

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему СТВОРЕННЯ ПРОГРАМНОЇ ВЕБСИСТЕМИ ДЛЯ
УПРАВЛІННЯ ЗАВДАННЯМИ З ВИКОРИСТАННЯМ
AGILE-ПІДХОДІВ

DEVELOPMENT OF A WEB APPLICATION FOR TASK
MANAGEMENT USING AGILE APPROACHES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-222

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

АРТЮХ В.О.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

АРТЮХ Вікторії Олександрівни

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Створення програмної вебсистеми для управління завданнями з використанням Agile-підходів. Development of a Web Application for Task Management Using Agile Approaches

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та існуючих рішень. 2. Проєктування вебзастосунку для управління завданнями. 3. Реалізація вебзастосунку. 4. Тестування та оцінка ефективності системи.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Вікторія АРТЮХ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
81 с., 5 табл., 45 рис., 2 дод., 33 джерела.

СИСТЕМА УПРАВЛІННЯ ЗАВДАННЯМИ, WEB-ЗАСТОСУНОК,
KANBAN-ДОШКА, DJANGO, PYTHON, БАЗА ДАНИХ, АВТОРИЗАЦІЯ
КОРИСТУВАЧІВ, УПРАВЛІННЯ ПРОЄКТАМИ.

Об'єкт дослідження - процес розроблення веборієнтованого програмного забезпечення для організації та управління завданнями в межах командної роботи над проєктами.

Предмет дослідження - методи, засоби та технології розроблення веборієнтованих систем управління завданнями.

Мета роботи - підвищення ефективності організації та контролю виконання завдань у межах проєктної діяльності шляхом розроблення веборієнтованої системи управління завданнями.

Матеріали, методи та технічні засоби: методи об'єктно-орієнтованого програмування, клієнт-серверна архітектура, фреймворк Django, мови програмування Python, HTML, CSS та JavaScript, база даних SQLite, середовище Visual Studio Code та операційна система Windows.

Результати. Розроблено вебзастосунок для управління завданнями з підтримкою Kanban-дошки, авторизації користувачів, призначення та редагування задач, зміни їх статусів і контролю виконання робіт. Розроблена система сприяє підвищенню ефективності управління завданнями та покращенню взаємодії між учасниками проєкту.

Галузь використання - ІТ-компанії, навчальні заклади, команди розробників програмного забезпечення та інші організації, діяльність яких пов'язана з управлінням завданнями та проєктами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 81 pages, 5 tables, 45 figures, 2 appendixes, 33 sources.

TASK MANAGEMENT SYSTEM, WEB APPLICATION, KANBAN BOARD, DJANGO, PYTHON, DATABASE, USER AUTHORIZATION, PROJECT MANAGEMENT.

Object of research - the process of developing web-based software for organizing and managing tasks within collaborative project work.

Subject of research - methods, tools, and technologies for developing web-based task management systems.

Purpose of the work - to improve the efficiency of task organization and execution control within project activities through the development of a web-based task management system.

Materials, methods, and technical tools: object-oriented programming methods, client-server architecture, the Django framework, Python, HTML, CSS, and JavaScript programming languages, the SQLite database, the Visual Studio Code development environment, and the Windows operating system.

Results. A web application for task management was developed with support for a Kanban board, user authorization, task assignment and editing, status management, and work progress control. The developed system contributes to improving task management efficiency and enhancing collaboration among project participants.

Field of application - IT companies, educational institutions, software development teams, and other organizations whose activities involve task and project management.

ЗМІСТ

	С.
Вступ.....	8
1 Аналіз предметної області та існуючих рішень.....	10
1.1 Поняття систем управління завданнями.....	10
1.2 Огляд Agile-методологій розробки програмного забезпечення.....	11
1.3 Особливості використання Scrum та Kanban у системах управління завданнями.....	13
1.4 Аналіз існуючих програмних рішень.....	15
1.5 Постановка задачі та визначення вимог до системи.....	18
1.6 Висновки до розділу 1.....	20
2 Проєктування вебзастосунку для управління завданнями.....	22
2.1 Загальна архітектура системи.....	22
2.2 Вибір технологій розробки.....	23
2.3 Проєктування структури бази даних.....	25
2.4 Моделювання системи за допомогою UML-діаграм.....	27
2.5 Проєктування інтерфейсу користувача.....	31
2.6 Висновки до розділу 2.....	34
3 Реалізація вебзастосунку.....	35
3.1 Реалізація серверної частини системи.....	35
3.2 Реалізація клієнтської частини вебзастосунку.....	37
3.3 Реалізація функціоналу управління завданнями.....	41
3.4 Система авторизації та управління користувачами.....	45
3.5 Реалізація серверної логіки та обробки запитів.....	46
3.6 Реалізація моделей та взаємодії з базою даних.....	49
3.7 Реалізація додаткового функціоналу системи.....	51
3.8 Висновки до розділу 3.....	53
4 Тестування та оцінка ефективності системи.....	55
4.1 Функціональне тестування системи.....	55

4.2 Аналіз результатів тестування	57
4.3 Висновки до розділу 4	59
Висновки	61
Перелік джерел посилання	62
Додаток А Текст програми.....	65
Додаток Б Слайди презентації	74

ВСТУП

Розвиток інформаційних технологій сприяє активному використанню програмних засобів для організації робочих процесів. Одним із важливих напрямів є системи управління завданнями, які застосовуються для планування діяльності, розподілу відповідальності між учасниками та контролю виконання робіт. Такі системи знаходять застосування як у сфері розробки програмного забезпечення, так і в інших галузях, де необхідна чітка координація командної роботи.

Зі зростанням складності проєктів і збільшенням кількості учасників особливої ваги набуває питання ефективної організації процесів. У цьому контексті значного поширення набули Agile-підходи, які орієнтовані на гнучкість, адаптивність і постійний контроль виконання завдань. Використання Kanban-дощок дає змогу візуально відобразити стан задач і зробити процес роботи більш зрозумілим для всіх учасників. Саме це визначає актуальність створення програмних рішень, що поєднують простоту використання з достатнім рівнем функціональності.

З огляду на зазначене постає потреба у розробленні веборієнтованої системи, яка забезпечує зручну роботу із завданнями та підтримує основні процеси управління ними. У межах цієї роботи основна увага приділяється створенню такого програмного засобу з урахуванням сучасних підходів до проєктування та розроблення вебзастосунків.

Об'єкт дослідження - процес розроблення веборієнтованого програмного забезпечення для організації та управління завданнями в межах командної роботи над проєктами.

Предмет дослідження - методи, засоби та технології розроблення веборієнтованих систем управління завданнями.

Мета роботи - розроблення веборієнтованої системи управління завданнями для підвищення ефективності організації, розподілу та контролю виконання робіт у межах командної проєктної діяльності.

Для досягнення поставленої мети визначено такі завдання:

- провести аналіз предметної області та існуючих програмних рішень;
- розглянути сучасні підходи до організації процесу розроблення програмного забезпечення;
- спроектувати архітектуру системи та структуру бази даних;
- обґрунтувати вибір технологій реалізації;
- розробити веборієнтовану систему управління завданнями;
- реалізувати механізми авторизації користувачів та управління завданнями;
- провести тестування системи та проаналізувати результати її роботи.

У процесі виконання роботи використано фреймворк Django для реалізації серверної частини, а також HTML і CSS для побудови інтерфейсу користувача. Вибір зазначених технологій зумовлений їх поширеністю та можливістю швидкої розробки вебзастосунків.

Практичне значення роботи полягає у створенні вебзастосунку, який може бути використаний для організації роботи з задачами та координації діяльності невеликих команд. Отримані результати можуть бути застосовані як основа для подальшого розвитку подібних систем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Поняття систем управління завданнями

Системи управління завданнями розглядаються як програмні інструменти, призначені для організації, координації та контролю виконання робіт у межах окремих проєктів або діяльності всієї організації. У сучасних умовах цифровізації такі системи відіграють важливу роль у забезпеченні ефективної взаємодії між учасниками процесу, дозволяючи структурувати роботу та зменшити кількість помилок, пов'язаних із людським фактором. Вони формують єдиний інформаційний простір, у якому відображається повний життєвий цикл завдання - від його створення до завершення [1].

Основною характеристикою подібних систем є можливість представлення складних процесів у вигляді набору взаємопов'язаних завдань. Наприклад, розробка програмного продукту може бути розбита на окремі етапи, такі як аналіз вимог, проєктування, реалізація та тестування. Кожен із цих етапів, у свою чергу, деталізується у вигляді конкретних задач, що призначаються окремим виконавцям. Даний підхід дозволяє не лише підвищити прозорість процесу, але й забезпечити контроль за виконанням кожного елементу роботи.

Важливою особливістю систем управління завданнями є наявність механізмів відстеження стану виконання. Кожне завдання має визначений статус, який змінюється в процесі роботи, наприклад: «до виконання», «в процесі», «на перевірці», «завершено». Це дає змогу оперативно оцінювати поточний стан проєкту та виявляти можливі затримки або проблеми. Крім того, зберігається історія змін, що дозволяє аналізувати хід виконання та приймати обґрунтовані управлінські рішення [2].

Окрему роль відіграють засоби взаємодії між користувачами, які інтегруються безпосередньо в систему. Наприклад, до кожного завдання можуть додаватися коментарі, уточнення або додаткові матеріали, що забезпечує централізоване зберігання інформації та спрощує комунікацію.

Завдяки цьому всі учасники проєкту мають доступ до актуальних даних і можуть швидко реагувати на зміни в процесі роботи.

Не менш важливим аспектом є можливість розподілу ролей і прав доступу. У більшості систем передбачено розмежування між адміністраторами, менеджерами проєктів і виконавцями. Адміністратори забезпечують технічне налаштування системи та керування правами доступу, менеджери проєктів відповідають за структуру проєкту та призначення задач, а виконавці безпосередньо виконують поставлені завдання [1], [3].

1.2 Огляд Agile-методологій розробки програмного забезпечення

У процесі розвитку інформаційних технологій підходи до організації розробки програмного забезпечення зазнали суттєвих змін. Традиційні моделі, такі як каскадна (Waterfall), передбачають чітку послідовність етапів, де кожна наступна стадія починається лише після завершення попередньої. Однак у реальних умовах це часто призводить до труднощів, оскільки вимоги до програмного продукту можуть змінюватися в процесі розробки. Незважаючи на це, традиційні моделі й надалі використовуються в проєктах із чітко визначеними вимогами. У відповідь на зазначені виклики сформувалася концепція гнучкої розробки, відома як Agile [4].

Agile розглядається не як окрема методологія, а як сукупність принципів і підходів, спрямованих на підвищення адаптивності процесу розробки. Основою цієї концепції є Agile Manifesto, який визначає ключові цінності та пріоритети, серед яких особлива увага приділяється взаємодії між людьми, швидкому отриманню результату та готовності до змін. У центрі підходу знаходиться не жорстке планування, а постійне вдосконалення продукту через короткі цикли розробки.

Однією з ключових ідей Agile є ітеративність. Робота над програмним продуктом поділяється на невеликі відрізки часу - ітерації, у межах яких команда створює частину функціоналу. Після завершення кожної ітерації

отриманий результат аналізується, тестується та може бути скоригований відповідно до нових вимог або зауважень користувачів. Даний підхід дозволяє зменшити ризики та швидше реагувати на зміни, що є особливо важливим у динамічному середовищі.

Важливою особливістю Agile є активна взаємодія між усіма учасниками процесу. Розробники, тестувальники, менеджери та замовники працюють у тісному контакті, що забезпечує краще розуміння вимог і підвищує якість кінцевого продукту. Наприклад, замовник може брати участь у регулярних демонстраціях результатів роботи, що дозволяє своєчасно вносити корективи та уникати ситуацій, коли готовий продукт не відповідає очікуванням.

Крім того, значна увага приділяється самоорганізації команди. У межах Agile підходу учасники команди самостійно визначають, як саме виконувати поставлені задачі, розподіляють обов'язки та приймають рішення щодо реалізації. Це сприяє підвищенню відповідальності та мотивації, а також дозволяє ефективніше використовувати ресурси [5]. Для кращого розуміння відмінностей між традиційними підходами та Agile доцільно узагальнити їх ключові характеристики у табл.1.1.

Таблиця 1.1 - Порівняння традиційного підходу та Agile

Критерій	Традиційний підхід (Waterfall)	Agile-підхід
Гнучкість	Низька	Висока
Зміни вимог	Обмежені	Приймаються
Планування	Жорстке, на початку	Гнучке, постійне
Взаємодія із замовником	Мінімальна	Постійна
Постачання результату	В кінці проєкту	Поступове

Як видно з таблиці, Agile забезпечує значно більшу адаптивність та орієнтацію на потреби користувача. Це пояснює його широку популярність у

сучасній розробці програмного забезпечення, де вимоги часто змінюються, а швидкість впровадження нових функцій має критичне значення.

У межах Agile сформувалося кілька конкретних підходів, які деталізують його принципи та пропонують практичні механізми реалізації. Найбільш поширеними серед них є Scrum, Kanban, Extreme Programming (XP) та інші. Кожен із цих підходів має свої особливості, проте всі вони базуються на спільних принципах гнучкості, ітеративності та тісної взаємодії між учасниками процесу [3], [6].

1.3 Особливості використання Scrum та Kanban у системах управління завданнями

У межах Agile-підходу сформувалося кілька практичних методів організації роботи, серед яких найбільш поширеними є Scrum та Kanban. Обидва підходи широко застосовуються в процесі реалізації проєктів, оскільки дозволяють ефективно організувати виконання робіт, забезпечити прозорість та контроль на всіх етапах розробки. Системи управління завданнями, у свою чергу, виступають інструментами, що підтримують реалізацію цих підходів.

Scrum розглядається як структурований підхід до організації командної роботи, що базується на чітко визначених ролях, подіях та артефактах. Основною одиницею роботи є ітерація, яка називається спринтом і зазвичай триває від одного до чотирьох тижнів. У межах кожного спринту команда виконує певний обсяг задач, визначений на етапі планування [7].

Важливою особливістю Scrum є наявність ролей, які визначають відповідальність учасників процесу. Наприклад, власник продукту формує перелік вимог, Scrum-майстер забезпечує дотримання процесу, а команда розробників виконує безпосередню реалізацію. Такий розподіл ролей дозволяє чітко організувати роботу та уникнути невизначеності.

Крім того, Scrum передбачає регулярні зустрічі, що сприяють постійній комунікації між учасниками. Наприклад, щоденні короткі обговорення допомагають відстежувати прогрес, виявляти проблеми та оперативно їх вирішувати. Після завершення спринту проводиться аналіз результатів, що дозволяє вдосконалювати процес у наступних ітераціях.

На відміну від Scrum, Kanban орієнтований на безперервний потік задач без жорсткого поділу на ітерації. Основним інструментом цього підходу є візуалізація процесу виконання роботи у вигляді дошки, яка складається з колонок, що відображають різні стани задач. Наприклад, типовими колонками можуть бути «до виконання», «в процесі» та «завершено».

Кожна задача представлена у вигляді окремого елемента, який переміщується між колонками в процесі виконання. Даний підхід дозволяє наочно відображати поточний стан роботи та швидко оцінювати завантаженість команди. Крім того, Kanban передбачає обмеження кількості задач, які можуть одночасно перебувати в певному стані, що сприяє зменшенню перевантаження та підвищенню якості виконання [8].

Суттєвою перевагою Kanban є його простота та гнучкість. Він не вимагає складної організації процесу і може бути легко інтегрований у різні типи діяльності. Саме тому цей підхід широко використовується у системах управління завданнями, зокрема у вигляді інтерактивних дошок. Для кращого розуміння відмінностей між Scrum та Kanban доцільно узагальнити їх основні характеристики в табл. 1.2.

Таблиця 1.2 - Порівняння Scrum та Kanban

Критерій	Scrum	Kanban
Організація роботи	Спринти (ітерації)	Безперервний процес
Планування	На початку кожного спринту	Постійне
Зміни під час роботи	Обмежені	Дозволені

Кінець таблиці 1.2

Критерій	Scrum	Kanban
Візуалізація задач	Може використовуватись	Основний елемент
Ролі	Чітко визначені	Необов'язкові
Контроль навантаження	Через планування	Через обмеження WIP

Як видно з таблиці, Scrum більше підходить для проєктів із чітко визначеною структурою та необхідністю регулярного планування, тоді як Kanban є більш гнучким і зручним для безперервної роботи.

У сучасних системах управління завданнями ці підходи часто поєднуються. Наприклад, може використовуватися Kanban-дошка для візуалізації задач, але при цьому робота організовується у вигляді ітерацій, характерних для Scrum. Такий комбінований підхід дозволяє отримати переваги обох методів і адаптувати процес під конкретні потреби команди

1.4 Аналіз існуючих програмних рішень

Сучасний ринок програмного забезпечення пропонує значну кількість систем управління завданнями, які відрізняються функціональністю, складністю використання та сферою застосування. Найбільш відомими серед них є Trello, Jira та Asana. Кожна з цих систем реалізує власний підхід до організації роботи, що дозволяє використовувати їх у різних умовах і для різних типів проєктів.

Платформа Trello є одним із найпростіших та найінтуїтивніших інструментів управління завданнями (рис. 1.1). Вона базується на концепції Kanban-дошки, де задачі представлені у вигляді карток, що переміщуються між колонками відповідно до стану виконання. Даний підхід забезпечує наочність і простоту використання, що робить Trello [9] зручним для невеликих команд або індивідуального використання.

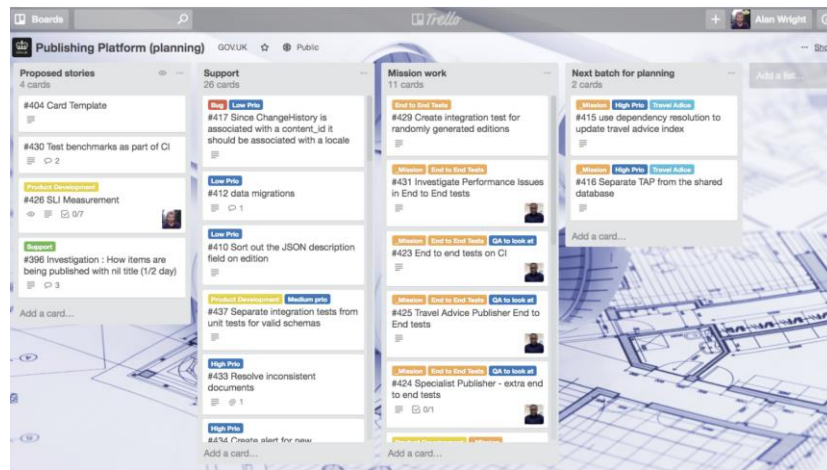


Рисунок 1.1 - Інтерфейс системи Trello [10]

До основних переваг Trello можна віднести простоту освоєння, зручну візуалізацію задач та швидке налаштування робочого середовища. Користувач може без спеціальної підготовки створювати дошки, додавати задачі та організовувати роботу. Крім того, система підтримує базові можливості співпраці, такі як коментарі, вкладення та сповіщення.

Разом із тим Trello має і певні обмеження. Зокрема, функціональність системи є недостатньою для складних проєктів, які потребують детального контролю, складної ієрархії задач або розширеної аналітики. Відсутність глибокої кастомізації та обмежені можливості звітності можуть ускладнювати використання Trello у великих організаціях.

Система Jira (рис. 1.2), розроблена компанією Atlassian, є значно більш потужним інструментом, орієнтованим передусім на команди розробки програмного забезпечення. Вона підтримує як Scrum, так і Kanban, а також надає широкі можливості для налаштування процесів, відстеження помилок і управління релізами [11].

Серед переваг Jira слід виділити високу гнучкість, можливість детального налаштування workflow, потужну систему звітності та інтеграцію з іншими інструментами розробки. Наприклад, система дозволяє відстежувати прогрес спринтів, аналізувати продуктивність команди та контролювати виконання задач на різних етапах.

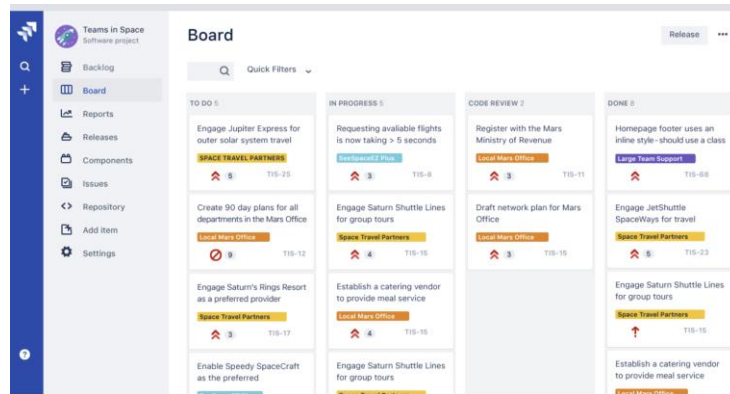


Рисунок 1.2 - Інтерфейс системи Jira [12]

Водночас складність Jira може виступати і як недолік. Для ефективного використання системи потрібен певний рівень підготовки, а налаштування може займати значний час. Крім того, надлишкова функціональність іноді ускладнює роботу невеликих команд, для яких така система є занадто громіздкою.

Платформа Asana (рис. 1.3) займає проміжне положення між простотою Trello та функціональністю Jira. Вона орієнтована на управління проєктами в різних сферах діяльності, включаючи бізнес, маркетинг та IT. Asana пропонує декілька варіантів представлення задач, зокрема списки, дошки та календарі, що дозволяє користувачам обирати найбільш зручний формат роботи [6], [13].

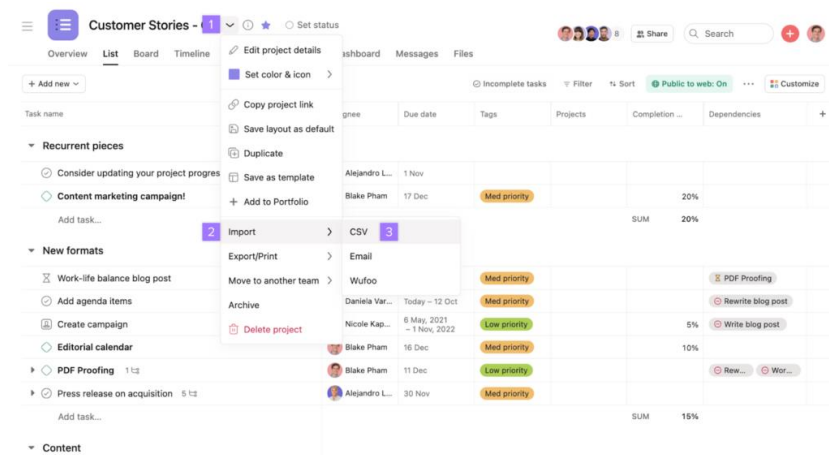


Рисунок 1.3 - Інтерфейс системи Asana [12]

Перевагами Asana є зручний інтерфейс, можливість гнучкого планування задач та хороші інструменти для командної роботи. Система дозволяє легко відстежувати прогрес, призначати виконавців та встановлювати дедлайни. Вона також підтримує інтеграцію з іншими сервісами, що розширює її функціональні можливості.

Разом із тим Asana має певні обмеження, пов'язані з недостатньою глибиною налаштування порівняно з Jira. Для складних технічних проєктів її можливостей може бути недостатньо. Крім того, деякі функції доступні лише у платних версіях, що може обмежувати використання системи.

Аналіз розглянутих програмних рішень дозволяє зробити висновок, що кожна система має власну сферу ефективного застосування. Trello підходить для простих задач і невеликих команд, Jira - для складних проєктів із високими вимогами до контролю та аналітики, тоді як Asana забезпечує баланс між функціональністю та зручністю використання [14].

1.5 Постановка задачі та визначення вимог до системи

Проведений аналіз предметної області та існуючих програмних рішень дозволяє визначити ключові проблеми, які виникають у процесі управління завданнями. Зокрема, сучасні системи або є надто простими та обмеженими у функціональності, або, навпаки, надмірно складними та перевантаженими можливостями, що ускладнює їх використання. У зв'язку з цим виникає необхідність розробки системи, яка поєднує зручність, гнучкість і достатній рівень функціональності.

Основною задачею є створення веборієнтованої системи управління завданнями, яка дозволить організувати роботу користувачів у межах проєктів, забезпечити контроль виконання задач та підтримувати ефективну взаємодію між учасниками. Система повинна бути інтуїтивно зрозумілою, мати сучасний інтерфейс і водночас надавати необхідні інструменти для управління робочими процесами.

У межах поставленої задачі передбачається реалізація функціоналу, який дозволяє створювати проєкти та організовувати їх структуру у вигляді дощок і колонок. Кожен проєкт має містити набір задач, що проходять певні етапи виконання. Наприклад, задача може перебувати у станах «до виконання», «в процесі», «на перевірці» та «завершено». Даний підхід забезпечує візуалізацію процесу роботи та відповідає принципам Kanban.

Особлива увага приділяється роботі з користувачами системи. Передбачається наявність механізму реєстрації та авторизації, що дозволяє ідентифікувати кожного учасника та забезпечити персоналізований доступ до функціоналу. Користувачі повинні мати можливість отримувати призначені їм задачі, переглядати їх, редагувати та змінювати статус виконання. Крім того, необхідно реалізувати особистий кабінет, у якому відображаються всі задачі конкретного користувача.

Важливою складовою системи є механізм призначення задач. Адміністратор або відповідальний користувач повинен мати можливість призначати задачі виконавцям, що дозволяє ефективно розподіляти навантаження між учасниками. Це забезпечує чітке визначення відповідальності та покращує організацію роботи.

Ще одним важливим аспектом є підтримка комунікації в межах задач. Кожна задача повинна мати можливість додавання коментарів, що дозволяє обговорювати деталі реалізації, уточнювати вимоги та фіксувати результати роботи. При цьому автор коментаря має визначатися автоматично на основі авторизованого користувача, що виключає необхідність ручного введення даних.

Для забезпечення контролю та аналізу процесу виконання задач необхідно реалізувати механізм збереження історії змін. Це дозволяє відстежувати переміщення задач між колонками, зміну статусів та дії користувачів. Така інформація може бути використана для аналізу ефективності роботи та виявлення проблемних етапів [8], [15].

З урахуванням зазначеного можна сформулювати основні функціональні вимоги до системи:

- історії змін задач;
- зміна створення та управління проєктами;
- візуалізація потоку задач у вигляді Kanban-дошки;
- створення, редагування та видалення задач;
- призначення задач користувачам;
- підтримка реєстрації та авторизації користувачів;
- реалізація особистого кабінету користувача;
- додавання коментарів до задач;
- збереження статусу задач (у тому числі через переміщення між колонками).

Окрім функціональних вимог, система повинна відповідати низці нефункціональних характеристик. Зокрема, важливими є зручність інтерфейсу, швидкість роботи, надійність збереження даних та можливість подальшого розширення функціоналу. Інтерфейс повинен бути адаптованим до різних типів користувачів і не вимагати спеціальної підготовки.

1.6 Висновки до розділу 1

У першому розділі було проведено аналіз предметної області систем управління завданнями, що дозволило сформулювати цілісне уявлення про їх призначення, функціональні можливості та роль у сучасних умовах розробки програмного забезпечення. Встановлено, що такі системи забезпечують організацію роботи, контроль виконання задач і ефективну взаємодію між учасниками проєкту.

Розгляд Agile-підходів показав, що сучасна розробка програмного забезпечення орієнтується на гнучкість, адаптивність до змін та ітеративність процесу. Особливу увагу було приділено методам Scrum і Kanban, які є найбільш поширеними серед гнучких методів розробки програмного

забезпечення. Визначено, що Scrum забезпечує структурованість процесу через ітерації та ролі, тоді як Kanban надає зручний механізм візуалізації та контролю виконання задач у реальному часі.

У процесі аналізу існуючих програмних рішень встановлено, що такі системи, як Trello, Jira та Asana, мають різний рівень складності та функціональності. Trello відзначається простотою та наочністю, Jira - широкими можливостями налаштування та контролю, а Asana поєднує зручність використання з достатнім функціоналом для управління проектами. Разом із тим жодне з розглянутих рішень не є універсальним, що обумовлює доцільність розробки власної системи з урахуванням їх переваг і недоліків.

На основі проведеного аналізу було сформульовано постановку задачі та визначено основні вимоги до розроблюваної системи. Зокрема, встановлено необхідність реалізації функціоналу управління проектами, задачами та користувачами, підтримки авторизації, призначення задач, ведення історії змін і забезпечення зручного інтерфейсу взаємодії.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ЗАВДАННЯМИ

2.1 Загальна архітектура системи

У межах даної роботи було обрано клієнт-серверну архітектуру, яка є стандартом для веборієнтованих застосунків і дозволяє ефективно розділити обов'язки між компонентами системи.

Клієнт-серверна модель передбачає взаємодію між користувачем і сервером через веббраузер. Користувач здійснює запити до системи, наприклад переглядає задачі, додає коментарі або змінює статус виконання, а сервер обробляє ці запити, звертається до бази даних і повертає результат у вигляді вебсторінки. Архітектура системи побудована за трирівневим принципом:

- рівень представлення (інтерфейс користувача);
- рівень бізнес-логіки (сервер);
- рівень даних (база даних).

Для наочного представлення структури системи наведено узагальнену схему, яка представлена на рисунку 2.1.

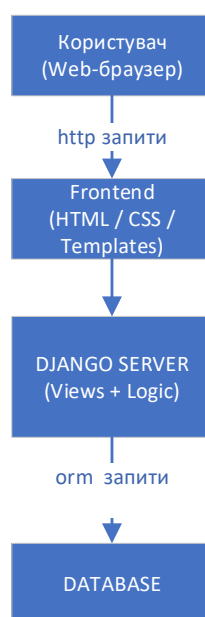


Рисунок 2.1 - Схематичне представлення архітектури системи

Як видно зі схеми, користувач взаємодіє із системою через веббраузер, який надсилає HTTP-запити до серверної частини. Сервер, реалізований за допомогою фреймворку Django, обробляє запити, виконує бізнес-логіку та звертається до бази даних для отримання або збереження інформації.

Рівень представлення відповідає за відображення даних користувачеві. Саме тут формуються сторінки з проєктами, дошками, задачами та коментарями. Використання шаблонів дозволяє динамічно генерувати HTML [16] залежно від отриманих даних.

Рівень бізнес-логіки забезпечує виконання основних операцій системи. Наприклад, обробляються дії користувача, такі як створення задачі, її переміщення між колонками, призначення виконавця або додавання коментаря. Усі ці операції реалізуються у відповідних представленнях (views).

Рівень даних відповідає за збереження інформації. База даних містить таблиці користувачів, проєктів, задач, коментарів та історії змін. Використання ORM дозволяє працювати з базою даних через об'єктно-орієнтований підхід, що значно спрощує розробку.

2.2 Вибір технологій розробки

Основою серверної частини системи є мова програмування Python, яка широко використовується у веброзробці завдяки своїй простоті, читабельності та великій кількості бібліотек. Python дозволяє швидко реалізовувати складну бізнес-логіку та зменшує час розробки, що є важливим фактором у навчальних і практичних проєктах.

Для реалізації вебзастосунку було використано фреймворк Django, який працює за принципом Model-View-Template (MVT) і забезпечує чітку структуру проєкту. Django надає вбудовані засоби для роботи з базою даних, обробки HTTP-запитів, маршрутизації та автентифікації користувачів.

Наприклад, використання вбудованої системи авторизації дозволяє швидко реалізувати реєстрацію та вхід користувачів без необхідності створення складної логіки з нуля.

На рівні збереження даних використовується система керування базами даних SQLite, яка є легкою у налаштуванні та не потребує окремого серверу. Це робить її зручною для розробки та тестування. SQLite добре підходить для невеликих і середніх проєктів, забезпечуючи достатню швидкість та надійність збереження даних. У разі необхідності система може бути масштабована шляхом переходу на більш потужні СУБД, наприклад PostgreSQL [17].

Для взаємодії з базою даних використовується вбудований ORM-механізм Django, який дозволяє працювати з даними через об'єктно-орієнтований підхід. Це спрощує написання запитів і підвищує читабельність коду. Наприклад, замість написання SQL-запитів розробник працює з моделями, що значно зменшує ймовірність помилок.

Клієнтська частина системи реалізована за допомогою стандартних вебтехнологій, таких як HTML, CSS та базового JavaScript. HTML відповідає за структуру сторінок, CSS - за їх оформлення, а JavaScript використовується для реалізації інтерактивних елементів, наприклад взаємодії з задачами або оновлення інтерфейсу без перезавантаження сторінки.

Для організації інтерфейсу використовується шаблонний механізм Django, який дозволяє розділити логіку та представлення даних. Це забезпечує зручність підтримки та дає можливість змінювати зовнішній вигляд системи без впливу на серверну частину.

Серед додаткових інструментів слід відзначити використання вбудованої адміністративної панелі Django, яка дозволяє здійснювати управління даними через зручний вебінтерфейс. Наприклад, адміністратор може створювати користувачів, проєкти або задачі без необхідності прямого доступу до бази даних.

2.3 Проектування структури бази даних

На основі проведеного аналізу предметної області було визначено ключові сутності системи, до яких належать користувачі, проекти, дошки, стовпці, завдання, коментарі та історія змін. Кожна із зазначених сутностей реалізована у вигляді окремої таблиці бази даних, що містить відповідний набір атрибутів.

Таблиця користувачів призначена для зберігання інформації про зареєстрованих осіб, які взаємодіють із системою. Вона містить ім'я користувача, електронну адресу, пароль та дату створення облікового запису. Користувач виступає центральним елементом системи, оскільки саме він створює проекти, виконує завдання та залишає коментарі [16].

Таблиця проектів використовується для організації роботи в межах окремих напрямів діяльності. Кожен проект має назву, опис та належить певному користувачеві, який є його власником.

Дошки слугують інструментом візуалізації процесу виконання завдань. Вони належать до конкретного проекту та містять набір стовпців, що відображають етапи виконання роботи.

Стовпці визначають послідовність виконання завдань і відповідають певним статусам, таким як "До виконання", "В процесі", "На перевірці" та "Завершено". Для забезпечення коректного відображення передбачено поле порядкового номера.

Завдання є основною функціональною одиницею системи. Воно містить назву, опис, статус, пріоритет, інформацію про виконавця, а також додаткові параметри, зокрема оцінку складності та орієнтовний час виконання.

Коментарі забезпечують можливість взаємодії між користувачами в межах конкретного завдання. Вони дозволяють уточнювати деталі, обговорювати виконання та фіксувати результати роботи.

З метою забезпечення контролю змін у системі реалізовано таблицю історії завдань, яка зберігає інформацію про зміну статусів, переміщення між стовпцями та користувача, який здійснив відповідну дію [18]. Для наочного представлення структури бази даних та зв'язків між її основними сутностями було побудовано ER-діаграму (рис. 2.2).

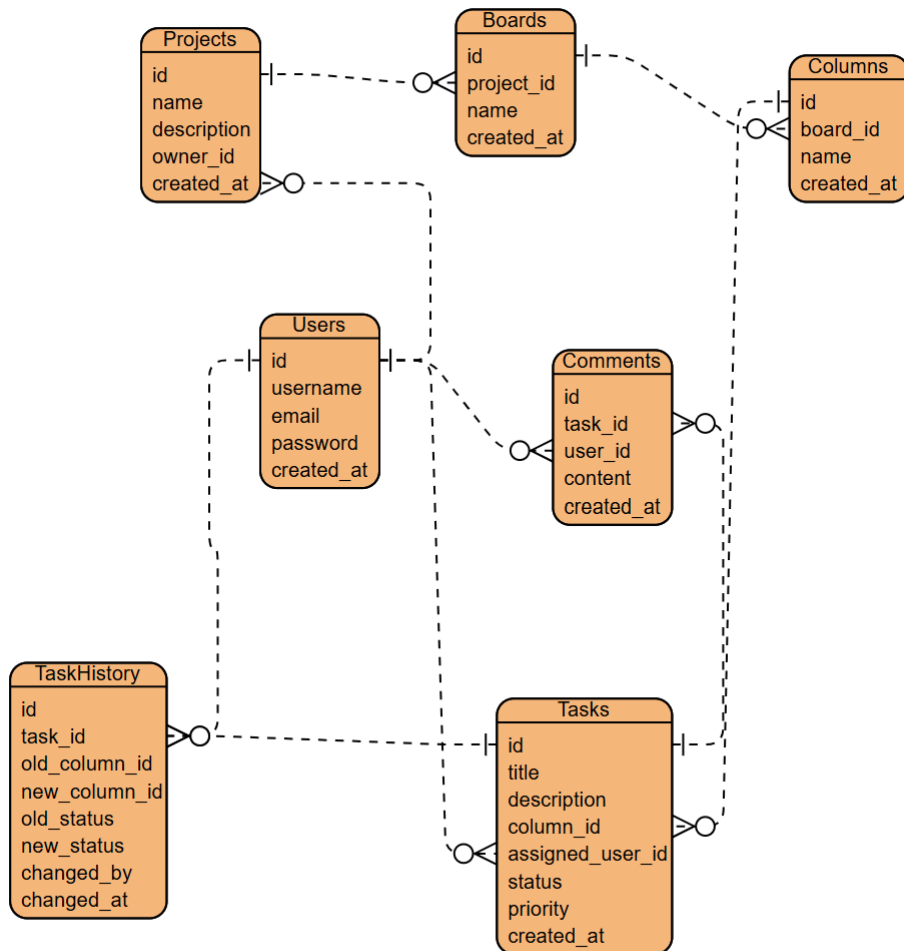


Рисунок 2.2 - ER-діаграма бази даних системи управління завданнями

Як видно з ER-діаграми, між таблицями реалізовано зв'язки типу «один-до-багатьох», що забезпечує логічну цілісність структури бази даних. Один користувач може створювати декілька проєктів, кожен проєкт може містити кілька дощок, а кожна дошка - набір стовпців. У свою чергу, кожен

стовпець містить декілька завдань, що відображає поетапний процес виконання робіт.

Кожне завдання належить одному стовпцю, проте може бути призначене одному користувачу як виконавцю. При цьому один користувач може бути відповідальним за виконання багатьох завдань, що формує додатковий зв'язок між сутностями Users і Tasks.

Крім того, кожне завдання може містити декілька коментарів, причому кожен коментар створюється конкретним користувачем. Таким чином, реалізується зв'язок між таблицями Tasks, Comments і Users, що забезпечує можливість організації комунікації в межах завдань.

Окремо передбачено таблицю історії змін, яка фіксує всі зміни, пов'язані із завданням. Кожен запис історії належить конкретному завданню та містить інформацію про користувача, який виконав зміну, що формує зв'язки між таблицями TaskHistory, Tasks і Users.

2.4 Моделювання системи за допомогою UML-діаграм

2.4.1 Діаграма послідовності

У межах розробленої системи управління завданнями діаграма послідовності демонструє процес виконання основних сценаріїв, зокрема створення задачі та зміни її статусу. У цих процесах беруть участь такі компоненти:

- користувач (адміністратор або виконавець);
- клієнтська частина (вебінтерфейс);
- сервер (Django-застосунок);
- база даних.

Процес створення задачі починається з дії адміністратора, який заповнює форму у вебінтерфейсі. Після відправлення форми дані передаються на сервер, де відбувається їх обробка та збереження у базі

даних. Після успішного збереження сервер повертає відповідь, і нова задача відображається у відповідному стовпці Kanban-дошки.

Аналогічно відбувається процес зміни статусу задачі. Користувач ініціює зміну статусу, після чого запит передається на сервер, де оновлюється відповідний запис у базі даних [19]. Після цього оновлена інформація повертається на клієнтську частину та відображається у вигляді переміщення задачі між стовпцями. Схематичне представлення взаємодії компонентів наведено на рисунку 2.3.

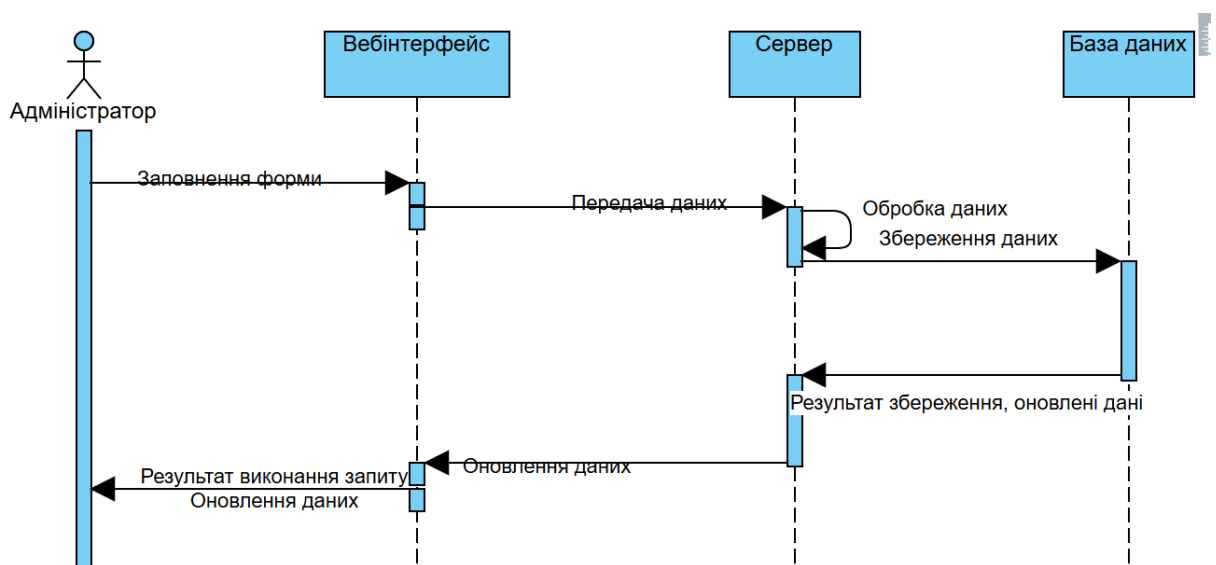


Рисунок 2.3 - Діаграма послідовності взаємодії компонентів системи

Як видно з діаграми, всі основні операції виконуються у чітко визначеній послідовності. Користувач взаємодіє лише з інтерфейсом, тоді як обробка логіки та робота з даними здійснюється на сервері.

2.4.2 Діаграма варіантів використання

Діаграма варіантів використання призначена для відображення функціональних можливостей системи та взаємодії користувачів із її основними компонентами. Вона дозволяє визначити перелік дій, які можуть

виконувати користувачі, а також відобразити межі системи та розподіл ролей.

У розробленій системі управління завданнями виділено дві основні ролі: адміністратор та користувач. Кожна з цих ролей має власний набір функціональних можливостей, що відповідає їхньому призначенню в системі [20].

Адміністратор виконує функції управління та контролю. Він має доступ до створення задач, призначення виконавців та редагування задач. Крім того, адміністратор, як і звичайний користувач, може здійснювати авторизацію, переглядати проекти та працювати із задачами.

Користувач взаємодіє із задачами, які призначені йому для виконання. Він має можливість переглядати задачі, змінювати їхній статус відповідно до етапу виконання, додавати коментарі, а також переглядати історію змін. Це дозволяє забезпечити зручну організацію роботи та контроль за виконанням завдань.

До основних варіантів використання системи належать:

- авторизація користувача;
- перегляд проектів;
- створення задачі;
- призначення задачі;
- редагування задачі;
- перегляд задачі;
- зміна статусу задачі;
- додавання коментаря;
- перегляд історії змін.

Схематичне представлення взаємодії користувачів із системою наведено на рисунку 2.4.

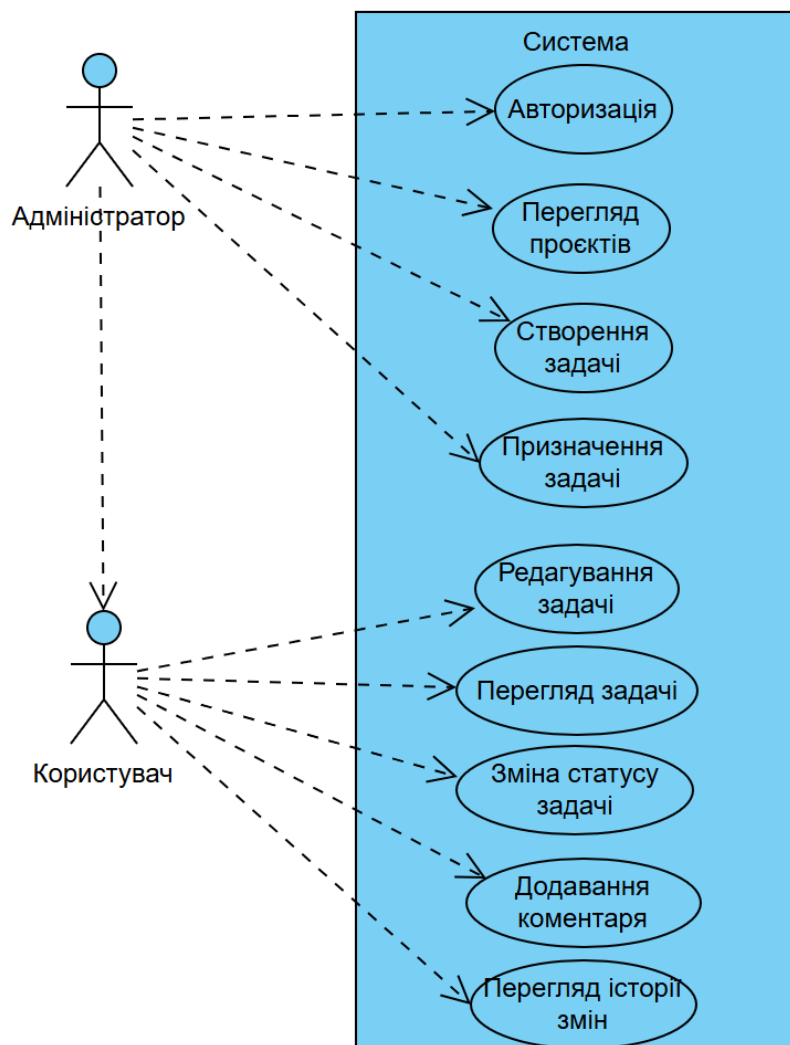


Рисунок 2.4 - Діаграма варіантів використання системи

Як видно з діаграми, адміністратор має ширший набір доступних дій, що пов'язано з його роллю у системі управління. Зокрема, лише адміністратор може створювати задачі, призначати їх виконавцям та редагувати їх параметри. У свою чергу, звичайний користувач працює з уже створеними задачами, виконуючи їх та взаємодіючи з іншими учасниками через коментарі.

Використання діаграми варіантів використання дозволило чітко визначити функціональні можливості системи, розмежувати права доступу між ролями та сформулювати загальне уявлення про взаємодію користувачів із системою. Це стало основою для подальшої реалізації функціоналу та забезпечило логічну узгодженість усіх компонентів застосунку [21].

2.5 Проектування інтерфейсу користувача

У процесі проектування системи управління завданнями було передбачено врахування основних принципів побудови сучасних інтерфейсів, зокрема простоти, логічної структурованості, інтуїтивної зрозумілості та мінімізації зайвих дій користувача.

Основною сторінкою системи є інформаційна панель, яка надає узагальнену інформацію про стан системи та забезпечує швидкий доступ до ключових функцій (рис. 2.4).

Як видно з рисунка 2.5, головна сторінка містить статистичні показники (кількість проєктів, дощок та завдань), а також список доступних проєктів. Кожен проєкт представлений у вигляді окремої картки, що містить основну інформацію та кнопку переходу до відповідної дошки. Даний підхід забезпечує швидку навігацію та зручний доступ до функціоналу [22].

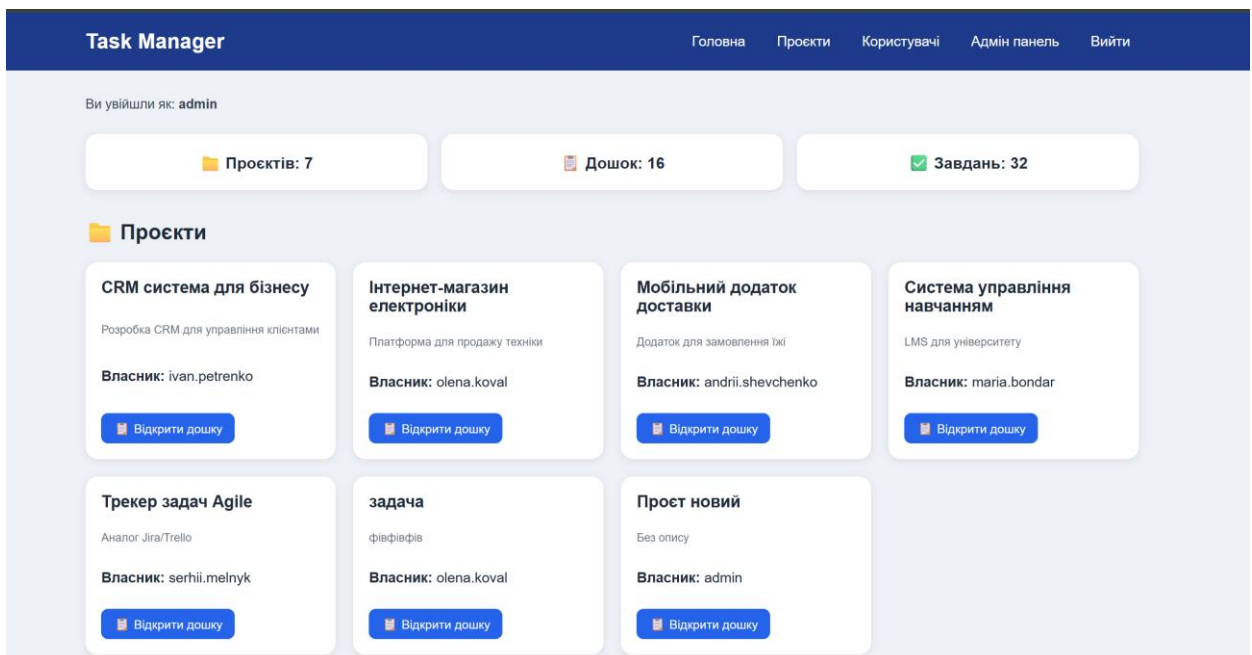


Рисунок 2.5 - Головна сторінка системи (Dashboard)

Ключовим елементом інтерфейсу є сторінка дошки, яка реалізує принципи Kanban та дозволяє візуалізувати процес виконання завдань (рис. 2.6).

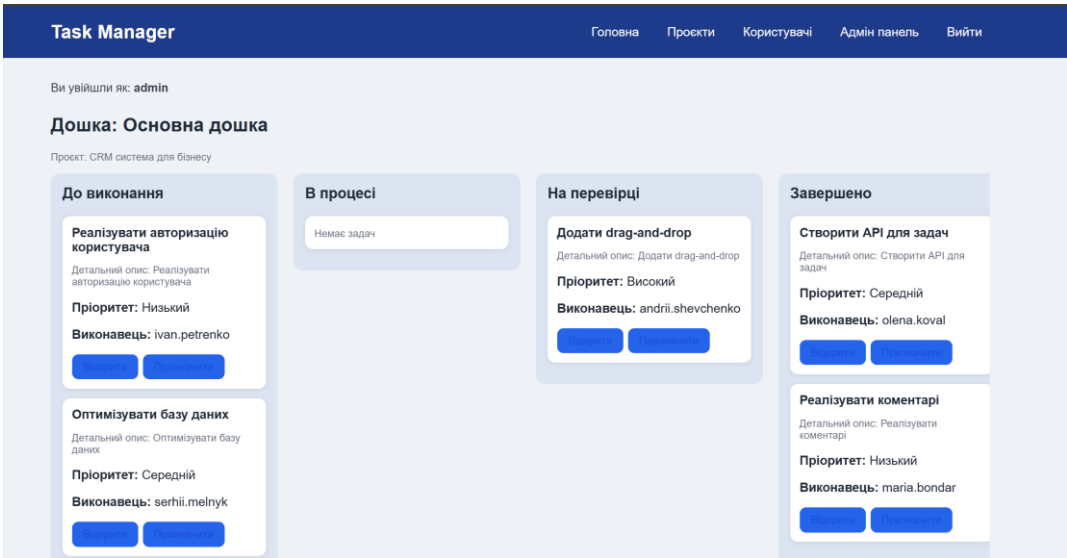


Рисунок 2.6 - Інтерфейс дошки завдань

Як показано на рисунку 2.6, дошка складається з кількох стовпців, які відповідають етапам виконання завдань. У межах кожного стовпця відображаються картки завдань із основною інформацією: назвою, описом, пріоритетом та виконавцем. Передбачено можливість переходу до детального перегляду завдання, а також виконання додаткових дій, зокрема призначення виконавця [23].

Для детального аналізу окремого завдання реалізовано спеціалізовану сторінку, яка містить розширену інформацію (рис. 2.7).

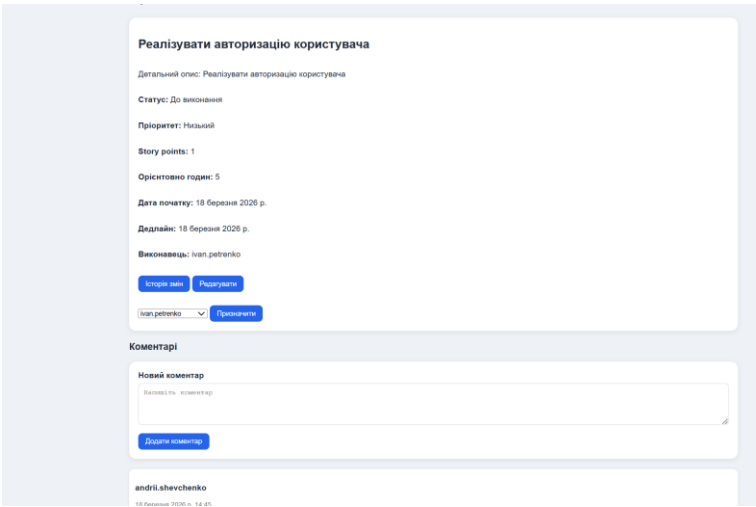


Рисунок 2.7 - Сторінка перегляду завдання

Як видно з рисунка 2.7, сторінка завдання містить повний опис, статус, пріоритет, часові параметри, а також інформацію про виконавця. Окрім цього, передбачено можливість призначення користувача та додавання коментарів, що забезпечує ефективну комунікацію між учасниками процесу.

Важливим елементом системи є механізм відстеження змін, який реалізовано у вигляді історії завдання.

На рисунку 2.8 представлено інтерфейс історії змін, який відображає всі дії, виконані над завданням. Це дозволяє відстежувати процес виконання, аналізувати зміни статусів та контролювати діяльність користувачів.

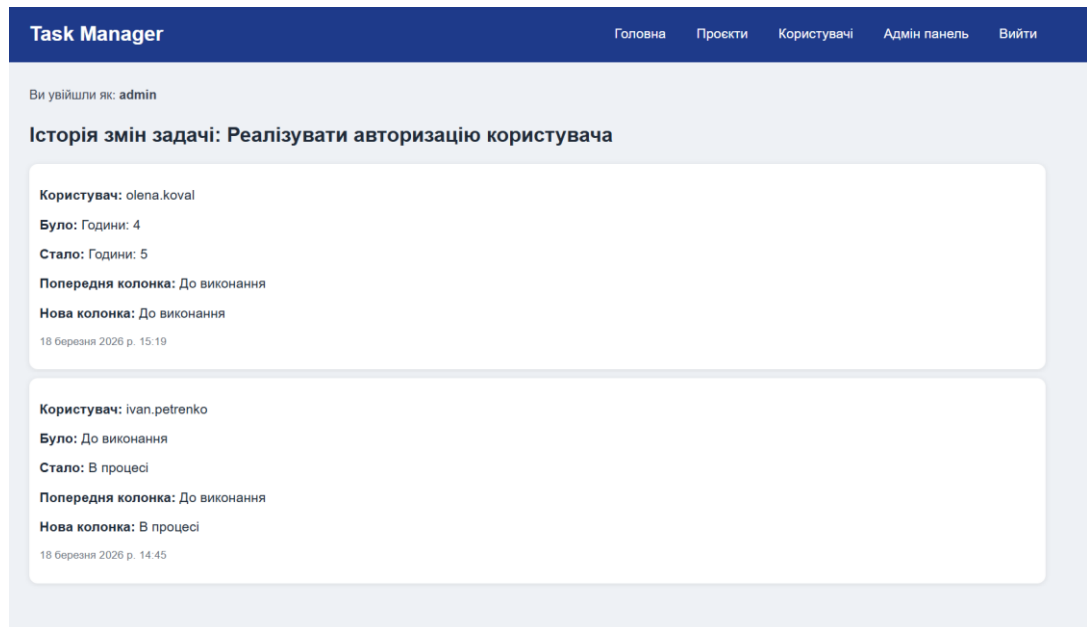


Рисунок 2.8 - Історія змін завдання

У межах системи також реалізовано персоналізований підхід до роботи користувачів. Кожен користувач після авторизації отримує доступ до власних завдань, може змінювати їх статус, брати нові задачі та взаємодіяти з іншими учасниками системи [24].

Інтерфейс побудовано з використанням єдиної стилістики, що забезпечує цілісність сприйняття та покращує користувацький досвід. Використання адаптивного дизайну дозволяє коректно відображати елементи інтерфейсу на різних пристроях.

2.6 Висновки до розділу 2

У другому розділі було розглянуто питання проектування системи управління завданнями та обґрунтовано вибір основних технічних рішень. На цьому етапі сформовано загальну архітектуру застосунку, визначено структуру бази даних та підібрано технології для реалізації системи.

Було обрано клієнт-серверну архітектуру, яка забезпечує розподіл логіки між серверною та клієнтською частинами. Даний підхід дозволяє ефективно організувати обробку даних, забезпечити масштабованість системи та спростити її подальшу підтримку.

У межах розділу обґрунтовано вибір технологій розробки, зокрема використання фреймворку Django для реалізації серверної частини та стандартних вебтехнологій для побудови інтерфейсу користувача. Обрані інструменти дозволяють швидко реалізувати необхідний функціонал і забезпечують стабільну роботу системи.

Особливу увагу приділено проектуванню структури бази даних. Визначено основні сутності предметної області, їх атрибути та взаємозв'язки. Розроблена ER-діаграма відображає логічну структуру даних і забезпечує цілісність інформації в системі. Запропонована структура дозволяє ефективно зберігати та обробляти дані, необхідні для роботи із задачами.

Також було розглянуто підходи до проектування інтерфейсу користувача. Визначено основні принципи побудови інтерфейсу, спрямовані на забезпечення зручності використання, зрозумілості та швидкого доступу до функціоналу системи.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Реалізація серверної частини системи

Реалізація серверної частини системи управління завданнями є одним із ключових етапів розробки, оскільки саме вона забезпечує обробку запитів користувачів, взаємодію з базою даних, реалізацію бізнес-логіки та контроль доступу до функціональних можливостей системи. Серверна частина виступає центральним елементом, який об'єднує всі компоненти системи в єдину узгоджену структуру.

У якості основної технології для реалізації серверної частини було використано фреймворк Django [25], який дозволяє ефективно організувати структуру вебдодатку, забезпечує високий рівень безпеки та містить вбудовані засоби для роботи з базами даних, маршрутизації запитів і управління користувачами.

Архітектура серверної частини побудована відповідно до підходу MTV (Model-Template-View), який є адаптацією класичної моделі MVC. У межах даного підходу модель відповідає за структуру та логіку роботи з даними, представлення обробляє HTTP-запити та формує відповіді, а шаблони забезпечують відображення інформації користувачу.

У системі було реалізовано набір моделей, які відображають сутності предметної області, зокрема користувачів, проекти, дошки, стовпці, завдання, коментарі та історію змін. Кожна модель пов'язана з іншими через зовнішні ключі, що забезпечує цілісність даних та коректну організацію зв'язків між об'єктами.

Обробка запитів користувачів реалізується за допомогою функцій-представлень (views), які приймають HTTP-запити, виконують необхідні операції з даними та передають результат у шаблони для подальшого відображення. Зокрема, у системі реалізовано функції для відображення списку проектів, перегляду дошки, детальної інформації про завдання,

додавання коментарів, переміщення завдань між стовпцями та перегляду історії змін [26].

Особливу увагу приділено реалізації логіки роботи із завданнями. Система дозволяє створювати завдання, призначати їх користувачам, змінювати їх статус, переміщувати між стовпцями та додавати коментарі. При кожній зміні статусу або положення завдання автоматично створюється запис у таблиці історії змін, що дозволяє відстежувати повний життєвий цикл завдання.

Важливим елементом серверної частини є система аутентифікації та авторизації користувачів. Завдяки вбудованим засобам Django реалізовано механізми реєстрації, входу в систему та обмеження доступу до окремих функцій. Наприклад, можливість додавання коментарів або зміни статусу завдання доступна лише авторизованим користувачам, а функція призначення завдань - адміністраторам.

Для підвищення зручності роботи користувачів реалізовано персональний кабінет, у якому відображаються всі призначені користувачу завдання. У межах цього функціоналу користувач може переглядати власні задачі, брати нові в роботу, змінювати їх статус та завершувати виконання.

З метою оптимізації роботи з базою даних використовуються механізми `select_related` та `prefetch_related`, що дозволяють зменшити кількість запитів та підвищити швидкодію системи.

Для кращого розуміння взаємодії компонентів серверної частини доцільно розглянути загальну схему обробки запиту в системі, яка представлена на рисунку 3.1.

Як видно зі схеми, після надсилання HTTP-запиту користувачем він обробляється системою маршрутизації, яка визначає відповідну функцію-представлення. Далі відбувається взаємодія з моделями та базою даних, після чого сформована відповідь передається у шаблон і відображається користувачу у вигляді вебсторінки.

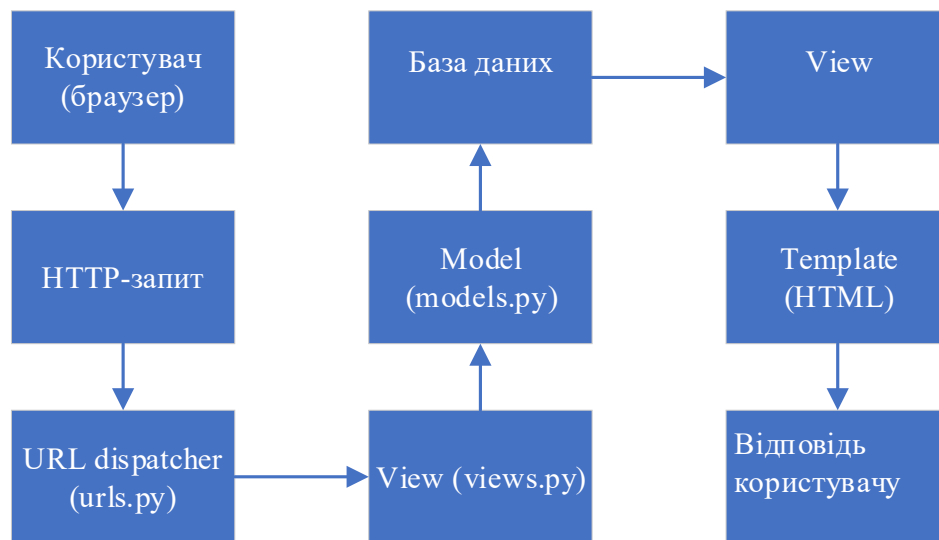


Рисунок 3.1 - Схема обробки запиту в серверній частині системи

3.2 Реалізація клієнтської частини вебзастосунку

Клієнтську частину вебзастосунку реалізовано з використанням мови розмітки HTML, каскадних таблиць стилів CSS [27]-[28] та шаблонізатора Django Templates. Підхід дозволяє формувати динамічні вебсторінки шляхом відображення даних, отриманих із серверної частини, та забезпечує взаємодію користувача із системою управління завданнями.

Інтерфейс користувача сформовано на основі принципів простоти та логічної структурованості. Навігацію реалізовано у вигляді верхнього меню, яке забезпечує доступ до основних розділів системи. Відображення елементів інтерфейсу здійснюється за допомогою шаблонів, у які передаються дані з контролерів Django.

Головну сторінку системи реалізовано як інформаційну панель, що формується на основі даних із бази даних. На ній відображається загальна статистика (кількість проєктів, дощок та задач), а також список доступних проєктів. Дані передаються з серверної частини у шаблон і відображаються у вигляді карток. Візуальне представлення головної сторінки наведено на рисунку 3.2.

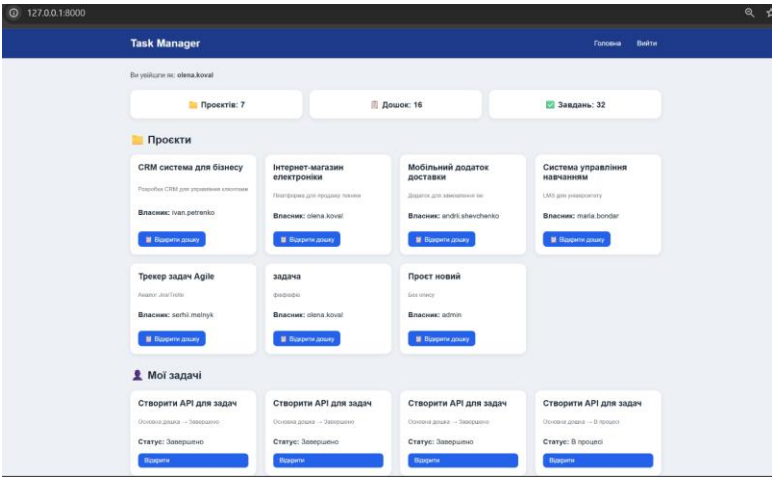


Рисунок 3.2 - Головна сторінка системи з переліком проєктів та статистикою

На цій же сторінці реалізовано блок «Мої задачі», формування якого здійснюється шляхом фільтрації задач за полем виконавця. Відповідні записи вибираються з бази даних і передаються у шаблон для відображення. Це дозволяє забезпечити персоналізований доступ користувача до своїх задач (рис. 3.3).

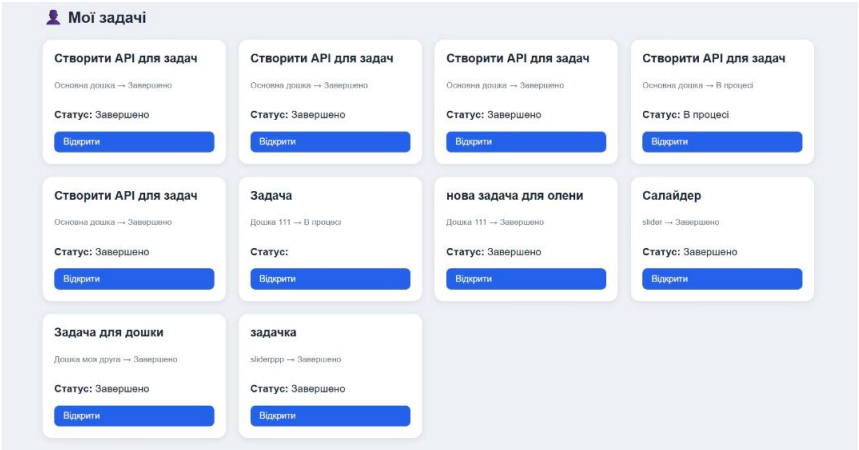


Рисунок 3.3 - Відображення задач, призначених користувачу

Для візуалізації процесу виконання задач використано інтерфейс Kanban-дошки. Вона поділена на стовпці відповідно до статусів, а задачі представлені у вигляді карток. Таке представлення дозволяє легко відстежувати, на якому етапі знаходиться кожне завдання, та спрощує роботу з ними (рис. 3.4).

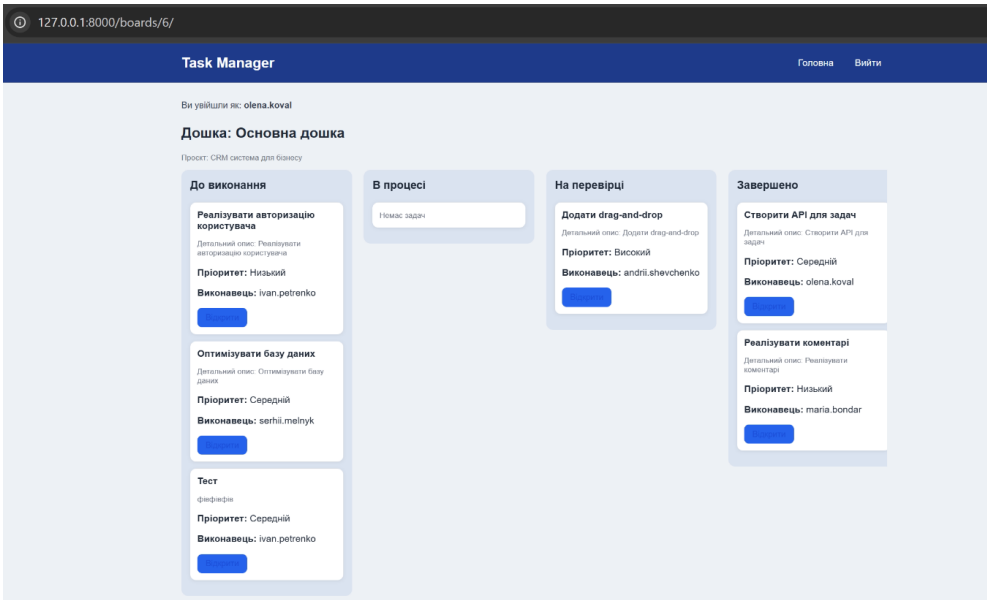


Рисунок 3.4 - Інтерфейс Kanban-дошки із розподілом задач за статусами

Для роботи із завданнями передбачено окрему сторінку їх детального перегляду. Вона містить повну інформацію про задачу, включаючи опис, статус, пріоритет, виконавця, а також додаткові параметри [29]. Це забезпечує можливість глибокого аналізу та редагування задачі. Приклад наведено на рисунку 3.5.

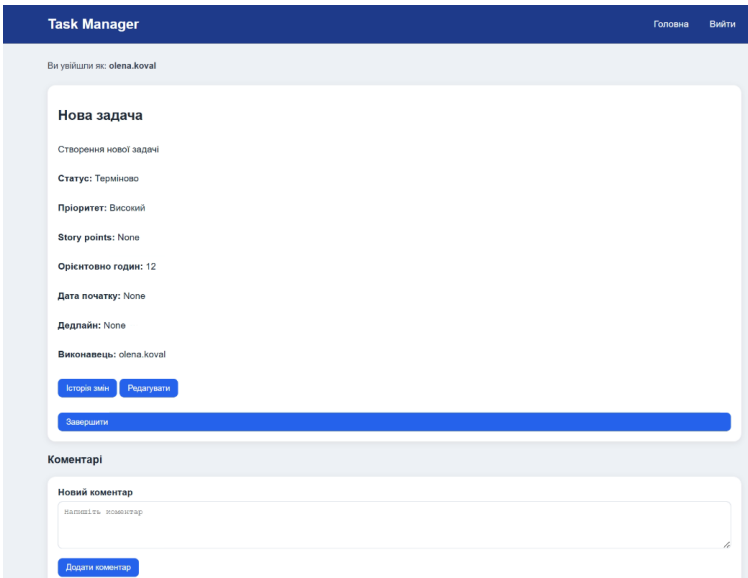
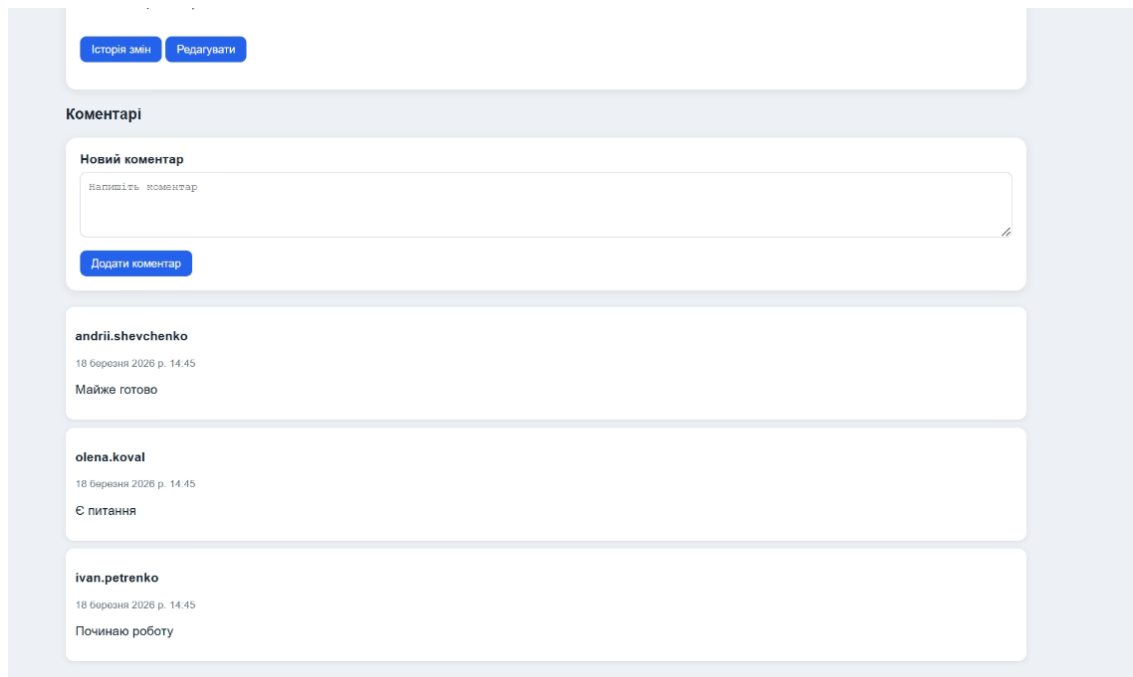


Рисунок 3.5 - Сторінка детального перегляду задачі

На сторінці задачі передбачено блок коментарів, який інтегровано в загальну структуру інтерфейсу. Введення нового коментаря здійснюється через форму, після чого дані передаються на сервер і відображаються у вигляді списку. Кожен запис містить ім'я користувача, дату створення та текст повідомлення. Приклад відображення коментарів наведено на рисунку 3.6.



Історія змін	Редагувати
Коментарі	
Новий коментар	
Напишіть коментар	
Додати коментар	
andrii.shevchenko	
18 березня 2026 р. 14:45	
Майже готово	
olena.koval	
18 березня 2026 р. 14:45	
Є питання	
ivan.petrenko	
18 березня 2026 р. 14:45	
Починаю роботу	

Рисунок 3.6 - Реалізація системи коментарів до задач

Для створення нових задач використовується окрема форма введення, що містить поля для задання основних параметрів, зокрема назви, опису, статусу, пріоритету та виконавця. Після заповнення форми дані передаються на сервер і зберігаються у базі даних, після чого задача відображається у відповідному розділі системи. Інтерфейс форми наведено на рисунку 3.7.

Окрім цього, у системі передбачено перегляд історії змін задач. Дані про зміни формуються на основі записів, що зберігаються у відповідній таблиці бази даних, і відображаються у вигляді списку. Кожен запис містить інформацію про попередній та новий стан задачі, а також користувача, який виконав зміну. Приклад відображення наведено на рисунку 3.8.

127.0.0.1:8000/create-task/

Task Manager Головна Проекти Користувачі Адмін панель Вийти

Ви увійшли як: admin

Створити задачу

Нова задача

Створення нової задачі

Колонка: Основна дошка --> До виконання

Користувач: olena.koval

Статус:

Терміново

Пріоритет:

Високий

Створити

Рисунок 3.7 - Форма створення нової задачі

127.0.0.1:8000/task/11/history/

Task Manager Головна Вийти

Ви увійшли як: olena.koval

Історія змін задачі: Реалізувати авторизацію користувача

Користувач: Ivan.petroenko

Було: До виконання

Стало: В процесі

Попередня колонка: До виконання

Нова колонка: В процесі

18 березня 2026 р. 14:45

Рисунок 3.8 - Відображення історії змін задачі

Клієнтська частина вебзастосунку охоплює основні операції роботи із задачами, зокрема їх перегляд, створення, редагування, призначення та відстеження виконання. Інтерфейс побудований відповідно до сучасних підходів до розробки вебзастосунків і орієнтований на зручну взаємодію користувача з системою.

3.3 Реалізація функціоналу управління завданнями

У застосунку охоплено основні етапи роботи із задачами - від їх створення до завершення. Створення задач виконується адміністратором через окрему форму, у якій задаються назва, опис, статус, пріоритет і

виконавець. Після заповнення форми дані передаються на сервер і зберігаються у базі даних.

Інтерфейс створення задачі представлений у вигляді форми з полями для введення основних параметрів. Відповідний вигляд сторінки наведено на рисунку 3.9.

The screenshot shows a web browser window with the address '127.0.0.1:8000/create-task/'. The page title is 'Task Manager'. The navigation bar includes links for 'Головна', 'Проекти', 'Користувачі', 'Адмін панель', and 'Вийти'. The user is logged in as 'admin'. The main heading is 'Створити задачу' (Create Task). The form contains the following fields: 'Створення задачі' (Task Creation), 'Сторінка: довідка' (Page: reference), 'Колонка: Основна дошка -> До виконання' (Column: Main board -> To be done), 'Користувач: olena.koval' (User: olena.koval), 'Статус: Терміново' (Status: Urgent), 'Пріоритет: Високий' (Priority: High), and a 'Створити' (Create) button.

Рисунок 3.9 - Сторінка створення задачі

Після створення задача відображається на дошці у відповідному стовпці. Організація даних базується на Kanban-підході [29]-[30], де задачі розподіляються за статусами та подаються у вигляді карток.

Відображення задач у вигляді колонок відповідає етапам їх виконання та дає змогу швидко оцінити поточний стан кожної задачі. Результат представлено на рисунку 3.10.

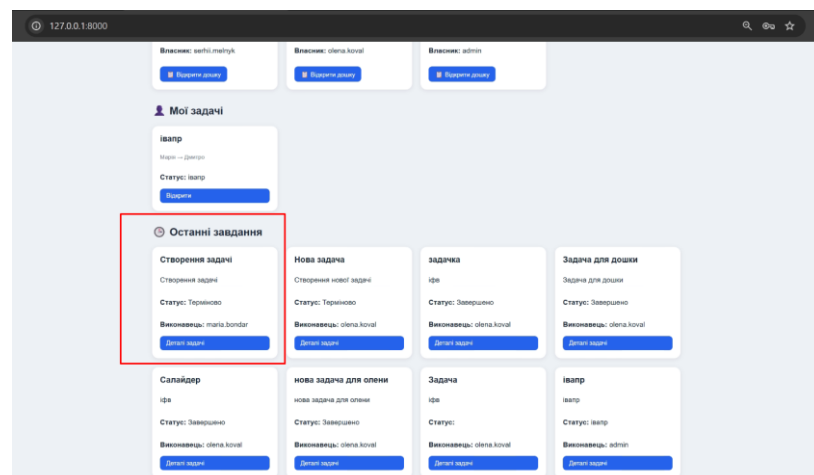


Рисунок 3.10 - Відображення задач на Kanban-дошці

Під час роботи із задачею передбачено зміну її стану шляхом переміщення між колонками. Такий підхід відповідає зміні статусу задачі, наприклад із «До виконання» у «В процесі», а далі у «Завершено». Відображення змін наведено на рисунках 3.11-3.12.

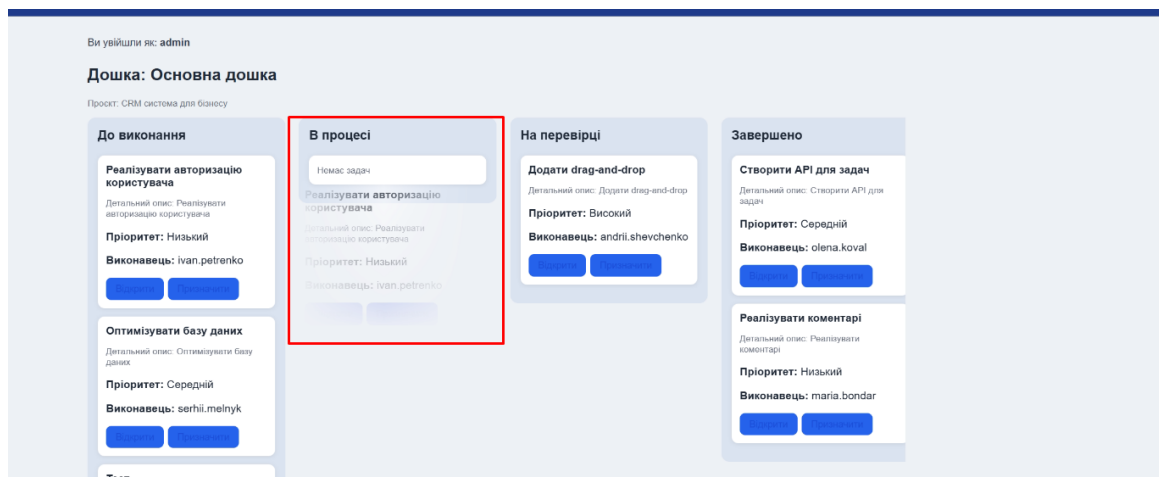


Рисунок 3.11 - Процес перетягування задачі

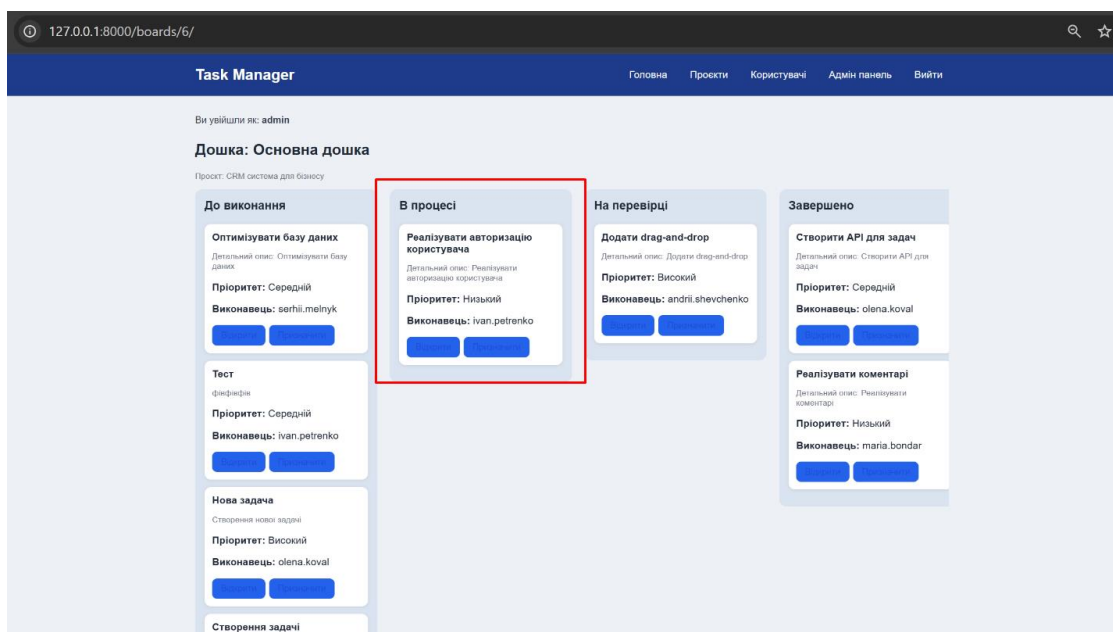


Рисунок 3.12 - Результат зміни статусу задачі

Окремо представлено сторінку задачі, на якій відображається детальна інформація про неї. Вона містить опис, пріоритет, виконавця, строки виконання та інші параметри (рис. 3.13).

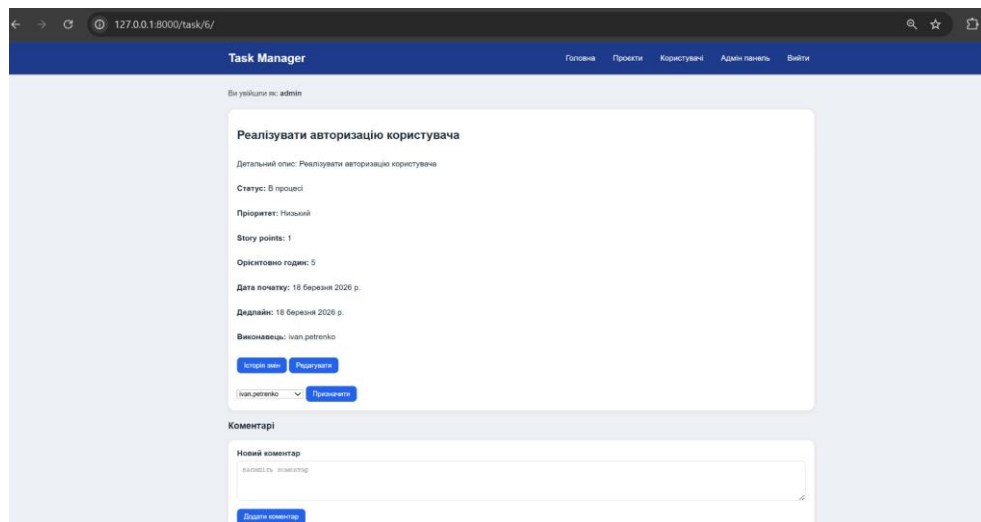


Рисунок 3.13 - Детальна інформація про задачу

На цій сторінці передбачено можливість редагування задачі та призначення іншого виконавця (для адміністратора). Це дає змогу оперативно вносити зміни до параметрів задачі відповідно до поточного стану роботи.

Також на сторінці реалізовано блок коментарів, через який користувачі можуть додавати повідомлення, пов'язані з виконанням задачі. Коментарі зберігаються у базі даних і відображаються у вигляді списку, що спрощує обмін інформацією в межах задачі. Узагальнено функціонал управління задачами охоплює:

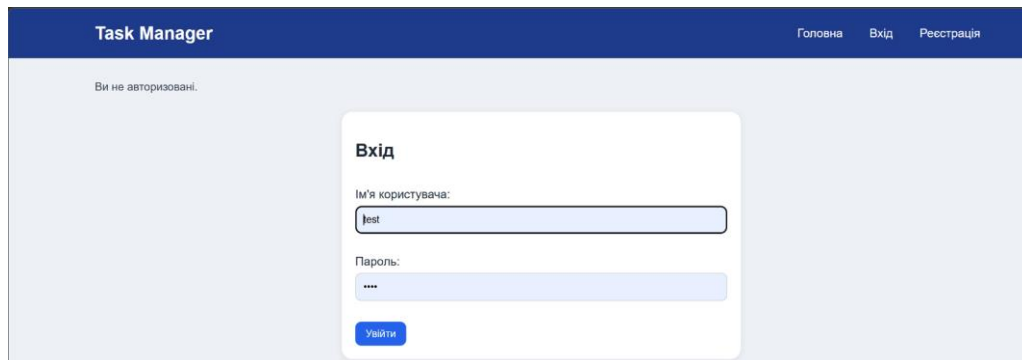
- створення задач (адміністратором);
- призначення виконавців;
- зміну статусу задач;
- перегляд детальної інформації;
- відстеження історії змін;
- роботу з коментарями.

Зазначені можливості інтегровані в межах єдиної системи, де кожен користувач працює з призначеними йому задачами, а адміністратор виконує функції управління та контролю.

3.4 Система авторизації та управління користувачами

У вебзастосунку передбачено систему авторизації, що розмежовує доступ до функціоналу залежно від ролі користувача. Це дає змогу обмежити виконання окремих дій та впорядкувати роботу із задачами.

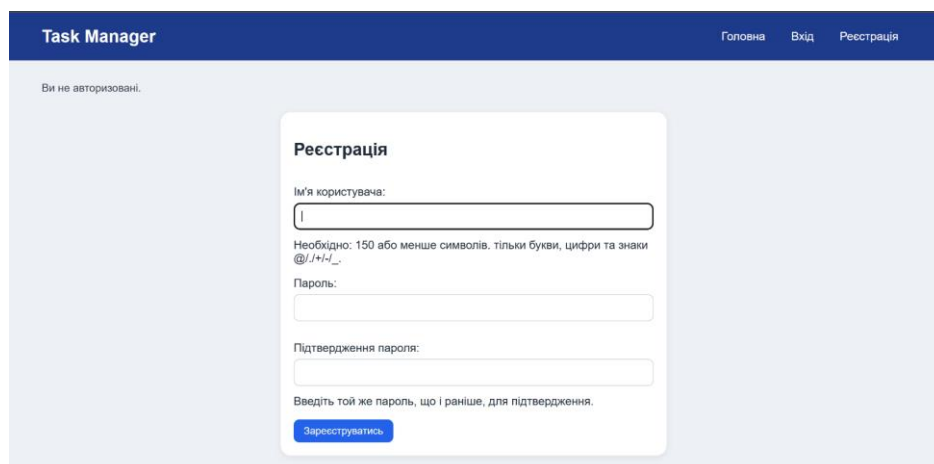
Вхід до системи здійснюється через форму авторизації, яка містить поля для введення імені користувача та пароля. Після перевірки введених даних відкривається доступ до відповідного функціоналу. Інтерфейс сторінки входу наведено на рисунку 3.14.



The screenshot shows the 'Task Manager' application interface. At the top, there is a dark blue header with the text 'Task Manager' on the left and navigation links 'Головна', 'Вхід', and 'Реєстрація' on the right. Below the header, the main content area has a light blue background. On the left, it says 'Ви не авторизовані.' In the center, there is a white box titled 'Вхід'. Inside this box, there are two input fields: 'Ім'я користувача:' with the value 'test' and 'Пароль:' with masked characters '****'. Below these fields is a blue button labeled 'Увійти'.

Рисунок 3.14 - Сторінка авторизації користувача

У системі передбачено можливість реєстрації нових користувачів (рис. 3.15). Під час реєстрації створюється обліковий запис, після чого відкривається доступ до системи.



The screenshot shows the 'Task Manager' application interface for registration. It has the same header as the login page. The main content area has a light blue background. On the left, it says 'Ви не авторизовані.' In the center, there is a white box titled 'Реєстрація'. Inside this box, there are three input fields: 'Ім'я користувача:' with the value 'I', 'Пароль:', and 'Підтвердження пароля:'. Below the password fields, there is a note: 'Необхідно: 150 або менше символів. тільки букви, цифри та знаки @/./+/-/_'. At the bottom of the box is a blue button labeled 'Зареєструватись'.

Рисунок 3.15 - Сторінка реєстрації користувача

Після авторизації відкривається доступ до особистого кабінету, у якому відображаються задачі, призначені поточному користувачу. Це спрощує орієнтацію в переліку завдань та їх подальше виконання (рис. 3.16).

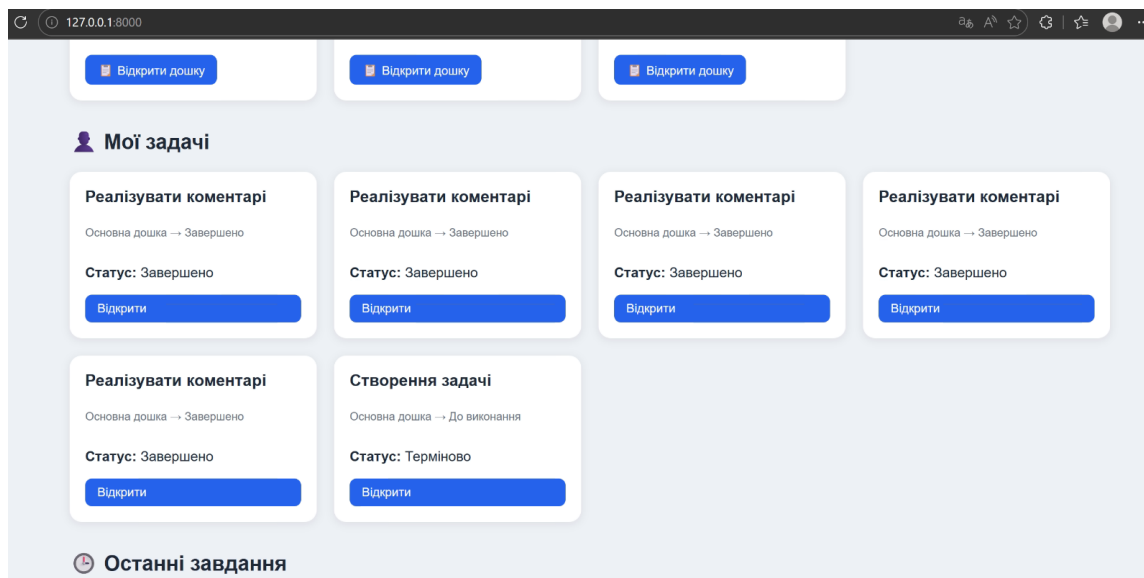


Рисунок 3.16 - Особистий кабінет користувача

У системі передбачено розподіл ролей. Адміністратор має розширені права, зокрема створення задач, призначення виконавців і зміну параметрів задач. Звичайні користувачі працюють із призначеними їм задачами та змінюють їх статус у процесі виконання.

3.5 Реалізація серверної логіки та обробки запитів

Серверна частина вебзастосунку побудована на основі фреймворку Django, який використовується для обробки запитів користувачів, взаємодії з базою даних та формування відповідей для клієнтської частини. Основна логіка системи зосереджена у файлах `views.py`, де обробляються HTTP-запити та виконується робота з моделями даних [31]-[32].

Для організації роботи застосунку використано механізм маршрутів Django. Кожен URL-адрес пов'язаний із відповідною функцією обробки, яка відповідає за виконання певної операції. Після отримання запиту сервер

виконує необхідну обробку даних, звертається до бази даних та передає результати у HTML-шаблони для подальшого відображення.

Однією з основних функцій серверної частини є формування інформаційної панелі головної сторінки. Для цього із бази даних отримуються проєкти, дошки та задачі, після чого дані передаються у шаблон для відображення статистики та списків елементів.

Окремо реалізовано механізм роботи із Kanban-дошкою. Під час відкриття дошки сервер отримує список стовпців та задач, які належать до відповідного проєкту. Після цього задачі групуються за статусами та передаються у шаблон для відображення у вигляді колонок.

Для перегляду детальної інформації про задачу використовується окрема функція обробки запиту, яка отримує об'єкт задачі за її ідентифікатором. Разом із задачею також завантажуються коментарі та історія змін, що пов'язані з нею.

Під час додавання коментарів використовується POST-запит. Після надсилання форми сервер отримує введені дані, створює новий запис у таблиці коментарів і виконує перенаправлення користувача назад на сторінку задачі.

Для зміни статусу задачі реалізовано окремий механізм переміщення між стовпцями. Після виконання відповідного запиту оновлюється статус задачі та створюється запис у таблиці історії змін.

У системі також використовується перевірка авторизації через декоратор `login_required`, який обмежує доступ до окремих функцій для неавторизованих користувачів. Це застосовується для роботи із задачами, коментарями та зміною статусів.

Приклад реалізації обробки запиту:

`@login_required`

```
def move_task(request, task_id, column_id):
```

```
    task = get_object_or_404(Task, id=task_id)
```

```
    new_column = get_object_or_404(Column, id=column_id)
```

```

task.column = new_column
task.status = new_column.name
task.save()

```

```

return redirect('board_detail', board_id=new_column.board.id)

```

У наведеному фрагменті коду виконується отримання задачі та нового стовпця, після чого оновлюється статус задачі і зберігаються зміни у базі даних. Основні маршрути системи наведено у таблиці 3.1.

Таблиця 3.1 - Основні ендпоінти системи

URL	Метод	Призначення
/	GET	Головна сторінка
/projects/	GET	Перегляд проєктів
/board/<id>/	GET	Перегляд Kanban-дошки
/task/<id>/	GET	Перегляд задачі
/task/<id>/comment/	POST	Додавання коментаря
/task/<id>/move/<column_id>/	POST	Зміна статусу задачі
/my-tasks/	GET	Особистий кабінет
/assign/<id>/	POST	Призначення виконавця
/accounts/login/	GET/POST	Авторизація
/accounts/register/	GET/POST	Реєстрація

Серверна логіка вебзастосунку охоплює основні процеси роботи із задачами та взаємодію з базою даних. Використання Django спрощує

організацію маршрутизації, обробку запитів і роботу з моделями даних, що робить структуру застосунку більш впорядкованою та зрозумілою.

3.6 Реалізація моделей та взаємодії з базою даних

Для організації зберігання даних у вебзастосунку використано ORM Django, яка забезпечує взаємодію між серверною частиною та реляційною базою даних. Використання ORM дозволяє працювати з даними через моделі Python без необхідності створення SQL-запитів вручну.

Структура бази даних побудована відповідно до предметної області системи управління завданнями. Основними моделями є користувачі, проєкти, дошки, стовпці, задачі, коментарі та історія змін. Кожна модель відповідає окремій таблиці бази даних і містить набір полів, необхідних для зберігання інформації.

Центральним елементом системи є модель задачі, оскільки саме вона пов'язує між собою основні компоненти застосунку. У моделі зберігаються назва задачі, опис, статус, пріоритет, виконавець та інші параметри.

Приклад реалізації моделі задачі наведено нижче:

```
class Task(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField()

    column = models.ForeignKey(
        Column,
        on_delete=models.CASCADE
    )

    assigned_user = models.ForeignKey(
        User,
        on_delete=models.SET_NULL,
```

```

    null=True,
    blank=True
)

```

```

status = models.CharField(max_length=100)
priority = models.CharField(max_length=50)

```

```

created_at = models.DateTimeField(auto_now_add=True)

```

У наведеному фрагменті коду використано поле `ForeignKey`, яке застосовується для створення зв'язків між таблицями. Зокрема, задача пов'язана зі стовпцем Канбан-дошки та користувачем, який відповідає за її виконання.

Для роботи з пов'язаними даними у системі використовуються ORM-запити Django [33]. Наприклад, під час відкриття сторінки задачі сервер одночасно отримує інформацію про виконавця, дошку та коментарі. Це спрощує доступ до взаємопов'язаних даних і зменшує кількість запитів до бази даних.

Окрему увагу приділено оптимізації вибірок. Для цього використовується метод `select_related`, який дозволяє завантажувати пов'язані об'єкти одним запитом.

Приклад такого запиту наведено нижче.

```

task = Task.objects.select_related(
    'assigned_user',
    'column',
    'column__board'
).get(id=task_id)

```

У наведеному прикладі разом із задачею завантажуються дані про користувача, стовпець і дошку, до якої належить задача. Це зменшує кількість окремих звернень до бази даних і покращує продуктивність застосунку.

Для зберігання коментарів використовується окрема модель, пов'язана із задачами та користувачами. Під час додавання нового коментаря створюється відповідний запис у таблиці коментарів, після чого інформація відображається на сторінці задачі.

Модель історії змін використовується для фіксації переміщення задач між стовпцями та зміни їх статусів. Кожен запис містить попередній і новий стан задачі, а також інформацію про користувача, який виконав зміну.

Взаємодія з базою даних у межах застосунку організована через ORM Django, що спрощує роботу з даними, забезпечує підтримку зв'язків між таблицями та робить структуру серверної частини більш зрозумілою і впорядкованою.

3.7 Реалізація додаткового функціоналу системи

Окрім базових можливостей роботи із задачами, у вебзастосунку передбачено додатковий функціонал, який розширює можливості системи та покращує взаємодію користувачів із задачами. До такого функціоналу належать система історії змін, персональний кабінет користувача, механізм призначення виконавців, система коментарів та перевірка ролей доступу.

Однією з важливих складових є механізм відстеження історії змін задач. Під час зміни статусу задачі або її переміщення між стовпцями створюється окремий запис у таблиці історії. У записі зберігаються попередній і новий статус, інформація про стовпці та користувача, який виконав зміну.

Приклад створення запису історії наведено нижче.

```
TaskHistory.objects.create(  
    task=task,  
    old_column=old_column,  
    new_column=new_column,  
    old_status=task.status,
```

```

        new_status=new_column.name,
        changed_by=request.user
    )

```

Після виконання цієї операції інформація про зміну відображається на сторінці історії задач. Такий підхід дозволяє фіксувати послідовність дій і відстежувати процес виконання задач.

У системі також передбачено персональний кабінет користувача. На сторінці відображаються лише ті задачі, які призначені поточному виконавцю. Формування списку здійснюється шляхом фільтрації даних за користувачем. Приклад вибірки задач:

```

tasks = Task.objects.filter(
    assigned_user=request.user
)

```

У результаті користувач отримує окремий список власних задач без необхідності переглядати всі записи системи.

Для адміністратора передбачено окремий механізм призначення задач виконавцям. Після вибору користувача система оновлює відповідний запис задачі та зберігає зміни у базі даних. Приклад зміни виконавця:

```

task.assigned_user = user
task.save()

```

Додатково у вебзастосунку реалізовано систему коментарів, інтегровану безпосередньо у сторінку задачі. Під час додавання нового повідомлення створюється окремий запис у таблиці коментарів із прив'язкою до задачі та автора повідомлення. Приклад створення коментаря:

```

Comment.objects.create(
    task=task,
    user=request.user,
    content=content
)

```

Коментарі відображаються у хронологічному порядку та містять інформацію про автора і дату створення.

Окрему увагу приділено перевірці ролей користувачів. Для обмеження доступу до окремих функцій використовується перевірка прав адміністратора. Наприклад, створення задач і призначення виконавців доступне лише користувачам із відповідними правами. Приклад перевірки ролі:

```
If not request.user.is_staff:  
    return redirect('dashboard')
```

У наведеному фрагменті виконується перевірка прав доступу, після чого користувач без ролі адміністратора перенаправляється на головну сторінку.

3.8 Висновки до розділу 3

У третьому розділі розглянуто практичну реалізацію вебзастосунку для управління завданнями. У процесі розробки сформовано серверну та клієнтську частини системи, що взаємодіють між собою та забезпечують роботу застосунку.

На серверному рівні організовано обробку даних і взаємодію з базою даних. Створено моделі, які відображають структуру предметної області, а також налаштовано обробку запитів, пов'язаних із управлінням задачами.

Клієнтська частина представлена вебінтерфейсом, через який здійснюється робота з проєктами, дошками, задачами та користувачами. Структура сторінок і навігація побудовані таким чином, щоб спростити доступ до основних функцій системи.

Окремо розглянуто функціонал управління завданнями, що охоплює створення задач, призначення виконавців, зміну статусів, перегляд детальної інформації та історії змін. Усі ці операції пов'язані між собою та формують єдиний процес роботи із задачами.

Візуалізація виконання задач організована у вигляді Kanban-дошки, де завдання розподіляються за статусами та відображаються у відповідних стовпцях.

Також розглянуто систему авторизації та управління користувачами, у межах якої визначено розподіл ролей і доступ до функціоналу. Адміністратор виконує функції управління задачами, тоді як користувачі працюють із призначеними їм завданнями.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Функціональне тестування системи

Під час тестування було перевірено основні функції системи: авторизацію користувачів, створення та редагування задач, призначення виконавців, зміну статусів, перегляд інформації про задачі та роботу з коментарями. Тестування здійснювалося шляхом виконання типових сценаріїв використання системи. Результати наведено у вигляді тест-кейсів у таблиці 4.1.

Таблиця 4.1 - Результати функціонального тестування системи

№	Функція	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Авторизація користувача	Логін: olena.koval, пароль: 1234	Успішний вхід у систему	Вхід виконано	Успішно
2	Невірна авторизація	Логін: olena.koval, пароль: 0000	Повідомлення про помилку	Помилка відображена	Успішно
3	Створення задачі	Заповнені всі поля	Задача створена	Задача збережена	Успішно
4	Призначення задачі	Вибір користувача	Виконавець змінений	Дані оновлено	Успішно
5	Зміна статусу задачі	Переміщення задачі	Статус оновлено	Статус змінено	Успішно
6	Перегляд задачі	Перехід до сторінки	Відображення інформації	Дані відображено	Успішно
7	Додавання коментаря	Введення тексту	Коментар додано	Коментар збережено	Успішно

Проведене тестування показало, що всі основні функції системи працюють стабільно та без помилок. Авторизація забезпечує доступ лише для зареєстрованих користувачів, створення та редагування задач відбувається коректно, а зміна статусів миттєво відображається у відповідних стовпцях Kanban-дошки.

Окремо було перевірено взаємодію між компонентами системи. Зокрема, при зміні статусу задачі відбувається не лише оновлення запису в базі даних, а й коректне відображення змін у користувацькому інтерфейсі.

Для наочності виконання окремих тестових сценаріїв наведено приклади роботи системи (рис. 4.1-4.4).

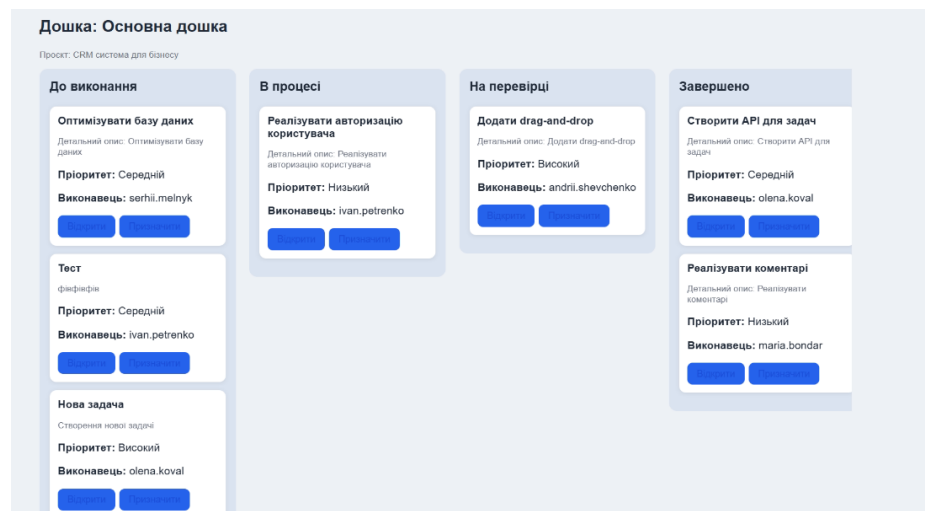


Рисунок 4.1 - Стан задач до зміни статусу

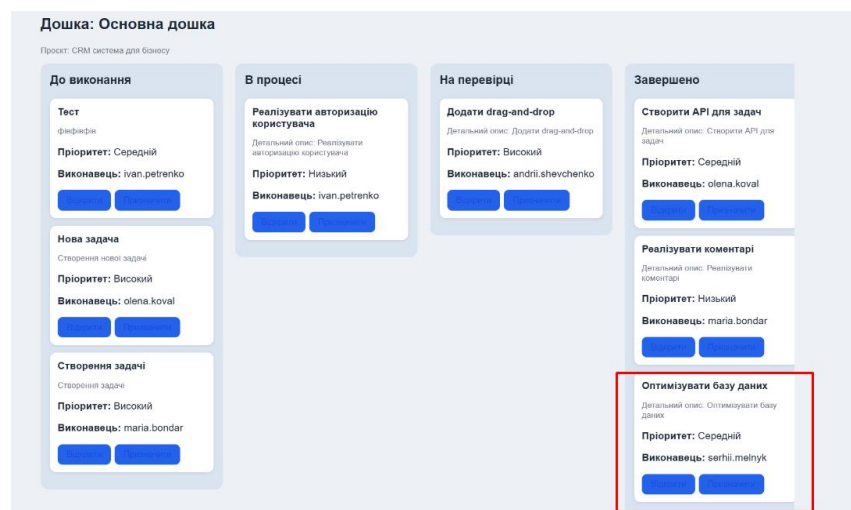
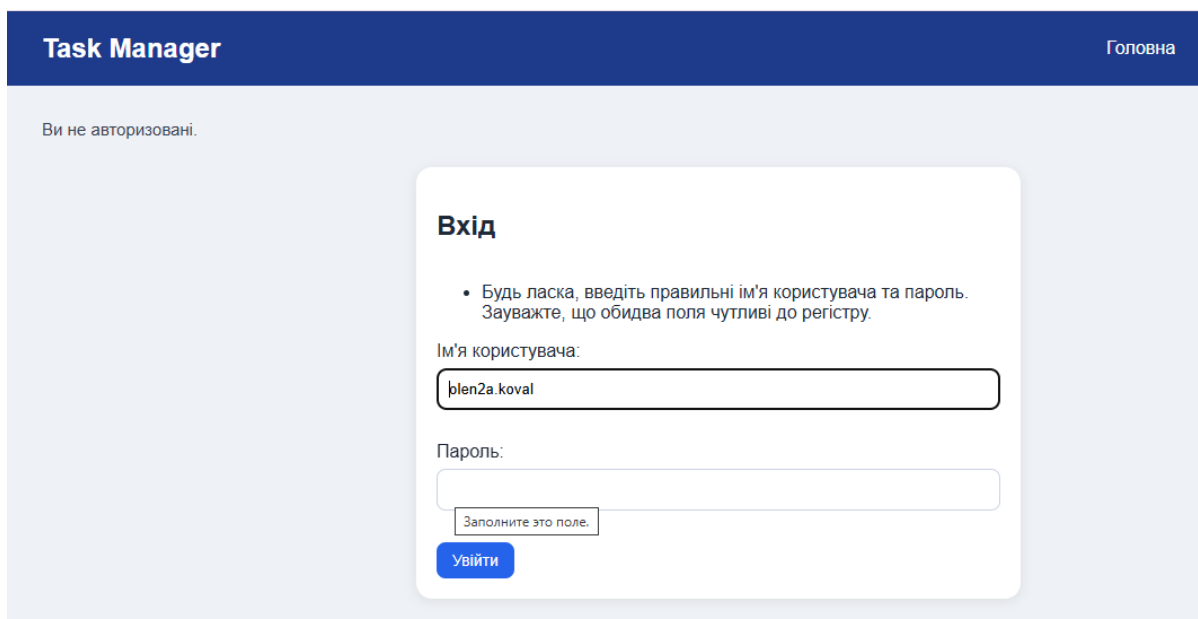
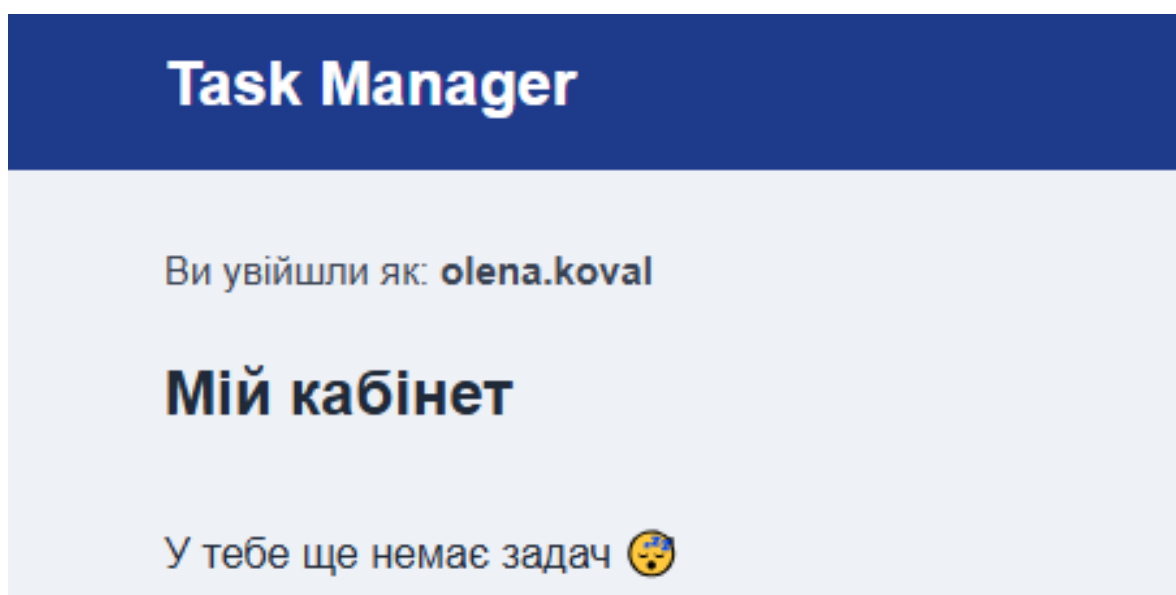


Рисунок 4.2 - Результат зміни статусу задачі



The screenshot shows the 'Task Manager' application interface. At the top, there is a dark blue header with the text 'Task Manager' on the left and 'Головна' (Home) on the right. Below the header, the main area has a light blue background. A message 'Ви не авторизовані.' (You are not authorized.) is displayed. In the center, there is a white login form titled 'Вхід' (Login). The form contains a list of instructions: 'Будь ласка, введіть правильні ім'я користувача та пароль. Зауважте, що обидва поля чутливі до регістру.' (Please enter the correct username and password. Note that both fields are case-sensitive.). Below this, there are two input fields: 'Ім'я користувача:' (Username:) with the value 'olen2a.koval' and 'Пароль:' (Password:). A small error message 'Заполните это поле.' (Fill in this field.) is visible below the password field. At the bottom of the form is a blue button labeled 'Увійти' (Login).

Рисунок 4.3 - Невірна авторизація



The screenshot shows the 'Task Manager' application interface after a successful login. The dark blue header at the top contains the text 'Task Manager'. Below the header, the main area has a light blue background. A message 'Ви увійшли як: olena.koval' (You have logged in as: olena.koval) is displayed. Below this, the text 'Мій кабінет' (My cabinet) is shown. At the bottom, there is a message 'У тебе ще немає задач' (You still have no tasks) followed by a puzzle piece icon.

Рисунок 4.4 - Успішна реєстрація

4.2 Аналіз результатів тестування

Після виконання функціонального тестування було проведено аналіз отриманих результатів з метою оцінки якості роботи системи, її стабільності та відповідності поставленим вимогам.

Усі перевірені функції системи продемонстрували коректну роботу. Зокрема, процес авторизації забезпечує надійну перевірку облікових даних

користувача. При введенні правильних даних виконується успішний вхід до системи, тоді як у випадку помилки користувач отримує відповідне повідомлення.

Функціонал створення задач працює стабільно: після заповнення форми всі дані коректно зберігаються у базі даних і одразу відображаються у відповідному стовпці Kanban-дошки. Аналогічно, при зміні виконавця або редагуванні задачі всі зміни миттєво застосовуються.

Особливу увагу було приділено перевірці зміни статусу задач. Під час тестування встановлено, що після оновлення статусу задача коректно переміщується між стовпцями, а також фіксується в історії змін. Це підтверджує узгоджену роботу між серверною частиною та інтерфейсом користувача.

Для більш детального аналізу було узагальнено результати тестування у вигляді таблиці 4.2.

Таблиця 4.2 - Оцінка результатів тестування системи

№	Функціональний модуль	Критерій перевірки	Результат	Оцінка
1	2	3	4	5
1	Авторизація	Коректність входу	Працює без помилок	Висока
2	Створення задач	Збереження даних	Дані зберігаються коректно	Висока
3	Призначення задач	Зміна виконавця	Виконавець оновлюється	Висока
4	Kanban-дошка	Відображення задач	Дані відображаються правильно	Висока
5	Зміна статусу	Переміщення задач	Працює без затримок	Висока

Кінець таблиці 4.2

1	2	3	4	5
6	Коментарі	Додавання та відображення	Коментарі зберігаються	Висока
7	Історія змін	Фіксація дій	Записи створюються	Висока
8	Інтерфейс	Зручність використання	Потребує покращення	Середня

Як видно з таблиці, більшість функціональних модулів отримали високу оцінку, що свідчить про стабільну роботу системи. Основні операції виконуються без помилок і відповідають очікуваному результату.

Разом із тим було виявлено окремі недоліки, що стосуються інтерфейсу користувача. Зокрема, спостерігаються незначні проблеми з вирівнюванням елементів та відображенням окремих блоків. Хоча ці недоліки не впливають на функціональність системи, їх усунення дозволить покращити загальне сприйняття застосунку.

Додатково варто зазначити, що система демонструє швидку реакцію на дії користувача: створення задач, зміна статусу та додавання коментарів виконуються без помітних затримок. Це є важливим показником ефективності роботи вебзастосунку.

4.3 Висновки до розділу 4

У четвертому розділі було проведено тестування розробленої системи управління завданнями та виконано аналіз отриманих результатів. Основну увагу приділено перевірці функціональності системи, стабільності її роботи та відповідності реалізованого функціоналу поставленим вимогам.

У процесі функціонального тестування було перевірено ключові можливості системи, зокрема авторизацію користувачів, створення та

редагування задач, призначення виконавців, зміну статусів, роботу Kanban-дошки, а також механізм коментарів і історії змін. Результати тестування показали, що всі основні функції працюють коректно та забезпечують виконання необхідних операцій без помилок.

Аналіз результатів тестування підтвердив узгоджену роботу серверної та клієнтської частин системи. Зміни, що виконуються користувачем, миттєво відображаються в інтерфейсі, що забезпечує зручність використання та ефективність взаємодії із системою.

У ході тестування критичних помилок виявлено не було. Водночас було визначено окремі незначні недоліки, пов'язані з оформленням інтерфейсу, зокрема вирівнюванням елементів та відображенням окремих компонентів. Дані зауваження не впливають на функціональність системи та можуть бути усунені у процесі подальшого вдосконалення.

Загалом результати тестування свідчать про те, що розроблена система є працездатною, стабільною та відповідає вимогам, поставленим на етапі проєктування. Вона може бути використана для організації роботи з завданнями та забезпечує ефективне управління діяльністю користувачів у межах проєкту.

ВИСНОВКИ

У роботі розглянуто процес розробки вебзастосунку для управління завданнями, який орієнтований на організацію роботи користувачів у межах проєкту. У межах дослідження проаналізовано предметну область, визначено основні вимоги до системи та розглянуто існуючі програмні рішення, що використовуються для управління завданнями.

На основі проведеного аналізу обрано підхід до побудови системи, визначено її архітектуру та спроектовано структуру бази даних. Особливу увагу приділено взаємозв'язкам між сутностями, що забезпечує цілісність даних і коректну роботу застосунку.

У процесі розробки реалізовано серверну частину на основі фреймворку Django та клієнтську частину з використанням HTML і CSS. Побудовано вебінтерфейс, що охоплює основні сценарії роботи із задачами, включаючи їх створення, призначення, зміну статусу, перегляд детальної інформації та ведення історії змін.

Окремо розглянуто реалізацію Kanban-дошки, яка використовується для візуалізації процесу виконання завдань. Представлення задач у вигляді карток і їх розподіл за статусами дозволяє відобразити поточний стан роботи в межах проєкту.

Також у системі передбачено механізм авторизації та розподіл ролей користувачів. Адміністратор виконує функції управління задачами, тоді як користувачі працюють із призначеними їм завданнями. Даний підхід забезпечує впорядковану організацію роботи та розмежування доступу до функціоналу.

Проведене тестування підтвердило коректність роботи основних функцій системи та відповідність реалізованого функціоналу поставленим вимогам. Отримані результати свідчать про працездатність застосунку та можливість його використання для організації роботи невеликих команд.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. Київ : ДП «УкрНДНЦ», 2016. 26 с.
2. ДСТУ ISO/IEC 12207:2016. Інженерія систем і програмних засобів. Процеси життєвого циклу програмного забезпечення. Київ : ДП «УкрНДНЦ», 2018. 98 с.
3. ДСТУ ISO/IEC/IEEE 15289:2017. Системна та програмна інженерія. Вміст інформаційних продуктів життєвого циклу програмного забезпечення. Київ : ДП «УкрНДНЦ», 2018. 44 с.
4. Морзе Н. В., Вембер В. П. Основи вебтехнологій : навч. посіб. Київ : Академвидав, 2021. 256 с.
5. Гужва В. М. Інформаційні системи і технології на підприємствах : навч. посіб. Київ : КНЕУ, 2019. 400 с.
6. Пасічник В. В., Резніченко В. А. Організація баз даних та знань : підручник. Київ : Видавнича група BVH, 2020. 384 с.
7. Кучерявий В. П. Бази даних : навч. посіб. Львів : Новий Світ-2000, 2021. 312 с.
8. Биков В. Ю. Інформаційні системи і технології : навч. посіб. Київ : Центр учбової літератури, 2020. 304 с.
9. Литвин В. В., Пасічник В. В. Проєктування інформаційних систем : навч. посіб. Львів : Магнолія-2006, 2020. 380 с.
10. How we use Trello on the GOV.UK Publishing Platform team. URL: <https://insidegovuk.blog.gov.uk/2017/03/13/how-we-use-trello-on-the-gov-uk-publishing-platform-team/> (date of access: 07.06.2026).
11. Рудаков В. О. Проєктування інформаційних систем : навч. посіб. Харків : ХНЕУ ім. С. Кузнеця, 2020. 268 с.

12. 6 найкращих онлайн-інструментів для управління завданнями в команді. URL: <https://www.chanty.com/blog/uk/task-management-tools-uk/> (дата звернення: 07.06.2026).
13. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2019. 880 p.
14. Sommerville I. Software Engineering. 10th ed. Boston : Pearson Education, 2016. 816 p.
15. Elmasri R., Navathe S. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2017. 1272 p.
16. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2004. 208 p.
17. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/> (date of access: 11.05.2026).
18. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Boston : Prentice Hall, 2008. 464 p.
19. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
20. Duckett J. HTML and CSS: Design and Build Websites. Indianapolis : Wiley, 2014. 490 p.
21. Duckett J. JavaScript and JQuery: Interactive Front-End Web Development. Indianapolis : Wiley, 2015. 640 p.
22. Mozilla Developer Network. Django Web Framework (Python). URL: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django> (date of access: 09.05.2026).
23. Django Software Foundation. Django documentation. URL: <https://docs.djangoproject.com/> (date of access: 05.05.2026).
24. Django Software Foundation. Django web framework. URL: <https://www.djangoproject.com/> (date of access: 13.05.2026).

25. Mozilla Developer Network. CSS: Cascading Style Sheets. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (date of access: 07.05.2026).
26. Atlassian. Scrum methodology. URL: <https://www.atlassian.com/agile/scrum> (date of access: 10.05.2026).
27. Mozilla Developer Network. HTTP overview. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview> (date of access: 01.05.2026).
28. Mozilla Developer Network. HTML: HyperText Markup Language. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (date of access: 13.05.2026).
29. Kanban University. The Official Guide to The Kanban Method. URL: <https://kanban.university/kanban-guide/> (date of access: 13.05.2026).
30. W3Schools. Python Tutorial. URL: <https://www.w3schools.com/python/> (date of access: 12.05.2026).
31. Atlassian. What is Kanban? URL: <https://www.atlassian.com/agile/kanban> (date of access: 11.05.2026).
32. W3Schools. SQL Tutorial. URL: <https://www.w3schools.com/sql/> (date of access: 13.05.2026).
33. Mozilla Developer Network. JavaScript Guide. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide> (date of access: 13.05.2026).

ДОДАТОК А
Текст програми

A.1 Текст файла Views.py

```

from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import login
from django.db.models import Q

from .models import Project, Board, Column, Task, Comment, TaskHistory
from django.contrib.admin.views.decorators import staff_member_required
from django.contrib.admin.views.decorators import staff_member_required

def dashboard(request):
    projects = Project.objects.all()
    boards = Board.objects.all()
    tasks = Task.objects.all()

    recent_tasks = Task.objects.order_by('-created_at')[:10]

    user_tasks = Task.objects.filter(
        assigned_user=request.user
    )

    return render(request, 'core/dashboard.html', {
        'projects': projects,
        'projects_count': projects.count(),
        'boards_count': boards.count(),
        'tasks_count': tasks.count(),
        'recent_tasks': recent_tasks,
        'user_tasks': user_tasks,
    })

def project_list(request):
    if request.user.is_authenticated:

        user_tasks = Task.objects.filter(
            assigned_user=request.user
        ).select_related('column__board')

        project_ids = set()

        for task in user_tasks:
            if task.column and task.column.board and task.column.board.project:

```

```

        project_ids.add(task.column.board.project.id)

    projects = Project.objects.filter(
        Q(owner=request.user) | Q(id__in=project_ids)
    ).distinct().select_related('owner')

else:
    projects = Project.objects.all().select_related('owner')

return render(request, 'core/project_list.html', {
    'projects': projects
})

def board_detail(request, board_id):
    board = get_object_or_404(Board, id=board_id)
    columns = Column.objects.filter(board=board).order_by('position')

    data = []
    for column in columns:
        tasks = Task.objects.filter(column=column).select_related('assigned_user')
        data.append({
            'column': column,
            'tasks': tasks
        })

    return render(request, 'core/board_detail.html', {
        'board': board,
        'data': data
    })

def task_detail(request, task_id):
    task = get_object_or_404(
        Task.objects.select_related('assigned_user', 'column', 'column__board'),
        id=task_id
    )

    comments = Comment.objects.filter(task=task)\
        .select_related('user')\
        .order_by('-created_at')

    users = User.objects.all()

```

```

return render(request, 'core/task_detail.html', {
    'task': task,
    'comments': comments,
    'users': users
})

```

```

@login_required
def add_comment(request, task_id):
    task = get_object_or_404(Task, id=task_id)

    if request.method == 'POST':
        content = request.POST.get('content', "").strip()
        if content:
            Comment.objects.create(
                task=task,
                user=request.user,
                content=content
            )

    return redirect('task_detail', task_id=task.id)

```

```

@login_required
def move_task(request, task_id, column_id):
    task = get_object_or_404(Task, id=task_id)
    new_column = get_object_or_404(Column, id=column_id)
    old_column = task.column

    TaskHistory.objects.create(
        task=task,
        old_column_id=old_column.id,
        new_column_id=new_column.id,
        old_status=task.status,
        new_status=new_column.name,
        changed_by=request.user
    )

    task.column = new_column
    task.status = new_column.name
    task.save()

    return redirect('board_detail', board_id=new_column.board.id)

def task_history(request, task_id):

```

```

task = get_object_or_404(Task, id=task_id)

history = TaskHistory.objects.filter(task=task)\
    .select_related('changed_by')\
    .order_by('-changed_at')

return render(request, 'core/history.html', {
    'task': task,
    'history': history
})

def users_list(request):
    users = User.objects.all().order_by('username')
    return render(request, 'core/users_list.html', {
        'users': users
    })

@login_required
def close_task(request, task_id):
    task = get_object_or_404(Task, id=task_id)

    done_column = Column.objects.filter(
        board=task.column.board
    ).order_by('-position').first()

    TaskHistory.objects.create(
        task=task,
        old_column_id=task.column.id,
        new_column_id=done_column.id if done_column else task.column.id,
        old_status=task.status,
        new_status="Завершено",
        changed_by=request.user
    )

    task.status = "Завершено"

    if done_column:
        task.column = done_column

    task.save()

    return redirect('task_detail', task_id=task.id)

@login_required

```

```

def assign_task(request, task_id):
    task = get_object_or_404(Task, id=task_id)

    if not request.user.is_staff:
        return redirect('board_detail', board_id=task.column.board.id)

    if request.method == 'POST':
        user_id = request.POST.get('user_id')
        user = get_object_or_404(User, id=user_id)

        task.assigned_user = user
        task.save()

        return redirect('task_detail', task.id)

    users = User.objects.all()

    return render(request, 'core/assign.html', {
        'task': task,
        'users': users
    })

```

```

@login_required
def edit_task(request, task_id):
    task = get_object_or_404(Task, id=task_id)

    if request.method == 'POST':
        old_hours = task.estimated_hours
        old_title = task.title

        new_title = request.POST.get('title')
        new_description = request.POST.get('description')
        new_priority = request.POST.get('priority')
        new_hours = request.POST.get('estimated_hours')

        task.title = new_title
        task.description = new_description
        task.priority = new_priority
        task.estimated_hours = new_hours
        task.save()

        if str(old_hours) != str(new_hours):
            TaskHistory.objects.create(
                task=task,

```

```

        old_column_id=task.column.id,
        new_column_id=task.column.id,
        old_status=f'Години: {old_hours}',
        new_status=f'Години: {new_hours}',
        changed_by=request.user
    )

    if old_title != new_title:
        TaskHistory.objects.create(
            task=task,
            old_column_id=task.column.id,
            new_column_id=task.column.id,
            old_status=f'Назва: {old_title}',
            new_status=f'Назва: {new_title}',
            changed_by=request.user
        )

    return redirect('task_detail', task_id=task.id)

return render(request, 'core/edit_task.html', {
    'task': task
})

@login_required
def profile(request):
    return render(request, 'core/profile.html')

def register(request):
    form = UserCreationForm()

    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('dashboard')

    return render(request, 'registration/register.html', {
        'form': form
    })

@login_required
def create_board(request, project_id):
    project = get_object_or_404(Project, id=project_id)

```

```

if request.method == 'POST':
    name = request.POST.get('name')

    board = Board.objects.create(
        name=name,
        project=project
    )

    Column.objects.create(name="До виконання", board=board, position=1)
    Column.objects.create(name="В процесі", board=board, position=2)
    Column.objects.create(name="На перевірці", board=board, position=3)
    Column.objects.create(name="Завершено", board=board, position=4)

    return redirect('board_detail', board_id=board.id)

return render(request, 'core/create_board.html', {
    'project': project
})

```

```

@staff_member_required
def admin_panel(request):
    tasks = Task.objects.select_related(
        'assigned_user',
        'column',
        'column__board'
    )

    # ФІЛЬТРИ
    status = request.GET.get('status')
    user_id = request.GET.get('user')

    if status:
        tasks = tasks.filter(status=status)

    if user_id:
        tasks = tasks.filter(assigned_user_id=user_id)

    users = User.objects.all()

    return render(request, 'core/admin_panel.html', {
        'tasks': tasks.order_by('-created_at'),
        'users': users,
        'tasks_count': tasks.count(),
        'projects_count': Project.objects.count(),
        'users_count': User.objects.count(),
    })

```

```

    })

    @staff_member_required
    def create_task(request):
        columns = Column.objects.all()
        users = User.objects.all()

        if request.method == 'POST':
            Task.objects.create(
                title=request.POST.get('title'),
                description=request.POST.get('description'),
                column_id=request.POST.get('column'),
                assigned_user_id=request.POST.get('user') or None,
                status=request.POST.get('status'),
                priority=request.POST.get('priority'),
            )
            return redirect('admin_panel')

        return render(request, 'core/create_task.html', {
            'columns': columns,
            'users': users
        })

    @login_required
    def create_project(request):
        if request.method == 'POST':
            name = request.POST.get('name')
            description = request.POST.get('description')

            Project.objects.create(
                name=name,
                description=description,
                owner=request.user
            )

            return redirect('project_list')

        return render(request, 'core/create_project.html')

```

ДОДАТОК Б
Слайди презентації

Дипломна кваліфікаційна робота бакалавра
Тема: «Створення програмної
вебсистеми для управління завданнями з
використанням Agile-підходів»

Виконала: ст. гр. КНТ-222

Керівник: к.т.н., доцент

Вікторія АРТЮХ

Тетяна ФЕДОРОНЧАК

Рисунок Б.1 - Слайд 1 «Титульний слайд»

Об'єкт, предмет та мета дослідження

Об'єкт дослідження

Процес організації та управління завданнями у межах командної роботи над проектами.

Предмет дослідження

Методи та програмні засоби управління завданнями і контролю виконання робіт у веборієнтованих системах.

Мета роботи

Підвищення ефективності організації та контролю виконання завдань шляхом розробки веборієнтованої системи управління завданнями.

- ① Система орієнтована на практичне застосування у реальних командних проектах з підтримкою Agile-методологій.

Рисунок Б.2 - Слайд 2 «Об'єкт, предмет та мета дослідження»

Основні завдання роботи

Аналіз предметної області Дослідження існуючих підходів до управління завданнями	Дослідження Agile-підходів Вивчення Scrum, Kanban та інших гнучких методологій	Аналіз існуючих систем Порівняльна оцінка аналогів та визначення вимог
Проектування архітектури Розробка структури системи та бази даних	Реалізація та тестування Розробка вебзастосунку та перевірка його працездатності	

3

Рисунок Б.3 - Слайд 3 «Основні завдання роботи»

Agile-підходи до розробки

- Scrum**
 - Робота короткими ітераціями — спринтами
 - Чітко визначені ролі учасників команди
 - Планування та ретроспектива кожного спринту
- Kanban**
 - Візуалізація завдань на дошці
 - Безперервний потік виконання робіт
 - Обмеження паралельного навантаження (WIP)

✔ **Переваги Agile:** гнучкість, швидка адаптація до змін, постійний контроль виконання задач.

4

Рисунок Б.4 - Слайд 4 «Agile-підходи до розробки»

Аналіз існуючих рішень

Порівняння популярних систем управління завданнями дозволило виявити ключові переваги та обмеження кожного рішення.

Система	Переваги	Недоліки
Trello	Простота використання, інтуїтивна Kanban-дошка	Обмежений функціонал для складних проєктів
Jira	Потужна аналітика, гнучке налаштування workflow	Висока складність, перевантажений інтерфейс
Asana	Зручний інтерфейс, гарна візуалізація	Значна частина функцій доступна лише у платній версії

⚠ Існуючі системи є або надто простими для командної роботи, або перевантаженими функціоналом. Виникає потреба у збалансованому рішенні.

Рисунок Б.5 - Слайд 5 «Аналіз існуючих рішень»

Архітектура вебзастосунку

Система побудована за класичною тривірневою архітектурою, що забезпечує чіткий поділ відповідальностей між компонентами.

- Клієнтська частина — HTML, CSS, JavaScript
- Серверна частина — Django (Python)
- База даних — SQLite

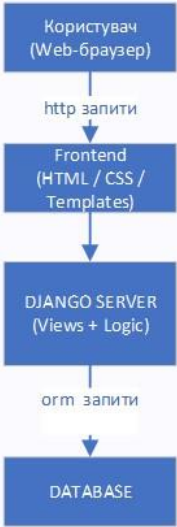


Рисунок Б.6 - Слайд 6 «Архітектура вебзастосунку»

Використані технології



Python

Мова програмування для реалізації серверної логіки та бізнес-правил системи



HTML / CSS

Структура та стилізація вебсторінок, адаптивний дизайн інтерфейсу



SQLite

Легка реляційна база даних для зберігання завдань, проєктів та користувачів



Django

Вебфреймворк для побудови backend-частини з вбудованою адмін-панеллю



JavaScript

Інтерактивність клієнтської частини, динамічне оновлення без перезавантаження



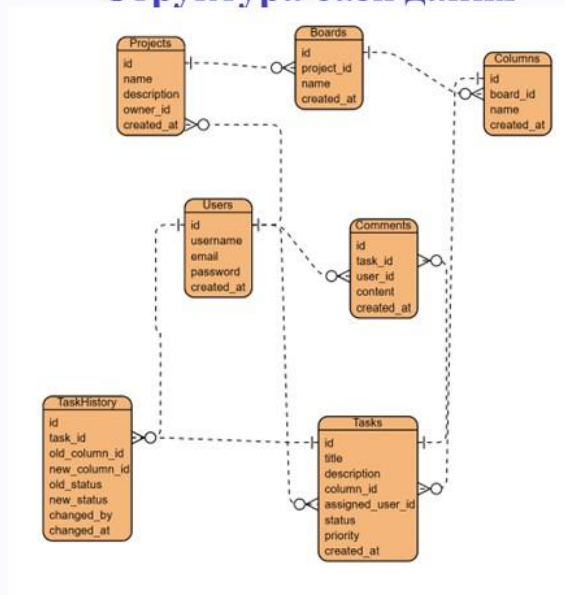
VS Code

Середовище розробки з підтримкою Python, Django та систем контролю версій

7

Рисунок Б.7 - Слайд 7 «Використані технології»

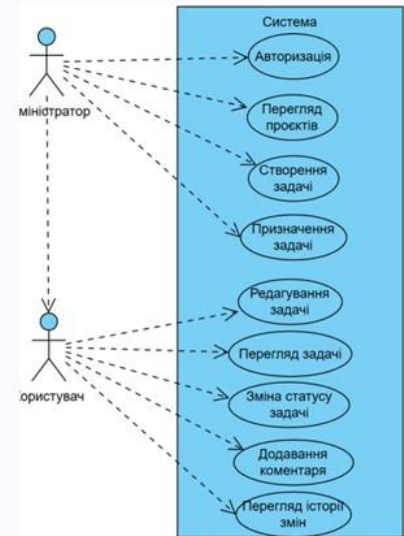
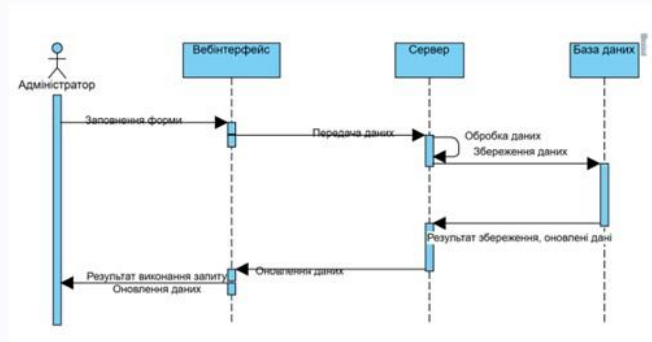
Структура бази даних



8

Рисунок Б.8 - Слайд 8 «Структура бази даних»

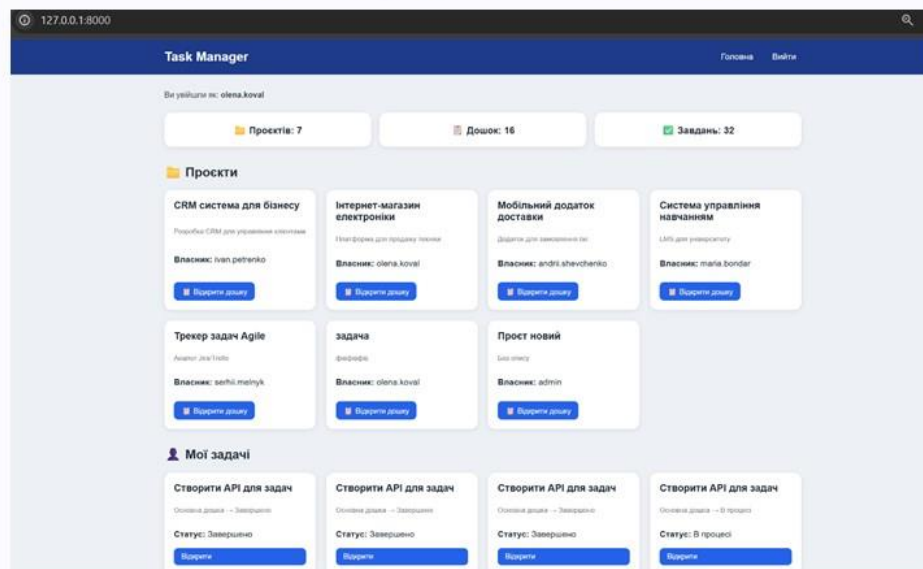
UML-діаграми системи



9

Рисунок Б.9 - Слайд 9 «UML-діаграми системи»

Головна сторінка — Dashboard



10

Рисунок Б.10 - Слайд 10 «Головна сторінка — Dashboard»

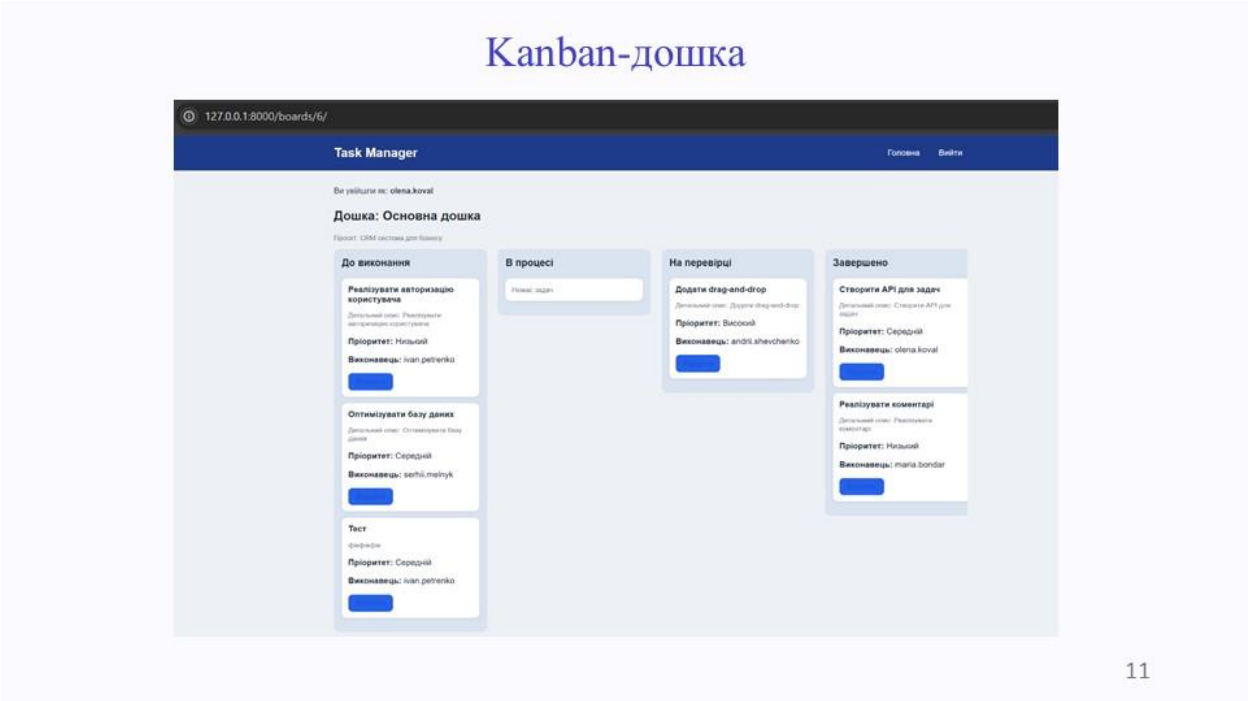


Рисунок Б.11 - Слайд 11 «Kanban-дошка»

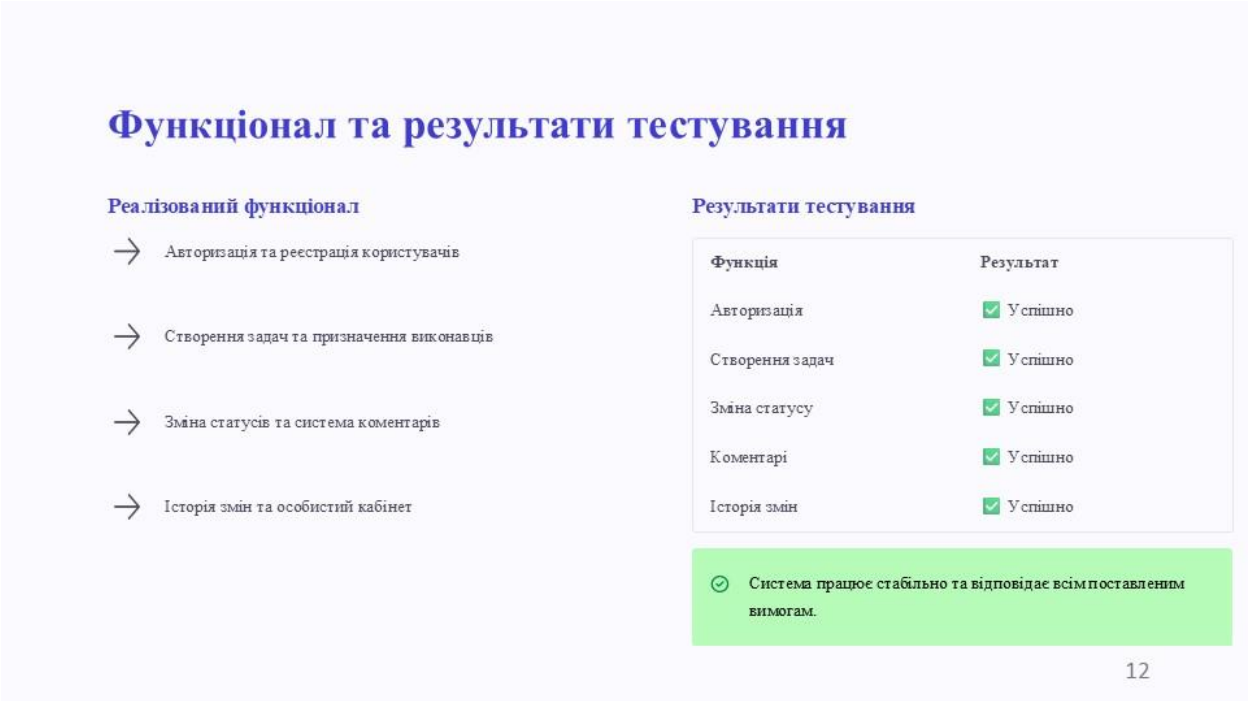


Рисунок Б.12 - Слайд 12 «Функціонал та результати тестування»

Висновки

Мету роботи досягнуто

Розроблено веборієнтовану систему управління завданнями для підтримки командної роботи над проектами.

Основний функціонал реалізовано

Забезпечено створення, призначення та контроль виконання завдань, а також управління їх статусами.

Працездатність системи підтверджено

Функціональне тестування засвідчило коректну роботу основних компонентів вебзастосунку.

Практичну цінність доведено

Система може використовуватися для організації та контролю виконання завдань у проектній діяльності.

Дякую за увагу!

Рисунок Б.13 - Слайд 13 «Висновки»