МІНІСТЕРСТВО ОСВІТИ И НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І. МЕЧНИКОВА

Факультет математики, фізики та інформаційних технологій

Кафедра математичного забезпечення комп'ютерних систем

КУРСОВИЙ ПРОЕКТ

з дисципліни

«Проектування інформаційних систем»

на тему:

_____СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ_____

ДЛЯ ОРГАНІЗАЦІЇ РОБОТИ КОМАНДИ РОЗРОБНИКІВ

Студента__ІІІ___ курсу _ групи ___ІПЗ-3.02___
спеціальності 121 «Інженерія програмного забезпечення»
Вайцеховський Олександр Сергійович

Керівник: _____

Національна шкала _____
Кількість балів: _____ Оцінка: ECTS _____

Члени комісії

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

Одеса – 2025

АНОТАЦІЯ

Метою даної роботи є проектування та реалізація інформаційної системи для організації роботи команди розробників. Система розроблена з використанням фреймворка Django (із застосуванням шаблонізатора Jinja2), стильової бібліотеки Bootstrap та СУБД PostgreSQL. Архітектура рішення спрямована на уніфікацію користувацького досвіду та надійну взаємодію клієнтської частини з сервером.

У створеній інформаційній системі реалізовано інструментарій для комунікації між членами команди, управління проектами, завданнями, підзавданнями та організацію заходів. Особливістю системи є реалізація прямого підключення до бази даних через бібліотеку psycopg2 для використання повного функціоналу PostgreSQL, а також суворе розмежування прав доступу та відображення контенту відповідно до ролі користувача.

Результатом роботи є надійний веб-сервіс, що відповідає вимогам мінімально життєздатного продукту (MVP). Застосунок дозволяє ефективно організувати робочий процес, забезпечує аудит виконаної роботи та має потенціал для подальшого масштабування, зокрема інтеграції з Git-репозиторіями.

ЗМІСТ

ВСТУП

В сучасності ми зіштовхуємся з різноманітними інформаційними системами реалізованими у вигляді кремнієвих гаджетів. Винахід примітивних електрично-логічних пристроїв дав змогу автоматизувати велику кількість практичних та уявних процесів. Будь-то робота автомата з напоями чи терміналу який зараховує оплату. У всіх цих пристроях та процесах є спільні риси, такі як накопичення та обробка інформації. Ревізія товару на складі та екземплярів книжок у електронному сховищі, єдині у своїй суті – це інформаційні системи із власними сутностями, їх екземплярами та інфраструктурою, яка відповідає за керування ними.

Предметна область – це застосунок для організації роботи команди розробників, в якому створюються технічні задачі, виконується мененджмент робочим ресурсом команди та зберігаються результати роботи цієї самої команди. Метою курсового проекту є проектування та створення інформаційної системи для автоматизації повсякденних процесів (створення та виконання завдань, комунікація між співробітниками, мененджмент команди, мененджмент проекту, Q/A сесії з клієнтами та ін.) у команді розробників. Для досягнення такої мети потрібно розв’язати наступні задачі:

- проаналізувати предметну область, виокремити користувачів системи;
- виконати проектування бази даних;
- обґрунтувати вибір засобів та технологій розробки;
- розробити базу даних на основі системи керування базами даних PostgreSQL;
- розробити Front-End частину застосунку;
- розробити Back-End частину застосунку;
- протестувати програмний застосунок.

1 ПОСТАНОВКА ЗАВДАННЯ

У плануванні ІС закладено виконання таких основних задач, як:

- авторизація користувачів системи;
- створення проектів для допомоги автоматизації роботи команди;
- створення команди, для облегшення комунікації та взаємодії між її членами;
- розподілення роботи між членами команди, шляхом використання завдань та підзавдань;
- надання можливості розробникам отримання повідомлень, доступу до заходів, доступу до завдань та їх виконання;
- надання можливості менеджерам перегляду усіх учасників, надсилання їм повідомлень, створення завдань, їх перевірки;
- надання можливості голові розробки перегляду статистики роботи команди та їх учасників, стану проекту та створення заходів;
- надання можливості клієнту, як замовнику, слідкувати за прогресом розробки та брати участь у корегуванні завдань, під-завдань та участі у заходах;
- аналіз діяльності команди, прогресу виконання проектів та статистики учасників.

В інформаційній системі передбачено п’ять типів користувачів:

1) гість — роль для підключення користувача до сервісу, для можливості авторизації;
2) рядовий розробник — користувач, роль котрого це виконувати створені та надані йому у доступ під-завдання, отримання повідомлень та участі у заходах;
3) менеджер — користувач, котрий має керувати роботою розробників, шляхом створення під-завдань, завдань, перевірки завдань та надсилання повідомлень;

4) голова розробки — користувач, котрий керує проектами та командами, він створює команду, запрошує користувачів, створює проект, запрошує команду, переглядає статистику проекту, команди, учасників команди, створює заходи;
5) клієнт — цей користувач є представником замовника, його задачею є нагляд за командою, щоб ті якнайкраще виконували вимоги замовника, він може створювати, видаляти задачі, переглядати список учасників, приймати участь у заходах.

У табл. 1.1. наведено перелік задач для кожного з користувачів із зазначенням вхідної та вихідної інформації.

Таблиця 1.1. – Задачі користувачів

Задача	Вхідна інформація	Вихідна інформація
Гість		
Г 1 Авторизація в акаунт	Електрона пошта Пароль	
Рядовий розробник		
Р 1. Відображення актуальних завдань за одним або декількаами критеріями	Дата створення Строк до дедлайну Пріоритет Виконавець	Список актуальних завдань за формою: Назва завдання Пріоритет Термін виконання Прогрес виконання(кількість виконаних підзавдань) Виконавці
Р 2. Виконання поставлених задач	Id користувача Посилання на результат роботи	
Р 3. Повідомлення від менеджера		Повідомлення за формою: Id відправника Назва повідомлення Текст повідомлення

Р 4 Перегляд дошки завдань	Id виконавця Id команди	Дошка завдань за формою: Назва завдання Пріоритет Термін виконання Id виконавців
Молодший мененджер		
М 1 Керування задачами на проекті		
М 1.1 Створення Підзавдання	Назва Опис Посилання на медіафайли завдання	Id підзавдання
М 1.2 Моніторинг завдання	Id завдання	Назва Підзавдання Посилання на медіафайли завдання Строк виконання Пріоритет Прогресс виконання завданя Id виконавців
М 1.3 Відміна завдання	Id завдання Текст причини Дата відміни	
М 1.4 Усунення виконавця із завдання	Id завдання Id виконавця	
М 1.5 Додавання виконавця до завдання	Id завдання Id виконавця	
М 2 Комунікація із розробниками		
М 2.1 Перегляд учасників команди	Id команди	Id виконавця Ім’я Прізвище Організація Роль Id завдань, на які назначений
М 2.2 Надсилання повідомлення робітнику	Id отримувача Назва повідомлення	Id повідомлення

	Текст повідомлення	
Голова команди		
К 1 Керування проектом		
К 1.1 Створення команди	Назва організації Опис	Id команди
К 1.2 Назначення проекту для команди	Id проекту Назва проекту Посилання на медіафайли Команда Строки	
К 1.3 Надання пріоритетності завданню	Id завдання Пріоритет	
К 1.4 Завершення проекту	Id проекту Статус	Фіналізування усієї робочої документації по проекту
К 2 Робота із персоналом		
К 2.1 Додавання учасника в команду	Id учасника	
К 2.2 Видалення учасника з команди	Id учасника	
К 2.3 Зміна ролі учасника команди	Id учасника Роль	
К 3 Аудит команди		
К 3.1 Перегляд продуктивності робітника	Id виконавця Термін за який потрібно переглянути продуктивності	Статистика учасника за формою: Ім’я Прізвище Роль Кількість виконаних завдань Кількість пропущених завдань
К 3.2 Перегляд продуктивності команди	Id команди Термін за який потрібно переглянути продуктивності	Статистика команди за формою: Назва команди Id проекту Середній термін виконання завдань

		Прогрес виконання завдань Кількість провалених Кількість виконаних
К 4 Організація заходів		
К 4.1 Створення заходу	Тема Час Пріоритет Тривалість Посилання на медіафайли	Id заходу
К 4.2 Відміна заходу	Id заходу Причина	Id повідомлення
К 4.3 Додавання учасника заходу	Id заходу Id учаснику	
К 4.4 Усунення учасника із заходу	Id заходу Id учаснику	
Клієнт		
І 1 Аудит команди		
І 1.1 Перегляд продуктивності робітника	Id виконавця Термін за який потрібно переглянути продуктивності	Статистика учасника за формою: Ім’я Прізвище Роль Кількість виконаних завдань Кількість пропущених завдань
І 1.2 Перегляд продуктивності команди	Id команди Термін за який потрібно переглянути продуктивності	Статистика команди за формою: Назва команди Id проекту Середній термін виконання завдань Прогрес виконання завдань Кількість провалених Кількість виконаних
І 2 Корегуваня проекту		

I 2.1 Надання пріоритетності завданню	Id завдання Пріоритет	
I 2.2 Завершення проекту	Id проекту Статус	Фіналізування усієї робочої документації по проекту
I 3 Робота із персоналом		
I 3.1 Додавання учасника в команду	Id учасника	
I 3.2 Видалення учасника з команди	Id учасника	
I 3.3 Зміна ролі учасника команди	Id учасника Роль	

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Для вирішення задачі обраним підходом є дво-ланкова архітектура на основі монолітного застосунку та реляційної бази даних. Оскільки система оперує чутливою інформацією та секретними ключами, така архітектура забезпечує перевагу у вигляді спрощеної моделі розгортання та централізованого керування безпекою. Крім того, це дозволяє уникнути зайвої складності розподілених систем на старті, зберігаючи при цьому потенціал для подальшого масштабування.

Рисунок 2.1 — Схема дво-ланкової архітектури

Основна концепція побудови цього моноліту полягає у його внутрішньому структуруванні, що передбачає логічний поділ застосунку на три рівні:

- Шар представлення (Presentation Layer) — відповідає за взаємодію з клієнтом, обробку HTTP-запитів та первинну валідацію вхідних даних;
- Шар бізнес-логіки (Business Layer) — централізує правила, алгоритми та механізми обробки сутностей предметної області, ізолюючи їх від інфраструктурних деталей;
- Шар доступу до даних (Data Access Layer) — забезпечує абстракцію над сховищем даних, відповідаючи за комунікацію з базою даних та передачу інформації вищим рівням системи.

Серед численних шаблонів проектування, вибір пав на MVT (англ.MODEL VIEW TEMPLATE design pattern):

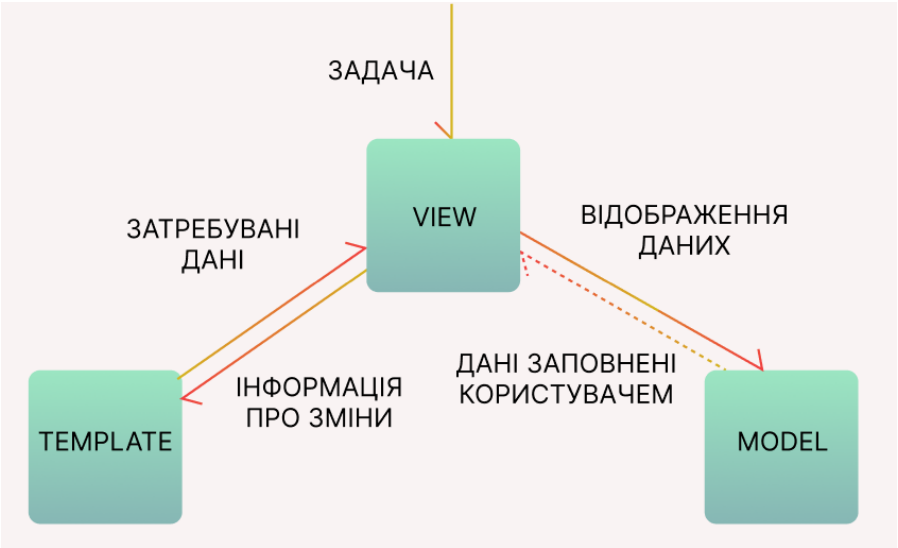

Рисунок 2.2 — Схема «Моноліт + База данних»

Така архітектура забезпечує чітке розмежування між описом даних, логікою їх обробки та візуальним представленням. Цей підхід є гнучким і спрощує підтримку коду, дозволяючи змінювати дизайн сторінок (шаблони) без втручання в основну бізнес-логіку застосунку, або ж модифікувати структуру даних, не переписуючи весь інтерфейс. Логіка цього шаблону розподіляється наступним чином:

- model (Модель) — відповідає за структуру даних та взаємодію з базою даних. Вона визначає поля, поведінку даних та правила їх валідації, абстрагуючись від SQL-запитів;
- view (Представлення) — шар бізнес-логіки, який обробляє вхідні запити від користувача (URL). Він отримує необхідні дані з Моделі, застосовує до них алгоритми обробки та передає їх у Шаблон для формування відповіді;
- template (Шаблон) — відповідає за генерацію інтерфейсу, який бачить користувач (HTML-код). Він отримує готові дані від Представлення (View) та динамічно вбудовує їх у статичну розмітку сторінки для відправки клієнту.

3 ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ

Сутності, їх властивості та обмеження, які використовуються у вирішені поставлених задач описані в таблиці 3.1.

Таблиця 3.1 — Опис сутностей, їх властивостей та обмежень

Атрибут	Опис	Обмеження
Client(Клієнт)		
ClientId	Ідентифікатор користувача	Первинний ключ
Name	Ім'я користувача	Обов’язкове
Surname	Прізвище користуваща	Обов’язкове
Email	Електрона адреса користувача	Унікальне, обов’язкове, за шаблоном: «[a-z]{6}@[a-z]{3}.[a-z]{2}(. [a-z]{2}){0,1}» «%_____@____%.__%»
Avatar	Медіапосилання на аватар користувача	
Role	Роль користувача у команді	Значення з множини(«Guest», «Developer», «Manager», «Lead»)
Team(Команда)		
TeamId	Ідентифікатор команди	Первинний ключ
Title	Назва команди	Обов’язкове
Description	Опис команди	Обов’язкове
TeamHandler(Командний розподільник)		
TeamId	Ідентифікатор команди	Зовнішній ключ зв’язку із Team, не порожнє
ClientId	Ідентифікатор клієнту	Зовнішній ключ зв’язку із Client, не порожнє
		Унікальна комбінація із двох зовнішніх ключів TeamId та ClientId

Продовження таблиці 3.1:

Project(Проект)		
ProjectId	Ідентифікатор проекту	Первинний ключ
Title	Назва проекту	Обов’язкове
Status	Статус підзавдання	Значення з множини(«Active», «Freezed», «Completed», «Failed», «Canceled»), обов’язкове
CreationDate	Дата початку проекту	Обов’язкове
Deadline	Дата завершення проекту	Обов’язкове
Medialink	Медіапосилання	Може бути порожнє
ProjectHandler (Проектний розподільник)		
TeamId	Ідентифікатор команди	Зовнішній ключ зв’язку із Team, не порожнє
ProjectId	Ідентифікатор проекту	Зовнішній ключ зв’язку із Project, не порожнє
		Унікальна комбінація із двох зовнішніх ключів TeamId та ProjectId
Message (Повідомлення)		
MessageId	Ідентифікатор повідомлення	Первинний ключ, унікальне, обов’язкове
SenderId	Ідентифікатор відправника	Зовнішній ключ зв’язку із Client, не порожнє
ReceiverId	Ідентифікатор отримувача	Зовнішній ключ зв’язку із Client, не порожнє
Title	Назва повідомлення	Обов’язкове
Text	Текст повідомлення	Обов’язкове
Date	Дата повідомлення	Обов’язкове

Продовження таблиці 3.1:

Task(Завдання)		
TaskId	Ідентифікатор завдання	Первинний ключ, обов’язкове
Priority	Пріоритет завдання	Значення з множини(«Extreme», «High», «Average», «Low»), обов’язкове
Title	Назва завдання	Обов’язкове
Description	Опис завдання	Обов’язкове
CreationDate	Дата створення	Обов’язкове
Deadline	Дедлайн	Обов’язкове
Status	Статус завдання	Значення з множини(«Active», «Freezed», «Completed», «Failed», «Canceled»), обов’язкове
ProjectId	Ідентифікатор проекту	Зовнішній ключ зв’язку із Project, не порожнє
ResultLink	Медіапосилання на результат виконаних робіт	Може бути порожнє
UnderTask (Підзавдання)		
UnderTaskId	Ідентифікатор завдання	Первинний ключ, обов’язкове
AuthorId	Ідентифікатор автора	Зовнішній ключ зв’язку із Client, не порожнє
DeveloperId	Ідентифікатор виконавця	Зовнішній ключ зв’язку із Client, не порожнє
Title	Назва підзавдання	Обов’язкове
Description	Опис підзавдання	Обов’язкове
CreationDate	Дата створення	Обов’язкове
CompletitionDate	Дата виконання	
Status	Статус підзавдання	Значення з множини(«Active», «Freezed», «Completed», «Failed», «Canceled»), обов’язкове
ResultLink	Медіапосилання на результат виконаних робіт	Може бути порожнє

Продовження таблиці 3.1:

TaskHandler (Розподільник завдання)		
Id	Ідентифікатор таблиці	Первичний ключ
TaskId	Ідентифікатор завдання	Зовнішній ключ зв’язку із Task, не порожнє
ClientId	Ідентифікатор клієнту	Зовнішній ключ зв’язку із Client, не порожнє
		Унікальна комбінація із двох зовнішніх ключів TaskId та ClienId
Event(Захід)		
EventId	Ідентифікатор заходу	Первинний ключ
Priority	Пріоритет заходу	Значення з множини(«Extreme», «High», «Average», «Low»)
AuthorId	Ідентифікатор організатора	Зовнішній ключ зв’язку із Client, не порожнє
Theme	Тема заходу	Обов’язкове
Date	Час проведення	Обов’язкове
Duration	Час заходу	Обов’язкове
Link	Посилання на захід	Непорожнє
Description	Опис заходу	Обов’язкове
EventHandler (Розподільник заходу)		
Id	Ідентифікатор таблиці	Первичний ключ
EventId	Ідентифікатор заходу	Зовнішній ключ зв’язку із Event, не порожнє
ClientId	Ідентифікатор клієнту	Зовнішній ключ зв’язку із Client, не порожнє
		Унікальна комбінація із двох зовнішніх ключів EventId та ClienId

Сутності мають між собою такі види зв’язків:

Один до багатьох, 1:N:

Event та Client формалізація відбулась із додаванням зовнішнього ключа authorId до відношення Event.

Client та Message формалізація відбулась завдяки додавання двох зовнішніх ключів до Message, senderId та receiverId.

Project та Task формалізація відбулась завдяки додавання зовнішнього ключа до відношення Task, projectId.

Client та UnderTask, формалізація завдяки додавання зовнішнього ключа на Client, authorId та developerId.

Багато до багатьох, N:N:

Client та Task формалізація завдяки додавання відношення TaskHandler з двома полями clientId та taskId.

Client та Event формалізація завдяки додавання відношення EventHandler з двома полями clientId та eventide.

Client та Team формалізація завдяки додавання відношення TeamHandler із полями clientId, teamId.

Team та Project формалізація завдяки додавання відношення ProjectHandler із полями teamId та projectId

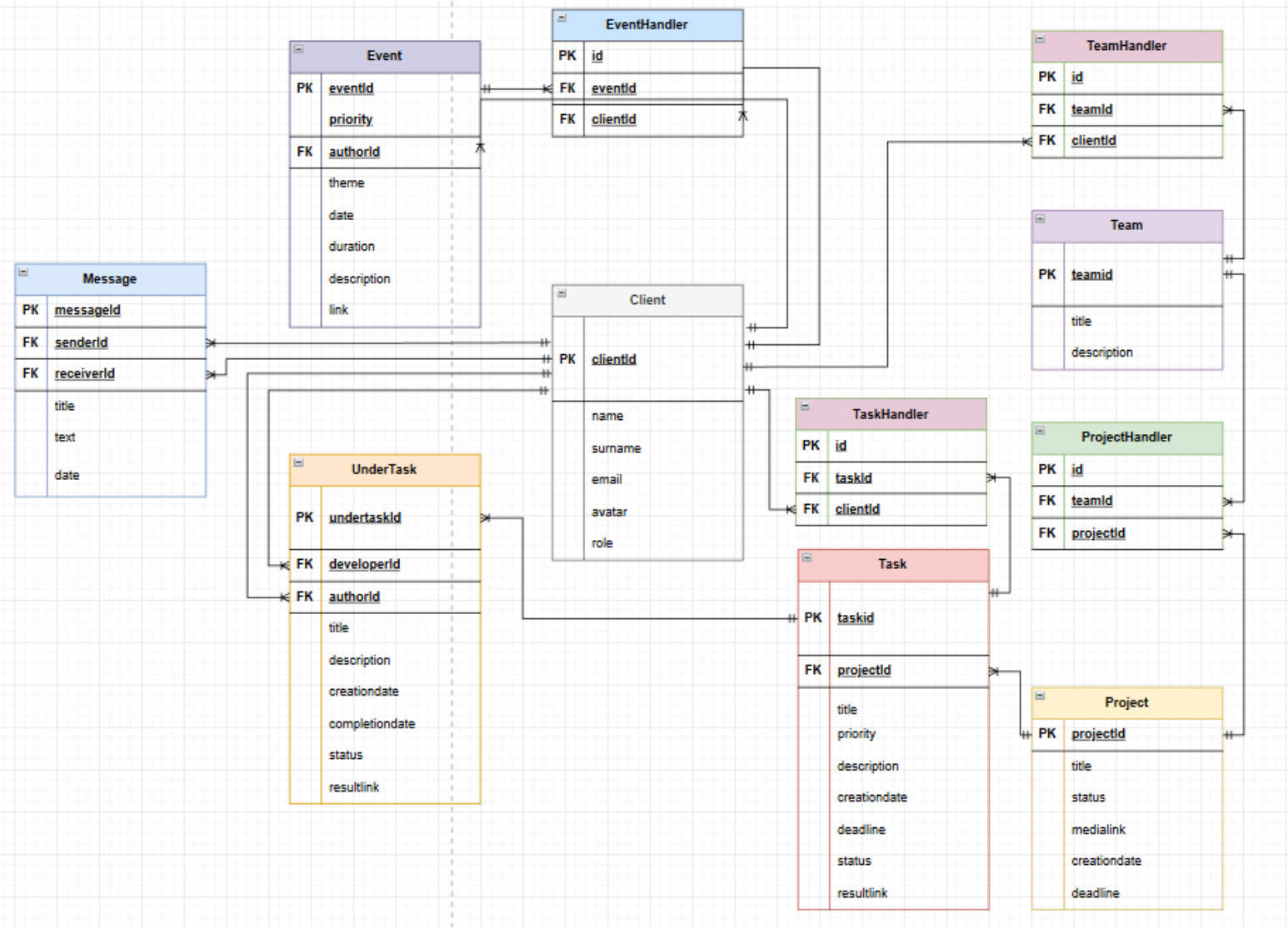

Рисунок 3.1 — ER-діаграма для предметної області «Організація роботи команди розробників»

4 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Інформаційна система розроблена у вигляді веб-застосунку з використанням монолітної архітектури. Для реалізації застосунку обрано наступний стек технологій:

Django — високорівневий Python фреймворк, використаний для реалізації бізнес-логіки, маршрутизації та генерації користувацького інтерфейсу (шар Templates);

PostgreSQL — об'єктно-реляційна система керування базами даних (СКБД) для надійного зберігання та обробки інформації.

Серед засобів веб-розробки, вибір зупинено на Django завдяки таким перевагам:

- архітектура MVT (Model-View-Template): ідеально підходить для монолітної структури, забезпечуючи чітке розділення роботи з даними, логікою та відображенням HTML-сторінок в межах одного проекту;
- висока швидкість розробки: наявність вбудованої панелі адміністрування, системи автентифікації та обробки форм дозволяє зосередитись на унікальних функціях системи;
- потужний ORM: дозволяє працювати з базою даних через об'єкти Python, автоматизуючи створення схем даних та міграцій;
- вбудована безпека: захист від основних вразливостей (SQL Injection, XSS, CSRF) реалізовано на рівні ядра фреймворку.

Серед великої кількості систем керування базами даних, перевага надана саме PostgreSQL через такі її характеристики:

- відкритий код та безкоштовна ліцензія, що є економічно вигідним для проекту;
- повна підтримка реляційної моделі, яка гарантує цілісність даних (ACID) та строгу структуру зв'язків;

- багатий функціонал SQL: підтримка складних вкладених запитів, віконних функцій, тригерів та збережених процедур;
- наявність розширення PL/pgSQL, котре спрощує написання складної логіки обробки даних та двє можливість зручніше писати збережені процедури, тощо;
- механізм MVCC, що дозволяє ефективно виконувати паралельні запити без блокування читання, підвищуючи швидкодію системи;
- гнучка система контролю доступу, яка дозволяє розмежувати права користувачів БД на рівні таблиць або окремих стовпців.

Для реалізації клієнтської частини (Front-End) обрано використання шаблонного рушія Jinja2 у поєднанні з CSS-фреймворком Bootstrap:

- jinja2 забезпечує швидку генерацію динамічного HTML-коду на стороні сервера, підтримує модульність, наслідування шаблонів та безпечний рендеринг даних, що дозволяє відділити логіку відображення від бізнес-логіки.
- bootstrap гарантує адаптивність інтерфейсу під мобільні та десктопні пристрої, а також значно пришвидшує розробку завдяки наявності готових компонентів (сітка, форми, кнопки).

Для серверної частини (Back-End) використано Django. Це популярний Python-фреймворк, який містить необхідні інструменти «з коробки»: вбудовану систему автентифікації, потужний ORM, механізм міграцій та адміністративну панель. Такий підхід дозволяє в найкоротші терміни розробити MVP (робочий прототип) із максимальним функціоналом.

Для розгортання веб-серверу обрано зв’язку Nginx та Gunicorn:

- Nginx виступає в ролі веб-сервера та реверс-проксі. Його задача — високоефективна роздача статичного контенту (CSS, зображення), обробка HTTPS-з'єднань та захист від перенавантажень.

- Gunicorn (Green Unicorn) виконує роль WSGI-сервера застосунків. Він слугує інтерфейсом для виконання коду Django. Використання Gunicorn є критично необхідним, оскільки стандартний сервер розробки Django (runserver) не оптимізований для навантажень, небезпечний для відкритого доступу і призначений виключно для процесу написання коду та налагодження.

Для забезпечення контролю версій та автоматизації процесів розгортання (CI/CD) обрано розподілену систему Git та хмарну платформу GitHub.

Серед численних переваг використання зв'язки Git та GitHub, є:

- розподілена архітектура всього репозиторію з повною історією змін, що дозволяє працювати на локальній машині без постійного підключення до центрального сервера;
- ефективна модель розгалуження, легке створення та злиття гілок дозволяє реалізовувати нові функції ізольовано від основного стабільного коду, мінімізуючи ризики конфліктів;
- надійність збереження історії завдяки механізмам хешування, які гарантують цілісність даних;
- інтегрований CI/CD, завдяки GitHub Actions. Вбудований інструментарій дозволяє створювати автоматизовані конвеєри для запуску тестів та автоматичного розгортання проекту на сервер при кожному оновленні гілки;

Засоби розробки, які використано для реалізації проекту, це — Visual Studio Code для Python програм та Django зокрема, Windows PowerShell для виконання команд розгортання середовищ розробки, PgAdmin, як зручний клієнт виконання та візуалізації PostgreSQL таблиць та запитів, PowerShell як розширений термінал консолі для виконання команд на ОС Windows.

5 МОДЕЛЬ ПРОГРАМНОГО ДОДАТКУ

Програмна реалізація базується на використанні архітектурного патерну MVT (Model-View-Template), який є адаптацією класичного MVC (Model-View-Controller), а також використовує елементи патернів Monolith (Монолітна архітектура) та Active Record (для роботи з даними). В архітектурі використовуються різні логічні шари (Model Layer для опису структури даних та ORM, View Layer для реалізації бізнес-логіки та маршрутизації, Template Layer для генерації інтерфейсу), це допомагає у розподілі відповідальності та структуруванні бізнес-логіки додатку. В даній частині ми використовуємо такі інструменти та бібліотеки, як фреймворк Django, шаблонізатор Jinja2 та адаптер бази даних Psycopg2.

Архітектура інтерфейсу веб-додатку базується на механізмі динамічного рендерингу контенту з використанням рольової моделі доступу (англ. Role-Based Access Control, RBAC). Така структура дозволяє використовувати єдину точку входу, змінюючи наповнення шаблону на рівні серверу моноліту залежно від ролі користувача. Така архітектура дозволяє створити уніфікований користувацький досвід, без ускладнення використання веб-додатку користувачами в різних ролях.

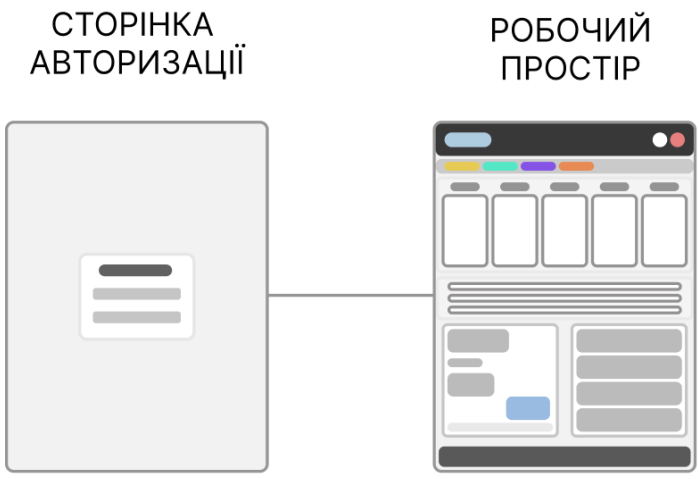

Рисунок 5.1 — Ієрархія сторінок веб-додатку ‘Task-manager’

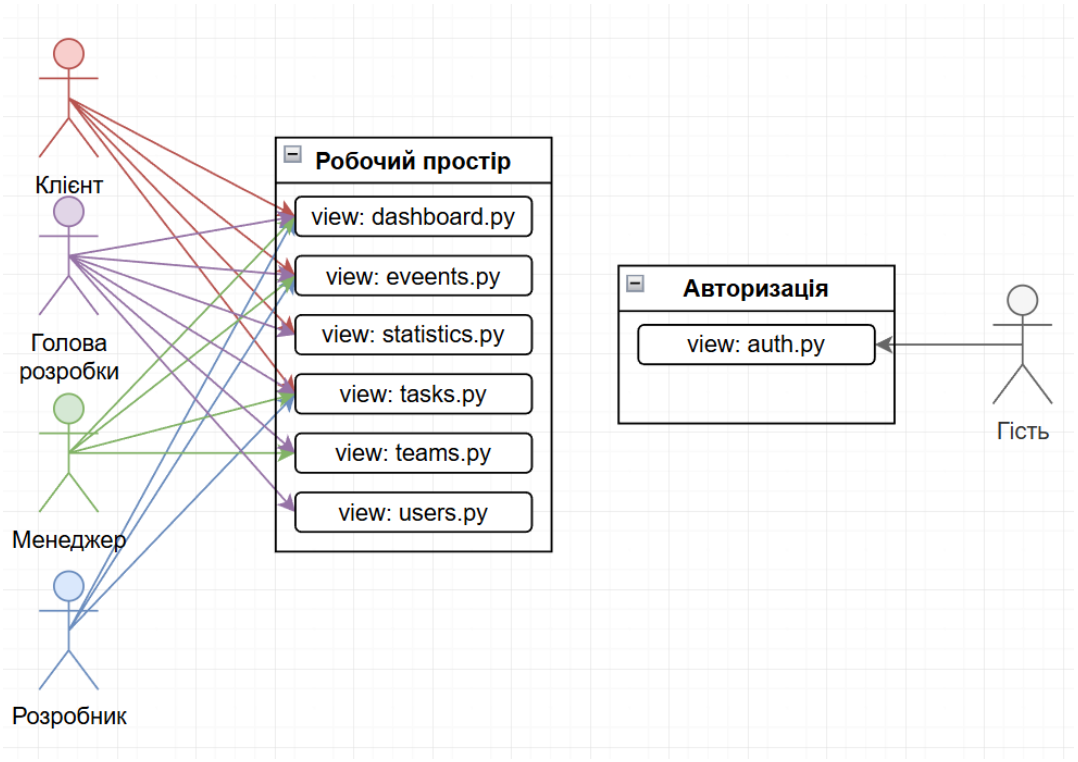

Рисунок 5.2 — Діаграма рольового доступу для веб-додатку ‘Task-Manager’

6 СТВОРЕННЯ БАЗИ ДАНИХ

Далі будуть описані об’єкти бази даних, права користувачів та інші положення. Повний код створення бази даних знаходиться в додатку В.

Домени:

1) PriorityDomain: допустимі значення для пріоритету завдання та заходу;
2) StatusDomain: допустимі значення для статусу проекту, завдання та під-завдання.

Таблиці:

1) Client — інформація про користувача;
2) Team — інформація про команду (назва, ідентифікатор, опис);
3) TeamHandler — частина формалізації зв’язку багато до багатьох між Client та Team;
4) Project — інформація про проект та час його створення та закінчення;
5) ProjectHandler — частина формалізації зв’язку багато до багатьох між Project та Team;
6) Task — інформація про завдання, разом із термінами та автором;
7) TaskHandler — частина формалізації зв’язку багато до багатьох між Task та Client, яка визначає виконавців;
8) UnderTask — інформація про завдання, разом із термінами та автором;
9) Message — повідомлення від одного користувача до іншого із текстом, назвою та датою відправлення;
10) Event — Інформація про захід із посиланням на автора та датою і тривалістю заходу;
11) EventHandler — частина формалізації зв’язку багато до багатьох між Event та Client, яка визначає учасників заходу.

Представлення:

1) LeadEmails — для показу лише пошти Голови розробки;
2) ManagerEmails — для показу лише пошти Менеджера;
3) ClientEmails — для показу лише пошти Клієнта;
4) Client_Ranking_View — для показу рейтингу користувачів;
5) Project_Structure_Hierarchy — для показу ієрархії проекту

Розподіл прав доступу для користувачів бази даних наведено в таблиці 6.1.

Таблиця 6.1 — Права ролей користувачів

Таблиця	Користувачі БД				
	connect_user	Developer	Manager	Lead	Individual
Client	S		S	ISUD	S
Team		S	S	IS	S
TeamHandler					
Project		S	S	IS	S
ProjectHandler					
Task		SU (resultlin)	ISUD	ISUD	ISUD
TaskHandler			ISD	ISD	ISD
UnderTask		SU (resultlin)	ISUD	ISUD	ISUD
Message		IS	ISD	ISD	IS
Event		S	S	ISUD	S
EventHandler		S	S	ISD	S
Представлення	Користувачі БД				
	connect_user	Developer	Manager	Lead	Individual
LeadEmails	S	S	S	S	S
ManagerEmails	S	S	S	S	S
ClientEmails	S	S	S	S	S
Client_Ranking_View				S	S
Project_Structure_Hierarchy			S	S	S

Продовження таблиці 6.1

Збережена процедура	Користувачі БД				
	connect_user	Developer	Manager	Lead	Individual
GetClientTasksStatistics				EXEC	EXEC
GetManagerStatisticsDetailed				EXEC	EXEC

Літери позначають:

- S: перегляд вмісту таблиці;
- I: додавання нової інформації до таблиці;
- U: редагування існуючої інформації;
- D: видалення інформації з таблиці;
- EXEC: право на виклик збереженої процедури

7 ЗАПИТИ ДО БАЗИ ДАНИХ ДЛЯ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ

7.1 Рядовий розробник

Відображення актуальних завдань:

```
SELECT
    t.Title AS "Назва завдання",
    t.Priority AS "Пріоритет",
    t.Deadline AS "Термін виконання",
    CONCAT(
        COUNT(CASE WHEN st.Status = 'Done' THEN 1 END),
        '/',
        COUNT(st.SubtaskId)
    ) AS "Прогрес виконання"
FROM Task t
LEFT JOIN TaskHandler th ON t.TaskId = th.TaskId
LEFT JOIN Subtask st ON t.TaskId = st.TaskId
WHERE
    t.Status != 'Done'
    AND (@creation_date IS NULL OR t.CreationDate =
@creation_date)
    AND (@deadline IS NULL OR t.Deadline <= @deadline)
    AND (@priority IS NULL OR t.Priority = @priority)
    AND (@executor_id IS NULL OR th.ClientId = @executor_id)
GROUP BY t.TaskId, t.Title, t.Priority, t.Deadline
ORDER BY
    CASE t.Priority
        WHEN 'Critical' THEN 1
        WHEN 'High' THEN 2
        WHEN 'Medium' THEN 3
        WHEN 'Low' THEN 4
    END,
    t.Deadline ASC;
```

Лістинг 7.1 — «Рядовий розробник»: Відображення актуальних завдань

Виконання завданя:

```
UPDATE Task
SET
    Status = 'Done',
```

```
    ResultLink = @result_link
WHERE TaskId = @task_id
    AND EXISTS (
        SELECT 1
        FROM TaskHandler th
        WHERE th.TaskId = @task_id
        AND th.ClientId = @executor_id
    );
```

Лістинг 7.2 — «Рядовий розробник»: Виконання завдання

7.2 «Менеджер»

Створення підзавдання:

```
INSERT INTO UnderTask (
    TaskId,
    DeveloperId,
    AuthorId,
    Title,
    Description,
    ResultLink
)
VALUES (
    @task_id,
    @developer_id,
    @author_id,
    @title,
    @description,
    @media_links
)
RETURNING UnderTaskId;
```

Лістинг 7.3 — «Менеджер»: Створення підзавдання

Надсилання повідомлення «Розробнику»:

```
INSERT INTO Message (
    SenderId,
    ReceiverId,
    Title,
    Text
)
VALUES (
    @sender_id,
    @receiver_id,
```

```
    @title,
    @text
)
RETURNING MessageId;
```

Лістинг 7.4 — «Менеджер»: Надсилання повідомлення робітнику

7.3 «Голова розробки»

Створення заходу:

```
INSERT INTO Event (
    AuthorId,
    Theme,
    Description,
    Date,
    Priority,
    Duration,
    Link
)
VALUES (
    @author_id,
    @theme,
    @description,
    @date,
    @priority,
    @duration,
    @media_link
)
RETURNING EventId;
```

Лістинг 7.5 — «Голова розробки»: Створення заходу

Додавання учаснику до заходу:

```
INSERT INTO EventHandler (
    EventId,
    ClientId
)
VALUES (
    @event_id,
    @client_id
);
```

Лістинг 7.6 — «Голова розробки»: Додавання учаснику до заходу

7.4 «Клієнт»

Перегляд продуктивності робітника:

```
SELECT
    c.Name AS "Ім'я",
    c.Surname AS "Прізвище",
    c.Role AS "Роль",
    COUNT(CASE
        WHEN t.Status = 'Done'
        AND t.Deadline >= @start_date
        AND t.Deadline <= @end_date
        THEN 1
    END) AS "Кількість виконаних завдань",
    COUNT(CASE
        WHEN t.Status != 'Done'
        AND t.Deadline < CURRENT_DATE
        AND t.Deadline >= @start_date
        AND t.Deadline <= @end_date
        THEN 1
    END) AS "Кількість пропущених завдань"
FROM Client c
LEFT JOIN TaskHandler th ON c.ClientId = th.ClientId
LEFT JOIN Task t ON th.TaskId = t.TaskId
    AND t.Deadline >= @start_date
    AND t.Deadline <= @end_date
WHERE c.ClientId = @executor_id
GROUP BY c.ClientId, c.Name, c.Surname, c.Role;
```

Лістинг 7.7 — «Клієнт»: Перегляд продуктивності робітника

Надання пріоритетності завданню:

```
UPDATE Task
SET Priority = @priority
WHERE TaskId = @task_id;
```

Лістинг 7.8 — «Клієнт»: Надання пріоритетності завданню

8 ПРОГРАМНА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОГО ДОДАТКУ

Далі буде наведено приклад роботи коду клієнтського додатку.

Найважливішою частиною роботи усього сервісу є авторизація користувача:

```
def custom_login(request):
    error = ''
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')

        try:
            user = Client.objects.get(email=email)
            db_user = f"user_{user.clientid}"
            db_conf = settings.DATABASES['default']

            try:
                with psycopg.connect(
                        dbname=db_conf['NAME'],
                        user=db_user,
                        password=password,
                        host=db_conf['HOST'],
                        port=db_conf['PORT']
                ) as conn:
                    pass

                request.session['user_id'] = user.clientid
                request.session['db_role'] = user.role
                request.session['db_user'] = db_user

                return redirect('dashboard')

            except psycopg.OperationalError:
                error = 'Incorrect password'

        except Client.DoesNotExist:
            error = 'User not found'
        except Exception as e:
            error = f'System error: {e}'

    return render(request, 'login.html', {'error': error})
```

Лістинг 8.1 — Функція custom_login

Ця функція обробляє процес входу користувача, таких як email та пароль. Логіка її роботи становить:

1) Отримання даних: Приймає email та пароль з POST-запиту
2) Пошук користувача: Шукає клієнта в таблиці Client за email
3) Формування імені користувача БД: Створює динамічне ім'я користувача PostgreSQL у форматі user_{clientid}
4) Перевірка пароля через підключення до БД:
5) Якщо підключення успішне — пароль правильний

Збереження сесії: Сесії це вбудований інструментарій Django, який дозволяє зберігати деякі дані під час сесії користувача на веб-сервісі. При успішній автентифікації зберігає в сесії такі поля як, user_id (ID клієнта), db_role (роль користувача), db_user (ім'я користувача БД).

Продовжує функціонал авторизації функція виходу з облікового запису:

```
def logout_view(request):
    with connection.cursor() as cursor:
        cursor.execute("RESET ROLE")
    request.session.flush()
    return redirect('login')
```

Лістинг 8.2 — функція logout_view

Логіка виконання цієї функції, це виклик курсору БД, та виконання функції зкиду користувача до connect_user, очищення сесії Django та перенаправлення на сторінку автентифікації.

9 ІНСТРУКЦІЯ КОРИСТУВАЧА З ІЛЮСТРАЦІЯМИ

Мета даного розділу показати, як розроблений веб-застосунок виконує описані задачі користувачів

9.1 Гість

При переході на посилання веб-застосунку, буде завантажена сторінка авторизації:

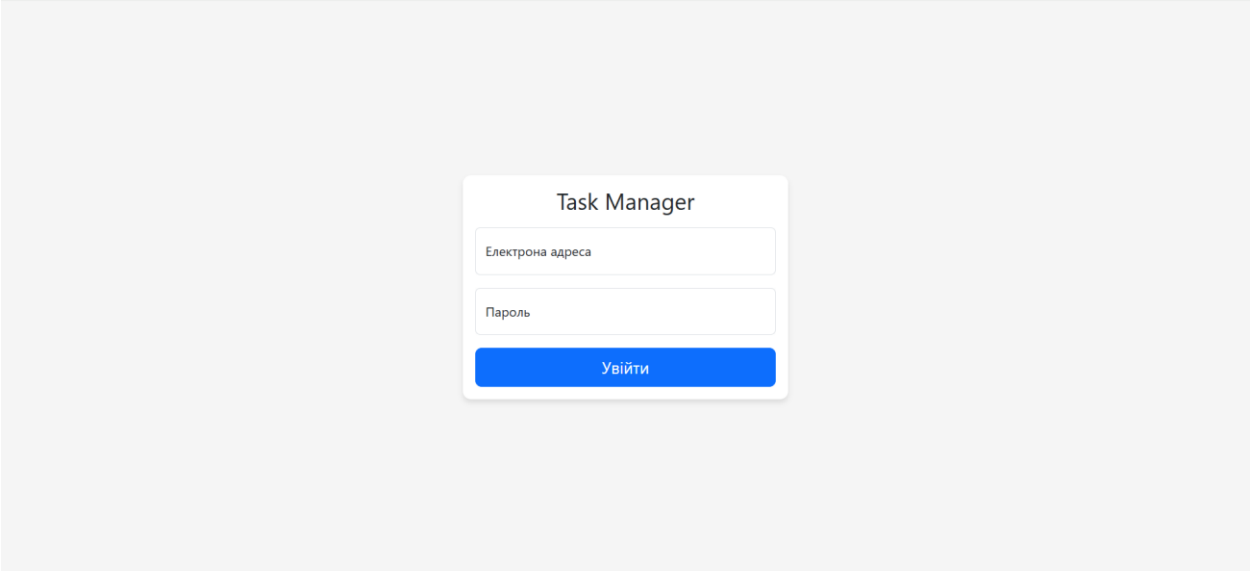

Рисунок 9.1 — Сторінка авторизації

Після успішної авторизації користувач, залежно від його ролі перейде на робочий простір веб-застосунку.

9.2 Рядовий розробник

Перед користувачем із такою ролю відкриється його власна версія головної сторінки, на якій в нього є можливість виконати розподілене на нього підзавдання, отримати інформації з приводу заходів у команді та отримати повідомлення, чий ого надіслати:

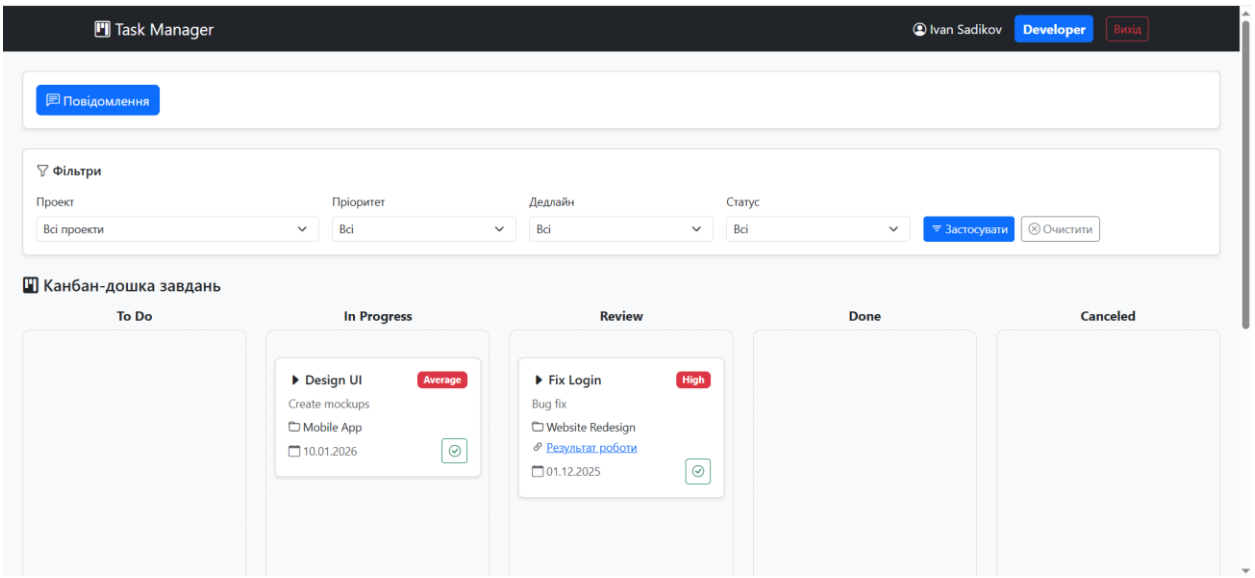

Рисунок 9.2 — Екран користувача з ролю «Рядовий розробник», частина 1

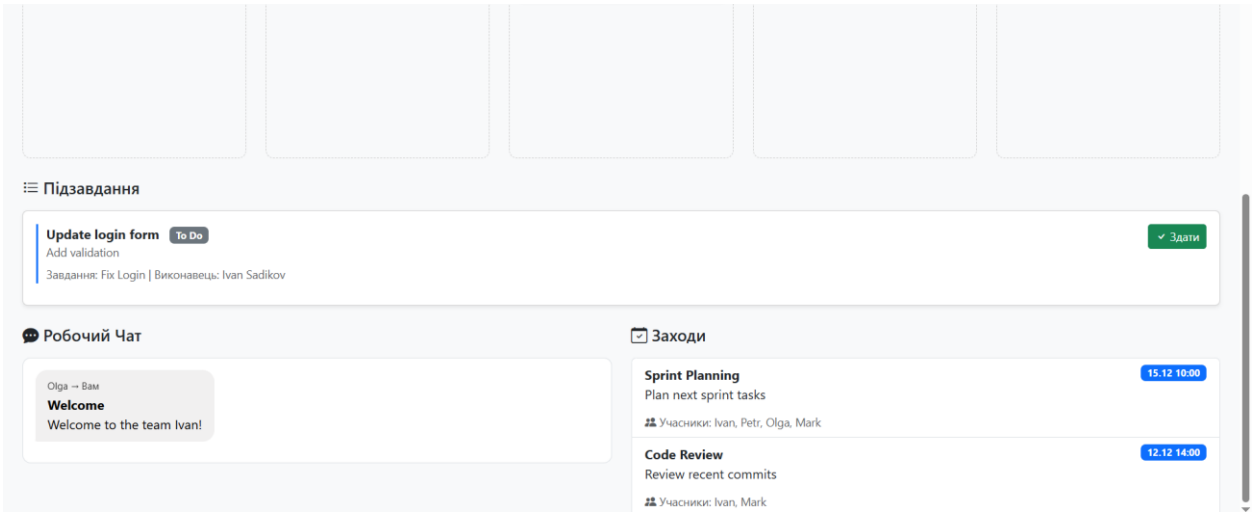

Рисунок 9.3 — Екран користувача з ролю «Рядовий розробник», частина 2

У кожного користувача, є такі можливості, та фільтрації завдань на Kanban дошці:

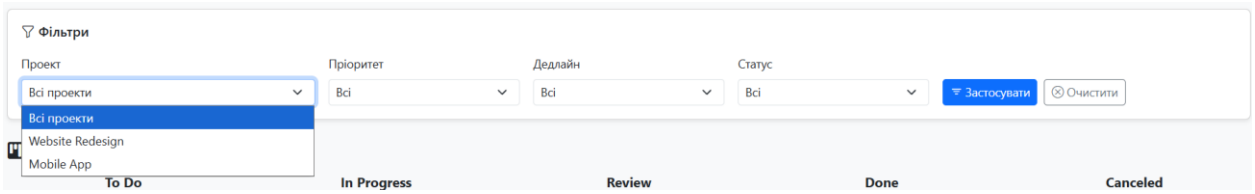

Рисунок 9.4 — панель з фільтрації завдань, фільтри за проєктом

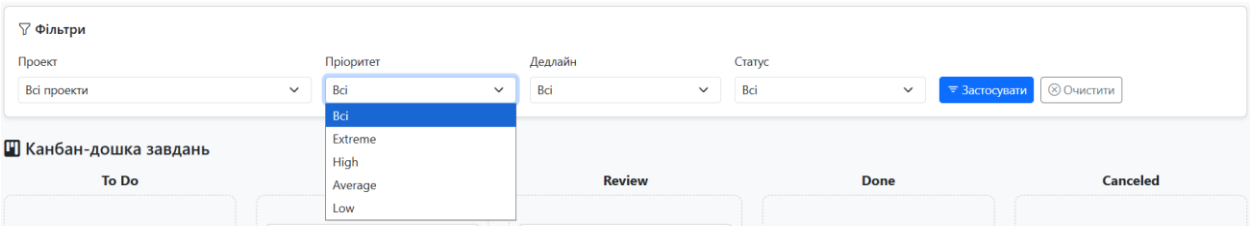

Рисунок 9.5 — панель з фільтрації завдань, фільтри за пріоритетом

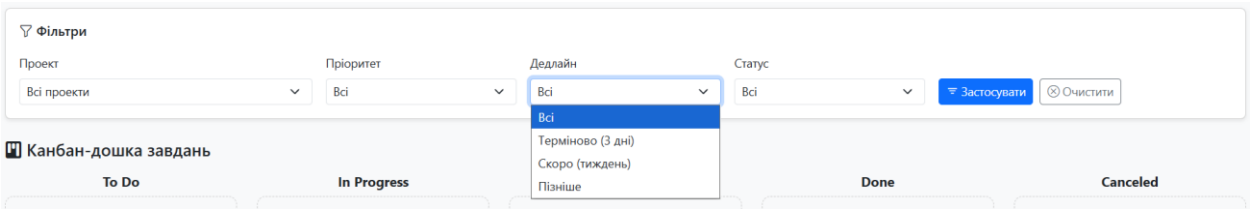

Рисунок 9.6 — панель з фільтрації завдань, фільтри за дедлайном

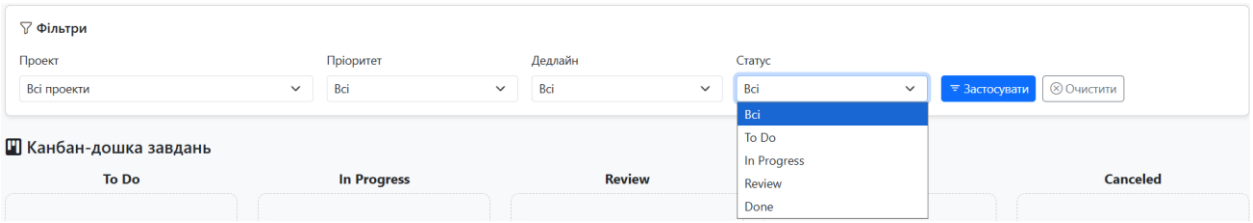

Рисунок 9.7 — панель з фільтрації завдань, фільтри за статусом завдання

Наступним елементом сторінки є Kanban дошка, на ній відображаються завдання на різних стадіях розробки, у користувача з ролю «Рядовий розробник» єдина можливість взаємодії це виконання шляхом додавання посилання результата роботи, алгоритм якого представлений нижче:

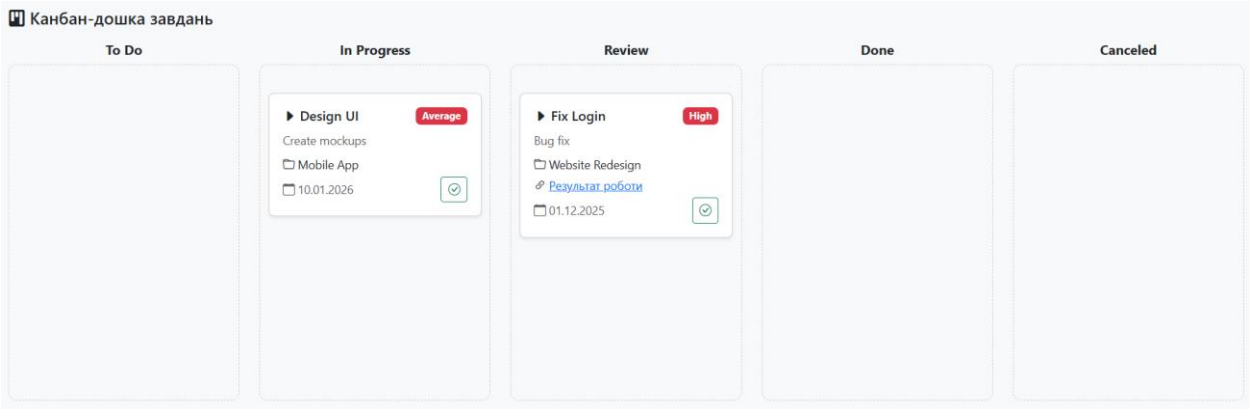

Рисунок 9.8 — Дошка Kanban із завданнями

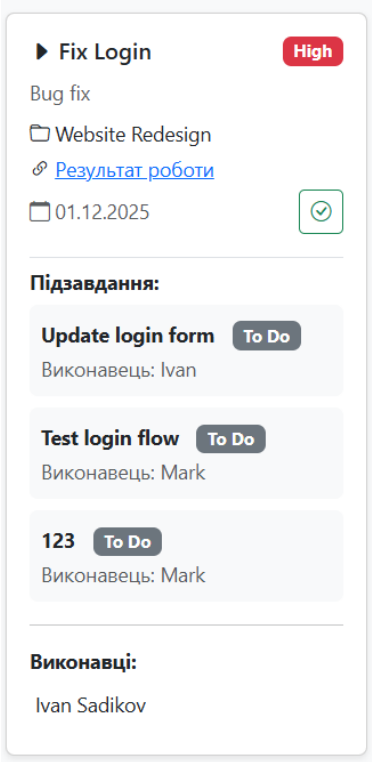

Рисунок 9.9 — Розгорнуте завданя

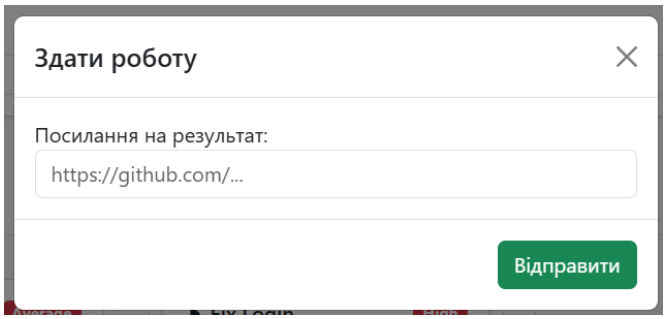

Рисунок 9.10 — Модальне вікно для надсилання посилання з результатами виконання завдання

Також у користувача є можливість перегляду та виконання підзавдань, шляхо перегляду у відведеному для цього розділі:

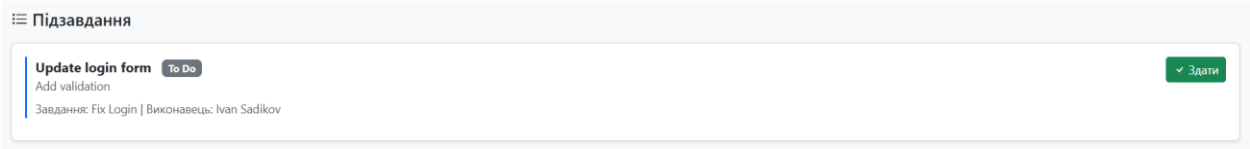

Рисунок 9.11 — Розділ підзавдань

Також у користувача ролі «Рядовий розробник» є можливість контактувати з іншими членами конманди:

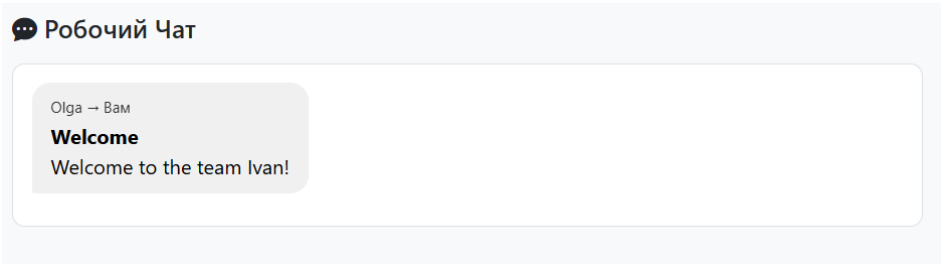

Рисунок 9.12 — Вигляд робочого чату

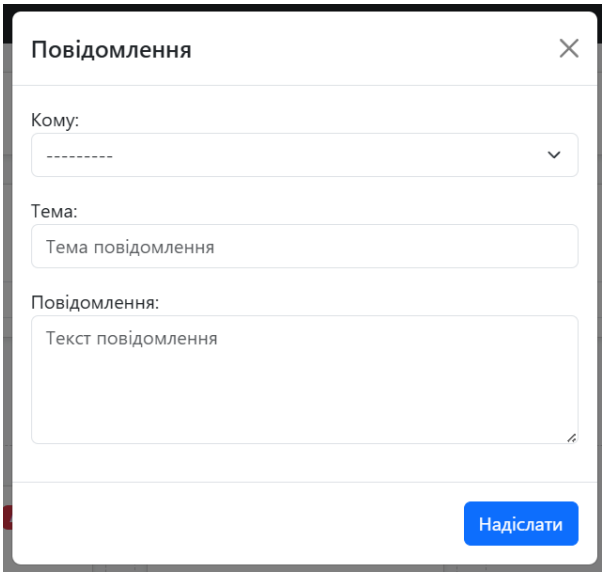

Рисунок 9.13 — Модальне вікно надсилання повідомлення

Також коривач може реглядати назначені події для команди:

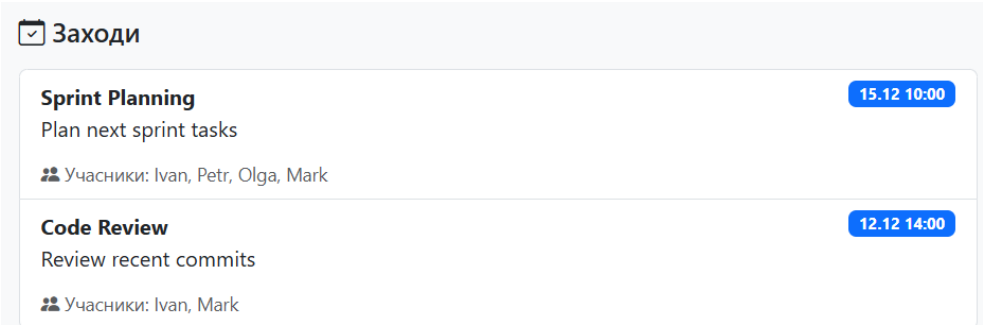

Рисунок 9.14 — Заплановані командні заходи

9.3 Менеджер

Загальна задача користувача з ролю Менеджер це координація розробників, перевірка виконаних завдань та комунікація із ними.

Після успішної авторизації, перед Менеджером з’явиться його персоналізоване робочий простір:

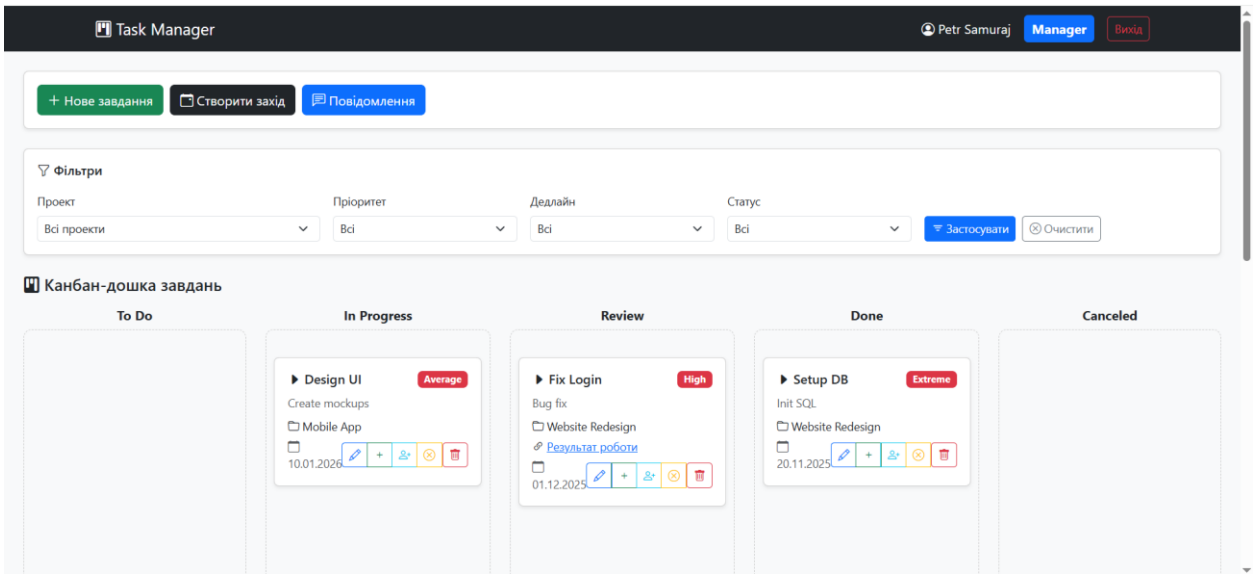

Рисунок 9.15 — Екран користувача з ролю «Менеджер»

Панель дій та завдань змінились, переглянемо новий функціонал:

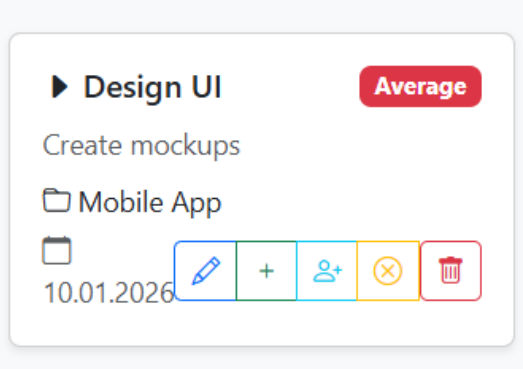

Рисунок 9.16 — Завдання із кнопками взаємодії

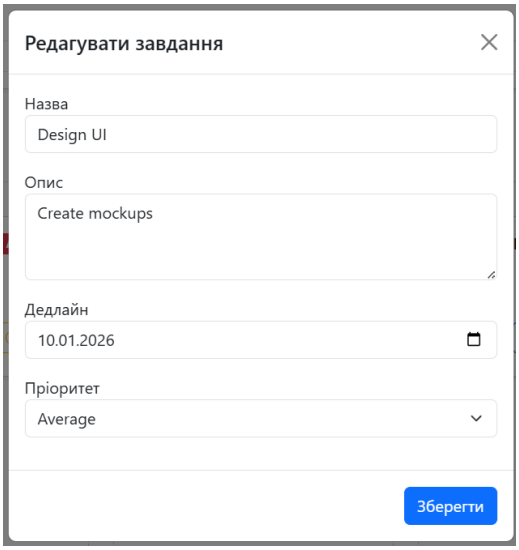

Рисунок 9.17 — Модальне вікно редагування завдання

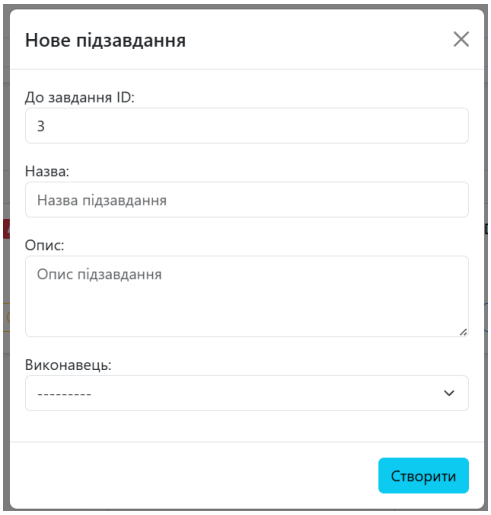

Рисунок 9.18 — Модальне вікно створення нового завдання

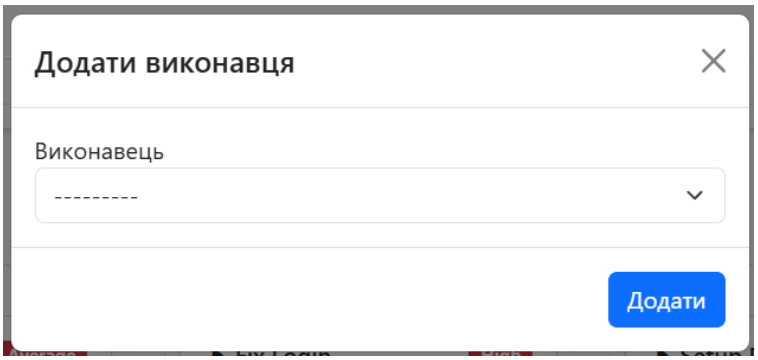

Рисунок 9.19 — Модальне вікно додавання виконавця

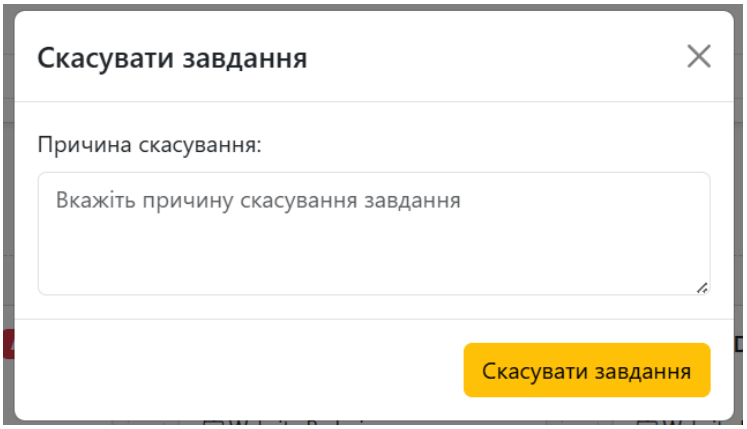

Рисунок 9.20 — Модальне вікно відміни завдання

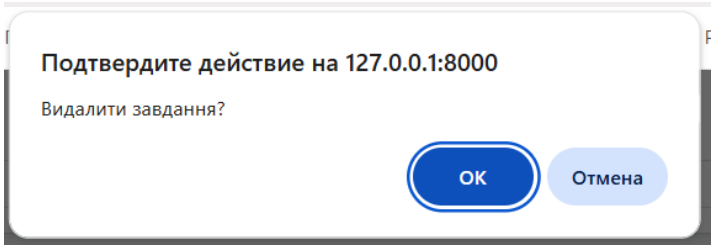

Рисунок 9.21 — Попередження про видалення завдання

Весь інший функціонал копіює загальні можливості будь якого члена команди.

9.4 Головний розробник

Загальна задача користувача із ролю «Голова розробки» це регуляція роботи менеджерів, перевірка темпу розробки проетку, аналізу

продуктивності членів команди та проекту загалом, створення заходів і комунікування з командою.

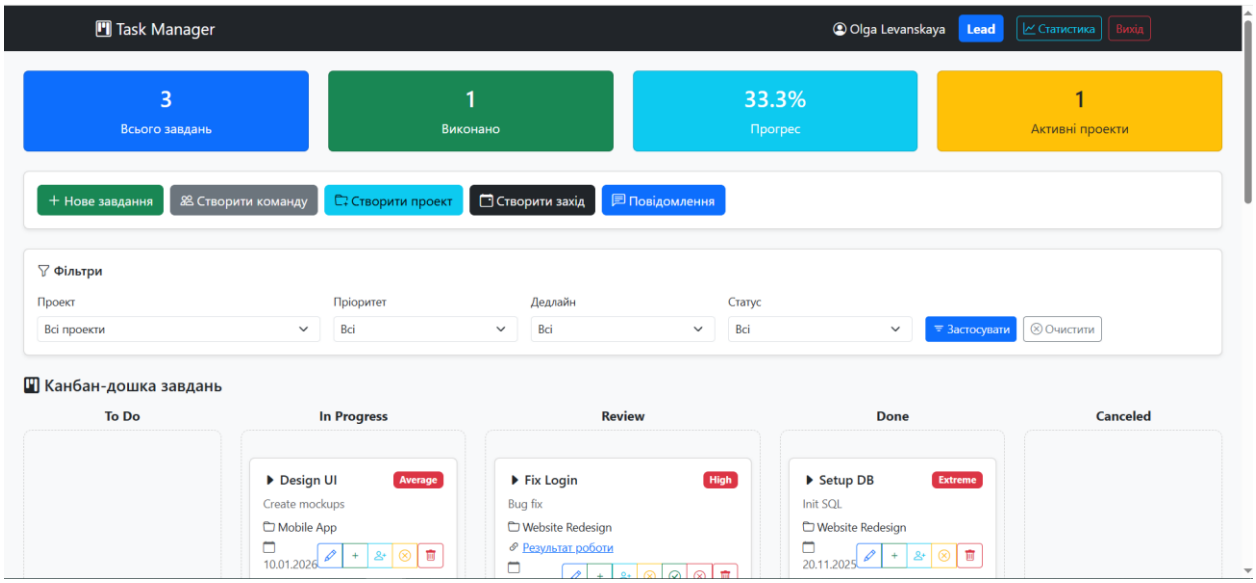

Рисунок 9.22 — Екран користувача з ролю «Голова розробки»

Серед функціоналу, який предоставляє сервіс для «Голови розробки», є створення команди, проекту, вилучення учасників зі складу команд чи проектів, перегляд списку команди та аналіз роботи учасників:

Рисунок 9.23 — Модальне вікно створення команди

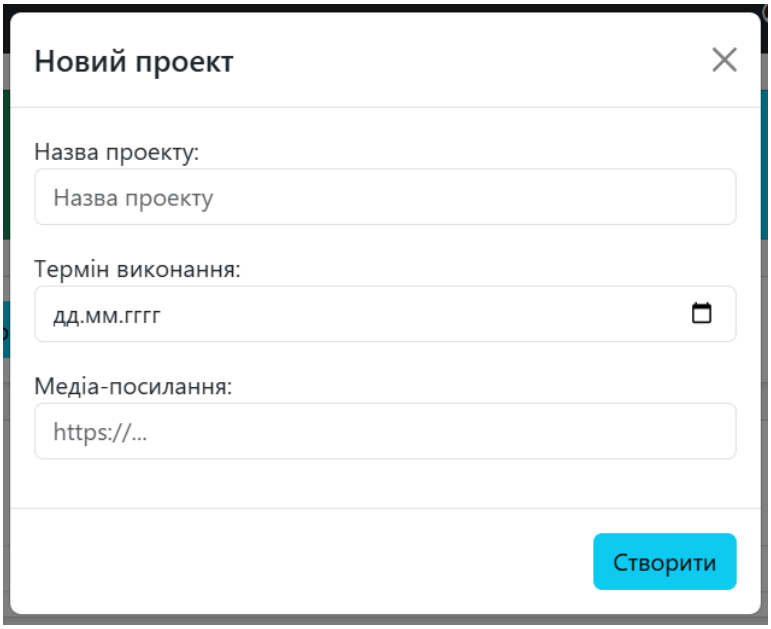

Рисунок 9.24 — Модальне вікно створення проекту

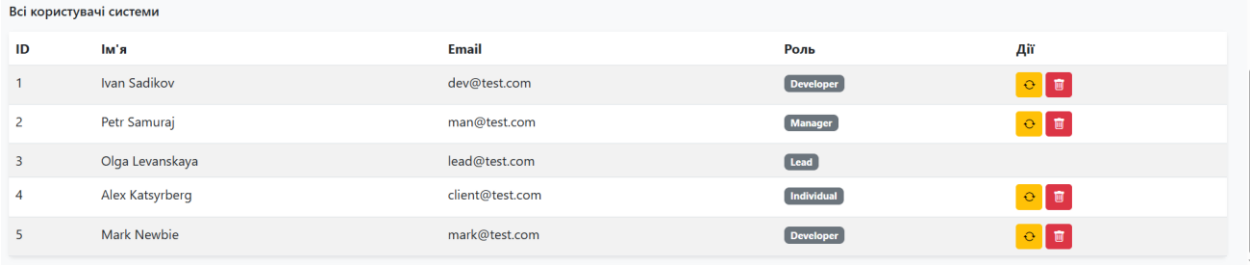

Рисунок 9.25 — Розділ із учасниками команди

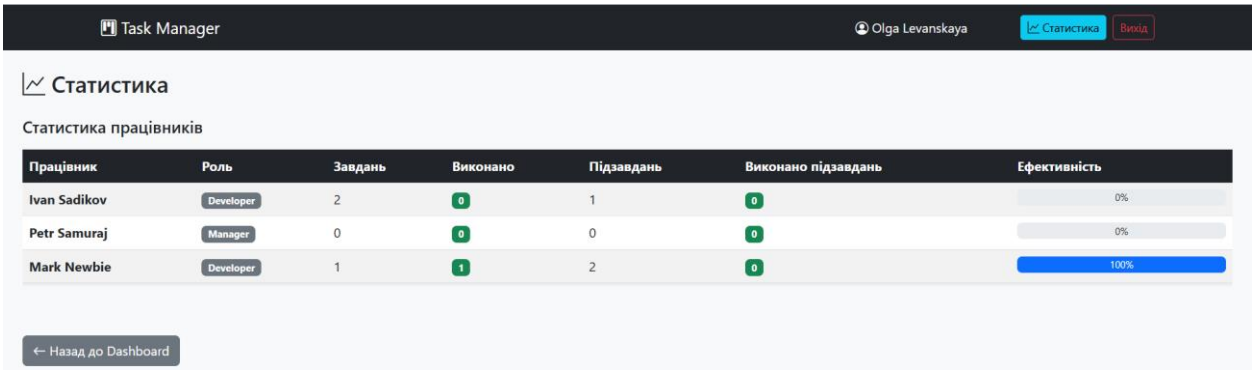

Рисунок 9.26 — Розділ зі статистикою учасників команди

Рисунок 9.27 — Розділ із аналітикою стану проекту

Також слід додати, що «Голова розробки» наслідує функціонал Менеджера.

9.5 Клієнт

Загальна задача користувача з ролю «Клієнту» це слідкування за правильністю розуміння вимог замовника, в нього є функціонал ролі «Менеджера», проте із аналітичними додатками від «Голови розробки».

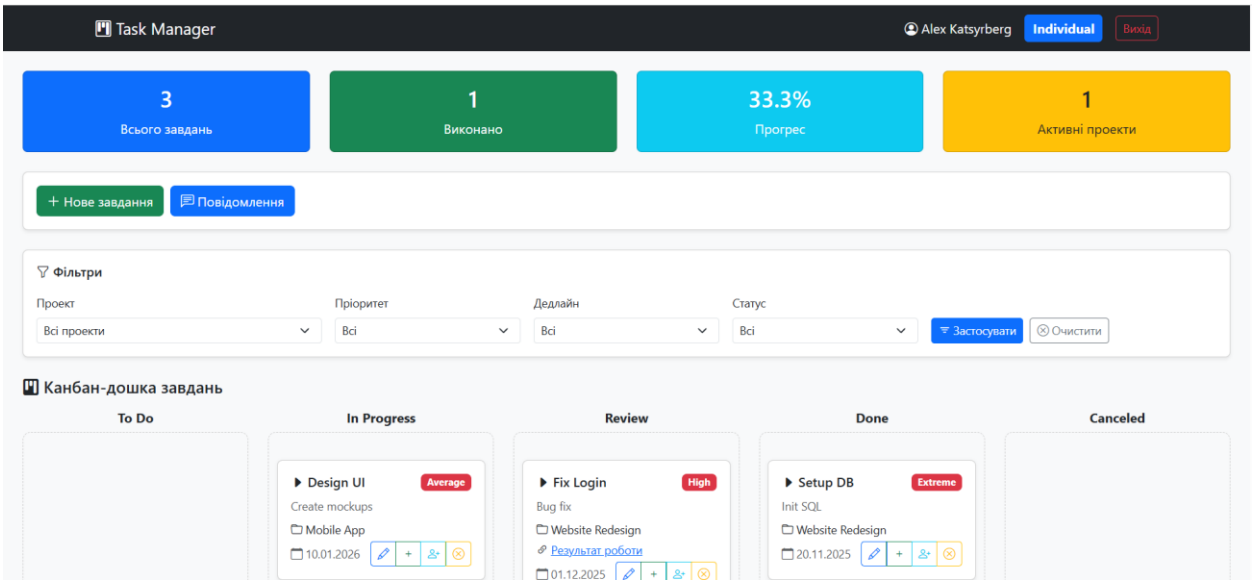

Рисунок 9.28 — Екран користувача з ролю «Клієнт»

ВИСНОВКИ

Виконання поставленої задачі, реалізації інформаційної системи для організації роботи команди розробників, вимагало урахування основних потреб обраної області. Задля цього визначені загальні вимоги та завдання, які повинна виконувати система. Цілю створення системи є надання інструментарію комунікації між членами команди, організації робочого процесу та аудит проведеної роботи.

На основі отриманих ролей користувачів, та їх задач, спроектовано схему інформаційної системи. Вона включала у себе необхідні сутності предметної області: користувач, проект, команда, завдання, підзавдання, захід, повідомлення. Для визначенних сутностей виділені характеризу.чи їх атрибути, та зв'язки між таблицями. Усі зв'язки були формалізовані, наскільки дозволял обрана область.

Реалізація серверу баз данних відбулась на системі керування баз данних PostgreSQL, її відкритість та длступність полегшила реалізоцію необхідного функціоналу. Клієнтський додаток створенний на базі фреймворку Django із додавання шаблонізатору Jinja2 та стильової біліотеки Bootstrap для уніфікації користувацього досвіду та його рівномірності. Django дозволив реалізувати API яке завдяки бібліотеки psycopg2 реалізувало пряме підключення до серверу бази данних, що дозволило використовувати повний функціонал PostgreSQL. Вміст сторінок та інформації і прав доступних користувачу, обмежений відповідно до його ролі.

Такий підхід дозволив реалізувати надійний веб-сервіс, який відповідає вимогам мінімально життєздатного продукту. Застосунок дозволяє створювати команди та проекти, додавати користувачів із різними ролями, створення завдань та під-завдань, створення заходів, надсилання повідомлень. Система має потенціал на буття корисним інструментом у процесі роботи невеличких груп людей та команд. Можливим розширенням є додавання детальнішої статистики проекту, інтеграція з Git репозиторіями, тощо.

ПЕРЕЛІК ПОСИЛАНЬ

1. Малахов Є. В., Розновець О. І. Проектування інформаційних систем. Методичні вказівки до курсового проектування — Одеса : Олді+, 2023.

2. Малахов Є. В. Бази даних. Конспект лекцій — Одеса : Кафедра інженерії програмного забезпечення, 2025.

3. Django Software Foundation. Django documentation [Електронний ресурс] — Django Project, 2025. — Режим доступу: https://docs.djangoproject.com/en/6.0/.

4. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс] — PostgreSQL Global Development Group, 2025. — Режим доступу: https://www.postgresql.org/docs/.

5. Bootstrap Team. Bootstrap 5.3 Documentation [Електронний ресурс] — Bootstrap, 2025. — Режим доступу: https://getbootstrap.com/docs/5.3/.

Додаток А

Схема бази данних

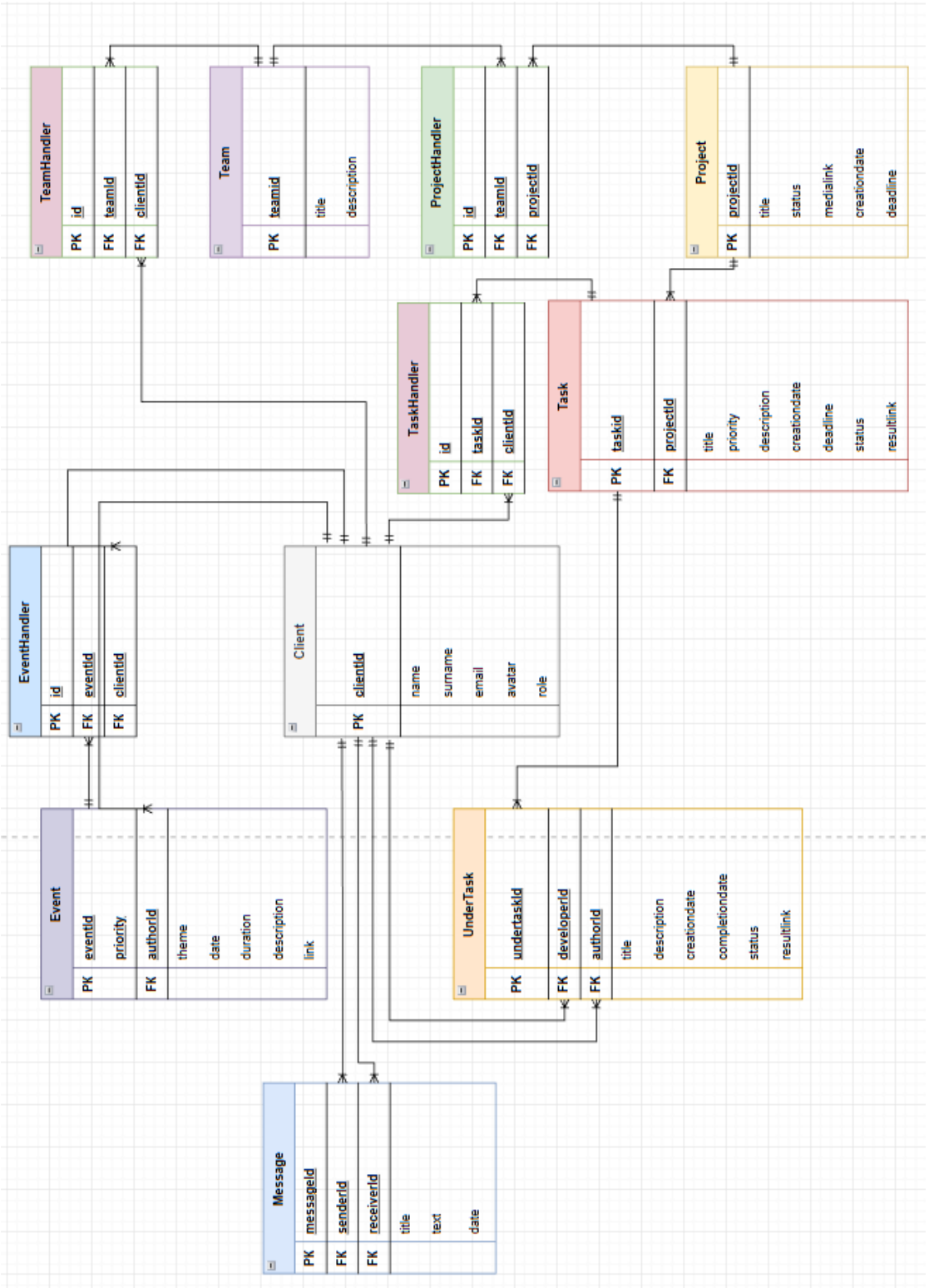

Рисунок 3.1 — ER-діаграма для предметної області «Організація роботи команди розробників»

Додаток Б

Ієрархія сторінок

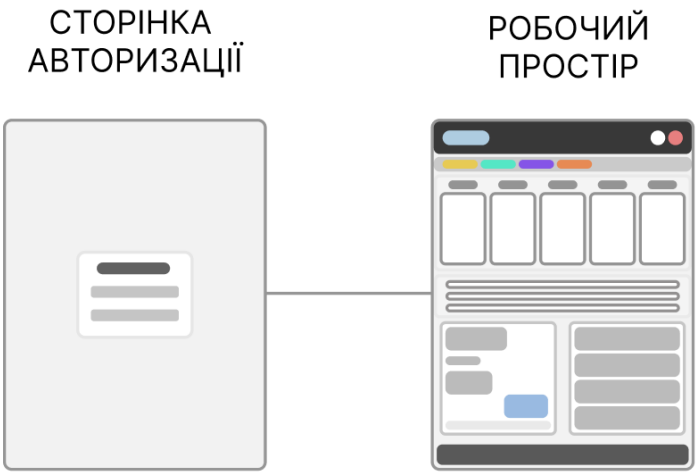

Рисунок 5.1 — Ієрархія сторінок веб-додатку ‘Task-manager’

Діаграма рольового доступу

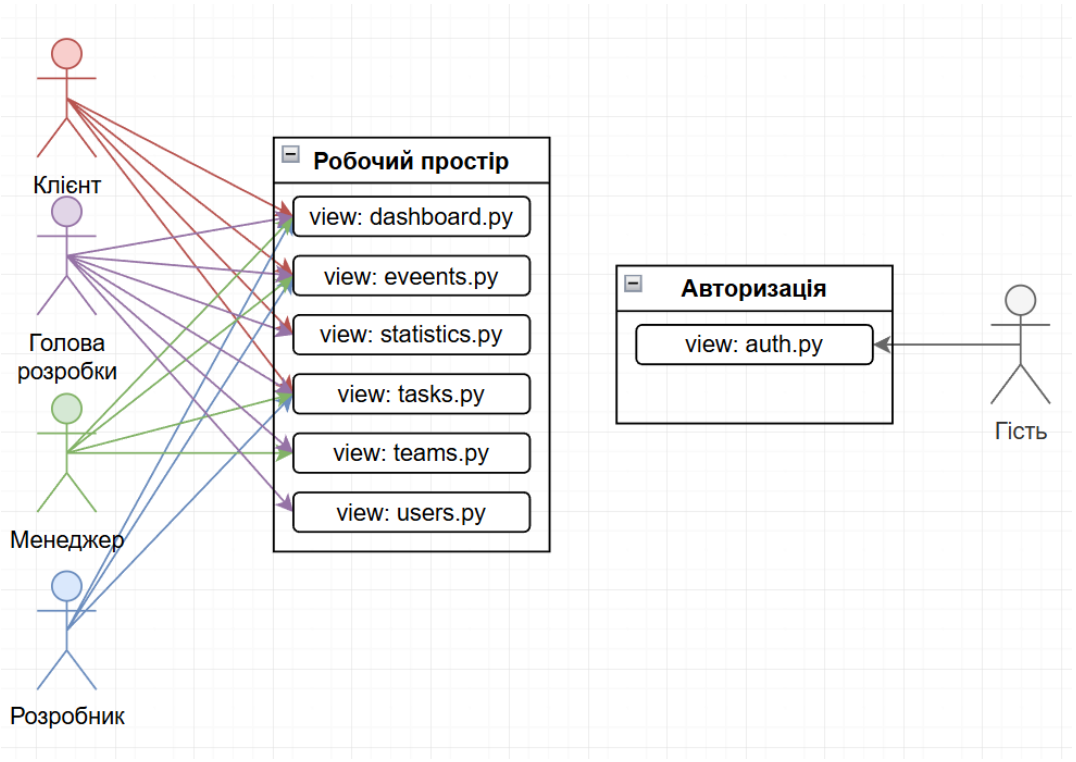

Рисунок 5.2 — Діаграма рольового доступу для веб-додатку ‘Task-Manager’

Додаток В

ЗАПИТИ ДЛЯ СТВОРЕННЯ БАЗИ ДАННИХ

Домени

```
/* DOMAIN: PRIORITY */
     CREATE DOMAIN Priority CHAR(7)
          CONSTRAINT ValitPriority
          CHECK(VALUE IN('Extreme','High','Average','Low'));

/* DOMAIN: STATUS */
     CREATE DOMAIN Status CHAR(9)
          CONSTRAINT ValitStatus
          CHECK(VALUE
IN('Active','Freezed','Completed','Failed','Canceled'));
```

Таблиці:

```
CREATE TABLE django_session (
    session_key varchar(40) NOT NULL PRIMARY KEY,
    session_data text NOT NULL,
    expire_date timestamp with time zone NOT NULL
);
CREATE  INDEX  django_session_expire_date_idx  ON  django_session
(expire_date);

CREATE TABLE Client (
    ClientId BIGSERIAL PRIMARY KEY,
    Name VARCHAR(30) NOT NULL,
    Surname VARCHAR(30) NOT NULL,
    Email VARCHAR(64) UNIQUE NOT NULL,
    Avatar VARCHAR(2048) DEFAULT 'idle.png' NOT NULL,
    Role VARCHAR(30) CHECK(Role IN ('No Commerce', 'Developer',
'Manager', 'Lead', 'Individual')) NOT NULL
);

CREATE TABLE Team (
    TeamId BIGSERIAL PRIMARY KEY,
    Title VARCHAR(50) NOT NULL,
    Description VARCHAR(255) NOT NULL
```

```
);

CREATE TABLE TeamHandler (
    id BIGSERIAL PRIMARY KEY,
    TeamId BIGINT REFERENCES Team(TeamId) ON DELETE CASCADE,
    ClientId BIGINT REFERENCES Client(ClientId) ON DELETE CASCADE,
    CONSTRAINT teamhandler_unique UNIQUE (TeamId, ClientId)
);

CREATE TABLE Project (
    ProjectId BIGSERIAL PRIMARY KEY,
    Title VARCHAR(50) NOT NULL,
    Status StatusDomain NOT NULL DEFAULT 'To Do',
    MediaLink VARCHAR(2048),
    CreationDate DATE NOT NULL DEFAULT CURRENT_DATE,
    Deadline DATE NOT NULL
);

CREATE TABLE ProjectHandler (
    id BIGSERIAL PRIMARY KEY,
    TeamId BIGINT REFERENCES Team(TeamId) ON DELETE CASCADE,
    ProjectId BIGINT REFERENCES Project(ProjectId) ON DELETE
CASCADE,
    CONSTRAINT projecthandler_unique UNIQUE (TeamId, ProjectId)
);

CREATE TABLE Task (
    TaskId BIGSERIAL PRIMARY KEY,
    ProjectId BIGINT REFERENCES Project(ProjectId) ON DELETE
CASCADE,
    Title VARCHAR(100) NOT NULL,
    Priority PriorityDomain NOT NULL,
    Description VARCHAR(1500) NOT NULL,
    CreationDate DATE NOT NULL DEFAULT CURRENT_DATE,
    Deadline DATE NOT NULL,
```

```
    Status StatusDomain NOT NULL DEFAULT 'To Do',
    ResultLink VARCHAR(2048)
);

CREATE TABLE TaskHandler (
    id BIGSERIAL PRIMARY KEY,
    TaskId BIGINT REFERENCES Task(TaskId) ON DELETE CASCADE,
    ClientId BIGINT REFERENCES Client(ClientId) ON DELETE CASCADE,
    CONSTRAINT taskhandler_unique UNIQUE (TaskId, ClientId)
);

CREATE TABLE UnderTask (
    UnderTaskId BIGSERIAL PRIMARY KEY,
    TaskId BIGINT REFERENCES Task(TaskId) ON DELETE CASCADE,
    DeveloperId BIGINT REFERENCES Client(ClientId) ON DELETE SET
NULL,
    AuthorId BIGINT REFERENCES Client(ClientId) ON DELETE SET
NULL,
    Title VARCHAR(100) NOT NULL,
    Description VARCHAR(1500) NOT NULL,
    CreationDate DATE NOT NULL DEFAULT CURRENT_DATE,
    CompletionDate DATE,
    Status StatusDomain NOT NULL DEFAULT 'To Do',
    ResultLink VARCHAR(2048)
);

CREATE TABLE Message (
    MessageId BIGSERIAL PRIMARY KEY,
    SenderId BIGINT REFERENCES Client(ClientId) ON DELETE CASCADE,
    ReceiverId BIGINT REFERENCES Client(ClientId) ON DELETE
CASCADE,
    Title VARCHAR(100) NOT NULL,
    Text VARCHAR(1500) NOT NULL,
    Date TIMESTAMP NOT NULL DEFAULT NOW()
);
```

```
CREATE TABLE Event (
    EventId BIGSERIAL PRIMARY KEY,
    Priority PriorityDomain NOT NULL,
    AuthorId BIGINT REFERENCES Client(ClientId) ON DELETE CASCADE,
    Theme VARCHAR(100) NOT NULL,
    Description VARCHAR(1500) NOT NULL,
    Date TIMESTAMP NOT NULL,
    Duration INTERVAL NOT NULL,
    Link VARCHAR(2048)
);

CREATE TABLE EventHandler (
    id BIGSERIAL PRIMARY KEY,
    EventId BIGINT REFERENCES Event(EventId) ON DELETE CASCADE,
    ClientId BIGINT REFERENCES Client(ClientId) ON DELETE CASCADE,
    CONSTRAINT eventhandler_unique UNIQUE (EventId, ClientId)
);

CREATE TABLE MediaLink (
    MediaLinkId BIGSERIAL PRIMARY KEY,
    Url VARCHAR(2048) NOT NULL,
    Description VARCHAR(255),
    TaskId BIGINT REFERENCES Task(TaskId) ON DELETE CASCADE,
    UnderTaskId BIGINT REFERENCES UnderTask(UnderTaskId) ON DELETE
CASCADE,
    MessageId BIGINT REFERENCES Message(MessageId) ON DELETE
CASCADE,
    EventId BIGINT REFERENCES Event(EventId) ON DELETE CASCADE
);

CREATE ROLE "connect_user" WITH LOGIN PASSWORD 'connect_pass';
ALTER ROLE "connect_user" CREATEROLE;

CREATE ROLE "Individual" NOLOGIN;
```

```
CREATE ROLE "Developer" NOLOGIN;
CREATE ROLE "Manager" NOLOGIN;
CREATE ROLE "Lead" NOLOGIN;

GRANT "Individual", "Developer", "Manager", "Lead" TO
"connect_user" WITH ADMIN OPTION;

GRANT ALL ON SCHEMA public TO "connect_user";
GRANT ALL ON ALL TABLES IN SCHEMA public TO "connect_user";
GRANT ALL ON ALL SEQUENCES IN SCHEMA public TO "connect_user";

GRANT ALL ON ALL TABLES IN SCHEMA public TO "Individual",
"Developer", "Manager", "Lead";
GRANT ALL ON ALL SEQUENCES IN SCHEMA public TO "Individual",
"Developer", "Manager", "Lead";

INSERT INTO Client (Name, Surname, Email, Role) VALUES
('Ivan', 'Sadikov', 'dev@test.com', 'Developer'),
('Petr', 'Samuraj', 'man@test.com', 'Manager'),
('Olga', 'Levanskaya', 'lead@test.com', 'Lead'),
('Alex', 'Katsyrberg', 'client@test.com', 'Individual'),
('Mark', 'Newbie', 'mark@test.com', 'Developer');

CREATE ROLE "user_1" WITH LOGIN PASSWORD '123' IN ROLE "Developer";
GRANT "user_1" TO "connect_user";

CREATE ROLE "user_2" WITH LOGIN PASSWORD '123' IN ROLE "Manager";
GRANT "user_2" TO "connect_user";

CREATE ROLE "user_3" WITH LOGIN PASSWORD '123' IN ROLE "Lead";
GRANT "user_3" TO "connect_user";

CREATE ROLE "user_4" WITH LOGIN PASSWORD '123' IN ROLE
"Individual";
GRANT "user_4" TO "connect_user";
```

```
CREATE ROLE "user_5" WITH LOGIN PASSWORD '123' IN ROLE "Developer";
GRANT "user_5" TO "connect_user";

INSERT INTO Project (Title, Status, Deadline) VALUES
('Website Redesign', 'In Progress', '2025-12-31'),
('Mobile App', 'To Do', '2026-03-15');

INSERT INTO Task (ProjectId, Title, Priority, Description,
Deadline, Status) VALUES
(1, 'Fix Login', 'High', 'Bug fix', '2025-12-01', 'To Do'),
(1, 'Setup DB', 'Extreme', 'Init SQL', '2025-11-20', 'Done'),
(2, 'Design UI', 'Average', 'Create mockups', '2026-01-10', 'In
Progress');

INSERT INTO TaskHandler (TaskId, ClientId) VALUES
(1, 1),
(2, 5),
(3, 1);

INSERT INTO UnderTask (TaskId, DeveloperId, AuthorId, Title,
Description, Status) VALUES
(1, 1, 2, 'Update login form', 'Add validation', 'To Do'),
(1, 5, 2, 'Test login flow', 'Write unit tests', 'To Do');

INSERT INTO Message (SenderId, ReceiverId, Title, Text) VALUES
(3, 1, 'Welcome', 'Welcome to the team Ivan!'),
(2, 5, 'Task assigned', 'Please check your new task');

INSERT INTO Event (Priority, AuthorId, Theme, Description, Date,
Duration) VALUES
('High', 3, 'Sprint Planning', 'Plan next sprint tasks', '2025-
12-15 10:00:00', '02:00:00'),
('Average', 2, 'Code Review', 'Review recent commits', '2025-12-
12 14:00:00', '01:00:00');
```

```
INSERT INTO EventHandler (EventId, ClientId) VALUES
(1, 1), (1, 2), (1, 3), (1, 5),
(2, 1), (2, 5);
```

Представлення

```
/* Leads email */
CREATE VIEW LeadEmails AS
SELECT Email
FROM Client
WHERE Role = 'Lead';

/*No commerce emails*/
CREATE VIEW ManagerEmails AS
SELECT Email
FROM Client
WHERE Role = 'Manager';

/*Client emails*/
CREATE VIEW ManagerEmails AS
SELECT Email
FROM Client
WHERE Role = 'Individual';

/*Users rating*/
CREATE OR REPLACE VIEW client_ranking_view AS
SELECT
    client.clientid,
    client.name,
    client.surname,

    RANK() OVER (ORDER BY COUNT(DISTINCT teamhandler.teamid)
DESC) AS team_rank,
    RANK() OVER (ORDER BY COUNT(DISTINCT
projecthandler.projectid) DESC) AS project_rank,
    RANK() OVER (ORDER BY COUNT(DISTINCT undertask.undertaskid)
DESC) AS undertask_rank,
    RANK() OVER (ORDER BY COUNT(DISTINCT event.eventid) DESC) AS
events_rank

FROM client
LEFT JOIN teamhandler ON client.clientid = teamhandler.clientid
LEFT JOIN projecthandler ON projecthandler.teamid =
teamhandler.teamid
```

```
LEFT JOIN undertask ON client.clientid = undertask.developerid
LEFT JOIN event ON client.clientid = event.authorid

GROUP BY client.clientid, client.name, client.surname;

DROP VIEW client_ranking_view;
SELECT * FROM client_ranking_view;


/*Project structure*/
CREATE OR REPLACE VIEW project_structure_hierarchy AS
WITH RECURSIVE project_structure(projectid, name, level,
project_title, path) AS (
    SELECT
        p.projectid,
        p.title::TEXT AS name,
        1 AS level,
        p.title::TEXT AS project_title,
        p.projectid::TEXT AS path
    FROM project p

    UNION ALL

    SELECT
        child.projectid,
        child.name,
        child.level,
        parent.project_title,
        parent.path || '-' || child.id::TEXT AS path
    FROM project_structure parent
    JOIN (
        SELECT
            t.projectid,
            t.taskid AS id,
            ('-- ' || t.title) AS name,
            2 AS level,
            t.taskid AS parent_id
        FROM task t

        UNION ALL

        SELECT
            t.projectid,
            st.undertaskid AS id,
            ('---- ' || st.title) AS name,
            3 AS level,
            st.taskid AS parent_id
        FROM undertask st
```

```
        JOIN task t ON st.taskid = t.taskid
    ) AS child
    ON (parent.level = 1 AND child.level = 2 AND child.projectid
= parent.projectid)
       OR (parent.level = 2 AND child.level = 3 AND
child.parent_id::TEXT = SPLIT_PART(parent.path, '-',
array_length(string_to_array(parent.path, '-'), 1)))
)
```

Тригери:

```
-- Complete

CREATE OR REPLACE FUNCTION CompletingTaskTriggerExtended()
RETURNS TRIGGER AS $$
DECLARE
    media_count INTEGER;
BEGIN
    IF NEW.Status = 'Done' AND OLD.Status != 'Done' THEN

        SELECT COUNT(*) INTO media_count
        FROM MediaLink
        WHERE TaskId = NEW.TaskId;

        IF media_count = 0 AND (NEW.ResultLink IS NULL OR
NEW.ResultLink = '') THEN
            RAISE EXCEPTION 'At least one media link or
ResultLink is required when completing task %', NEW.TaskId;
        END IF;

        IF NEW.ResultLink IS NOT NULL AND NEW.ResultLink != ''
THEN
            IF NOT EXISTS (
                SELECT 1 FROM MediaLink
                WHERE TaskId = NEW.TaskId AND Url =
NEW.ResultLink
            ) THEN
                INSERT INTO MediaLink (Url, Description, TaskId)
                VALUES (
                    NEW.ResultLink,
                    'Task completion result: ' || NEW.Title,
                    NEW.TaskId
                );
            END IF;
        END IF;
```

```
        RAISE NOTICE 'Task % (%) completed. Total media links:
%',
            NEW.TaskId, NEW.Title, media_count + 1;

    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS completing_task_trigger ON Task;

CREATE TRIGGER completing_task_trigger
AFTER UPDATE ON Task
FOR EACH ROW
WHEN (NEW.Status = 'Done' AND OLD.Status IS DISTINCT FROM
'Done')
EXECUTE FUNCTION CompletingTaskTrigger();
```

Хранимі функції:

```
-- client productivity

CREATE OR REPLACE FUNCTION GetClientTasksStatistics(p_ClientId
BIGINT DEFAULT NULL)
RETURNS TABLE (
    ClientId BIGINT,
    ClientName VARCHAR(30),
    ClientSurname VARCHAR(30),
    CompletedTasks BIGINT,
    FailedTasks BIGINT
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        c.ClientId,
        c.Name,
        c.Surname,
        COUNT(CASE
            WHEN t.Status = 'Done' AND t.Deadline >=
CURRENT_DATE
            THEN 1
        END) AS CompletedTasks,
        COUNT(CASE
            WHEN t.Status IN ('To Do', 'In Progress') AND
t.Deadline < CURRENT_DATE
            THEN 1
```

```
        END) AS FailedTasks
    FROM
        Client c
    LEFT JOIN
        TaskHandler th ON c.ClientId = th.ClientId
    LEFT JOIN
        Task t ON th.TaskId = t.TaskId
    WHERE
        p_ClientId IS NULL OR c.ClientId = p_ClientId
    GROUP BY
        c.ClientId, c.Name, c.Surname
    ORDER BY
        c.ClientId;
END;
$$ LANGUAGE plpgsql;


-- manager productivity


CREATE OR REPLACE FUNCTION
GetManagerStatisticsDetailed(p_ManagerId BIGINT DEFAULT NULL)
RETURNS TABLE (
    ManagerId BIGINT,
    ManagerName VARCHAR(30),
    ManagerSurname VARCHAR(30),
    ManagerEmail VARCHAR(64),
    CreatedUnderTasks BIGINT,
    AssignedTasks BIGINT,
    TasksInProgress BIGINT,
    TasksToDo BIGINT,
    CancelledTasks BIGINT,
    CompletedTasks BIGINT,
    OverdueTasks BIGINT
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        c.ClientId AS ManagerId,
        c.Name AS ManagerName,
        c.Surname AS ManagerSurname,
        c.Email AS ManagerEmail,

        COUNT(DISTINCT ut.UnderTaskId) AS CreatedUnderTasks,

        COUNT(DISTINCT th.TaskId) AS AssignedTasks,

        COUNT(DISTINCT CASE
```

```
            WHEN t.Status = 'In Progress'
            THEN t.TaskId
        END) AS TasksInProgress,

        COUNT(DISTINCT CASE
            WHEN t.Status = 'To Do'
            THEN t.TaskId
        END) AS TasksToDo,

        COUNT(DISTINCT CASE
            WHEN t.Status = 'Cancelled'
            THEN t.TaskId
        END) AS CancelledTasks,

        COUNT(DISTINCT CASE
            WHEN t.Status = 'Done'
            THEN t.TaskId
        END) AS CompletedTasks,

        COUNT(DISTINCT CASE
            WHEN t.Status IN ('To Do', 'In Progress')
                AND t.Deadline < CURRENT_DATE
            THEN t.TaskId
        END) AS OverdueTasks

    FROM
        Client c
    LEFT JOIN
        UnderTask ut ON c.ClientId = ut.AuthorId
    LEFT JOIN
        TaskHandler th ON c.ClientId = th.ClientId
    LEFT JOIN
        Task t ON th.TaskId = t.TaskId
    WHERE
        c.Role = 'Manager'
        AND (p_ManagerId IS NULL OR c.ClientId = p_ManagerId)
    GROUP BY
        c.ClientId, c.Name, c.Surname, c.Email
    ORDER BY
        c.ClientId;
END;
$$ LANGUAGE plpgsql;
```

Додаток Г

ПРОГРАМНИЙ КОД

auth.py:

```
from django.shortcuts import render, redirect
from django.db import connection
from django.conf import settings
from core.models import Client
import psycopg


def custom_login(request):
    error = ''
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')

        try:
            user = Client.objects.get(email=email)
            db_user = f"user_{user.clientid}"
            db_conf = settings.DATABASES['default']

            try:
                with psycopg.connect(
                        dbname=db_conf['NAME'],
                        user=db_user,
                        password=password,
                        host=db_conf['HOST'],
                        port=db_conf['PORT']
                ) as conn:
                    pass

                request.session['user_id'] = user.clientid
                request.session['db_role'] = user.role
                request.session['db_user'] = db_user

                return redirect('dashboard')

            except psycopg.OperationalError:
                error = 'Incorrect password'

        except Client.DoesNotExist:
            error = 'User not found'
        except Exception as e:
            error = f'System error: {e}'
```

```
    return render(request, 'login.html', {'error': error})


def logout_view(request):
    with connection.cursor() as cursor:
        cursor.execute("RESET ROLE")
    request.session.flush()
    return redirect('login')
```

dashboard.py

```
from django.shortcuts import render, redirect
from django.db.models import Q
from core.models import Client, Task, Project, Message,
UnderTask, Event, Team
from core.forms import TaskForm, MessageForm, StatusUpdateForm,
UnderTaskForm, EventForm, AddExecutorForm, \
    SubmitResultForm, ProjectForm, TeamForm
from datetime import datetime, timedelta


def dashboard(request):
    role = request.session.get('db_role', 'connect_user')
    user_id = request.session.get('user_id')
    if not user_id:
        return redirect('login')

    try:
        user = Client.objects.get(clientid=user_id)
    except Client.DoesNotExist:
        return redirect('logout')

    context = {
        'role': role,
        'user': user,
        'tasks': [],
        'projects': [],
        'messages_list': [],
        'events': [],
        'stats': None,
        'team_members': [],
        'teams': [],
        'undertasks': [],
        'forms': {
            'task': TaskForm(),
            'message': MessageForm(),
```

```
            'status': StatusUpdateForm(),
            'undertask': UnderTaskForm(),
            'event': EventForm(),
            'add_executor': AddExecutorForm(),
            'submit_result': SubmitResultForm(),
            'project': ProjectForm(),
            'team': TeamForm(),
        }
    }

    try:
        if role == 'Developer':
            tasks_query =
Task.objects.filter(taskhandler__client__clientid=user_id).selec
t_related('project')
            context['undertasks'] =
UnderTask.objects.filter(developer__clientid=user_id).select_rel
ated('task',

'author')

        elif role == 'Individual':
            tasks_query =
Task.objects.all().select_related('project')
            context['undertasks'] =
UnderTask.objects.all().select_related('task', 'author',
'developer')

        else:
            tasks_query =
Task.objects.all().select_related('project')
            context['undertasks'] =
UnderTask.objects.all().select_related('task', 'author',
'developer')

        project_filter = request.GET.get('project')
        if project_filter:
            tasks_query =
tasks_query.filter(project_id=project_filter)

        priority_filter = request.GET.get('priority')
        if priority_filter:
            tasks_query =
tasks_query.filter(priority=priority_filter)

        status_filter = request.GET.get('status')
        if status_filter:
```

```
            tasks_query =
tasks_query.filter(status=status_filter)

        deadline_filter = request.GET.get('deadline')
        if deadline_filter:
            today = datetime.now().date()
            if deadline_filter == 'urgent':
                tasks_query =
tasks_query.filter(deadline__lte=today + timedelta(days=3),
deadline__gte=today)
            elif deadline_filter == 'soon':
                tasks_query =
tasks_query.filter(deadline__lte=today + timedelta(days=7),

deadline__gt=today + timedelta(days=3))
            elif deadline_filter == 'later':
                tasks_query =
tasks_query.filter(deadline__gt=today + timedelta(days=7))

        context['tasks'] = tasks_query.order_by('-creationdate')

        context['messages_list'] = Message.objects.filter(
            Q(receiver__clientid=user_id) |
Q(sender__clientid=user_id)
        ).select_related('sender', 'receiver').order_by('-date')

        context['events'] =
Event.objects.all().select_related('author').order_by('-date')

        context['projects'] = Project.objects.all()

        if role in ['Manager', 'Lead']:
            context['team_members'] = Client.objects.all()
            context['teams'] = Team.objects.all()

        if role == 'Lead':
            total_tasks = Task.objects.count()
            done_tasks =
Task.objects.filter(status='Done').count()
            canceled_tasks =
Task.objects.filter(status='Canceled').count()
            progress_percent = round((done_tasks / total_tasks *
100), 1) if total_tasks > 0 else 0

            context['stats'] = {
                'total_tasks': total_tasks,
                'done': done_tasks,
                'canceled': canceled_tasks,
```

```
                'progress': progress_percent,
                'active_projects':
Project.objects.filter(status='In Progress').count(),
                'total_projects': Project.objects.count(),
            }

        elif role == 'Individual':
            total_tasks = Task.objects.count()
            done_tasks =
Task.objects.filter(status='Done').count()
            progress = round((done_tasks / total_tasks * 100),
1) if total_tasks > 0 else 0

            context['stats'] = {
                'total_tasks': total_tasks,
                'done': done_tasks,
                'progress': progress,
                'active_projects':
Project.objects.filter(status='In Progress').count()
            }

    except Exception as e:
        context['error'] = f"Помилка БД: {str(e)}"

    return render(request, 'dashboard.html', context)
```

events.py

```
from django.shortcuts import redirect
from django.contrib import messages as django_messages
from django.http import JsonResponse
from core.models import Event, EventHandler, Client


def create_event(request):
    if request.method == 'POST':
        try:
            evt = Event.objects.create(
                theme=request.POST.get('theme'),
                description=request.POST.get('description'),
                priority=request.POST.get('priority'),
                date=request.POST.get('date'),
                duration=request.POST.get('duration'),
                link=request.POST.get('link', ''),
                author_id=request.session.get('user_id')
            )
```

```
            participants = request.POST.getlist('participants')
            for participant_id in participants:
                EventHandler.objects.create(event=evt,
client_id=participant_id)

            django_messages.success(request, f"Захід
'{evt.theme}' створено!")
        except Exception as e:
            django_messages.error(request, f"Помилка створення
заходу: {e}")
    return redirect('dashboard')


def edit_event(request):
    if request.method == 'POST':
        event_id = request.POST.get('event_id')
        try:
            event = Event.objects.get(eventid=event_id)
            if str(event.author.clientid) ==
str(request.session.get('user_id')) or
request.session.get('db_role') in [
                'Lead', 'Manager']:
                event.theme = request.POST.get('theme')
                event.description =
request.POST.get('description')
                event.priority = request.POST.get('priority')
                event.date = request.POST.get('date')
                event.duration = request.POST.get('duration')
                event.link = request.POST.get('link')
                event.save()
                django_messages.success(request, f"Захід
'{event.theme}' оновлено!")
            else:
                django_messages.error(request, "У вас немає прав
редагувати цей захід")
        except Exception as e:
            django_messages.error(request, f"Помилка
редагування: {e}")
    return redirect('dashboard')


def delete_event(request):
    if request.method == 'POST':
        event_id = request.POST.get('event_id')
        try:
            event = Event.objects.get(eventid=event_id)
            theme = event.theme
```

```
            event.delete()
            django_messages.success(request, f"Захід '{theme}'
видалено")
        except Exception as e:
            django_messages.error(request, f"Помилка видалення
заходу: {e}")
    return redirect('dashboard')


def get_event_participants(request, event_id):
    try:
        participants =
EventHandler.objects.filter(event_id=event_id).select_related('c
lient')
        data = {
            'participants': [
                {'id': p.client.clientid, 'name': p.client.name,
'surname': p.client.surname}
                for p in participants
            ]
        }
        return JsonResponse(data)
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=400)


def add_event_participant(request):
    if request.method == 'POST':
        event_id = request.POST.get('event_id')
        client_id = request.POST.get('client_id')
        try:
            if not
EventHandler.objects.filter(event_id=event_id,
client_id=client_id).exists():
                EventHandler.objects.create(event_id=event_id,
client_id=client_id)
                django_messages.success(request, "Учасника
додано!")
            else:
                django_messages.warning(request, "Цей учасник
вже доданий")
        except Exception as e:
            django_messages.error(request, f"Помилка додавання:
{e}")
    return redirect('dashboard')


def remove_event_participant(request):
```

```
    if request.method == 'POST':
        event_id = request.POST.get('event_id')
        client_id = request.POST.get('client_id')
        try:
            EventHandler.objects.filter(event_id=event_id,
client_id=client_id).delete()
            django_messages.success(request, "Учасника видалено
з заходу")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')
```

statistics.py

```
from django.shortcuts import render, redirect
from django.db.models import Q
from core.models import Client, Team, Project, Task, UnderTask


def view_statistics(request):
    role = request.session.get('db_role')
    user_id = request.session.get('user_id')

    if not user_id:
        return redirect('login')

    context = {
        'role': role,
        'user': Client.objects.get(clientid=user_id),
    }

    try:
        if role == 'Lead':
            context['team_stats'] = []
            teams = Team.objects.all()
            for team in teams:
                members =
Client.objects.filter(teamhandler__team=team)
                projects =
Project.objects.filter(projecthandler__team=team)
                tasks =
Task.objects.filter(project__in=projects)

                context['team_stats'].append({
                    'team': team,
                    'members_count': members.count(),
                    'projects_count': projects.count(),
                    'tasks_total': tasks.count(),
```

```
                    'tasks_done':
tasks.filter(status='Done').count(),
                    'tasks_active': tasks.filter(status__in=['To
Do', 'In Progress', 'Review']).count(),
                })

            context['worker_stats'] = []
            workers =
Client.objects.filter(role__in=['Developer', 'Manager'])
            for worker in workers:
                tasks_assigned =
Task.objects.filter(taskhandler__client=worker)
                undertasks =
UnderTask.objects.filter(developer=worker)

                context['worker_stats'].append({
                    'worker': worker,
                    'tasks_total': tasks_assigned.count(),
                    'tasks_done':
tasks_assigned.filter(status='Done').count(),
                    'undertasks_total': undertasks.count(),
                    'undertasks_done':
undertasks.filter(status='Done').count(),
                })

        elif role == 'Individual':
            my_tasks = Task.objects.filter(
                Q(undertask__author__clientid=user_id) |
Q(taskhandler__client__clientid=user_id)
            ).distinct()

            context['my_stats'] = {
                'total_tasks': my_tasks.count(),
                'done_tasks':
my_tasks.filter(status='Done').count(),
                'active_tasks': my_tasks.filter(status__in=['To
Do', 'In Progress', 'Review']).count(),
                'canceled_tasks':
my_tasks.filter(status='Canceled').count(),
            }

    except Exception as e:
        context['error'] = f"Помилка отримання статистики: {e}"

    return render(request, 'statistics.html', context)
```

tasks.py

```
from django.shortcuts import redirect
from django.db import transaction
from django.contrib import messages as django_messages
from core.models import Task, TaskHandler, UnderTask, Client
from core.forms import TaskForm, UnderTaskForm
import datetime


def create_task(request):
    if request.method == 'POST':
        form = TaskForm(request.POST)
        if form.is_valid():
            try:
                with transaction.atomic():
                    task = form.save(commit=False)
                    task.status = 'To Do'
                    task.save()

                    # Логика для добавления исполнителя или
автора
                    user_role = request.session.get('db_role')
                    user_id = request.session.get('user_id')

                    assignee = form.cleaned_data.get('assignee')

                    # Если создал Individual, он автоматически
прикрепляется, чтобы видеть задачу
                    if user_role == 'Individual':
                        TaskHandler.objects.create(task=task,
client_id=user_id)

                    if assignee:
                        # Проверяем, чтобы не дублировать, если
сам себе назначил
                        if not (user_role == 'Individual' and
str(assignee.clientid) == str(user_id)):

TaskHandler.objects.create(task=task, client=assignee)

                django_messages.success(request, f"Завдання
'{task.title}' створено!")
            except Exception as e:
                django_messages.error(request, f"Помилка
створення завдання: {e}")
        else:
            django_messages.error(request, "Невірні дані форми")
    return redirect('dashboard')
```

```
def edit_task(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        try:
            task = Task.objects.get(taskid=task_id)
            task.title = request.POST.get('title')
            task.description = request.POST.get('description')
            task.priority = request.POST.get('priority')
            task.deadline = request.POST.get('deadline')
            task.save()
            django_messages.success(request, f"Завдання
'{task.title}' оновлено!")
        except Exception as e:
            django_messages.error(request, f"Помилка
редагування: {e}")
    return redirect('dashboard')


def update_task_status(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        new_status = request.POST.get('status')
        try:
            task = Task.objects.get(taskid=task_id)
            old_status = task.status
            task.status = new_status
            task.save()
            django_messages.success(request, f"Статус змінено:
{old_status} → {new_status}")
        except Exception as e:
            django_messages.error(request, f"Помилка оновлення
статусу: {e}")
    return redirect('dashboard')


def reject_task(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        reason = request.POST.get('reject_reason', 'Не вказано')
        try:
            task = Task.objects.get(taskid=task_id)
            task.status = 'To Do'
            task.description =
f"{task.description}\n\n[ВІДХИЛЕНО]: {reason}"
            task.save()
```

```
            django_messages.warning(request, f"Завдання
'{task.title}' відхилено. Причина: {reason}")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')


def delete_task(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        try:
            task = Task.objects.get(taskid=task_id)
            title = task.title
            task.delete()
            django_messages.success(request, f"Завдання
'{title}' видалено")
        except Exception as e:
            django_messages.error(request, f"Помилка видалення:
{e}")
    return redirect('dashboard')


def cancel_task(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        reason = request.POST.get('reason', 'Не вказано')
        try:
            task = Task.objects.get(taskid=task_id)
            task.status = 'Canceled'
            task.description =
f"{task.description}\n\n[СКАСОВАНО]: {reason}"
            task.save()
            django_messages.warning(request, f"Завдання
'{task.title}' скасовано. Причина: {reason}")
        except Exception as e:
            django_messages.error(request, f"Помилка скасування:
{e}")
    return redirect('dashboard')


def submit_result(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        result_link = request.POST.get('result_link')
        try:
            task = Task.objects.get(taskid=task_id)
            task.resultlink = result_link
            task.status = 'Review'
```

```
            task.save()
            django_messages.success(request, f"Результат
завдання '{task.title}' відправлено на перевірку!")
        except Exception as e:
            django_messages.error(request, f"Помилка здачі
роботи: {e}")
    return redirect('dashboard')


def add_executor(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        executor_id = request.POST.get('executor')
        try:
            if not TaskHandler.objects.filter(task_id=task_id,
client_id=executor_id).exists():
                TaskHandler.objects.create(task_id=task_id,
client_id=executor_id)
                django_messages.success(request, "Виконавця
додано до завдання!")
            else:
                django_messages.warning(request, "Цей виконавець
вже призначений на завдання")
        except Exception as e:
            django_messages.error(request, f"Помилка додавання
виконавця: {e}")
    return redirect('dashboard')


def remove_executor(request):
    if request.method == 'POST':
        task_id = request.POST.get('task_id')
        executor_id = request.POST.get('executor_id')
        try:
            TaskHandler.objects.filter(task_id=task_id,
client_id=executor_id).delete()
            django_messages.success(request, "Виконавця видалено
з завдання")
        except Exception as e:
            django_messages.error(request, f"Помилка видалення
виконавця: {e}")
    return redirect('dashboard')


def create_undertask(request):
    if request.method == 'POST':
        form = UnderTaskForm(request.POST)
        parent_task_id = request.POST.get('parent_task_id')
```

```
        if form.is_valid() and parent_task_id:
            try:
                ut = form.save(commit=False)
                ut.task_id = parent_task_id
                ut.author_id = request.session.get('user_id')
                ut.developer = form.cleaned_data['developer']
                ut.status = 'To Do'
                ut.save()
                django_messages.success(request, f"Підзавдання
'{ut.title}' створено!")
            except Exception as e:
                django_messages.error(request, f"Помилка
створення підзавдання: {e}")
        else:
            django_messages.error(request, "Невірні дані")
    return redirect('dashboard')


def submit_undertask_result(request):
    if request.method == 'POST':
        undertask_id = request.POST.get('undertask_id')
        result_link = request.POST.get('result_link')
        try:
            ut = UnderTask.objects.get(undertaskid=undertask_id)
            ut.resultlink = result_link
            ut.status = 'Review'
            ut.completiondate = datetime.date.today()
            ut.save()
            django_messages.success(request, f"Результат
підзавдання '{ut.title}' відправлено на перевірку тімліду!")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')


def update_undertask_status(request):
    if request.method == 'POST':
        undertask_id = request.POST.get('undertask_id')
        new_status = request.POST.get('status')
        try:
            ut = UnderTask.objects.get(undertaskid=undertask_id)
            ut.status = new_status
            if new_status == 'Done':
                ut.completiondate = datetime.date.today()
            ut.save()
            django_messages.success(request, f"Статус
підзавдання оновлено!")
        except Exception as e:
```

```
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')


def reject_undertask(request):
    if request.method == 'POST':
        undertask_id = request.POST.get('undertask_id')
        reason = request.POST.get('reject_reason', 'Не вказано')
        try:
            ut = UnderTask.objects.get(undertaskid=undertask_id)
            ut.status = 'To Do'
            ut.description = f"{ut.description}\n\n[ВІДХИЛЕНО]:
{reason}"
            ut.save()
            django_messages.warning(request, f"Підзавдання
'{ut.title}' відхилено. Причина: {reason}")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')
```

teams.py

```
from django.shortcuts import redirect
from django.db import transaction
from django.contrib import messages as django_messages
from django.http import JsonResponse
from core.models import Team, TeamHandler, Client, Project,
ProjectHandler, Task
from core.forms import TeamForm, ProjectForm


def create_team(request):
    if request.session.get('db_role') not in ['Lead',
'Manager']:
        django_messages.error(request, "У вас немає прав на
створення команд")
        return redirect('dashboard')

    if request.method == 'POST':
        form = TeamForm(request.POST)
        if form.is_valid():
            try:
                with transaction.atomic():
                    team = form.save()
                    user_id = request.session.get('user_id')
                    TeamHandler.objects.create(team=team,
client_id=user_id)
```

```
                    members = request.POST.getlist('members')
                    for member_id in members:
                        if member_id != str(user_id):

TeamHandler.objects.create(team=team, client_id=member_id)

                    django_messages.success(request, f"Команду
'{team.title}' створено!")
            except Exception as e:
                django_messages.error(request, f"Помилка
створення команди: {e}")
        else:
            django_messages.error(request, "Невірні дані форми")
    return redirect('dashboard')


def delete_team(request):
    if request.method == 'POST':
        if request.session.get('db_role') != 'Lead':
            django_messages.error(request, "Тільки Lead може
видаляти команди")
            return redirect('dashboard')

        team_id = request.POST.get('team_id')
        try:
            Team.objects.get(teamid=team_id).delete()
            django_messages.success(request, "Команду
видалено!")
        except Exception as e:
            django_messages.error(request, f"Помилка видалення:
{e}")
    return redirect('dashboard')


def get_team_members(request, team_id):
    if request.session.get('db_role') not in ['Lead',
'Manager']:
        return JsonResponse({'error': 'Forbidden'}, status=403)

    try:
        handlers =
TeamHandler.objects.filter(team_id=team_id).select_related('clie
nt')
        members = [{
            'id': h.client.clientid,
            'name': h.client.name,
            'surname': h.client.surname,
```

```
            'role': h.client.role
        } for h in handlers]
        return JsonResponse({'members': members})
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=400)


def add_team_member(request):
    if request.method == 'POST':
        if request.session.get('db_role') not in ['Lead',
'Manager']:
            django_messages.error(request, "Немає прав")
            return redirect('dashboard')

        team_id = request.POST.get('team_id')
        user_id = request.POST.get('user_id')
        try:
            if not TeamHandler.objects.filter(team_id=team_id,
client_id=user_id).exists():
                TeamHandler.objects.create(team_id=team_id,
client_id=user_id)
                django_messages.success(request, "Учасника
додано!")
            else:
                django_messages.warning(request, "Користувач вже
в команді")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')


def remove_team_member(request):
    if request.method == 'POST':
        if request.session.get('db_role') not in ['Lead',
'Manager']:
            django_messages.error(request, "Немає прав")
            return redirect('dashboard')

        team_id = request.POST.get('team_id')
        client_id = request.POST.get('client_id')
        try:
            TeamHandler.objects.filter(team_id=team_id,
client_id=client_id).delete()
            django_messages.success(request, "Учасника видалено
з команди")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')
```

```
def create_project(request):
    if request.session.get('db_role') not in ['Lead',
'Manager']:
        django_messages.error(request, "Немає прав")
        return redirect('dashboard')

    if request.method == 'POST':
        form = ProjectForm(request.POST)
        if form.is_valid():
            try:
                with transaction.atomic():
                    project = form.save()
                    team_id = request.POST.get('team_id')
                    if team_id:

ProjectHandler.objects.create(project=project, team_id=team_id)
                    django_messages.success(request, f"Проект
'{project.title}' створено!")
            except Exception as e:
                django_messages.error(request, f"Помилка
створення проекту: {e}")
        else:
            django_messages.error(request, "Невірні дані форми")
    return redirect('dashboard')


def complete_project(request):
    if request.method == 'POST':
        project_id = request.POST.get('project_id')
        try:
            project = Project.objects.get(projectid=project_id)
            project.status = 'Done'
            project.save()
            Task.objects.filter(project=project, status__in=['To
Do', 'In Progress', 'Review']).update(
                status='Canceled')
            django_messages.success(request, f"Проект
'{project.title}' завершено!")
        except Exception as e:
            django_messages.error(request, f"Помилка завершення
проекту: {e}")
    return redirect('dashboard')


def delete_project(request):
    if request.method == 'POST':
```

```
        project_id = request.POST.get('project_id')
        if request.session.get('db_role') not in ['Lead',
'Manager']:
            django_messages.error(request, "Немає прав на
видалення проекту")
            return redirect('dashboard')
        try:
            project = Project.objects.get(projectid=project_id)
            title = project.title
            project.delete()
            django_messages.success(request, f"Проект '{title}'
видалено!")
        except Exception as e:
            django_messages.error(request, f"Помилка видалення:
{e}")
    return redirect('dashboard')


def get_project_details(request, project_id):
    if request.session.get('db_role') not in ['Lead',
'Manager']:
        return JsonResponse({'error': 'Forbidden'}, status=403)

    try:
        project = Project.objects.get(projectid=project_id)
        handlers =
ProjectHandler.objects.filter(project=project).select_related('t
eam')
        teams = [{'id': h.team.teamid, 'title': h.team.title}
for h in handlers]
        task_count =
Task.objects.filter(project=project).count()

        return JsonResponse({
            'id': project.projectid,
            'title': project.title,
            'task_count': task_count,
            'teams': teams
        })
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=400)


def add_project_team(request):
    if request.method == 'POST':
        if request.session.get('db_role') not in ['Lead',
'Manager']:
            django_messages.error(request, "Немає прав")
```

```
            return redirect('dashboard')

        project_id = request.POST.get('project_id')
        team_id = request.POST.get('team_id')
        try:
            if not
ProjectHandler.objects.filter(project_id=project_id,
team_id=team_id).exists():

ProjectHandler.objects.create(project_id=project_id,
team_id=team_id)
                django_messages.success(request, "Команду
прикріплено до проекту!")
            else:
                django_messages.warning(request, "Ця команда вже
працює над проектом")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')


def remove_project_team(request):
    if request.method == 'POST':
        if request.session.get('db_role') not in ['Lead',
'Manager']:
            django_messages.error(request, "Немає прав")
            return redirect('dashboard')

        project_id = request.POST.get('project_id')
        team_id = request.POST.get('team_id')
        try:
            ProjectHandler.objects.filter(project_id=project_id,
team_id=team_id).delete()
            django_messages.success(request, "Команду
відкріплено від проекту")
        except Exception as e:
            django_messages.error(request, f"Помилка: {e}")
    return redirect('dashboard')
```

users.py

```
from django.shortcuts import redirect
from django.contrib import messages as django_messages
from django.db import connection
from core.models import Client, Message
```

```
def add_user(request):
    if request.session.get('db_role') != 'Lead':
        django_messages.error(request, "Тільки тімлид може
змінювати ролі")
        return redirect('dashboard')

    if request.method == 'POST':
        try:
            name = request.POST.get('name')
            surname = request.POST.get('surname')
            email = request.POST.get('email')
            raw_password = request.POST.get('password')
            role_input = request.POST.get('role')

            client = Client.objects.create(
                name=name,
                surname=surname,
                email=email,
                role=role_input
            )

            db_user = f"user_{client.clientid}"
            safe_password = raw_password.replace("'", "''")

            with connection.cursor() as cursor:
                cursor.execute("RESET ROLE")
                cursor.execute(
                    f'CREATE ROLE "{db_user}" WITH LOGIN
PASSWORD \'{safe_password}\' IN ROLE "{role_input}";')
                cursor.execute(f'GRANT "{db_user}" TO
"connect_user";')

            django_messages.success(request, f"Користувача
додано. Роль у базі даних: {db_user}")
        except Exception as e:
            django_messages.error(request, f"Помилка додавання
користувача: {e}")
    return redirect('dashboard')


def delete_user(request):
    if request.session.get('db_role') != 'Lead':
        django_messages.error(request, "Тільки тімлид може
змінювати ролі")
        return redirect('dashboard')

    if request.method == 'POST':
        user_id = request.POST.get('user_id')
```

```
        try:
            user = Client.objects.get(clientid=user_id)
            db_user = f"user_{user.clientid}"

            with connection.cursor() as cursor:
                cursor.execute("RESET ROLE")
                cursor.execute(f'REASSIGN OWNED BY "{db_user}"
TO "connect_user";')
                cursor.execute(f'DROP OWNED BY "{db_user}";')
                cursor.execute(f'DROP ROLE IF EXISTS
"{db_user}";')

            user.delete()
            django_messages.success(request, f"Користувача
видалено")
        except Exception as e:
            django_messages.error(request, f"Error deleting
user: {e}")
    return redirect('dashboard')


def change_user_role(request):
    if request.session.get('db_role') != 'Lead':
        django_messages.error(request, "Тільки тімлид може
змінювати ролі")
        return redirect('dashboard')

    if request.method == 'POST':
        user_id = request.POST.get('user_id')
        new_role = request.POST.get('role')
        try:
            user = Client.objects.get(clientid=user_id)
            old_role = user.role
            db_user = f"user_{user.clientid}"

            user.role = new_role
            user.save()

            with connection.cursor() as cursor:
                cursor.execute("RESET ROLE")
                cursor.execute(f'REVOKE "{old_role}" FROM
"{db_user}"')
                cursor.execute(f'GRANT "{new_role}" TO
"{db_user}"')

            django_messages.success(request, f"Роль змінено:
{old_role} -> {new_role}")
        except Exception as e:
```

```
            django_messages.error(request, f"Error changing
role: {e}")
    return redirect('dashboard')


def send_message(request):
    if request.method == 'POST':
        try:
            receiver_id = request.POST.get('receiver')
            title = request.POST.get('title')
            text = request.POST.get('text')
            sender_id = request.session.get('user_id')

            if not all([receiver_id, title, text, sender_id]):
                django_messages.error(request, "Заповніть усі
поля")
                return redirect('dashboard')

            Message.objects.create(
                sender_id=sender_id,
                receiver_id=receiver_id,
                title=title,
                text=text
            )

            django_messages.success(request, "Повідомлення
надіслано!")
        except Exception as e:
            django_messages.error(request, f"Error: {e}")
    return redirect('dashboard')
```

models.py

```
from django.db import models

class Priority(models.TextChoices):
    EXTREME = 'Extreme', 'Extreme'
    HIGH = 'High', 'High'
    AVERAGE = 'Average', 'Average'
    LOW = 'Low', 'Low'

class Status(models.TextChoices):
    TODO = 'To Do', 'To Do'
    IN_PROGRESS = 'In Progress', 'In Progress'
    REVIEW = 'Review', 'Review'
    DONE = 'Done', 'Done'
    CANCELED = 'Canceled', 'Canceled'
```

```
class Role(models.TextChoices):
    NO_COMMERCE = 'No Commerce', 'No Commerce'
    DEVELOPER = 'Developer', 'Developer'
    MANAGER = 'Manager', 'Manager'
    LEAD = 'Lead', 'Lead'
    INDIVIDUAL = 'Individual', 'Individual'

class Client(models.Model):
    clientid = models.BigAutoField(primary_key=True,
db_column='clientid')
    name = models.CharField(max_length=30)
    surname = models.CharField(max_length=30)
    email = models.EmailField(unique=True, max_length=64)
    avatar = models.CharField(max_length=2048,
default='idle.png')
    role = models.CharField(max_length=30, choices=Role.choices)

    class Meta:
        managed = False
        db_table = 'client'

    def __str__(self):
        return f"{self.name} {self.surname}"

class Team(models.Model):
    teamid = models.BigAutoField(primary_key=True,
db_column='teamid')
    title = models.CharField(max_length=50)
    description = models.CharField(max_length=255)

    class Meta:
        managed = False
        db_table = 'team'

    def __str__(self):
        return self.title

class TeamHandler(models.Model):
    id = models.BigAutoField(primary_key=True)
    team = models.ForeignKey(Team, on_delete=models.CASCADE,
db_column='teamid')
    client = models.ForeignKey(Client, on_delete=models.CASCADE,
db_column='clientid')

    class Meta:
        managed = False
        db_table = 'teamhandler'
```

```
        unique_together = (('team', 'client'),)

class Project(models.Model):
    projectid = models.BigAutoField(primary_key=True,
db_column='projectid')
    title = models.CharField(max_length=50)
    status = models.CharField(max_length=20,
choices=Status.choices, default=Status.TODO)
    medialink = models.CharField(max_length=2048, blank=True,
null=True)
    creationdate = models.DateField(auto_now_add=True)
    deadline = models.DateField()

    class Meta:
        managed = False
        db_table = 'project'

    def __str__(self):
        return self.title

class ProjectHandler(models.Model):
    id = models.BigAutoField(primary_key=True)
    team = models.ForeignKey(Team, on_delete=models.CASCADE,
db_column='teamid')
    project = models.ForeignKey(Project,
on_delete=models.CASCADE, db_column='projectid')

    class Meta:
        managed = False
        db_table = 'projecthandler'
        unique_together = (('team', 'project'),)

class Task(models.Model):
    taskid = models.BigAutoField(primary_key=True,
db_column='taskid')
    project = models.ForeignKey(Project,
on_delete=models.CASCADE, blank=True, null=True,
db_column='projectid')
    title = models.CharField(max_length=100)
    priority = models.CharField(max_length=7,
choices=Priority.choices)
    description = models.CharField(max_length=1500)
    creationdate = models.DateField(auto_now_add=True)
    deadline = models.DateField()
    status = models.CharField(max_length=20,
choices=Status.choices, default=Status.TODO)
    resultlink = models.CharField(max_length=2048, blank=True,
null=True)
```

```
    class Meta:
        managed = False
        db_table = 'task'

    def __str__(self):
        return self.title

class TaskHandler(models.Model):
    id = models.BigAutoField(primary_key=True)
    task = models.ForeignKey(Task, on_delete=models.CASCADE,
db_column='taskid')
    client = models.ForeignKey(Client, on_delete=models.CASCADE,
db_column='clientid')

    class Meta:
        managed = False
        db_table = 'taskhandler'
        unique_together = (('task', 'client'),)

class UnderTask(models.Model):
    undertaskid = models.BigAutoField(primary_key=True,
db_column='undertaskid')
    task = models.ForeignKey(Task, on_delete=models.CASCADE,
db_column='taskid')
    developer = models.ForeignKey(Client,
on_delete=models.SET_NULL, blank=True, null=True,
related_name='undertasks_dev', db_column='developerid')
    author = models.ForeignKey(Client,
on_delete=models.SET_NULL, blank=True, null=True,
related_name='undertasks_author', db_column='authorid')
    title = models.CharField(max_length=100)
    description = models.CharField(max_length=1500)
    creationdate = models.DateField(auto_now_add=True)
    completiondate = models.DateField(blank=True, null=True)
    status = models.CharField(max_length=20,
choices=Status.choices, default=Status.TODO)
    resultlink = models.CharField(max_length=2048, blank=True,
null=True)

    class Meta:
        managed = False
        db_table = 'undertask'

class Message(models.Model):
    messageid = models.BigAutoField(primary_key=True,
db_column='messageid')
```

```
    sender = models.ForeignKey(Client, on_delete=models.CASCADE,
related_name='sent_messages', db_column='senderid')
    receiver = models.ForeignKey(Client,
on_delete=models.CASCADE, related_name='received_messages',
db_column='receiverid')
    title = models.CharField(max_length=100)
    text = models.CharField(max_length=1500)
    date = models.DateTimeField(auto_now_add=True)

    class Meta:
        managed = False
        db_table = 'message'

class Event(models.Model):
    eventid = models.BigAutoField(primary_key=True,
db_column='eventid')
    priority = models.CharField(max_length=7,
choices=Priority.choices)
    author = models.ForeignKey(Client, on_delete=models.CASCADE,
db_column='authorid')
    theme = models.CharField(max_length=100)
    description = models.CharField(max_length=1500)
    date = models.DateTimeField()
    duration = models.DurationField()
    link = models.CharField(max_length=2048, blank=True,
null=True)

    class Meta:
        managed = False
        db_table = 'event'

class EventHandler(models.Model):
    id = models.BigAutoField(primary_key=True)
    event = models.ForeignKey(Event, on_delete=models.CASCADE,
db_column='eventid')
    client = models.ForeignKey(Client, on_delete=models.CASCADE,
db_column='clientid')

    class Meta:
        managed = False
        db_table = 'eventhandler'
        unique_together = (('event', 'client'),)

class MediaLink(models.Model):
    medialinkid = models.BigAutoField(primary_key=True,
db_column='medialinkid')
    url = models.CharField(max_length=2048)
```

```
    description = models.CharField(max_length=255, blank=True,
null=True)
    task = models.ForeignKey(Task, on_delete=models.CASCADE,
blank=True, null=True, db_column='taskid')
    undertask = models.ForeignKey(UnderTask,
on_delete=models.CASCADE, blank=True, null=True,
db_column='undertaskid')
    message = models.ForeignKey(Message,
on_delete=models.CASCADE, blank=True, null=True,
db_column='messageid')
    event = models.ForeignKey(Event, on_delete=models.CASCADE,
blank=True, null=True, db_column='eventid')

    class Meta:
        managed = False
        db_table = 'medialink'
```