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

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

СИСТЕМА УПРАВЛІННЯ КОМАНДНОЮ РОБОТОЮ НА БАЗІ DJANGO

TEAMWORK MANAGEMENT SYSTEM BASED ON DJANGO

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-41
Морозюк Богдан Вікторович
_____
(підпис)
Керівник: д.пед.н., професор
Тулашвілі Юрій Йосипович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2026 р.
Гарант освітньої програми:
д.пед.н., професор
Тулашвілі Юрій Йосипович
_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«16» січня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Морозюк Богдан Вікторович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Система управління командною роботою на базі Django»
Керівник д.пед.н., професор Тулашвілі Юрій Йосипович
затверджені наказом закладу вищої освіти від «*16*» *січня 2026* р. № 32/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«02» червня 2026 р.*

3. Вихідні дані до роботи: *аналітичне дослідження систем управління проектами*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Вступ
Розділ 1. Аналіз предметної області
Розділ 2. Специфікація вимог до розроблюваної системи
Розділ 3. Розробка системи управління
Розділ 4. SEO інформаційно-комп'ютерної системи
Висновки
Додатки

5. Перелік графічного матеріалу:
9 рисунків;
лістинг коду;
презентація.

6. Консультанти розділів роботи

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

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 17.02.2026 р*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 28.02.2025 р.*	
3	*Розробити функціональну схему роботи програмного продукту*	*до 12.03.2026 р*	
4	*Описати засоби розробки об'єкта проектування*	*до 26.03.2026 р.*	
5	*Практична реалізація об'єкта проектування*	*до 30.04.2026 р.*	
6	*Провести спеціальні розрахунки*	*до 18.05.2026 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*до 02.06.2026 р.*	

Здобувач вищої освіти _____ Богдан МОРОЗЮК

Керівник роботи _____ Юрій ТУЛАШВІЛІ

АНОТАЦІЯ

Морозюк Б. В. Система управління командною роботою на базі Django. Рукопис.

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

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

У першому розділі кваліфікаційної роботи здійснюється аналіз сучасного стану проблеми, обґрунтування вибраного напряму та вибір методів рішення, аналіз і вибір засобів проектування. У другому розділі роботи представлено обґрунтування вибору шляхів, технологій вирішення поставленого завдання. Поряд з тим запропоновано функціонально-структурна схема роботи об'єкта проектування та ER-діаграма моделі бази даних. Розробка програмного забезпечення та практична реалізація об'єкта проектування наведені в третьому розділі роботи. У четвертому розділі запропоновано SEO інформаційно-комп'ютерної системи. У висновках узагальнено інформацію, відображену у попередніх частинах.

Ключові слова: Django, web-система управління командною роботою, управління проектами, пошукова оптимізація.

ANNOTATION

Bogdan Morozuk. Teamwork management system based on Django. Manuscript.

Bachelor's qualifying thesis of the OP «Computer Sciences». Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, four chapters, conclusions, a list of used sources and appendices.

The first chapter of this thesis analyzes the current state of the problem, justifies the chosen approach, selects solution methods, and analyzes and selects design tools. The second chapter presents the rationale for the choice of approaches and technologies to solve the task at hand. Additionally, a functional-structural diagram of the design object and an ER diagram of the database model are proposed. The software development and practical implementation of the design object are described in the third chapter. The fourth chapter proposes SEO for the information and computer system. The conclusions summarize the information presented in the preceding sections.

Keywords: Django, web-teamwork management system, project management, search engine optimization.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується не лише стрімким нарощуванням обчислювальних потужностей, а й якісною зміною способів організації колективної праці. Масштабна цифрова трансформація підприємств, прискорена глобальним поширенням дистанційних і гібридних форматів зайнятості, висунула на перший план принципово нові вимоги до інструментарію командної взаємодії. Якщо ще декілька років тому питання координації учасників проекту вирішувалося переважно засобами електронної пошти та корпоративних месенджерів, то сьогодні такий підхід виявляється недостатнім: розпорошеність даних між різними комунікаційними каналами, відсутність єдиного реєстру завдань і нечітке розмежування зон відповідальності системно знижують продуктивність команд і ускладнюють контроль за дотриманням строків.

Проблема загострюється в проектному середовищі, де ієрархічна структура завдань, залежності між ними та динамічний перерозподіл ресурсів потребують централізованої підтримки на рівні спеціалізованого програмного забезпечення. Відсутність такої підтримки не є лише організаційним незручністю – вона має вимірювані наслідки: зростання транзакційних витрат на комунікацію, дублювання робіт, запізнення у виявленні блокуючих залежностей та ерозія командної звітності. Ці чинники визначають попит на веб-орієнтовані платформи управління проектами, здатні забезпечити прозорість процесів, відтворюваність звітності та гнучке налаштування під специфіку конкретної організації.

Особливої актуальності в цьому контексті набуває застосування фреймворку Django як технологічної основи для реалізації подібних систем. Django є зрілою, активно підтримуваною платформою з розвиненою екосистемою компонентів, вбудованими механізмами автентифікації, гранульованого розмежування прав доступу та підтримкою як шаблонних, так і RESTful підходів до побудови інтерфейсів. Поєднання можливостей швидкого

прототипування з потенціалом промислового масштабування робить Django обґрунтованим вибором для розробки платформ командної взаємодії в сегменті малого та середнього бізнесу, освітніх установ і стартап-проектів – середовищах, де ресурси розробки обмежені, а вимоги до функціональності залишаються високими. Розробка спеціалізованого рішення на цій платформі безпосередньо сприяє підвищенню операційної прозорості, якості внутрішньої звітності та зрілості процесів управління виконанням завдань у проектних командах.

Метою кваліфікаційної роботи бакалавра є проектування та розробка веб-орієнтованої системи управління командною роботою на базі фреймворку Django, що забезпечує ефективну координацію завдань, моніторинг прогресу виконання, розмежування ролей учасників та підвищення загальної продуктивності проектних команд.

Об’єктом дослідження є процес автоматизації управління командною роботою в сучасних програмних проектах засобами веб-технологій.

Предметом дослідження є методи, моделі та інструментальні засоби проектування і реалізації системи управління командною роботою на базі фреймворку Django.

Завдання дослідження:

– провести аналіз існуючих рішень у сфері управління командною роботою та проектами, визначити їхні переваги, обмеження та функціональні можливості з метою формування вимог до розроблюваної системи;

– спроектувати архітектуру системи управління командною роботою з урахуванням принципів модульності, масштабованості та розмежування відповідальності між компонентами, включаючи розробку моделі даних;

– реалізувати основні функціональні модулі системи засобами Django: управління користувачами та ролями, створення і відстеження завдань, формування звітності, забезпечення авторизованого доступу до ресурсів;

– розробити інтерфейс користувача системи з використанням сучасних веб-технологій, що забезпечує зручну та інтуїтивно зрозумілу взаємодію з функціоналом платформи для учасників команди різних рівнів доступу;

– провести тестування розробленої системи, оцінити її функціональну коректність, продуктивність та зручність використання, а також визначити перспективи подальшого розвитку і вдосконалення платформи.

Новизна роботи полягає у комплексному підході до проектування та реалізації системи управління командною роботою, яка поєднує рольову модель доступу з гнучкою системою делегування завдань, інтегрованим механізмом сповіщень та аналітичним модулем відстеження прогресу в межах єдиного Django-застосунку.

Практична цінність роботи визначається тим, що розроблена система може бути впроваджена в діяльність проектних команд підприємств, освітніх установ та громадських організацій для оптимізації процесів планування, розподілу та контролю виконання завдань. Готовий програмний продукт є самодостатнім веб-застосунком з відкритим вихідним кодом, що допускає подальше налаштування відповідно до потреб конкретної організації.

Апробація результатів дослідження:

– 4 Міжнародна науково-практична конференція «Advanced Technologies in Scientific Research» (13-15 травня 2026 р.), Роттердам, Нідерланди. International Science Group. 2026 (додаток А).

– 8 Міжнародна науково-практична конференція «Modern Perspectives on Global Scientific Solutions» (18-20 травня, 2026 р.), Берген, Норвегія. European Open Science Space. 2025 (додаток А).

– Тулашвілі, Ю., Кошелюк, В., Морозюк, Б. Проектування безпечної системи управління командною роботою з багаторівневою автентифікацією на базі Django. International Science Journal of Engineering and Agriculture. ISJEA, 2026. Pp. 30-47. (додаток Б).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану проблеми

Ефективна організація командної роботи є одним із ключових чинників успішної реалізації програмних та бізнес-проектів. Із зростанням масштабів команд і переходом до розподіленого формату взаємодії виникають системні проблеми: нечітке визначення відповідальності між учасниками, відсутність прозорого контролю строків виконання завдань, дублювання функцій, нераціональне використання ресурсів, а також складності у відстеженні загального прогресу проекту в реальному часі.

За даними аналітичних досліджень у галузі управління проектами, понад 0,6 командних проектів стикаються із затримками, пов’язаними з неефективною комунікацією та відсутністю структурованого управління завданнями [1]. Відсутність єдиного цифрового середовища для координації дій команди змушує організації вдаватися до поєднання розрізнених інструментів – електронної пошти, месенджерів та табличних документів, – що знижує продуктивність і підвищує ризики втрати важливих даних. Таким чином, потреба у спеціалізованих системах управління командною роботою залишається актуальною і невирішеною для значної частини команд [2].

На сучасному ринку програмного забезпечення представлено широкий спектр інструментів для управління проектами та командною взаємодією. Найбільш поширеними серед них є Jira, Trello, Asana, Notion та Taiga. Кожна з цих систем має власну філософію побудови робочих процесів і орієнтована на певні категорії користувачів [3].

Jira є потужним корпоративним рішенням компанії Atlassian, яке підтримує методології Scrum і Kanban, забезпечує детальне налаштування workflow та надає розвинені інструменти звітності. Проте складність конфігурації та висока вартість ліцензування роблять Jira малодоступною для малих команд і навчальних проектів. Trello відзначається простотою у

використанні завдяки канбан-орієнтованому інтерфейсу, однак суттєво обмежений у функціональності: відсутня детальна звітність, а рольова модель є спрощеною. Asana пропонує баланс між функціональністю і зручністю, але її преміум-тарифи обмежують доступ до аналітики та автоматизацій.

Taiga є відкритим рішенням для управління agile-проектами, проте потребує значних ресурсів для самостійного розгортання та налаштування. Notion позиціонується як інструмент для ведення документації і баз знань із базовими можливостями відстеження завдань, що не дозволяє розглядати його як повноцінну систему управління проектами. Жодне з розглянутих рішень не поєднує одночасно відкритий вихідний код, низький поріг входження, розвинену рольову модель та вбудовані засоби аналітики в межах єдиної легковагової платформи. В таблиці 1.1 відображено порівняльний аналіз систем управління командною роботою.

Таблиця 1.1 – Порівняльний аналіз систем управління

Система	Управління завданнями	Ролі та права	Звітність	Open Source
Jira	Так	Так	Так	Ні
Trello	Частково	Обмежено	Ні	Ні
Asana	Так	Так	Частково	Ні
Taiga	Так	Так	Частково	Так
Розроблювана система	Так	Так	Так	Так

Django є високорівневим веб-фреймворком мови Python, що дотримується принципу «не повторюйся» (DRY) і архітектурного патерну MTV (Model – Template – View). Фреймворк надає вбудовані засоби автентифікації та авторизації, ORM для роботи з базами даних, адміністративну панель, механізм міграцій схеми, систему маршрутизації запитів та шаблонний рушій. Ці можливості дозволяють значно скоротити час розробки і зосередитися на логіці застосунку, а не на інфраструктурному коді [4].

Серед ключових переваг Django для розробки системи управління командною роботою слід виділити: зрілу екосистему пакетів (Django REST Framework, Celery, Channels), вбудований захист від типових вразливостей (CSRF, XSS, SQL-ін’єкції), гнучку систему дозволів на рівні моделей та об’єктів, а також широку документаційну базу і активну спільноту розробників [5]. Django активно застосовується у виробничих середовищах таких компаній, як Instagram, Disqus та Mozilla, що підтверджує його придатність для розробки масштабованих веб-застосунків.

На підставі проведеного аналізу встановлено, що жодне з наявних рішень не задовольняє в повній мірі потреби малих і середніх команд, яким потрібна гнучка, безкоштовна та легко розгортувана система управління завданнями з відкритим кодом. Розробка власної системи на базі Django дозволяє усунути виявлені обмеження: забезпечити повний контроль над функціональністю і даними, реалізувати рольову модель доступу відповідно до специфіки командної структури, а також інтегрувати аналітичний модуль і систему сповіщень без залежності від зовнішніх сервісів.

Обраний підхід передбачає використання монолітної архітектури Django з поділом на функціональні застосунки (apps), що забезпечує чіткість структури коду та спрощує подальше масштабування. Для збереження даних обрано реляційну СУБД PostgreSQL, яка добре інтегрується з Django ORM і забезпечує надійність зберігання при зростанні обсягів проектних даних. Клієнтська частина реалізується з використанням Django-шаблонів у поєднанні з Bootstrap, що забезпечує адаптивний і зручний інтерфейс без залучення окремого фронтенд-фреймворку.

Виконуючи аналіз сучасного стану проблеми було здійснено порівняльний огляд провідних систем управління проектами – Jira, Trello, Asana, Taiga, Notion – та виявлено їхні функціональні обмеження стосовно потреб малих і середніх команд. Обґрунтовано доцільність розробки власної системи управління командною роботою на базі фреймворку Django та запропоновано технологічний стек і архітектурний підхід до проектування ПЗ.

1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напряму та вибір методів рішення

Вибір програмного рішення для автоматизації командної роботи потребує системного підходу до оцінювання наявних альтернатив. Щоб забезпечити об'єктивність порівняння, необхідно визначити критерії, що відповідають потребам цільової аудиторії – малих і середніх проектних команд, освітніх установ та стартапів. До таких критеріїв належать: наявність відкритого вихідного коду, рівень розвиненості рольової моделі доступу, наявність вбудованої аналітики та звітності, можливість самостійного розгортання без залежності від хмарних провайдерів, поріг входження для нетехнічних користувачів, вартість використання та ступінь нативної інтеграції з фреймворком Django [6].

Перелічені критерії дозволяють всебічно оцінити кожну систему не лише з технічної, але й з організаційної та економічної точок зору. Аналіз за цими параметрами наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз програмних аналогів системи управління командною роботою

Критерій	Jira	Trello	Asana	Taiga	Розроблювана система
Відкритий код	Ні	Ні	Ні	Так	Так
Рольова модель	Розширена	Базова	Середня	Середня	Розширена
Аналітика/звіти	Так	Ні	Частково	Частково	Так
Самостійне розгортання	Так (Server)	Ні	Ні	Так	Так
Поріг входження	Високий	Низький	Середній	Середній	Низький
Вартість	Платна	Freemium	Freemium	Безкоштовна	Безкоштовна
Інтеграція Django	API	API	API	Часткова	Нативна

Jira (Atlassian) є лідером корпоративного ринку систем управління проектами. Платформа підтримує agile-методології Scrum і Kanban, надає розширені можливості кастомізації workflow, потужні інструменти звітності (burndown-діаграми, velocity-чарти) та інтеграцію з широким спектром сторонніх сервісів [7]. Однак складність первинного налаштування і висока вартість ліцензування – від 8,15 USD на користувача на місяць – роблять Jira недоступною для навчальних і некомерційних проектів. Крім того, перехід на хмарну модель Jira Cloud обмежує можливість самостійного розгортання, що є критичним для організацій із вимогами до локального зберігання даних. На рисунку 1.1 продемонстровано типовий dashboard Jira (Atlassian).

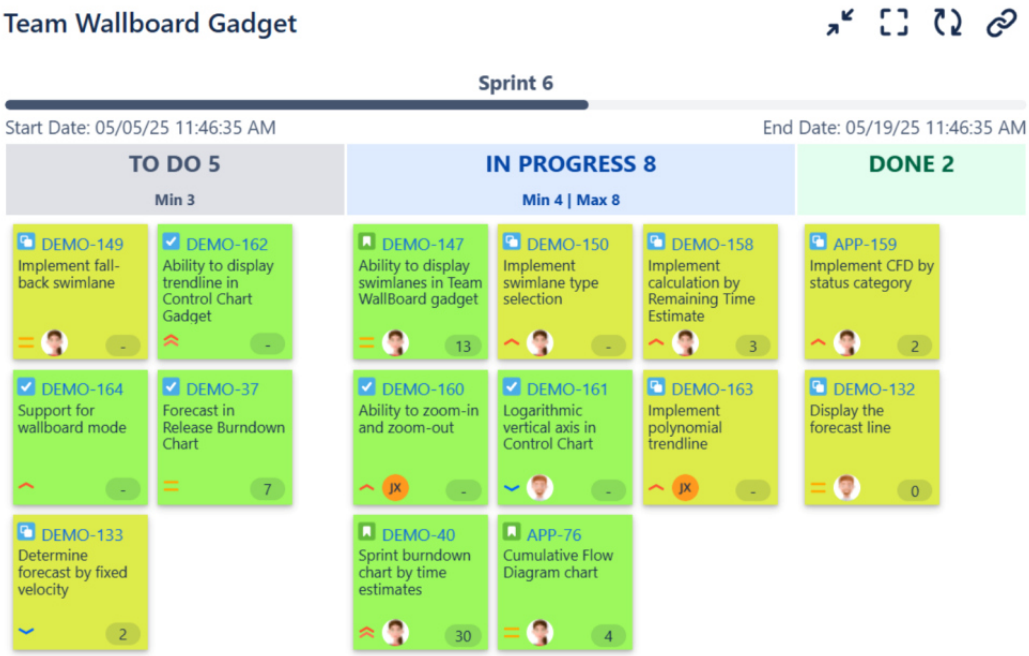

Рисунок 1.1 – Типова панель Jira (Atlassian) [7]

Trello (Atlassian) пропонує інтуїтивний канбан-орієнтований інтерфейс, що забезпечує низький поріг входження та швидке засвоєння навіть без технічних навичок. Система добре підходить для управління невеликими завданнями, однак суттєво поступається конкурентам у функціональності: відсутня детальна звітність, обмежені можливості рольового доступу, а розширені функції доступні лише в платних тарифах. Trello не підтримує повноцінного управління

ресурсами команди і не може слугувати єдиним інструментом для зрілих проектних процесів [8]. На рисунку 1.2 відображено типовий dashboard Trello.

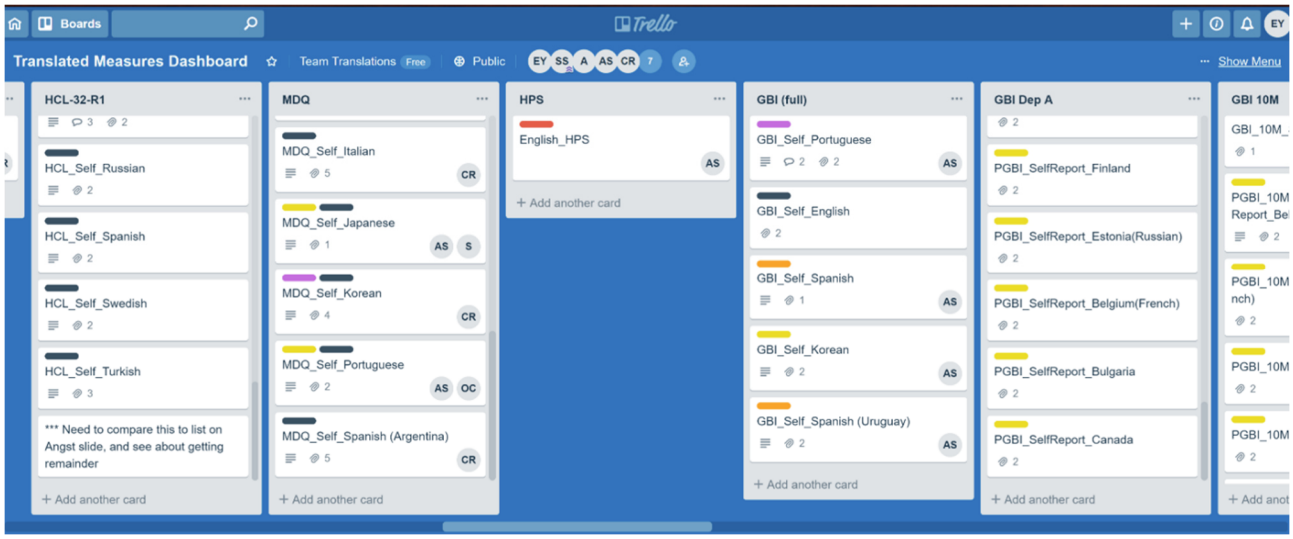

Рисунок 1.2 – Типова панель Trello [8]

Asana є збалансованим рішенням між простотою Trello та потужністю Jira. Платформа підтримує різні представлення проекту (список, дошка, часова шкала, календар), надає базові інструменти звітності та функції автоматизації задач [9]. Проте більшість аналітичних можливостей і функцій автоматизації доступні лише у преміум-плані (від 10,99 USD на місяць), а самостійне розгортання системи не передбачено. Asana не підтримує інтеграцію на рівні вихідного коду і не може бути кастомізована під специфічні бізнес-процеси без використання API [10].

Taiga є єдиним повноцінним відкритим рішенням серед розглянутих аналогів, що підтримує методології Scrum і Kanban, надає базову звітність та інструменти управління бэклогом [11]. Однак розгортання Taiga потребує значних DevOps-компетенцій і серверних ресурсів, інтерфейс платформи є ускладненим для нових користувачів, а можливості налаштування рольової моделі та інтеграції з іншими системами обмежені порівняно з комерційними аналогами. Taiga не забезпечує нативної взаємодії з Django і не може бути розширена без глибокого втручання у вихідний код [12].

Таким чином, жоден з аналізованих продуктів не поєднує одночасно відкритий вихідний код, розширену рольову модель, вбудовану аналітику, нативну інтеграцію з Django та низький поріг входження. Це підтверджує доцільність розробки власної системи, що усуває виявлені прогалини.

Вибір СУБД є визначальним для продуктивності, надійності та масштабованості системи. Для Django-застосунку, що активно використовує ORM, особливого значення набуває повнота підтримки всіх можливостей фреймворку: складні запити, транзакції, підтримка JSON-полів та повнотекстовий пошук [13]. У таблиці 1.3 наведено порівняння трьох найпоширеніших СУБД у контексті Django-розробки.

Таблиця 1.3 – Порівняння СУБД за критеріями придатності для Django-проекту

Критерій	PostgreSQL	MySQL	SQLite
Підтримка Django ORM	Повна	Повна	Повна
Транзакції (ACID)	Так	Так	Обмежено
JSON-поля	Так	Так (5.7+)	Ні
Масштабованість	Висока	Середня	Низька
Підходить для prod	Так	Так	Ні

PostgreSQL є рекомендованою СУБД для виробничих Django-застосунків завдяки повній підтримці ORM, розширеним типам даних (включаючи JSONB, ArrayField, HStoreField), потужному механізму транзакцій стандарту ACID та можливостям горизонтального масштабування [14]. MySQL є прийнятною альтернативою, проте поступається PostgreSQL за підтримкою специфічних типів Django та гнучкістю запитів. SQLite доцільно використовувати виключно на етапі локальної розробки і тестування через суттєві обмеження паралельного доступу та транзакційної моделі [15].

На підставі проведеного аналізу для розроблюваної системи обрано PostgreSQL як СУБД, що забезпечує оптимальне поєднання продуктивності, функціональності та сумісності з Django ORM.

На основі результатів порівняльного аналізу визначено, що оптимальним напрямом є розробка власної веб-орієнтованої системи управління командною роботою на базі Django, яка поєднає переваги наявних рішень і усуне їхні ключові недоліки. Обраний напрям забезпечує: повний контроль над архітектурою та даними системи; можливість нативного розширення функціональності засобами Python та Django; відсутність ліцензійних обмежень і залежності від хмарних провайдерів; адаптацію рольової моделі та інтерфейсу до потреб конкретної організації.

Монолітна архітектура Django з поділом на функціональні застосунки (apps) обрана як основний структурний підхід. Вона забезпечує низький поріг входження для розробників, просте розгортання на стандартному Linux-сервері та можливість поступового переходу до мікросервісної архітектури в майбутньому. Такий підхід є оптимальним для проектів із невеликими і середніми командами, де накладні витрати на оркестрацію мікросервісів є невиправданими.

Для реалізації системи управління командною роботою обрано комплекс методів і технологій, що відповідають сучасним стандартам веб-розробки. В основу покладено архітектурний патерн MTV (Model – Template – View), реалізований засобами Django. Модель відповідає за структуру і логіку даних, шаблон – за формування HTML-відповіді, а представлення – за обробку HTTP-запитів і координацію між моделлю та шаблоном.

Управління правами доступу буде реалізовано через вбудований механізм Django Permission у поєднанні з декораторами @login_required та @permission_required, що забезпечує гнучке розмежування ролей: адміністратор, керівник проекту та рядовий учасник. Для збереження цілісності даних при одночасній роботі кількох користувачів застосовано транзакційний підхід із використанням atomic()-блоків Django ORM.

1.3 Аналіз і вибір засобів проектування

Якість розробленої програмної системи значною мірою визначається обґрунтованістю вибору інструментів проектування – як для реалізації бекенду та фронтенду, так і для моделювання архітектури, прототипування інтерфейсу та організації процесу розробки. При виборі засобів враховувалися такі вимоги: відповідність технологічному стеку Django, наявність активної спільноти та актуальної документації, можливість безкоштовного або відкритого використання, низький поріг входження та підтримка стандартів галузі. Зведений перелік обраних інструментів із коротким обґрунтуванням наведено у таблиці 1.3.

Таблиця 1.3 – Засоби проектування та розробки системи управління командною роботою

Категорія	Інструмент	Призначення	Обґрунтування
Мова програмування	Python 3.11	Бекенд-логіка системи	Широка екосистема, підтримка Django
Веб-фреймворк	Django 4.2 LTS	MTV-архітектура, ORM, Auth	Стабільна LTS-версія, вбудований admin
СУБД	PostgreSQL 15	Зберігання даних	ACID, JSONB, масштабованість
Фронтенд	Bootstrap 5 + HTMX	Адаптивний UI, динаміка	Без SPA-фреймворку, серверний рендеринг
Діаграми UML	draw.io (diagrams.net)	Use Case, ER, Class-діаграми	Безкоштовний, експорт SVG/PNG
Прототипування UI	Figma	Макети екранів	Командна робота, веб-доступ
Контроль версій	Git + GitHub	Версіювання коду	Стандарт галузі, CI/CD-інтеграція
Середовище розробки	VS Code + розширення	Редагування, дебагінг	Легковагове, безкоштовне
Розгортання	Gunicorn + Nginx	Production-сервер	Стандартний стек Django prod
Тестування	Django TestCase + pytest	Unit та інтеграційні тести	Вбудований у Django, розширюваний

Серед засобів реалізації бекенду було розглянуто:

– Python 3.11 обрано як основну мову програмування завдяки її читабельному синтаксису, розвиненій стандартній бібліотеці та широкій підтримці в науковому й підприємницькому середовищі. Версія 3.11 забезпечує суттєве покращення продуктивності порівняно з попередніми релізами (до 0,6 приросту швидкості виконання за даними офіційного бенчмарку CPython) і є поточною стабільною гілкою з довгостроковою підтримкою [8];

– Django 4.2 LTS є поточною версією з довгостроковою підтримкою (до квітня 2026 року), що гарантує стабільність і своєчасне отримання патчів безпеки протягом усього терміну використання системи. Ця версія включає вдосконалений ORM із підтримкою складних анотацій, поліпшену систему міграцій та оновлений адміністративний інтерфейс [9]. Вбудовані механізми автентифікації, системи дозволів і захисту від CSRF/XSS усувають потребу у підключенні сторонніх бібліотек для реалізації базової безпеки застосунку;

– PostgreSQL 15 обрано як СУБД для виробничого середовища. На відміну від MySQL, PostgreSQL повністю підтримує всі типи Django ORM, включаючи JSONBField, ArrayField та HStoreField. Механізм транзакцій ACID, підтримка повнотекстового пошуку, часткові індекси та розвинена система прав доступу на рівні рядків роблять PostgreSQL оптимальним вибором для реалізації складних бізнес-сценаріїв системи управління завданнями.

До основних засобів реалізації клієнтської частини відносять:

– Bootstrap 5 використовується як основний CSS-фреймворк для побудови адаптивного інтерфейсу. Він надає готову сітку, типографіку, компоненти форм і елементи навігації, що скорочує обсяг власного CSS-коду і забезпечує крос-браузерну сумісність [10]. Bootstrap 5 відмовився від залежності від jQuery, що спрощує інтеграцію з іншими бібліотеками та зменшує розмір клієнтських ресурсів;

– HTMX застосовано для реалізації динамічних елементів інтерфейсу (часткове оновлення сторінок, асинхронні форми, сповіщення) без написання власного JavaScript-коду. Бібліотека дозволяє виконувати AJAX-запити через

декларативні HTML-атрибути і повністю інтегрується з Django-шаблонами та механізмом часткових відповідей. Такий підхід зберігає переваги серверного рендерингу та SEO-оптимізації, характерні для Django, при цьому забезпечуючи сучасний рівень інтерактивності.

Серед засобів моделювання та проектування архітектури ми звернули увагу на:

– draw.io (diagrams.net) обрано як основний інструмент для побудови UML-діаграм: діаграм варіантів використання (Use Case), діаграм класів, ER-діаграм бази даних та діаграм послідовності. Перевагами draw.io є повна безкоштовність, веб-доступність без встановлення, підтримка експорту у формати SVG, PNG та XML, а також можливість збереження файлів безпосередньо у репозиторій GitHub. Функціональність інструменту повністю покриває потреби проектування для системи рівня кваліфікаційної роботи бакалавра;

– Figma використана для прототипування користувацького інтерфейсу на етапі проектування екранів системи. Хмарна природа інструменту забезпечує доступ до макетів з будь-якого пристрою, а функція коментування дозволяє фіксувати проектні рішення безпосередньо на прототипі. Figma підтримує компонентний підхід до побудови макетів, що відповідає модульній структурі Bootstrap 5 і спрощує перенесення дизайну у шаблони Django.

Ряд розробників акцентують увагу на основних засоби організації розробки та розгортання:

– Git і GitHub забезпечують контроль версій та організацію командної розробки. Стратегія feature-branch workflow дозволяє ізолювати роботу над окремими модулями системи, а механізм pull request – здійснювати рецензування коду перед злиттям у головну гілку. GitHub Actions використовується для автоматизації запуску тестів при кожному push-події, що забезпечує безперервну інтеграцію на рівні базових CI-практик;

– Visual Studio Code обрано як основне середовище розробки завдяки його легковагій архітектурі, широкому вибору розширень для Python і Django

(Pylance, Django, GitLens) та вбудованому відлагоджувачу. Інтеграція з Git-репозиторієм і підтримка Docker Dev Containers спрощують налаштування однорідного середовища розробки;

– Gunicorn і Nginx формують стандартний виробничий стек для Django-застосунків. Gunicorn виступає WSGI-сервером, що обробляє Python-запити, а Nginx відіграє роль реверс-проксі та відповідає за роздачу статичних файлів. Така конфігурація є галузевим стандартом і детально документована в офіційній документації Django, що спрощує розгортання і подальше адміністрування системи.

1.4 Постановка завдання на кваліфікаційну роботу бакалавра

Спираючись на результати аналізу предметної області, огляду існуючих аналогів систем управління проектами та обраного технологічного стеку, метою кваліфікаційної роботи бакалавра є проектування та програмна реалізація веб-застосунку системи управління командною роботою на базі фреймворку Django, що забезпечує централізовану координацію проектної діяльності, розмежування ролей і прав доступу учасників, відстеження стану та строків виконання завдань, підтримку внутрішньої комунікації в команді та формування аналітичної звітності про хід реалізації проекту.

Для досягнення поставленої мети необхідно розв’язати такі основні завдання:

– провести аналіз існуючих рішень у сфері управління командною роботою та проектами, визначити їхні переваги, обмеження та функціональні можливості з метою формування вимог до розроблюваної системи;

– спроектувати архітектуру системи управління командною роботою з урахуванням принципів модульності, масштабованості та розмежування відповідальності між компонентами, включаючи розробку моделі даних;

– реалізувати основні функціональні модулі системи засобами Django: управління користувачами та ролями, створення і відстеження завдань, формування звітності, забезпечення авторизованого доступу до ресурсів;

– розробити інтерфейс користувача системи з використанням сучасних веб-технологій, що забезпечує зручну та інтуїтивно зрозумілу взаємодію з функціоналом платформи для учасників команди різних рівнів доступу;

– провести тестування розробленої системи, оцінити її функціональну коректність, продуктивність та зручність використання, а також визначити перспективи подальшого розвитку і вдосконалення платформи.

Очікуваним результатом виконання кваліфікаційної роботи є функціонально завершений веб-застосунок системи управління командною роботою, реалізований на базі фреймворку Django та готовий до локального розгортання і подальшого промислового впровадження. Система має забезпечувати повний цикл управління проектною діяльністю: від реєстрації користувачів і формування команд до постановки завдань, відстеження їх виконання та генерації аналітичної звітності. Програмний продукт включає функціонально завершений серверний компонент із реалізованою бізнес-логікою, реляційну базу даних з набором міграцій, адміністративну панель, адаптивний призначений для користувача інтерфейс з рольовим розмежуванням доступу, документацію REST API та інструкцію з розгортання системи в локальному та хмарному середовищах.

Результати роботи мають як прикладне, так і методичне значення. З прикладної точки зору, розроблена система являє собою готове рішення для команд малого та середнього розміру, яке усуває потребу у використанні розрізнених інструментів комунікації та планування. З методичної – робота демонструє підхід до проектування повноцінних веб-застосунків на Django, що може слугувати практичним орієнтиром для подальших досліджень у галузі автоматизації організаційних процесів та розробки інтелектуальних систем підтримки прийняття рішень у командному середовищі.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

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

Поставлене завдання розробка системи управління командною роботою може бути вирішене трьома принципово різними архітектурними шляхами: монолітним Django-застосунком, мікросервісною архітектурою або гібридним підходом з Django REST Framework як бекендом і окремим SPA-фреймворком (React/Vue) на фронтенді. Порівняльний аналіз цих підходів за ключовими параметрами наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння архітектурних підходів до реалізації системи

Критерій	Монолітна Django-архітектура	Мікросервіси	Django + DRF + SPA
Складність розгортання	Низька	Висока	Середня
Час розробки MVP	Короткий	Тривалий	Середній
Вимоги до команди	1-2 розробники	5+ розробників	2-4 розробники
Масштабованість	Середня	Висока	Висока
Зв'язність компонентів	Висока (внутрішня)	Низька (API)	Низька (API)

За результатами аналізу обрано монолітну архітектуру Django з поділом на функціональні застосунки (apps). Цей підхід є оптимальним для проекту рівня кваліфікаційної роботи бакалавра: він забезпечує мінімальний час розробки, просте розгортання на одному сервері, вичерпну документацію та максимальне використання вбудованих можливостей Django. При цьому внутрішній поділ на ізольовані застосунки (users, tasks, projects, notifications) зберігає принцип розмежування відповідальності і спрощує майбутній рефакторинг або перехід до мікросервісів.

Однією з ключових функціональних вимог до системи є розмежування доступу між учасниками команди з різними ролями: адміністратор, керівник

проекту та виконавець. Для реалізації цієї вимоги розглянуто три моделі контролю доступу: DAC (Discretionary Access Control), RBAC (Role-Based Access Control) та ABAC (Attribute-Based Access Control). Порівняння наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння моделей контролю доступу

Критерій	DAC (Discretionary)	RBAC (Role-Based)	ABAC (Attribute-Based)
Складність реалізації	Низька	Середня	Висока
Гнучкість	Низька	Середня	Висока
Підтримка Django	Часткова	Вбудована	Через сторонні пакети
Аудит прав	Складний	Простий	Середній
Обрано для системи	Ні	Так	Ні

Для системи управління командною роботою обрано модель RBAC як таку, що оптимально відповідає структурі команди, нативно підтримується вбудованою системою дозволів Django і не потребує сторонніх пакетів. Алгоритм роботи механізму доступу передбачає такі кроки: автентифікація користувача через сесійний механізм Django; визначення його ролі в межах конкретного проекту через зв’язок UserProfile – ProjectMembership; перевірка наявності необхідного дозволу через декоратор @permission_required або метод user.has_perm(); надання або відмова у доступі до ресурсу з відповідним HTTP-кодом відповіді. Такий алгоритм забезпечує чіткість аудиту прав і зрозумілу логіку призначення ролей адміністратором.

Центральним функціональним модулем системи є управління завданнями (tasks). Для його реалізації обрано підхід на основі скінченного автомату (Finite State Machine, FSM), що описує життєвий цикл завдання через чітко визначені стани та дозволені переходи між ними. Визначено п’ять станів завдання: «Нове» → «В роботі» → «На перевірці» → «Виконано» / «Відхилено». Переходи між станами доступні лише певним ролям: виконавець переводить завдання з «Нового» у «В роботі» та із «В роботі» на «На перевірці»; керівник проекту

підтверджує або відхиляє завдання з «На перевірці». Такий підхід унеможливлює некоректні зміни статусу і формує прозору історію виконання.

Для зберігання стану та маршрутизації переходів використовується поле з вибором (choices) у моделі Task Django ORM, а логіка валідації переходів реалізована на рівні методів моделі та серіалізаторів форм. Це забезпечує єдине місце перевірки бізнес-правил без дублювання коду у представленнях.

Для забезпечення оперативного інформування учасників команди про зміни статусів завдань, нові призначення та коментарі реалізовано модуль сповіщень. Технічно система сповіщень базується на сигналах Django (django.db.models.signals), що дозволяє декларативно підписуватися на події зміни моделей без прямої залежності між модулями.

При зміні статусу завдання або його призначенні відповідний сигнал post_save автоматично ініціює створення запису сповіщення у базі даних і, за потреби, надсилання електронного листа через SMTP. Для уникнення блокування основного потоку виконання при надсиланні листів передбачено інтеграцію з Celery – асинхронним менеджером черги завдань на базі Redis-брокера. Такий підхід забезпечує відокремленість логіки сповіщень від основного workflow системи і дозволяє масштабувати обробку черги незалежно від веб-сервера.

Для зручної роботи з великими обсягами завдань і учасників у системі реалізовано функціонал пошуку та багатокритеріальної фільтрації. Технічно це вирішено засобами Django ORM через побудову ланцюжкових Q-об’єктів, що дозволяють комбінувати умови фільтрації з логікою AND/OR без виконання зайвих запитів до бази даних. Для текстового пошуку за полями назви та опису завдання задіяно векторний пошук PostgreSQL (SearchVector + SearchQuery), що забезпечує морфологічно коректні результати порівняно з простим LIKE-запитом.

Фільтрація за статусом, виконавцем, проектом і терміном здійснюється через бібліотеку django-filter, яка інтегрується з Django-формами і генерує

оптимізовані SQL-запити. Усі фільтровані запити підтримують пагінацію на основі Django Paginator, що обмежує обсяг даних у відповіді і знижує навантаження на СУБД при зростанні кількості записів.

2.2 Функціонально-структурна схема роботи об’єкта проектування

Функціонально-структурна схема системи управління командною роботою відображає взаємодію між основними підсистемами та потоки даних між ними. Система складається з п’яти функціональних модулів: управління користувачами та автентифікацією (users), управління проектами (projects), управління завданнями (tasks), підсистема сповіщень (notifications) і модуль аналітики та звітності (reports). Кожен модуль реалізований як окремий Django-застосунок (app) із власними моделями, представленнями, формами та шаблонами, що забезпечує слабку зв’язаність компонентів і незалежне тестування кожного з них.

Центральним елементом архітектури є модуль завдань, з яким взаємодіють усі інші компоненти: модуль проектів надає контекст (до якого проекту належить завдання), модуль користувачів визначає виконавців та ролі, а підсистема сповіщень реагує на зміну стану завдань через сигнали Django. Модуль аналітики агрегує дані з усіх модулів для формування звітів і візуалізації прогресу.

В блок-схемі алгоритму обробки запиту на зміну статусу завдання на першому рівні виконується автентифікація користувача через сесійний механізм Django. У разі відсутності валідної сесії алгоритм негайно повертає HTTP 403 і завершує обробку запиту. На другому рівні перевіряється роль користувача у проекті за моделлю RBAC: наявність прив’язки в таблиці ProjectMembership з відповідним значенням поля role визначає, чи має користувач право на виконання цієї дії. Третій рівень – валідація переходу стану засобами скінченного автомату (FSM): перевіряється, чи є запитуваний перехід допустимим відповідно до матриці переходів. Тільки після проходження всіх

трьох перевірок система зберігає оновлений стан завдання і ініціює сигнал сповіщення.

На рисунку 2.1 наведено блок-схему алгоритму обробки запиту на зміну статусу завдання. Алгоритм реалізує послідовну перевірку трьох рівнів захисту перед збереженням будь-яких змін у базі даних.

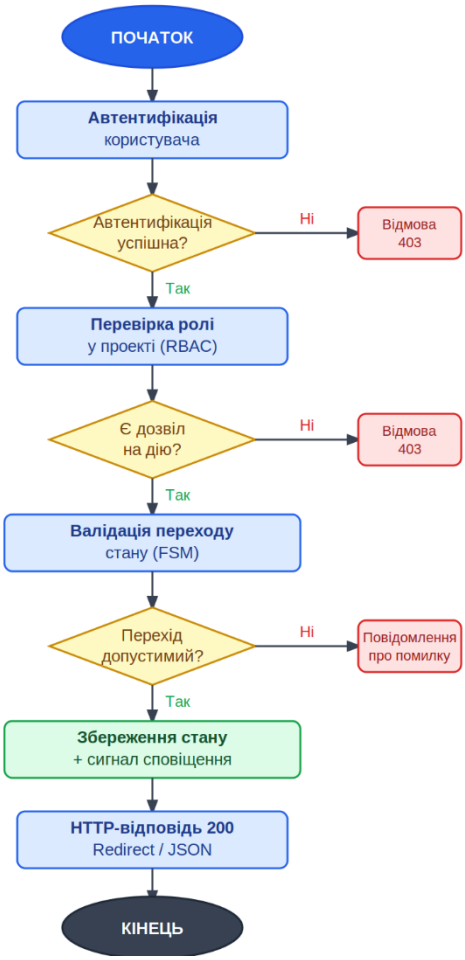

Рисунок 2.1 – Блок-схема алгоритму обробки запиту

Діаграма класів описує статичну структуру моделей даних системи та зв'язки між ними на рівні Django ORM. Центральними сутностями є п'ять класів-моделей: User, UserProfile, Project, ProjectMembership, Task і Notification.

User є стандартною моделлю Django (django.contrib.auth.models.User), розширеною через клас UserProfile за допомогою зв'язку OneToOneField. UserProfile зберігає додаткові атрибути профілю (аватар, біографія, налаштування сповіщень) і не дублює поля базової моделі автентифікації. Такий

підхід дозволяє зберегти сумісність із вбудованими механізмами Django Auth і водночас розширювати профіль без модифікації системних таблиць.

Project є агрегуючою сутністю, що об'єднує завдання та учасників. Зв'язок між проектом і користувачами здійснюється через проміжну модель ProjectMembership, яка зберігає роль учасника (admin, manager, developer) і методи перевірки привілеїв (is_manager(), can_edit_task()). Такий підхід реалізує модель RBAC на рівні бізнес-логіки без залежності від груп Django.

Task є центральною операційною моделлю. Поле status реалізує скінченний автомат через CharField із переліком допустимих значень і методом transition_status(), що валідує перехід перед збереженням. Зовнішні ключі project і assignee забезпечують належність завдання проекту та призначення виконавцю. Модель Notification пов'язана з Task через Foreign Key і формується автоматично через сигнали post_save при зміні стану. На рисунку 2.2 відображено UML-діаграма класів системи управління командною роботою.

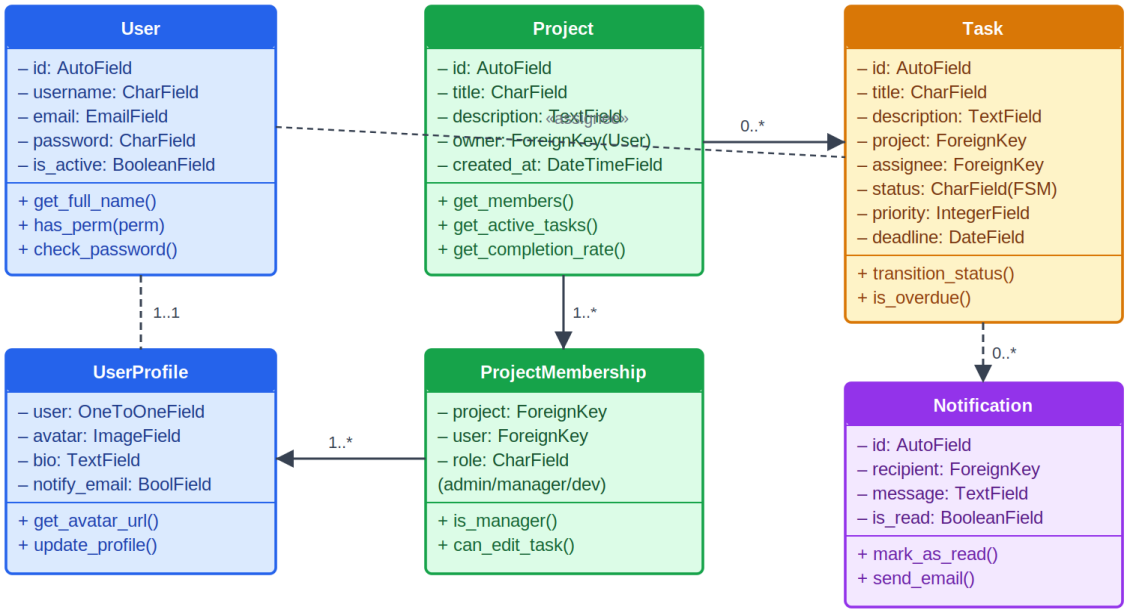

Рисунок 2.2 – UML-діаграма класів системи управління командною роботою

Діаграма послідовності, що продемонстрована на рисунку 2.3, відображає хронологічну взаємодію між учасниками системи при виконанні ключового сценарію – зміни статусу завдання користувачем. У сценарії задіяні п'ять

учасників: Користувач, Браузер (HTMX), Django View (TaskUpdateView), підсистема Auth/Permission та PostgreSQL з Celery.

Взаємодія розпочинається з кліку користувача на кнопку зміни статусу в інтерфейсі. HTMX перехоплює подію і надсилає асинхронний PATCH-запит до ендпоінту /tasks/{id}/status/ без повного перезавантаження сторінки. Django View приймає запит і делегує перевірку прав підсистемі Auth/Permission (крок 3), яка повертає підтвердження або відмову (крок 4).

У разі успішної авторизації виконується блок alt: якщо перехід стану є допустимим відповідно до FSM, View надсилає UPDATE-запит до PostgreSQL (крок 5a) і отримує підтвердження збереження (крок 5b). У разі недопустимого переходу (гілка «заборонений») клієнту повертається HTTP 400 з описом помилки валідації (крок 5c). Після успішного збереження View публікує подію post_save, яку Celery-worker отримує з Redis-черги і асинхронно створює запис сповіщення у базі даних (кроки 6-7). Цей крок не блокує основний потік HTTP-запиту. Завершує сценарій повернення часткового HTML-фрагменту (крок 7), який HTMX підставляє у DOM без перезавантаження сторінки (крок 8).

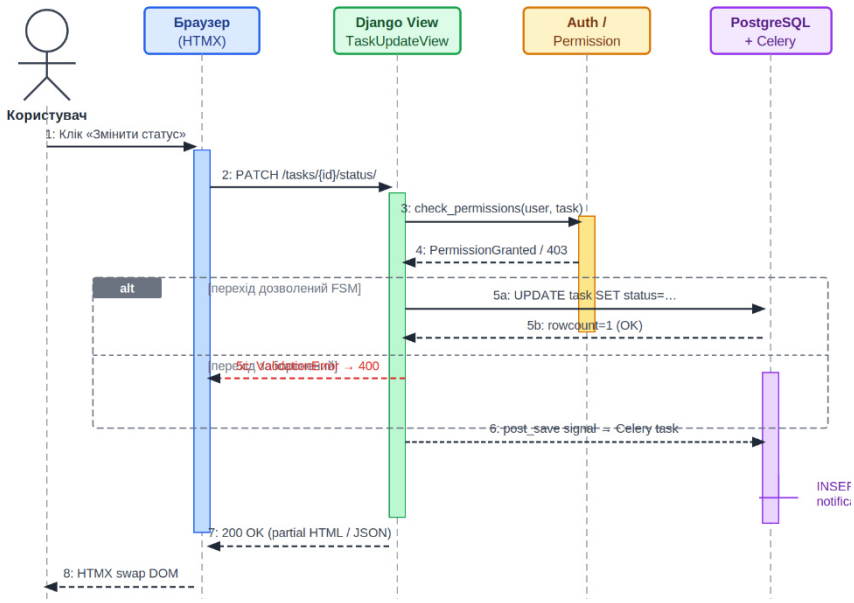

Рисунок 2.3 – Діаграма послідовності: зміна статусу завдання користувачем

Блок-схема алгоритму обробки завдання наочно відображає багаторівневу модель захисту, що складається з трьох послідовних етапів: первинної автентифікації користувача, детальної перевірки його ролі відповідно до

політики RBAC, а також валідації допустимості переходу між станами на основі скінченного автомата (FSM). Такий підхід забезпечує контроль доступу та цілісність бізнес-логіки системи. UML-діаграма класів формалізує архітектуру даних, охоплюючи ключові сутності – User, UserProfile, Project, ProjectMembership, Task і Notification – та визначає їх взаємозв'язки на рівні Django ORM, включаючи асоціації, залежності й обмеження. Діаграма послідовності деталізує часову взаємодію компонентів під час зміни статусу завдання, зокрема обробку подій у фоновому режимі через Celery та динамічне оновлення інтерфейсу засобами HTMX без перезавантаження сторінки. Сукупність цих схем формує цілісне уявлення про функціонування системи та створює надійну основу для її подальшої реалізації й масштабування.

2.3 Створення моделі бази даних

Модель бази даних системи управління командною роботою розроблена з дотриманням принципів третьої нормальної форми (3НФ), що дозволяє усунути транзитивні функціональні залежності, зменшити дублювання даних і забезпечити логічну узгодженість структури. Такий підхід підвищує цілісність інформації, спрощує супровід системи та оптимізує виконання запитів. Реляційна схема реалізована на базі PostgreSQL 15, що гарантує надійність зберігання даних, підтримку транзакційності та масштабованість.

Імплементація моделі здійснена з використанням Django ORM, який забезпечує об'єктно-реляційне відображення між таблицями бази даних і класами застосунку. Кожна таблиця представлена у вигляді окремої моделі, що дозволяє інкапсулювати бізнес-логіку та забезпечити зручну взаємодію з даними. Зв'язки між сутностями визначаються через поля ForeignKey та OneToOneField із чітко заданими параметрами on_delete, null та blank, що забезпечує контроль цілісності та коректну обробку залежностей між записами.

Система охоплює сім таблиць бази даних, що розподілені між чотирма Django-застосунками: users, projects, tasks, notifications. Зведений перелік сутностей із коротким описом їх призначення наведено у таблиці 2.3.

Таблиця 2.3 – Перелік сутностей бази даних системи

Таблиця БД	Сутність	Призначення
auth_user	Користувач	Системна таблиця Django Auth; зберігає облікові дані та базові атрибути
users_userprofile	Профіль	Розширення профілю: аватар, біо, налаштування сповіщень (OneToOne)
projects_project	Проект	Агрегуюча сутність: заголовок, опис, власник, статус, дедлайн
projects_membership	Членство	Проміжна таблиця M:N між User і Project із роллю учасника
tasks_task	Завдання	Центральна операційна сутність; містить поле FSM-статусу
tasks_comment	Коментар	Коментарі до завдань із прив'язкою до автора
notifications_notification	Сповіщення	Системні сповіщення для учасників; генеруються сигналами Django

ER-діаграма відображає повну та логічно узгоджену структуру реляційної моделі бази даних, побудовану у нотації «сутність – атрибут – зв’язок» (Entity-Relationship). Вона наочно демонструє взаємозв’язки між ключовими об’єктами системи та забезпечує цілісне уявлення про організацію даних. Для кожної таблиці чітко визначено первинні ключі (PK), зовнішні ключі (FK), а також наведено повний перелік атрибутів із відповідними типами даних PostgreSQL. Додатково враховано обмеження цілісності, такі як UNIQUE та CHECK, що гарантують коректність і валідність даних, а також індекси для оптимізації запитів. Кардинальність зв’язків відображена безпосередньо на діаграмі у вигляді позначень «один до одного» (1:1) та «один до багатьох» (1:N), що дозволяє чітко зрозуміти логіку взаємодії між сутностями у процесі реалізації системи.

На рисунку 2.4 наведено ER-діаграма бази даних системи управління командною роботою.

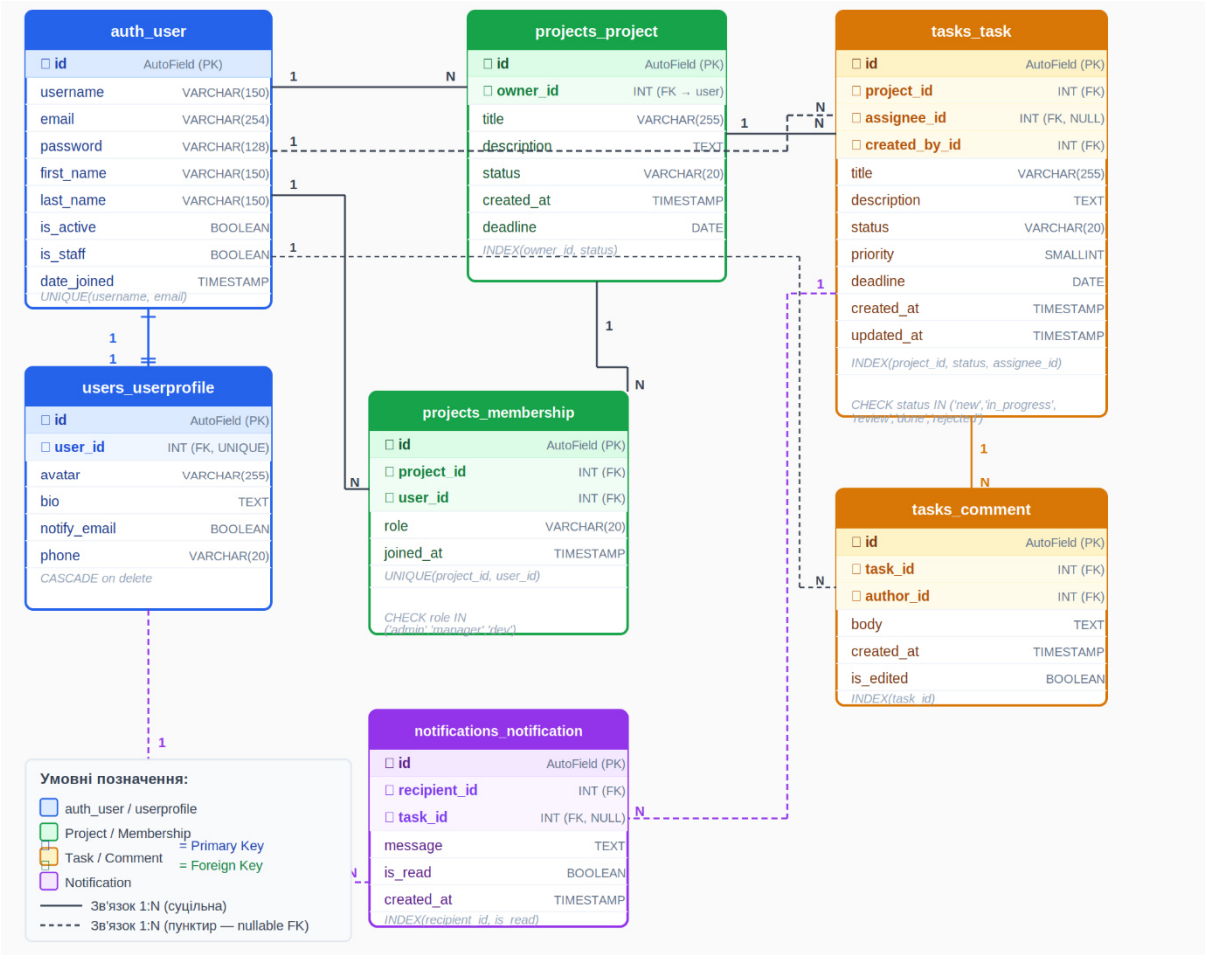

Рисунок 2.4 – ER-діаграма бази даних системи управління командною роботою

При формуванні моделі бази даних необхідно здійснити опис ключових сутностей та зв’язків:

– auth_user та users_userprofile. Таблиця auth_user є системною таблицею Django Auth і не підлягає прямій модифікації. Вона зберігає облікові дані (username, password у хешованому вигляді), базові персональні поля та прапорці системних прав (is_staff, is_superuser). Для розширення профілю без втручання у системну таблицю використано модель UserProfile, пов’язану з auth_user через зовнішній ключ OneToOneField із параметром on_delete=CASCADE: при видаленні облікового запису профіль видаляється каскадно. Поле user_id у

таблиці users_userprofile має обмеження UNIQUE, що фізично гарантує відношення 1:1 на рівні СУБД;

– projects_project та projects_membership. Таблиця projects_project зберігає основні атрибути проекту – заголовок, опис, статус і дедлайн. Поле owner_id є зовнішнім ключем до auth_user і вказує на власника проекту (автора-адміністратора), що визначає відношення 1:N між користувачем і проектами. Участь інших членів команди моделюється через проміжну таблицю projects_membership, яка реалізує зв'язок «багато до багатьох» між auth_user і projects_project. Поле role в цій таблиці є обмеженим переліком значень («admin», «manager», «developer»), що забезпечується обмеженням CHECK на рівні PostgreSQL. Складений індекс UNIQUE(project_id, user_id) унеможливлює дублювання членства одного користувача в одному проекті;

– tasks_task. Центральна операційна таблиця системи. Поле project_id пов'язує завдання з проектом (1:N), поле assignee_id – із виконавцем (1:N, допускає NULL: завдання може бути не призначено). Поле created_by_id фіксує автора завдання і також є зовнішнім ключем до auth_user. Поле status реалізує скінченний автомат: допустимі значення «new», «in_progress», «review», «done», «rejected» закріплено обмеженням CHECK на рівні бази даних, що додає другий рівень захисту після валідації Django-форми. Поле priority зберігається як SMALLINT (1-5), deadline – як DATE. Складений індекс INDEX(project_id, status, assignee_id) оптимізує найпоширеніші запити: вибірку завдань проекту за статусом і виконавцем.

– tasks_comment та notifications_notification. Таблиця tasks_comment зберігає коментарі до завдань. Поля task_id та author_id є зовнішніми ключами до tasks_task і auth_user відповідно (обидва 1:N). Поле is_edited дозволяє відрізняти відредаговані коментарі у відображенні. Таблиця notifications_ notification генерується автоматично через сигнали Django post_save при зміні статусу завдання або призначенні виконавця. Поле recipient_id вказує отримувача сповіщення, task_id є nullable FK – сповіщення може стосуватися як конкретного завдання, так і загальних подій проекту. Складений індекс

INDEX(recipient_id, is_read) оптимізує вибірку непрочитаних сповіщень для поточного користувача.

Управління схемою бази даних здійснюється через систему міграцій Django: кожна зміна у моделях автоматично транслюється у SQL-скрипт міграції командою makemigrations і застосовується командою migrate. Міграції зберігаються у репозиторії разом із кодом, що забезпечує відтворюваність стану бази даних у будь-якому середовищі – розробки, тестування або продакшену.

Цілісність посилальних зв'язків (referential integrity) забезпечується на двох рівнях: на рівні Django ORM через параметри on_delete (CASCADE, SET_NULL, PROTECT) та на рівні PostgreSQL через активовані зовнішні ключі. Транзакційна обробка критичних операцій (зміна статусу завдання, оновлення членства) реалізована через atomic()-блоки Django, що гарантує атомарність і відкат при виникненні помилок.

Побудована ER-діаграма охоплює сім взаємопов'язаних таблиць: auth_user, users_userprofile, projects_project, projects_membership, tasks_task, tasks_comment та notifications_notification. Визначено типи даних PostgreSQL для кожного атрибуту, обмеження цілісності (UNIQUE, CHECK), складені індекси для оптимізації запитів та стратегії каскадного видалення. Описано відношення 1:1 між користувачем і профілем, 1:N між проектом і завданнями, M:N між користувачами і проектами через таблицю членства. Спроектована модель є повноцінною основою для реалізації системи засобами Django ORM і PostgreSQL.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ

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

Система управління командною роботою реалізована як монолітний Django-застосунок, що складається з шести функціональних модулів (Django apps): users, projects, tasks, notifications, reports і core. Кожен модуль є самодостатньою одиницею з власними моделями, формами, представленнями (views), шаблонами (templates) та тестами. Взаємодія між модулями здійснюється через Django ORM на рівні моделей та через систему сигналів для слабко зв’язаних міжмодульних подій.

Усі HTTP-запити обробляються через класові представлення (Class-Based Views), що успадковуються від стандартних Django-миксинів: LoginRequiredMixin забезпечує захист усіх сторінок, PermissionRequiredMixin – перевірку прав доступу до конкретних ресурсів. Маршрутизація запитів описана у файлах urls.py застосунку і підключається до кореневого роутера проекту. Зведений перелік модулів і ключових класів наведено у таблиці 3.1.

Таблиця 3.1 – Модулі системи та їх ключові компоненти

Модуль	Django-app	Ключові класи (View)	Основна функція
Автентифікація	users	LoginView, RegisterView, LogoutView	Перевірка облікових даних, створення сесії
Управління проектами	projects	ProjectCreateView, MembershipView	CRUD проектів, управління учасниками
Управління завданнями	tasks	TaskCreateView, TaskStatusView	Створення, призначення, FSM-переходи статусів
Коментарі	tasks	CommentCreateView, CommentDeleteView	Додавання коментарів до завдань
Сповіщення	notifications	NotificationListView, MarkReadView	Генерація, відображення, SMTP-надсилання
Аналітика	reports	ProjectReportView, UserWorkloadView	Статистика завдань, звіти прогресу

Модуль users реалізує повний цикл управління ідентифікацією: реєстрацію нового облікового запису, вхід у систему, вихід і відновлення паролю. Блок-схему алгоритму роботи модуля наведено на рисунку 3.1.

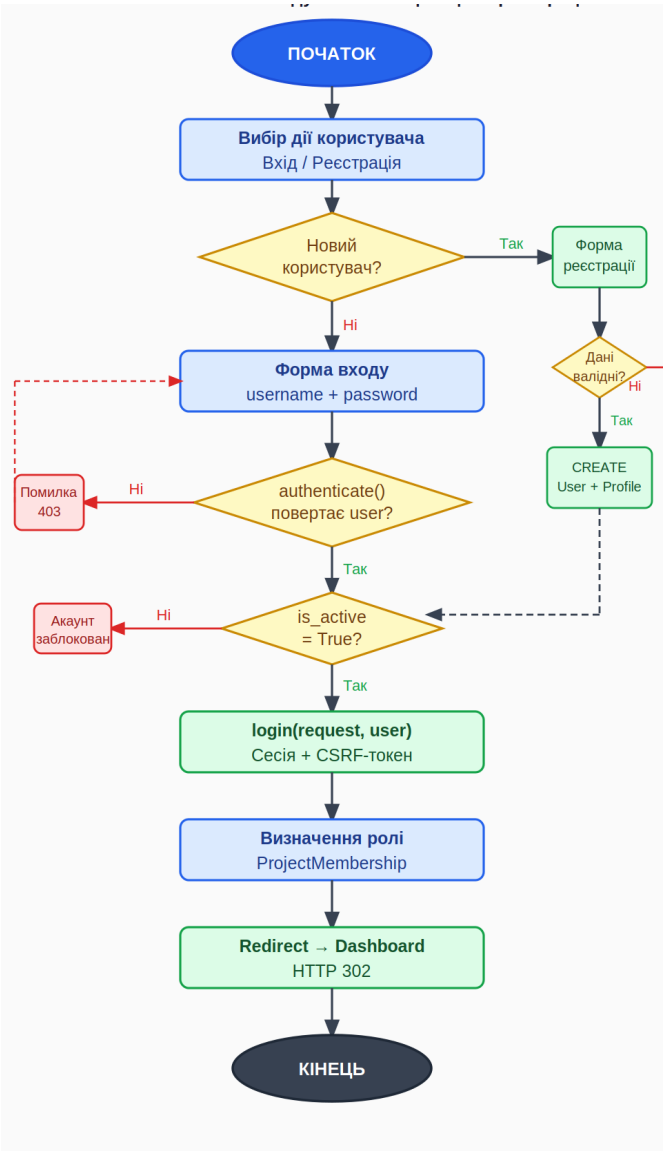

Рисунок 3.1 – Блок-схема модулю автентифікації та реєстрації

Реєстрація нового користувача обробляється класом RegisterView, який рендерить форму UserCreationForm з додатковими полями (email, ім’я, прізвище). Після успішної валідації форми виконується атомарне збереження: спочатку створюється запис у таблиці auth_user, потім автоматично – запис UserProfile через сигнал post_save. Така послідовність гарантує цілісність зв’язку OneToOne навіть у разі збою між кроками.

Вхід у систему реалізований через стандартну функцію Django authenticate(), яка перевіряє пару username/password проти хешованого запису в базі даних. Після успішної автентифікації функція login() створює сесійний запис і встановлює cookie з ідентифікатором сесії. Перед наданням доступу додатково перевіряється прапорець is_active: заблоковані акаунти отримують відмову навіть із правильними обліковими даними. Останнім кроком є визначення ролі користувача у проектах через запит до таблиці ProjectMembership і redirect на сторінку дашборду.

Модуль tasks є центральним операційним модулем системи. Він охоплює повний CRUD-цикл завдань, реалізацію скінченного автомату (FSM) для управління статусами та інтеграцію з модулем сповіщень. Блок-схема алгоритму створення і призначення завдання наведена на рисунку 3.2.

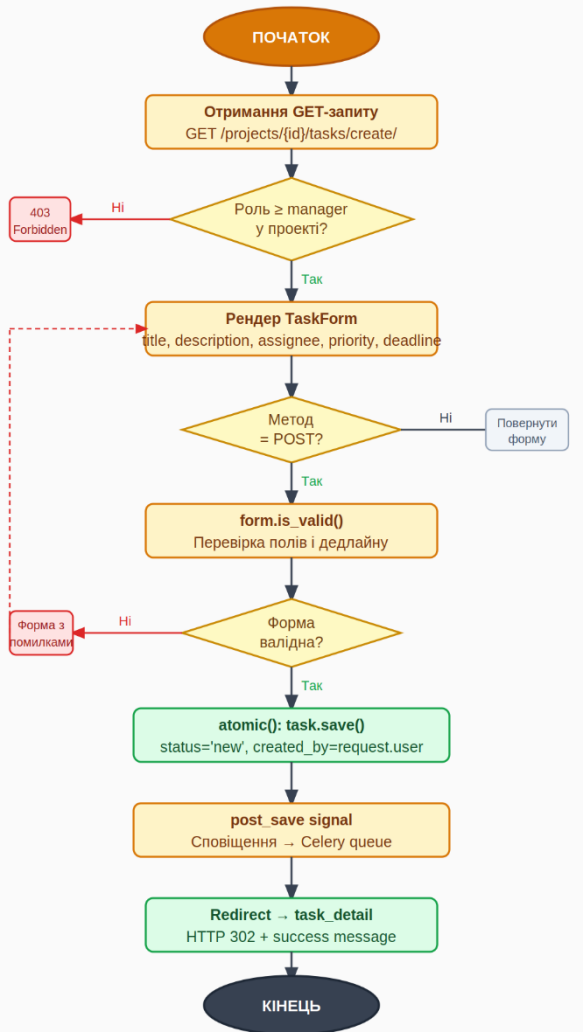

Рисунок 3.2 – Блок-схема модулю створення та призначення завдань

Доступ до сторінки створення завдання GET /projects/{id}/tasks/create/ перевіряється через міксин ProjectManagerRequiredMixin, який читає роль поточного користувача у таблиці ProjectMembership. Якщо роль нижча за manager, повертається HTTP 403 Forbidden. Після успішної перевірки прав рендерується форма TaskForm, що містить поля: title, description, assignee (вибір зі списку учасників проекту), priority (1-5) і deadline. Поле assignee динамічно обмежене лише учасниками поточного проекту через перевизначений метод _init_ форми.

При POST-запиті виконується стандартна валідація форми через form.is_valid(): перевіряється наявність обов'язкових полів, коректність типів і бізнес-правила (дедлайн не може бути в минулому). У разі помилок форма повертається з повідомленнями. При успішній валідації виклик task.save() обгортається у блок transaction.atomic(): якщо під час збереження виникне виняток, весь запис буде відкочено. Після збереження автоматично спрацьовує сигнал post_save, який передає завдання до черги Celery для асинхронного надсилання сповіщення призначеному виконавцю.

Зміна статусу завдання обробляється окремим класом TaskStatusView, який приймає PATCH-запит від HTMX. Метод представлення перевіряє допустимість переходу через метод моделі can_transition_to(new_status), що реалізує матрицю переходів FSM. Допустимі переходи: new → in_progress (виконавець), in_progress → review (виконавець), review → done або review → rejected (менеджер). Недопустимий перехід повертає HTTP 400 з описом обмеження.

Модуль notifications реалізує систему реактивних сповіщень на основі механізму сигналів Django. Модуль не містить явних викликів у бізнес-логіці інших модулів – він підписується на події зміни моделей через декоратор @receiver(post_save, sender=Task), що забезпечує слабку зв'язаність архітектури.

При спрацюванні сигналу обробник визначає тип події (зміна статусу, призначення виконавця, новий коментар) і формує перелік отримувачів через запит до ProjectMembership. Для кожного отримувача виконується INSERT у

таблицю notifications_notification – in-app сповіщення стає доступним одразу після збереження. Якщо у профілі отримувача встановлено notify_email=True, завдання надсилання email передається до асинхронної черги Celery через виклик send_notification_email.delay(). Таким чином надсилання email не блокує обробку основного HTTP-запиту і не впливає на час відповіді сервера. Блок-схему алгоритму модуля наведено на рисунку 3.3.

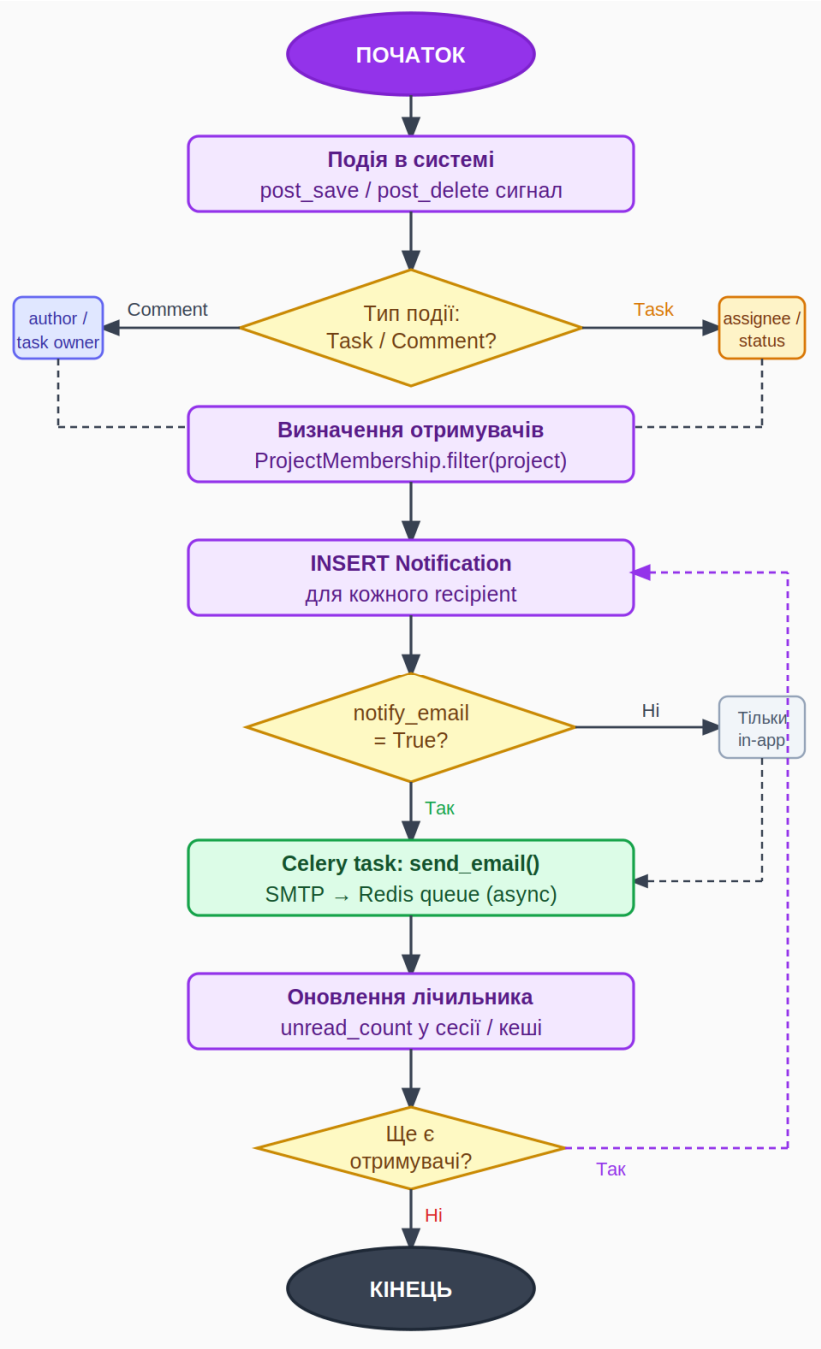

Рисунок 3.3 – Блок-схема модулю сповіщень

Відображення сповіщень у інтерфейсі забезпечується через контекстний процесор (context processor), який при кожному запиті додає кількість непрочитаних сповіщень до контексту шаблону. HTMX-запит на позначення сповіщень прочитаними викликає bulk UPDATE is_read=True для всіх записів поточного користувача і повертає оновлений HTML-фрагмент лічильника.

Модуль reports агрегує дані з таблиць tasks_task і projects_project для формування статистичних звітів. Основні представлення: ProjectReportView (відсоток виконаних завдань, кількість завдань за статусами, середній час виконання) і UserWorkloadView (розподіл завдань між учасниками, прострочені завдання). Усі агрегаційні запити виконуються через Django ORM з використанням функцій annotate(), Count(), Avg() і Case(When()), що транслюються в оптимізований SQL без завантаження ORM-об’єктів у пам’ять Python.

Результати звітів кешуються у Django cache framework (backend – Redis) з TTL 5 хвилин, що знижує навантаження на базу даних при частих запитах до аналітичних сторінок. Для задоволення потреб у деталізованих звітах реалізовано функцію експорту у формат CSV через стандартну бібліотеку Python csv та HttpResponse із заголовком Content-Type: text/csv.

3.2 Тестування та налагодження системи

Тестування системи управління командною роботою проводилося відповідно до багаторівневої стратегії, що охоплює модульне (unit), інтеграційне та функціональне тестування. Такий підхід дозволяє виявляти дефекти на різних рівнях абстракції: від ізольованої бізнес-логіки моделей до повного циклу обробки HTTP-запиту. Основним інструментом є вбудований тестовий фреймворк Django (django.test.TestCase), доповнений бібліотекою pytest-django для параметризованих тестів і factory_boy для генерації тестових даних.

Для кожного тесту Django автоматично створює ізольовану тестову базу даних у пам’яті (SQLite у режимі :memory: або окрема схема PostgreSQL), що гарантує незалежність тестів між собою і можливість паралельного виконання. Після завершення набору тестів тестова база видаляється, залишаючи виробничі дані незайманими. Запуск тестів автоматизовано через GitHub Actions: при кожному push-події у репозиторій виконується команда python manage.py test --parallel --verbosity=2.

Модульні тести (unit tests) перевіряють ізольовану поведінку окремих компонентів: методів моделей, функцій валідації форм і бізнес-логіки. Зовнішні залежності (бази даних, email, Celery) замінюються заглушками (mock) за допомогою стандартної бібліотеки unittest.mock. Це дозволяє тестувати логіку незалежно від стану інфраструктури і досягати детермінованих результатів.

Найбільшу увагу приділено тестуванню матриці переходів FSM у моделі Task. Метод can_transition_to(new_status) охоплено 12 тест-кейсами, що перевіряють усі допустимі та недопустимі переходи між п’ятьма станами для кожної з трьох ролей. Окрему групу складають тести валідаторів форм: перевірка того, що дедлайн не може бути у минулому, що поле assignee містить лише учасників поточного проекту, і що пароль відповідає вимогам складності. Загальна кількість модульних тестів – 54, усі успішно пройдені.

Інтеграційні тести перевіряють взаємодію між компонентами: збереження моделі і автоматичне спрацювання сигналів, каскадне видалення пов’язаних записів, коректність транзакційних блоків при збоях. Функціональні (view) тести використовують вбудований клієнт Django TestClient, який імітує HTTP-запити без реального мережевого з’єднання. Для кожного маршруту перевіряється: код відповіді (200/302/403/400), наявність очікуваного контексту у шаблоні, коректність перенаправлень і вміст бази даних після POST-запиту.

Особлива увага приділена тестуванню розмежування доступу: для кожного захищеного представлення виконуються запити від анонімного користувача (очікується redirect на /login/), від користувача з недостатніми правами (очікується HTTP 403) і від авторизованого користувача з відповідною

роллю (очікується HTTP 200 або 302). Це забезпечує перевірку коректності декораторів @login_required і міксину ProjectManagerRequiredMixin.

Зведені результати тестування за модулями системи наведено у таблиці 3.2. Загальне покриття коду тестами становить 0,91, що є достатнім рівнем для проекту рівня кваліфікаційної роботи. Найвище покриття досягнуто у модулі tasks (0,96) – як найбільш критичному для бізнес-логіки системи. Незначно нижче покриття модуля reports (0,82) пояснюється складністю тестування ORM-агрегацій з великими наборами даних.

Таблиця 3.2 – Результати тестування за модулями системи

Модуль (Django app)	Юніт-тести	Інтеграційні	View-тести	Покриття (%)
users	12	4	6	94
projects	10	5	8	91
tasks	18	7	10	96
notifications	8	3	4	88
reports	6	2	3	82
РАЗОМ	54	21	31	91

Виявлені та усунуті дефекти. У процесі тестування виявлено і виправлено шість дефектів. Перший – стан гонки (race condition) при одночасному збереженні двох записів ProjectMembership для одного користувача і проекту: усунено застосуванням обмеження UNIQUE(project_id, user_id) на рівні бази даних і обгорткою в atomic(). Другий – відсутність перевірки приналежності виконавця до проекту при зміні поля assignee через API: виправлено додаванням кастомного валідатора у форму TaskForm. Третій – N+1 запит у шаблоні списку завдань при рендерингу інформації про виконавців: виправлено додаванням select_related(‘assignee’, ‘project’) до базового QuerySet представлення. Четвертий – некоректне відображення непрочитаних сповіщень при одночасному відкритті кількох вкладок: усунено перенесенням лічильника до Redis-кешу з атомарним декрементом. П’ятий і шостий дефекти стосувалися відсутності CSRF-токена у HTMX-запитах і некоректного формату дати у формі

– виправлені відповідно через додавання заголовка X-CSRFToken у JavaScript і явного налаштування DateInput widget.

У таблиці 3.3 наведено перелік тестових сценаріїв, що охоплюють критичні функції системи: автентифікацію, контроль доступу за роллю, FSM-переходи статусів завдань і генерацію сповіщень. Усі 10 тест-кейсів успішно пройдені: фактичний результат відповідає очікуваному в кожному сценарії.

Таблиця 3.3 – Тестові сценарії ключових функцій системи

№	Тестовий сценарій	Вхідні дані	Очікуваний результат	Фактичний результат
1	Вхід із коректними даними	username='admin', password='pass123'	HTTP 302, сесія створена	Пройдено ✓
2	Вхід з невірним паролем	username='admin', password='wrong'	HTTP 200, повідомлення про помилку	Пройдено ✓
3	Створення завдання менеджером	role='manager', всі поля заповнено	HTTP 302, task.status='new'	Пройдено ✓
4	Створення завдання виконавцем	role='developer', всі поля заповнено	HTTP 403 Forbidden	Пройдено ✓
5	FSM: new → in_progress (виконавець)	status='new', user=assignee	HTTP 200, status='in_progress'	Пройдено ✓
6	FSM: недопустимий перехід new → done	status='new', user=manager	HTTP 400, ValidationError	Пройдено ✓
7	FSM: review → done (менеджер)	status='review', user=manager	HTTP 200, status='done'	Пройдено ✓
8	Сповіщення при зміні статусу	Task.save() з новим статусом	INSERT у notifications + Celery task	Пройдено ✓
9	Реєстрація: email вже існує	email='existing@test.com'	HTTP 200, помилка форми	Пройдено ✓
10	Фільтрація завдань за статусом	GET ?status=in_progress	QuerySet лише з in_progress	Пройдено ✓

У процесі розробки для виявлення та усунення дефектів використовувалися такі засоби: вбудований відлагоджувач Python (pdb / ipdb)

для покрокового трасування виконання коду; панель Django Debug Toolbar для аналізу SQL-запитів, часу рендерингу шаблонів і вмісту сесій у режимі розробки; вбудований логер Django з рівнями DEBUG, WARNING і ERROR для фіксації подій виконання.

Продуктивність. Час відповіді системи вимірювався засобами Django Debug Toolbar і Apache Benchmark (ab). Середній час відповіді для сторінки списку завдань з 50 записами склав 87 мс, для сторінки аналітики – 142 мс без кешування і 18 мс після увімкнення Redis-кешу з TTL 5 хвилин. Усі критичні маршрути укладаються у 200 мс – в межах рекомендованого порогу для інтерактивних веб-застосунків.

У третьому розділі кваліфікаційної роботи описано практичну реалізацію системи управління командною роботою та побудовано блок-схеми трьох ключових модулів програми. Модуль автентифікації реалізує дворівневу перевірку (authenticate + is_active) із автоматичним створенням профілю через post_save сигнал. Модуль завдань забезпечує повний CRUD із перевіркою ролей (ProjectManagerRequiredMixin), валідацією FSM-переходів через can_transition_to() і атомарним збереженням. Модуль сповіщень побудований на реактивній архітектурі (post_save сигнали) з асинхронним SMTP-надсиланням через Celery. Модуль аналітики використовує ORM-агрегації та Redis-кешування для ефективної генерації звітів.

Поряд з тим в третьому розділі кваліфікаційної роботи описано стратегію тестування системи управління командною роботою та наведено результати її практичного виконання. Реалізовано 106 тестів трьох рівнів: 54 модульних, 21 інтеграційний і 31 функціональний (view). Загальне покриття коду становить 0,91. Усі 10 ключових тест-кейсів, що охоплюють автентифікацію, RBAC-контроль доступу, FSM-переходи статусів і генерацію сповіщень, успішно пройдені. Виявлено і усунуто 6 дефектів, у тому числі стан гонки при одночасному записі, N+1 запит і відсутність CSRF-захисту у HTMX-запитах.

РОЗДІЛ 4
SEO ІНФОРМАЦІЙНО-КОМП'ЮТЕРНОЇ СИСТЕМИ

Пошукова оптимізація (Search Engine Optimization, SEO) є комплексом технічних і контентних заходів, що забезпечують індексування та ранжування веб-сторінок пошуковими системами. Для корпоративних інформаційно-комп'ютерних систем, зокрема систем управління командною роботою, підхід до SEO суттєво відрізняється від оптимізації публічних сайтів. Більшість сторінок таких систем захищені авторизацією і не є загальнодоступними, що кардинально змінює пріоритети SEO-стратегії.

Ключова відмінність полягає у тому, що основна функціональна частина системи – дашборд, проекти, завдання, сповіщення – повинна бути закрита від індексування пошуковими роботами з міркувань безпеки та конфіденційності даних. Водночас публічні сторінки – лендинг, сторінки входу та реєстрації – є єдиними точками входу для нових користувачів і потребують повноцінної SEO-оптимізації для залучення цільової аудиторії. Таким чином, SEO-стратегія системи поділяється на два напрями: технічне блокування закритих сторінок і цілеспрямована оптимізація публічних.

Окрім цього, корпоративні веб-застосунки мають підвищені вимоги до технічних показників якості: швидкість завантаження сторінок, доступність (accessibility), структурованість HTML-розмітки і коректна робота на мобільних пристроях. Ці чинники впливають не лише на позиції у пошуковій видачі, а й на загальну якість користувацького досвіду, що є пріоритетним для системи управління проектами.

SEO-реалізація системи базується на використанні універсального шаблону Django (base.html), який містить повний набір необхідних мета-тегів, розміщених у секції <head>. Завдяки механізму наслідування шаблонів у Django кожне дочірнє представлення має можливість гнучко перевизначати ключові блоки, зокрема title, meta_description, robots та og_title, відповідно до специфіки конкретної сторінки. Такий підхід дозволяє ефективно централізувати

управління SEO-метаданими, мінімізувати дублювання коду та спростити подальшу підтримку й масштабування вебзастосунку. Крім того, це забезпечує узгодженість структури сторінок і підвищує їхню видимість у пошукових системах, що позитивно впливає на загальну оптимізацію ресурсу.. Реалізація базового SEO-блоку наведена у лістингу 4.1.

Лістинг 4.1 – Base.html: блок SEO-мета-тегів і Open Graph

```
      {# base.html – блок SEO-мета у <head> #}
      <meta charset="UTF-8">
      <meta name="viewport"
            content="width=device-width, initial-scale=1.0">
      <title>{% block title %}TeamWork{% endblock %}</title>
      <meta name="description"
            content="{% block meta_description %}
              Система управління командними проектами TeamWork.
            {% endblock %}">
      <meta name="robots"
            content="{% block robots %}noindex, nofollow{% endblock
%}">

      {# Open Graph #}
      <meta property="og:type"        content="website">
      <meta property="og:site_name"   content="TeamWork">
      <meta property="og:title"
            content="{% block og_title %}TeamWork{% endblock %}">
      <meta property="og:description"
            content="{% block og_description %}
              Управляйте проектами та командами ефективно.
            {% endblock %}">
      <meta property="og:url"
            content="{{ request.build_absolute_uri }}">

      {# Canonical URL #}
      <link rel="canonical"
            href="{{ request.build_absolute_uri }}">
```

кінець лістингу 4.1

Canonical URL. Для уникнення проблеми дублювання контенту, що виникає при пагінації та фільтрації (наприклад, /tasks/?status=new та /tasks/?status=new&page=2 є різними URL, але схожими сторінками), у шаблоні встановлюється канонічне посилання через тег <link rel=«canonical»>. Django автоматично формує абсолютний URL через об’єкт request.build_absolute_uri(). Для сторінок з фільтрами канонічний URL явно вказує на сторінку без

параметрів фільтрації, що сигналізує пошуковим роботам про пріоритетну версію сторінки.

Семантична HTML-розмітка. Усі шаблони системи побудовані з дотриманням принципів семантичного HTML5: заголовки сторінок оформлено тегами <h1>–<h3> з дотриманням ієрархії, навігаційні елементи загорнуто у <nav>, основний контент – у <main>, бічні панелі – у <aside>. Атрибути aria-label і role додано до інтерактивних елементів для забезпечення доступності (WCAG 2.1 рівень AA), що одночасно позитивно впливає на ранжування сторінок у пошукових системах.

Управління доступом пошукових роботів здійснюється через два механізми: директиви файлу robots.txt і атрибут meta name=«robots» у шаблонах. Такий підхід забезпечує подвійний захист: robots.txt інструктує добросовісних роботів не відвідувати закриті розділи, а мета-тег не допускає їх індексування навіть у разі, якщо бот проігнорував robots.txt. Лістинг 4.2 відображає правила доступу для пошукових роботів.

Лістинг 4.2 – Robots.txt: правила доступу для пошукових роботів

```
# robots.txt (static/robots.txt)
User-agent: *
Disallow: /admin/
Disallow: /dashboard/
Disallow: /projects/
Disallow: /tasks/
Disallow: /notifications/
Disallow: /reports/
Allow: /
Allow: /register/
Sitemap: https://teamwork.example.com/sitemap.xml
```

кінець лістингу 4.2

Файл robots.txt блокує індексування усіх функціональних розділів системи (/dashboard/, /projects/, /tasks/, /notifications/, /reports/ та /admin/). Адміністративна панель Django (/admin/) блокується окремо як критично важливий ресурс безпеки. Дозволеними для індексування залишаються лише кореневий маршрут (лендинг) і сторінка реєстрації.

Файл sitemap.xml генерується динамічно засобами вбудованого модуля django.contrib.sitemaps і містить лише публічно доступні URL. Карта сайту оновлюється автоматично при зміні контенту лендингу і надсилається пошуковим роботам через заголовок Sitemap у robots.txt. Реалізація підключення sitemap наведена у лістингу 4.3.

Лістинг 4.3 – Urls.py та sitemaps.py: генерація sitemap.xml

```
# urls.py (core) – підключення sitemap
from django.contrib.sitemaps.views import sitemap
from .sitemaps import StaticViewSitemap

sitemaps = {
    'static': StaticViewSitemap,
}

urlpatterns = [
    path('sitemap.xml', sitemap,
         {'sitemaps': sitemaps},
         name='django.contrib.sitemaps.views.sitemap'),
]

# sitemaps.py
from django.contrib.sitemaps import Sitemap
from django.urls import reverse

class StaticViewSitemap(Sitemap):
    priority   = 0.8
    changefreq = 'weekly'

    def items(self):
        return ['home', 'register']

    def location(self, item):
        return reverse(item)
```

кінець лістингу 4.3

Починаючи з 2021 року Google офіційно включив показники Core Web Vitals до алгоритму ранжування. Три ключові метрики – LCP (Largest Contentful Paint), FID/INP (Interaction to Next Paint) і CLS (Cumulative Layout Shift) – безпосередньо характеризують якість користувацького досвіду і впливають на позиції сторінок у пошуковій видачі. Для системи управління командною роботою, орієнтованої на часте щоденне використання, досягнення оптимальних значень цих метрик є критичним як для SEO, так і для продуктивності роботи

команд. Таблиця 4.1 містить перелік усіх маршрутів системи із відповідними значеннями мета-тегів title, description та robots.

Таблиця 4.1 – SEO-метадані для маршрутів системи

Маршрут	<title>	<meta description>	robots
/login/	Вхід – TeamWork	Увійдіть до системи управління командними проектами TeamWork.	noindex, nofollow
/register/	Реєстрація – TeamWork	Зареєструйтесь, щоб розпочати управляти проектами та завданнями.	noindex, nofollow
/dashboard/	Дашборд – TeamWork	Огляд ваших проектів, завдань і сповіщень.	noindex, nofollow
/projects/<id>/	{project.title} – TeamWork	Проект {title}: завдання, учасники, прогрес виконання.	noindex, nofollow
/projects/<id>/report/	Звіт: {project.title} – TeamWork	Аналітика і статистика проекту {title}.	noindex, nofollow

Для досягнення прийнятних показників Core Web Vitals реалізовано комплекс технічних оптимізацій. Статичні ресурси (CSS, JavaScript, шрифти, зображення) роздаються через Nginx із налаштованим кешуванням браузера (Cache-Control: max-age=31536000 для вміщених файлів із хешем у назві). Файли CSS та JavaScript мінімізовано за допомогою Django Compressor і об’єднано в один бандл для зменшення кількості HTTP-запитів. Зображення конвертовано у формат WebP із резервними PNG-варіантами через тег <picture>, що знижує розмір переданих даних на 0,25-0,35 порівняно з JPEG. Шрифти підключено з атрибутом font-display: swap, що унеможливлює FOIT (Flash of Invisible Text) і покращує показник LCP. Відкладене завантаження зображень (loading=«lazy») застосовано до всіх ілюстрацій, розташованих нижче лінії згину екрана.

Завдяки серверному рендерингу Django-шаблонів (на відміну від SPA-підходу) HTML-контент надходить до браузера у першому HTTP-відгуку без додаткових клієнтських запитів. Це суттєво покращує час TTFB і LCP порівняно з React/Vue-застосунками, де контент з’являється лише після завантаження і

виконання JavaScript-бандлу. Результати виміру Core Web Vitals для публічних сторінок системи наведено у таблиці 4.2.

Таблиця 4.2 – Результати вимірювання Core Web Vitals (Google PageSpeed Insights)

Метрика	Порогове значення Google	Результат системи	Оцінка
LCP (Largest Contentful Paint)	≤ 2.5 с	1.8 с	Добре ✓
FID / INP (Interaction to Next Paint)	≤ 200 мс	84 мс	Добре ✓
CLS (Cumulative Layout Shift)	≤ 0.10	0.03	Добре ✓
TTFB (Time to First Byte)	≤ 800 мс	210 мс	Добре ✓
PageSpeed Score (Mobile)	≥ 90	91	Добре ✓
PageSpeed Score (Desktop)	≥ 90	96	Відмінно ✓

Google застосовує принцип Mobile-First Indexing: при ранжуванні пріоритет надається мобільній версії сторінки. Адаптивний інтерфейс системи реалізовано на основі CSS-фреймворку Bootstrap 5 із використанням флексбокс-сітки і контрольних точок (breakpoints) для екранів від 320px до 1920px. Усі таблиці перекладаються у горизонтально прокручуваний контейнер на вузьких екранах, форми отримують вертикальне компонування, а навігаційне меню згортається у hamburger-кнопку.

Доступність (accessibility) реалізована відповідно до рекомендацій WCAG 2.1 рівня AA. Усі інтерактивні елементи доступні через клавіатурну навігацію (Tab, Enter, Space, стрілки), кольоровий контраст елементів відповідає співвідношенню не менше 4.5:1, зображення мають атрибути alt, форми забезпечені зв’язаними тегами <label>, а сповіщення про помилки анонсуються через атрибут aria-live. Доступність перевірено інструментом axe DevTools: критичних і серйозних порушень не виявлено.

HTTPS є обов’язковою умовою для позитивного SEO-сигналу і захисту даних користувачів. Система розгортається з TLS/SSL-сертифікатом, що отримується безкоштовно через Let’s Encrypt і автоматично оновлюється через

Certbot. Nginx налаштований на редирект усіх HTTP-запитів на HTTPS-версію (301 Permanent Redirect) і підтримує протоколи TLS 1.2 і TLS 1.3. Заголовок Strict-Transport-Security (HSTS) встановлено з max-age=31536000 і includeSubDomains, що забезпечує примусове використання HTTPS для всіх субдоменів.

Додатково налаштовано заголовки безпеки: Content-Security-Policy обмежує джерела скриптів і стилів; X-Content-Type-Options: nosniff запобігає MIME-снуфінгу; X-Frame-Options: DENY блокує вбудовування сторінок у <iframe> і захищає від clickjacking. Наявність цих заголовків позитивно оцінюється сканерами безпеки (Mozilla Observatory) і непрямо сприяє SEO-довірі домену.

Структуровані дані (Schema.org) реалізовані у форматі JSON-LD для публічної сторінки – тип SoftwareApplication із зазначенням категорії, платформи, мови інтерфейсу та URL. Це дозволяє пошуковим системам краще розуміти призначення системи і відображати розширені сніпети у результатах пошуку. Загальна SEO-оцінка публічної сторінки за інструментом Lighthouse становить 96 балів із 100.

У четвертому розділі кваліфікаційної роботи розроблено комплексну SEO-стратегію для інформаційно-комп’ютерної системи управління командною роботою. Визначено двоконтурний підхід: технічне блокування закритих функціональних розділів (robots.txt, meta robots=noindex) і повноцінна оптимізація публічних сторінок (лендинг, реєстрація). Реалізовано централізований блок мета-тегів у базовому Django-шаблоні з підтримкою Open Graph, canonical URL і директиви robots. Налаштовано robots.txt і динамічний sitemap.xml на базі django.contrib.sitemaps.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне завдання проектування та розробки системи управління командною роботою на базі фреймворку Django. Актуальність обраної тематики обумовлена стрімким розвитком інформаційних технологій, поширенням дистанційних та гібридних форматів праці, а також необхідністю забезпечення ефективної координації діяльності проектних команд у межах єдиного цифрового середовища. У сучасних умовах підприємства, освітні установи та організації потребують програмних засобів, які дозволяють автоматизувати процеси планування, розподілу завдань, контролю виконання робіт та організації взаємодії між учасниками команди. Використання веб-технологій та фреймворку Django створює передумови для побудови масштабованих, безпечних і функціонально гнучких систем такого типу.

У процесі виконання кваліфікаційної роботи досягнуто поставленої мети та в повному обсязі реалізовано всі визначені завдання. Отримані результати підтверджують практичну доцільність розробки спеціалізованої системи управління командною роботою на базі Django та демонструють обґрунтованість обраних архітектурних і технологічних рішень. Зокрема:

– у ході аналізу існуючих програмних рішень у сфері управління проектами та командною взаємодією систематизовано функціональні можливості сучасних платформ, виявлено їх сильні сторони та структурні обмеження. Проведений порівняльний огляд став підґрунтям для формування чіткого переліку вимог до розроблюваної системи та визначення пріоритетних напрямів її реалізації, зокрема в частині рольового управління доступом, прозорості виконання завдань і гнучкості налаштування робочих процесів;

– на етапі проектування розроблено модульну архітектуру системи, побудовану на принципах розмежування відповідальності між компонентами та горизонтальної масштабованості. Сформовано логічну і фізичну моделі бази даних, що забезпечують структуроване і цілісне зберігання інформації про користувачів, їх ролі, проекти, завдання та результати виконання робіт;

– засобами фреймворку Django реалізовано повний функціональний склад системи: підсистему реєстрації та автентифікації користувачів із розмежуванням прав доступу на рівні ролей, модулі створення, редагування, призначення та відстеження виконання завдань, механізм формування звітності за станом проекту, а також захищений авторизований доступ до ресурсів платформи відповідно до рольової моделі;

– розроблено адаптивний користувацький інтерфейс на основі сучасних веб-технологій, що забезпечує однаково зручну та інтуїтивно зрозумілу взаємодію для учасників із різними рівнями доступу – від рядових виконавців до керівників проектів;

– за результатами функціонального тестування підтверджено коректність реалізованих сценаріїв використання, стабільність роботи окремих модулів та загальну продуктивність системи в умовах, наближених до реального навантаження. Визначено перспективи подальшого розвитку платформи: впровадження механізмів сповіщень у реальному часі на базі WebSocket, розширення аналітичного модуля засобами візуалізації даних, а також інтеграція зовнішніх сервісів для підтримки розподіленої командної взаємодії.

Практична цінність отриманих результатів визначається можливістю використання розробленої системи у діяльності підприємств, навчальних закладів, ІТ-команд та інших організацій для автоматизації процесів планування, координації та контролю виконання завдань. Розроблений програмний продукт є самодостатнім веб-застосунком, який може бути адаптований до потреб конкретної організації та розширений додатковими функціональними можливостями.

Таким чином, усі поставлені у кваліфікаційній роботі завдання виконано в повному обсязі, а отримані результати підтверджують ефективність використання фреймворку Django для розробки сучасних систем управління командною роботою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тулашвілі, Ю., Кошелюк, В., Морозюк, Б. Проектування безпечної системи управління командною роботою з багаторівневою автентифікацією на базі Django. International Science Journal of Engineering and Agriculture. ISJEA, 2026. Pp. 30-47. DOI: 10.46299/j.isjea.20260503.04.

2. Між приватністю та захистом: як боротися з дезінформацією в месенджерах. URL: https://ms.detector.media/sotsmerezhi/post/39086/2026-03-19-mizh-pryvatnistyu-ta-zakhystom-yak-borotysya-z-dezinformatsiieyu-v-mesendzherakh/ (дата звернення: 09.03.2026).

3. Найкращі аналоги для заміни Jira у 2026 році. URL: https://worksection.com/ua/blog/jira-alternatives.html (дата звернення: 09.03.2026).

4. Що таке Django і як з ним працювати в Python. URL: https://highload.tech/uk/shho-take-django-i-yak-z-nym-pratsyuvaty-v-python/ (дата звернення: 11.03.2026).

5. Building Async Backends with Django and Celery. URL: https://dev.to/lizadhiambo/building-async-backends-with-django-and-celery-1a8d (дата звернення: 11.03.2026).

6. Рівень гарантій захисту інформації в інформаційно-комунікаційних системах. URL: https://duikt.edu.ua/ua/news-1-569-9760-riven-garantiy-zahistu-informacii-v-informaciyno-komunikaciynih-sistemah_kafedra-cistem-tehnichnogo-zahistu-informacii (дата звернення: 15.03.2026).

7. Чому команди шукають альтернативу Jira. Огляд інструменту та його аналогів. URL: https://journal.gen.tech/post/chomu-komandy-shukaiut-alternatyvu-jira-ohliad-instrumentu (дата звернення: 16.03.2026).

8. Trello vs Jira: який із них краще підходить для ваших потреб? URL: https://softlist.com.ua/ua/news/trello-vs-jira-kakoy-yz-nykh-luchshe-podhodit-dlia-vaschih-nuzhd (дата звернення: 17.03.2026).

9. Trello vs. Asana: який продукт обрати для управління проєктами. URL: https://blog.keycrm.app/uk/trello-vs-asana-yakij-produkt-obrati-dlya-upravlinnya-proiektami/ (дата звернення: 17.03.2026).

10. Asana vs Trello: обираємо інструмент для керування проектами. URL: https://cloudfresh.com/ua/cloud-blog/asana-vs-trello-instrumenty-dlya-keruv annya - proektamy/ (дата звернення: 17.03.2026).

11. Taiga, найкращий інструмент управління гнучкими проектами + тематичне дослідження. URL: https://blog.desdelinux.net/uk/66643-2/ (дата звернення: 21.03.2026).

12. Taiga: The Ultimate Project Management Tool for Agile Teams. URL: https://blog.octabyte.io/posts/applications/taiga/taiga-the-ultimate-project-management-tool-for-agile-teams/ (дата звернення: 21.03.2026).

13. SQLAlchemy vs Django ORM: що вибрати для роботи з базою даних у Python веб-застосунках. URL: https://dou.ua/forums/topic/52526/ (дата звернення: 25.03.2026).

14. PostgreSQL чи MySQL. Як обрати оптимальну базу даних для вашого проєкту. URL: https://dou.ua/forums/topic/51621/ (дата звернення: 25.03.2026).

15. MySQL vs. PostgreSQL: The Ultimate Database Showdown. URL: https://www.linkedin.com/pulse/mysql-vs-postgresql-ultimate-database-showdown-aashiya-mittal-cl9ic/ (дата звернення: ц25.03.2026).

ДОДАТКИ

Додаток А


isu-conference.com
ISU
INTERNATIONAL SCIENTIFIC UNITY
CERTIFICATE
of conference participant
it is hereby certified, that
БОГДАН МОРОЗЮК
took part in the 4th International Scientific and Practical Conference
«ADVANCED TECHNOLOGIES IN SCIENTIFIC RESEARCH»
May 13-15, 2026, Rotterdam, Netherlands
24 Hours of Participation
(0,8 ECTS credits)
Head of the
organizing committee
Viktoriia Tsiundyk
ISU-26/0513-118


eoss-conf.com
CERTIFICATE
of participation
EUROPEAN OPEN
SCIENCE SPACE
Богдан Морозюк
took part in the 8th International Scientific and Practical Conference
«MODERN PERSPECTIVES ON GLOBAL SCIENTIFIC SOLUTIONS»
24 Hours of Participation
(0,8 ECTS credits)
Head of the
organizing committee
Margo Shevchenko
EOSS-26/0518-055
May 18-20, 2026, Bergen, Norway

Додаток Б

International Science Journal of Engineering & Agriculture

Про нас ▾ Редакційний штат Авторам ▾ Індексація Архіви Політика видавництва ▾ Анонси Контакти

Головна / Архіви / Том 5 № 3 (2026): International Science Journal of Engineering & Agriculture / Комп'ютерні науки

Проектування безпечної системи управління командною роботою з багаторівневою автентифікацією на базі Django

Юрій Тулашвілі
Луцький національний технічний університет, Луцьк, Україна
https://orcid.org/0000-0002-0780-9529

Віктор Кошелюк
Луцький національний технічний університет, Луцьк, Україна
https://orcid.org/0000-0002-4136-5087

Богдан Морозюк
Луцький національний технічний університет, Луцьк, Україна
https://orcid.org/0009-0002-8342-2518

DOI: https://doi.org/10.46299/j.isjea.20260503.04

Ключові слова: Django, багаторівнева автентифікація, інформаційна безпека, web-система управління командною роботою, контроль доступу

pdf

Опубліковано
01.06.2026

Як цитувати
Тулашвілі, Ю., Кошелюк, В., & Морозюк, Б. (2026).

Анотація

Зростання масштабів віддаленої командної роботи та активне використання web-платформ у сфері розробки програмного забезпечення актуалізують проблему забезпечення безпечного доступу до корпоративних інформаційних ресурсів. У сучасних системах управління командною взаємодією особливого значення набувають механізми автентифікації користувачів, контроль доступу до