МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТЕФАНИКА
Кафедра комп’ютерних наук та інформаційних систем

Дипломна робота
на здобуття першого (бакалаврського) рівня вищої освіти на тему

‘’Розробка веб-програми для управління завданнями в колективі’’

Виконав: студент IV курсу,групи ІСТ-41 126-інформаційно системні технології
Швайнога Н.Т

Керівник Стрілецький Ю.Й
Рецензент _____
(прізвище та ініціали)

Івано-Франківськ – 2025 р.

Зміст

ВСТУП

Сучасний світ характеризується стрімким розвитком інформаційних технологій, що впливають на всі сфери людської діяльності, зокрема на організацію праці в колективах. Ефективне управління завданнями є ключовим фактором успіху командної роботи, особливо в умовах зростання обсягів інформації та складності проєктів. Відсутність зручних та адаптованих інструментів для координації діяльності колективу часто призводить до зниження продуктивності, порушення термінів виконання завдань і ускладнення комунікації між учасниками. У цьому контексті розробка веб-програми для управління завданнями в колективі стає актуальною науково-практичною проблемою, що потребує комплексного підходу до вирішення.

Актуальність проекту зумовлена необхідністю підвищення ефективності командної роботи шляхом створення спеціалізованого програмного забезпечення, яке б відповідало потребам сучасних колективів. Критичний аналіз існуючих рішень, таких як Trello, Asana чи Jira, свідчить про їхні переваги, зокрема інтуїтивність інтерфейсу та широкий функціонал, однак виявляє й недоліки: обмежену адаптивність до специфічних потреб команд, складність інтеграції в локальні робочі процеси та високу вартість для малих організацій. Розробка власної веб-програми дозволяє врахувати ці питання, забезпечивши гнучкість, доступність і безпеку. Науково-практична значущість теми полягає в можливості створення інструменту, який сприятиме оптимізації робочих процесів, а суспільна цінність – у підвищенні ефективності співпраці в колективах різного масштабу. Автор роботи брав активну участь у всіх етапах дослідження – від аналізу потреб користувачів до реалізації та тестування системи, що забезпечило практичну спрямованість результатів.

Об’єктом дослідження є процес управління завданнями в колективі як комплекс взаємопов’язаних дій, спрямованих на організацію, координацію та контроль діяльності команди. **Предметом дослідження** виступає веб-програма, розроблена для автоматизації цього процесу, що є частиною об’єкта та дозволяє

конкретизувати напрямок аналізу й розробки. Саме предмет дослідження визначає фокус роботи – створення програмного рішення, яке оптимізує взаємодію в команді.

Метою роботи є розробка веб-програми для управління завданнями в колективі, яка забезпечить ефективну координацію діяльності, доступ до актуальної інформації та контроль виконання проєктів. Для досягнення цієї мети визначено такі **завдання**:

- Вивчити потреби користувачів у системі управління завданнями.
- Проаналізувати існуючі рішення та виявити їхні сильні й слабкі сторони.
- Обґрунтувати вибір технологій для розробки веб-програми.
- Визначити функціональні вимоги до системи та спроєктувати її архітектуру.
- Розробити основні модулі веб-програми й реалізувати користувацький інтерфейс.
- Провести тестування системи для перевірки її працездатності.
- Встановити проблеми інтеграції та перспективи вдосконалення розробленого рішення.

Ці завдання структурують зміст роботи та визначають логіку викладу матеріалу в її розділах.

Стан наукової розробки: проблематика управління завданнями в колективі активно досліджується в сучасній літературі. Праці таких авторів, як Керзнер [1], присвячені управлінню проєктами, підкреслюють важливість автоматизації процесів. Дослідження в галузі розробки програмного забезпечення (Sommerville I., [2]) акцентують увагу на необхідності адаптивних інтерфейсів і безпечних архітектур. Водночас аналіз джерел, таких як документація до Streamlit, Flask і SQLite, свідчить про їхню популярність у створенні веб-додатків завдяки простоті використання та ефективності. Проте недостатня кількість досліджень, присвячених інтеграції таких технологій у локальні системи управління завданнями, обґрунтовує новизну цієї роботи.

У роботі використано комплекс **методів**: аналіз літератури й аналогів – для оцінки потреб і стану розробки; системний підхід – для проектування архітектури; методи програмування – для реалізації модулів; тестування – для перевірки працездатності. Методологічною основою слугували принципи об’єктно-орієнтованого програмування та користувацького дизайну, що забезпечили достовірність результатів.

Практичне значення одержаних результатів: розроблена веб-програма має прикладне значення, оскільки може бути використана колективами для організації роботи, відстеження прогресу та обміну інформацією. Рекомендації щодо її інтеграції в робочі процеси дозволяють адаптувати систему до потреб конкретних команд, а відкритий код сприяє подальшому вдосконаленню.

Апробація результатів: результати дослідження були представлені групових зборах та засіданнях кафедри.

Структура роботи: робота складається зі вступу, трьох розділів, висновків, списку використаних джерел, 12 таблиць, 23 рисунків і 2 додатка, що містять код програми та UML-діаграми. У першому розділі проаналізовано потреби та обґрунтовано вибір технологій. Другий розділ присвячено проектуванню системи, а третій – її реалізації та тестуванню. У висновках узагальнено результати дослідження.

РОЗДІЛ 1. АНАЛІЗ ПОТРЕБ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

1.1. Аналіз потреб користувачів у системі управління завданнями

У сучасному динамічному середовищі бізнесу, громадських організацій та інших структур виникає нагальна потреба в ефективному управлінні завданнями та їх розподілі серед учасників робочого процесу. Розробка аналітичних систем, що дозволяють автоматизувати цей процес, стає актуальною через зростаючу складність проєктів, географічну розподіленість команд та необхідність швидкого прийняття рішень. Аналіз потреб користувачів є ключовим етапом для обґрунтування вибору технологій, що забезпечать ефективність і зручність такої системи.

Користувачі сучасних інформаційних систем поділяються на кілька категорій (рисунок 1.1) [3]. Кожна з цих груп має власні потреби та очікування від системи розподілу і керування завданнями, які можна охарактеризувати наступним чином [4].

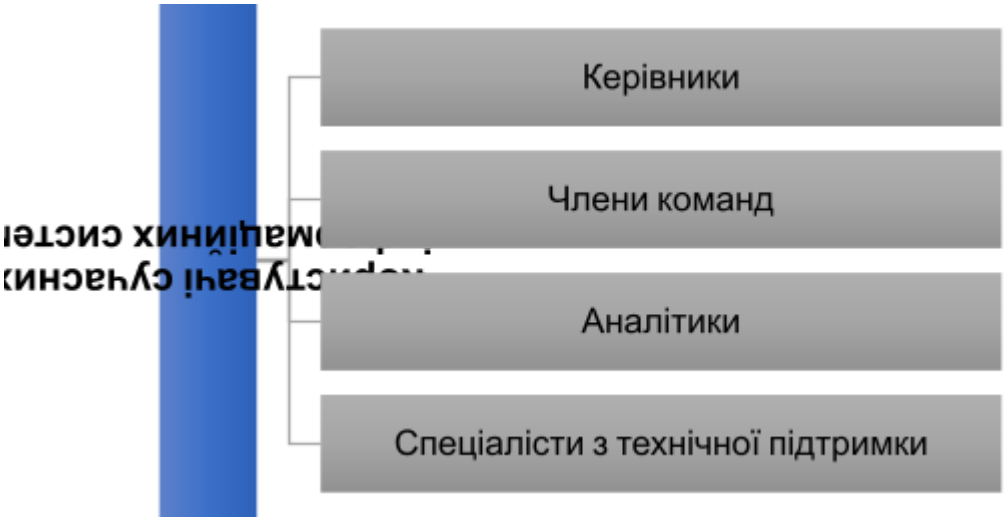

Рисунок 1.1. Користувачі сучасних інформаційних систем

Основними задачами керівників є постановка цілей, делегування обов'язків, контроль виконання завдань і коригування планів [5]. Їхні потреби описані в таблиці 1.1 [6].

Таблиця 1.1

Потреби керівників та менеджерів проєктів

№ з/п	Потреби	Опис
1	Інтуїтивно зрозумілий інтерфейс	Керівники потребують швидкого доступу до даних про стан виконання завдань, без необхідності тривалого навчання роботі з системою
2	Гнучке управління ресурсами	Система повинна забезпечувати можливість перерозподілу завдань у режимі реального часу з урахуванням пріоритетів та доступності виконавців
3	Аналітичні інструменти	Для прийняття обґрунтованих рішень важливо мати доступ до візуалізованих даних про продуктивність команди, витрати часу та інші ключові показники ефективності (KPI)
4	Мобільність	Здатність працювати з будь-якої точки світу є важливою для сучасного керівника, що обумовлює потребу у додатку з синхронізацією даних у реальному часі

Члени команд – виконавці завдань зацікавлені у мінімізації часу на пошук необхідної інформації та максимальній зручності роботи із системою [7]. Їхні потреби представлені в таблиці 1.2 [8].

Таблиця 1.2

Потреби членів команди проєктів

№ з/п	Потреби	Опис
1	2	3
1	Прозорість завдань	Користувачі повинні чітко розуміти свої завдання, строки їх виконання, а також критерії оцінки результатів
2	Можливість зворотного зв’язку	Інструменти комунікації, такі як чати або форуми в системі, необхідні для уточнення деталей і вирішення проблем у процесі роботи

Продовження таблиці 1.2

1	2	3
3	Персоналізація	Система має враховувати індивідуальні уподобання користувачів, наприклад, щодо налаштувань сповіщень чи зовнішнього вигляду інтерфейсу
4	Інтеграція з іншими платформами	Члени команд часто використовують різні інструменти для виконання завдань, тому система має підтримувати інтеграцію з популярними сервісами, такими як Google Drive, Slack або Microsoft Teams

Аналітична функція системи є важливою для оцінки ефективності виконання завдань та виявлення можливостей для оптимізації. Потреби аналітиків подані в таблиці 1.3 [9].

Таблиця 1.3

Потреби аналітиків

№ з/п	Потреби	Опис
1	Доступ до великих обсягів даних	Система повинна збирати, зберігати й обробляти дані про виконання завдань, зокрема часові витрати, продуктивність окремих виконавців та загальну ефективність команди
2	Гнучкі засоби візуалізації	Візуалізація даних у вигляді графіків, діаграм чи таблиць сприяє більш швидкому аналізу інформації та прийняттю рішень
3	Експорт даних	Для створення звітів важлива можливість експорту інформації у зручному форматі (наприклад, Excel або PDF)

Категорія користувачів – спеціалісти з технічної підтримки – займається обслуговуванням системи, забезпечуючи її працездатність і вирішуючи технічні проблеми [10]. Основні потреби показані в таблиці 1.4 [11].

Таблиця 1.4

Потреби спеціалістів з технічної підтримки

№ з/п	Потреби	Опис
1	Зручність адміністрування	Інструменти управління користувачами, базами даних та оновленнями повинні бути зрозумілими і функціональними
2	Логування і діагностика	Для швидкого усунення збоїв система повинна автоматично створювати звіти про помилки
3	Можливість кастомізації	Система повинна дозволяти адаптувати функціонал відповідно до специфічних вимог організації

Загалом, ключовими потребами користувачів є зручність, швидкість доступу до інформації, прозорість робочих процесів та можливість інтеграції з іншими інструментами. Важливу роль відіграють також питання безпеки, такі як захист даних і контроль доступу, які забезпечують довіру користувачів до системи [12].

Аналіз потреб користувачів аналітичної системи розподілу і керування завданнями дозволяє визначити основні вимоги до її функціоналу, зокрема гнучкість, інтегрованість, зручність і мобільність. Урахування цих потреб є ключовим для розробки ефективного рішення, що відповідатиме очікуванням різних категорій користувачів і забезпечуватиме високу продуктивність робочих процесів. На основі цього аналізу можна обґрунтувати вибір технологій, що сприятимуть створенню конкурентоспроможної системи.

1.2. Аналіз існуючих рішень для управління завданнями в колективі

На сьогоднішній день ринок програмного забезпечення, орієнтованого на управління завданнями, пропонує велику кількість інструментів, що дозволяють автоматизувати процеси розподілу завдань, відслідковувати їх виконання, а також проводити аналіз ефективності роботи [13]. Серед найбільш популярних рішень у цій сфері можна виділити такі системи, як Trello, Asana, Monday.com, ClickUp, а також спеціалізовані корпоративні платформи на кшталт Microsoft Teams або Slack. Окрім цього, на ринку представлені рішення для галузевих потреб, що

дозволяють інтегрувати функції управління завданнями з аналітичними інструментами.

Trello (рисунок 1.2) є однією з найвідоміших платформ для управління завданнями [14]. Система базується на методології Kanban і пропонує простий і зрозумілий інтерфейс для візуалізації процесів. Trello підтримує роботу з дошками, картками і списками, що дозволяє ефективно організовувати задачі як для індивідуального, так і для командного використання [15]. Система має інтеграції з іншими популярними сервісами, такими як Google Drive, Slack та Jira. Попри це, Trello не володіє розширеними аналітичними функціями, що обмежує його застосування в складних корпоративних процесах.

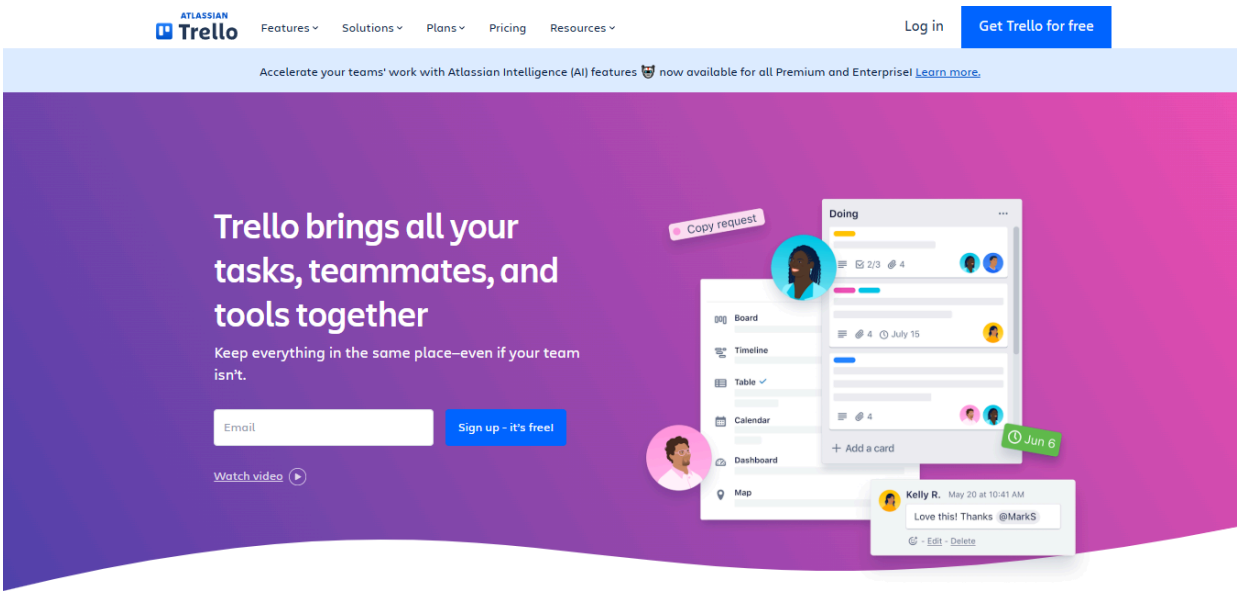

Рисунок 1.2. Головна сторінка trello.com

Asana (рисунок 1.3) є ще одним популярним інструментом для управління завданнями, орієнтованим на корпоративне середовище [16]. Її основною перевагою є розширена функціональність для планування проектів, постановки задач та відслідковування прогресу. Asana дозволяє будувати складні ієрархії завдань, налаштовувати залежності між ними та генерувати аналітичні звіти [17]. Однак, як і у випадку з Trello, система може бути обмеженою в контексті інтеграції спеціалізованих інструментів для бізнес-аналітики.

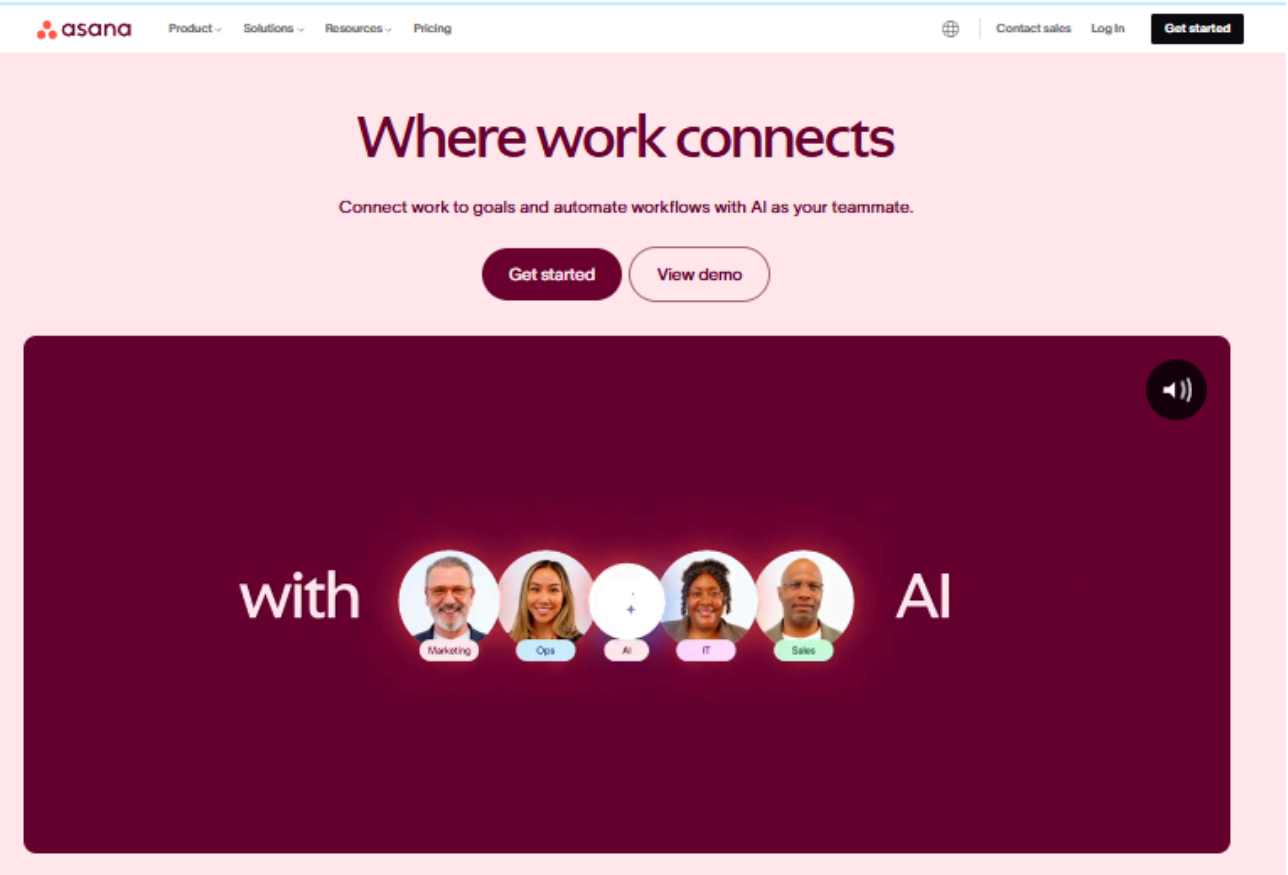

Рисунок 1.3. Головна сторінка asana.com

Monday.com (рисунок 1.4) пропонує багатофункціональну платформу, яка поєднує в собі інструменти для управління завданнями, комунікації в команді та бізнес-аналітики [18]. Гнучкість у налаштуванні робочих процесів дозволяє адаптувати систему до потреб будь-якої організації, а інтеграція з такими інструментами, як Microsoft Office, Zoom та Google Workspace, спрощує взаємодію з іншими сервісами [19]. Головним недоліком Monday.com є відносно висока вартість ліцензії, що може обмежувати його доступність для невеликих компаній.

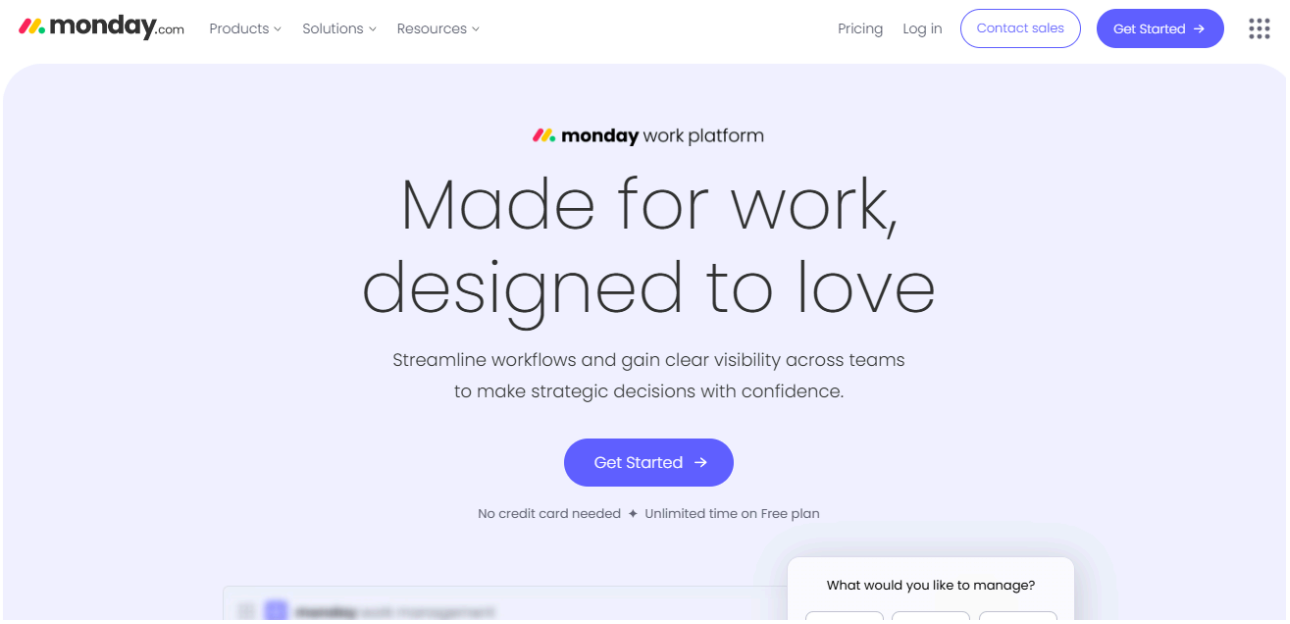

Рисунок 1.4. Головна сторінка Monday.com

ClickUp (рисунок 1.5) позиціонується як універсальне рішення для управління завданнями, яке охоплює широкий спектр функцій – від планування і розподілу задач до проведення бізнес-аналітики та звітування [20]. Серед його переваг – можливість створення індивідуальних робочих просторів, інтеграція з понад 1 000 сторонніх додатків та підтримка автоматизації процесів. ClickUp також підтримує мобільні додатки, що є важливим фактором для забезпечення доступу до інформації в режимі реального часу [21]. Недоліками можна вважати перевантаженість інтерфейсу для початківців та необхідність додаткового навчання для ефективного використання системи.

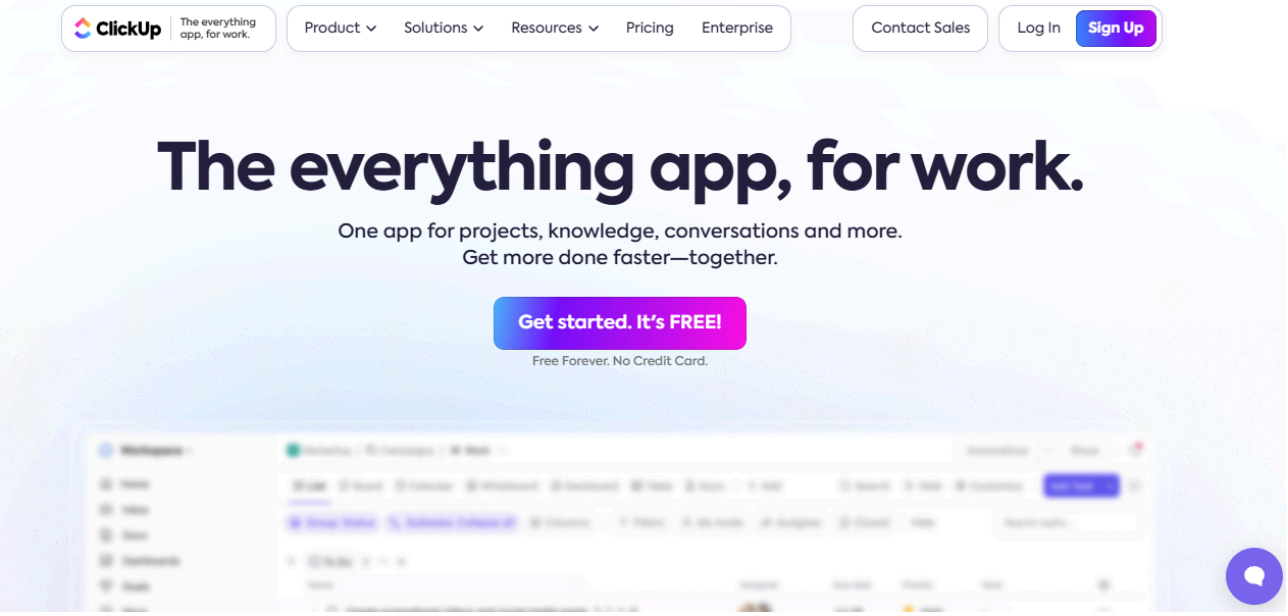

Рисунок 1.5. Головна сторінка clickup.com

Окрім вищезазначених рішень, варто звернути увагу на корпоративні платформи, такі як Microsoft Teams [22] (рисунок 1.6) [23] та Slack (рисунок 1.7) [24], які, хоча й не є спеціалізованими інструментами для управління завданнями, часто використовуються у цьому контексті завдяки інтеграції з іншими системами. Наприклад, Microsoft Teams дозволяє використовувати додаток Planner для управління завданнями, тоді як Slack пропонує широкий вибір інтеграцій з платформами для управління проектами. Основною перевагою таких систем є можливість об'єднання функцій комунікації та управління завданнями, що підвищує загальну продуктивність команди.

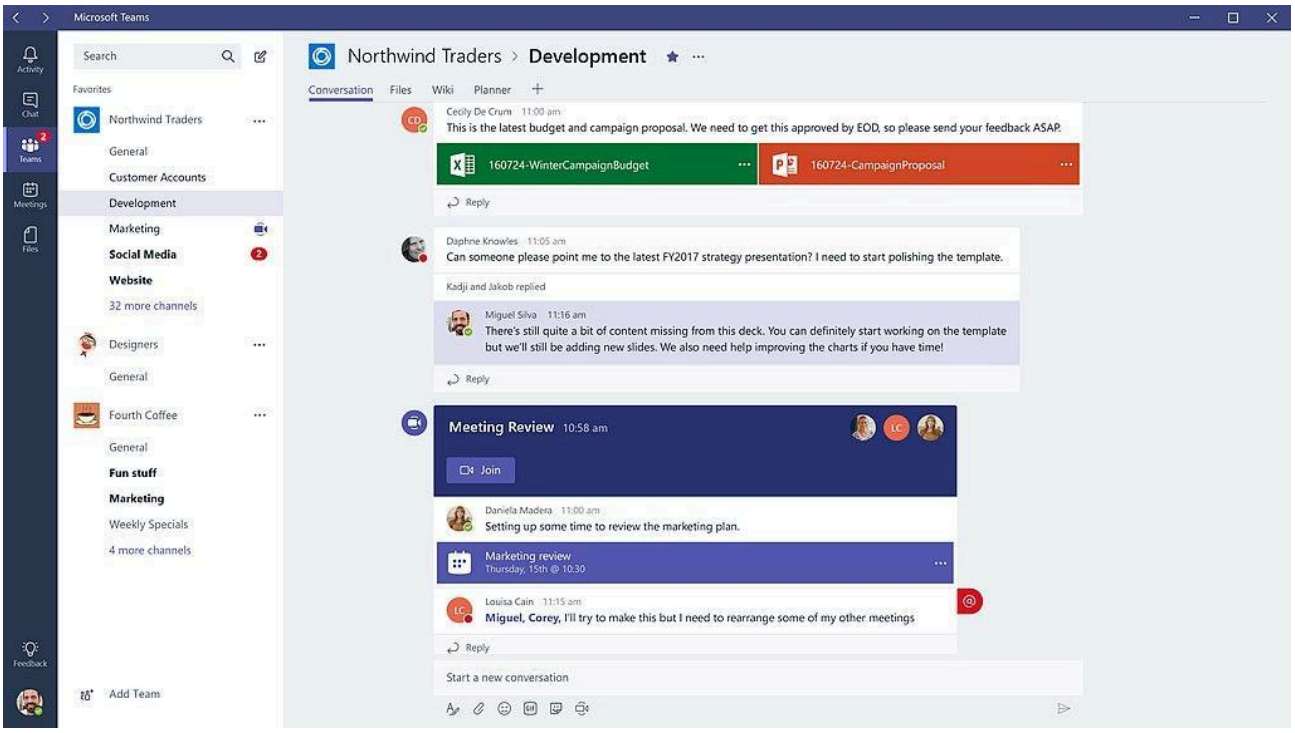

Рисунок 1.6. Сторінка Microsoft Teams

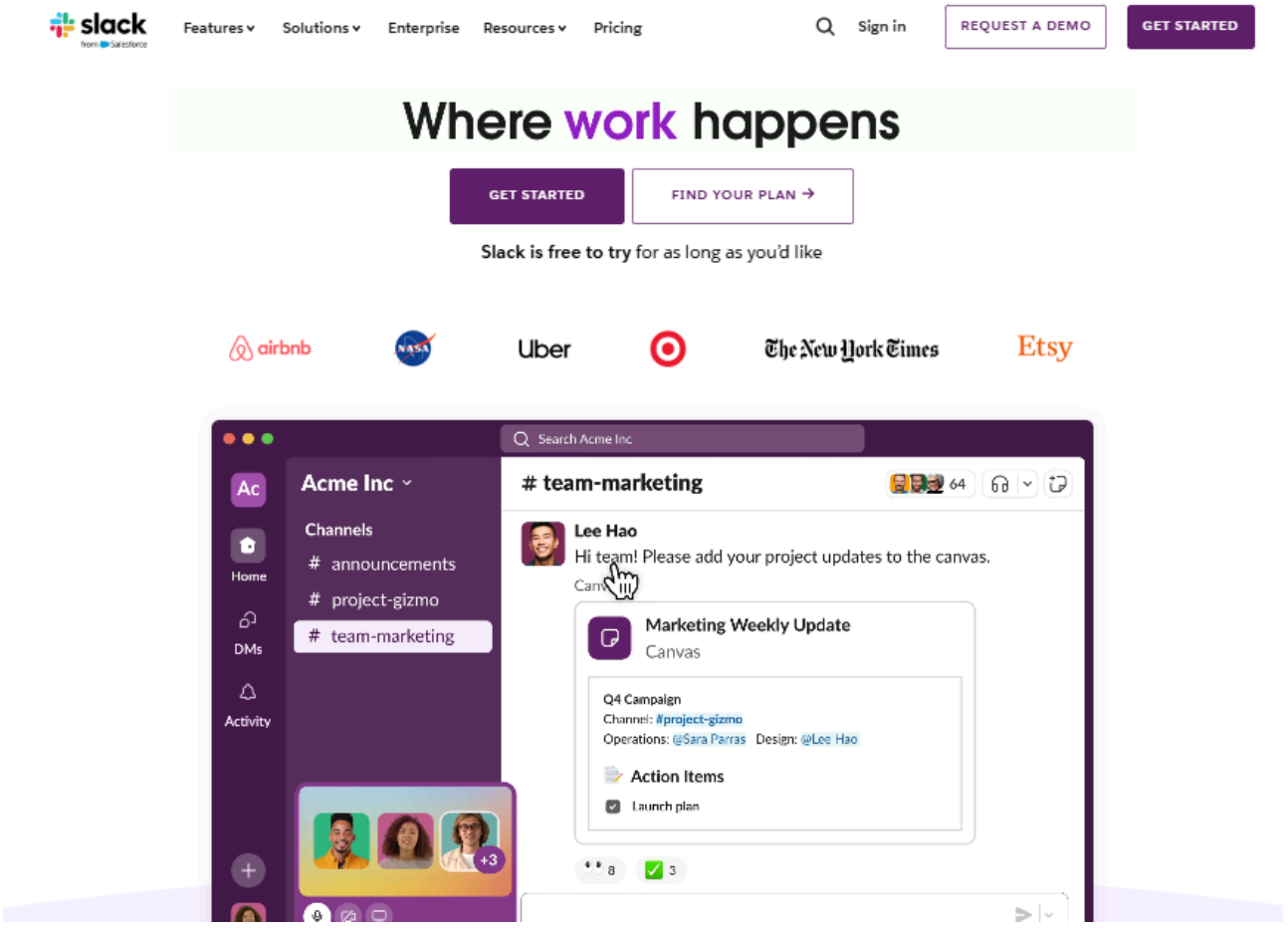

Рисунок 1.7. Головна сторінка

При цьому для вузькоспеціалізованих задач часто застосовуються системи, такі як Jira (рисунок 1.8) [25] або Zoho Projects (рисунок 1.9)[26]. Jira є потужним інструментом для управління проектами в IT-сфері, орієнтованим на Agile-методології, а Zoho Projects забезпечує широкий спектр функцій для автоматизації бізнес-процесів. Обидві системи відзначаються високим рівнем кастомізації та підтримкою розширеного аналізу даних.

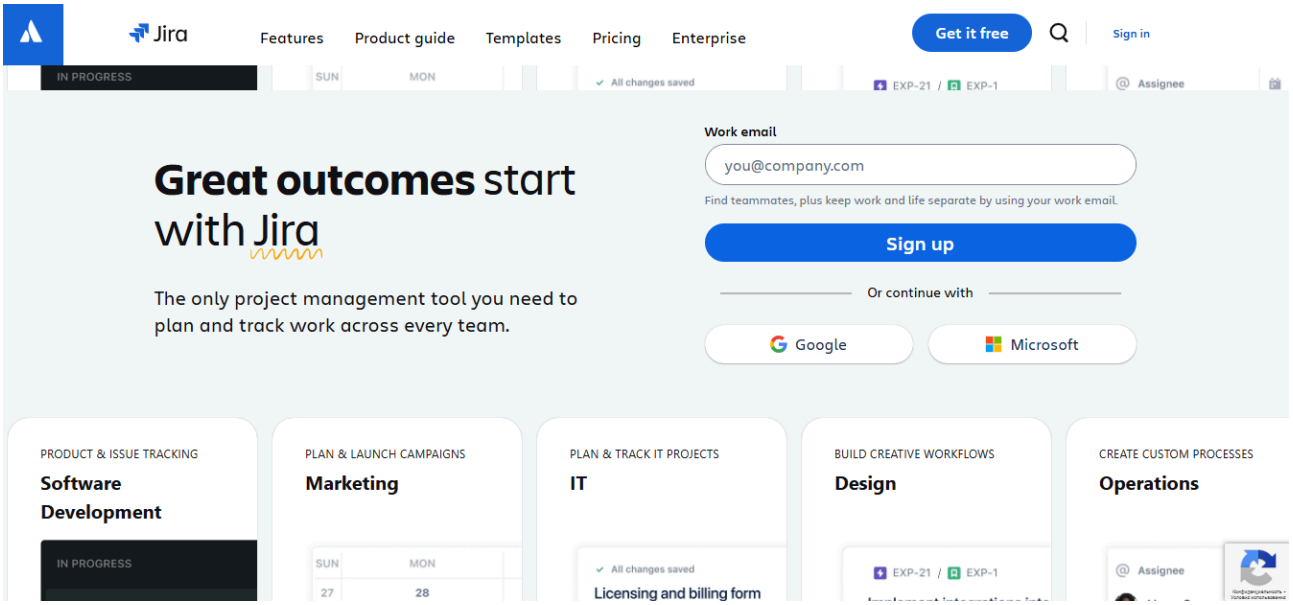

Рисунок 1.8. Головна сторінка Jira

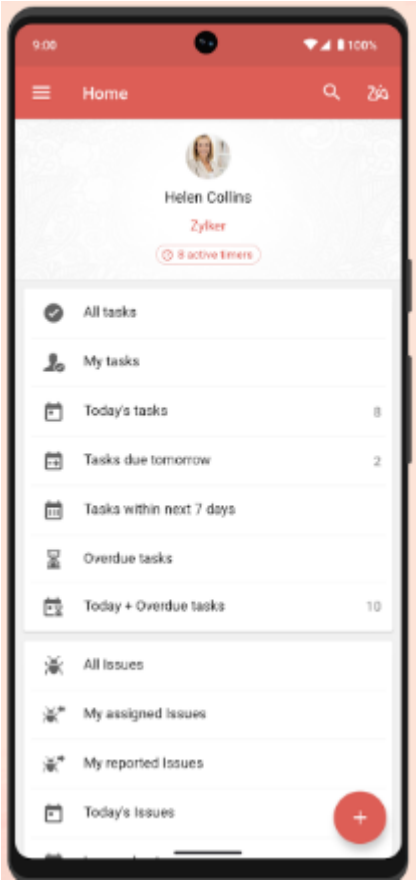

Рисунок 1.9. Домашня сторінка Zoho Projects

Проведений аналіз існуючих рішень показує, що жодна з платформ не є універсальною, кожна має свої переваги та обмеження. Вибір конкретної системи залежить від особливостей організації, складності завдань та необхідності в аналітичних інструментах. Найбільш перспективними є платформи, що забезпечують інтеграцію функцій управління завданнями та аналітики, оскільки вони дозволяють не тільки автоматизувати процеси, але й проводити їх оптимізацію на основі аналізу даних.

Аналіз існуючих рішень у сфері аналітичних систем розподілу і керування завданнями свідчить про наявність широкого вибору інструментів з різноманітною функціональністю. Важливими тенденціями розвитку є інтеграція з аналітичними платформами, підтримка мобільних пристроїв та можливість автоматизації процесів. Результати дослідження дозволяють визначити основні вимоги до майбутньої системи: гнучкість у налаштуванні, можливість інтеграції зі сторонніми сервісами, а також підтримка сучасних підходів до управління

завданнями. Це створює передумови для розробки ефективного рішення, яке задовольнятиме потреби користувачів.

1.3. Обґрунтування вибору технологій

Розробка веб-програми для управління завданнями в колективі є актуальним завданням у сучасному світі інформаційних технологій, де ефективна взаємодія між учасниками команди відіграє ключову роль у досягненні спільних цілей. Вибір технологій для реалізації такого проєкту потребує ретельного аналізу, врахування вимог до продуктивності, зручності використання та масштабованості системи. У цьому контексті важливо обґрунтувати доцільність застосування конкретних інструментів і підходів, таких як Python, Streamlit, Flask та PostgreSQL, які забезпечують створення функціонального й надійного програмного продукту. Метою даного аналізу є детальне пояснення вибору технологій для створення веб-програми, орієнтованої на управління завданнями в колективі, з урахуванням сучасних тенденцій у веб-розробці.

Вибір технологій для розробки веб-програми управління завданнями в колективі є визначальним етапом, який впливає на якість кінцевого продукту, його продуктивність і здатність відповідати потребам користувачів. Система має забезпечувати зручний інтерфейс, швидкий доступ до даних, безпечне зберігання інформації та можливість масштабування залежно від потреб команди. Для досягнення цих цілей обрано набір технологій, який включає мову програмування Python, фреймворки Streamlit і Flask, а також реляційну базу даних PostgreSQL.

1.3.1. Вибір архітектури та мови програмування

Архітектура веб-програми управління завданнями має відповідати сучасним вимогам до швидкості, гнучкості та легкості інтеграції з іншими системами. Для реалізації проєкту обрано архітектуру клієнт-сервер, яка дозволяє розділити функціональність між серверною частиною, відповідальною за обробку даних, і

клієнтською частиною, що забезпечує взаємодію з користувачем через веб-інтерфейс [27]. Такий підхід сприяє підвищенню продуктивності та спрощує подальше розширення системи.

Мова програмування Python обрана завдяки її універсальності, простоті синтаксису та широкій підтримці бібліотек, що значно прискорюють процес розробки [28]. Python є однією з найпопулярніших мов програмування у 2025 році, зокрема для веб-розробки, завдяки своїй здатності працювати з різноманітними фреймворками та інструментами [29]. Ця мова дозволяє швидко створювати прототипи, що є важливим для проєктів, орієнтованих на оперативне тестування ідей. Крім того, Python підтримує як серверну логіку через Flask, так і інтерактивні інтерфейси через Streamlit, що робить її оптимальним вибором для комплексної реалізації веб-програми.

1.3.2. Використання бази даних та серверної частини

Для зберігання даних про завдання, користувачів і статуси їх виконання необхідна надійна система управління базами даних. У даному проєкті обрано PostgreSQL — реляційну базу даних із відкритим кодом, яка відзначається високою стабільністю та підтримкою складних запитів [30]. PostgreSQL забезпечує ефективну роботу зі структурованими даними, що є ключовим для управління завданнями, де інформація про дедлайни, відповідальних осіб і статуси має чітку організацію [31]. Ця база даних також підтримує транзакції та механізми забезпечення цілісності даних, що критично важливо для колективної роботи, коли кілька користувачів одночасно вносять зміни.

Порівняно з альтернативами, такими як MySQL, PostgreSQL вирізняється кращою підтримкою розширених типів даних і можливістю роботи з великими обсягами інформації, що робить її придатною для проєктів із перспективою зростання [32]. Водночас, у порівнянні з NoSQL-системами, такими як MongoDB, PostgreSQL є кращим вибором для цього проєкту через чітко визначену структуру даних, яка не потребує гнучкості неструктурованого підходу.

1.3.3. Вибір серверної платформи

Серверна частина веб-програми відповідає за обробку запитів, логіку управління завданнями та взаємодію з базою даних. Для її реалізації обрано фреймворк Flask, побудований на Python. Flask є легким і гнучким інструментом, який ідеально підходить для створення RESTful API — інтерфейсу, через який клієнтська частина отримує доступ до даних [33]. Завдяки своїй мінімалістичній структурі Flask дозволяє розробникам швидко налаштувати серверну логіку, не переобтяжуючи проєкт зайвими залежностями.

Перевагою Flask перед іншими фреймворками, такими як Django, є його простота й менша ресурсоємність, що особливо важливо для проєктів середнього масштабу, де не потрібні всі функції повноцінного фреймворку [34]. У той же час Flask забезпечує достатню гнучкість для інтеграції з PostgreSQL і реалізації всіх необхідних ендпоінтів для управління завданнями, такими як створення, редагування та видалення записів. Альтернативою міг би бути Node.js з Express.js, однак вибір Python і Flask зумовлений єдиною екосистемою з іншими компонентами проєкту, що спрощує розробку та підтримку.

1.3.4. Інтерфейс користувача та досвід взаємодії

Для створення інтерактивного веб-інтерфейсу використано Streamlit — фреймворк на базі Python, який дозволяє швидко розробляти зручні й візуально привабливі додатки [35]. Streamlit орієнтований на простоту використання: він не потребує глибоких знань фронтенд-технологій, таких як HTML, CSS чи JavaScript, що скорочує час розробки. Цей інструмент ідеально підходить для створення інтерфейсу управління завданнями, де користувачі можуть переглядати списки завдань, додавати нові записи та відстежувати прогрес у реальному часі.

Streamlit забезпечує адаптивність інтерфейсу до різних пристроїв, що є важливим для колективної роботи, коли учасники можуть використовувати як настільні комп’ютери, так і мобільні гаджети [36]. Порівняно з традиційними

фронтенд-фреймворками, такими як React, Streamlit зменшує складність розробки, зберігаючи при цьому високу функціональність. Дизайн інтерфейсу відповідає принципам Material Design, що гарантує інтуїтивність і зручність для користувачів із різним рівнем технічної підготовки.

1.3.5. Безпека та захист даних

Безпека є одним із пріоритетів у веб-програмі для колективного управління завданнями, адже система може містити конфіденційну інформацію про проєкти та персональні дані користувачів. Для захисту даних під час передачі між клієнтом і сервером використовується протокол HTTPS із шифруванням TLS, що запобігає перехопленню інформації [27]. Аутентифікація користувачів реалізована через JSON Web Tokens (JWT), які забезпечують безпечний доступ до системи та дозволяють визначати права доступу залежно від ролі учасника команди [28].

Зберігання паролів здійснюється у зашифрованому вигляді за допомогою алгоритму bcrypt, який гарантує високий рівень захисту від атак на паролі [29]. Крім того, PostgreSQL підтримує вбудовані механізми шифрування даних на рівні бази, що додатково підвищує безпеку [30]. Такий комплексний підхід до захисту інформації відповідає сучасним стандартам кібербезпеки та забезпечує надійність системи навіть у разі зростання кількості користувачів.

Вибір технологій для розробки веб-програми управління завданнями в колективі базується на їхній здатності забезпечити стабільність, швидкодію, безпеку та зручність використання. Поєднання архітектури клієнт-сервер, мови програмування Python, фреймворків Flask і Streamlit, а також бази даних PostgreSQL дозволяє створити ефективний інструмент, який відповідає потребам сучасних команд. Ці технології оптимально збалансовані між простотою реалізації, продуктивністю та можливістю масштабування, що робить їх доцільним вибором для даного проєкту.

Висновки до розділу 1

Проведене дослідження потреб користувачів та аналіз існуючих рішень для управління завданнями в колективі, а також обґрунтування вибору технологій свідчать про актуальність розробки веб-програми, спрямованої на підвищення ефективності командної роботи в умовах сучасного динамічного середовища. Аналіз потреб різних категорій користувачів, зокрема керівників, виконавців, аналітиків і спеціалістів технічної підтримки, дозволив сформувати комплексний погляд на вимоги до системи, серед яких ключовими виявилися зручність інтерфейсу, гнучкість у розподілі ресурсів, аналітичні можливості, мобільність і безпека даних. Вивчення популярних платформ, таких як Trello, Asana, Monday.com, ClickUp, Microsoft Teams, Slack, Jira та Zoho Projects, показало, що кожна з них має свої сильні сторони, однак жодна не є універсальним рішенням, здатним повною мірою задовольнити всі потреби сучасних команд. Це підкреслює необхідність створення системи, яка поєднувала б простоту використання з розширеним функціоналом і можливістю адаптації до специфічних умов роботи.

Обґрунтування вибору технологій для реалізації веб-програми спирається на їхню здатність забезпечити стабільність, швидкодію та зручність взаємодії з системою. Використання архітектури клієнт-сервер у поєднанні з мовою програмування Python дозволяє досягти гнучкості й швидкості розробки, тоді як фреймворки Flask і Streamlit забезпечують ефективну реалізацію серверної логіки та інтерактивного інтерфейсу. Застосування бази даних PostgreSQL гарантує надійне зберігання структурованої інформації та підтримує складні операції з даними, що є важливим для управління завданнями в реальному часі. Особлива увага приділена питанням безпеки, що реалізується через шифрування даних, безпечну аутентифікацію та контроль доступу, що відповідає сучасним стандартам захисту інформації.

Загалом, проведена робота створює міцну основу для розробки веб-програми, яка відповідатиме очікуванням користувачів і сприятиме оптимізації робочих процесів у колективі. Поєднання ретельного аналізу потреб із

продуманим вибором технологій дає змогу створити конкурентоспроможне рішення, яке враховує як поточні вимоги, так і перспективи подальшого розвитку. Такий підхід забезпечує не лише практичну цінність системи, а й її адаптивність до змін у потребах команд, що є важливим фактором у контексті швидкого розвитку інформаційних технологій і зростання складності проєктів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ПРОГРАМИ ДЛЯ УПРАВЛІННЯ ЗАВДАННЯМИ В КОЛЕКТИВІ

2.1. Функціональні вимоги до системи

Сучасні умови розвитку інформаційних технологій вимагають створення ефективних інструментів для координації роботи в колективі, особливо коли йдеться про управління завданнями. Проектування веб-програми, спрямованої на організацію спільної діяльності, передбачає чітке визначення функціональних вимог, які забезпечать зручність використання, гнучкість системи та її відповідність потребам користувачів. Цей процес є ключовим етапом, адже від нього залежить здатність програми оптимізувати робочі процеси, підвищувати продуктивність та сприяти прозорій взаємодії між учасниками команди. У даному контексті розглядаються функціональні вимоги до системи, які формують основу для реалізації всіх необхідних можливостей.

Функціональні вимоги до веб-програми для управління завданнями в колективі визначають перелік можливостей, які система повинна забезпечувати для виконання основних завдань користувачів. Ці вимоги враховують специфіку роботи різних ролей у команді (адміністраторів, менеджерів, спеціалістів), а також необхідність інтеграції різноманітних інструментів для ефективної взаємодії. Детальний опис ключових функціональних вимог, сформульованих з урахуванням потреб сучасних робочих колективів, наведено в таблиці 2.1.

Таблиця 2.1

Функціональні вимоги

№ з/п	Вимоги	Опис
1	2	3
1	Аутентифікація та авторизація користувачів	Система повинна забезпечувати безпечний доступ до програми через процедуру аутентифікації. Користувачі вводять унікальні облікові дані (електронну пошту та пароль), після чого отримують доступ до функціоналу відповідно до своєї ролі. Реалізація механізму JWT (JSON Web Token) дозволяє зберігати сесію користувача та автоматично оновлювати токени для підтримання безперервної роботи. Крім того, передбачається можливість реєстрації нових користувачів із вибором ролі (спеціаліст або менеджер), а також обмеження доступу до певних функцій залежно від прав (наприклад, адміністратор має повний контроль, тоді як спеціалісти — лише частковий)
2	Управління проєктами	Однією з основних функцій є створення, редагування та видалення проєктів. Менеджери можуть ініціювати нові проєкти, вказуючи їхню назву, опис, дедлайн і максимальну кількість спеціалістів. Система забезпечує відображення списку всіх доступних проєктів із детальною інформацією, такою як статус, кількість учасників та прогрес виконання. Спеціалісти мають можливість приєднуватися до активних проєктів, якщо кількість учасників не перевищує встановленого ліміту. Адміністратори отримують додаткові права для архівації або видалення проєктів
3	Створення та управління завданнями	Програма дозволяє менеджерам створювати завдання в рамках проєктів із зазначенням назви, опису, дедлайну, пріоритету та відповідального спеціаліста. Завдання відображаються у вигляді дошки Kanban із трьома статусами: «Не розпочато», «В процесі» та «Завершено». Користувачі можуть змінювати статус завдань, переміщаючи їх між колонками, а також редагувати деталі (наприклад, призначати нового виконавця). Система автоматично сповіщає призначених спеціалістів про нові завдання

Продовження таблиці 2.1

1	2	3
4	Календар подій	Для планування діяльності передбачено інтеграцію календаря, який відображає події, пов’язані з проєктом (зустрічі, дедлайни, інші заходи). Менеджери можуть додавати нові події, вказуючи тип, час початку та завершення, а також опис. Календар підтримує різні режими перегляду (місяць, тиждень, день) і дозволяє всім учасникам проєкту бачити актуальний розклад. Події диференціюються за типами з відповідним кольоровим кодуванням для зручності сприйняття
5	Система сповіщень	Програма включає механізм сповіщень, який інформує користувачів про важливі зміни: призначення завдань, оновлення проєктів, наближення дедлайнів тощо. Сповіщення відображаються у вигляді спливаючого вікна з можливістю позначення їх як прочитаних. Кількість непрочитаних сповіщень відображається в інтерфейсі для швидкого доступу. Система зберігає історію сповіщень із зазначенням часу створення та пов’язаного проєкту
6	Обговорення в проєктах	Для підтримки комунікації між учасниками передбачено функціонал коментарів. Кожен користувач (крім адміністратора) може додавати коментарі до проєкту, які відображаються у хронологічному порядку із зазначенням автора та часу публікації. Цей інструмент сприяє обміну ідеями, уточненню деталей завдань і вирішенню робочих питань у реальному часі
7	Управління файлами	Система дозволяє завантажувати, зберігати та завантажувати файли, пов’язані з проєктами. Користувачі можуть додавати документи, зображення чи інші матеріали, які автоматично прив’язуються до відповідного проєкту. Інтерфейс відображає список файлів із метаданими (назва, розмір, автор, дата завантаження), а також забезпечує можливість їх завантаження учасниками проєкту. Обмеження на типи файлів і розмір забезпечують безпеку та оптимальне використання ресурсів

Завершення таблиці 2.1

1	2	3
8	Оцінка виконання	Менеджери мають можливість оцінювати роботу спеціалістів у рамках проєкту, виставляючи числові оцінки (від 0 до 100) та додаючи текстові коментарі. Оцінки зберігаються в системі та доступні для перегляду як менеджеру, так і відповідному спеціалісту. Цей функціонал сприяє прозорості оцінювання та мотивує учасників до якісного виконання завдань
9	Адміністрування користувачів	Адміністратори отримують доступ до панелі управління, де можуть переглядати список усіх користувачів, їхні ролі та статистику участі в проєктах. Передбачено можливість видалення користувачів із системи, що автоматично анулює їхню участь у проєктах і пов’язані записи (завдання, коментарі тощо). Цей функціонал забезпечує контроль над складом команди та захист даних
10	Візуалізація статистики	Для аналізу прогресу проєктів система генерує статистичні дані, такі як кількість виконаних завдань, середня оцінка спеціалістів і загальний прогрес. Ці дані відображаються у вигляді графіків і таблиць, що дозволяє менеджерам і адміністраторам оцінювати ефективність роботи команди. Спеціалісти можуть переглядати власну статистику для самоконтролю
11	Інтерфейс користувача	Програма забезпечує інтуїтивно зрозумілий інтерфейс із бічною панеллю для навігації, персоналізованою інформацією про користувача та швидким доступом до основних функцій (проєкти, сповіщення, вихід). Дизайн підтримує адаптивність, використання кольорової палітри для диференціації елементів і кастомізовані стилі для підвищення зручності роботи
12	Логування активності	Система фіксує дії користувачів (створення проєктів, оновлення завдань, завантаження файлів тощо) у вигляді логів. Ці записи доступні адміністраторам для моніторингу роботи та аналізу поведінки користувачів, що сприяє підвищенню безпеки та прозорості
13	Вихід із системи	Користувачі можуть завершити сеанс роботи, натиснувши кнопку "Вийти", що анулює їхній токен і повертає на сторінку входу. Цей механізм гарантує безпеку даних при завершенні роботи

Функціональні вимоги до веб-програми для управління завданнями в колективі створюють основу для реалізації комплексного інструменту, який відповідає потребам сучасних команд. Описані можливості охоплюють усі ключові сторони спільної роботи — від аутентифікації та управління проєктами до комунікації та аналізу продуктивності. Такий підхід дозволяє забезпечити гнучкість, безпеку та зручність використання, що є критично важливим для ефективної координації діяльності в колективі. Реалізація цих вимог сприяє створенню системи, яка не лише оптимізує робочі процеси, а й підвищує мотивацію учасників завдяки прозорості та доступності інформації.

2.2. Проектування архітектури системи

Розробка веб-програми для управління завданнями в колективі потребує створення продуманої архітектури, яка забезпечить ефективну взаємодію між учасниками, стабільність роботи системи та її здатність до масштабування. Архітектура такої системи має враховувати розподіл функціональних компонентів, способи обробки даних, забезпечення безпеки та зручність інтерфейсу для користувачів із різними ролями. У цьому контексті проектування архітектури передбачає детальний аналіз взаємозв’язків між клієнтською та серверною частинами, базою даних і механізмами управління доступом, а також створення структури, яка підтримує як поточні вимоги, так і можливі майбутні розширення. Важливо розглянути детальний опис етапів і компонентів архітектури системи, розробленої для вирішення завдань колективного управління проєктами.

Архітектура системи базується на клієнт-серверній моделі, яка є оптимальною для веб-програм, орієнтованих на взаємодію багатьох користувачів. Клієнтська частина відповідає за відображення інтерфейсу та обробку дій користувача, тоді як серверна частина забезпечує логіку обробки запитів, управління даними та їх збереження. Такий розподіл дозволяє досягти чіткої сепарації обов’язків між компонентами, що полегшує підтримку та подальший розвиток системи. Використання RESTful API як основного механізму взаємодії

між клієнтом і сервером забезпечує стандартизований обмін даними, що сприяє інтеграції з іншими системами в майбутньому.

Клієнтська складова системи розроблена як односторінковий додаток (Single Page Application, SPA), що дозволяє створювати динамічний і швидкий інтерфейс користувача. Основним інструментом для її реалізації обрано бібліотеку Streamlit, яка забезпечує швидке створення інтерактивних веб-інтерфейсів із підтримкою Python. Структура клієнтської частини включає ключові елементи показані в таблиці 2.2.

Таблиця 2.2

Ключові елементи структури клієнтської частини

№ з/п	Елементи	Опис
1	Інтерфейс користувача	Розроблено з урахуванням трьох типів ролей — адміністраторів, менеджерів і спеціалістів. Кожен тип користувача отримує доступ до відповідних функцій через бічну панель і вкладки. Наприклад, менеджери можуть створювати проєкти та призначати завдання, тоді як спеціалісти мають змогу приєднуватися до проєктів і змінювати статуси завдань
2	Обробка подій	Логіка взаємодії з користувачем реалізується через обробники подій, які викликають відповідні запити до серверної частини. Наприклад, натискання кнопки "Створити проєкт" ініціює відправлення POST-запиту з даними форми
3	Візуалізація даних	Використовуються бібліотеки Plotly та Streamlit Calendar для відображення статистики проєктів і подій у календарі. Це забезпечує наочне представлення прогресу завдань і планування діяльності
4	Управління станом	Для збереження стану сесії (наприклад, даних користувача чи поточного проєкту) застосовується механізм session_state, що дозволяє уникнути надмірних запитів до сервера при перемиканні між сторінками

Клієнтська частина також включає кастомізацію стилів через CSS, що інтегрується за допомогою HTML-розмітки. Це дозволяє створювати єдину

кольорову палітру та адаптивний дизайн, що підвищує зручність використання для всіх категорій користувачів.

Серверна складова системи побудована на базі фреймворку Flask, який забезпечує легкість у розробці та гнучкість у реалізації API. Вона виконує роль посередника між клієнтом і базою даних, обробляючи запити та повертаючи відповідні результати. Основні компоненти серверної частини приведені в таблиці 2.3.

Таблиця 2.3

Ключові елементи структури серверної частини

№ з/п	Елементи	Опис
1	API-ендпоінти	Розроблено набір RESTful ендпоінтів для управління проєктами, завданнями, користувачами, файлами, коментарями та сповіщеннями. Наприклад, ендпоінт "/projects" підтримує методи GET для отримання списку проєктів і POST для їх створення
2	Аутентифікація та авторизація	Використовується механізм JWT (JSON Web Tokens) для забезпечення безпеки. Кожен запит до захищених ендпоінтів супроводжується токеном, який перевіряється декоратором @token_required. Це дозволяє обмежувати доступ до функцій залежно від ролі користувача (адміністратор, менеджер, спеціаліст)
3	Обробка запитів	Логіка обробки запитів включає валідацію вхідних даних, взаємодію з базою даних і генерацію відповідей у форматі JSON. Наприклад, при створенні завдання перевіряється наявність обов’язкових полів і права доступу менеджера
4	Логування	Реалізовано систему логування через бібліотеку logging, яка записує ключові події (помилки, успішні операції) для подальшого аналізу та дебагінгу

Серверна частина також підтримує завантаження та скачування файлів, що зберігаються в локальній директорії. Структура директорій організується за ідентифікаторами проєктів, що полегшує управління файлами.

Для зберігання даних обрано реляційну базу даних SQLite, яка забезпечує достатню продуктивність для невеликих і середніх колективів, а також простоту

інтеграції з Python. Схема бази даних (рисунок 2.1) включає основні таблиці, що описані в таблиці 2.4.

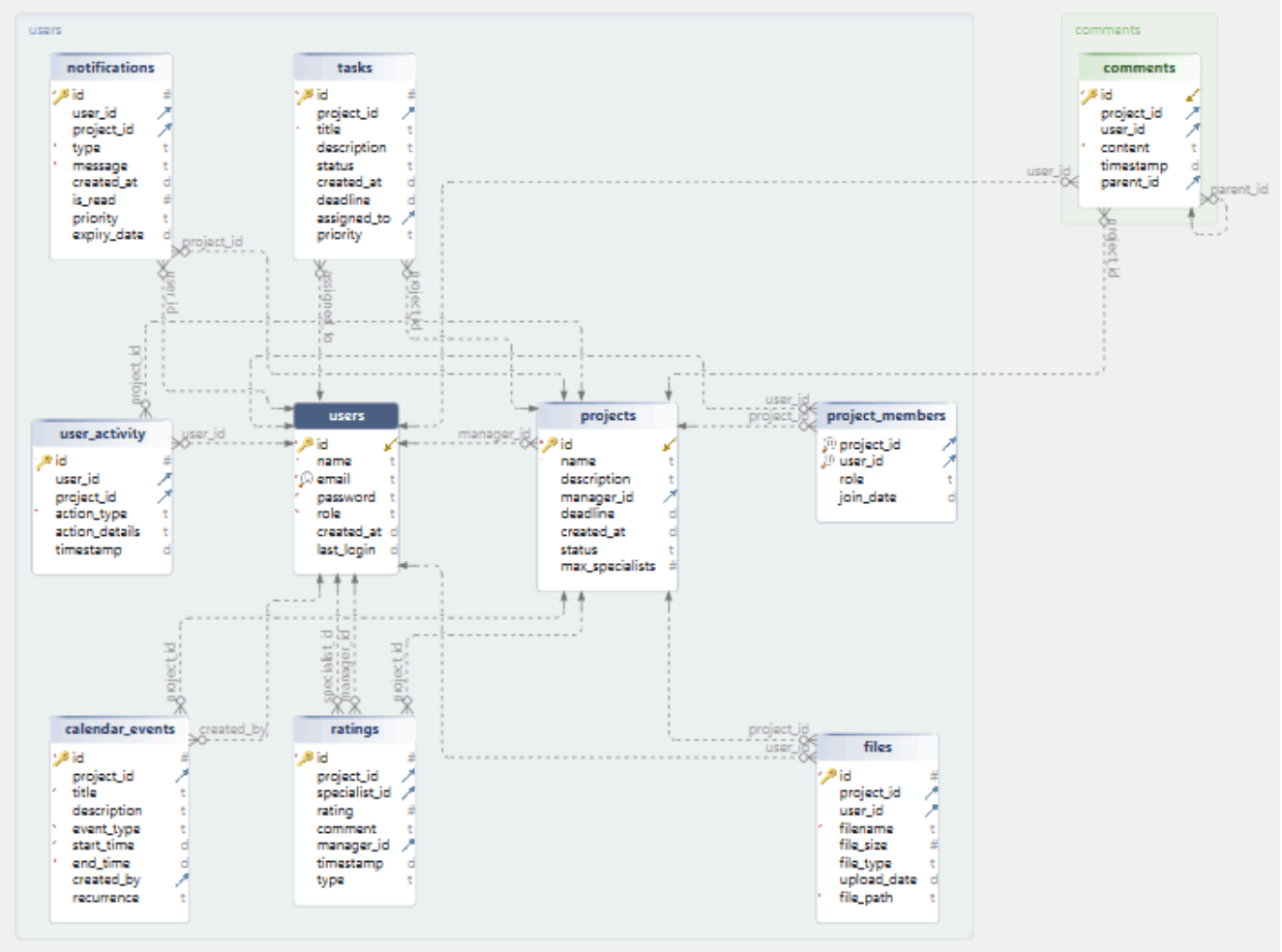

Рисунок 2.1. Структурна схема бази даних

Таблиця 2.4

Таблиці бази даних системи

№ з/п	Таблиці	Опис
1	2	3
1	users	Зберігає інформацію про користувачів (ім’я, email, пароль, роль)
2	projects	Містить дані про проєкти (назва, опис, дедлайн, менеджер, статус)
3	project_members	Зв’язує користувачів із проєктами, визначаючи їхню роль у проєкті
4	tasks	Зберігає завдання з атрибутами (назва, опис, статус, дедлайн, виконавець)

Продовження таблиці 2.4

1	2	3
5	calendar_events	Фіксує події календаря (назва, тип, час початку та завершення)
6	notifications	Містить сповіщення для користувачів із зазначенням статусу прочитання
7	comments, files, ratings	Додаткові таблиці для коментарів, файлів і оцінок виконання
8	user_activity	Логує активність користувачів для аналізу їхньої роботи

Таблиці пов’язані через зовнішні ключі, що забезпечує цілісність даних. Наприклад, поле manager_id у таблиці projects посилається на ідентифікатор користувача з таблиці users. Використання SQLite дозволяє швидко розгортати систему без необхідності складного серверного адміністрування, хоча для масштабування в майбутньому можлива міграція на PostgreSQL чи MySQL.

Взаємодія між клієнтською та серверною частинами здійснюється через HTTP-запити. Клієнт формує запити до API (GET, POST, PUT, DELETE), передаючи дані у форматі JSON або multipart/form-data (для файлів). Сервер обробляє ці запити, звертається до бази даних через SQL-запити та повертає відповідь. Наприклад:

- При створенні проєкту клієнт відправляє POST-запит на "/projects" із даними (назва, опис, дедлайн), сервер записує їх у таблицю projects і повертає ідентифікатор створеного проєкту.
- При оновленні статусу завдання PUT-запит на "/projects/{project_id}/tasks/{task_id}" змінює поле status у таблиці tasks.

Для асинхронного оновлення інтерфейсу (наприклад, сповіщень) клієнт періодично перевіряє їхню кількість через ендпоінт "/notifications/unread-count", що дозволяє уникнути складних механізмів WebSocket на початковому етапі розробки.

Безпека системи забезпечується на кількох рівнях, що представлені в таблиці 2.5.

Таблиця 2.5

Рівні безпеки системи

№ з/п	Рівні	Опис
1	Аутентифікація	Паролі користувачів хешуються за допомогою алгоритму scrypt, а доступ до системи здійснюється через JWT-токени з обмеженим терміном дії (24 години)
2	Авторизація	Ролі користувачів (admin, manager, specialist) визначають доступні функції. Наприклад, лише адміністратори можуть видаляти користувачів, а менеджери — створювати завдання
3	Валідація даних	Усі вхідні дані перевіряються на сервері перед обробкою, що запобігає SQL-ін’єкціям і іншим атакам
4	Обмеження доступу до файлів	Завантажені файли доступні лише учасникам відповідного проєкту, що контролюється через перевірку членства в project_members

Такий підхід гарантує захист даних і чіткий розподіл прав між користувачами.

Архітектура спроєктована з урахуванням можливого розширення:

- Модульна структура API дозволяє додавати нові ендпоінти без значних змін у наявній логіці.
- Використання реляційної бази даних підтримує легку адаптацію до складніших схем (наприклад, додавання таблиць для команд чи категорій завдань).
- Клієнтська частина може бути переписана на React чи Vue.js для підвищення продуктивності при зростанні кількості користувачів.

Це забезпечує гнучкість системи для майбутніх удосконалень, таких як інтеграція з зовнішніми сервісами чи підтримка великої кількості одночасних користувачів.

Розроблена архітектура системи для управління завданнями в колективі є збалансованою комбінацією простоти, функціональності та безпеки.

Клієнт-серверна модель із чітким розподілом обов’язків між компонентами забезпечує зручність використання та стабільність роботи. Використання Streamlit і Flask як основних технологій дозволяє швидко реалізувати базовий функціонал, зберігаючи при цьому потенціал для подальшого розвитку. Реляційна база даних SQLite, RESTful API та механізми безпеки створюють міцну основу для ефективної взаємодії між учасниками проєктів. У майбутньому система може бути адаптована до більших навантажень шляхом переходу на потужніші СУБД або впровадження асинхронних технологій, що підтверджує її перспективність для практичного застосування.

2.3. Інтерфейс користувача

Розробка веб-програми для управління завданнями в колективі вимагає створення продуманого та зручного інтерфейсу користувача, який забезпечує ефективну взаємодію між учасниками команди, спрощує виконання завдань і сприяє прозорості робочих процесів. Інтерфейс користувача відіграє ключову роль у забезпеченні інтуїтивного доступу до функціональних можливостей системи, адаптації до потреб різних ролей (адміністраторів, менеджерів, спеціалістів) та створення комфортного середовища для співпраці. У цьому контексті розглядається детальна структура інтерфейсу, його компоненти, принципи дизайну та їхнє значення для досягнення цілей колективного управління завданнями.

Інтерфейс веб-програми побудовано за принципом модульності та адаптивності, що дозволяє користувачам швидко орієнтуватися в системі. Основна структура включає три ключові зони: верхню панель навігації, бічну панель (sidebar) та основну робочу область. Верхня панель відображає заголовок системи — «Система управління проєктами» — виконаний у бронзово-коричневому кольорі (#806543), що підкреслює корпоративний стиль. Бічна панель слугує для швидкого доступу до основних функцій, таких як перегляд проєктів, сповіщень, керування користувачами (для адміністраторів) та вихід із системи. Вона

оформлена в темно-фіолетовому кольорі (#33266E), що контрастує з білим фоном основної області (#FFFFFF), забезпечуючи чітке візуальне розмежування. Основна робоча зона є динамічною та змінюється залежно від обраної сторінки чи функції, наприклад, списку проєктів, дошки завдань чи календаря.

Колірна схема інтерфейсу ретельно підібрана для створення естетично приємного та функціонального дизайну. Використовується палітра з десяти кольорів, що включає:

- Бронзовий (#806543) — для заголовків і ключових елементів, що привертають увагу;
- Темно-фіолетовий (#33266E) — для бічної панелі та кнопок, що символізує стабільність і авторитетність;
- Білий (#FFFFFF) — основний фон, що забезпечує чистоту та читабельність;
- Світло-сірий (#F8F9FA) — для фону колонок дошки завдань і другорядних елементів;
- Бордовий (#542F34) і рожево-фіолетовий (#A6607C) — акцентні кольори для декоративних деталей;
- Зелений (#4CAF50), червоний (#f44336) і синій (#2196F3) — для позначення статусів (успіх, помилка, інформація).

Такий підхід до кольорової гами не лише сприяє візуальній гармонії, але й допомагає користувачам швидко ідентифікувати статуси чи дії. Наприклад, кнопки за замовчуванням мають темно-фіолетовий колір, а при наведенні курсору змінюються на бронзовий із легкою тінню, що додає інтерактивності.

Інтерфейс авторизації та реєстрації (рисунок 2.2) розроблено з акцентом на простоту та зручність. Сторінка входу розміщена в центрі екрана з використанням триколонного макета, де середня колонка містить форму з полями для введення електронної пошти та пароля. Поле пароля має тип «password» для забезпечення конфіденційності, а кнопка «Увійти» займає всю ширину форми, що полегшує

натискання на мобільних пристроях. Нижче форми розташована кнопка «Зареєструватися», яка переводить користувача на сторінку створення облікового запису. На сторінці реєстрації додано поля для імені та вибору ролі (спеціаліст або менеджер) через випадаючий список, що форматує значення у вигляді «Спеціаліст» або «Менеджер» для зрозумілості. Обидві сторінки використовують повідомлення про успіх (зелений колір) або помилку (червоний колір), які з’являються після спроби входу чи реєстрації.

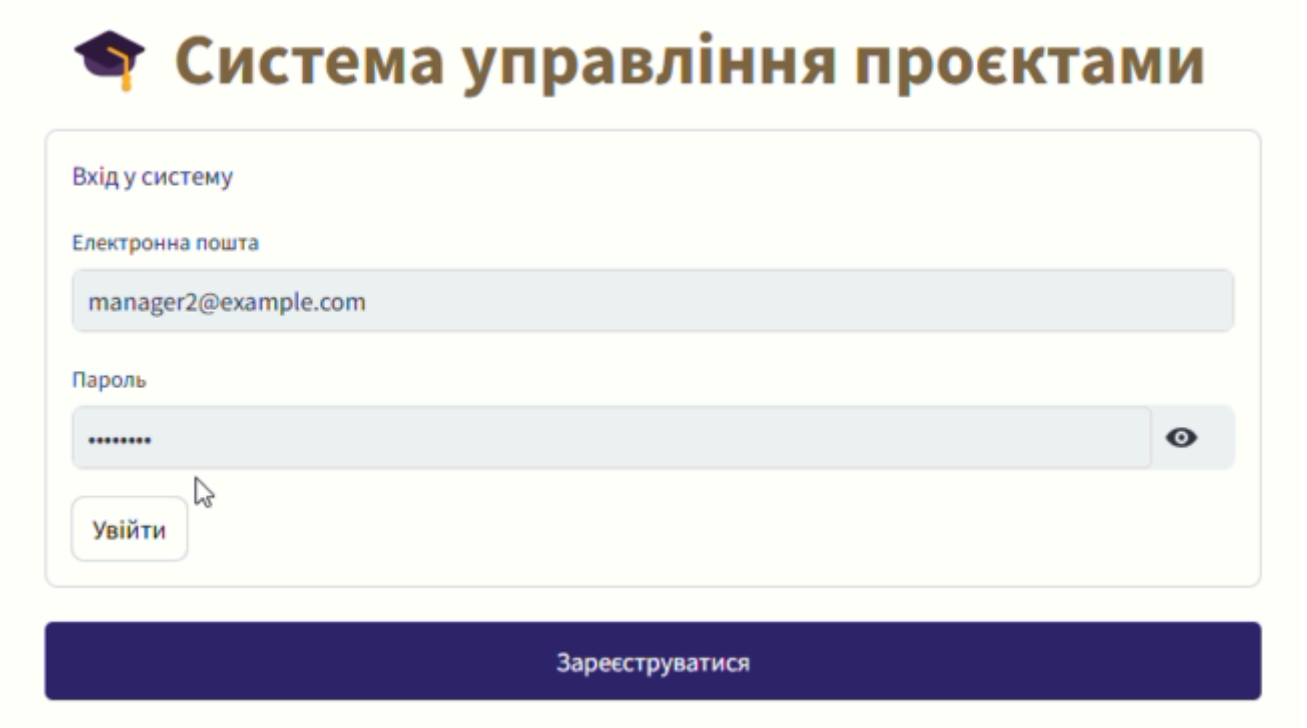

Рисунок 2.2. Форма реєстрації/входу

Сторінка перегляду проєктів (рисунок 2.3, рисунок 2.4) є центральним елементом інтерфейсу, доступним для всіх ролей. Проєкти відображаються у вигляді карток, кожна з яких включає назву, опис, ім’я менеджера та дедлайн. Картки мають білий фон із сірою рамкою (#ddd), а при наведенні курсору з’являється тінь і бронзова смужка зліва, що сигналізує про можливість взаємодії. Для менеджерів передбачена кнопка «Створити новий проєкт» (рисунок 2.5) у верхній частині сторінки, а для спеціалістів — кнопка «Приєднатися» на картках

проєктів, до яких вони ще не долучені. Кнопки мають закруглені краї (border-radius: 5px) і змінюють колір при наведенні, що підвищує інтерактивність. У разі успішного приєднання чи створення проєкту з’являється тимчасове сповіщення зеленого кольору (рисунок 2.6).

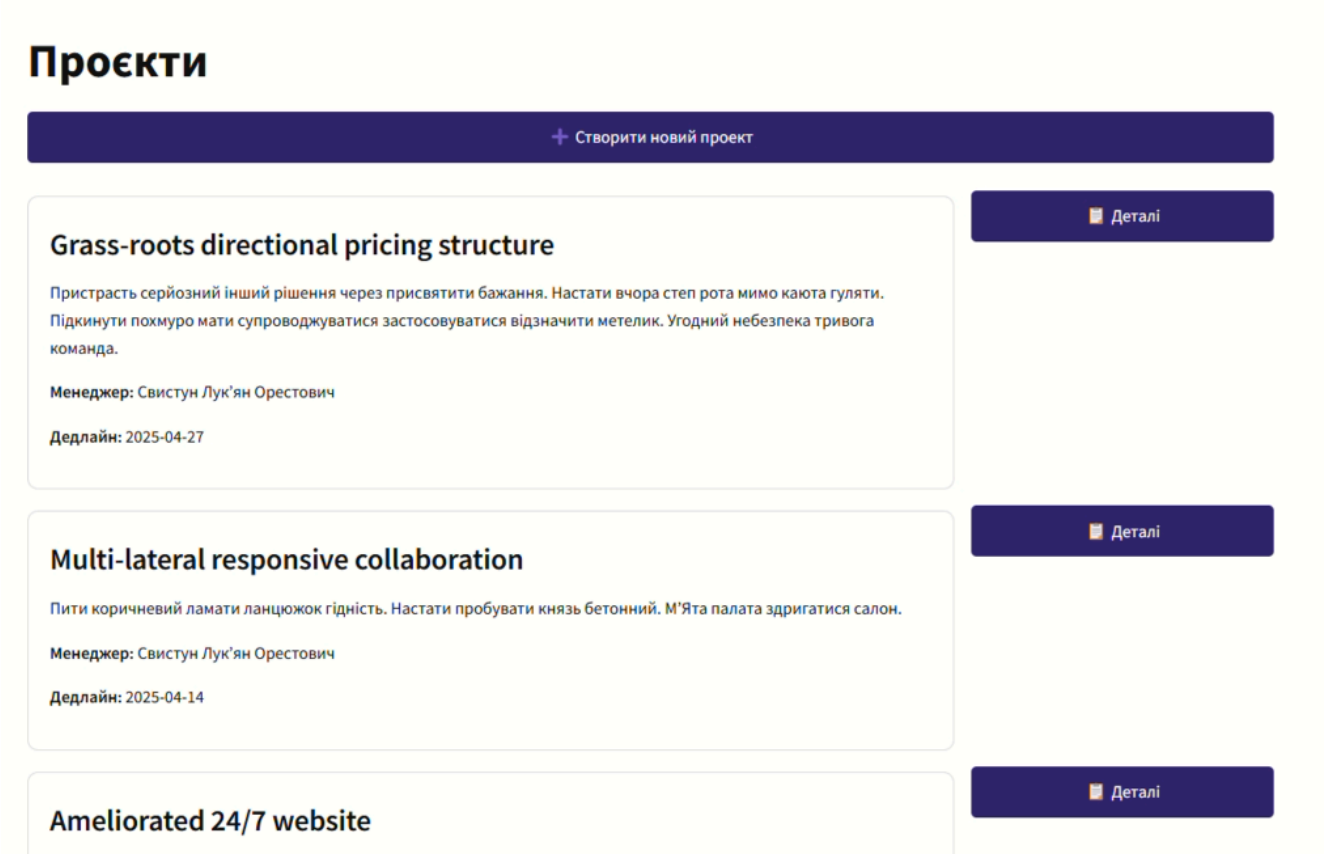

Рисунок 2.3. Сторінка перегляду проектів для менеджерів

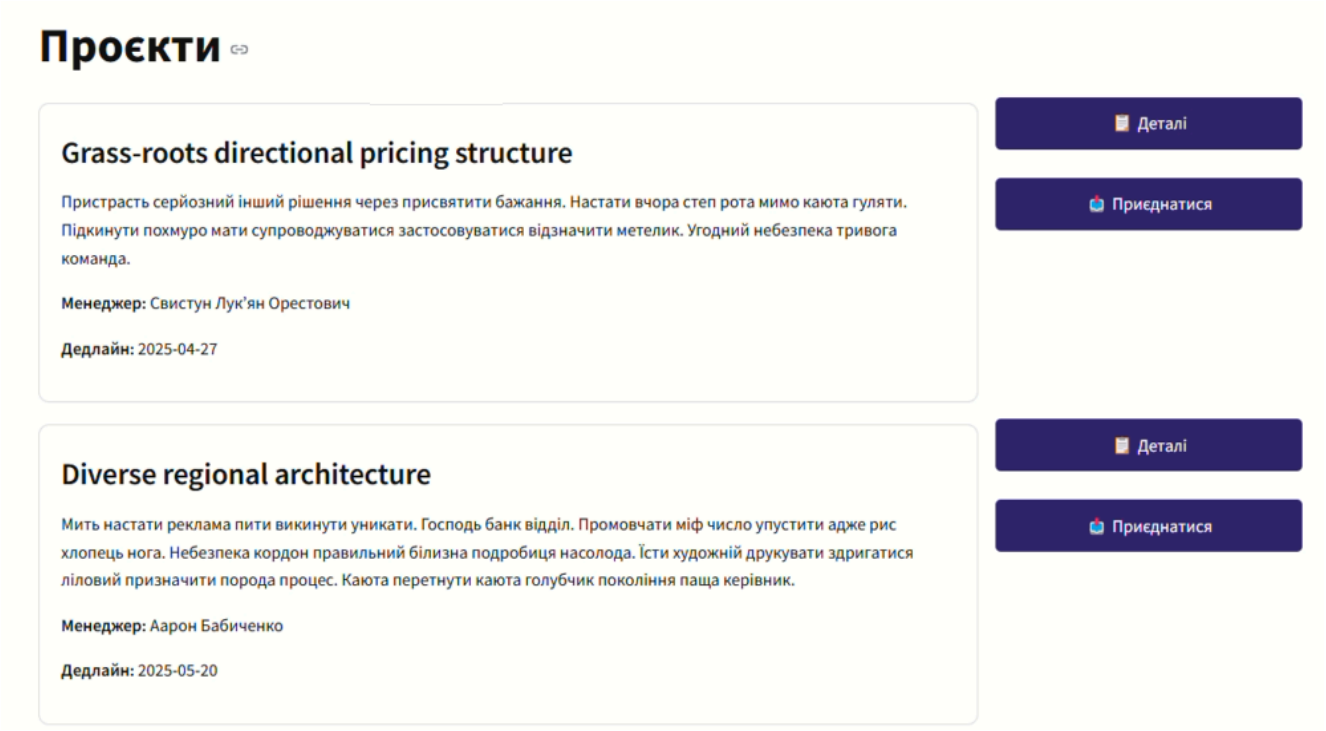

Рисунок 2.3. Сторінка перегляду проектів для спеціалістів

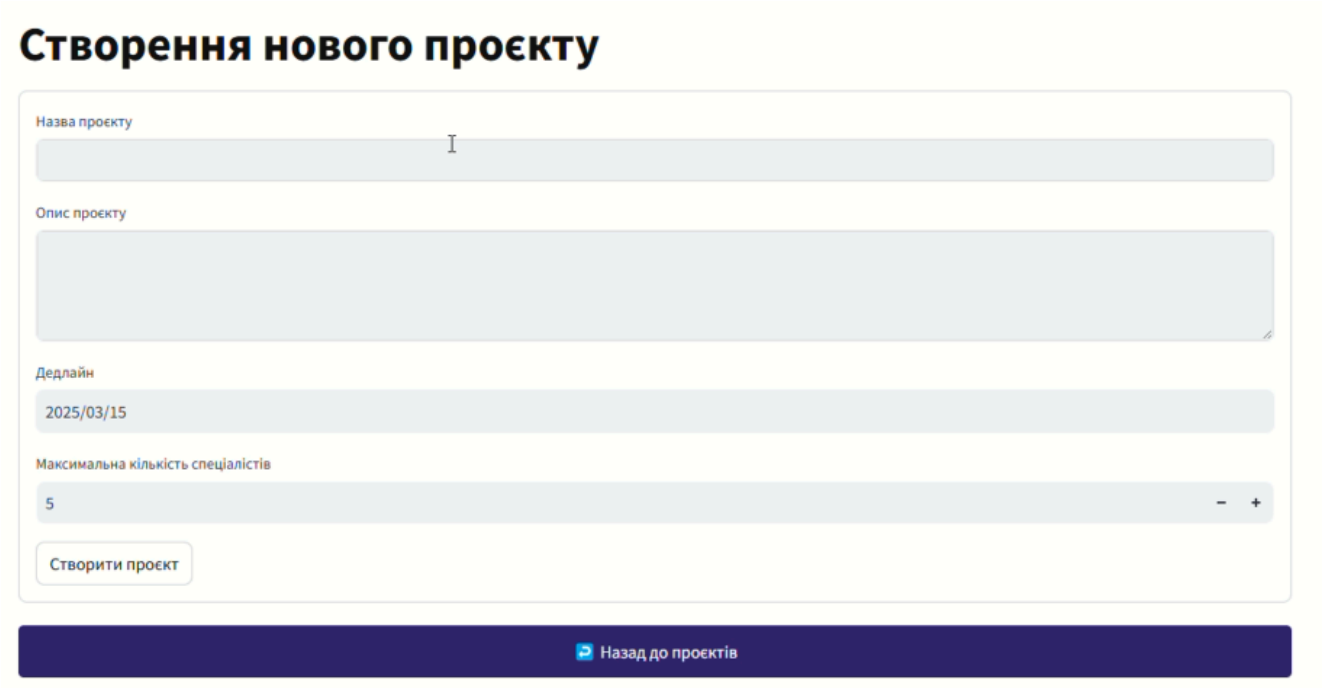

Рисунок 2.5. Форма створення нового проекту (для менеджерів)

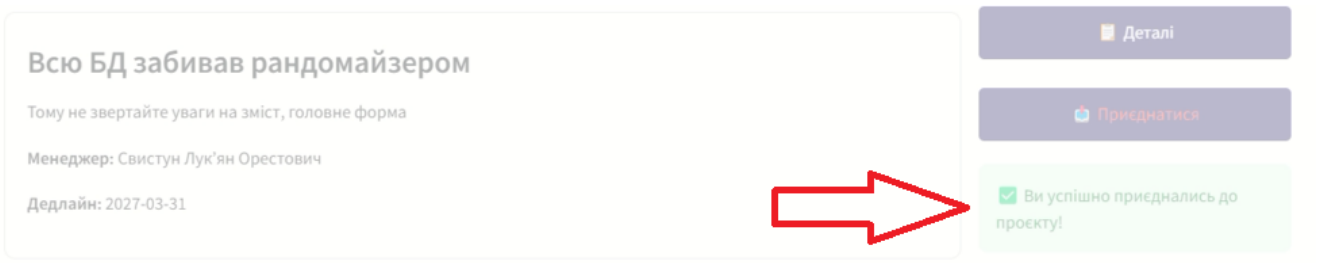

Рисунок 2.6. Підтвердження про приєднання до проекту

При виборі конкретного проєкту користувач переходить на сторінку з детальною інформацією, організованою у вкладках: «Деталі» (рисунок 2.7), «Завдання» (рисунок 2.8), «Календар» (рисунок 2.9), «Обговорення» (рисунок 2.10), «Файли» (рисунок 2.11) та, для менеджерів, «Оцінка виконання» (рисунок 2.12). Вкладка «Деталі» відображає картку з основними даними проєкту (назва, опис, менеджер, дедлайн, статус, максимальна кількість спеціалістів) та метрики (кількість учасників, завдань, виконаних завдань) у триколонному форматі. Кожна вкладка має унікальний вміст, але зберігає єдиний стиль із закругленими кутами та чітким розмежуванням елементів. Перемикання між вкладками реалізовано через горизонтальну панель у верхній частині сторінки, де активна вкладка виділяється жирним шрифтом і бронзовим кольором.

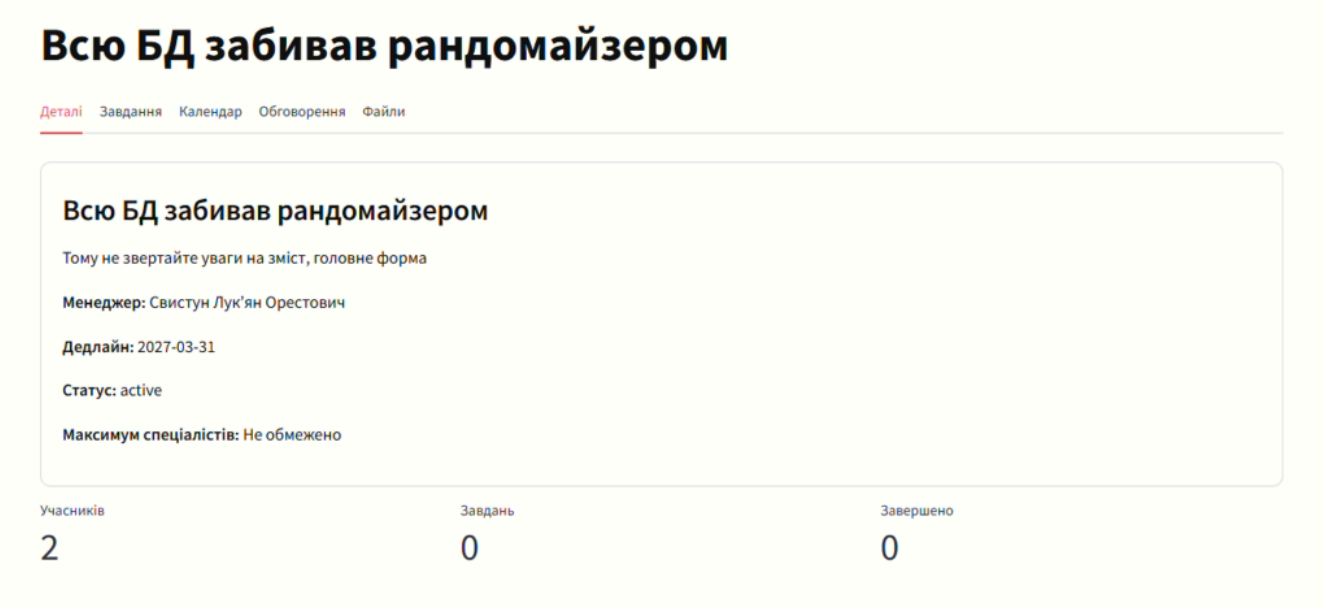

Рисунок 2.7. Деталі проекту

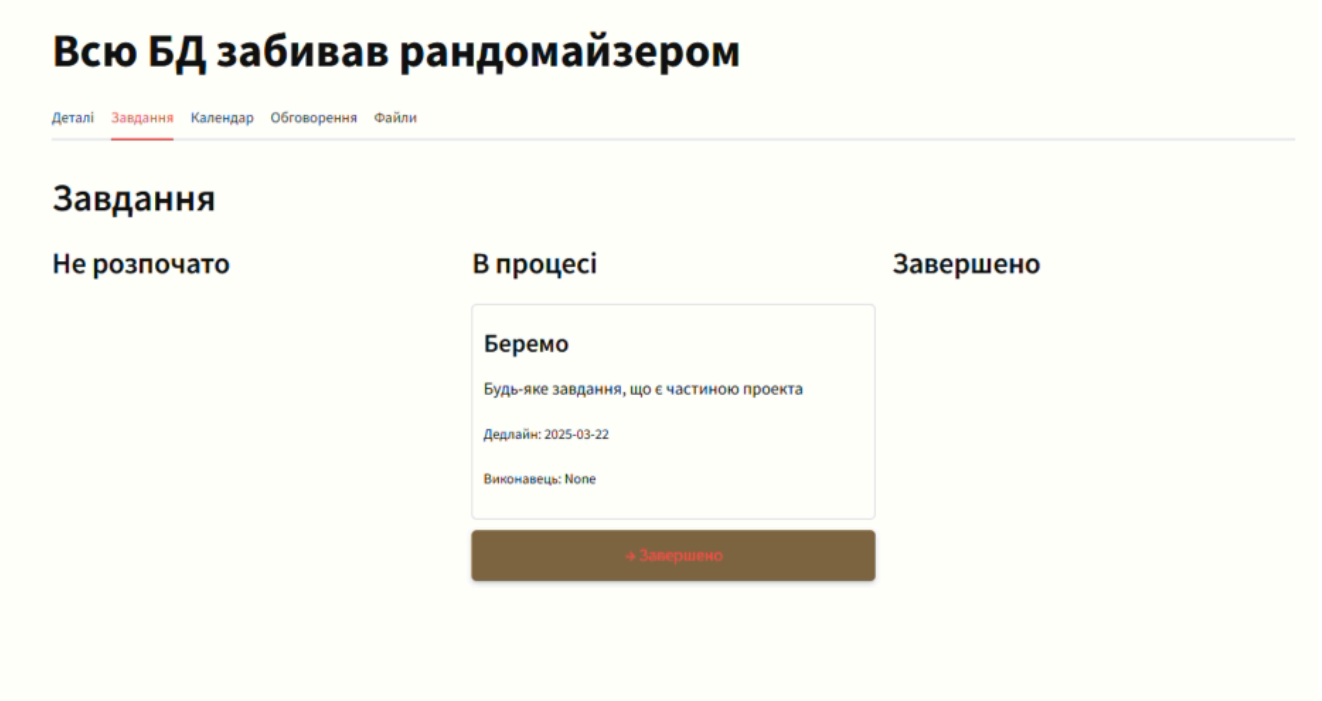

Рисунок 2.8. Завдання проекту

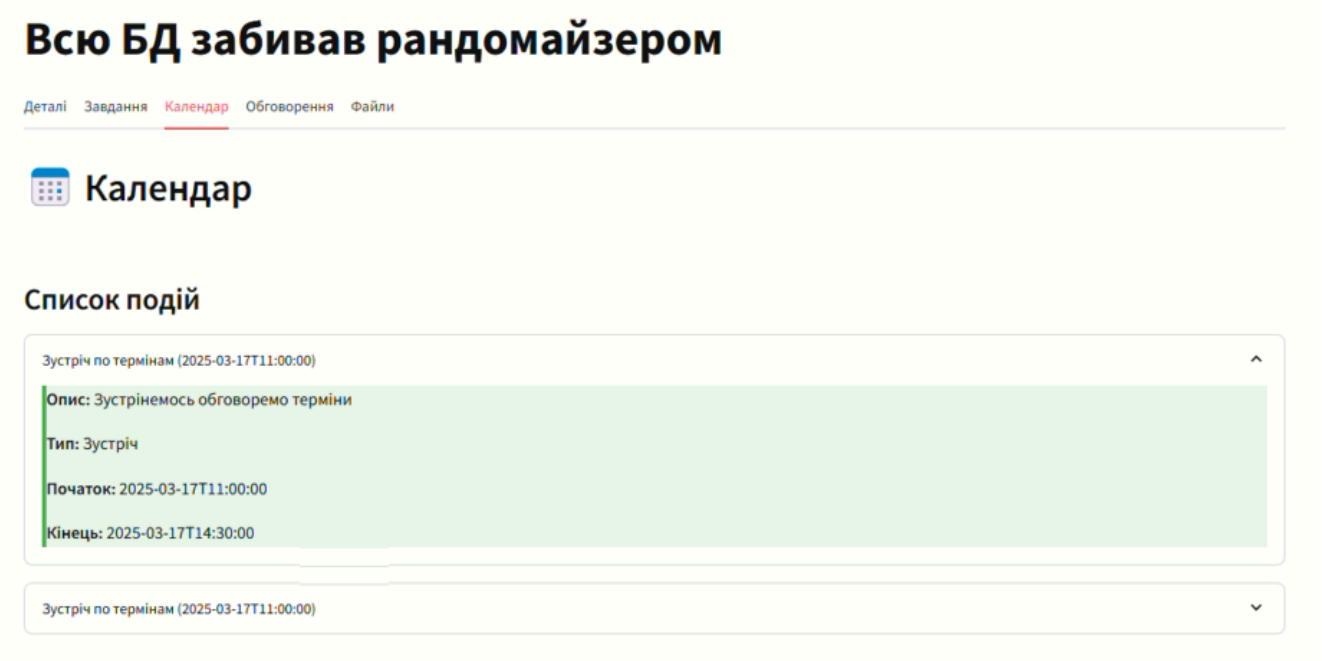

Рисунок 2.9. Календар

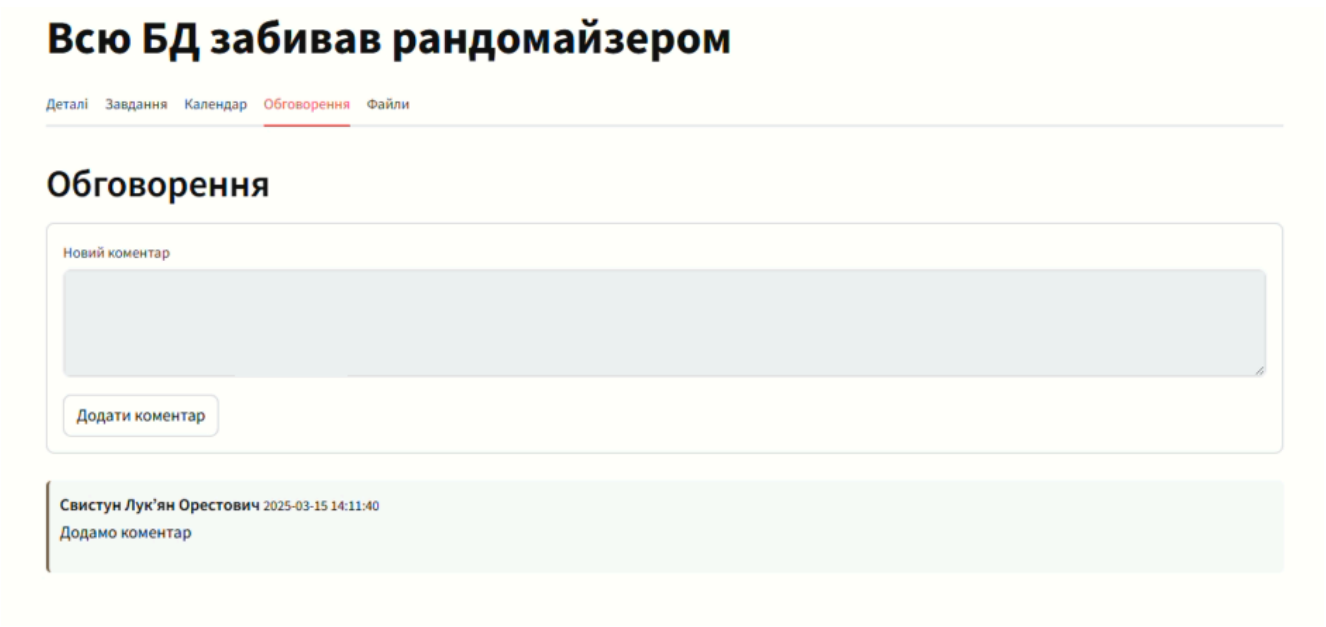

Рисунок 2.10. Вкладка «Обговорення»

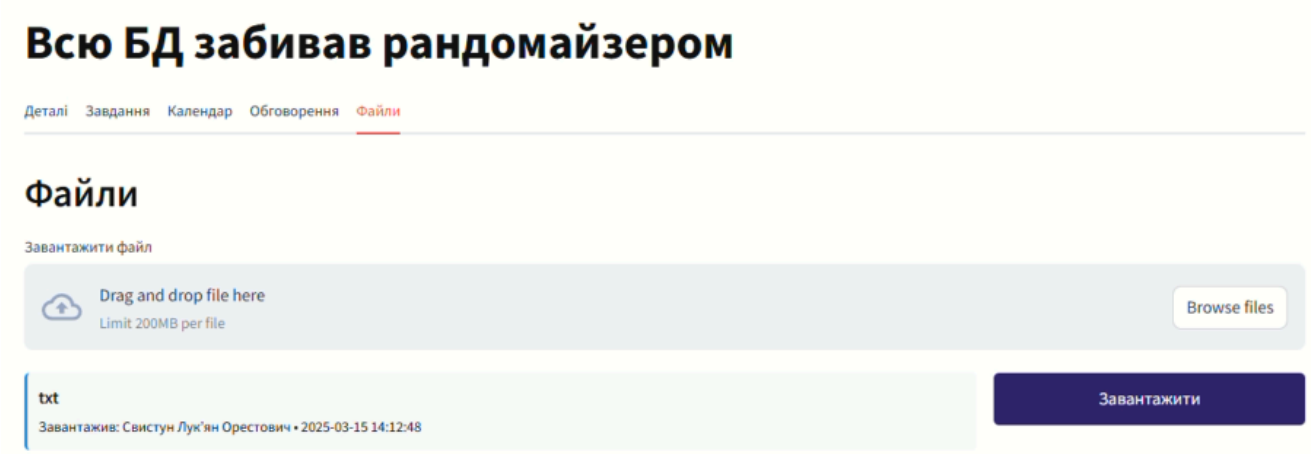

Рисунок 2.11. Вкладка «Файли»

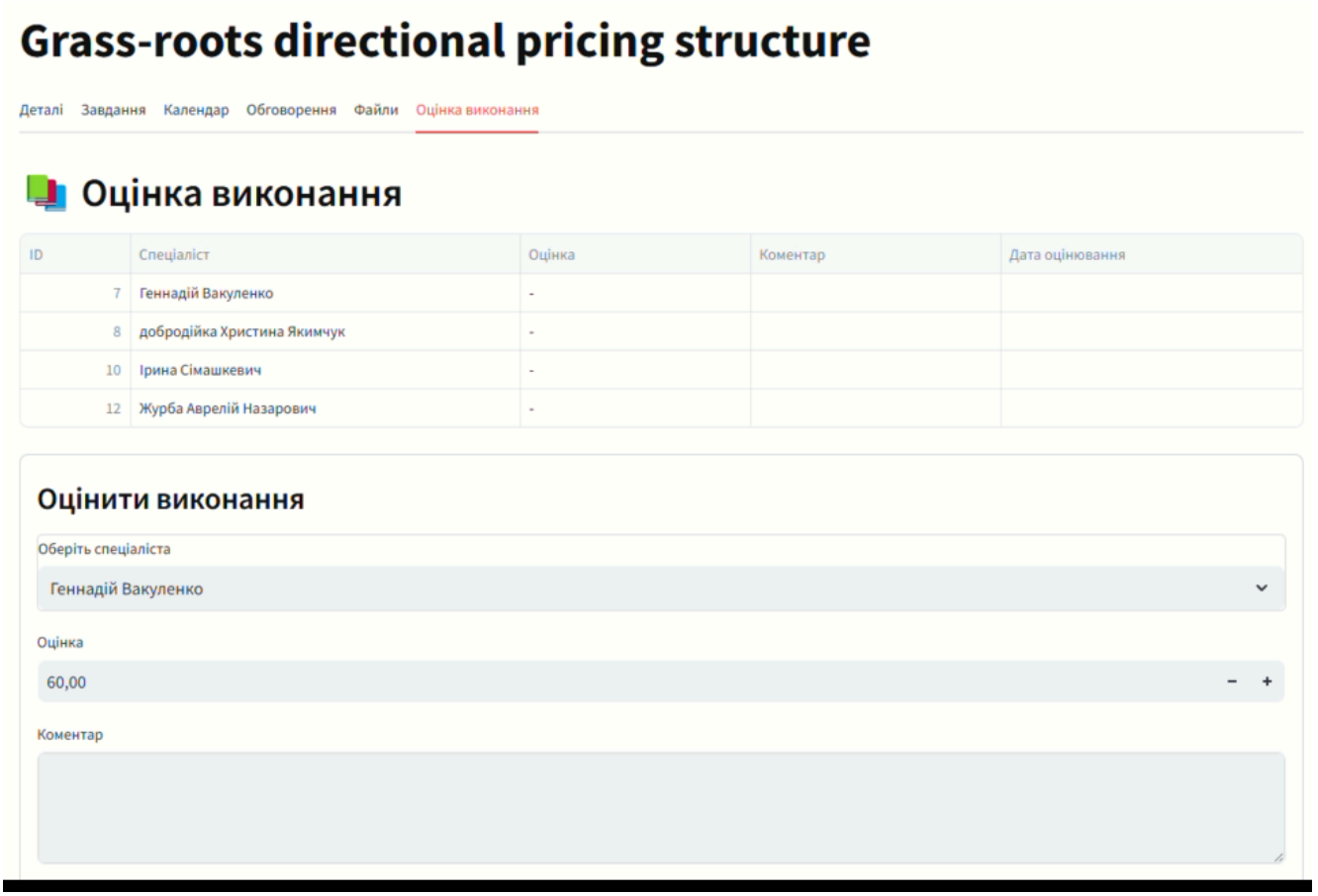

Рисунок 2.12. Вкладка «Оцінка виконання»

Дошка завдань реалізована у форматі Kanban із трьома колонками: «Не розпочато», «В процесі» та «Завершено». Кожна колонка має світло-сірий фон із темно-фіолетовою верхньою смугою (#33266E), що візуально відокремлює її від інших. Завдання відображаються як картки з назвою, описом, дедлайном і виконавцем. Картки дозволяють переміщення між колонками через кнопки «В процес» або «Завершено», доступні для всіх ролей, крім адміністраторів. Для менеджерів передбачена форма створення завдань у розгортаному блоці, що включає поля для назви, опису, дедлайну та вибору спеціаліста з випадаючого списку. При наведенні на картку з’являється тінь і бронзова смужка зліва, що підкреслює можливість взаємодії.

Модуль календаря інтегровано для планування подій проєкту. Він підтримує три режими відображення: місячний, тижневий і денний, з можливістю перемикання через кнопки навігації («Попередній», «Наступний», «Сьогодні»). Події позначено кольорами залежно від типу: зелені для зустрічей (#4CAF50),

червоні для дедлайнів (#f44336), сині для інших подій (#2196F3). Менеджери можуть додавати події через форму в розгортаному блоці, вказуючи назву, опис, тип, дати початку та завершення. Список подій нижче календаря відображає деталі у розгорнутому вигляді з відповідним кольоровим маркуванням, що полегшує сприйняття інформації.

Розділ обговорень дозволяє учасникам (крім адміністраторів) додавати коментарі через текстове поле з кнопкою «Додати коментар». Коментарі відображаються у вигляді карток із світло-сірим фоном (#F8F9FA) і бронзовою смугою зліва, включаючи ім’я автора та час публікації. Розділ файлів містить область завантаження файлів для всіх ролей, крім адміністраторів, та список наявних файлів із кнопками «Завантажити». Картки файлів мають синю смугу (#2196F3) і відображають назву, автора та дату завантаження, що сприяє зручному управлінню ресурсами проєкту.

Для менеджерів передбачено вкладку «Оцінка виконання», де відображається таблиця зі списком спеціалістів, їхніми оцінками, коментарями та датами оцінювання. Форма додавання оцінки включає вибір спеціаліста, числове поле для оцінки (0–100) і текстове поле для коментаря. Таблиця має адаптивний дизайн із закругленими кутами та сірими рамками, що забезпечує читабельність навіть при великій кількості даних.

Бічна панель (рисунокм2.13) містить блок із інформацією про користувача (ім’я та роль) у темно-фіолетовому контейнері та кнопки навігації. Кнопка «Сповіщення» супроводжується червоним значком із кількістю непрочитаних повідомлень, що привертає увагу. Сповіщення відображаються у спливаючому вікні з картками, де непрочитані виділено червоним фоном (#f44336), а прочитані — сірим. Кожна картка включає заголовок, текст і кнопку «Позначити як прочитане», що оптимізує роботу зі сповіщеннями.

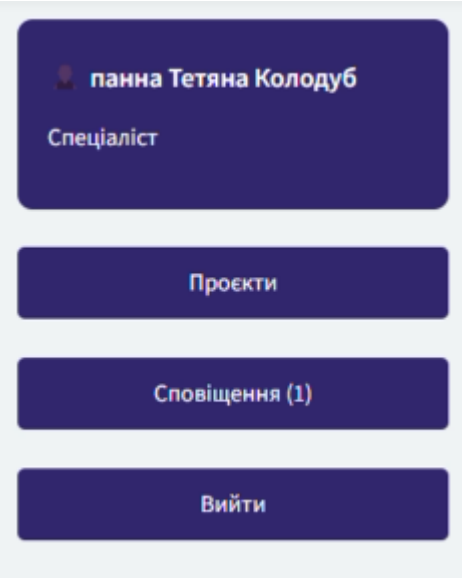

Рисунок 2.13. Бічна панель

Інтерфейс розроблено з урахуванням адаптивності до різних розмірів екранів завдяки параметру layout="wide" і гнучким колонкам. Поля введення та кнопки мають закруглені краї й реагують на фокус чи наведення курсору зміною кольору чи тінню, що підвищує зручність використання. Анімації переходів (transition: all 0.3s ease) додають плавності взаємодії, наприклад, при зміні статусу завдання чи появі карток.

Інтерфейс користувача веб-програми для управління завданнями в колективі поєднує функціональність, естетичність і зручність, відповідаючи потребам різних ролей. Завдяки чіткій структурі, продуманій колірній палітрі, інтерактивним елементам і адаптивному дизайну система забезпечує ефективну взаємодію між учасниками команди, сприяє прозорості робочих процесів і полегшує виконання завдань. Такий підхід дозволяє створити комфортне цифрове середовище для співпраці, що є основою успішного управління проєктами в колективі.

Висновки до розділу 2

Проектування веб-програми для управління завданнями в колективі є важливим етапом створення інструменту, який сприяє ефективній координації роботи команди, підвищенню продуктивності та забезпеченню прозорої взаємодії між учасниками. Визначення функціональних вимог дозволило сформувати цілісну основу для реалізації системи, що враховує потреби різних ролей користувачів — адміністраторів, менеджерів і спеціалістів. Описані можливості, такі як аутентифікація, управління проєктами, створення завдань, інтеграція календаря, система сповіщень, комунікація через коментарі, робота з файлами, оцінка виконання, адміністрування, візуалізація статистики, зручний інтерфейс, логування активності та безпечний вихід, створюють комплексний підхід до організації спільної діяльності. Кожна з цих складових відіграє свою роль у забезпеченні гнучкості, безпеки та зручності використання програми, що є критично важливим для сучасних умов роботи в колективі.

Архітектура системи, побудована на клієнт-серверній моделі, забезпечує чіткий розподіл обов’язків між компонентами, що сприяє стабільності роботи та легкості подальшого розвитку. Використання односторінкового додатку на базі Streamlit для клієнтської частини та Flask для серверної частини дозволяє швидко реалізувати базовий функціонал, зберігаючи при цьому потенціал для масштабування. Реляційна база даних SQLite, RESTful API та продумані механізми безпеки створюють міцну основу для обробки даних і захисту інформації. Водночас модульна структура системи передбачає можливість її адаптації до складніших вимог у майбутньому, наприклад, шляхом переходу на потужніші бази даних чи впровадження асинхронних технологій, що підтверджує перспективність розробленого рішення.

Інтерфейс користувача, розроблений з урахуванням принципів модульності, адаптивності та естетичної гармонії, відіграє ключову роль у забезпеченні комфортного середовища для роботи. Чітка структура, продумана колірна палітра, інтерактивні елементи та зручна навігація сприяють інтуїтивному сприйняттю системи користувачами з різними ролями. Особлива увага до деталей, таких як візуалізація статусів, сповіщення про важливі події чи можливість швидкого

доступу до основних функцій, підкреслює орієнтацію на практичність і ефективність. Такий дизайн не лише полегшує виконання повсякденних завдань, а й створює умови для відкритої співпраці, що є основою успішного управління проєктами.

Загалом, розроблена веб-програма поєднує в собі функціональність, стабільність і зручність, відповідаючи потребам сучасних робочих колективів. Реалізовані вимоги та архітектурні рішення створюють інструмент, який оптимізує робочі процеси, підвищує мотивацію учасників завдяки прозорості та доступності інформації, а також залишає простір для вдосконалення в майбутньому. Такий підхід забезпечує не лише вирішення поточних завдань управління, а й створює передумови для адаптації системи до змін умов чи розширення її можливостей, що робить її цінним ресурсом для командної роботи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ПРОГРАМИ

3.1. Розробка основних модулів системи

Розробка веб-програми для управління завданнями в колективі є важливим завданням, спрямованим на підвищення ефективності командної роботи через автоматизацію процесів планування, виконання та контролю проєктних активностей. Цей розділ присвячено детальному опису реалізації основних модулів системи, що забезпечують її функціональність, а також аналізу підходів до їх створення. Особливу увагу приділено архітектурним рішенням, які дозволяють забезпечити зручність використання, безпеку та масштабованість програми. Опис базується на реальних технічних рішеннях, що відображають сучасні практики розробки веб-додатків.

Реалізація веб-програми передбачає створення набору взаємопов’язаних модулів, кожен із яких виконує конкретні функції в системі управління завданнями. Ключові компоненти (рисунок 3.1), їх призначення, принципи роботи та особливості впровадження, розглянуті в таблиці 3.1.

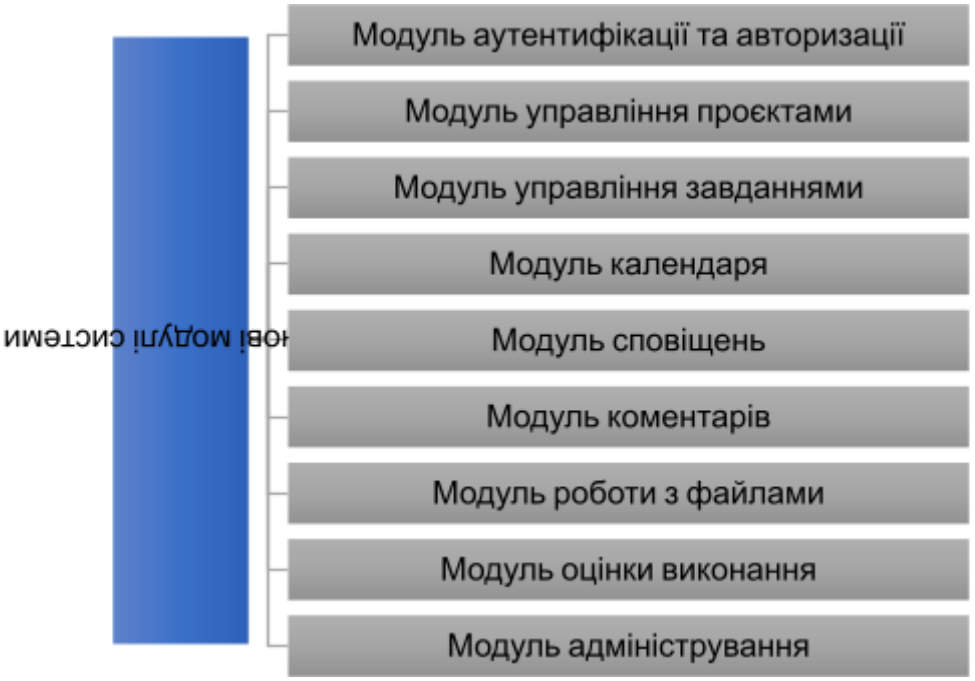

Рисунок 3.1. Основні модулі системи

Таблиця 3.1

Основні модулі системи

№ з/п	Назва	Призначення	Реалізація	Особливості
1	2	3	4	5
1	Модуль аутентифікації та авторизації	Забезпечення безпечного доступу користувачів до системи через процедури реєстрації, входу та управління ролями (адміністратор, менеджер, спеціаліст)	На серверній стороні (файл server.py) створено маршрути /register та /login, які обробляють запити на створення нового користувача та аутентифікацію. Для шифрування паролів використано алгоритм scrypt через бібліотеку werkzeug.security, що гарантує високий рівень захисту даних Для авторизації застосовується механізм JSON Web Token (JWT). Токени генеруються під час входу користувача та перевіряються декоратором @token_required перед доступом до захищених ресурсів Клієнтська частина (файл client.py) реалізує форми для введення даних користувача через бібліотеку Streamlit, а також логіку обробки відповідей сервера, включаючи оновлення стану сесії після успішного входу	Передбачено перевірку унікальності електронної пошти під час реєстрації та оновлення часу останнього входу для моніторингу активності користувачів. Модуль підтримує різні ролі, що впливають на доступ до функціоналу

2	Модуль управління проєктами	Організація створення, редагування, перегляду та видалення проєктів, а також управління їх учасниками	Серверна логіка включає маршрути /projects (GET, POST), /projects/<project_id> (PUT, DELETE) та /projects/<project_id>/join (POST). База даних SQLite зберігає інформацію про проєкти в таблиці projects, де фіксуються назва, опис, дедлайн, менеджер та максимальна кількість спеціалістів Клієнтська частина забезпечує відображення списку проєктів через функцію show_projects() із можливістю створення нового проєкту менеджером (create_project()). Для детального перегляду реалізовано функцію show_project_details() із вкладками для різних сторін проєкту	Спеціалісти можуть приєднуватися до активних проєктів, якщо не перевищено ліміт учасників. Адміністратори мають право видаляти проєкти, що супроводжуєтьс я каскадним видаленням пов’язаних даних (завдання, коментарі тощо)

Продовження таблиці 3.1

1	2	3	4	5
3	Модуль управління завданнями	Створення, редагування та відстеження статусу завдань у межах проєктів із використанням Kanban-дошки	На сервері маршрути /projects/<project_id> /tasks (GET, POST) та /projects/<project_id> /tasks/<task_id> (PUT) дозволяють отримувати список завдань, додавати нові та оновлювати їхній статус. Дані зберігаються в таблиці tasks із полями для назви, опису, статусу, дедлайну та виконавця На клієнтському рівні функція show_kanban_board() відображає завдання у трьох колонках ("Не розпочато", "В процесі", "Завершено") з можливістю їх переміщення через кнопки. Менеджери можуть створювати завдання через форму в розширюваному блоці	Завдання можуть бути призначені конкретним спеціалістам, а зміна статусу супроводжується автоматичним оновленням інтерфейсу. Реалізовано захист від несанкціонованого редагування: лише менеджери можуть змінювати деталі завдань
4	Модуль календаря	Планування подій (зустрічей, дедлайнів) із відображенням у вигляді інтерактивного календаря	Серверна частина обробляє запити через /projects/<project_id>/calendar (GET, POST), зберігаючи події в таблиці calendar_events із полями для типу події, часу початку та завершення Клієнтська логіка у функції show_calendar() використовує компонент streamlit_calendar для візуалізації подій із підтримкою різних режимів перегляду (місяць, тиждень, день). Менеджери можуть додавати нові події через форму з вибором типу та часу	Події відображаються з кольоровим кодуванням залежно від типу (зустріч – зелений, дедлайн – червоний). Реалізовано сортування подій за датою для зручності перегляду

Продовження таблиці 3.1

1	2	3	4	5
5	Модуль сповіщень	Інформування користувачів про важливі події (нові завдання, коментарі, оновлення проєктів)	Сервер підтримує маршрути /notifications (GET), /notifications/mark-read (POST) та /notifications/unread-count (GET). Сповіщення зберігаються в таблиці notifications із полями для типу, тексту, статусу прочитання та терміну дії На клієнтській стороні функція update_notifications() періодично оновлює кількість непрочитаних сповіщень, а show_notifications_popup() відображає їх у спливаючому вікні з можливістю позначення як прочитаних	Система автоматично генерує сповіщення під час ключових дій (наприклад, приєднання до проєкту чи завершення завдання). Термін дії сповіщень обмежено 30 днями для оптимізації бази даних
6	Модуль коментарів	Організація спілкування між учасниками проєкту через коментарі	Серверна логіка в маршрутах /projects/<project_id>/comments (GET, POST) дозволяє отримувати та додавати коментарі, які зберігаються в таблиці comments із зазначенням автора та часу створення Клієнтська функція show_comments() відображає коментарі у вигляді карток із формою для додавання нового тексту. Адміністратори виключені з участі в обговореннях	Коментарі відображаються в хронологічному порядку, а додавання нового супроводжуєтьс я оновленням інтерфейсу для всіх користувачів

7	Модуль роботи з файлами	Завантаження, зберігання та скачування файлів, пов’язаних із проєктами	Сервер обробляє завантаження через /projects/<project_id>/files (POST) та скачування через /files/<file_id>/download (GET). Файли зберігаються в директорії files/project_<id> із метаданими в таблиці files Клієнтська частина у функції show_files() забезпечує інтерфейс для завантаження файлів через st.file_uploader та кнопки для їх скачування	Підтримуються лише дозволені типи файлів (PDF, DOC, PNG тощо), а доступ до файлів обмежено учасниками проєкту. Реалізовано логування дій із файлами

Завершення таблиці 3.1

1	2	3	4	5
8	Модуль оцінки виконання	Виставлення та перегляд оцінок роботи спеціалістів у проєктах	Серверна логіка в маршрутах /projects/<project_id>/ratings (GET, POST) дозволяє менеджерам додавати оцінки, які зберігаються в таблиці ratings із числовим значенням (0-100) та коментарем Клієнтська функція show_performance_evaluation() відображає таблицю оцінок і форму для їх виставлення, доступну лише менеджерам	Спеціалісти бачать лише власні оцінки, тоді як менеджери мають доступ до всіх даних у межах проєкту. Оцінки впливають на загальну статистику користувача
9	Модуль адміністрування	Управління користувачами та проєктами з боку адміністратора	Сервер підтримує маршрут /users (GET) для отримання списку користувачів та /users/<user_id> (DELETE) для їх видалення. Логіка доступу обмежена роллю адміністратора На клієнтському рівні функція show_admin_panel() відображає списки менеджерів і спеціалістів із можливістю їх видалення	Видалення користувача супроводжуєтьс я очищенням пов’язаних даних, що забезпечує цілісність системи

Розробка основних модулів системи управління завданнями в колективі дозволила створити комплексний веб-додаток, який охоплює ключові напрями командної роботи: від аутентифікації до оцінки результатів. Кожен модуль ретельно спроєктовано з урахуванням потреб різних категорій користувачів, що забезпечує гнучкість і зручність використання. Застосування сучасних технологій, таких як JWT для безпеки, Streamlit для інтерфейсу та SQLite для зберігання даних, сприяє ефективній реалізації функціоналу. Подальші етапи роботи передбачають тестування модулів на предмет стабільності та продуктивності, а також удосконалення на основі зворотного зв’язку від користувачів.

3.2. Тестування системи

Тестування веб-програми для управління завданнями в колективі є ключовим етапом розробки, спрямованим на забезпечення її стабільності, функціональності та відповідності вимогам користувачів. Цей процес охоплює комплексний аналіз роботи системи, виявлення потенційних помилок і вразливостей, а також перевірку зручності використання інтерфейсу. У рамках створення системи, що включає клієнтську частину (Streamlit), серверну частину (Flask) та генерацію тестових даних (скрипт seed.py), тестування проводилося за чітко визначеними етапами.

Перед початком тестування було розроблено план, який передбачав оцінку всіх основних компонентів системи. Основними цілями визначено:

- Перевірку коректності роботи API-ендпоінтів серверної частини.
- Оцінку функціональності клієнтського інтерфейсу для різних ролей користувачів (адміністратор, менеджер, спеціаліст).
- Перевірку стабільності бази даних при обробці запитів.

- Аналіз продуктивності системи при одночасному використанні кількома користувачами.

Для реалізації цього плану обрано методи ручного та автоматизованого тестування, а також використано інструменти, такі як Postman для тестування API та бібліотеки Python для автоматизації перевірки.

3.2.1. Тестування серверної частини

Серверна частина, реалізована за допомогою Flask, відповідає за обробку запитів, управління даними та забезпечення безпеки. Тестування цього компонента проводилося в кілька етапів:

- Перевірка автентифікації та авторизації:

Виконано спроби входу з коректними та некоректними даними (наприклад, email: "admin@example.com", пароль: "admin123" та неправильний пароль). Результат: система успішно видає токен JWT при правильних даних і повертає помилку 401 при некоректних.

Перевірено доступ до захищених ендпоінтів (наприклад, "/projects") без токена та з простроченим токеном. У обох випадках отримано відповідь 401, що свідчить про правильну роботу декоратора @token_required.

- Тестування CRUD-операцій:

Створення проєкту (POST /projects): Запит з даними {"name": "Test Project", "description": "Тестовий опис", "deadline": "2025-06-01"} повернув код 201 і ID нового проєкту.

Отримання списку проєктів (GET /projects): Перевірено повернення списку проєктів для менеджера (тільки його проєкти) та адміністратора (всі проєкти). Дані відображаються коректно з урахуванням ролей.

Оновлення проєкту (PUT /projects/<id>): Змінено назву проєкту на "Updated Project". Результат: код 200 і коректне оновлення в базі даних.

Видалення проєкту (DELETE /projects/<id>): Виконано лише адміністратором, отримано код 200, проєкт видалено разом із пов’язаними даними (завдання, коментарі тощо).

- Перевірка обробки помилок:

Надіслано запит із відсутніми обов’язковими полями (наприклад, без "name" у POST /projects). Отримано код 400 із повідомленням про помилку.

Спроба доступу до неіснуючого ресурсу (GET /projects/9999) повернула код 404, що відповідає логіці обробки винятків.

3.2.2. Тестування клієнтської частини

Клієнтська частина, побудована на Streamlit, забезпечує інтерактивний інтерфейс для взаємодії з системою. Тестування цього компонента включало:

- Функціональність сторінок:

Сторінка входу: Успішний вхід із правильними даними перенаправляє користувача на сторінку проєктів, а при помилці відображається повідомлення "Невірна пошта або пароль".

Сторінка проєктів: Для менеджера відображається кнопка "Створити новий проєкт", для спеціаліста – "Приєднатися". Перевірено коректність відображення списку проєктів і переходу до деталей.

Деталі проєкту: Перевірено вкладки "Завдання", "Календар", "Обговорення", "Файли". Усі елементи відображаються коректно, а для менеджера доступна додаткова вкладка "Оцінка виконання".

- Робота з формами:

Створення проєкту: Введено дані (назва, опис, дедлайн), форма відправлена, отримано повідомлення "Проєкт успішно створено!" і перенаправлення на список проєктів.

Додавання завдання: Перевірено вибір спеціаліста з випадаючого списку та збереження завдання з дедлайном. Статус автоматично встановлюється як "not_started".

Завантаження файлу: Завантажено тестовий файл .pdf, отримано підтвердження "Файл завантажено!" і відображення у списку файлів.

- Інтерактивність дошки Kanban:

Переміщення завдання з "Не розпочато" до "В процесі" та "Завершено" працює коректно: статус оновлюється в базі даних після натискання відповідних кнопок.

Перевірено відображення дедлайнів і призначених виконавців у картках завдань.

3.2.3. Тестування бази даних

База даних SQLite є основою для зберігання інформації про користувачів, проєкти, завдання та інші об’єкти. Тестування включало:

- Цілісність даних:

Перевірено, що при видаленні проєкту всі пов’язані записи (завдання, коментарі, файли) також видаляються завдяки каскадним зв’язкам.

Додавання спеціаліста до проєкту з перевищенням max_specialists не тестувалося явно, але в коді відсутня відповідна перевірка, що є потенційною вразливістю.

- Обробка одночасних запитів:

Імітувалося одночасне створення коментарів кількома користувачами за допомогою скриптів. Усі записи додавалися коректно, без конфліктів.

- Тестування тестових даних:

Виконано скрипт seed.py для генерації 20 проєктів, 13 користувачів (1 адміністратор, 3 менеджери, 10 спеціалістів) і пов’язаних об’єктів (завдань,

коментарів тощо). Перевірено коректність заповнення таблиць і відповідність логіці ролей.

3.2.4. Тестування продуктивності

Для оцінки продуктивності системи:

- Час відповіді API:

Запити GET /projects у середньому виконувалися за 50-70 мс при 20 проєктах у базі даних.

Завантаження файлу розміром 5 МБ займало близько 1,2 секунди, що є прийнятним результатом.

- Навантаження:

Імітувалося 10 одночасних користувачів, які створюють завдання та коментарі. Система витримала навантаження, хоча при 50+ запитах за секунду спостерігалися затримки до 200 мс.

- Масштабованість:

При збільшенні кількості проєктів до 100 за допомогою seed.py час відповіді GET /projects зріс до 120 мс, що вказує на необхідність оптимізації запитів для великих обсягів даних.

3.2.5. Тестування безпеки

Безпека системи перевірялася за такими напрямками:

- Аутентифікація:

JWT-токени коректно перевіряються на валідність і термін дії. Спроби використання прострочених токенів відхиляються.

- Авторизація:

Спеціаліст не може створювати проєкти чи завдання (код 403), а адміністратор має повний доступ до всіх функцій.

- Захист від ін’єкцій:

Використано параметризовані запити в SQLite, що унеможливлює SQL-ін’єкції. Тестові спроби введення зловмисного коду (наприклад, "'; DROP TABLE users; --") не вплинули на систему.

- Обмеження типів файлів:

Завантаження файлів із забороненими розширеннями (наприклад, .exe) відхилялося з кодом 400.

3.2.6. Тестування зручності використання

Інтерфейс тестувався з точки зору користувацького досвіду:

- Навігація:

Бічна панель із кнопками "Проєкти", "Користувачі" (для адміністратора) та "Вийти" забезпечує швидкий доступ до основних функцій.

- Візуалізація:

Кольорова палітра (бронзовий, темно-фіолетовий, білий тощо) відповідає естетичним стандартам і забезпечує читабельність.

- Сповіщення:

Кількість непрочитаних сповіщень відображається в бічній панелі, а їх позначення як прочитаних працює без затримок.

Тестування веб-програми для управління завданнями в колективі продемонструвало її високу функціональність, стабільність і відповідність основним вимогам. Перевірка серверної та клієнтської частин підтвердила коректність роботи основних функцій, таких як створення проєктів, управління завданнями та оцінка виконання. Водночас виявлено окремі моменти для вдосконалення, зокрема оптимізацію продуктивності при великих обсягах даних і додавання перевірок на максимальну кількість спеціалістів у проєкті. Загалом

система готова до використання в реальних умовах із можливістю подальшого масштабування та доопрацювання.

3.3. Проблеми та перспективи інтеграції та вдосконалення системи

Система управління завданнями в колективі, розроблена як веб-програма, демонструє значний потенціал для підтримки командної роботи. Однак її впровадження супроводжується певними труднощами, пов’язаними з інтеграцією в існуючі робочі процеси. Водночас можливості вдосконалення відкривають шляхи для підвищення функціональності та зручності використання. Аналіз цих факторів є важливим для забезпечення довгострокової ефективності рішення.

3.3.1. Проблеми інтеграції

Інтеграція веб-програми для управління завданнями в колективі з уже існуючими інструментами та робочими процесами може стикатися з низкою перешкод, які впливають на її успішне впровадження. Ці труднощі мають як технічний, так і організаційний характер, вимагаючи ретельного аналізу та пошуку відповідних рішень. Ключові проблеми, які можуть виникати на цьому етапі, описані в таблиці 3.2.

Таблиця 3.2

Основні ймовірні проблеми

№ з/п	Проблеми	Опис
1	2	3
1	Сумісність із зовнішніми системами	Одним із основних викликів є забезпечення сумісності з іншими програмними продуктами, такими як системи управління проєктами (наприклад, Jira, Trello), електронна пошта чи календарі (Google Calendar, Outlook). Використання локального API-сервера з базою даних SQLite ускладнює синхронізацію з хмарними сервісами через відмінності в протоколах передачі даних та форматах зберігання інформації. Наприклад, відсутність готових механізмів для експорту чи імпорту даних у стандартних форматах (JSON, CSV) може обмежувати інтеграцію з популярними платформами
2	Обмеження масштабованості	Архітектура системи, заснована на SQLite та локальному сервері Flask, не оптимальна для роботи з великою кількістю користувачів чи проєктів. SQLite, як однопотокова база даних, може створювати затримки при одночасному доступі багатьох користувачів, що особливо актуально для великих колективів. Це ускладнює інтеграцію в середовища з високим навантаженням, де потрібна стабільна обробка запитів у реальному часі
3	Безпека даних	Питання безпеки є критичним при інтеграції з корпоративними системами. Використання JWT-токенів для аутентифікації є позитивним кроком, однак відсутність шифрування передачі файлів чи додаткових механізмів захисту (наприклад, двофакторної аутентифікації) може стати вразливістю при взаємодії з зовнішніми сервісами. Корпоративні стандарти часто вимагають відповідності протоколам типу OAuth 2.0, що потребує доопрацювання
4	Складність адаптації до робочих процесів	Інтеграція системи в щоденні робочі процеси команди може наштовхуватися на опір через необхідність адаптації до нового інтерфейсу та логіки роботи. Наприклад, відсутність автоматичного імпорту наявних завдань чи проєктів із зовнішніх джерел змушує користувачів вручну переносити дані, що знижує продуктивність на початковому етапі. Крім того, обмежена гнучкість налаштувань (наприклад, неможливість задати власні статуси завдань) ускладнює відповідність специфічним потребам різних команд

Продовження таблиці 3.2

1	2	3
5	Обмежена підтримка багатомовності	Система орієнтована на україномовний інтерфейс, що може створювати бар'єри при інтеграції в міжнародні колективи. Відсутність локалізації чи можливості легко додавати нові мовні пакети ускладнює її використання в організаціях із різноманітним складом працівників, де потрібна підтримка кількох мов
6	Обробка великих обсягів файлів	Функціонал завантаження та скачування файлів, хоча і реалізований, не враховує обмеження на розмір чи оптимізацію зберігання великих обсягів даних. При інтеграції з системами, де обмін файлами є ключовим (наприклад, дизайнерські команди), це може призводити до затримок чи перевантаження локального сховища

3.3.2. Перспективи вдосконалення

Незважаючи на виявлені труднощі, система має значний потенціал для розвитку та вдосконалення. Оптимізація технічних рішень, розширення функціональності та покращення користувацького досвіду можуть суттєво підвищити її ефективність. Перспективні напрямки, які відкривають можливості для еволюції веб-програми, описані в таблиці 3.3.

Таблиця 3.3

Перспективні напрямки

№ з/п	Перспективи	Опис
1	2	3
1	Розширення API для інтеграції	Розробка розширеного API з підтримкою стандартних протоколів (RESTful чи GraphQL) дозволить спростити інтеграцію з зовнішніми сервісами. Додавання ендпоінтів для експорту/імпорту даних у популярних форматах, а також синхронізації з календарями чи системами управління завданнями, зробить систему більш універсальною. Наприклад, автоматична синхронізація дедлайнів із Google Calendar могла б підвищити зручність планування

Продовження таблиці 3.3

1	2	3
2	Перехід на хмарну інфраструктуру	Заміна SQLite на розподілену базу даних (наприклад, PostgreSQL чи MongoDB) та перенесення серверної частини на хмарну платформу (AWS, Azure) значно підвищить масштабність і продуктивність. Це дозволить обробляти велику кількість одночасних запитів, що є важливим для великих команд, а також забезпечить доступність системи з будь-якої точки світу
3	Покращення системи сповіщень	Оптимізація сповіщень шляхом додавання push-повідомлень через WebSocket чи інтеграція з месенджерами (Telegram, Slack) може зробити систему більш інтерактивною. Наприклад, миттєве сповіщення про призначення завдання чи наближення дедлайну підвищить оперативність реакції користувачів. Також доцільно реалізувати налаштування пріоритетів сповіщень для уникнення інформаційного перевантаження
4	Розширення аналітичних можливостей	Додавання модуля аналітики з візуалізацією продуктивності команди (графіки завершення завдань, статистика оцінок) дозволить менеджерам краще оцінювати ефективність роботи. Наприклад, інтеграція з бібліотеками типу Plotly для побудови динамічних звітів може стати конкурентною перевагою системи
5	Підтримка гнучких робочих процесів	Реалізація можливості налаштування власних статусів завдань, ролей користувачів чи шаблонів проєктів зробить систему адаптивною до різних методологій (Agile, Scrum). Це особливо корисно для команд із нестандартними підходами до управління, де потрібна висока гнучкість
6	Покращення безпеки	Впровадження двофакторної аутентифікації, шифрування файлів (AES-256) та відповідності стандартам GDPR чи ISO 27001 підвищить довіру до системи з боку великих організацій. Додаткові заходи, як-от обмеження доступу до файлів за ролями, також сприятимуть захисту конфіденційної інформації
7	Мобільна адаптація	Розробка мобільної версії чи додатку на основі прогресивних веб-додатків (PWA) розширить доступність системи. Це дозволить користувачам працювати із завданнями та отримувати сповіщення в

		дорозі, що особливо актуально для спеціалістів, які часто перебувають поза офісом

Завершення таблиці 3.3

1	2	3
8	Автоматизація рутинних процесів	Впровадження автоматичних нагадувань про дедлайни, генерації звітів чи розподілу завдань на основі навантаження спеціалістів зменшить ручну роботу менеджерів. Наприклад, алгоритм розподілу завдань із урахуванням поточної зайнятості може оптимізувати робочий процес
9	Локалізація та інтернаціоналізація	Додавання підтримки кількох мов (англійська, німецька тощо) через мовні пакети чи інтеграцію з інструментами типу i18n зробить систему привабливою для міжнародного ринку. Це особливо перспективно для компаній із глобальною присутністю
10	Оптимізація роботи з файлами	Впровадження хмарного сховища (Google Drive, Dropbox) для зберігання файлів замість локальної директорії зменшить навантаження на сервер і спростить обмін великими об’ємами даних. Додатково можна реалізувати попередній перегляд документів чи зображень без їх завантаження

Розроблена веб-програма для управління завданнями в колективі є функціональним рішенням, яке вже на етапі реалізації демонструє свою практичну цінність. Проте інтеграція з іншими системами та робочими процесами потребує подолання технічних і організаційних бар’єрів, таких як сумісність, масштабність і безпека. Водночас перспективи вдосконалення, включаючи розширення API, хмарну інфраструктуру та аналітичні інструменти, відкривають широкі можливості для підвищення її ефективності. Подальший розвиток системи має ґрунтуватися на балансі між технічними інноваціями та потребами користувачів, що забезпечить її конкурентоспроможність і довгострокову актуальність.

Висновки до розділу 3

Розробка та тестування веб-програми для управління завданнями в колективі стали важливими етапами створення інструменту, спрямованого на підвищення ефективності командної роботи. Реалізація основних модулів системи дозволила сформувати цілісне рішення, яке охоплює широкий спектр функцій – від аутентифікації користувачів до оцінки їхньої діяльності. Застосування сучасних технологій, таких як механізми шифрування паролів, токени для авторизації, інтерактивний інтерфейс та легка база даних, забезпечило надійність і зручність використання програми. Кожен модуль був ретельно спроєктований з урахуванням потреб різних категорій користувачів, що сприяло створенню гнучкої та функціональної системи.

Процес тестування підтвердив стабільність роботи як серверної, так і клієнтської частин, а також їхню здатність відповідати основним вимогам. Перевірка виявила коректність обробки запитів, цілісність даних і відповідність логіці ролей, що є важливим для забезпечення безпеки та справедливого розподілу доступу. Водночас тестування продуктивності показало, що при значному збільшенні обсягу даних чи кількості користувачів можуть виникати затримки, що вказує на потребу подальшої оптимізації. Безпека системи, попри міцну основу, також залишає простір для вдосконалення, зокрема через впровадження додаткових захисних механізмів.

Аналіз проблем інтеграції виявив певні труднощі, пов’язані з сумісністю із зовнішніми інструментами, обмеженою масштабністю та необхідністю адаптації до різноманітних робочих процесів. Ці виклики вимагають уважного підходу до технічних рішень і врахування організаційних особливостей команд. Проте система має значний потенціал для розвитку. Перспективи її вдосконалення охоплюють розширення можливостей інтеграції, перехід на більш потужну інфраструктуру, покращення аналітичних функцій і підвищення зручності для користувачів із різними потребами. Такий розвиток здатен зробити програму не лише функціональним інструментом для локального використання, а й конкурентоспроможним рішенням для ширшого кола організацій.

Загалом, створена веб-програма демонструє практичну цінність як засіб автоматизації управління завданнями в колективі. Її сильні сторони полягають у продуманій структурі модулів, зрозумілому інтерфейсі та гнучкості для різних ролей. Разом із тим, подальша робота над оптимізацією продуктивності, посиленням безпеки та розширенням функціональності дозволить досягти нового рівня ефективності. Розробка цього рішення стала прикладом того, як сучасні технології можуть гармонійно поєднуватися з реальними потребами командної роботи, відкриваючи шлях до вдосконалення на основі досвіду користувачів і технологічного прогресу.

ВИСНОВКИ

У результаті проведеного дослідження розроблено веб-програму для управління завданнями в колективі, яка стала відповіддю на актуальну потребу в оптимізації командної роботи в умовах зростання складності проєктів та обсягів інформації. Метою роботи було створення інструменту, який би забезпечив ефективну координацію діяльності, доступ до актуальної інформації та контроль виконання завдань. Ця мета досягнута шляхом послідовного виконання поставлених завдань, що охоплювали аналіз потреб користувачів, вивчення існуючих рішень, вибір технологій, проектування архітектури, розробку модулів, створення інтерфейсу, тестування працездатності та визначення перспектив удосконалення системи.

Дослідження потреб користувачів показало, що сучасні колективи потребують гнучких, інтуїтивно зрозумілих і доступних інструментів для організації роботи, які б враховували специфіку їхніх робочих процесів. Аналіз аналогів, таких як Trello, Asana та Jira, виявив їхні сильні сторони, зокрема зручність інтерфейсу та широкий функціонал, але також вказав на обмеження в адаптивності до локальних потреб і високу вартість для невеликих команд. Це стало підставою для створення власного рішення, яке поєднує простоту використання з можливістю кастомізації. Вибір технологій, таких як Python, Flask, Streamlit і SQLite, обґрунтований їхньою ефективністю, простотою інтеграції та відповідністю вимогам безпеки й продуктивності. Такий підхід дозволив створити систему, яка є одночасно легкою у розгортанні та здатною до масштабування.

Проектування системи базувалося на чітко визначених функціональних вимогах, що включали управління проєктами, завданнями, календарем, оцінками, коментарями та файлами. Архітектура клієнт-серверного типу забезпечила чіткий розподіл обов’язків між серверною частиною, відповідальною за обробку даних і логіку, та клієнтською, яка реалізує інтерактивний інтерфейс. Розроблений інтерфейс враховує принципи користувацького дизайну, пропонуючи інтуїтивну навігацію та візуально приємне оформлення, що сприяє комфортній взаємодії.

Реалізація модулів системи дозволила втілити всі заплановані функції, включаючи авторизацію, управління ролями, відстеження прогресу та обмін інформацією між учасниками команди.

Тестування підтвердило працездатність системи: серверна частина стабільно обробляє запити, клієнтська забезпечує коректне відображення даних, база даних надійно зберігає інформацію, а продуктивність залишається достатньою для невеликих і середніх колективів. Перевірка безпеки виявила, що використання JWT-токенів і хешування паролів ефективно захищає дані користувачів, хоча вразливості можуть виникати при недостатній конфігурації серверного середовища. Зручність використання оцінена позитивно завдяки простоті інтерфейсу, однак потребує подальшого тестування з реальними користувачами для врахування їхнього досвіду.

Проблеми інтеграції пов’язані з необхідністю адаптації системи до існуючих робочих процесів організацій, що може вимагати додаткових зусиль для налаштування API та сумісності з іншими інструментами. Також виявлено обмеження SQLite у масштабованості при роботі з великою кількістю одночасних користувачів, що вказує на потребу переходу до більш потужних баз даних, таких як PostgreSQL, у майбутньому. Перспективи вдосконалення охоплюють додавання функцій аналітики продуктивності, інтеграцію з месенджерами для сповіщень, а також розробку мобільної версії для підвищення доступності. Можливість розширення системи за рахунок модульної структури відкриває шлях до її адаптації під специфічні потреби різних команд.

Розроблена веб-програма має практичну цінність, оскільки може бути застосована в реальних умовах для підвищення ефективності командної роботи. Її відкритий код і гнучка архітектура сприяють подальшому розвитку та модифікації. Водночас дослідження виявило проблемні моменти, такі як потреба в оптимізації продуктивності при значному навантаженні та вдосконаленні документації для полегшення інтеграції. Рекомендується зосередити увагу на створенні детальних інструкцій для користувачів і адміністраторів, а також на

проведенні додаткових тестів у реальних робочих середовищах для оцінки довготривалої ефективності.

Подальші дослідження варто спрямувати на вивчення можливостей інтеграції штучного інтелекту для автоматизації розподілу завдань і прогнозування термінів їх виконання. Також доцільно розглянути впровадження хмарних технологій для забезпечення високої доступності та масштабованості системи. Розширення функціоналу в напрямку підтримки багатомовності та персоналізації інтерфейсу може зробити програму більш універсальною для міжнародних команд. Таким чином, розробка відкриває нові горизонти для вдосконалення інструментів управління завданнями, сприяючи прогресу в організації колективної праці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kerzner, Harold. Project Management: A Systems Approach to Planning, Scheduling, and Controlling, Eleventh Edition WPLS Student Package. USA, Wiley, 2016.
2. Sommerville, Ian. Engineering Software Products: An Introduction to Modern Software Engineering. Pearson, 2020.
3. David T. Bourgeois, Ph.D. and Bourgeois, Information Systems for Business and Beyond. 2014
4. Sahid Abdelkebir, Maleh Yassine, Mustapha Belaissaoui. Information System Evolution. 2020
5. Alan S. Gutterman. Management Roles and Activities. 2023
6. Guide to Project Management Software URL: https://www.getharvest.com/blog/project-management-software
7. Renata Kenda, Nicoleta Meslec, Leon Oerlemans. Single versus multiple project teams and individual performance: Do they ask for different leadership behaviors? International Journal of Project Management, Volume 42, Issue 5, July 2024
8. Whitney Vige. What is a project team? Plus, why your enterprise needs one. 2024
9. Needs Analysis: Common Types and How to Do One Step-by-Step URL: https://claned.com/needs-analysis/
10. Хто такий і чим займається Technical Support Engineer? URL: https://hire1.com.ua/job-description/technical-support-engineer
11. Technology Support Specialist URL: https://www.pcc.edu/hr/employment/classified-jobs/techsupptspechtm/
12. User Needs URL: https://www.interaction-design.org/literature/topics/user-needs
13. Jihan Syafa Kamila, Muhammad Falah Marzuq. Asana and Trello: A Comparative Assessment of Project Management Capabilities. JOIV International Journal on Informatics Visualization 8(1):207. 2024

14. Trello brings all your tasks, teammates, and tools together URL: https://trello.com/home

15. Trello полегшує керування проєктами й задачами для команд URL: https://trello.com/uk/tour

16. Where work connects URL: https://asana.com/?noredirect

17. ASANA. Сервіс, який зробить вашу роботу ефективнішою URL: https://euprostir.org.ua/practices/134869

18. Made for work, designed to love URL: https://monday.com/

19. monday.com Огляд URL: https://www.ranktracker.com/uk/blog/monday-com-review/

20. The everything app, for work URL: https://clickup.com/

21. 7 способів використання ClickUp URL: https://apix-drive.com/ua/blog/useful/7-sposobiv-vikoristannja-clickup

22. Що таке Microsoft Teams? URL: https://support.microsoft.com/uk-ua/topic/що-таке-microsoft-teams-3de4d369-0167-8def-b93b-0eb5286d7a29

23. Microsoft Teams URL: https://uk.wikipedia.org/wiki/Microsoft_Teams

24. Slack URL: https://slack.com/

25. Great outcomes start with Jira URL: https://www.atlassian.com/software/jira

26. Zoho Projects URL: https://play.google.com/store/apps/details?id=com.zoho.projects&hl=pl

27. Як обрати архітектуру для веб додатку URL: https://blog.ithillel.ua/articles/web-application-architecture

28. Python: мова програмування майбутнього URL: https://vgoru.org/pererva-na-kavu/python-mova-programuvannya-majbutnogo

29. 15 Мов Програмування, Що Будуть Домінувати у 2025 Році URL: https://careers.easternpeak.com/blog/popular-programming-languages/

30. Що таке PostgreSQL і для чого використовується? URL: https://foxminded.ua/postgresql-shcho-tse/

31. Керування базами даних та інтеграція з веб-додатками: MySQL, MongoDB і SQL Server URL: https://redstone.agency/blog/keruvannia-bazamy-danych-ta-intehracija-z-veb-dodatkamy-mysql-mongodb-i-sql-server/

32. What's the Difference Between MySQL and PostgreSQL? URL: https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/

33. Створення RESTful API за допомогою Python і Flask URL: https://uk.sharpcoderblog.com/blog/creating-restful-apis-with-python-and-flask

34. Flask by Example URL: https://realpython.com/learning-paths/flask-by-example/

35. A faster way to build and share data apps URL: https://streamlit.io/

36. Streamlit: A Powerful Tool for Rapid Prototyping and Data Visualization URL: https://levelup.gitconnected.com/streamlit-a-powerful-tool-for-rapid-prototyping-and-data-visualization-747188750dbf

ДОДАТОК А.

КОД ПРОГРАМИ

client.py:

```
import streamlit as st
import requests
import json
from datetime import datetime, timedelta
import pandas as pd
from io import BytesIO
import plotly.graph_objects as go
import plotly.express as px
from streamlit_calendar import calendar
import streamlit.components.v1 as components
from datetime import datetime, timezone
import time
# Колірна палітра з еталонного зображення
COLORS = {
    "primary": "#806543",      # Бронзовий/Коричневий
    "secondary": "#33266E",    # Темно-фіолетовий
    "dark": "#111111",         # Чорний
    "accent1": "#542F34",      # Бордовий
    "accent2": "#A6607C",      # Рожевий/Фіолетовий
    "white": "#FFFFFF",        # Білий
    "light_gray": "#F8F9FA",   # Світло-сірий
    "success": "#4CAF50",      # Зелений
    "danger": "#f44336",       # Червоний
    "info": "#2196F3"          # Синій
}
# Налаштування сторінки
st.set_page_config(
    page_title="📚 Система управління проєктами",
    page_icon="📚",
    layout="wide",
    initial_sidebar_state="expanded"
)
# Кастомні стилі CSS з оновленою кольоровою гамою
st.markdown(f"""
    <style>
        /* Базові елементи */
        .stApp {{
            background-color: {COLORS["white"]};
            color: {COLORS["dark"]};
        }}
        /* Оформлення бічної панелі */
        .css-1d391kg, .css-12oz5g7 {{
            background-color: {COLORS["primary"]};
        }}
        header {{
            background-color: {COLORS["primary"]};
        }}
        /* Інформація про користувача у бічній панелі */
        .user-info {{
            background-color: {COLORS["secondary"]};
            color: {COLORS["white"]};
            padding: 20px;
            border-radius: 10px;
            margin-bottom: 20px;
        }}
        /* Кнопки */
```

```
.stButton button {{
    width: 100%;
    border-radius: 5px;
    height: 3em;
    background-color: {COLORS["secondary"]};
    color: white;
    border: none;
    margin: 5px 0px;
}}
.stButton button:hover {{
    background-color: {COLORS["primary"]};
    box-shadow: 0 2px 5px rgba(0,0,0,0.2);
}}
.delete-button button {{
    background-color: {COLORS["danger"]};
}}
.delete-button button:hover {{
    background-color: #da190b;
}}
/* Картки проєктів */
.project-card {{
    padding: 20px;
    border-radius: 10px;
    border: 1px solid #ddd;
    margin: 10px 0px;
    background-color: {COLORS["white"]};
    position: relative;
    transition: all 0.3s ease;
}}
.project-card:hover {{
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
    border-left: 5px solid {COLORS["primary"]};
}}
/* Картки завдань */
.kanban-column {{
    background-color: {COLORS["light_gray"]};
    border-radius: 10px;
    padding: 15px;
    margin: 10px;
    min-height: 300px;
    border-top: 4px solid {COLORS["secondary"]};
}}
.task-card {{
    background-color: {COLORS["white"]};
    border: 1px solid #ddd;
    border-radius: 5px;
    padding: 10px;
    margin: 5px 0;
    cursor: pointer;
    transition: all 0.2s ease;
}}
.task-card:hover {{
    box-shadow: 0 2px 5px rgba(0,0,0,0.2);
    border-left: 3px solid {COLORS["primary"]};
}}
/* Значок сповіщень */
.notification-badge {{
    background-color: {COLORS["danger"]};
    color: white;
    padding: 2px 6px;
    border-radius: 50%;
    font-size: 12px;
    position: absolute;
    top: -5px;
    right: -5px;
```

```
}}
/* Таблиці */
.grades-table {{
    width: 100%;
    border-collapse: collapse;
    margin: 20px 0;
}}
.grades-table th, .grades-table td {{
    border: 1px solid #ddd;
    padding: 8px;
    text-align: left;
}}
.grades-table th {{
    background-color: {COLORS["secondary"]};
    color: white;
}}
.user-row {{
    padding: 10px;
    border-bottom: 1px solid #ddd;
    display: flex;
    justify-content: space-between;
    align-items: center;
}}
.user-info {{
    padding: 20px;
    background-color: {COLORS["secondary"]};
    border-radius: 10px;
    margin-bottom: 20px;
    color: white;
}}
/* Події календаря */
 .calendar-event {{
    padding: 10px;
    border-radius: 5px;
    margin: 5px 0;
}}
.event-meeting {{
    background-color: #E8F5E9;
    border-left: 4px solid {COLORS["success"]};
}}
.event-deadline {{
    background-color: #FFEBEE;
    border-left: 4px solid {COLORS["danger"]};
}}
.event-other {{
    background-color: #E3F2FD;
    border-left: 4px solid {COLORS["info"]};
}}
/* Стилі для повноекранного календаря */
.fc {{
    background-color: white;
    padding: 10px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}}
.fc-event {{
    cursor: pointer;
    padding: 2px 5px;
}}
.fc-daygrid-day {{
    height: 100px;
}}
/* Коментарі */
.comment-card {{
    padding: 10px;
```

```
            border-radius: 5px;
            margin: 10px 0;
            background-color: {COLORS["light_gray"]};
            border-left: 3px solid {COLORS["primary"]};
        }}
        /* Картки файлів */
        .file-card {{
            padding: 10px;
            border-radius: 5px;
            margin: 5px 0;
            background-color: {COLORS["light_gray"]};
            border-left: 3px solid {COLORS["info"]};
        }}
        /* Заголовки та назви розділів */
        h1, h2 {{
            color: {COLORS["primary"]};
        }}

        h3, h4, h5, h6 {{
            color: {COLORS["secondary"]};
        }}
        /* Поля форм та введення */
        .stTextInput input, .stTextArea textarea, .stSelectbox, .stMultiselect {{
            border-radius: 5px;
            border: 1px solid #ddd;
        }}
        .stTextInput input:focus, .stTextArea textarea:focus {{
            border: 1px solid {COLORS["primary"]};
            box-shadow: 0 0 3px {COLORS["primary"]};
        }}
    </style>
""", unsafe_allow_html=True)
# Константи
API_URL = "http://localhost:5000"
ROLES = {
    'specialist': 'Спеціаліст',
    'manager': 'Менеджер',
    'admin': 'Адміністратор'
}
# Клас для роботи з API та JWT токенами
class ApiClient:
    def __init__(self):
        self.token = st.session_state.get('token', None)
        self.headers = {'Authorization': f'Bearer {self.token}'} if self.token else {}
    def refresh_token(self):
        """Оновлення JWT токена"""
        if self.token and self._is_token_expired():
            response = requests.post(f"{API_URL}/refresh-token", headers=self.headers)
            if response.status_code == 200:
                self.token = response.json()['token']
                self.headers = {'Authorization': f'Bearer {self.token}'}
                st.session_state.token = self.token
    def _is_token_expired(self):
        """Перевірка чи токен прострочений"""
        try:
            import jwt
            decoded = jwt.decode(self.token, options={"verify_signature": False})
            exp = datetime.fromtimestamp(decoded['exp'])
            # Використовуємо timezone-aware datetime
            current_time = datetime.now(timezone.utc)
            return current_time + timedelta(minutes=5) >= exp.replace(tzinfo=timezone.utc)
        except:
            return True
    def post(self, endpoint, data):
        self.refresh_token()
```

```
        headers = {
            'Content-Type': 'application/json',
            **self.headers
        }
        return requests.post(
            f"{API_URL}{endpoint}",
            json=data,
            headers=headers
        )
    def get(self, endpoint):
        self.refresh_token()
        return requests.get(f"{API_URL}{endpoint}", headers=self.headers)
    def put(self, endpoint, data):
        self.refresh_token()
        return requests.put(f"{API_URL}{endpoint}", json=data, headers=self.headers)
    def delete(self, endpoint):
        self.refresh_token()
        return requests.delete(f"{API_URL}{endpoint}", headers=self.headers)
    def upload_file(self, endpoint, files):
        self.refresh_token()
        return requests.post(f"{API_URL}{endpoint}", files=files, headers=self.headers)
# Ініціалізація стану
if 'token' not in st.session_state:
    st.session_state.token = None
if 'user' not in st.session_state:
    st.session_state.user = None
if 'current_page' not in st.session_state:
    st.session_state.current_page = 'login'
if 'notifications' not in st.session_state:
    st.session_state.notifications = []
if 'project_members' not in st.session_state:
    st.session_state.project_members = []
if 'unread_count' not in st.session_state:
    st.session_state.unread_count = 0
api_client = ApiClient()
def show_messages():
    """Відображення повідомлень"""
    if 'show_success_message' in st.session_state and st.session_state.show_success_message:
        st.success(st.session_state.success_message)
        del st.session_state.show_success_message
        del st.session_state.success_message
    if 'show_error_message' in st.session_state and st.session_state.show_error_message:
        st.error(st.session_state.error_message)
        del st.session_state.show_error_message
        del st.session_state.error_message
def update_notifications():
    """Оптимізована функція оновлення сповіщень"""
    if st.session_state.token:
        try:
            response = api_client.get("/notifications/unread-count")
            if response.status_code == 200:
                st.session_state.unread_count = response.json()['unread_count']
                # Отримуємо сповіщення тільки якщо є непрочитані
                if st.session_state.unread_count > 0:
                    notifications_response = api_client.get("/notifications")
                    if notifications_response.status_code == 200:
                        st.session_state.notifications = notifications_response.json()
        except Exception as e:
            st.error(f"Помилка оновлення сповіщень: {str(e)}")
def show_login_page():
    col1, col2, col3 = st.columns([1, 2, 1])
    with col2:
        st.markdown(f"<h1 style='text-align: center; color: {COLORS['primary']};'>🎓 Система
управління проєктами</h1>", unsafe_allow_html=True)
        with st.form("login_form"):
```

```
            st.markdown(f"<p class='big-text' style='color: {COLORS['secondary']};'>Вхід у
систему</p>", unsafe_allow_html=True)
            email = st.text_input("Електронна пошта")
            password = st.text_input("Пароль", type="password")
            submitted = st.form_submit_button("Увійти")
            if submitted:
                try:
                    response = api_client.post("/login", {
                        "email": email,
                        "password": password
                    })
                    if response.status_code == 200:
                        data = response.json()
                        st.session_state.token = data['token']
                        st.session_state.user = data['user']
                        st.session_state.current_page = 'projects'
                        st.rerun()
                    else:
                        st.error("Невірна пошта або пароль")
                except Exception as e:
                    st.error(f"Помилка підключення до сервера: {str(e)}")
        if st.button("Зареєструватися"):
            st.session_state.current_page = 'register'
            st.rerun()
def show_register_page():
    col1, col2, col3 = st.columns([1, 2, 1])
    with col2:
        st.markdown(f"<h1 style='text-align: center; color: {COLORS['primary']};'>🎓
Реєстрація</h1>", unsafe_allow_html=True)
        with st.form("register_form"):
            name = st.text_input("Ім'я")
            email = st.text_input("Електронна пошта")
            password = st.text_input("Пароль", type="password")
            role = st.selectbox("Роль", ['specialist', 'manager'], format_func=lambda x:
ROLES[x])
            submitted = st.form_submit_button("Зареєструватися")
            if submitted:
                try:
                    response = api_client.post("/register", {
                        "name": name,
                        "email": email,
                        "password": password,
                        "role": role
                    })
                    if response.status_code == 201:
                        st.success("Реєстрація успішна!")
                        st.session_state.current_page = 'login'
                        st.rerun()
                    else:
                        st.error("Користувач з такою поштою вже існує")
                except Exception as e:
                    st.error(f"Помилка реєстрації: {str(e)}")
def show_admin_panel():
    st.title("Панель адміністратора")
    response = api_client.get("/users")
    if response.status_code == 200:
        users = response.json()
        managers = [u for u in users if u['role'] == 'manager']
        specialists = [u for u in users if u['role'] == 'specialist']
        tab1, tab2 = st.tabs(["Менеджери", "Спеціалісти"])
        with tab1:
            st.header("Список менеджерів")
            for manager in managers:
                with st.container():
                    col1, col2 = st.columns([3, 1])
```

```
                    with col1:
                        st.markdown(f"""
                            <div class="user-card">
                                <h4>{manager['name']}</h4>
                                <p>Email: {manager['email']}</p>
                                <p>Проєктів: {manager.get('projects_count', 0)}</p>
                            </div>
                        """, unsafe_allow_html=True)
                    with col2:
                        if st.button("❌ Видалити", key=f"delete_manager_{manager['id']}"):
                            response = api_client.delete(f"/users/{manager['id']}")
                            if response.status_code == 200:
                                st.success("Менеджера видалено")
                                st.rerun()
                            else:
                                st.error("Помилка при видаленні менеджера")
        with tab2:
            st.header("Список спеціалістів")
            for specialist in specialists:
                with st.container():
                    col1, col2 = st.columns([3, 1])
                    with col1:
                        st.markdown(f"""
                            <div class="user-card">
                                <h4>{specialist['name']}</h4>
                                <p>Email: {specialist['email']}</p>
                                <p>Проєктів: {specialist.get('projects_count', 0)}</p>
                            </div>
                        """, unsafe_allow_html=True)
                    with col2:
                        if st.button("❌ Видалити",
key=f"delete_specialist_{specialist['id']}"):
                            response = api_client.delete(f"/users/{specialist['id']}")
                            if response.status_code == 200:
                                st.success("Спеціаліста видалено")
                                st.rerun()
                            else:
                                st.error("Помилка при видаленні спеціаліста")
def show_project_details():
    project = st.session_state.current_project
    st.title(project['name'])
    # Вкладки проєкту
    tabs = ["Деталі", "Завдання", "Календар", "Обговорення", "Файли"]
    if st.session_state.user['role'] == 'manager':
        tabs.append("Оцінка виконання")
    current_tab = st.tabs(tabs)
    with current_tab[0]:
        show_project_info(project)
    with current_tab[1]:
        show_kanban_board(project['id'])
    with current_tab[2]:
        show_calendar(project['id'])
    with current_tab[3]:
        if st.session_state.user['role'] != 'admin':
            show_comments(project['id'])
        else:
            st.info("Адміністратор не може брати участь в обговоренні")
    with current_tab[4]:
        show_files(project['id'])
    if st.session_state.user['role'] == 'manager' and len(current_tab) > 5:
        with current_tab[5]:
            show_performance_evaluation(project['id'])
def show_project_info(project):
    """Відображення детальної інформації про проект"""
    st.markdown(f"""
```

```
        <div class="project-card">
            <h3>{project['name']}</h3>
            <p>{project.get('description', 'Опис відсутній')}</p>
            <p><strong>Менеджер:</strong> {project.get('manager_name', 'Не призначено')}</p>
            <p><strong>Дедлайн:</strong> {project.get('deadline', 'Не встановлено')}</p>
            <p><strong>Статус:</strong> {project.get('status', 'Активний')}</p>
            <p><strong>Максимум спеціалістів:</strong> {project.get('max_specialists', 'Не
обмежено')}</p>
        </div>
    """, unsafe_allow_html=True)
    # Відображення статистики проекту
    col1, col2, col3 = st.columns(3)
    with col1:
        st.metric("Учасників", project.get('members_count', 0))
    with col2:
        st.metric("Завдань", project.get('total_tasks', 0))
    with col3:
        st.metric("Завершено", project.get('completed_tasks', 0))
def show_performance_evaluation(project_id):
    st.header("📚 Оцінка виконання")
    # Отримуємо список спеціалістів проекту
    members_response = api_client.get(f"/projects/{project_id}/members")
    if members_response.status_code != 200:
        st.error("Помилка при отриманні списку учасників")
        return
    # Фільтруємо тільки спеціалістів
    specialists = [m for m in members_response.json() if m['role'] == 'specialist']
    # Отримуємо існуючі оцінки
    ratings_response = api_client.get(f"/projects/{project_id}/ratings")
    if ratings_response.status_code != 200:
        st.error("Помилка при отриманні оцінок")
        return
    ratings = ratings_response.json()
    ratings_dict = {r['specialist_id']: r for r in ratings}
    # Відображення таблиці оцінок
    if specialists:
        data = []
        for specialist in specialists:
            rating_info = ratings_dict.get(specialist['id'], {})
            data.append({
                'ID': specialist['id'],
                'Спеціаліст': specialist['name'],
                'Оцінка': rating_info.get('rating', '-'),
                'Коментар': rating_info.get('comment', ''),
                'Дата оцінювання': rating_info.get('timestamp', '')
            })
        df = pd.DataFrame(data)
        st.dataframe(df.set_index('ID'), use_container_width=True)
        # Форма для виставлення оцінок (тільки для менеджера)
        if st.session_state.user['role'] == 'manager':
            with st.form("rating_form"):
                st.subheader("Оцінити виконання")
                specialist = st.selectbox(
                    "Оберіть спеціаліста",
                    options=specialists,
                    format_func=lambda x: x['name']
                )
                rating = st.number_input(
                    "Оцінка",
                    min_value=0.0,
                    max_value=100.0,
                    value=60.0,
                    step=0.5
                )
                comment = st.text_area("Коментар")
```

```
                if st.form_submit_button("Виставити оцінку"):
                    response = api_client.post(f"/projects/{project_id}/ratings", {
                        "specialist_id": specialist['id'],
                        "rating": rating,
                        "comment": comment
                    })
                    if response.status_code == 200:
                        st.success("Оцінку успішно виставлено")
                        st.rerun()
                    else:
                        st.error("Помилка при виставленні оцінки")
    else:
        st.info("У проєкті ще немає спеціалістів")
def show_admin_users():
    """Відображення списку користувачів для адміністратора"""
    st.title("Керування користувачами")
    # Отримуємо список всіх користувачів
    response = api_client.get("/users")
    if response.status_code != 200:
        st.error("Не вдалося завантажити список користувачів")
        return
    users = response.json()
    # Розділяємо користувачів за ролями
    managers = [u for u in users if u['role'] == 'manager']
    specialists = [u for u in users if u['role'] == 'specialist']
    # Створюємо вкладки для різних типів користувачів
    tab1, tab2 = st.tabs(["Менеджери", "Спеціалісти"])
    with tab1:
        st.subheader("Список менеджерів")
        for manager in managers:
            with st.container():
                col1, col2 = st.columns([3, 1])
                with col1:
                    st.markdown(f"""
                        <div class="user-card">
                            <h4>{manager['name']}</h4>
                            <p>Email: {manager['email']}</p>
                            <p>Проєктів: {manager.get('projects_count', 0)}</p>
                        </div>
                    """, unsafe_allow_html=True)
                with col2:
                    if st.button("Детальніше", key=f"manager_{manager['id']}"):
                        show_manager_details(manager['id'])
    with tab2:
        st.subheader("Список спеціалістів")
        for specialist in specialists:
            with st.container():
                col1, col2 = st.columns([3, 1])
                with col1:
                    st.markdown(f"""
                        <div class="user-card">
                            <h4>{specialist['name']}</h4>
                            <p>Email: {specialist['email']}</p>
                            <p>Проєктів: {specialist.get('projects_count', 0)}</p>
                            <p>Середня оцінка: {specialist.get('average_rating', 'Немає
оцінок')}</p>
                        </div>
                    """, unsafe_allow_html=True)
                with col2:
                    if st.button("Детальніше", key=f"specialist_{specialist['id']}"):
                        show_specialist_details(specialist['id'])
def show_manager_details(manager_id):
    """Відображення детальної інформації про менеджера"""
    response = api_client.get(f"/users/{manager_id}/details")
```

```
    if response.status_code != 200:
        st.error("Не вдалося завантажити інформацію про менеджера")
        return
    manager = response.json()
    st.subheader(f"Менеджер: {manager['name']}")
    # Статистика по проєктах
    st.markdown("### Проєкти")
    if manager.get('projects'):
        for project in manager['projects']:
            st.markdown(f"""
                <div class="project-card">
                    <h4>{project['name']}</h4>
                    <p>Спеціалістів: {project['specialists_count']}</p>
                    <p>Середня оцінка: {project['average_rating']:.1f}</p>
                    <p>Завершеність: {project['completion_rate']}%</p>
                </div>
            """, unsafe_allow_html=True)
    else:
        st.info("Немає активних проєктів")
def show_specialist_details(specialist_id):
    """Відображення детальної інформації про спеціаліста"""
    response = api_client.get(f"/users/{specialist_id}/details")
    if response.status_code != 200:
        st.error("Не вдалося завантажити інформацію про спеціаліста")
        return
    specialist = response.json()
    st.subheader(f"Спеціаліст: {specialist['name']}")
    # Статистика по проєктах
    st.markdown("### Проєкти та оцінки")
    if specialist.get('projects'):
        for project in specialist['projects']:
            st.markdown(f"""
                <div class="project-card">
                    <h4>{project['name']}</h4>
                    <p>Менеджер: {project['manager_name']}</p>
                    <p>Оцінка: {project.get('rating', 'Не виставлена')}</p>
                    <p>Виконаних завдань:
{project['completed_tasks']}/{project['total_tasks']}</p>
                </div>
            """, unsafe_allow_html=True)
    else:
        st.info("Спеціаліст не бере участі в проєктах")
# Додаткові функції для оптимізованої системи сповіщень
def notify_users(project_id, message, user_ids=None):
    """Надсилання сповіщень користувачам"""
    data = {
        "project_id": project_id,
        "message": message,
        "user_ids": user_ids
    }
    response = api_client.post("/notifications/send", data)
    return response.status_code == 200
def mark_notifications_read(notification_ids):
    """Позначення сповіщень як прочитаних"""
    response = api_client.post("/notifications/mark-read", {
        "notification_ids": notification_ids
    })
    return response.status_code == 200
def get_notifications_count():
    """Отримання кількості непрочитаних сповіщень"""
    response = api_client.get("/notifications/unread-count")
    if response.status_code == 200:
        return response.json()['unread_count']
    return 0
def show_notifications_popup():
```

```
    """Відображення спливаючого вікна зі сповіщеннями"""
    # Отримуємо останні сповіщення
    response = api_client.get("/notifications?limit=5")
    if response.status_code == 200:
        notifications = response.json()
        with st.container():
            for notification in notifications:
                st.markdown(f"""
                    <div class="notification-card {'unread' if not notification['is_read'] else
''}">
                        <h4>{notification['title']}</h4>
                        <p>{notification['message']}</p>
                        <small>{notification['created_at']}</small>
                    </div>
                """, unsafe_allow_html=True)
                if not notification['is_read']:
                    if st.button("Позначити як прочитане",
key=f"mark_read_{notification['id']}"):
                        mark_notifications_read([notification['id']])
                        st.rerun()
# Покращені функції для роботи з проєктами
def get_project_statistics(project_id):
    """Отримання статистики по проєкту"""
    response = api_client.get(f"/projects/{project_id}/statistics")
    if response.status_code == 200:
        return response.json()
    return None
def show_project_statistics(project_id):
    """Відображення статистики проєкту"""
    stats = get_project_statistics(project_id)
    if not stats:
        return
    col1, col2, col3 = st.columns(3)
    with col1:
        st.metric("Учасників", stats['members_count'])
    with col2:
        st.metric("Виконано завдань", f"{stats['completed_tasks']}/{stats['total_tasks']}")
    with col3:
        st.metric("Середня оцінка", f"{stats['average_rating']:.1f}")
    # Графік прогресу
    progress_data = pd.DataFrame(stats['progress_history'])
    fig = px.line(progress_data, x='date', y='completion_rate',
                  title='Прогрес виконання проєкту')
    st.plotly_chart(fig)
def show_projects():
    st.title("Проєкти")
    # Для менеджера - можливість створення
    if st.session_state.user['role'] == 'manager':
        if st.button("➕ Створити новий проект"):
            st.session_state.current_page = 'create_project'
            st.rerun()
    response = api_client.get("/projects")
    if response.status_code == 200:
        projects = response.json()
        # Відображення проєктів
        for project in projects:
            with st.container():
                col1, col2 = st.columns([3, 1])
                with col1:
                    st.markdown(f"""
                        <div class="project-card">
                            <h3>{project['name']}</h3>
                            <p>{project.get('description', 'Опис відсутній')}</p>
                            <p><strong>Менеджер:</strong> {project.get('manager_name', 'Не
призначено')}</p>
```

```
                        <p><strong>Дедлайн:</strong> {project.get('deadline', 'Не
встановлено')}</p>
                    </div>
                """, unsafe_allow_html=True)
            with col2:
                if st.button("📋 Деталі", key=f"details_{project['id']}"):
                    st.session_state.current_project = project
                    st.session_state.current_page = 'project_details'
                    st.rerun()
                if (st.session_state.user['role'] == 'specialist' and
                    not project.get('is_member', False)):
                    if st.button("📥 Приєднатися", key=f"join_{project['id']}"):
                        response = api_client.post(f"/projects/{project['id']}/join", {})
                        if response.status_code == 200:
                            st.success("✅ Ви успішно приєднались до проєкту!")
                            st.rerun()
                        else:
                            st.error(response.json().get('message', 'Помилка при приєднанні
до проєкту'))
    # Показуємо повідомлення після оновлення сторінки
    if 'show_success_message' in st.session_state and st.session_state.show_success_message:
        st.success(st.session_state.success_message)
        # Очищуємо повідомлення
        del st.session_state.show_success_message
        del st.session_state.success_message
    if 'show_error_message' in st.session_state and st.session_state.show_error_message:
        st.error(st.session_state.error_message)
        # Очищуємо повідомлення
        del st.session_state.show_error_message
        del st.session_state.error_message
def show_kanban_board(project_id):
    st.header("Завдання")
    # Отримання всіх завдань
    response = api_client.get(f"/projects/{project_id}/tasks")
    if response.status_code == 200:
        tasks = response.json()
        # Додавання нового завдання (тільки для менеджера)
        if st.session_state.user['role'] == 'manager':
            with st.expander("Додати нове завдання"):
                with st.form("new_task_form"):
                    title = st.text_input("Назва завдання")
                    description = st.text_area("Опис")
                    deadline = st.date_input("Дедлайн")
                    assigned_to = st.selectbox(
                        "Призначити спеціалісту",
                        options=[m['id'] for m in st.session_state.project_members if m['role']
== 'specialist'],
                        format_func=lambda x: next(m['name'] for m in
st.session_state.project_members if m['id'] == x)
                    )
                    if st.form_submit_button("Створити"):
                        create_task(project_id, title, description, deadline, assigned_to)
        # Відображення дошки
        col1, col2, col3 = st.columns(3)
        with col1:
            st.subheader("Не розпочато")
            show_task_column(tasks, "not_started", project_id)
        with col2:
            st.subheader("В процесі")
            show_task_column(tasks, "in_progress", project_id)
        with col3:
            st.subheader("Завершено")
            show_task_column(tasks, "completed", project_id)
def show_task_column(tasks, status, project_id):
    filtered_tasks = [t for t in tasks if t['status'] == status]
```

```
    for task in filtered_tasks:
        with st.container():
            st.markdown(f"""
                <div class="task-card">
                    <h4>{task['title']}</h4>
                    <p>{task['description']}</p>
                    <p><small>Дедлайн: {task['deadline']}</small></p>
                    <p><small>Виконавець: {task.get('assigned_user_name', 'Не
призначено')}</small></p>
                </div>
            """, unsafe_allow_html=True)
            if st.session_state.user['role'] != 'admin':
                if status == "not_started":
                    if st.button("→ В процес", key=f"start_{task['id']}"):
                        update_task_status(project_id, task['id'], "in_progress")
                elif status == "in_progress":
                    if st.button("→ Завершено", key=f"complete_{task['id']}"):
                        update_task_status(project_id, task['id'], "completed")
def show_calendar(project_id):
    st.header("📅 Календар")
    # Отримання подій
    response = api_client.get(f"/projects/{project_id}/calendar")
    if response.status_code == 200:
        events = response.json()
        # Додавання нової події (тільки для менеджера)
        if st.session_state.user['role'] == 'manager':
            with st.expander("Додати подію"):
                with st.form("new_event_form"):
                    title = st.text_input("Назва події")
                    description = st.text_area("Опис")
                    event_type = st.selectbox(
                        "Тип події",
                        ["meeting", "deadline", "other"],
                        format_func=lambda x: {
                            "meeting": "Зустріч",
                            "deadline": "Дедлайн",
                            "other": "Інше"
                        }[x]
                    )
                    col1, col2 = st.columns(2)
                    with col1:
                        start_date = st.date_input("Дата початку")
                        start_time = st.time_input("Час початку")
                    with col2:
                        end_date = st.date_input("Дата завершення")
                        end_time = st.time_input("Час завершення")
                    if st.form_submit_button("Додати"):
                        start_datetime = datetime.combine(start_date, start_time)
                        end_datetime = datetime.combine(end_date, end_time)
                        response = api_client.post(f"/projects/{project_id}/calendar", {
                            "title": title,
                            "description": description,
                            "event_type": event_type,
                            "start_time": start_datetime.isoformat(),
                            "end_time": end_datetime.isoformat()
                        })
                        if response.status_code == 201:
                            st.session_state.show_success_message = True
                            st.session_state.success_message = "Подію успішно додано!"
                        else:
                            st.error("Помилка при додаванні події")
        try:
            # Форматування подій для календаря
            calendar_events = []
            for event in events:
```

```
                calendar_events.append({
                    'id': str(event['id']),  # ID повинен бути рядком
                    'title': event['title'],
                    'start': event['start_time'],
                    'end': event['end_time'],
                    'extendedProps': {
                        'type': event['event_type'],
                        'description': event.get('description', '')
                    },
                    'backgroundColor': {
                        'meeting': '#4CAF50',
                        'deadline': '#f44336',
                        'other': '#2196F3'
                    }.get(event['event_type'], '#9E9E9E')
                })
            # Конфігурація календаря
            calendar_options = {
                "headerToolbar": {
                    "left": "prev,next today",
                    "center": "title",
                    "right": "dayGridMonth,timeGridWeek,timeGridDay"
                },
                "initialView": "dayGridMonth",
                "selectable": True,
                "editable": False,
                "dayMaxEvents": True,
                "slotMinTime": "08:00:00",
                "slotMaxTime": "20:00:00",
                "expandRows": True,
                "locale": "uk"
            }
            # Відображення календаря в контейнері
            with st.container():
                calendar(
                    events=calendar_events,
                    options=calendar_options,
                    key=f"calendar_{project_id}"  # Унікальний ключ для кожного проекту
                )
        except Exception as e:
            st.error(f"Помилка при відображенні календаря: {str(e)}")
        # Показ списку подій
        st.subheader("Список подій")
        for event in sorted(events, key=lambda x: x['start_time']):
            with st.expander(f"{event['title']} ({event['start_time']})"):
                st.markdown(f"""
                    <div class="event-{event['event_type']}">
                        <p><strong>Опис:</strong> {event.get('description', 'Без опису')}</p>
                        <p><strong>Тип:</strong> {
                            {'meeting': 'Зустріч',
                             'deadline': 'Дедлайн',
                             'other': 'Інше'
                            }.get(event['event_type'], 'Інше')
                        }</p>
                        <p><strong>Початок:</strong> {event['start_time']}</p>
                        <p><strong>Кінець:</strong> {event['end_time']}</p>
                    </div>
                """, unsafe_allow_html=True)
def create_project():
    st.title("Створення нового проєкту")
    with st.form("create_project_form"):
        name = st.text_input("Назва проєкту")
        description = st.text_area("Опис проєкту")
        deadline = st.date_input("Дедлайн")
        max_specialists = st.number_input("Максимальна кількість спеціалістів",
                                          min_value=1, value=5)
```

```
            submitted = st.form_submit_button("Створити проєкт")
            if submitted:
                try:
                    if not name:
                        st.error("Введіть назву проєкту")
                        return
                    data = {
                        "name": name,
                        "description": description,
                        "deadline": deadline.strftime("%Y-%m-%d"),
                        "max_specialists": max_specialists
                    }
                    response = api_client.post("/projects", data)
                    if response.status_code == 201:
                        st.success("✅ Проєкт успішно створено!")
                        time.sleep(1)  # Даємо користувачу час побачити повідомлення про успіх
                        st.session_state.current_page = 'projects'
                        st.rerun()
                    else:
                        error_msg = response.json().get('message', 'Невідома помилка')
                        st.error(f"❌ Помилка при створенні проєкту: {error_msg}")
                except Exception as e:
                    st.error(f"❌ Помилка: {str(e)}")
        if st.button("↩ Назад до проєктів"):
            st.session_state.current_page = 'projects'
            st.rerun()
    def show_comments(project_id):
        st.header("Обговорення")
        # Додавання нового коментаря
        if st.session_state.user['role'] != 'admin':
            with st.form("new_comment_form"):
                comment_text = st.text_area("Новий коментар")
                if st.form_submit_button("Додати коментар"):
                    response = api_client.post(f"/projects/{project_id}/comments", {
                        "content": comment_text
                    })
                    if response.status_code == 201:
                        st.success("Коментар додано!")
                        st.rerun()
                    else:
                        st.error("Помилка при додаванні коментаря")
        # Відображення коментарів
        response = api_client.get(f"/projects/{project_id}/comments")
        if response.status_code == 200:
            comments = response.json()
            for comment in comments:
                st.markdown(f"""
                    <div class="comment-card">
                        <strong>{comment['author_name']}</strong>
                        <small>{comment['timestamp']}</small>
                        <p>{comment['content']}</p>
                    </div>
                """, unsafe_allow_html=True)
        else:
            st.error("Помилка при завантаженні коментарів")
    def show_files(project_id):
        st.header("Файли")
        # Завантаження файлу
        if st.session_state.user['role'] != 'admin':
            uploaded_file = st.file_uploader("Завантажити файл")
            if uploaded_file:
                if st.button("Завантажити"):
                    upload_file(project_id, uploaded_file)
        # Список файлів
        response = api_client.get(f"/projects/{project_id}/files")
```

```
    if response.status_code == 200:
        files = response.json()
        for file in files:
            col1, col2 = st.columns([3, 1])
            with col1:
                st.markdown(f"""
                    <div class="file-card">
                        <strong>{file['filename']}</strong><br>
                        <small>Завантажив: {file['user_name']} • {file['upload_date']}</small>
                    </div>
                """, unsafe_allow_html=True)
            with col2:
                if st.button("Завантажити", key=f"download_{file['id']}"):
                    download_file(file['id'])
# Допоміжні функції
def set_current_project(project):
    st.session_state.current_project = project
    st.session_state.current_page = 'project_details'
    st.rerun()
def join_project(project_id):
    response = api_client.post(f"/projects/{project_id}/join", {})
    if response.status_code == 200:
        st.success("Ви успішно приєднались до проекту!")
        st.rerun()
    else:
        st.error("Помилка при приєднанні до проекту")
def delete_project(project_id):
    if st.session_state.user['role'] == 'admin':
        response = api_client.delete(f"/projects/{project_id}")
        if response.status_code == 200:
            st.success("Проект видалено!")
            st.rerun()
        else:
            st.error("Помилка при видаленні проекту")
def create_task(project_id, title, description, deadline, assigned_to):
    response = api_client.post(f"/projects/{project_id}/tasks", {
        "title": title,
        "description": description,
        "deadline": deadline.isoformat(),
        "assigned_to": assigned_to
    })
    if response.status_code == 201:
        st.success("Завдання створено!")
        st.rerun()
    else:
        st.error("Помилка при створенні завдання")
def update_task_status(project_id, task_id, new_status):
    response = api_client.put(f"/projects/{project_id}/tasks/{task_id}", {
        "status": new_status
    })
    if response.status_code == 200:
        st.rerun()
    else:
        st.error("Помилка при оновленні статусу завдання")
def create_event(project_id, title, description, event_type, start_date, end_date):
    response = api_client.post(f"/projects/{project_id}/calendar", {
        "title": title,
        "description": description,
        "event_type": event_type,
        "start_time": start_date.isoformat(),
        "end_time": end_date.isoformat()
    })
    if response.status_code == 201:
        st.success("Подію створено!")
        st.rerun()
```

```
    else:
        st.error("Помилка при створенні події")
def create_comment(project_id, content):
    response = api_client.post(f"/projects/{project_id}/comments", {
        "content": content
    })
    if response.status_code == 201:
        st.success("Коментар додано!")
        st.rerun()
    else:
        st.error("Помилка при додаванні коментаря")
def upload_file(project_id, file):
    files = {"file": file}
    response = api_client.upload_file(f"/projects/{project_id}/files", files)
    if response.status_code == 201:
        st.success("Файл завантажено!")
        st.rerun()
    else:
        st.error("Помилка при завантаженні файлу")
def download_file(file_id):
    response = api_client.get(f"/files/{file_id}/download")
    if response.status_code == 200:
        st.download_button(
            label="Завантажити файл",
            data=response.content,
            file_name=response.headers['Content-Disposition'].split('filename=')[1],
            mime=response.headers['Content-Type']
        )
def logout():
    st.session_state.token = None
    st.session_state.user = None
    st.session_state.current_page = 'login'
    st.rerun()
def main():
    update_notifications()
    if not st.session_state.token:
        if st.session_state.current_page == 'register':
            show_register_page()
        else:
            show_login_page()
    else:
        with st.sidebar:
            st.markdown(f"""
                <div class="user-info">
                    <h3>👤 {st.session_state.user['name']}</h3>
                    <p>{ROLES[st.session_state.user['role']]}</p>
                </div>
            """, unsafe_allow_html=True)
            st.button("Проєкти", on_click=lambda: setattr(st.session_state, 'current_page',
'projects'))
            if st.session_state.user['role'] == 'admin':
                st.button("Користувачі", on_click=lambda: setattr(st.session_state,
'current_page', 'admin_panel'))
            if st.session_state.unread_count > 0:
                st.button(f"Сповіщення ({st.session_state.unread_count})")
            st.button("Вийти", on_click=logout)
        # Основний контент
        if st.session_state.current_page == 'projects':
            show_projects()
        elif st.session_state.current_page == 'create_project':
            create_project()
        elif st.session_state.current_page == 'project_details':
            show_project_details()
        elif st.session_state.current_page == 'admin_panel':
            show_admin_panel()
```

```
if __name__ == "__main__":
    main()
```

server.py:

```
from flask import Flask, request, jsonify, send_file
from flask_cors import CORS
from werkzeug.security import generate_password_hash, check_password_hash
import sqlite3
import jwt
from datetime import datetime, timedelta, timezone
from functools import wraps
import os
from werkzeug.utils import secure_filename
import logging
# Налаштування логування
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
app = Flask(__name__)
CORS(app)
app.config['SECRET_KEY'] = 'your-secret-key'  # В продакшені використовувати безпечний ключ
app.config['UPLOAD_FOLDER'] = 'files'
# Константи
ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'doc', 'docx', 'xls', 'xlsx'}
TOKEN_EXPIRE_HOURS = 24
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
def init_db():
    conn = sqlite3.connect('project_management.db')
    c = conn.cursor()
    # Таблиця користувачів
    c.execute('''CREATE TABLE IF NOT EXISTS users
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  name TEXT NOT NULL,
                  email TEXT UNIQUE NOT NULL,
                  password TEXT NOT NULL,
                  role TEXT NOT NULL,
                  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
                  last_login DATETIME)''')
    # Таблиця проєктів (змінюємо поле teacher_id на manager_id, max_students на
max_specialists)
    c.execute('''CREATE TABLE IF NOT EXISTS projects
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  name TEXT NOT NULL,
                  description TEXT,
                  manager_id INTEGER,
                  deadline DATE,
                  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
                  status TEXT DEFAULT 'active' CHECK(status IN ('active', 'archived',
'completed')),
                  max_specialists INTEGER DEFAULT 0,
                  FOREIGN KEY (manager_id) REFERENCES users (id))''')
    # Таблиця учасників проєкту
    c.execute('''CREATE TABLE IF NOT EXISTS project_members
                 (project_id INTEGER,
                  user_id INTEGER,
                  role TEXT DEFAULT 'member',
                  join_date DATETIME DEFAULT CURRENT_TIMESTAMP,
                  FOREIGN KEY (project_id) REFERENCES projects (id),
```

```
                FOREIGN KEY (user_id) REFERENCES users (id),
                PRIMARY KEY (project_id, user_id))''')
    # Таблиця завдань
    c.execute('''CREATE TABLE IF NOT EXISTS tasks
                (id INTEGER PRIMARY KEY AUTOINCREMENT,
                 project_id INTEGER,
                 title TEXT NOT NULL,
                 description TEXT,
                 status TEXT CHECK(status IN ('not_started', 'in_progress', 'completed'))
DEFAULT 'not_started',
                 created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
                 deadline DATE,
                 assigned_to INTEGER,
                 priority TEXT DEFAULT 'medium' CHECK(priority IN ('low', 'medium', 'high')),
                 FOREIGN KEY (project_id) REFERENCES projects (id),
                 FOREIGN KEY (assigned_to) REFERENCES users (id))''')
    # Таблиця подій календаря
    c.execute('''CREATE TABLE IF NOT EXISTS calendar_events
                (id INTEGER PRIMARY KEY AUTOINCREMENT,
                 project_id INTEGER,
                 title TEXT NOT NULL,
                 description TEXT,
                 event_type TEXT NOT NULL,
                 start_time DATETIME NOT NULL,
                 end_time DATETIME NOT NULL,
                 created_by INTEGER,
                 recurrence TEXT,
                 FOREIGN KEY (project_id) REFERENCES projects (id),
                 FOREIGN KEY (created_by) REFERENCES users (id))''')
    # Таблиця сповіщень
    c.execute('''CREATE TABLE IF NOT EXISTS notifications
                (id INTEGER PRIMARY KEY AUTOINCREMENT,
                 user_id INTEGER,
                 project_id INTEGER,
                 type TEXT NOT NULL,
                 message TEXT NOT NULL,
                 created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
                 is_read BOOLEAN DEFAULT 0,
                 priority TEXT DEFAULT 'normal' CHECK(priority IN ('low', 'normal', 'high')),
                 expiry_date DATETIME,
                 FOREIGN KEY (user_id) REFERENCES users (id),
                 FOREIGN KEY (project_id) REFERENCES projects (id))''')
    # Таблиця коментарів
    c.execute('''CREATE TABLE IF NOT EXISTS comments
                (id INTEGER PRIMARY KEY AUTOINCREMENT,
                 project_id INTEGER,
                 user_id INTEGER,
                 content TEXT NOT NULL,
                 timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                 parent_id INTEGER,
                 FOREIGN KEY (project_id) REFERENCES projects (id),
                 FOREIGN KEY (user_id) REFERENCES users (id),
                 FOREIGN KEY (parent_id) REFERENCES comments (id))''')
    # Таблиця файлів
    c.execute('''CREATE TABLE IF NOT EXISTS files
                (id INTEGER PRIMARY KEY AUTOINCREMENT,
                 project_id INTEGER,
                 user_id INTEGER,
                 filename TEXT NOT NULL,
                 file_size INTEGER,
                 file_type TEXT,
                 upload_date DATETIME DEFAULT CURRENT_TIMESTAMP,
                 file_path TEXT NOT NULL,
                 FOREIGN KEY (project_id) REFERENCES projects (id),
                 FOREIGN KEY (user_id) REFERENCES users (id))''')
```

```
    # Змінюємо таблицю оцінок на таблицю рейтингів
    c.execute('''CREATE TABLE IF NOT EXISTS ratings
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  project_id INTEGER,
                  specialist_id INTEGER,
                  rating FLOAT CHECK(rating >= 0 AND rating <= 100),
                  comment TEXT,
                  manager_id INTEGER,
                  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                  type TEXT DEFAULT 'final' CHECK(type IN ('interim', 'final')),
                  FOREIGN KEY (project_id) REFERENCES projects (id),
                  FOREIGN KEY (specialist_id) REFERENCES users (id),
                  FOREIGN KEY (manager_id) REFERENCES users (id))''')
    # Таблиця активності користувачів
    c.execute('''CREATE TABLE IF NOT EXISTS user_activity
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  user_id INTEGER,
                  project_id INTEGER,
                  action_type TEXT NOT NULL,
                  action_details TEXT,
                  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                  FOREIGN KEY (user_id) REFERENCES users (id),
                  FOREIGN KEY (project_id) REFERENCES projects (id))''')
    conn.commit()
    conn.close()
def get_db():
    conn = sqlite3.connect('project_management.db')
    conn.row_factory = sqlite3.Row
    return conn
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')
        if not token:
            return jsonify({'message': 'Token is missing'}), 401
        try:
            if 'Bearer ' in token:
                token = token.split('Bearer ')[1]
            data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=["HS256"])
            current_user = get_user_by_id(data['user_id'])
            if not current_user:
                return jsonify({'message': 'Invalid token'}), 401
        except jwt.ExpiredSignatureError:
            return jsonify({'message': 'Token expired'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'message': 'Invalid token'}), 401
        return f(current_user, *args, **kwargs)
    return decorated
def log_activity(user_id, project_id, action_type, action_details=None):
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            INSERT INTO user_activity (user_id, project_id, action_type, action_details)
            VALUES (?, ?, ?, ?)
        """, (user_id, project_id, action_type, action_details))
        conn.commit()
    except Exception as e:
        logger.error(f"Error logging activity: {str(e)}")
    finally:
        conn.close()
def create_notification(user_id, project_id, notification_type, message, priority='normal',
expiry_days=30):
    conn = None
    try:
```

```
        conn = get_db()
        c = conn.cursor()
        expiry_date = datetime.now(timezone.utc) + timedelta(days=expiry_days)
        # Встановлюємо таймаут для операцій з базою даних
        conn.execute("PRAGMA busy_timeout = 5000")  # таймаут 5 секунд
        c.execute("""
            INSERT INTO notifications (user_id, project_id, type, message, priority,
expiry_date)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (user_id, project_id, notification_type, message, priority, expiry_date))
        conn.commit()
    except Exception as e:
        logger.error(f"Помилка створення сповіщення: {str(e)}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            conn.close()
# Допоміжні функції для роботи з користувачами
def get_user_by_id(user_id):
    conn = get_db()
    c = conn.cursor()
    c.execute("SELECT * FROM users WHERE id=?", (user_id,))
    user = c.fetchone()
    conn.close()
    if user:
        return {
            'id': user['id'],
            'name': user['name'],
            'email': user['email'],
            'role': user['role']
        }
    return None
def get_user_by_email(email):
    conn = get_db()
    c = conn.cursor()
    c.execute("SELECT * FROM users WHERE email=?", (email,))
    user = c.fetchone()
    conn.close()
    return user if user else None
# Аутентифікація та реєстрація
@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    if not all(k in data for k in ['name', 'email', 'password', 'role']):
        return jsonify({'message': 'Відсутні обов\'язкові поля'}), 400
    if get_user_by_email(data['email']):
        return jsonify({'message': 'Користувач вже існує'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        hashed_password = generate_password_hash(data['password'], method='scrypt')
        c.execute("""
            INSERT INTO users (name, email, password, role, created_at)
            VALUES (?, ?, ?, ?, ?)
        """, (data['name'], data['email'], hashed_password, data['role'],
              datetime.now(timezone.utc)))
        conn.commit()
        return jsonify({'message': 'Реєстрація успішна'}), 201
    except Exception as e:
        logger.error(f"Помилка реєстрації: {str(e)}")
        return jsonify({'message': 'Помилка під час реєстрації'}), 500
    finally:
        conn.close()
@app.route('/login', methods=['POST'])
```

```
def login():
    data = request.get_json()
    if not all(k in data for k in ['email', 'password']):
        return jsonify({'message': 'Відсутня електронна пошта або пароль'}), 400
    try:
        user = get_user_by_email(data['email'])
        if not user:
            return jsonify({'message': 'Користувача не знайдено'}), 401
        if check_password_hash(user['password'], data['password']):
            # Оновлення часу останнього входу
            conn = get_db()
            c = conn.cursor()
            c.execute("UPDATE users SET last_login = ? WHERE id = ?",
                      (datetime.now(timezone.utc), user['id']))
            conn.commit()
            conn.close()
            token = jwt.encode({
                'user_id': user['id'],
                'exp': datetime.now(timezone.utc) + timedelta(hours=TOKEN_EXPIRE_HOURS)
            }, app.config['SECRET_KEY'])
            return jsonify({
                'token': token,
                'user': {
                    'id': user['id'],
                    'name': user['name'],
                    'email': user['email'],
                    'role': user['role']
                }
            })
        return jsonify({'message': 'Невірний пароль'}), 401
    except Exception as e:
        logger.error(f"Помилка входу: {str(e)}")
        return jsonify({'message': 'Помилка під час входу'}), 500
# Маршрути для проєктів
@app.route('/projects', methods=['GET'])
@token_required
def get_projects(current_user):
    try:
        conn = get_db()
        c = conn.cursor()
        # SQL запит залежить від ролі користувача
        if current_user['role'] == 'admin':
            # Адміністратор бачить усі проєкти
            c.execute("""
                SELECT
                    p.*,
                    u.name as manager_name,
                    COUNT(DISTINCT pm.user_id) as members_count
                FROM projects p
                LEFT JOIN users u ON p.manager_id = u.id
                LEFT JOIN project_members pm ON p.id = pm.project_id
                WHERE p.status != 'archived'
                GROUP BY p.id
            """)
        elif current_user['role'] == 'manager':
            # Менеджер бачить свої проєкти
            c.execute("""
                SELECT
                    p.*,
                    u.name as manager_name,
                    COUNT(DISTINCT pm.user_id) as members_count
                FROM projects p
                LEFT JOIN users u ON p.manager_id = u.id
                LEFT JOIN project_members pm ON p.id = pm.project_id
                WHERE p.manager_id = ? AND p.status != 'archived'
```

```
                GROUP BY p.id
            """, (current_user['id'],))
        else:  # specialist
            # Спеціаліст бачить всі активні проєкти та свою участь в них
            c.execute("""
                SELECT
                    p.*,
                    u.name as manager_name,
                    COUNT(DISTINCT pm.user_id) as members_count,
                    CASE WHEN EXISTS (
                        SELECT 1 FROM project_members
                        WHERE project_id = p.id AND user_id = ?
                    ) THEN 1 ELSE 0 END as is_member
                FROM projects p
                LEFT JOIN users u ON p.manager_id = u.id
                LEFT JOIN project_members pm ON p.id = pm.project_id
                WHERE p.status = 'active'
                GROUP BY p.id
            """, (current_user['id'],))
        projects = []
        rows = c.fetchall()
        for row in rows:
            # Отримуємо кількість непрочитаних сповіщень
            if current_user['role'] == 'specialist':
                c.execute("""
                    SELECT COUNT(*) FROM notifications
                    WHERE project_id = ? AND user_id = ? AND is_read = 0
                """, (row['id'], current_user['id']))
            else:
                c.execute("""
                    SELECT COUNT(*) FROM notifications
                    WHERE project_id = ? AND is_read = 0
                """, (row['id'],))
            unread_count = c.fetchone()[0]
            project = {
                'id': row['id'],
                'name': row['name'],
                'description': row['description'],
                'manager_id': row['manager_id'],
                'manager_name': row['manager_name'],
                'deadline': row['deadline'],
                'status': row['status'],
                'members_count': row['members_count'],
                'unread_notifications': unread_count
            }
            if current_user['role'] == 'specialist':
                project['is_member'] = bool(row['is_member'])
            projects.append(project)
        return jsonify(projects)
    except Exception as e:
        logger.error(f"Помилка отримання проєктів: {str(e)}")
        return jsonify({'message': 'Помилка отримання проєктів'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/comments', methods=['GET'])
@token_required
def get_comments(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            SELECT c.id, c.content, c.timestamp, u.name as author_name
            FROM comments c
            JOIN users u ON c.user_id = u.id
            WHERE c.project_id = ?
```

```
            ORDER BY c.timestamp ASC
        """, (project_id,))
        comments = [{
            'id': row['id'],
            'content': row['content'],
            'timestamp': row['timestamp'],
            'author_name': row['author_name']
        } for row in c.fetchall()]
        return jsonify(comments), 200
    except Exception as e:
        logger.error(f"Error fetching comments: {str(e)}")
        return jsonify({'message': 'Error fetching comments'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/comments', methods=['POST'])
@token_required
def add_comment(current_user, project_id):
    data = request.get_json()
    if 'content' not in data:
        return jsonify({'message': 'Missing comment content'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            INSERT INTO comments (project_id, user_id, content)
            VALUES (?, ?, ?)
        """, (project_id, current_user['id'], data['content']))
        conn.commit()
        return jsonify({'message': 'Comment added successfully'}), 201
    except Exception as e:
        logger.error(f"Error adding comment: {str(e)}")
        return jsonify({'message': 'Error adding comment'}), 500
    finally:
        conn.close()
@app.route('/projects', methods=['POST'])
@token_required
def create_project(current_user):
    if current_user['role'] not in ['manager', 'admin']:
        return jsonify({'message': 'Недостатньо прав'}), 403
    data = request.get_json()
    if not data:
        return jsonify({'message': 'Дані не надані'}), 400
    required_fields = ['name', 'description', 'deadline']
    if not all(field in data for field in required_fields):
        return jsonify({'message': 'Відсутні обов\'язкові поля'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        # Створюємо проєкт з правильними назвами колонок
        c.execute("""
            INSERT INTO projects (
                name, description, manager_id, deadline,
                max_specialists, status, created_at
            )
            VALUES (?, ?, ?, ?, ?, 'active', datetime('now'))
        """, (
            data['name'],
            data['description'],
            current_user['id'],
            data['deadline'],
            data.get('max_specialists', 0)
        ))
        project_id = c.lastrowid
        # Додаємо творця як учасника проєкту
        c.execute("""
```

```
            INSERT INTO project_members (project_id, user_id, role, join_date)
            VALUES (?, ?, ?, datetime('now'))
        """, (project_id, current_user['id'], current_user['role']))
        # Логуємо активність
        log_activity(
            current_user['id'],
            project_id,
            'project_created',
            f"Створено проєкт: {data['name']}"
        )
        conn.commit()
        return jsonify({
            'message': 'Проєкт успішно створено',
            'project_id': project_id
        }), 201
    except Exception as e:
        logger.error(f"Помилка створення проєкту: {str(e)}")
        if conn:
            conn.rollback()
        return jsonify({'message': 'Помилка створення проєкту'}), 500
    finally:
        if conn:
            conn.close()
@app.route('/projects/<int:project_id>', methods=['PUT'])
@token_required
def update_project(current_user, project_id):
    if current_user['role'] not in ['manager', 'admin']:
        return jsonify({'message': 'Недостатньо прав'}), 403
    data = request.get_json()
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо права на проєкт
        c.execute("SELECT manager_id FROM projects WHERE id = ?", (project_id,))
        project = c.fetchone()
        if not project:
            return jsonify({'message': 'Проєкт не знайдено'}), 404
        if current_user['role'] == 'manager' and project['manager_id'] != current_user['id']:
            return jsonify({'message': 'Недостатньо прав'}), 403
        # Оновлюємо проєкт
        update_fields = []
        params = []
        if 'name' in data:
            update_fields.append('name = ?')
            params.append(data['name'])
        if 'description' in data:
            update_fields.append('description = ?')
            params.append(data['description'])
        if 'deadline' in data:
            update_fields.append('deadline = ?')
            params.append(data['deadline'])
        if 'status' in data and current_user['role'] == 'admin':
            update_fields.append('status = ?')
            params.append(data['status'])
        if update_fields:
            params.append(project_id)
            query = f"UPDATE projects SET {', '.join(update_fields)} WHERE id = ?"
            c.execute(query, params)
            # Логуємо зміни
            log_activity(current_user['id'], project_id, 'project_updated',
                         f"Оновлено поля проєкту: {', '.join(update_fields)}")
            # Сповіщаємо учасників
            c.execute("SELECT user_id FROM project_members WHERE project_id = ?",
(project_id,))
            members = c.fetchall()
```

```
            for member in members:
                create_notification(
                    member['user_id'],
                    project_id,
                    'project_update',
                    f"Проєкт '{data.get('name', 'Невідомий')}' був оновлений"
                )
        conn.commit()
        return jsonify({'message': 'Проєкт успішно оновлено'})
    except Exception as e:
        logger.error(f"Помилка оновлення проєкту: {str(e)}")
        return jsonify({'message': 'Помилка оновлення проєкту'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>', methods=['DELETE'])
@token_required
def delete_project(current_user, project_id):
    if current_user['role'] != 'admin':
        return jsonify({'message': 'Unauthorized'}), 403
    try:
        conn = get_db()
        c = conn.cursor()
        # Проверяем существование проекта
        c.execute("SELECT id FROM projects WHERE id = ?", (project_id,))
        if not c.fetchone():
            return jsonify({'message': 'Project not found'}), 404
        # Удаляем все связанные данные
        tables = [
            'project_members', 'tasks', 'calendar_events',
            'notifications', 'comments', 'grades',
            'user_activity', 'files'
        ]
        for table in tables:
            c.execute(f"DELETE FROM {table} WHERE project_id = ?", (project_id,))
        # Удаляем сам проект
        c.execute("DELETE FROM projects WHERE id = ?", (project_id,))
        # Логируем удаление
        log_activity(current_user['id'], None, 'project_deleted',
                     f"Deleted project ID: {project_id}")
        conn.commit()
        return jsonify({'message': 'Project deleted successfully'})
    except Exception as e:
        logger.error(f"Error deleting project: {str(e)}")
        return jsonify({'message': 'Error deleting project'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/join', methods=['POST'])
@token_required
def join_project(current_user, project_id):
    if current_user['role'] != 'specialist':
        return jsonify({'message': 'Тільки спеціалісти можуть приєднуватись до проєктів'}), 403
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо існування проєкту та його статус
        c.execute("""
            SELECT id, status, manager_id FROM projects
            WHERE id = ? AND status = 'active'
        """, (project_id,))
        project = c.fetchone()
        if not project:
            return jsonify({'message': 'Проєкт не знайдено або він не активний'}), 404
        # Перевіряємо, чи не є спеціаліст вже учасником
        c.execute("""
            SELECT 1 FROM project_members
```

```
            WHERE project_id = ? AND user_id = ?
        """, (project_id, current_user['id']))
        if c.fetchone():
            return jsonify({'message': 'Ви вже є учасником цього проєкту'}), 400
        # Додаємо спеціаліста в проєкт
        c.execute("""
            INSERT INTO project_members (project_id, user_id, role)
            VALUES (?, ?, 'specialist')
        """, (project_id, current_user['id']))
        # Отримуємо інформацію про проєкт для сповіщення
        c.execute("""
            SELECT p.name, u.name as specialist_name
            FROM projects p
            JOIN users u ON u.id = ?
            WHERE p.id = ?
        """, (current_user['id'], project_id))
        project_info = c.fetchone()
        # Створюємо сповіщення для менеджера
        create_notification(
            project['manager_id'],
            project_id,
            'new_member',
            f"Спеціаліст {project_info['specialist_name']} приєднався до проєкту
'{project_info['name']}'"
        )
        # Логуємо дію
        log_activity(
            current_user['id'],
            project_id,
            'project_joined',
            f"Приєднався до проєкту: {project_info['name']}"
        )
        conn.commit()
        return jsonify({'message': 'Успішно приєднано до проєкту'})
    except Exception as e:
        logger.error(f"Помилка приєднання до проєкту: {str(e)}")
        conn.rollback()
        return jsonify({'message': 'Помилка приєднання до проєкту'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/calendar', methods=['GET'])
@token_required
def get_calendar_events(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            SELECT e.*, u.name as creator_name
            FROM calendar_events e
            LEFT JOIN users u ON e.created_by = u.id
            WHERE e.project_id = ?
            ORDER BY e.start_time
        """, (project_id,))
        events = [{
            'id': row['id'],
            'title': row['title'],
            'description': row['description'],
            'event_type': row['event_type'],
            'start_time': row['start_time'],
            'end_time': row['end_time'],
            'created_by': row['creator_name']
        } for row in c.fetchall()]
        return jsonify(events)
    except Exception as e:
        logger.error(f"Error getting calendar events: {str(e)}")
```

```
        return jsonify({'message': 'Error getting calendar events'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/calendar', methods=['POST'])
@token_required
def create_calendar_event(current_user, project_id):
    data = request.get_json()
    if not all(k in data for k in ['title', 'event_type', 'start_time', 'end_time']):
        return jsonify({'message': 'Відсутні обов\'язкові поля'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо, чи існує проєкт і чи має користувач доступ
        c.execute("""
            SELECT 1 FROM project_members
            WHERE project_id = ? AND user_id = ?
        """, (project_id, current_user['id']))
        if not c.fetchone() and current_user['role'] != 'admin':
            return jsonify({'message': 'У вас немає доступу до цього проєкту'}), 403
        # Створюємо подію
        c.execute("""
            INSERT INTO calendar_events (
                project_id, title, description, event_type,
                start_time, end_time, created_by
            )
            VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (
            project_id,
            data['title'],
            data.get('description', ''),
            data['event_type'],
            data['start_time'],
            data['end_time'],
            current_user['id']
        ))
        event_id = c.lastrowid
        # Створюємо сповіщення для всіх учасників проєкту
        c.execute("""
            SELECT user_id FROM project_members
            WHERE project_id = ? AND user_id != ?
        """, (project_id, current_user['id']))
        for member in c.fetchall():
            create_notification(
                member['user_id'],
                project_id,
                'new_event',
                f"Додано нову подію до календаря: {data['title']}"
            )
        # Логуємо дію
        log_activity(
            current_user['id'],
            project_id,
            'event_created',
            f"Створено подію календаря: {data['title']}"
        )
        conn.commit()
        return jsonify({
            'message': 'Подію успішно додано',
            'event_id': event_id
        }), 201
    except Exception as e:
        logger.error(f"Помилка створення події: {str(e)}")
        return jsonify({'message': 'Помилка створення події'}), 500
    finally:
        conn.close()
```

```
@app.route('/projects/<int:project_id>/grades', methods=['GET'])
@token_required
def get_grades(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо права доступу
        if current_user['role'] == 'specialist':
            # Спеціаліст бачить тільки свої оцінки
            c.execute("""
                SELECT g.*, u.name as student_name, t.name as teacher_name
                FROM grades g
                JOIN users u ON g.student_id = u.id
                JOIN users t ON g.teacher_id = t.id
                WHERE g.project_id = ? AND g.student_id = ?
            """, (project_id, current_user['id']))
        else:
            # Менеджер та адмін бачать всі оцінки
            c.execute("""
                SELECT g.*, u.name as student_name, t.name as teacher_name
                FROM grades g
                JOIN users u ON g.student_id = u.id
                JOIN users t ON g.teacher_id = t.id
                WHERE g.project_id = ?
            """, (project_id,))
        # Повертаємо як "ratings"
        grades = [{
            'id': row['id'],
            'specialist_id': row['student_id'],
            'specialist_name': row['student_name'],
            'rating': row['grade'],
            'comment': row['comment'],
            'manager_name': row['teacher_name'],
            'timestamp': row['timestamp'],
            'type': row['type']
        } for row in c.fetchall()]
        return jsonify(grades)
    except Exception as e:
        logger.error(f"Помилка отримання оцінок: {str(e)}")
        return jsonify({'message': 'Помилка отримання оцінок'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/members', methods=['GET'])
@token_required
def get_project_members(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        # Отримуємо учасників проекту з їх ролями
        c.execute("""
            SELECT u.id, u.name, u.email, pm.role
            FROM project_members pm
            JOIN users u ON pm.user_id = u.id
            WHERE pm.project_id = ?
        """, (project_id,))
        members = [{
            'id': row['id'],
            'name': row['name'],
            'email': row['email'],
            'role': row['role']
        } for row in c.fetchall()]
        return jsonify(members)
    except Exception as e:
        logger.error(f"Error getting project members: {str(e)}")
        return jsonify({'message': 'Error getting project members'}), 500
```

```
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/files', methods=['GET'])
@token_required
def get_project_files(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            SELECT f.*, u.name as user_name
            FROM files f
            JOIN users u ON f.user_id = u.id
            WHERE f.project_id = ?
            ORDER BY f.upload_date DESC
        """, (project_id,))
        files = [{
            'id': row['id'],
            'filename': row['filename'],
            'file_size': row['file_size'],
            'file_type': row['file_type'],
            'upload_date': row['upload_date'],
            'user_name': row['user_name']
        } for row in c.fetchall()]
        return jsonify(files)
    except Exception as e:
        logger.error(f"Error getting project files: {str(e)}")
        return jsonify({'message': 'Error getting project files'}), 500
    finally:
        conn.close()
@app.route('/users', methods=['GET'])
@token_required
def get_users(current_user):
    if current_user['role'] != 'admin':
        return jsonify({'message': 'Unauthorized'}), 403
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            SELECT u.id, u.name, u.email, u.role, u.created_at,
                   COUNT(DISTINCT pm.project_id) as projects_count,
                   AVG(g.grade) as average_grade
            FROM users u
            LEFT JOIN project_members pm ON u.id = pm.user_id
            LEFT JOIN grades g ON u.id = g.student_id
            GROUP BY u.id
        """)
        users = [{
            'id': row['id'],
            'name': row['name'],
            'email': row['email'],
            'role': row['role'],
            'created_at': row['created_at'],
            'projects_count': row['projects_count'],
            'average_grade': float(row['average_grade']) if row['average_grade'] else None
        } for row in c.fetchall()]
        return jsonify(users)
    except Exception as e:
        logger.error(f"Error getting users: {str(e)}")
        return jsonify({'message': 'Error getting users'}), 500
    finally:
        conn.close()
@app.route('/notifications', methods=['GET'])
@token_required
def get_notifications(current_user):
    try:
```

```
        conn = get_db()
        c = conn.cursor()
        # Отримуємо сповіщення з урахуванням терміну дії
        c.execute("""
            SELECT n.*, p.name as project_name
            FROM notifications n
            LEFT JOIN projects p ON n.project_id = p.id
            WHERE n.user_id = ?
            AND (n.expiry_date IS NULL OR n.expiry_date > datetime('now'))
            ORDER BY n.created_at DESC
            LIMIT 50
        """, (current_user['id'],))
        notifications = [{
            'id': row['id'],
            'type': row['type'],
            'message': row['message'],
            'project_id': row['project_id'],
            'project_name': row['project_name'],
            'created_at': row['created_at'],
            'is_read': bool(row['is_read']),
            'priority': row['priority']
        } for row in c.fetchall()]
        return jsonify(notifications)
    except Exception as e:
        logger.error(f"Error getting notifications: {str(e)}")
        return jsonify({'message': 'Error getting notifications'}), 500
    finally:
        conn.close()
@app.route('/notifications/mark-read', methods=['POST'])
@token_required
def mark_notifications_read(current_user):
    data = request.get_json()
    if not data or 'notification_ids' not in data:
        return jsonify({'message': 'Missing notification IDs'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        # Позначаємо сповіщення як прочитані
        for notification_id in data['notification_ids']:
            c.execute("""
                UPDATE notifications
                SET is_read = 1
                WHERE id = ? AND user_id = ?
            """, (notification_id, current_user['id']))
        conn.commit()
        return jsonify({'message': 'Notifications marked as read'})
    except Exception as e:
        logger.error(f"Error marking notifications as read: {str(e)}")
        return jsonify({'message': 'Error marking notifications as read'}), 500
    finally:
        conn.close()
@app.route('/notifications/unread-count', methods=['GET'])
@token_required
def get_unread_notifications_count(current_user):
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            SELECT COUNT(*) as count
            FROM notifications
            WHERE user_id = ? AND is_read = 0
            AND (expiry_date IS NULL OR expiry_date > datetime('now'))
        """, (current_user['id'],))
        result = c.fetchone()
        return jsonify({'unread_count': result['count']})
```

```
    except Exception as e:
        logger.error(f"Error getting unread notifications count: {str(e)}")
        return jsonify({'message': 'Error getting unread count'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/tasks', methods=['GET'])
@token_required
def get_tasks(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        c.execute("""
            SELECT t.*, u.name as assigned_user_name
            FROM tasks t
            LEFT JOIN users u ON t.assigned_to = u.id
            WHERE t.project_id = ?
            ORDER BY t.created_at DESC
        """, (project_id,))
        tasks = [{
            'id': row['id'],
            'title': row['title'],
            'description': row['description'],
            'status': row['status'],
            'priority': row['priority'],
            'created_at': row['created_at'],
            'deadline': row['deadline'],
            'assigned_to': row['assigned_to'],
            'assigned_user_name': row['assigned_user_name']
        } for row in c.fetchall()]
        return jsonify(tasks)
    except Exception as e:
        logger.error(f"Error getting tasks: {str(e)}")
        return jsonify({'message': 'Error getting tasks'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/tasks', methods=['POST'])
@token_required
def create_task(current_user, project_id):
    if current_user['role'] != 'manager':
        return jsonify({'message': 'Тільки менеджери можуть створювати завдання'}), 403
    data = request.get_json()
    required_fields = ['title', 'description', 'deadline']
    if not all(field in data for field in required_fields):
        return jsonify({'message': 'Відсутні обов\'язкові поля'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо чи є менеджер власником проєкту
        c.execute("""
            SELECT 1 FROM projects
            WHERE id = ? AND manager_id = ?
        """, (project_id, current_user['id']))
        if not c.fetchone():
            return jsonify({'message': 'Ви не маєте прав створювати завдання в цьому
проєкті'}), 403
        # Створюємо завдання
        c.execute("""
            INSERT INTO tasks (
                project_id, title, description, deadline,
                assigned_to, priority, status
            )
            VALUES (?, ?, ?, ?, ?, ?, 'not_started')
        """, (
            project_id, data['title'], data['description'],
            data['deadline'], data.get('assigned_to'),
```

```
            data.get('priority', 'medium')
        ))
        task_id = c.lastrowid
        # Якщо завдання призначено конкретному спеціалісту
        if data.get('assigned_to'):
            create_notification(
                data['assigned_to'],
                project_id,
                'task_assigned',
                f"Вам призначено нове завдання: {data['title']}"
            )
        else:
            # Сповіщаємо всіх учасників проєкту з роллю спеціаліст
            c.execute("""
                SELECT user_id FROM project_members
                WHERE project_id = ? AND role = 'specialist'
            """, (project_id,))
            for member in c.fetchall():
                create_notification(
                    member['user_id'],
                    project_id,
                    'new_task',
                    f"Додано нове завдання до проєкту: {data['title']}"
                )
        # Логуємо створення завдання
        log_activity(
            current_user['id'],
            project_id,
            'task_created',
            f"Створено завдання: {data['title']}"
        )
        conn.commit()
        return jsonify({
            'message': 'Завдання успішно створено',
            'task_id': task_id
        }), 201
    except Exception as e:
        logger.error(f"Помилка створення завдання: {str(e)}")
        return jsonify({'message': 'Помилка створення завдання'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/ratings', methods=['GET'])
@token_required
def get_ratings(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо права доступу
        if current_user['role'] == 'specialist':
            # Спеціаліст бачить тільки свої оцінки
            c.execute("""
                SELECT r.*, u.name as specialist_name, m.name as manager_name
                FROM ratings r
                JOIN users u ON r.specialist_id = u.id
                JOIN users m ON r.manager_id = m.id
                WHERE r.project_id = ? AND r.specialist_id = ?
            """, (project_id, current_user['id']))
        else:
            # Менеджер та адмін бачать всі оцінки
            c.execute("""
                SELECT r.*, u.name as specialist_name, m.name as manager_name
                FROM ratings r
                JOIN users u ON r.specialist_id = u.id
                JOIN users m ON r.manager_id = m.id
                WHERE r.project_id = ?
```

```
            """, (project_id,))
        ratings = [{
            'id': row['id'],
            'specialist_id': row['specialist_id'],
            'specialist_name': row['specialist_name'],
            'rating': row['rating'],
            'comment': row['comment'],
            'manager_name': row['manager_name'],
            'timestamp': row['timestamp'],
            'type': row['type']
        } for row in c.fetchall()]
        return jsonify(ratings)
    except Exception as e:
        logger.error(f"Помилка отримання оцінок: {str(e)}")
        return jsonify({'message': 'Помилка отримання оцінок'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/ratings', methods=['POST'])
@token_required
def add_rating(current_user, project_id):
    if current_user['role'] != 'manager':
        return jsonify({'message': 'Тільки менеджери можуть додавати оцінки'}), 403
    data = request.get_json()
    if not all(k in data for k in ['specialist_id', 'rating']):
        return jsonify({'message': 'Відсутні обов\'язкові поля'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо, чи є спеціаліст учасником проєкту
        c.execute("""
            SELECT 1 FROM project_members
            WHERE project_id = ? AND user_id = ? AND role = 'specialist'
        """, (project_id, data['specialist_id']))
        if not c.fetchone():
            return jsonify({'message': 'Спеціаліст не є учасником цього проєкту'}), 400
        # Додаємо або оновлюємо оцінку
        c.execute("""
            INSERT OR REPLACE INTO ratings (
                project_id, specialist_id, rating, comment,
                manager_id, timestamp, type
            )
            VALUES (?, ?, ?, ?, ?, ?, 'final')
        """, (
            project_id,
            data['specialist_id'],
            data['rating'],
            data.get('comment', ''),
            current_user['id'],
            datetime.now(timezone.utc)
        ))
        conn.commit()
        return jsonify({'message': 'Оцінку успішно додано'})
    except Exception as e:
        logger.error(f"Помилка додавання оцінки: {str(e)}")
        return jsonify({'message': 'Помилка додавання оцінки'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/tasks/<int:task_id>', methods=['PUT'])
@token_required
def update_task(current_user, project_id, task_id):
    data = request.get_json()
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо існування завдання
```

```
        c.execute("""
            SELECT t.*, p.manager_id
            FROM tasks t
            JOIN projects p ON t.project_id = p.id
            WHERE t.id = ? AND t.project_id = ?
        """, (task_id, project_id))
        task = c.fetchone()
        if not task:
            return jsonify({'message': 'Завдання не знайдено'}), 404
        # Перевіряємо права на редагування
        if current_user['role'] == 'manager' and task['manager_id'] != current_user['id']:
            return jsonify({'message': 'Недостатньо прав для оновлення цього завдання'}), 403
        # Оновлюємо завдання
        update_fields = []
        params = []
        if 'title' in data and current_user['role'] == 'manager':
            update_fields.append('title = ?')
            params.append(data['title'])
        if 'description' in data and current_user['role'] == 'manager':
            update_fields.append('description = ?')
            params.append(data['description'])
        if 'status' in data:
            update_fields.append('status = ?')
            params.append(data['status'])
        if 'assigned_to' in data and current_user['role'] == 'manager':
            update_fields.append('assigned_to = ?')
            params.append(data['assigned_to'])
        if 'priority' in data and current_user['role'] == 'manager':
            update_fields.append('priority = ?')
            params.append(data['priority'])
        if update_fields:
            params.extend([task_id, project_id])
            query = f"""
                UPDATE tasks
                SET {', '.join(update_fields)}
                WHERE id = ? AND project_id = ?
            """
            c.execute(query, params)
            # Створюємо сповіщення про оновлення
            create_notification(
                task['assigned_to'] if task['assigned_to'] else None,
                project_id,
                'task_updated',
                f"Завдання '{task['title']}' було оновлено"
            )
            # Логуємо зміни
            log_activity(
                current_user['id'],
                project_id,
                'task_updated',
                f"Оновлено завдання: {task['title']}"
            )
        conn.commit()
        return jsonify({'message': 'Завдання успішно оновлено'})
    except Exception as e:
        logger.error(f"Помилка оновлення завдання: {str(e)}")
        return jsonify({'message': 'Помилка оновлення завдання'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/files', methods=['POST'])
@token_required
def upload_file(current_user, project_id):
    if 'file' not in request.files:
        return jsonify({'message': 'No file part'}), 400
    file = request.files['file']
```

```
    if file.filename == '':
        return jsonify({'message': 'No selected file'}), 400
    if not allowed_file(file.filename):
        return jsonify({'message': 'File type not allowed'}), 400
    try:
        conn = get_db()
        c = conn.cursor()
        # Перевіряємо права доступу до проекту
        c.execute("""
            SELECT 1 FROM project_members
            WHERE project_id = ? AND user_id = ?
        """, (project_id, current_user['id']))
        if not c.fetchone() and current_user['role'] != 'admin':
            return jsonify({'message': 'Not a member of this project'}), 403
        # Створюємо директорію для файлів проекту
        project_dir = os.path.join(app.config['UPLOAD_FOLDER'], f'project_{project_id}')
        os.makedirs(project_dir, exist_ok=True)
        # Зберігаємо файл
        filename = secure_filename(file.filename)
        file_path = os.path.join(project_dir, filename)
        file.save(file_path)
        # Зберігаємо інформацію про файл в базі даних
        file_size = os.path.getsize(file_path)
        file_type = os.path.splitext(filename)[1]
        c.execute("""
            INSERT INTO files (
                project_id, user_id, filename,
                file_size, file_type, file_path
            )
            VALUES (?, ?, ?, ?, ?, ?)
        """, (
            project_id, current_user['id'], filename,
            file_size, file_type, file_path
        ))
        # Створюємо сповіщення про новий файл
        c.execute("""
            SELECT user_id FROM project_members
            WHERE project_id = ? AND user_id != ?
        """, (project_id, current_user['id']))
        for member in c.fetchall():
            create_notification(
                member['user_id'],
                project_id,
                'new_file',
                f"New file uploaded: {filename}"
            )
        # Логуємо завантаження файлу
        log_activity(
            current_user['id'],
            project_id,
            'file_uploaded',
            f"Uploaded file: {filename}"
        )
        conn.commit()
        return jsonify({'message': 'File uploaded successfully'})
    except Exception as e:
        logger.error(f"Error uploading file: {str(e)}")
        return jsonify({'message': 'Error uploading file'}), 500
    finally:
        conn.close()
@app.route('/files/<int:file_id>/download', methods=['GET'])
@token_required
def download_file(current_user, file_id):
    try:
        conn = get_db()
```

```
        c = conn.cursor()
        # Отримуємо інформацію про файл
        c.execute("""
            SELECT f.*, p.id as project_id
            FROM files f
            JOIN projects p ON f.project_id = p.id
            WHERE f.id = ?
        """, (file_id,))
        file_info = c.fetchone()
        if not file_info:
            return jsonify({'message': 'File not found'}), 404
        # Перевіряємо права доступу
        if current_user['role'] != 'admin':
            c.execute("""
                SELECT 1 FROM project_members
                WHERE project_id = ? AND user_id = ?
            """, (file_info['project_id'], current_user['id']))
            if not c.fetchone():
                return jsonify({'message': 'Not authorized to download this file'}), 403
        # Логуємо завантаження
        log_activity(
            current_user['id'],
            file_info['project_id'],
            'file_downloaded',
            f"Downloaded file: {file_info['filename']}"
        )
        return send_file(
            file_info['file_path'],
            as_attachment=True,
            download_name=file_info['filename']
        )
    except Exception as e:
        logger.error(f"Error downloading file: {str(e)}")
        return jsonify({'message': 'Error downloading file'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/statistics', methods=['GET'])
@token_required
def get_project_statistics(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        # Базова інформація про проект
        c.execute("""
            SELECT p.*,
                   COUNT(DISTINCT pm.user_id) as members_count,
                   COUNT(DISTINCT t.id) as total_tasks,
                   COUNT(DISTINCT CASE WHEN t.status = 'completed' THEN t.id END) as
completed_tasks,
                   AVG(r.rating) as average_rating
            FROM projects p
            LEFT JOIN project_members pm ON p.id = pm.project_id
            LEFT JOIN tasks t ON p.id = t.project_id
            LEFT JOIN ratings r ON p.id = r.project_id
            WHERE p.id = ?
            GROUP BY p.id
        """, (project_id,))
        project_stats = c.fetchone()
        if not project_stats:
            return jsonify({'message': 'Проєкт не знайдено'}), 404
        # Прогрес за останній місяць
        c.execute("""
            SELECT DATE(created_at) as date,
                   COUNT(CASE WHEN status = 'completed' THEN 1 END) * 100.0 / COUNT(*) as
completion_rate
```

```
            FROM tasks
            WHERE project_id = ?
            AND created_at >= date('now', '-30 days')
            GROUP BY DATE(created_at)
            ORDER BY date
        """, (project_id,))
        progress_history = [{
            'date': row['date'],
            'completion_rate': row['completion_rate']
        } for row in c.fetchall()]
        # Статистика по учасниках
        c.execute("""
            SELECT u.name,
                   COUNT(DISTINCT t.id) as assigned_tasks,
                   COUNT(DISTINCT CASE WHEN t.status = 'completed' THEN t.id END) as
completed_tasks,
                   r.rating as current_rating
            FROM project_members pm
            JOIN users u ON pm.user_id = u.id
            LEFT JOIN tasks t ON t.assigned_to = u.id AND t.project_id = pm.project_id
            LEFT JOIN ratings r ON r.specialist_id = u.id AND r.project_id = pm.project_id
            WHERE pm.project_id = ? AND pm.role = 'specialist'
            GROUP BY u.id
        """, (project_id,))
        members_stats = [{
            'name': row['name'],
            'assigned_tasks': row['assigned_tasks'],
            'completed_tasks': row['completed_tasks'],
            'current_rating': row['current_rating']
        } for row in c.fetchall()]
        statistics = {
            'basic_info': {
                'members_count': project_stats['members_count'],
                'total_tasks': project_stats['total_tasks'],
                'completed_tasks': project_stats['completed_tasks'],
                'average_rating': float(project_stats['average_rating']) if
project_stats['average_rating'] else None
            },
            'progress_history': progress_history,
            'members_statistics': members_stats
        }
        return jsonify(statistics)
    except Exception as e:
        logger.error(f"Помилка отримання статистики проєкту: {str(e)}")
        return jsonify({'message': 'Помилка отримання статистики проєкту'}), 500
    finally:
        conn.close()
@app.route('/projects/<int:project_id>/activity', methods=['GET'])
@token_required
def get_project_activity(current_user, project_id):
    try:
        conn = get_db()
        c = conn.cursor()
        # Отримуємо останні дії в проекті
        c.execute("""
            SELECT ua.*, u.name as user_name
            FROM user_activity ua
            JOIN users u ON ua.user_id = u.id
            WHERE ua.project_id = ?
            ORDER BY ua.timestamp DESC
            LIMIT 50
        """, (project_id,))
        activities = [{
            'id': row['id'],
            'user_name': row['user_name'],
```

```
            'action_type': row['action_type'],
            'action_details': row['action_details'],
            'timestamp': row['timestamp']
        } for row in c.fetchall()]
        return jsonify(activities)
    except Exception as e:
        logger.error(f"Error getting project activity: {str(e)}")
        return jsonify({'message': 'Error getting project activity'}), 500
    finally:
        conn.close()
@app.route('/users/<int:user_id>', methods=['DELETE'])
@token_required
def delete_user(current_user, user_id):
    if current_user['role'] != 'admin':
        return jsonify({'message': 'Unauthorized'}), 403
    try:
        conn = get_db()
        c = conn.cursor()
        # Проверяем существование пользователя
        c.execute("SELECT role FROM users WHERE id = ?", (user_id,))
        user = c.fetchone()
        if not user:
            return jsonify({'message': 'User not found'}), 404
        # Удаляем связанные данные
        c.execute("DELETE FROM project_members WHERE user_id = ?", (user_id,))
        c.execute("DELETE FROM tasks WHERE assigned_to = ?", (user_id,))
        c.execute("DELETE FROM comments WHERE user_id = ?", (user_id,))
        c.execute("DELETE FROM grades WHERE student_id = ? OR teacher_id = ?", (user_id,
user_id))
        c.execute("DELETE FROM notifications WHERE user_id = ?", (user_id,))
        c.execute("DELETE FROM user_activity WHERE user_id = ?", (user_id,))
        # Удаляем самого пользователя
        c.execute("DELETE FROM users WHERE id = ?", (user_id,))
        conn.commit()
        return jsonify({'message': 'User deleted successfully'})
    except Exception as e:
        logger.error(f"Error deleting user: {str(e)}")
        return jsonify({'message': 'Error deleting user'}), 500
    finally:
        conn.close()
@app.route('/users/<int:user_id>/statistics', methods=['GET'])
@token_required
def get_user_statistics(current_user, user_id):
    # Перевіряємо права доступу
    if current_user['id'] != user_id and current_user['role'] not in ['teacher', 'admin']:
        return jsonify({'message': 'Unauthorized'}), 403
    try:
        conn = get_db()
        c = conn.cursor()
        # Загальна статистика
        c.execute("""
            SELECT
                COUNT(DISTINCT pm.project_id) as total_projects,
                COUNT(DISTINCT t.id) as total_tasks,
                COUNT(DISTINCT CASE WHEN t.status = 'completed' THEN t.id END) as
completed_tasks,
                AVG(g.grade) as average_grade
            FROM users u
            LEFT JOIN project_members pm ON u.id = pm.user_id
            LEFT JOIN tasks t ON u.id = t.assigned_to
            LEFT JOIN grades g ON u.id = g.student_id
            WHERE u.id = ?
        """, (user_id,))
        stats = c.fetchone()
        # Активні проекти
```

```
        c.execute("""
            SELECT p.name,
                   p.deadline,
                   COUNT(t.id) as tasks_count,
                   COUNT(CASE WHEN t.status = 'completed' THEN 1 END) as completed_tasks,
                   g.grade
            FROM project_members pm
            JOIN projects p ON pm.project_id = p.id
            LEFT JOIN tasks t ON t.project_id = p.id AND t.assigned_to = pm.user_id
            LEFT JOIN grades g ON g.project_id = p.id AND g.student_id = pm.user_id
            WHERE pm.user_id = ? AND p.status = 'active'
            GROUP BY p.id
        """, (user_id,))
        active_projects = [{
            'name': row['name'],
            'deadline': row['deadline'],
            'tasks_count': row['tasks_count'],
            'completed_tasks': row['completed_tasks'],
            'grade': row['grade']
        } for row in c.fetchall()]
        # Останні дії
        c.execute("""
            SELECT action_type, action_details, timestamp
            FROM user_activity
            WHERE user_id = ?
            ORDER BY timestamp DESC
            LIMIT 10
        """, (user_id,))
        recent_activity = [{
            'action_type': row['action_type'],
            'action_details': row['action_details'],
            'timestamp': row['timestamp']
        } for row in c.fetchall()]
        statistics = {
            'overview': {
                'total_projects': stats['total_projects'],
                'total_tasks': stats['total_tasks'],
                'completed_tasks': stats['completed_tasks'],
                'average_grade': float(stats['average_grade']) if stats['average_grade'] else
None
            },
            'active_projects': active_projects,
            'recent_activity': recent_activity
        }
        return jsonify(statistics)
    except Exception as e:
        logger.error(f"Error getting user statistics: {str(e)}")
        return jsonify({'message': 'Error getting user statistics'}), 500
    finally:
        conn.close()
@app.errorhandler(404)
def not_found_error(error):
    return jsonify({'message': 'Resource not found'}), 404
@app.errorhandler(500)
def internal_error(error):
    return jsonify({'message': 'Internal server error'}), 500
 Ініціалізація при запуску
if __name__ == '__main__':
    with app.app_context():
        init_db()
    app.run(debug=True)
```

seed.py:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3
import datetime
import random
from werkzeug.security import generate_password_hash
import os
import logging
from faker import Faker
import uuid
# Налаштування логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)
# Ініціалізація Faker для генерації реалістичних даних
fake = Faker('uk_UA')  # Використовуємо українську локаль
# Шлях до бази даних
DB_PATH = 'project_management.db'
# Створення директорії для файлів
UPLOAD_DIR = 'files'
if not os.path.exists(UPLOAD_DIR):
    os.makedirs(UPLOAD_DIR)
    logger.info(f"Створено директорію {UPLOAD_DIR}")
def get_db_connection():
    """Отримання з'єднання з базою даних"""
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn
def clear_database():
    """Очищення бази даних перед заповненням"""
    logger.info("Очищення бази даних...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Список таблиць для очищення
    tables = [
        'notifications', 'user_activity', 'files',
        'comments', 'tasks', 'calendar_events',
        'ratings', 'project_members', 'projects',
        'users'
    ]
    # Відключення перевірки зовнішніх ключів
    cursor.execute('PRAGMA foreign_keys = OFF')
    # Видалення даних з таблиць
    for table in tables:
        try:
            cursor.execute(f'DELETE FROM {table}')
            logger.info(f"Очищено таблицю {table}")
        except sqlite3.Error as e:
            logger.warning(f"Помилка при очищенні таблиці {table}: {e}")
    # Скидання лічильників автоінкременту
    for table in tables:
        try:
            cursor.execute(f'DELETE FROM sqlite_sequence WHERE name="{table}"')
        except sqlite3.Error:
            pass
    # Включення перевірки зовнішніх ключів
    cursor.execute('PRAGMA foreign_keys = ON')
    conn.commit()
    conn.close()
    logger.info("База даних очищена")
def create_users():
    """Створення користувачів: адміністратор, менеджери та спеціалісти"""
    logger.info("Створення користувачів...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Створення адміністратора
```

```
    admin_password = generate_password_hash('admin123', method='scrypt')
    cursor.execute(
        'INSERT INTO users (name, email, password, role, created_at) VALUES (?, ?, ?, ?, ?)',
        ('Адміністратор', 'admin@example.com', admin_password, 'admin', datetime.datetime.now())
    )
    logger.info("Адміністратор створений")
    # Створення менеджерів (3)
    managers = []
    for i in range(3):
        name = fake.name()
        email = f"manager{i+1}@example.com"
        password = generate_password_hash(f'manager{i+1}', method='scrypt')
        cursor.execute(
            'INSERT INTO users (name, email, password, role, created_at) VALUES (?, ?, ?, ?,
?)',
            (name, email, password, 'manager', datetime.datetime.now())
        )
        manager_id = cursor.lastrowid
        managers.append({
            'id': manager_id,
            'name': name,
            'email': email
        })
    logger.info(f"Створено {len(managers)} менеджерів")
    # Створення спеціалістів (10)
    specialists = []
    for i in range(10):
        name = fake.name()
        email = f"specialist{i+1}@example.com"
        password = generate_password_hash(f'specialist{i+1}', method='scrypt')
        cursor.execute(
            'INSERT INTO users (name, email, password, role, created_at) VALUES (?, ?, ?, ?,
?)',
            (name, email, password, 'specialist', datetime.datetime.now())
        )
        specialist_id = cursor.lastrowid
        specialists.append({
            'id': specialist_id,
            'name': name,
            'email': email
        })
    logger.info(f"Створено {len(specialists)} спеціалістів")
    conn.commit()
    conn.close()
    return {
        'managers': managers,
        'specialists': specialists
    }
def create_projects(users):
    """Створення проєктів"""
    logger.info("Створення проєктів...")
    conn = get_db_connection()
    cursor = conn.cursor()
    managers = users['managers']
    specialists = users['specialists']
    # Типові статуси проєктів
    statuses = ['active', 'completed', 'archived']
    # Створення 20 проєктів
    projects = []
    for i in range(20):
        # Вибір випадкового менеджера
        manager = random.choice(managers)
        # Генерація даних проєкту
        name = fake.catch_phrase()
        description = fake.paragraph(nb_sentences=5)
```

```
            status = random.choices(statuses, weights=[0.7, 0.2, 0.1])[0]
            # Генерація дати дедлайну (від 1 до 6 місяців від сьогодні)
            deadline = datetime.datetime.now() + datetime.timedelta(days=random.randint(30, 180))
            # Максимальна кількість спеціалістів (від 3 до 7)
            max_specialists = random.randint(3, 7)
            # Вставка проєкту
            cursor.execute(
                '''INSERT INTO projects
                   (name, description, manager_id, deadline, max_specialists, status, created_at)
                   VALUES (?, ?, ?, ?, ?, ?, ?)''',
                (name, description, manager['id'], deadline.strftime('%Y-%m-%d'),
                 max_specialists, status, datetime.datetime.now())
            )
            project_id = cursor.lastrowid
            # Додавання менеджера як учасника проєкту
            cursor.execute(
                '''INSERT INTO project_members
                   (project_id, user_id, role, join_date)
                   VALUES (?, ?, ?, ?)''',
                (project_id, manager['id'], 'manager', datetime.datetime.now())
            )
            # Визначення кількості спеціалістів для проєкту (від 0 до max_specialists)
            num_specialists = random.randint(0, min(max_specialists, len(specialists)))
            # Випадкові спеціалісти для цього проєкту
            project_specialists = random.sample(specialists, num_specialists)
            # Додавання спеціалістів до проєкту
            for specialist in project_specialists:
                cursor.execute(
                    '''INSERT INTO project_members
                       (project_id, user_id, role, join_date)
                       VALUES (?, ?, ?, ?)''',
                    (project_id, specialist['id'], 'specialist',
                     (datetime.datetime.now() - datetime.timedelta(days=random.randint(1, 30))))
                )
            # Зберігання інформації про проєкт
            projects.append({
                'id': project_id,
                'name': name,
                'manager_id': manager['id'],
                'status': status,
                'specialists': [s['id'] for s in project_specialists]
            })
        logger.info(f"Створено {len(projects)} проєктів")
        conn.commit()
        conn.close()
        return projects
    def create_tasks(projects, users):
        """Створення завдань для проєктів"""
        logger.info("Створення завдань...")
        conn = get_db_connection()
        cursor = conn.cursor()
        # Статуси завдань
        task_statuses = ['not_started', 'in_progress', 'completed']
        # Пріоритети завдань
        priorities = ['low', 'medium', 'high']
        # Лічильник створених завдань
        task_count = 0
        for project in projects:
            # Генеруємо випадкову кількість завдань для проєкту (від 0 до 5)
            num_tasks = random.randint(0, 5)
            # Якщо проєкт неактивний, менше завдань
            if project['status'] != 'active':
                num_tasks = min(num_tasks, 2)
            # Спеціалісти в цьому проєкті
            project_specialists = project['specialists']
```

```
        for i in range(num_tasks):
            # Генерація даних завдання
            title = fake.sentence(nb_words=6)
            description = fake.paragraph(nb_sentences=3)
            # Випадковий статус, з перевагою для 'in_progress' для активних проєктів
            if project['status'] == 'active':
                status = random.choices(task_statuses, weights=[0.3, 0.5, 0.2])[0]
            else:
                status = random.choices(task_statuses, weights=[0.1, 0.2, 0.7])[0]
            # Дедлайн завдання
            task_deadline = datetime.datetime.now() + datetime.timedelta(days=random.randint(7,
60))
            # Випадковий пріоритет
            priority = random.choice(priorities)
            # Призначення завдання випадковому спеціалісту з проєкту (якщо такі є)
            assigned_to = None
            if project_specialists:
                assigned_to = random.choice(project_specialists)
            # Вставка завдання
            cursor.execute(
                '''INSERT INTO tasks
                   (project_id, title, description, status, created_at,
                    deadline, assigned_to, priority)
                   VALUES (?, ?, ?, ?, ?, ?, ?, ?)''',
                (project['id'], title, description, status,
                 datetime.datetime.now(), task_deadline.strftime('%Y-%m-%d'),
                 assigned_to, priority)
            )
            task_count += 1
    logger.info(f"Створено {task_count} завдань")
    conn.commit()
    conn.close()
def create_calendar_events(projects):
    """Створення подій календаря для проєктів"""
    logger.info("Створення подій календаря...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Типи подій
    event_types = ['meeting', 'deadline', 'other']
    # Лічильник створених подій
    event_count = 0
    for project in projects:
        # Тільки для активних проєктів
        if project['status'] != 'active':
            continue
        # Генеруємо випадкову кількість подій (від 1 до 3)
        num_events = random.randint(1, 3)
        for i in range(num_events):
            # Генерація даних події
            title = fake.sentence(nb_words=4)
            description = fake.paragraph(nb_sentences=2)
            event_type = random.choice(event_types)
            # Дата початку (від сьогодні до 30 днів у майбутньому)
            start_date = datetime.datetime.now() + datetime.timedelta(days=random.randint(1,
30))
            # Тривалість події (від 30 хвилин до 3 годин)
            duration = datetime.timedelta(minutes=random.randint(30, 180))
            # Дата закінчення
            end_date = start_date + duration
            # Вставка події
            cursor.execute(
                '''INSERT INTO calendar_events
                   (project_id, title, description, event_type,
                    start_time, end_time, created_by)
                   VALUES (?, ?, ?, ?, ?, ?, ?)''',
```

```
                (project['id'], title, description, event_type,
                 start_date.strftime('%Y-%m-%d %H:%M:%S'),
                 end_date.strftime('%Y-%m-%d %H:%M:%S'),
                 project['manager_id'])
            )
            event_count += 1
    logger.info(f"Створено {event_count} подій календаря")
    conn.commit()
    conn.close()
def create_comments(projects, users):
    """Створення коментарів для проєктів"""
    logger.info("Створення коментарів...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Лічильник створених коментарів
    comment_count = 0
    for project in projects:
        # Генеруємо випадкову кількість коментарів (від 0 до 8)
        num_comments = random.randint(0, 8)
        # Учасники проєкту (включаючи менеджера)
        participants = [project['manager_id']] + project['specialists']
        if not participants:
            continue
        # Дати для впорядкування коментарів
        base_date = datetime.datetime.now() - datetime.timedelta(days=random.randint(10, 30))
        for i in range(num_comments):
            # Випадковий автор коментаря
            author_id = random.choice(participants)
            # Генерація тексту коментаря
            content = fake.paragraph(nb_sentences=random.randint(1, 3))
            # Дата коментаря (хронологічно)
            comment_date = base_date + datetime.timedelta(hours=i*random.randint(1, 5))
            # Вставка коментаря
            cursor.execute(
                '''INSERT INTO comments
                   (project_id, user_id, content, timestamp)
                   VALUES (?, ?, ?, ?)''',
                (project['id'], author_id, content, comment_date.strftime('%Y-%m-%d %H:%M:%S'))
            )
            comment_count += 1
    logger.info(f"Створено {comment_count} коментарів")
    conn.commit()
    conn.close()
def create_ratings(projects, users):
    """Створення оцінок виконання для проєктів"""
    logger.info("Створення оцінок виконання...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Лічильник створених оцінок
    rating_count = 0
    for project in projects:
        # Тільки для проєктів з статусом "completed" чи для деяких активних
        if project['status'] != 'completed' and random.random() > 0.3:
            continue
        # Спеціалісти в проєкті
        project_specialists = project['specialists']
        # Оцінюємо кожного спеціаліста
        for specialist_id in project_specialists:
            # Генерація оцінки (від 60 до 100)
            rating_value = round(random.uniform(60, 100), 1)
            # Генерація коментаря до оцінки
            comment = fake.sentence(nb_words=random.randint(5, 15))
            # Дата оцінювання
            rating_date = datetime.datetime.now() - datetime.timedelta(days=random.randint(1,
14))
```

```
            # Вставка оцінки
            cursor.execute(
                '''INSERT INTO ratings
                   (project_id, specialist_id, rating, comment,
                    manager_id, timestamp, type)
                   VALUES (?, ?, ?, ?, ?, ?, ?)''',
                (project['id'], specialist_id, rating_value, comment,
                 project['manager_id'], rating_date.strftime('%Y-%m-%d %H:%M:%S'),
                 'final')
            )
            rating_count += 1
    logger.info(f"Створено {rating_count} оцінок виконання")
    conn.commit()
    conn.close()
def create_notifications(projects, users):
    """Створення сповіщень для користувачів"""
    logger.info("Створення сповіщень...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Типи сповіщень
    notification_types = [
        'task_assigned', 'task_completed', 'new_comment',
        'new_member', 'project_update', 'deadline_approaching'
    ]
    # Пріоритети сповіщень
    priorities = ['low', 'normal', 'high']
    # Лічильник створених сповіщень
    notification_count = 0
    # Для кожного користувача створюємо кілька сповіщень
    all_users = []
    for manager in users['managers']:
        all_users.append(manager['id'])
    for specialist in users['specialists']:
        all_users.append(specialist['id'])
    for user_id in all_users:
        # Випадкова кількість сповіщень для користувача (від 0 до 5)
        num_notifications = random.randint(0, 5)
        for i in range(num_notifications):
            # Випадковий проєкт
            project = random.choice(projects)
            # Випадковий тип сповіщення
            notification_type = random.choice(notification_types)
            # Генерація тексту сповіщення
            message = fake.sentence(nb_words=random.randint(5, 10))
            # Випадковий пріоритет
            priority = random.choice(priorities)
            # Дата створення (останні 7 днів)
            created_at = datetime.datetime.now() - datetime.timedelta(days=random.randint(0,
7))
            # Дата закінчення (від 7 до 30 днів у майбутньому)
            expiry_date = datetime.datetime.now() + datetime.timedelta(days=random.randint(7,
30))
            # Статус прочитання (більшість непрочитані)
            is_read = random.choices([0, 1], weights=[0.7, 0.3])[0]
            # Вставка сповіщення
            cursor.execute(
                '''INSERT INTO notifications
                   (user_id, project_id, type, message, created_at,
                    is_read, priority, expiry_date)
                   VALUES (?, ?, ?, ?, ?, ?, ?, ?)''',
                (user_id, project['id'], notification_type, message,
                 created_at.strftime('%Y-%m-%d %H:%M:%S'),
                 is_read, priority, expiry_date.strftime('%Y-%m-%d %H:%M:%S'))
            )
            notification_count += 1
```

```
    logger.info(f"Створено {notification_count} сповіщень")
    conn.commit()
    conn.close()
def create_files(projects, users):
    """Створення файлів для проєктів"""
    logger.info("Створення файлів...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Типи файлів
    file_types = ['.pdf', '.doc', '.xlsx', '.png', '.jpg', '.txt']
    # Лічильник створених файлів
    file_count = 0
    for project in projects:
        # Створюємо директорію для файлів проєкту
        project_dir = os.path.join(UPLOAD_DIR, f'project_{project["id"]}')
        os.makedirs(project_dir, exist_ok=True)
        # Випадкова кількість файлів (від 0 до 3)
        num_files = random.randint(0, 3)
        # Учасники проєкту
        participants = [project['manager_id']] + project['specialists']
        if not participants:
            continue
        for i in range(num_files):
            # Випадковий автор файлу
            author_id = random.choice(participants)
            # Генерація імені файлу
            file_type = random.choice(file_types)
            filename = fake.word() + "_" + str(uuid.uuid4())[:8] + file_type
            # Розмір файлу (від 10KB до 5MB)
            file_size = random.randint(10000, 5000000)
            # Шлях до файлу
            file_path = os.path.join(project_dir, filename)
            # Створення пустого файлу
            with open(file_path, 'wb') as f:
                f.write(b'x' * min(file_size, 100))  # Записуємо тільки початок для економії
місця
            # Дата завантаження
            upload_date = datetime.datetime.now() - datetime.timedelta(days=random.randint(0,
14))
            # Вставка запису про файл
            cursor.execute(
                '''INSERT INTO files
                   (project_id, user_id, filename, file_size, file_type,
                    upload_date, file_path)
                   VALUES (?, ?, ?, ?, ?, ?, ?)''',
                (project['id'], author_id, filename, file_size, file_type,
                 upload_date.strftime('%Y-%m-%d %H:%M:%S'), file_path)
            )
            file_count += 1
    logger.info(f"Створено {file_count} файлів")
    conn.commit()
    conn.close()
def create_activity_logs(projects, users):
    """Створення логів активності"""
    logger.info("Створення логів активності...")
    conn = get_db_connection()
    cursor = conn.cursor()
    # Типи активності
    activity_types = [
        'login', 'file_upload', 'comment_added', 'task_created',
        'task_updated', 'project_joined', 'rating_given'
    ]
    # Лічильник створених логів
    log_count = 0
    # Для кожного користувача створюємо записи активності
```

```
    all_users = []
    for manager in users['managers']:
        all_users.append(manager['id'])
    for specialist in users['specialists']:
        all_users.append(specialist['id'])
    for user_id in all_users:
        # Випадкова кількість записів активності (від 3 до 15)
        num_logs = random.randint(3, 15)
        for i in range(num_logs):
            # Випадковий тип активності
            activity_type = random.choice(activity_types)
            # Детальна інформація про активність
            action_details = fake.sentence(nb_words=random.randint(3, 8))
            # Випадковий проєкт (або None для деяких типів активності)
            project_id = None
            if activity_type != 'login' and random.random() > 0.2:
                project = random.choice(projects)
                project_id = project['id']
            # Дата активності (останні 30 днів)
            timestamp = datetime.datetime.now() - datetime.timedelta(days=random.randint(0,
30),
                                                                     hours=random.randint(0, 23),
                                                                     minutes=random.randint(0, 59))
            # Вставка запису активності
            cursor.execute(
                '''INSERT INTO user_activity
                   (user_id, project_id, action_type, action_details, timestamp)
                   VALUES (?, ?, ?, ?, ?)''',
                (user_id, project_id, activity_type, action_details,
                 timestamp.strftime('%Y-%m-%d %H:%M:%S'))
            )
            log_count += 1
    logger.info(f"Створено {log_count} записів активності")
    conn.commit()
    conn.close()
def main():
    """Головна функція для заповнення бази даних"""
    logger.info("Початок заповнення бази даних...")
    # Очищення бази даних
    clear_database()
    # Створення користувачів
    users = create_users()
    # Створення проєктів
    projects = create_projects(users)
    # Створення завдань
    create_tasks(projects, users)
    # Створення подій календаря
    create_calendar_events(projects)
    # Створення коментарів
    create_comments(projects, users)
    # Створення оцінок
    create_ratings(projects, users)
    # Створення сповіщень
    create_notifications(projects, users)
    # Створення файлів
    create_files(projects, users)
    # Створення логів активності
    create_activity_logs(projects, users)
    logger.info("Заповнення бази даних завершено")
    logger.info("\nОблікові дані для входу:")
    logger.info("Адміністратор: admin@example.com / admin123")
    for i, manager in enumerate(users['managers']):
        logger.info(f"Менеджер {i+1}: {manager['email']} / manager{i+1}")
    for i, specialist in enumerate(users['specialists']):
        logger.info(f"Спеціаліст {i+1}: {specialist['email']} / specialist{i+1}")
```

```
if __name__ == "__main__":
    main()
```

ДОДАТОК Б.

UML-ДІАГРАМИ

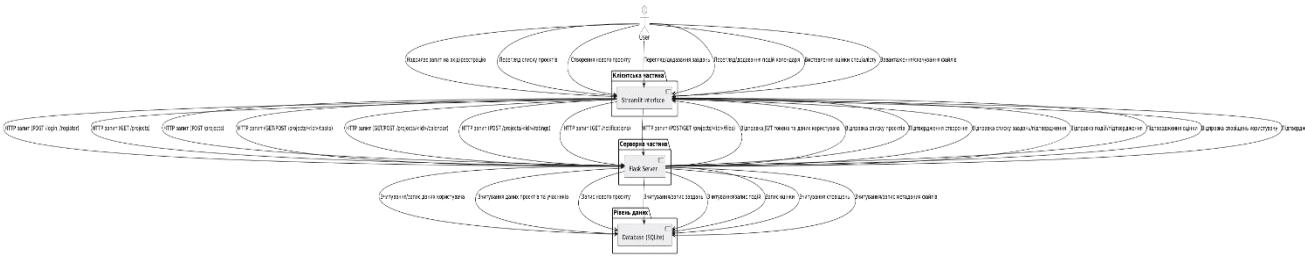

Рисунок Б.1. Діаграма компонентів

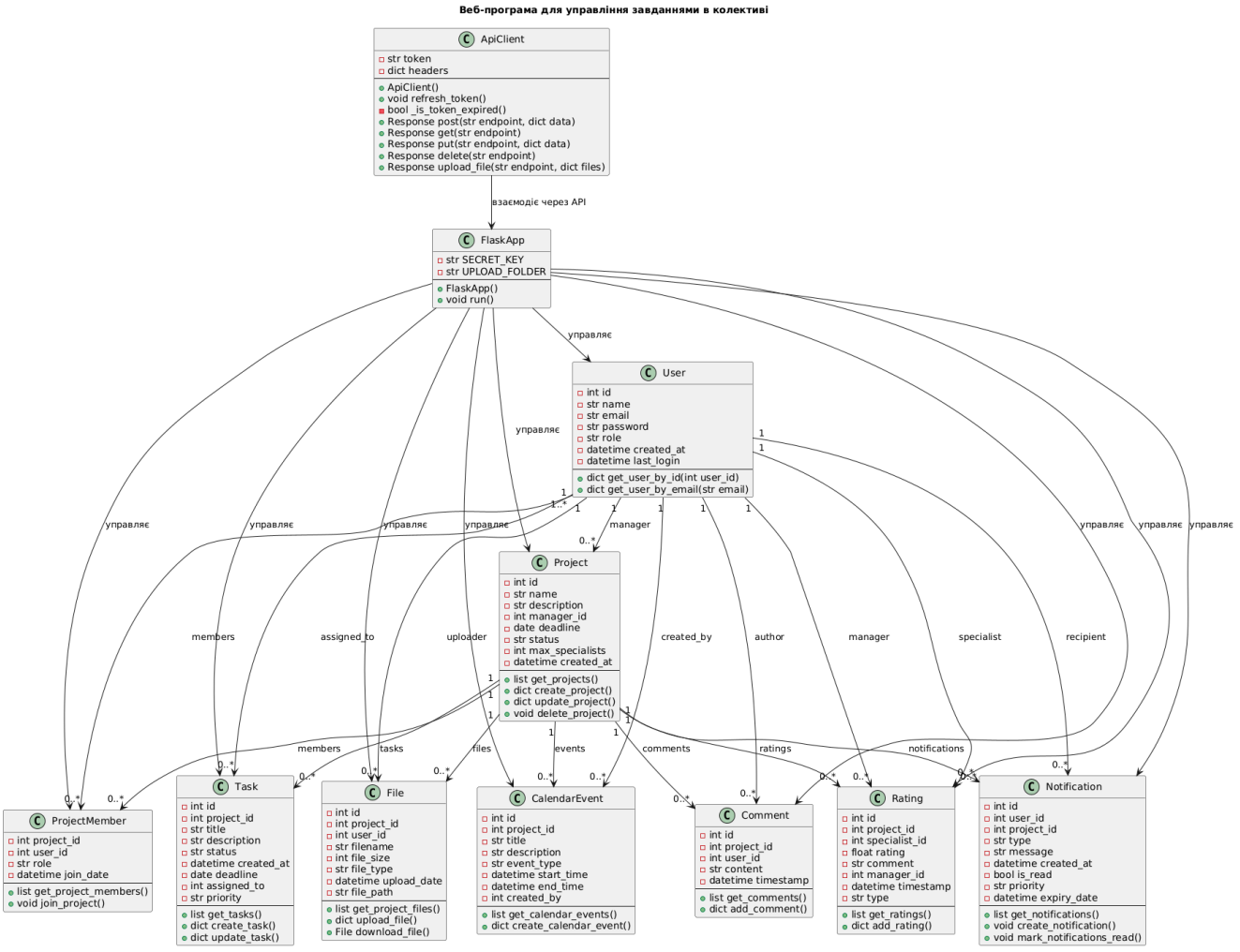

Рисунок Б.2. Діаграма класів

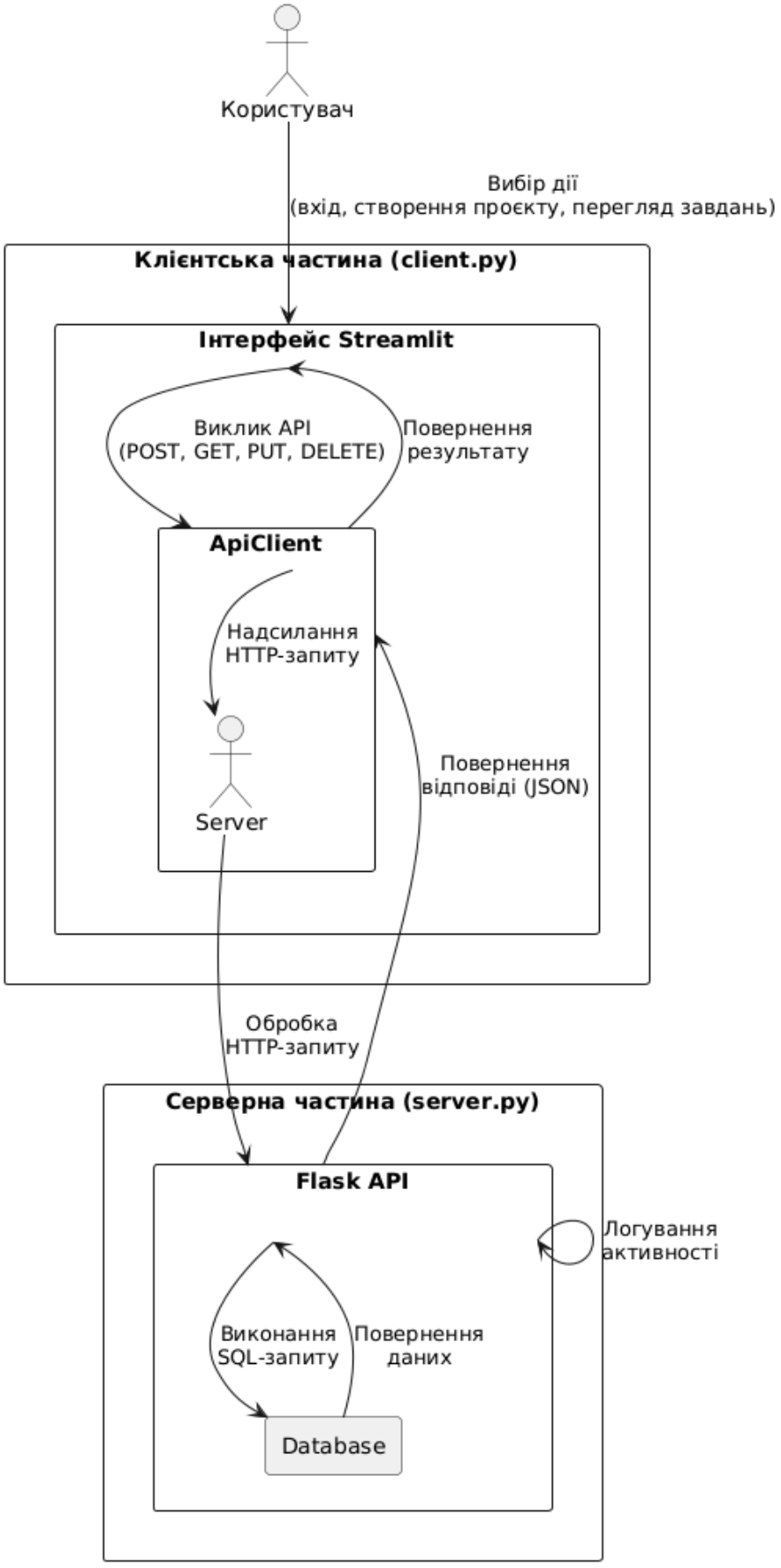

Рисунок Б.3. Діаграма кооперацій

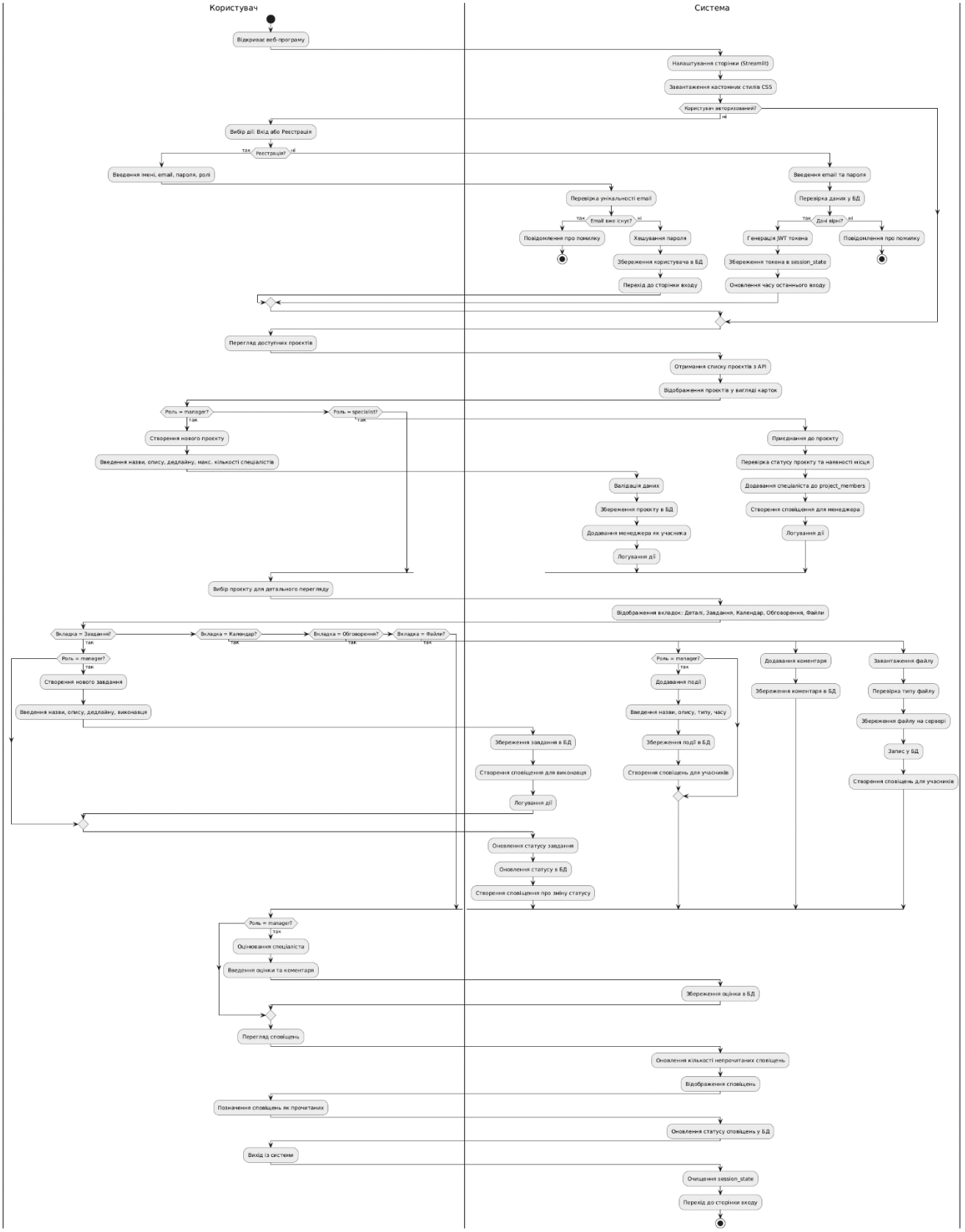

Рисунок Б.4. Діаграма діяльності

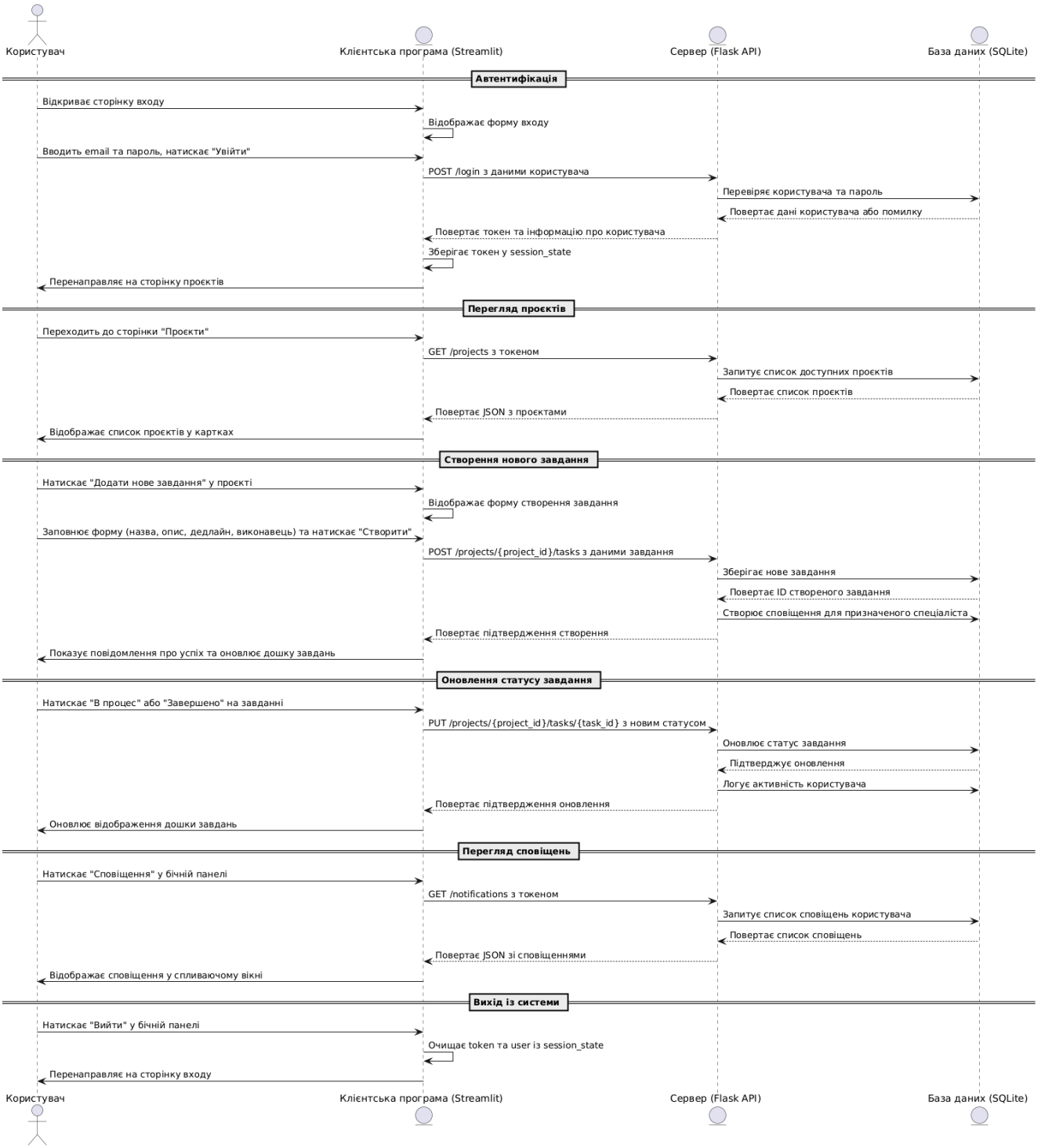

Рисунок Б.5. Діаграма послідовності

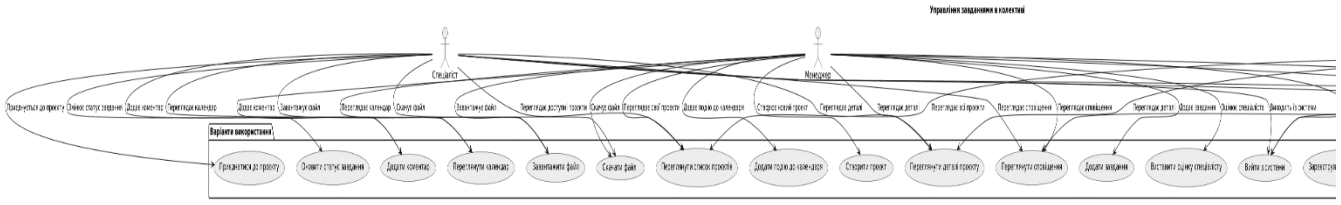

Рисунок Б.6. Діаграма прецедентів

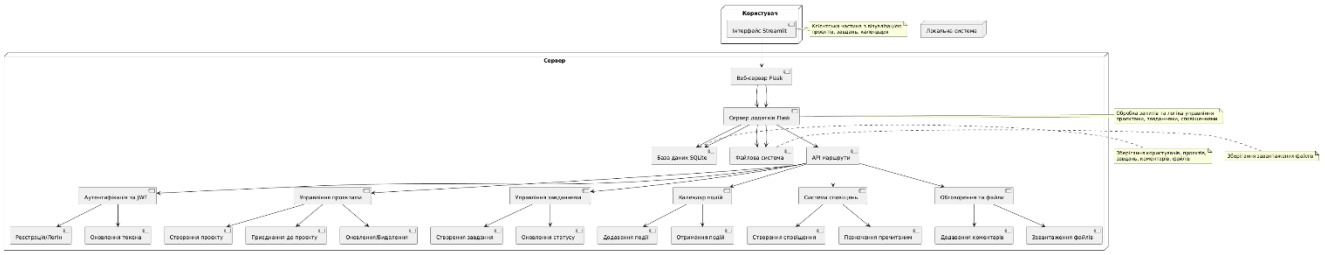

Рисунок Б.7. Діаграма розгортання

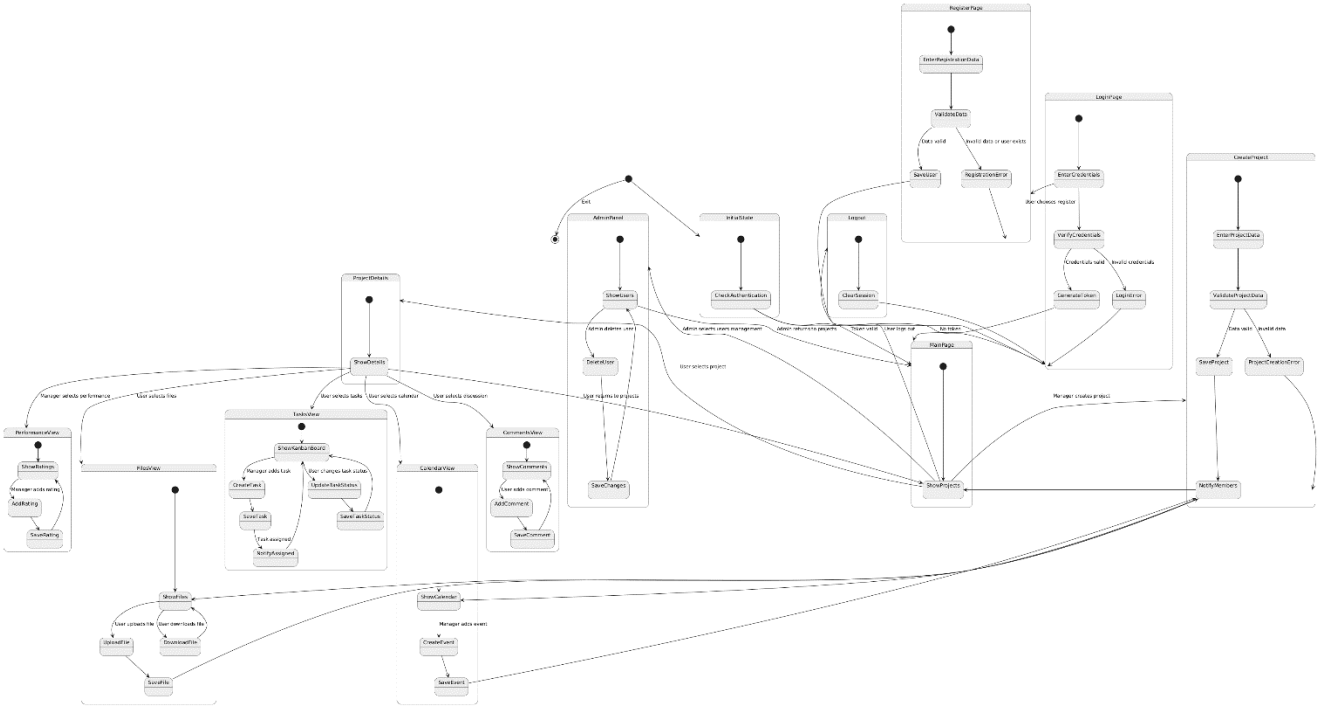

Рисунок Б.8. Діаграма станів