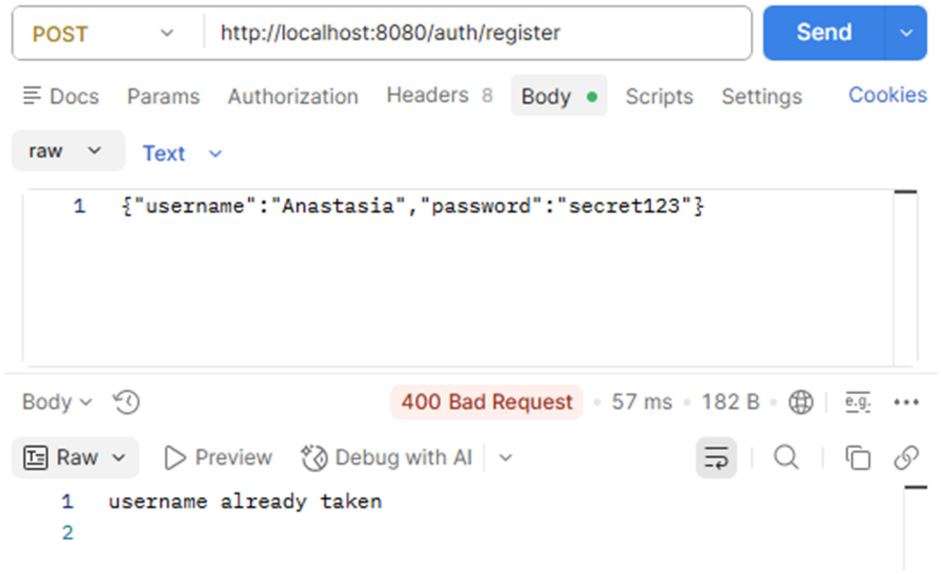

Рисунок 3.8 – Відповідь 400 при спробі зареєструватись з існуючим іменем

Даний тестовий сценарій призначений для перевірки коректності роботи механізму контролю унікальності імен користувачів у системі. Під час виконання тесту здійснюється повторна спроба реєстрації користувача з уже існуючим ім'ям, яке раніше було збережене в базі даних. Система повинна коректно обробити таку ситуацію та заборонити створення дубльованого облікового запису. Повернення HTTP-статусу 400 Bad Request свідчить про правильне виявлення помилки на стороні сервера, а текстове повідомлення дозволяє користувачу зрозуміти причину відмови у реєстрації. Успішне проходження цього тесту підтверджує цілісність даних та правильність перевірки унікальності користувачів у базі даних [15].

Тест 3: Успішний логін. Запит із коректними обліковими даними. Очікуваний результат: статус 200 OK, тіло {"token": "eyJ..."} (рис. 3.9).

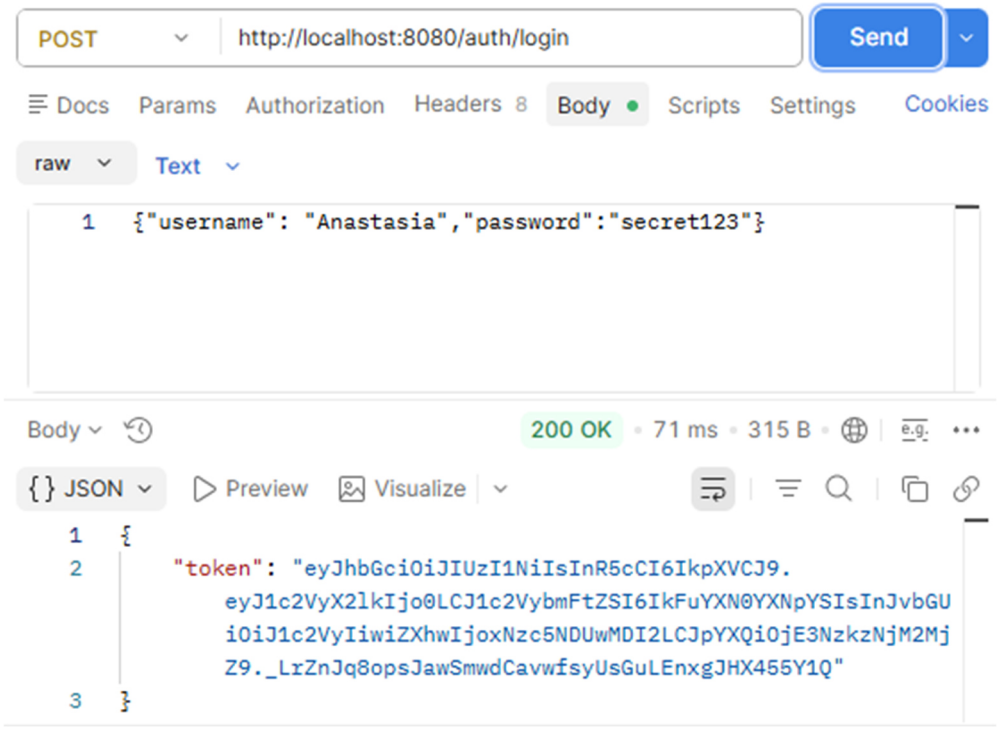

Рисунок 3.9 – Скріншот Postman: відповідь на POST /auth/login з JWT-токеном

Успішне виконання цього тестового сценарію підтверджує коректну роботу механізму автентифікації користувачів у системі. Після введення правильних облікових даних сервер формує та повертає JWT-токен, який використовується

для подальшої авторизації запитів. Це забезпечує захищений доступ до API та дозволяє ідентифікувати користувача без повторної передачі логіна і пароля [16].

Додатково тест підтверджує правильність інтеграції компонента автентифікації з бізнес-логікою системи та генерацією токенів доступу.

Тест 4: Запит без токена до захищеного ендпоінту. Запит GET /tasks без заголовка Authorization. Очікуваний результат: статус 401 Unauthorized (рис. 3.10).

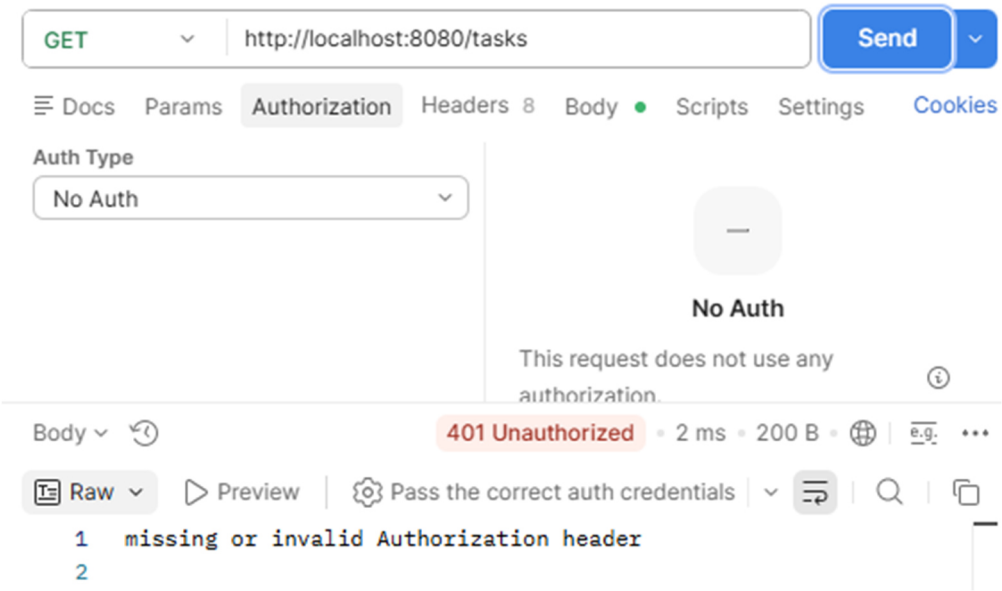

Рисунок 3.10 – Скріншот Postman: відповідь 401 при відсутньому токені

Наступним етапом є тестування модуля задач. Тест 5: Створення задачі, мета якого перевірити коректність створення нової задачі в системі та її збереження в базі даних. Передумовами є успішно авторизований у системі користувач та доступний для використанням модуль управління задачами (ліст. 3.2).

Лістинг 3.2 – REST запит на створення задачі

```
POST /tasks
Authorization: Bearer <token>
{ "title": "Написати диплом", "deadline": "2026-06-01T23:59:00Z" }
```

кінець лістингу 3.2

Очікуваний результат: 201 Created, задача з полями id, owner_id, title, completed: false, deadline.

Даний результат свідчить про успішне створення нового запису задачі в системі та коректну роботу відповідного API-ендпоінта. HTTP-статус 201 Created використовується відповідно до стандартів REST та підтверджує факт успішного створення ресурсу на сервері. У відповіді сервер повертає основні параметри створеної задачі, включаючи унікальний ідентифікатор задачі, ідентифікатор власника, назву задачі, статус виконання та встановлений термін виконання.

Значення поля completed автоматично встановлюється у стан false, що означає невиконану задачу на момент створення. Успішне проходження цього тесту підтверджує правильність роботи механізму створення задач та коректність взаємодії між API, бізнес-логікою та базою даних (рис. 3.11).

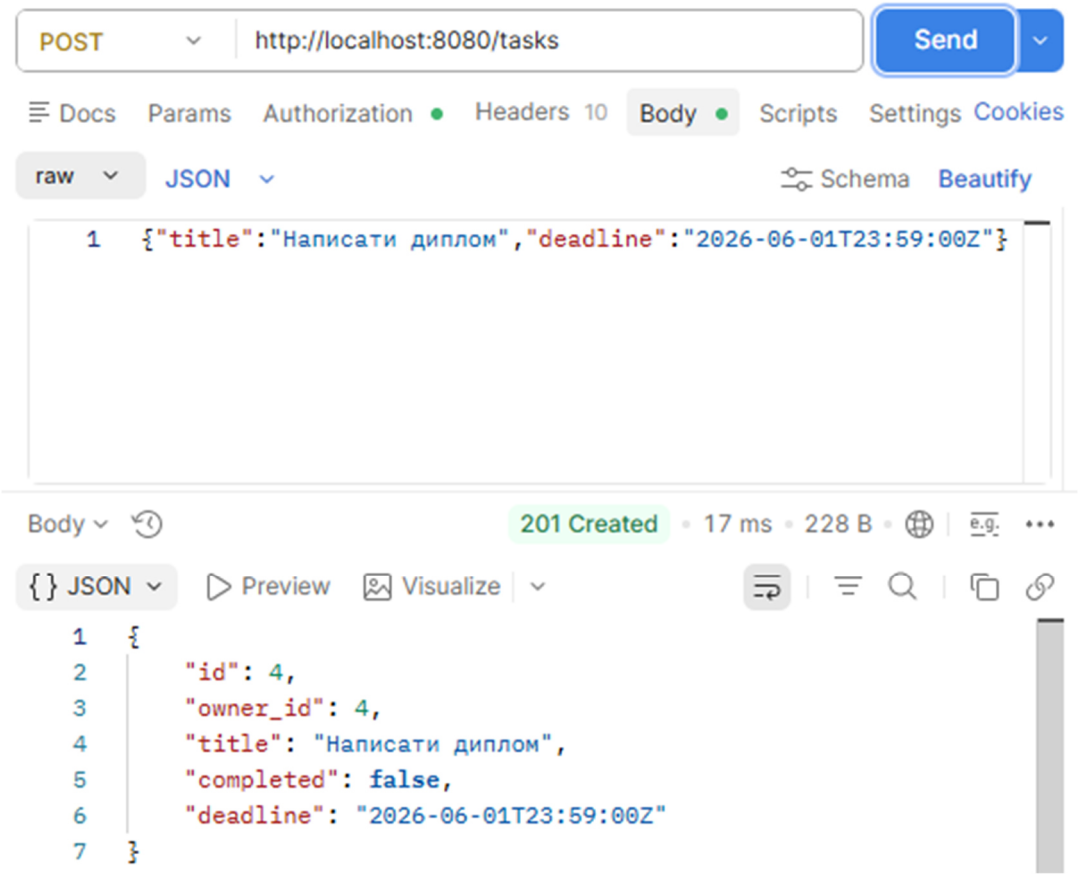

Рисунок 3.11 – Скріншот Postman: створення задачі та відповідь з повними даними

Тест 6: Отримання списку задач (ліст. 3.3).

Лістинг 3.3 – REST запит на отрисвння списку задач

```
GET /tasks
Authorization: Bearer <token>
```

кінець лістингу 3.3

Очікуваний результат: масив задач, де поточний користувач є власником або учасником (рис. 3.12).

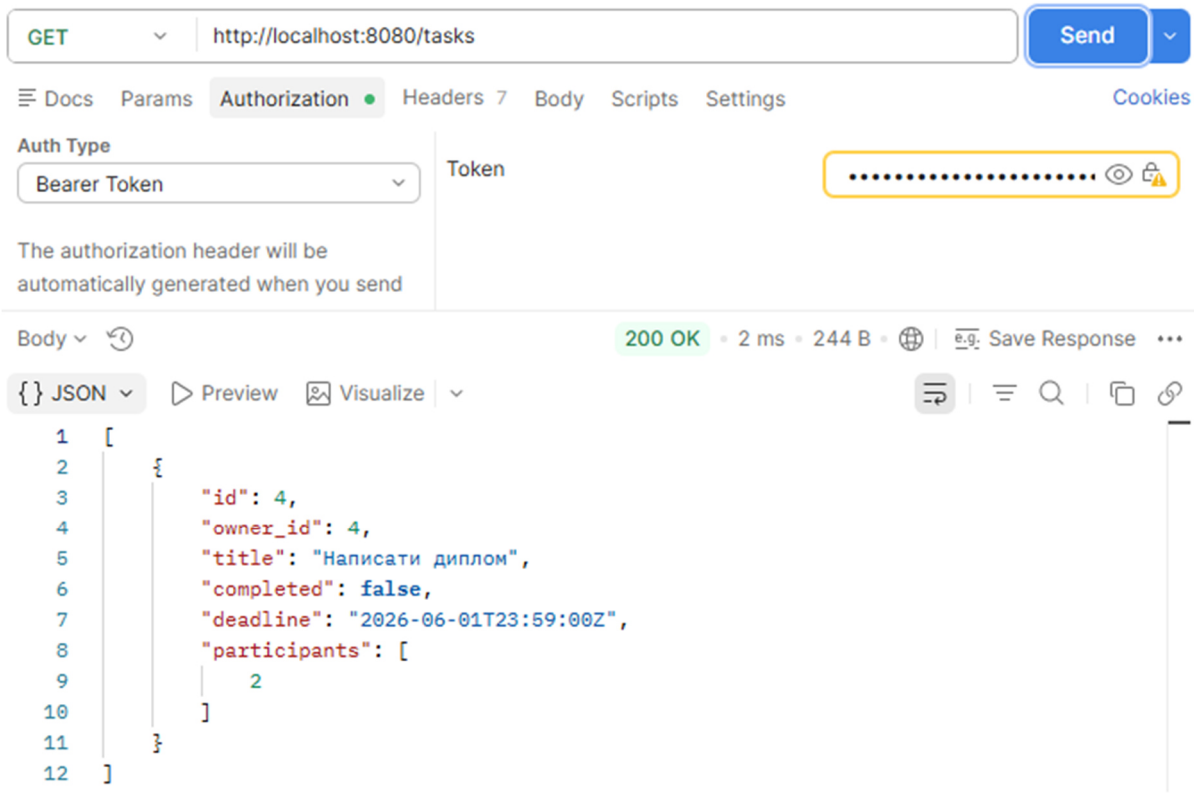

Рисунок 3.12 – Скріншот Postman: отримання задач

Тест 7: Додавання учасника власником задачі (ліст. 3.4).

Лістинг 3.4 – REST запит на додавання учасника до задачі

```
PUT /tasks/1/participants
Authorization: Bearer <owner_token>
{ "user_id": 2 }
```

кінець лістингу 3.4

Очікуваний результат: 200 OK з підтвердженням. Після цього GET /tasks від імені user_id=2 повертає цю задачу (рис. 3.13-3.14).

Даний тест підтверджує коректність реалізації механізму спільного доступу до задач. Після успішного додавання учасника система оновлює зв’язки в базі даних, що дозволяє іншому користувачу отримати доступ до задачі через відповідний API-запит. Успішне проходження тесту також підтверджує правильність роботи перевірки прав доступу та механізму взаємодії між користувачами системи.

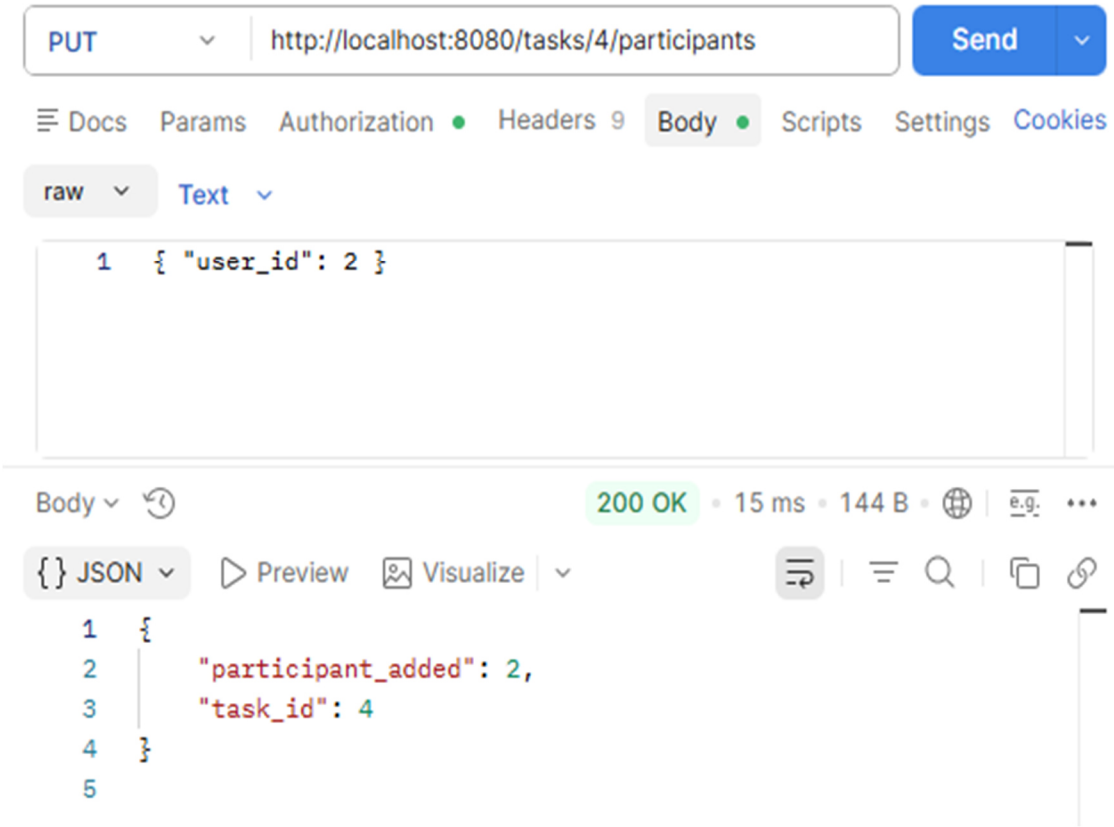

Рисунок 3.13 – Скріншот Postman: додавання учасника

Крім цього, тест демонструє узгодженість роботи бізнес-логіки та шару зберігання даних. Отримання задачі іншим користувачем підтверджує коректність реалізації багатокористувацького режиму роботи системи. Також це свідчить про правильну організацію перевірки прав доступу, що гарантує безпечну взаємодію користувачів із загальними ресурсами. У результаті система

забезпечує стабільну роботу в умовах одночасного доступу кількох користувачів без втрати цілісності даних (рис. 3.14).

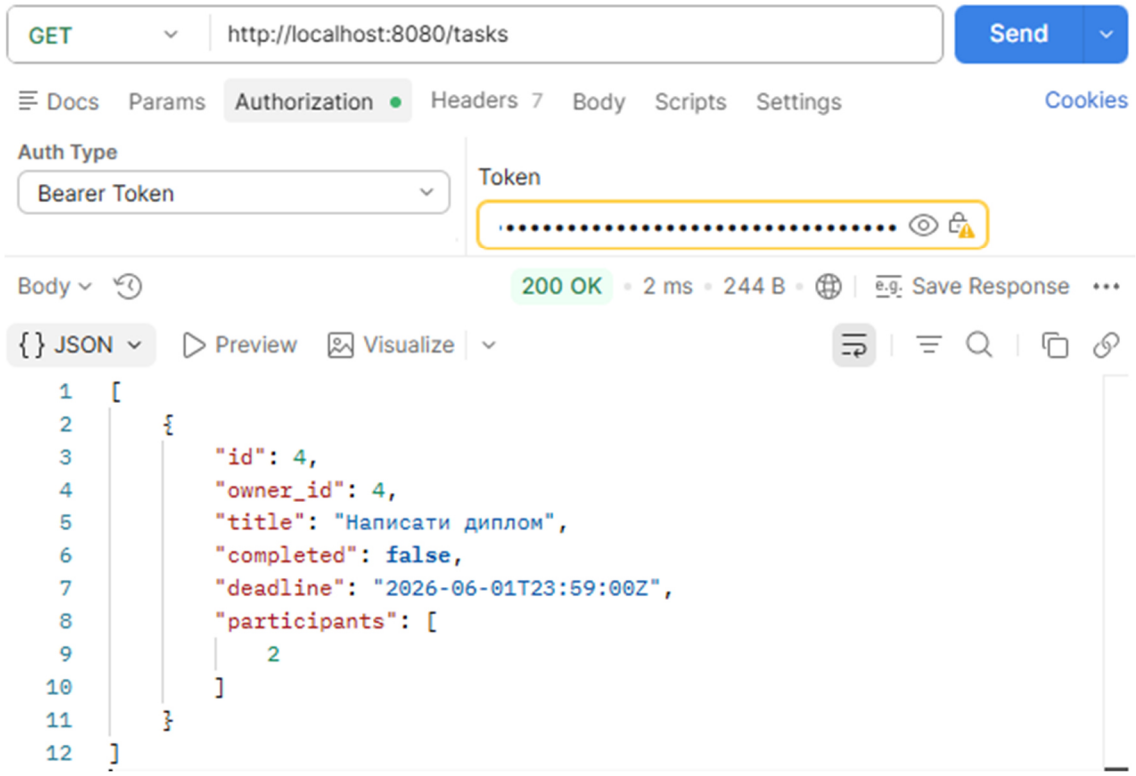

Рисунок 3.14 – Скріншот Postman: перевірка видимості задачі від імені учасника

Тест 8: Спроба додати учасника стороннім користувачем. Запит від користувача, який не є власником задачі. Очікуваний результат: 403 Forbidden, повідомлення «forbidden: only task owner or admin can add participants».

Тест 9: Видалення задачі учасником. Запит DELETE /tasks/1 від імені гостьового учасника. Очікуваний результат: 204 No Content. Задача залишається, але учасник більше її не бачить.

Тест 10: Видалення задачі власником. Запит DELETE /tasks/1 від імені власника. Очікуваний результат: 204 No Content. Задача повністю видалена з бази.

Тестування модуля адміністрування. Тест 11: Доступ до /admin/ зі звичайним токеном. Запит GET /admin/users із токеном звичайного користувача. Очікуваний результат: 403 Forbidden.

Тест 12: Зміна паролю користувача адміном (ліст. 3.5)

Лістинг 3.5 – REST запит зміна адміном паролю

```
PUT /admin/users/2/password
Authorization: Bearer <admin_token>
{ "password": "newpass123" }
```

кінець лістингу 3.5

Очікуваний результат: 200 OK, {"message": "password updated"}. Після цього логін зі старим паролем повертає 401 (рис. 3.15).

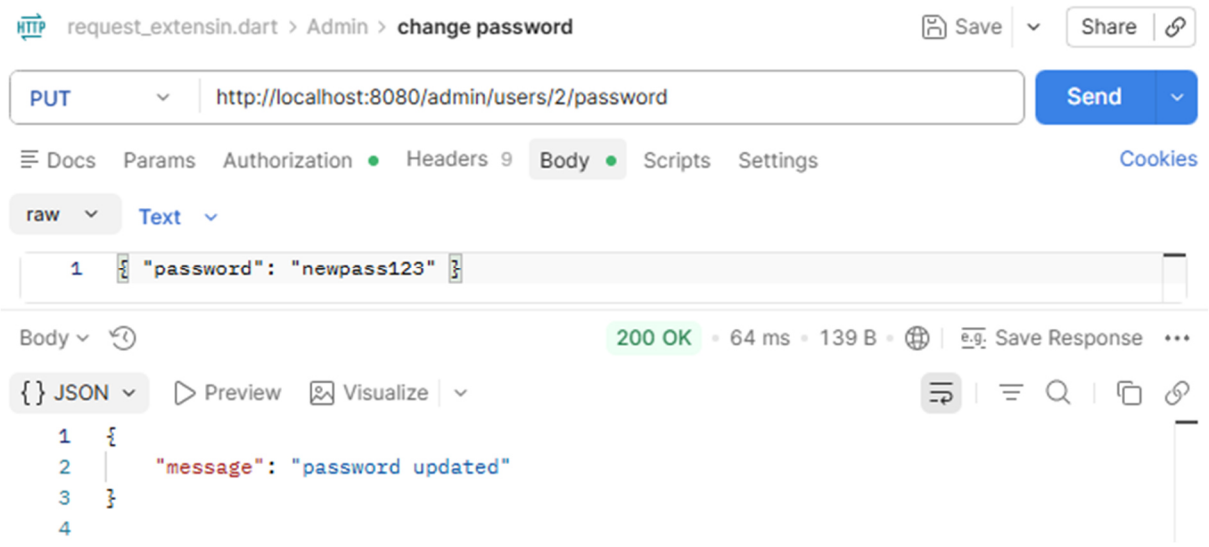

Рисунок 3.15 – Скріншот Postman: перевірка авторизації зі старим паролем

Тест 13: Виключення учасника адміном. Відбувається перевірка на коректність роботи функціоналу шляхом виключення учасника з команди адміністратором та оновлення списку учасників (ліст. 3.6).

Лістинг 3.6 – REST запит на виключення учасника адміном

```
DELETE /admin/tasks/1/participants/2
Authorization: Bearer <admin_token>
```

кінець лістингу 3.6

Очікуваний результат: сервер повертає код 200 OK, учасника успішно виключено з команди, а пов'язані з ним задачі залишаються в системі без змін (рис. 3.17).

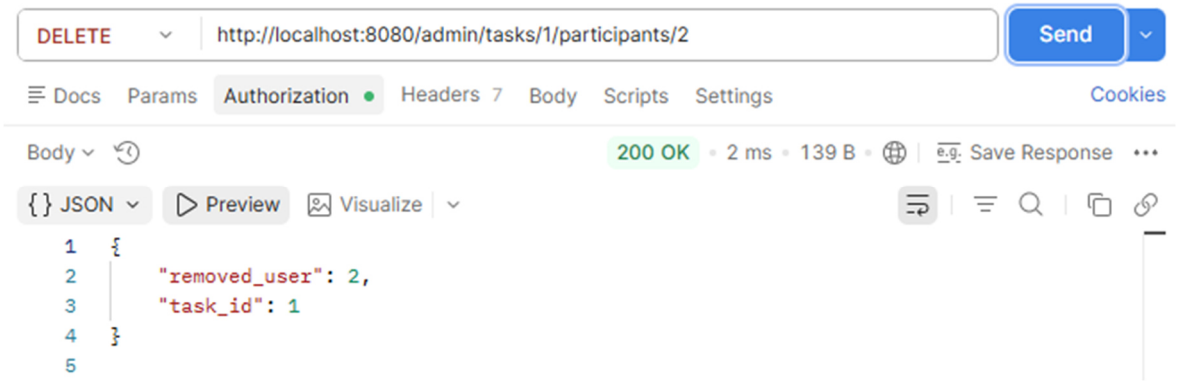

Рисунок 3.17 – Скріншот Postman: адмін виключає учасника із задачі

У процесі розробки та тестування було виявлено та усунено наступні технічні проблеми.

Помилка CGO при компіляції – драйвер github.com/mattn/go-sqlite3 вимагає компілятора C. Проблему вирішено заміною на modernc.org/sqlite – чисту реалізацію SQLite на Go без залежності від CGO.

Помилка схеми БД – при додаванні нових колонок (user_id, deadline, role) до вже існуючої бази даних оператор CREATE TABLE IF NOT E□ISTS не оновлює схему. Вирішено видаленням файлу tasks.db перед повторним запуском та наступними запусками.

Конфлікт порту 8080 – при повторному запуску сервера без зупинки попереднього процесу. Вирішено завершенням процесу через Stop-Process -Id <PID> -Force у PowerShell.

3.3 Інструкція користувача з інсталяції та використання програмного продукту

Для розгортання та використання програмного продукту необхідно:

– операційна система: Windows 10/11, macOS 12+, Linux (Ubuntu 20.04+);

– Go: Версія 1.21 або вище;
– вільний порт: 8080 (для HTTP API);
– дисковий простір: Не менше 50 МБ;
– оперативна пам’ять: Не менше 64 МБ.

Завантажити інсталятор Go з офіційного сайту: https://go.dev/dl/. Встановити відповідно до операційної системи [17]. Перевірити встановлення у терміналі (ліст. 3.7).

Лістинг 3.7 – Перевірка версії Go

```
go version
```

кінець лістингу 3.7

Склонувати репозиторій або розпакувати архів із вихідним кодом у обрану директорію (ліст. 3.8).

Лістинг 3.8 – Клонування репозиторію із GitHub

```
git clone https://github.com/anastasia/toDoDiploma.git
cd toDoDiploma
```

кінець лістингу 3.8

Або, якщо отримано у вигляді архіву: розпакувати архів у довільну папку із відповідною назвою. Відкрити термінал (PowerShell / CMD / bash) та перейти в папку проєкту (ліст. 3.9):

Лістинг 3.9 – Зміна директорії

```
cd C:\Projects\toDoDiploma
```

кінець лістингу 3.9

Виконати у кореневій директорії проєкту (ліст. 3.10).

Лістинг 3.10 – Завантаженння всіх залежностей

```
go mod download
```

кінець лістингу 3.10

Після встановлення проєкту необхідно автоматично завантажити всі залежності, зазначені у файлі go.mod. Для цього використовується стандартний механізм керування модулями мови Go. Після завантаження бібліотек запуск HTTP API виконується у кореневій директорії проєкту за допомогою даної команди (ліст. 3.11)

Лістинг 3.11 – Запуск HTTP API сервера

```
go run ./cmd/api
кінець лістингу 3.11
```

кінець лістингу 3.11

У разі успішного запуску в терміналі відобразяться службові повідомлення про ініціалізацію системи та запуск сервера (ліст. 3.12).

Лістинг 3.12 – Повідомлення про успішний запуск API сервера

```
2026/05/15 10:00:00 Admin user ensured (login: admin / admin123)
2026/05/15 10:00:00 API server running on :8080
```

кінець лістингу 3.12

Після цього API-сервер буде доступний за адресою http://localhost:8080. Якщо порт 8080 вже використовується іншим процесом, його можна звільнити за допомогою команд PowerShell (ліст. 3.13).

Лістинг 3.13 – Перевірка зайнятості порту 8080 у PowerShell та завершення процесу

```
netstat -ano | findstr :8080
Stop-Process -Id <PID> -Force
```

кінець лістингу 3.13

Для запуску консольного інтерфейсу застосунку необхідно виконати команду нижче (ліст. 3.14).

Лістинг 3.14 – Запуск CLI-застосунку

```
go run ./cmd/cli
```

кінець лістингу 3.14

Після запуску застосунку користувачу відображається інтерактивне меню довідки, яке автоматично формує перелік підтримуваних команд, їх параметрів та призначення (ліст. 3.15).

Лістинг 3.15 – Запуск CLI-застосунку

```
===                    ToDo                    CLI                    ===
Commands: register <user> <pass> | login <user> <pass> | list | add <title>
[deadline    RFC3339]    |    done    <id>    |    delete    <id>    |    quit
>
```

кінець лістингу 3.15

У відповідь сервер повертає JWT-токен, значення якого необхідно скопіювати та використовувати у всіх наступних запитах для проходження автентифікації. Токен передається через HTTP-заголовок Authorization у форматі поданому в лістингу (ліст. 3.16).

Лістинг 3.16 – Інтерактивне меню CLI-застосунку

```
Authorization: Bearer <token>
```

кінець лістингу 3.16

Для початку роботи із системою через HTTP API використовується інструмент Postman або будь-який інший клієнт для виконання HTTP-запитів. На першому етапі необхідно зареєструвати нового користувача шляхом виконання POST-запиту на адресу: http://localhost:8080/auth/register.

У Postman: вкладка Authorization → тип Bearer Token → вставити токен (рис. 3.18).

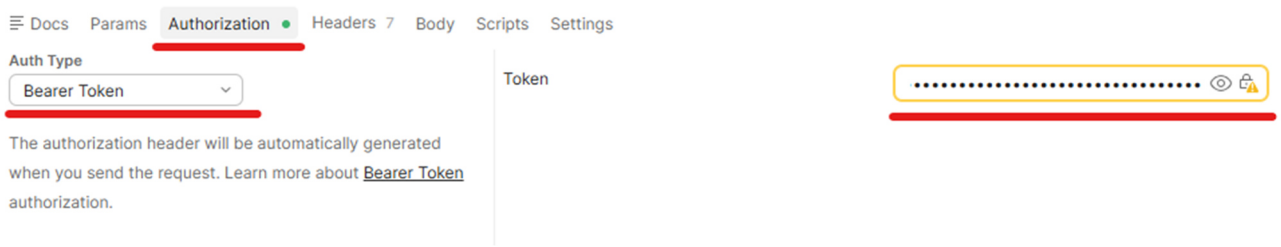

Рисунок 3.18 – Скріншот Postman: налаштування Bearer Token у вкладці Authorization

У таблиці 3.1 наведено основні сценарії взаємодії адміністратора із системою, зокрема автентифікацію, керування обліковими записами користувачів, перегляд інформації та виконання інших адміністративних операцій. Представлені варіанти використання відображають функціональні можливості, доступні користувачу з адміністративними правами, та демонструють механізми контролю й керування сутностями системи.

Таблиця 3.1 – Адмін-операції

Дія	Метод	URL
Список усіх користувачів	GET	/admin/users
Змінити пароль користувача	PUT	/admin/users/{id}/password
Список усіх задач	GET	/admin/tasks
Видалити будь-яку задачу	DELETE	/admin/tasks/{id}
Виключити учасника з задачі	DELETE	/admin/tasks/{id}/participants/{userID}

У таблиці 3.2 представлено базові операції взаємодії з REST API, що забезпечують управління користувачами та задачами системи.

Таблиця 3.2 – Основні команди CLI

Команда	Опис	Приклад
register <user> <pass>	Реєстрація	register alice pass123
login <user> <pass>	Вхід у систему	login alice pass123
list	Список задач	list
add <title> [deadline]	Додати задачу	add "Зробити звіт" 2026-06-01T18:00:00Z
done <id>	Позначити виконаною	done 3
delete <id>	Видалити / вийти	delete 3
token	Показати поточний токен	token
quit	Вийти з програми	quit

Для входу як адміністратор використовувати облікові дані: логін: admin та пароль: admin123. Рекомендація змінити пароль адміністратора після першого входу в продуктивному середовищі.

РОЗДІЛ 4
СПЕЦІАЛЬНІ РОЗРАХУНКИ

Ергономіка програмного забезпечення – це галузь, що вивчає відповідність програмних продуктів потребам і можливостям користувача з метою підвищення ефективності, зручності та безпеки роботи.

Відповідно до стандарту ISO 9241-11, зручність використання (usability) визначається трьома основними характеристиками: ефективністю (можливістю досягнення мети), продуктивністю (витратами ресурсів) та задоволеністю (суб'єктивним сприйняттям користувача).

Розроблений програмний продукт надає два інтерфейси взаємодії: HTTP REST API для інтеграції з клієнтськими застосунками та інтерактивний командний рядок (CLI) для прямої роботи користувача. Ергономічна оцінка проводилась для обох інтерфейсів.

API є основним інтерфейсом взаємодії для клієнтських застосунків та розробників. Ергономіка API оцінюється за критеріями передбачуваності, консистентності та інформативності відповідей.

– структура URL-маршрутів відповідає загальноприйнятим REST-конвенціям;

– іменники у множині для ресурсів (/tasks, /users);

– ієрархічна структура для вкладених ресурсів (/tasks/{id}/participants);

– логічне розмежування публічних, захищених та адміністративних маршрутів (/auth/, /tasks/, /admin/).

Інформативність відповідей (табл. 4.1). Кожен ендпоінт повертає стандартизовані відповіді, що відповідають принципам REST та забезпечують однаковий формат обміну даними між клієнтською та серверною частинами системи. Це дозволяє спростити інтеграцію різних клієнтів, зокрема CLI та зовнішніх застосунків, які можуть використовувати API без додаткової адаптації. Стандартизація відповідей також підвищує передбачуваність поведінки системи при різних сценаріях використання.

Таблиця 4.1 – HTTP-статус-коди

Ситуація	HTTP-статус
Успішне отримання даних	200 OK
Успішне створення ресурсу	201 Created
Успішне видалення	204 No Content
Некоректний запит	400 Bad Request
Відсутня автентифікація	401 Unauthorized
Недостатньо прав	403 Forbidden
Ресурс не знайдено	404 Not Found
Неприпустимий метод	405 Method Not Allowed

При виникненні помилки у тілі відповіді передається текстове повідомлення, яке пояснює причину, наприклад: «forbidden: only task owner or admin can add participants» або «invalid deadline format, use RFC3339». Це суттєво скорочує час діагностики проблем під час розробки та інтеграції.

Консистентність формату даних. Усі відповіді API повертаються у форматі JSON із заголовком Content-Type: application/json.

Поля дат використовують єдиний стандарт RFC3339 («2026-06-01T23:59:00Z»), що забезпечує сумісність із будь-якою мовою програмування.

Виявлений недолік та усунення. У початковій реалізації JSON-поле власника задачі називалося user_id, що створювало неоднозначність – незрозуміло, чи це поточний користувач, автор або призначена особа. Недолік усунено: поле перейменовано на owner_id, що однозначно вказує на роль.

Ергономічна оцінка CLI-інтерфейсу**.** Командний інтерфейс розроблено відповідно до принципів мінімалізму та передбачуваності.

При запуску відображається список доступних команд (табл. 4.2), що усуває необхідність звертатися до документації під час роботи.

Таблиця 4.2 – Оцінка за критеріями ергономіки CLI

Критерій	Оцінка	Коментар
Підказки при запуску	Реалізовано	Список команд виводиться автоматично
Зрозумілі повідомлення про помилки	Реалізовано	"Usage: add <title>", "Invalid id"
Відображення стану задачі	Реалізовано	[x] / [ ] перед назвою

Продовження таблиці 4.2

Критерій	Оцінка	Коментар
Відображення дедлайну	Реалізовано	| deadline: 2026-06-01T23:59:00Z
Підтвердження дій	Реалізовано	"Task #3 deleted.", "Added task #1"
Безпечна робота без логіну	Реалізовано	Захист перевіркою currentUserID == 0

Виявлений недолік. CLI не підтримує команди для управління учасниками задач. Користувач CLI може лише переглядати задачі, до яких його додали через API. Для повноцінного управління учасниками необхідно використовувати HTTP API. Це задокументовано в інструкції користувача.

Прийоми безпечної роботи. При роботі з програмним продуктом рекомендується дотримуватися наступних правил безпечної роботи.

Робоче місце. Монітор розташовувати на відстані 50-70 см від очей, верхній край екрана – на рівні очей або нижче. Кут нахилу екрана – 10-20° від вертикалі.

Режим праці та відпочинку. Відповідно до санітарних норм ДСанПіН 3.3.2-007-98, при роботі за комп’ютером слід робити перерви тривалістю 10-15 хвилин кожну годину роботи.

Освітлення. Природне освітлення має падати зліва. Рівень штучного освітлення на робочому столі – не менше 300-500 люкс. Уникати відблисків на екрані.

Пароль адміністратора. Після першого запуску системи обов’язково змінити стандартний пароль адміністратора (admin123) на надійний пароль довжиною не менше 12 символів із використанням літер різного регістру, цифр та спеціальних символів.

Зберігання токенів. JWT-токени не слід зберігати у відкритому вигляді у текстових файлах або передавати через незахищені канали зв’язку.

Після першого запуску системи обов’язково необхідно змінити стандартний пароль адміністратора (admin123) на надійний пароль довжиною не

менше 12 символів із використанням літер різного регістру, цифр та спеціальних символів. Це дозволить знизити ризик несанкціонованого доступу до системи та захистити службові функції адміністрування від потенційних загроз.

JWT-токени не слід зберігати у відкритому вигляді в текстових файлах або передавати через незахищені канали зв’язку. У разі компрометації токена стороння особа може отримати доступ до функцій системи від імені користувача. Для підвищення рівня безпеки рекомендується використовувати захищені механізми зберігання облікових даних та регулярно виконувати повторну авторизацію.

Під час експлуатації програмного продукту також необхідно забезпечувати належний рівень захисту даних шляхом регулярного створення резервних копій бази даних. Це дозволяє уникнути втрати інформації внаслідок технічних збоїв, помилок користувачів або інших непередбачуваних ситуацій. Особливу увагу слід приділяти захисту доступу до файлів бази даних та обмеженню прав користувачів, які не мають безпосереднього відношення до адміністрування системи.

У разі розгортання застосунку в мережевому середовищі рекомендується використовувати захищені протоколи передачі даних, зокрема HTTPS, що забезпечує шифрування інформації під час обміну між клієнтом і сервером. Додатково необхідно своєчасно оновлювати програмне забезпечення, середовище виконання Go та сторонні бібліотеки, оскільки оновлення часто містять виправлення виявлених помилок і вразливостей безпеки.

ВИСНОВОК

У ході виконання кваліфікаційної роботи бакалавра було розроблено програмний продукт «Система управління задачами з REST API та CLI-інтерфейсом», призначений для організації, контролю та спільного виконання задач у багатокористувацькому середовищі. На початковому етапі роботи проведено аналіз існуючих систем управління задачами, визначено їхні переваги та недоліки, а також сформовано функціональні вимоги до розроблюваного програмного забезпечення.

У процесі проєктування було розроблено архітектуру системи з розділенням на окремі логічні шари, що забезпечує незалежність бізнес-логіки від механізмів зберігання даних та інтерфейсів взаємодії з користувачем. Для зберігання інформації використано реляційну базу даних SQLite, що дозволило створити автономне рішення без необхідності розгортання окремого сервера баз даних. Спроєктована структура бази даних підтримує зв’язки типу «багато до багатьох» між користувачами та задачами, завдяки чому реалізовано механізм спільного доступу до задач із можливістю призначення декількох учасників.

Програмний продукт реалізовано мовою програмування Go з використанням сучасних підходів до розробки серверних застосунків. У системі впроваджено механізми реєстрації та автентифікації користувачів на основі JWT-токенів, рольову модель доступу з розподілом прав між звичайними користувачами та адміністраторами, а також засоби захисту облікових записів шляхом зберігання паролів у вигляді bcrypt-хешів. Реалізований REST API забезпечує виконання всіх основних операцій над користувачами та задачами через HTTP-запити, а CLI-інтерфейс надає можливість працювати із системою без використання графічного середовища.

Функціональні можливості розробленого програмного продукту включають створення, перегляд, редагування та видалення задач, керування статусами виконання, спільний доступ до задач для декількох користувачів, а також адміністративні операції з керування користувачами та системними

даними. Особливістю реалізованого рішення є підтримка колаборативної роботи, за якої одна задача може належати одному власнику та одночасно бути доступною іншим учасникам.

Для перевірки працездатності системи проведено функціональне та інтеграційне тестування. Під час тестування було перевірено коректність роботи REST API, механізмів автентифікації та авторизації, обробку помилкових запитів, а також взаємодію між окремими модулями системи. Отримані результати підтвердили правильність реалізації основних функцій та відповідність програмного продукту поставленим вимогам.

Практична цінність роботи полягає у створенні готової до використання системи управління задачами, яка може застосовуватися як персональний органайзер, навчальний проєкт або основа для подальшого розвитку корпоративних систем планування та контролю виконання робіт. Завдяки модульній архітектурі програмний продукт може бути розширений шляхом додавання вебінтерфейсу, системи сповіщень, календарного планування, аналітичних модулів та інших компонентів.

Таким чином, поставлену мету кваліфікаційної роботи досягнуто повністю. Розроблена система забезпечує повний цикл управління задачами, підтримує багатокористувацький режим роботи, реалізує сучасні механізми автентифікації та розмежування прав доступу, а також відповідає вимогам надійності, безпеки та можливості подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Go Blog. The Go Programming Language. URL: https://go.dev/blog/ (дата звернення: 15.04.2026).

2. SQLite documentation. SQLite Home Page. URL: https://sqlite.org/docs.html (дата звернення: 01.03.2026).

3. Izaguirre E. SQLite in Modern Web Production: Dreams Becoming Reality. Medium. URL: https://medium.com/data-science/sqlite-in-production-dreams-becoming-reality-94557bec095b (дата звернення: 01.02.2026).

4. Go packages – go packages. Go Packages – Go Packages. URL: https://pkg.go.dev/ (дата звернення: 01.02.2026).

5. OWASP REST Security Cheat Sheet. URL: https://cheatsheetseries.owasp.org/REST_Security_Cheat_Sheet.html (дата звернення: 01.03.2026).

6. OWASP Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/Authentication_Cheat_Sheet.html (дата звернення: 15.02.2026).

7. OWASP foundation, the open source foundation for application security | OWASP foundation. OWASP Foundation, the Open Source Foundation for Application Security. URL: https://owasp.org/ (дата звернення: 18.03.2026).

8. Go standard library documentation. URL: https://pkg.go.dev/std (дата звернення: 16.02.2026).

9. OWASP Top Ten. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 15.03.2026).

10. Сучасні підходи до проектування програмного забезпечення. Київ : КМ Акад., 2021. 371 с.

11. Modernc.org/sqlite package documentation. URL: https://pkg.go.dev/modernc.org/sqlite (дата звернення: 01.04.2026).

12. Tsoukalos M. Mastering Go: Harness the power of Go to build professional utilities and concurrent servers and services. Packt, 2021. 660 p.

13. Harsanyi T. 100 go mistakes and how to avoid them. Manning, 2022. 384 p.

14. Sheffer Y., Hardt D., Jones M. RFC 8725: JSON web token best current practices. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc8725 (дата звернення: 10.04.2026.

15. Go 1.21 release notes – the go programming language. The Go Programming Language. URL: https://go.dev/doc/go1.21#introduction (дата звернення: 29.03.2026).

16. Brypt package documentation. URL: https://pkg.go.dev/golang.org/x/crypto/bcrypt (дата звернення: 29.03.2026).

17. Bodner J. Learning go: an idiomatic approach to real-world go programming. O'Reilly Media, 2024. 400 p.

ДОДАТКИ

ДОДАТОК А

Сертифікат про участь у міжнародній науково-практичній конференції

СІНТЕХ 2026

СЕРТИФІКАТ

засвідчує, що

Бобела Анастасія Сергіївна

взяла участь у

І Міжнародній науково-практичній конференції:
Сучасні інформаційні технології: від теорії до практики

загальним обсягом 15 годин (0,5 кредитів ECTS),
яка проводилася 22-23 травня 2026 року
в Луцькому національному технічному університеті

Ректор ЛНТУ Ірина ВАХОВИЧ

mintech-conf.lntu.edu.ua
/cert/2026-365

Луцький національний технічний університет
вул. Львівська, 75 м. Луцьк
Волинська обл. 43018