ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки,
молоді та спорту України
29.03.2012 N 384

Форма N Н-6.01

Міністерство освіти і науки України
Державний університет інтелектуальних технологій і зв'язку

Кафедра інженерії програмного забезпечення

КУРСОВИЙ ПРОЄКТ

з дисципліни «Веб-технології»

на тему: «Розробка клієнт-серверної платформи для організації роботи команди розробників»

Студента__3___ курсу _ групи ___ІПЗ-3.02___
спеціальності 121 «Інженерія програмного забезпечення»
Вайцеховський Олександр Сергійович

Керівник: ст.викл. каф. КН Северин М.В.

Національна шкала _____
Кількість балів: _____ Оцінка: ECTS _____

Члени комісії

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

м. Одеса – 2025 рік

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ І ЗВ'ЯЗКУ

Факультет інформаційних технологій та кібербезпеки
Кафедра інженерії програмного забезпечення

З А В Д А Н Н Я

НА КУРСОВИЙ ПРОЄКТ

з дисципліни «Веб-технології»

Студенту Вайцеховському Олександру Сергійовичу

Спеціальності 121 «Інженерія програмного забезпечення»

Третього курсу групи ІПЗ-3.02

Тема завдання «Розробити інформаційну систему для аналізу вимог до програмного забезпечення»

Варіант: 78

Вхідні дані
Документація по HTML, CSS, JavaScript, Django (Python), вимоги до оформлення роботи, нормативні документи.

Курсовий проєкт виконується в наступному об'ємі:

I. РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНА ЗАПИСКА
1) АНАЛІЗ ПРОЄКТУ «РОЗРОБИТИ ІНФОРМАЦІЙНУ СИСТЕМУ ДЛЯ АНАЛІЗУ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»;
2) РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА (UI/UX ДИЗАЙН);
3) РОЗРОБКА БАЗИ ДАННИХ;
4) ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ МОНОЛІТУ;

II ГРАФІЧНА ЧАСТИНА
не передбачено

Рекомендована література:

1. Северин М. Веб-технології та веб-дизайн : методичні вказівки до практичних занять [для здобувачів першого (бакалаврський) рівня вищої освіти галузі знань F «Інформаційні технології»] Одеса : ДУІТЗ, 2025. 91 с. URL: https://metod.suitt.edu.ua/download/939 (дата звернення: 14.02.2025)
2. HTML Підручник : веб-сайт. URL: https://w3schoolsua.github.io/html/index.html#gsc.tab=0 (дата звернення: 14.02.2025)
3. CSS Підручник : веб-сайт. URL: https://w3schoolsua.github.io/css/index.html#gsc.tab=0 (дата звернення: 14.02.2025)
4. JavaScript Підручник. Основи вебпрограмування : веб-сайт. URL: https://w3schoolsua.github.io/js/index.html#gsc.tab=0 (дата звернення: 14.02.2025)
5. Сучасний підручник з JavaScript : веб-сайт. URL: https://uk.javascript.info (дата звернення: 14.02.2025)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Отримання та узгодження теми проекту.		
2.	Аналіз проекту. Порівняння з аналогами, встановлення вимог та постановка задачі.		
3.	Проєктування та розробка інтерфейсу користувача (UX/UI дизайну).		
4.	Проєктування та програмна реалізація бізнес-логіки проєкту.		
7.	Оформлення пояснювальної записки		
8.	Подання курсового проєкту на захист керівнику		
9	Захист курсового проєкту		

Дата видачі роботи ”_”_____ _____ 2025 р.

Дата захисту роботи ”__”_____ _____ 2025 р.

Студент _____ _Вайцеховський О. С._
(підпис) (прізвище та ініціали,)

Керівник курсової роботи _____ ____Северин М.В.___
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Курсовий проєкт: 80 с., 10 рис., 10 джерел.

Актуальність теми Актуальність теми зумовлена стрімким зростанням складності процесів розробки програмного забезпечення та високими вимогами ринку до швидкості впровадження продуктів. У сучасних умовах ефективність роботи IT-команди залежить не лише від професійних навичок окремих розробників, а й від якості організації їхньої взаємодії. Відсутність централізованого керування задачами, хаотична комунікація та втрата контексту при обговоренні вимог призводять до зриву дедлайнів, зниження якості коду та фінансових втрат [9]. Впровадження спеціалізованих інформаційних систем для управління проектами стає критичною необхідністю, що дозволяє структурувати робочий процес, забезпечити прозорість виконання завдань та налагодити ефективний зворотний зв'язок між виконавцями та замовниками (Q/A сесії). Автоматизація рутинних адміністративних дій звільняє час для вирішення творчих та технічних завдань, що є ключовим фактором конкурентоспроможності команди в умовах динамічного цифрового світу.

Метою роботою є створення інформаційної системи для ефективного, прозорого та структурованого управління процесами розробки програмного забезпечення. Додатковою метою є інтегрування інструментів для прямої комунікації з клієнтами (Q/A сесій) та автоматизація менеджменту робочих ресурсів команди.

Структура та обсяг. Курсова робота складається з чотирьох основних частин.

У першій частині продемонстровано перехід від створення мокапів і прототипів до повноцінного дизайну інтерфейсу системи управління проектами.

У другій частині буде розроблено структуру бази даних із використанням PostgreSQL, що забезпечить зберігання та ефективне керування даними про співробітників, проекти, технічні задачі, звіти та історію комунікації з клієнтами.

У третій частині буде детально розглянуто процес створення клієнтської частини, зокрема інтерфейсу, що дозволить розробникам та менеджерам зручно взаємодіяти з платформою для постановки та контролю виконання завдань.

У четвертій частині буде реалізовано серверну частину, яка забезпечить бізнес-логіку менеджменту ресурсів, взаємодію між клієнтом і базою даних, включаючи налаштування API для обробки запитів і відповідей.

Ключові слова: сервер, клієнт, таск, захід, менеджер, розробник, лід.

ABSTRACT

The relevance of the topic is driven by the rapid increase in the complexity of software development processes and high market demands for the speed of product deployment. In the modern environment, the efficiency of an IT team depends not only on the professional skills of individual developers but also on the quality of their collaboration organization. The lack of centralized task management, chaotic communication, and loss of context when discussing requirements lead to missed deadlines, reduced code quality, and financial losses. The implementation of specialized information systems for project management is becoming a critical necessity, allowing for the structuring of the workflow, ensuring transparency in task execution, and establishing effective feedback between performers and clients (Q/A sessions). Automation of routine administrative actions frees up time for solving creative and technical tasks, which is a key factor in a team's competitiveness in the dynamic digital world.

The aim of the work is to create an information system for effective, transparent, and structured management of software development processes. An additional aim is to integrate tools for direct communication with clients (Q/A sessions) and to automate the management of the team's working resources.

Structure and volume. The coursework consists of four main parts.

The first part demonstrates the transition from creating mockups and prototypes to a full-fledged interface design for the project management system.

The second part involves the development of the database structure using PostgreSQL, which will ensure the storage and effective management of data regarding employees, projects, technical tasks, reports, and the history of communication with clients.

The third part details the process of creating the client side, specifically the interface, which will allow developers and managers to conveniently interact with the platform for assigning and monitoring task execution.

The fourth part implements the server side, which will provide the business logic for resource management and the interaction between the client and the database, including the configuration of the API for processing requests and responses.

Keywords: server, client, task, event, manager, developer, lead.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DAL (Data Access Layer) — це рівень доступу до даних, який відповідає за взаємодію програми з базою даних. У даному проекті він реалізований гібридно: через високорівневі моделі Django та через прямі SQL-з'єднання драйвера.

MVT (Model-View-Template) — це архітектурний патерн проектування, що використовується у фреймворку Django. Він розділяє логіку програми на три частини: Модель (дані), Представлення (бізнес-логіка) та Шаблон (інтерфейс користувача).

API (Application Programming Interface) — це інтерфейс прикладного програмування. У межах проекту використовується внутрішній API для забезпечення асинхронної взаємодії між клієнтським JavaScript (Fetch API) та сервером для динамічного оновлення контенту без перезавантаження сторінки.

RBAC (Role-Based Access Control) — це механізм розмежування доступу, заснований на ролях користувачів. У проекті цей підхід реалізовано на рівні СУБД PostgreSQL, де кожен користувач системи діє в межах прав відповідної ролі бази даних (наприклад, Manager, Developer).

XSS (Cross-Site Scripting) — це тип атаки на вебсайт, за якого зловмисник впроваджує шкідливий скрипт у вебсторінку. У проекті захист від XSS забезпечується автоматичним екрануванням даних у шаблонному рушії Jinja2.

Django ORM (Object-Relational Mapping) — це вбудований інструмент фреймворку Django, який дозволяє взаємодіяти з базою даних PostgreSQL, використовуючи об'єктно-орієнтовані концепції (класи Python), що спрощує виконання стандартних операцій (CRUD) та керування структурою БД.

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних (СУБД), яка використовується в проекті для зберігання інформації та забезпечення безпеки даних через систему ролей та політик доступу.

psycopg — це адаптер бази даних PostgreSQL для мови програмування Python, який використовується в проекті для реалізації безпечного підключення та виконання команд перемикання ролей користувачів.

ВСТУП

Організація командної розробки програмного забезпечення є одним із найбільш відповідальних напрямків в IT-індустрії, і сьогодні підвищення її ефективності стає можливим завдяки поєднанню сучасних методологій та спеціалізованих цифрових інструментів. У зв'язку зі зростаючою складністю проектів та необхідністю швидкого випуску продуктів на ринок, з'являється гостра потреба у інтегрованих платформах для управління проектами, які не лише забезпечують фіксацію задач, а й адаптують робочі процеси під специфіку конкретної команди.

Зокрема, автоматизація рутинних процесів менеджменту має великий потенціал для оптимізації роботи розробників, забезпечуючи прозорий контроль ресурсів та часу. Ідея полягає в тому, щоб інформаційна система сприяла як у безпосередньому виконанні завдань (через зручний трекінг та комунікацію), так і в стратегічному плануванні, що дозволяє створити єдиний простір для взаємодії між розробниками, менеджерами та клієнтами (через Q/A сесії).

Клієнт-серверна архітектура є основою цієї платформи, що дозволяє забезпечити надійність зберігання даних та гнучкість доступу до системи. Вона дає можливість не тільки обробляти великі масиви інформації про проекти та співробітників, а й розмежовувати права доступу, гарантуючи, що кожен учасник процесу — від Junior-розробника до замовника — отримає актуальні дані відповідно до своєї ролі.

Такі технології дозволяють розробити потужні інформаційні системи, здатні ефективно координувати роботу розподілених команд, надаючи інструменти для менеджменту повного циклу розробки. Це створює продуктивне та організоване середовище для створення програмного забезпечення, де система виступає не просто як сховище даних, а як ключовий інструмент забезпечення успіху проекту.

1 АНАЛІЗ ПРОЄКТУ «РОЗРОБИТИ ІНФОРМАЦІЙНУ СИСТЕМУ ДЛЯ АНАЛІЗУ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

1.1 Аналіз предметної області та порівняння з аналогами

Управління проектами та організація командної роботи є актуальним і критично важливим напрямком в IT-індустрії. У сучасному світі зростає потреба в інтегрованих платформах, що дозволяють командам різного розміру вести розробку ефективно, прозоро та у зручному для них форматі. Традиційні методи контролю (електронні таблиці, месенджери) все більше витісняються спеціалізованими технологічними рішеннями — таск-трекерами, CRM-системами та комплексними інструментами для менеджменту ресурсів.

Для створення конкурентоспроможної інформаційної системи необхідно врахувати досвід, функціонал та особливості робочих процесів у існуючих аналогах. Для порівняльного аналізу використано такі популярні рішення, як Jira, Trello та Asana — це три платформи, які є лідерами ринку у сфері організації роботи команд.

- Jira. Одна з найпотужніших платформ для розробки програмного забезпечення, що є фактичним стандартом в індустрії. Основні переваги включають гнучкість налаштувань, підтримку методологій Agile (Scrum, Kanban) та глибоку інтеграцію з інструментами розробки. Основними недоліками є перевантажений інтерфейс та високий поріг входження. Jira — це інструмент здебільшого для великих ентерпрайз-команд, і для малого проекту її функціонал може бути надлишковим та складним у налаштуванні.
- Trello. Орієнтується на візуалізацію процесів через систему карток (Kanban-дошки). Trello фокусується на простоті та інтуїтивності використання, що дозволяє швидко почати роботу без довгого навчання. Проте його функціонал є досить обмеженим для складних технічних

проектів: тут важко відстежувати залежності між задачами, вести детальний облік часу розробників або керувати великою кількістю спринтів.

- Asana. На мою думку — це «золота середина» серед аналогів, оскільки має й зручний сучасний інтерфейс, й достатню функціональну базу для менеджменту. Проте, безкоштовні версії мають суттєві обмеження, а сама система є універсальною, тобто не заточеною суто під специфіку розробки (наприклад, Q/A сесії чи специфічні ролі розробників тут доводиться налаштовувати вручну або через сторонні плагіни).

Розробляєму платформу, буде орієнтована на вирішення конкретних проблем IT-команд, зокрема на поєднання внутрішнього менеджменту з комунікацією із замовником, що є ключовим для успішної здачі проекту. Для цього буде створено структуровану систему ролей, інтуїтивний інтерфейс для постановки завдань та окремий модуль для Q/A сесій. Після реалізації базового функціоналу, перспективним завданням стане інтеграція засобів автоматизації та аналітики. Це дозволить системі не лише зберігати дані, а й допомагати у розподілі навантаження на розробників, прогнозувати дедлайни та формувати звіти успішності на основі реальних показників команди.

1.2 Визначення ключових вимог до платформи

Уніфікованість інтерфейсу — Платформа повинна дотримуватися єдиної візуальної стилістики та логіки взаємодії в усіх модулях системи. Це знижує поріг входження для нових співробітників та дозволяє користувачам інтуїтивно орієнтуватися як у створенні задач, так і в адміністративних налаштуваннях без необхідності додаткового навчання.

Рольова адаптивність та гнучкість — Система повинна підлаштовувати функціонал та інтерфейс під конкретну роль користувача (менеджер, розробник, клієнт), забезпечуючи персоналізований вигляд дашбордів та відповідний рівень доступу до інструментів редагування.

Масштабованість — Система повинна бути архітектурно готовою до збільшення навантаження: зростання кількості активних проектів, розширення штату співробітників та збільшення обсягу збереженої технічної документації без втрати швидкодії.

Безпека даних — Платформа повинна гарантувати захист комерційної таємниці, технічних завдань та історії комунікації (Q/A сесій). Це передбачає надійну автентифікацію, розмежування прав доступу до бази даних та захист від витоку корпоративної інформації.

1.3 Постановка завдань

1) Розробка мокапів основних екранів системи.
2) Розробка фінального дизайну інтерфейсу користувача.
3) Проектування та реалізація схеми бази даних PostgreSQL.
4) Розробка представлень (Views/ViewSets) на стороні сервера для реалізації бізнес-логіки керування проектами та задачами.
5) Розробка шаблонів (Templates) на стороні сервера для відображення клієнтських сторінок.
6) Створення клієнтських сторінок: дашборд виконавця, списки проектів, канбан-дошка задач та чат. Використання фіктивних даних для візуалізації верстки.
7) Налаштування маршрутизації на стороні клієнта (Frontend routing) для навігації між модулями системи.

8) Реалізація функціоналу для менеджерів: форми створення, редагування та видалення задач/проектів, тестування коректності оновлення даних.
9) Реалізація системи авторизації користувачів із розподілом ролей (менеджер, розробник, лід, клієнт).

1.4 Вибір програмного забезпечення

В якості інтегрованого середовища розробки, IDE обрано Visual Studio Code (VS Code). Цей редактор коду є легковаговим, але потужним кросплатформним інструментом, що підтримує широкий спектр розширень для Python та Django [1]. Його вибір обґрунтовано наявністю вбудованого терміналу, зручних засобів для дебагінгу та глибокої інтеграції з системою контролю версій Git, що значно підвищує продуктивність написання та підтримки коду [2].

Дані проекту зберігаються в об’єктно-реляційній базі даних, для чого було обрано PostgreSQL. Ця СУБД забезпечує надійність збереження даних (ACID-сумісність), підтримку складних транзакцій та високу продуктивність при роботі з великими обсягами інформації [3]. Для зручного адміністрування, візуалізації структури бази даних та виконання SQL-запитів використовується графічний клієнт PgAdmin, який дозволяє ефективно керувати сервером PostgreSQL та моніторити його стан без необхідності постійного використання командного рядка [4].

Серверна частина розробляється на базі фреймворку Django (мова програмування Python). Django є високорівневим веб-фреймворком, що дотримується архітектурного патерну MTV (Model-Template-View) та принципу «batteries included» [5]. Це забезпечує вбудований захист від поширених веб-атак, таких як CSRF та SQL Injection. Взаємодія з базою даних реалізується через Django ORM, що дозволяє маніпулювати даними як об'єктами Python, абстрагуючись від синтаксису чистого SQL [6]. Для генерації HTML-сторінок використовується шаблонний рушій Jinja, який

було обрано замість стандартного DTL через його вищу швидкість рендерингу, гнучкість та розширені можливості для створення модульних шаблонів [7].

Клієнтський інтерфейс створюється з використанням CSS-фреймворку Bootstrap. Вибір Bootstrap обумовлений його потужною системою сіток (Grid System), яка дозволяє легко реалізувати адаптивний веб-дизайн (RWD) для коректного відображення системи як на десктопах, так і на мобільних пристроях [8]. Використання готових компонентів Bootstrap (форм, таблиць, модальних вікон) дозволяє стандартизувати зовнішній вигляд платформи та значно скоротити час на стилізацію інтерфейсу.

2 РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА (UI/UX ДИЗАЙН)

2.1 Розробка прототипу інтерфейсу користувача

Мокап — це статичне візуальне представлення інтерфейсу користувача, що демонструє основну структуру екрану програми чи вебсайту. Мокап дозволяє швидко побачити загальне компонування елементів інтерфейсу без інтерактивних функцій, що є основою для подальшого опрацювання дизайну та верстки.

Для огляду узятий мокап головної робочої панелі користувача (Dashboard). На макеті показано місце для сайдбару (зліва), в якому розташоване меню навігації по командах та проєктах, а також блок з персональною інформацією користувача (ім'я, роль, пошта). У верхній частині розміщено шапку сторінки (header) з кнопками перемикання основних розділів («Завдання», «Заходи» тощо) та іконкою сповіщень.

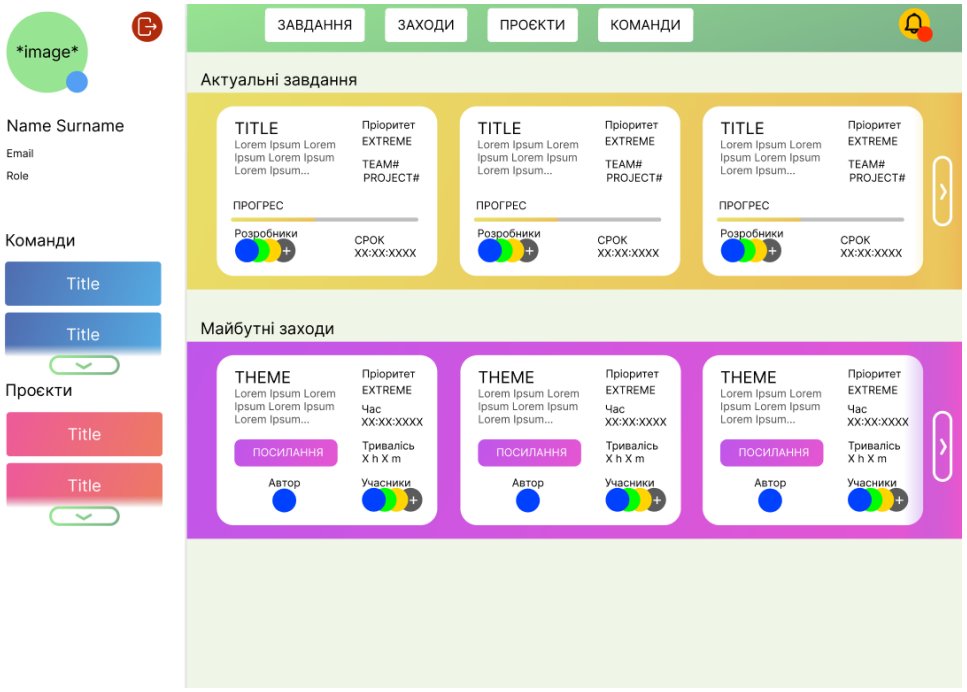

Рисунок 2.1 – Мокап дашборд

Також, був створенний мокап стандартної сторінки авторизації:

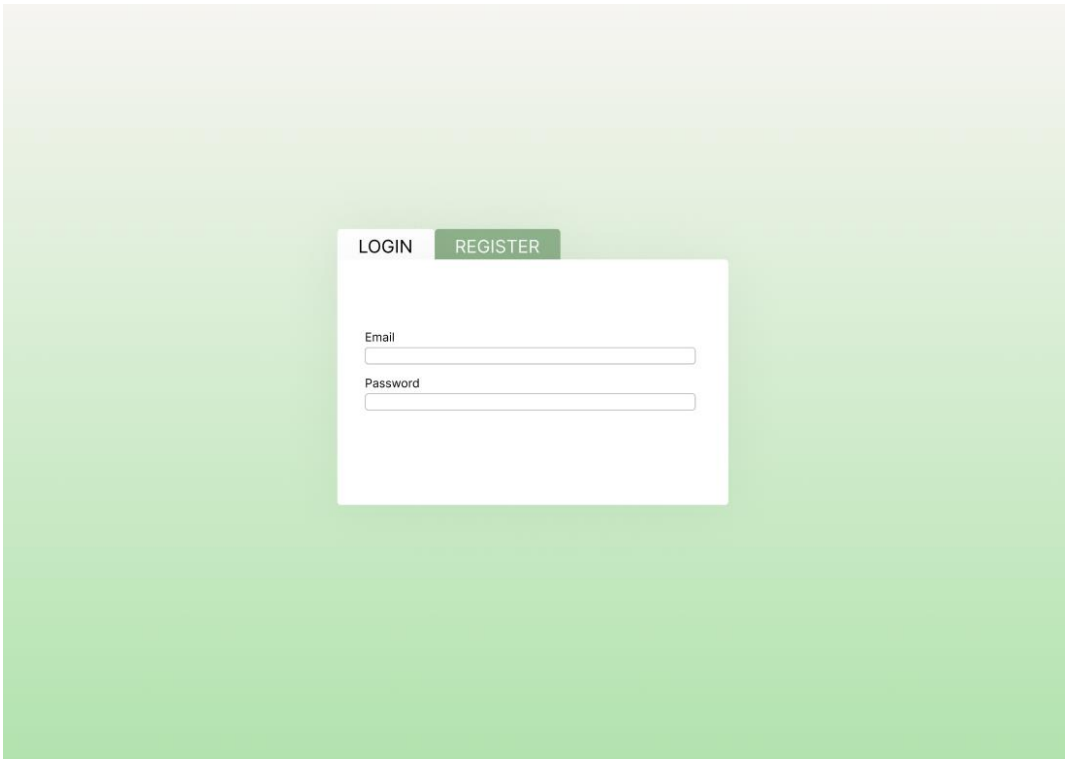

Рисунок 2.2 — Мокап сторінки авторизації

3 РОЗРОБКА БАЗИ ДАННИХ

3.1 Аналіз вимог до бази даних

Для забезпечення надійного зберігання, швидкої обробки та централізованого управління інформаційними потоками в системі необхідно розробити відповідну архітектуру бази даних. У контексті Task Manager база даних служить фундаментом для зберігання інформації про структуру команд, технічні завдання, спринти, історію комунікації (Q/A) та звіти про виконання робіт. Етап проектування включає аналіз предметної області, вибір реляційної моделі (PostgreSQL), визначення зв'язків між сутностями (проекти, виконавці, задачі) та налаштування інструментів ORM для безпечної взаємодії з даними.

У цьому пункті розглянуті ключові вимоги до бази даних, розділивши їх на функціональні та нефункціональні.

Функціональні вимоги:

1) Гнучкість вибірок та фільтрації. Підтримка складних запитів для формування дашбордів та Канбан-дошки. Це включає можливість миттєвого отримання списку завдань з урахуванням множинних фільтрів: за статусом (To Do, In Progress), пріоритетом, виконавцем та дедлайном.
2) Рольова модель доступу (RBAC). База даних повинна зберігати не лише облікові дані користувачів, а й інформацію про їхні ролі (Manager, Developer, Client) та права доступу, що дозволяє динамічно обмежувати видимість певних проектів чи функцій редагування.

Нефункціональні вимоги:

1) Цілісність даних (Data Integrity). Забезпечення узгодженості даних при одночасній роботі всієї команди. Наприклад, при переміщенні картки на

дошці статус повинен оновлюватись атомарно, запобігаючи конфліктам редагування.

2) Безпека та конфіденційність. Оскільки система містить технічну документацію та комерційну інформацію, необхідно реалізувати захист від несанкціонованого доступу та атак (SQL-ін’єкцій) на рівні ORM, а також шифрування чутливих даних (паролів, токенів доступу).
3) Масштабованість та архівування. Архітектура бази даних повинна передбачати можливість зростання кількості записів (історії виконаних завдань, логів активності) без суттєвого зниження швидкодії системи, з можливістю подальшого архівування завершених проектів.

3.2 Обґрунтування вибору засобів розробки

Для реалізації серверної частини та збереження даних було обрано об’єктно-реляційну систему управління базами даних PostgreSQL. Вибір цієї СУБД обґрунтовано її повною відповідністю стандартам ACID, високою продуктивністю при паралельній обробці транзакцій (MVCC) та розвиненою системою безпеки. Ключовою особливістю, що визначила вибір PostgreSQL для даного проекту, є потужна підтримка Role-Based Access Control (RBAC) та політик захисту на рівні рядків (Row-Level Security) [10]. Це дозволяє розмежувати права доступу до даних не лише на рівні логіки програми, а й безпосередньо на рівні ядра бази даних.

Для взаємодії між веб-застосунком Django та базою даних обрано драйвер psycopg2. Це найпопулярніший і найстабільніший адаптер для Python, який забезпечує низькорівневий контроль над з'єднанням. Використання psycopg2 у даному проекті є критично важливим, оскільки воно дозволяє реалізувати специфічну архітектуру безпеки: замість використання одного стандартного користувача для всіх запитів, система динамічно ініціалізує з'єднання або перемикає контекст виконання під

конкретну роль PostgreSQL, що відповідає поточному користувачеві системи (наприклад, developer_role, manager_role).

Такий підхід гарантує, що навіть у випадку компрометації програмного коду Django, зловмисник не зможе виконати SQL-запити, які виходять за межі повноважень конкретної ролі бази даних. При цьому, для побудови запитів та маніпуляції даними використовується високорівневий Django ORM, що забезпечує швидкість розробки та зручність підтримки коду, працюючи поверх безпечного з'єднання, встановленого через psycopg2.

3.3 Проектування ER-діаграми

ER-діаграма (Entity-Relationship Diagram) розроблена для візуалізації схеми бази даних і відображення ключових сутностей, їх атрибутів та реляційних зв’язків у системі управління проектами. Ця модель дозволяє чітко структурувати інформаційні потоки та зрозуміти логіку взаємодії компонентів системи (таких як співробітники, команди, проекти, спринти, технічні завдання та звіти). Її побудова забезпечує дотримання цілісності даних на рівні СУБД та є фундаментом для коректної реалізації рольової моделі доступу і бізнес-логіки менеджменту ресурсів.

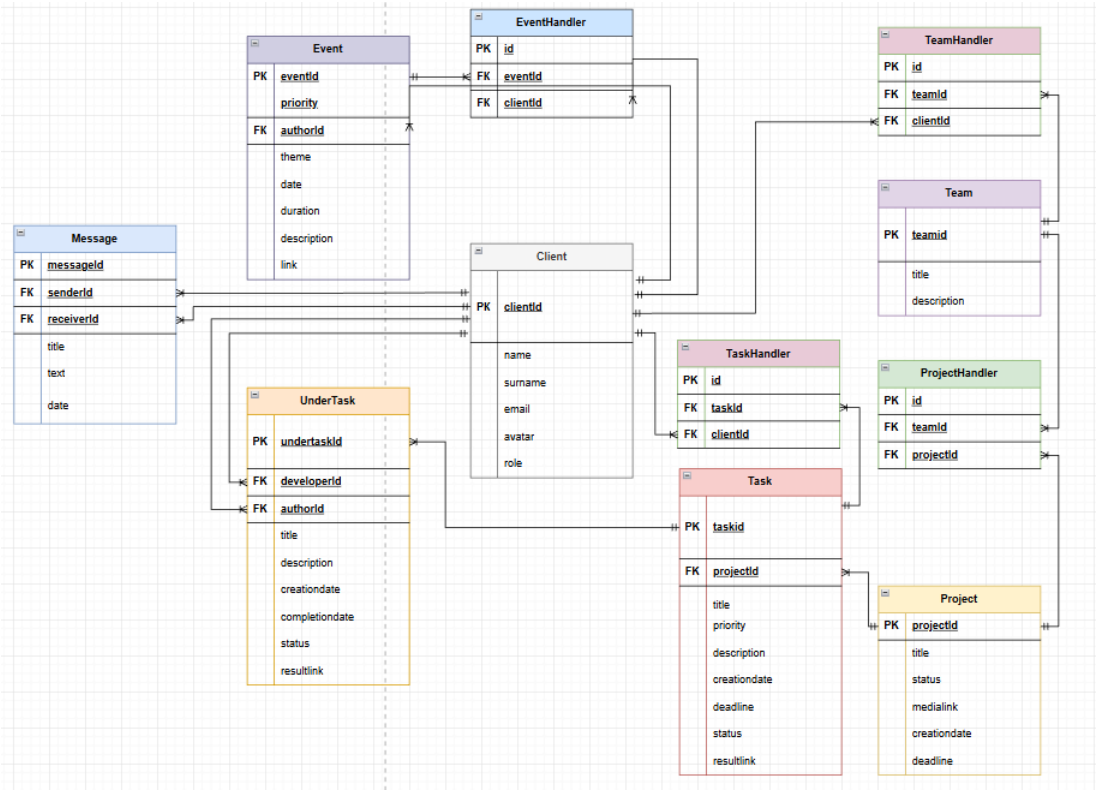

Рисунок 3.1 — ER-діаграма для предметної області «Організація роботи команди розробників»

Опис сутностей БД:

Ось опис сутностей вашої ER-діаграми, виконаний за наведеним зразком, із додаванням таблиці сесій:

- Client — центральна сутність системи, що містить персональні дані користувача (ім'я, прізвище, email, посилання на аватар) та визначає його роль у системі (менеджер, розробник тощо).
- Team — відображає команду розробників з її назвою та описом, слугує контейнером для групування співробітників.
- TeamHandler — проміжна сутність для реалізації зв'язку «багато-до-багатьох», що фіксує приналежність конкретних клієнтів (співробітників) до певних команд.

- Project — зберігає інформацію про проекти, над якими ведеться робота, включаючи назву, поточний статус, дедлайн та дату створення.
- ProjectHandler — пов'язує команди з проектами, визначаючи, яка саме команда відповідальна за виконання конкретного проекту.
- Task — представляє основну одиницю роботи (технічне завдання) в межах проекту, містить пріоритет, опис, статус, посилання на результат та часові рамки.
- TaskHandler — відповідає за розподіл навантаження, зв'язуючи задачу з конкретним виконавцем (Client).
- UnderTask — деталізує основну задачу на менші підзадачі, зберігаючи дані про виконавця, автора, статус виконання та результати роботи.
- Event — описує заплановані заходи (мітинги, Q/A сесії) з атрибутами теми, дати, тривалості, пріоритету та посиланням на зустріч.
- EventHandler — таблиця учасників, яка пов'язує користувачів із запланованими заходами (Events).
- Message — зберігає історію комунікації в робочому чаті, включаючи текст повідомлення, тему, дату відправлення та зв'язки з відправником і отримувачем.
- django_session — службова таблиця фреймворку Django, яка забезпечує зберігання даних сесій користувачів, необхідних для підтримки авторизації та збереження стану між HTTP-запитами.

3.4 Проектування даних у коді

Опис даних таблиць зіставлений у поданій таблиці:

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

Продовження таблиці 3.1:

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
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

Продовження таблиці 3.1:

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

3.5 Програмна реалізація БД

За поданими критеріями та обмеженнями атрибутів, був сформований поданий SQL код:

```
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
);
```

Лістинг 3.1 — SQL код створення таблиць

Потрібно відмітити, що через фізичне розділення БД та серверу моноліту із Django, міграції створюються на основі вже існуючого SQL коду.

3.6 Підключення до бази данних

Лістинг коду демонструє реалізацію алгоритму автентифікації користувачів, який базується на перевірці облікових даних безпосередньо на рівні системи управління базами даних PostgreSQL. На відміну від стандартних методів Django, де паролі звіряються з хешами у таблиці, даний підхід ініціює реальне з'єднання з БД для підтвердження прав доступу.

```
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')
```

Лістинг 3.2 — Код auth.py

Функція custom_login виконує ключову роль у цьому процесі та реалізує наступну логіку:

1) Отримання даних: Обробляє POST-запит, витягуючи email та пароль, введені користувачем.

2) Ідентифікація клієнта: Здійснює пошук користувача в моделі Client за вказаною поштою для отримання його унікального ідентифікатора (clientid).
3) Формування контексту БД: На основі ID користувача динамічно генерується ім'я користувача бази даних (формат user_{id}), яке відповідає попередньо створеній ролі в PostgreSQL.
4) Валідація через з'єднання: Використовуючи бібліотеку psycopg, система намагається встановити тестове з'єднання з базою даних, використовуючи згенероване ім'я ролі та введений пароль.
    - Якщо з'єднання успішне (pass), це підтверджує, що користувач знає пароль від своєї ролі в БД.
    - У разі помилки psycopg.OperationalError, система ідентифікує це як невірний пароль.
5) Ініціалізація сесії: Після успішної перевірки у сесію (request.session) записуються ключові параметри: ID користувача, його роль (db_role) та ім'я користувача БД (db_user). Ці дані надалі використовуються для виконання запитів під відповідними правами.
6) Функція logout_view відповідає за коректне завершення роботи. Вона виконує скидання поточної ролі (RESET ROLE) через курсор з'єднання та повністю очищає дані сесії (request.session.flush()), перенаправляючи користувача на сторінку входу.

Такий підхід забезпечує високий рівень безпеки, оскільки паролі не зберігаються в додатку, а валідація делегується безпосередньо ядру СУБД.

3.7 Висновки до частини «Розробка бази даних»

У даному розділі було спроектовано та реалізовано архітектуру бази даних інформаційної системи Task Manager, яка виступає фундаментом для надійного зберігання та обробки даних проекту. В ході виконання роботи було вирішено наступні задачі:

1) Обґрунтовано вибір технологічного стеку. Вибір СУБД PostgreSQL дозволив забезпечити високу продуктивність транзакцій (MVCC) та відповідність стандартам ACID. Використання драйвера psycopg у зв'язці з Django ORM забезпечило необхідний баланс між зручністю розробки та низькорівневим контролем над з'єднаннями.
2) Спроектовано структуру даних. Розроблена ER-діаграма детально відображає предметну область, охоплюючи зв'язки між ключовими сутностями (Команди, Проекти, Задачі, Заходи). Нормалізація бази даних та використання проміжних таблиць (Handler-сутностей) дозволили реалізувати гнучкі зв'язки «багато-до-багатьох» та забезпечити цілісність даних.
3) Реалізовано просунуту модель безпеки. Ключовою особливістю розробки стала імплементація Role-Based Access Control (RBAC) безпосередньо на рівні бази даних. Механізм автентифікації, реалізований через спробу прямого з'єднання під роллю користувача, гарантує, що права доступу контролюються ядром СУБД, а не лише логікою веб-додатку. Це мінімізує ризики SQL-ін'єкцій та несанкціонованого доступу до конфіденційної інформації.
4) Налаштовано програмну взаємодію. Реалізовано алгоритми динамічного перемикання контексту БД в залежності від авторизованого користувача, що дозволяє системі масштабуватися та ефективно обробляти запити від різних ролей (менеджерів, розробників) в межах однієї платформи.

Таким чином, розроблена інформаційна система повністю відповідає функціональним та нефункціональним вимогам, забезпечуючи безпеку, масштабованість та швидкодію, необхідні для подальшої реалізації бізнес-логіки серверної частини.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ МОНОЛІТУ

4.1 Опис архітектури серверної частити моноліту

У цьому підрозділі описана загальна архітектура серверної частини додатку «Task Manager». Розроблена система базується на монолітній архітектурі (Monolithic Architecture), де всі функціональні компоненти (інтерфейс, бізнес-логіка, робота з даними) об'єднані в єдиний програмний модуль. В якості основного архітектурного патерну використано MVT (Model-View-Template), який є стандартом для фреймворку Django і забезпечує чітке розділення відповідальності між компонентами системи.

4.2 Шар даних

Шар даних у розроблюваній системі відповідає за визначення структури інформації, її валідацію та взаємодію з системою управління базами даних PostgreSQL. У фреймворку Django цей шар реалізовано через механізм Models та ORM (Object-Relational Mapping), що дозволяє описувати таблиці бази даних як класи Python, а записи — як об'єкти цих класів.

Такі класи зберігаються в файлі 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'
```

Лістинг 4.1 — Зміст файлу models.py

Задля створення міграцій необхідно ввести в віртуальному середовищі таку команду: python manage.py migrate --fake initial

4.3 Шар бізнес-логіки

Шар бізнес-логіки у додатку реалізовано з використанням функціональних представлень (Function-Based Views) фреймворку Django. Для забезпечення модульності та зручності підтримки коду, логіка розділена на окремі файли в пакеті

core/views, які імпортуються через __init__.py. Кожен модуль відповідає за конкретну доменну область системи.

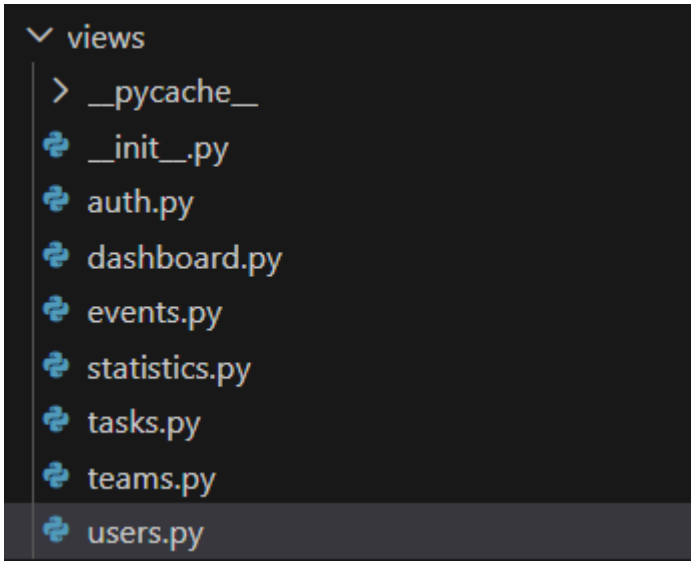

Рисунок 4.1 — Директорія views

```
    from .auth import custom_login, logout_view
    from .dashboard import dashboard
    from .tasks import (
        create_task,   edit_task,   update_task_status,   delete_task,
cancel_task,
        submit_result, add_executor, remove_executor, reject_task,
        create_undertask,                    submit_undertask_result,
update_undertask_status, reject_undertask
    )
    from .events import (
        create_event, delete_event, get_event_participants,
        remove_event_participant, edit_event, add_event_participant
    )
    from .teams import (
        create_team, add_team_member, remove_team_member, delete_team,
get_team_members,
        create_project,        complete_project,        delete_project,
get_project_details, add_project_team, remove_project_team
    )
    from  .users  import  add_user,  delete_user,  change_user_role,
send_message
    from .statistics import view_statistics
```

```
__all__ = [
    'custom_login',
    'logout_view',
    'dashboard',
    'create_task',
    'edit_task',
    'update_task_status',
    'delete_task',
    'cancel_task',
    'reject_task',
    'submit_result',
    'add_executor',
    'remove_executor',
    'create_undertask',
    'submit_undertask_result',
    'update_undertask_status',
    'reject_undertask',
    'create_event',
    'delete_event',
    'get_event_participants',
    'remove_event_participant',
    'edit_event',
    'add_event_participant',
    'create_team',
    'add_team_member',
    'remove_team_member',
    'delete_team',
    'get_team_members',
    'create_project',
    'complete_project',
    'delete_project',
    'get_project_details',
    'add_project_team',
    'remove_project_team',
    'add_user',
    'delete_user',
    'change_user_role',
    'send_message',
    'view_statistics',
]
```

Лістинг 4.2 — __init__.py

Модуль автентифікації auth.py уже був описаний в 3 пункті.

Модуль основного інтерфейсу (dashboard.py) Відповідає за формування головного робочого простору користувача.

- Функція dashboard агрегує дані з різних моделей (Task, Project, Event, Message) в єдиний контекст.
- Реалізовано гнучку логіку фільтрації даних: залежно від ролі користувача (Lead, Developer, Individual), система відображає лише релевантні завдання та проекти.
- Обробляються GET-параметри для сортування завдань за пріоритетом, статусом та дедлайнами («термінові», «скоро», «пізніше»).

```
    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).select_relat
ed('project')
                context['undertasks']                                       =
UnderTask.objects.filter(developer__clientid=user_id).select_related('t
ask',

'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)
```

Лістинг 4.3 — вміст файлу dashboard.py

Модуль управління завданнями (tasks.py) Містить найскладнішу бізнес-логіку проекту, пов'язану з життєвим циклом технічних завдань.

- Використання транзакцій: Функції створення (create_task) та редагування завдань загорнуті у блок transaction.atomic(), що забезпечує цілісність

даних при одночасному оновленні кількох таблиць (завдання та його виконавці).

- Реалізовано повний набір дій (CRUD): створення, редагування, зміна статусу, скасування (cancel_task), відхилення (reject_task) та подання звіту про виконання (submit_result).
- Логіка підзавдань: Окремі функції відповідають за декомпозицію завдань (create_undertask), що дозволяє створювати ієрархічну структуру робіт.

```
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()

                    user_role = request.session.get('db_role')
                    user_id = request.session.get('user_id')

                    assignee = form.cleaned_data.get('assignee')
                    if user_role == 'Individual':
                        TaskHandler.objects.create(task=task,
client_id=user_id)

                    if assignee:

                        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')
```

Лістинг 4.4 — вміст файлу tasks.py

Модуль адміністрування користувачів (users.py) Цей модуль демонструє пряму взаємодію з системою керування доступом PostgreSQL (DCL - Data Control Language).

- Функції add_user, delete_user та change_user_role виконують сирі SQL-запити через connection.cursor().
- Створення користувача супроводжується виконанням команди CREATE ROLE ... WITH LOGIN, а зміна ролі — командами REVOKE та GRANT. Це забезпечує синхронізацію користувачів додатку з реальними ролями бази даних.

```
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')
```

Лістинг 4.4 — Вміст файлу users.py

Модуль управління ресурсами (teams.py, events.py) Забезпечує організаційну складову роботи команди.

- Функції create_team, add_team_member, create_project містять перевірки прав доступу (доступні лише для ролей Lead та Manager).
- Реалізовано зв'язок «багато-до-багатьох» між командами та проектами через проміжні сутності (ProjectHandler, TeamHandler).
- Модуль подій (events.py) дозволяє створювати зустрічі та динамічно керувати списком учасників, включаючи API-endpoint get_event_participants для асинхронного отримання даних на фронтенді.

```
    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('client')
            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('team')
            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')
```

Лістинг 4.5 — Вміст файлу teams.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('client')
            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')
```

Лістинг 4.6 — Вміст файлу events.py

Модуль аналітики (statistics.py) Відповідає за збір та обрахунок KPI (ключових показників ефективності).

- Логіка розділена за ролями: для Lead збирається статистика по всім командам та проектам; для Manager — ефективність окремих розробників; для Individual — особистий прогрес виконання завдань (співвідношення виконаних до запланованих).

```
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)
```

Лістинг 4.7 — Вміст файлу statistics.py

Така архітектура Views забезпечує чіткий розподіл відповідальності, безпеку виконання операцій на рівні БД та гнучкість у налаштуванні інтерфейсу під різні ролі користувачів.

4.4 Шар представлень

Шар представлення у системі відповідає за візуалізацію даних, отриманих від серверної частини, та забезпечення інтерактивної взаємодії з користувачем. Замість стандартного шаблонного рушія Django Templates (DTL), у проекті інтегровано Jinja2. Цей вибір обґрунтований вищою швидкістю рендерингу, розширеними можливостями успадкування шаблонів та синтаксисом, максимально наближеним до чистого Python.

Архітектура фронтенду побудована за компонентним принципом, що дозволяє уникнути дублювання коду та спростити підтримку інтерфейсу. Файлова структура, представлена в директорії templates, чітко розділена на логічні блоки:

Основні сторінки (Layouts & Pages):

- login.html — автономна сторінка для автентифікації користувачів, яка не містить елементів навігації робочого простору.
- dashboard.html — головний шаблон системи (точка входу), який слугує контейнером для підключення інших інтерфейсних модулів. Він формує загальну сітку сторінки (Grid Layout).
- navbar.html — навігаційна панель, що відображає профіль користувача та меню, і включається на всіх внутрішніх сторінках.

Інтерфейсні компоненти (Includes): Для забезпечення читабельності коду dashboard.html, окремі секції інтерфейсу винесені у незалежні файли, які підключаються через директиву Jinja2 {% include %}:

- kanban_board.html — відповідає за відображення завдань у колонках (To Do, In Progress, Review, Done) та логіку карток (task_card.html).
- chat_section.html та messages.html — реалізують інтерфейс робочого чату.
- events_section.html — блок для відображення списку запланованих заходів.
- statistics.html та stats.html — компоненти для візуалізації KPI користувача або команди.

Модальні вікна (Modals): Усі спливаючі форми винесені в окрему директорію templates/modals. Це дозволяє динамічно завантажувати їх або приховувати без перезавантаження основної сторінки.

- Управління завданнями: task_modal.html (створення), edit_task_modal.html (редагування), cancel_task_modal.html (скасування), submit_modal.html (здача результату).
- Управління ресурсами: team_modal.html, project_modal.html, user_modal.html — форми для адміністративних дій менеджерів.
- Комунікація: message_modal.html — форма відправки повідомлень.

Використання можливостей Jinja2

Інтеграція Jinja2 дозволила реалізувати складну логіку відображення безпосередньо в шаблонах:

- Цикли та умови: Використання конструкцій {% for task in tasks %} та {% if role == 'Manager' %} дозволяє динамічно генерувати контент залежно від переданого контексту та прав доступу користувача.
- Фільтри: Застосування кастомних фільтрів для форматування дат (наприклад, дедлайнів) та статусів завдань.

- Макроси: Повторювані елементи (наприклад, картка завдання або кнопка дії action_buttons.html) можуть використовуватися багаторазово з різними параметрами.

Така організація шару представлення забезпечує модульність, легкість масштабування та високу швидкість відгуку інтерфейсу.

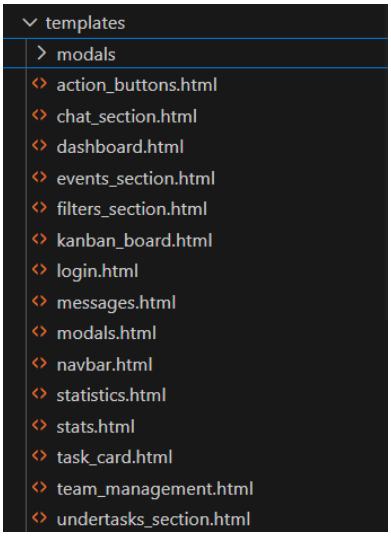

Рисунок 4.2 — Структура templates

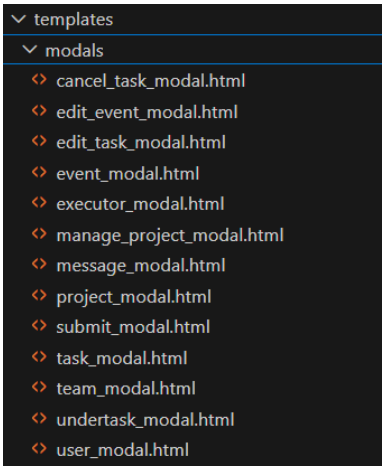

Рисунок 4.3 — Структура templates\modals

4.5 Реалізація клієнтської логіки

Для забезпечення інтерактивності інтерфейсу без повного перезавантаження сторінки використовується клієнтський JavaScript, розміщений у файлі static/js/dashboard.js. Логіка скриптів реалізує взаємодію з DOM-деревом, асинхронні запити до сервера та управління модальними вікнами Bootstrap.

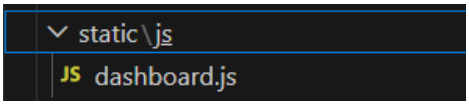

Рисунок 4.4 — Структура static\js

Основні функціональні блоки клієнтської частини:

1) Kanban Drag-and-Drop: Реалізовано нативний HTML5 Drag API через функції drag(ev) та drop(ev, newStatus). Це дозволяє користувачеві змінювати статус завдання простим перетягуванням картки між колонками. При події drop автоматично заповнюється прихована форма та відправляється POST-запит на оновлення статусу.
2) Асинхронна робота з даними (AJAX/Fetch): Для підвищення швидкодії системи реалізовано динамічне підвантаження даних без перезавантаження сторінки:
    - Функції openManageTeamModal(teamId) та openManageProjectModal(projectId) використовують fetch API для отримання списку учасників команди або деталей проекту у форматі JSON.
    - Отримані дані динамічно рендеряться в HTML-таблицю модального вікна через маніпуляції з innerHTML.
3) Динамічна фільтрація: Функція applyFilters() зчитує значення з полів фільтрації (проект, пріоритет, дедлайн) та формує GET-запит за

допомогою URLSearchParams, оновлюючи поточну URL-адресу сторінки. Це дозволяє зберігати стан фільтрів при навігації або передачі посилання.

4) Управління модальними вікнами: Використовується JavaScript API бібліотеки Bootstrap (new bootstrap.Modal(...)) для програмного відкриття вікон редагування завдань (openEditModal), подій (openEditEventModal) та призначення виконавців. Скрипт автоматично заповнює поля форм поточними даними перед відображенням вікна.
5) Безпека та валідація: У скриптах реалізовано підтвердження критичних дій (наприклад, confirmDeleteProject), що запобігає випадковому видаленню даних. Також при відправці асинхронних форм (наприклад, видалення учасника) скрипт автоматично додає CSRF-токен для захисту від міжсайтової підробки запитів.

4.6 Роутинг проекту

Система маршрутизації (URL Dispatcher) у Django відповідає за зіставлення запитуваних URL-адрес із відповідними функціями-обробниками (Views). Конфігурація маршрутів реалізована у файлі urls.py модуля core, де визначено список urlpatterns. У проекті використано чітку ієрархічну структуру шляхів, що полегшує навігацію та підтримку API.

Усі URL-адреси можна розділити на кілька логічних груп, що відповідають модулям бізнес-логіки:

1) Точки входу та аутентифікація:
   - path('', ...) — Кореневий маршрут, який веде на сторінку входу (custom_login). Це означає, що неавторизований користувач одразу потрапляє на екран логіну.

- path('dashboard/', ...) — Головна сторінка системи, куди користувач перенаправляється після успішної авторизації.
- path('logout/', ...) — Маршрут для завершення сесії та очищення контексту ролі БД.

2) Управління завданнями (Task Management): Для роботи із завданнями виділено групу маршрутів з префіксом task/. Вони покривають повний життєвий цикл задачі:
   - Створення та редагування: create/, edit/.
   - Зміна станів: update/ (переміщення по канбан-дошці), submit/ (здача роботи), reject/ (відхилення), cancel/ (скасування).
   - Управління виконавцями: add_executor/, remove_executor/. Аналогічна логіка реалізована для підзавдань через префікс undertask/.
3) Адміністративні маршрути (Resource Management): Ці маршрути призначені для менеджерів та тімлідів і дозволяють керувати структурою організації:
   - user/ — додавання нових співробітників, видалення та зміна їхніх ролей у базі даних (change_role).
   - team/ та project/ — створення та видалення команд і проектів, а також керування їхніми зв'язками (add_team, remove_team).
4) Комунікація та події:
   - event/ — маршрути для CRUD-операцій з подіями календаря.
   - message/send/ — ендпоінт для відправки повідомлень у робочому чаті.
5) Особливістю даної конфігурації є наявність динамічних маршрутів, які використовуються клієнтським JavaScript (розділ 4.5) для отримання

даних у форматі JSON без перезавантаження сторінки. Вони використовують конвертер шляху <int:parameter>:

- path('event/participants/<int:event_id>/', ...) — повертає список учасників конкретного заходу.
- path('team/members/<int:team_id>/', ...) — повертає склад команди для відображення у модальному вікні.
- path('project/details/<int:project_id>/', ...) — повертає детальну інформацію про проект та прикріплені до нього команди.

Для кожного шляху задано параметр name (наприклад, name='login', name='create_task'). Це дозволяє використовувати механізм реверсивної генерації URL (Reverse URL resolution) у шаблонах Jinja2 (наприклад, {{ url('dashboard') }}) та у коді Python (redirect('dashboard')). Такий підхід робить систему гнучкою: при зміні URL-адреси у файлі urls.py не потрібно правити посилання у всьому проекті.

```
from django.urls import path
from core import views

urlpatterns = [
    path('', views.custom_login, name='login'),
    path('dashboard/', views.dashboard, name='dashboard'),
    path('logout/', views.logout_view, name='logout'),
    path('statistics/', views.view_statistics, name='statistics'),

    path('task/create/', views.create_task, name='create_task'),
    path('task/edit/', views.edit_task, name='edit_task'),
    path('task/update/',                views.update_task_status,
name='update_task'),
    path('task/reject/', views.reject_task, name='reject_task'),
    path('task/delete/', views.delete_task, name='delete_task'),
    path('task/submit/',                    views.submit_result,
name='submit_result'),
    path('task/cancel/', views.cancel_task, name='cancel_task'),
    path('task/add_executor/',              views.add_executor,
name='add_executor'),
    path('task/remove_executor/',          views.remove_executor,
name='remove_executor'),
```

```
        path('undertask/create/',              views.create_undertask,
name='create_undertask'),
        path('undertask/submit/',       views.submit_undertask_result,
name='submit_undertask'),
        path('undertask/update/',       views.update_undertask_status,
name='update_undertask'),
        path('undertask/reject/',              views.reject_undertask,
name='reject_undertask'),

        path('event/create/',                      views.create_event,
name='create_event'),
        path('event/edit/', views.edit_event, name='edit_event'),
        path('event/delete/',                      views.delete_event,
name='delete_event'),
        path('event/participants/<int:event_id>/',
views.get_event_participants, name='event_participants'),
        path('event/add_participant/',    views.add_event_participant,
name='add_event_participant'),
        path('event/remove_participant/',
views.remove_event_participant, name='remove_event_participant'),

        path('message/send/',                      views.send_message,
name='send_message'),

        path('user/add/', views.add_user, name='add_user'),
        path('user/delete/', views.delete_user, name='delete_user'),
        path('user/change_role/',              views.change_user_role,
name='change_user_role'),

        path('team/create/', views.create_team, name='create_team'),
        path('team/delete/', views.delete_team, name='delete_team'),
        path('team/members/<int:team_id>/',    views.get_team_members,
name='get_team_members'),
        path('team/add_member/',               views.add_team_member,
name='add_team_member'),
        path('team/remove_member/',          views.remove_team_member,
name='remove_team_member'),

        path('project/create/',                  views.create_project,
name='create_project'),
        path('project/complete/',              views.complete_project,
name='complete_project'),
        path('project/delete/',                  views.delete_project,
name='delete_project'),
```

```
        path('project/details/<int:project_id>/',
views.get_project_details, name='get_project_details'),
        path('project/add_team/',              views.add_project_team,
name='add_project_team'),
        path('project/remove_team/',        views.remove_project_team,
name='remove_project_team'),
    ]
```

Лістинг 4.8 — Вміст файлу urls.py

4.7 Висновки розділу «Програмна реалізація серверної частини моноліту»

У даному розділі здійснено програмну реалізацію серверної та клієнтської частин інформаційної системи «Task Manager» на базі фреймворку Django. В ході роботи було досягнуто наступних результатів:

Реалізовано надійну архітектуру додатку. Побудовано монолітну систему за патерном MVT (Model-View-Template), яка забезпечує чітке розмежування логіки обробки даних, бізнес-правил та інтерфейсу користувача. Це дозволяє легко масштабувати та підтримувати код у майбутньому.

Імплементовано гібридну модель роботи з даними. Поєднано зручність Django ORM для стандартних операцій з потужністю прямих SQL-запитів через драйвер psycopg. Реалізовано унікальний механізм автентифікації та авторизації, де права доступу контролюються безпосередньо ролями PostgreSQL, що забезпечує максимальний рівень захисту даних від несанкціонованого доступу.

Розроблено складну бізнес-логіку. Програмно реалізовано ключові процеси управління проектами: життєвий цикл завдань (від створення до рев'ю), менеджмент команд та ресурсів, організація заходів та внутрішня комунікація. Використання транзакцій (transaction.atomic) гарантує цілісність даних при виконанні складних операцій.

Створено сучасний інтерактивний інтерфейс. Інтеграція шаблонного рушія Jinja2 забезпечила високу швидкість рендерингу сторінок. Реалізація клієнтської

логіки на JavaScript (Fetch API, Drag-and-Drop) дозволила створити динамічні елементи, такі як Канбан-дошка та модальні вікна, що значно покращує досвід користувача (UX) та наближає веб-додаток до рівня SPA (Single Page Application).

Налаштовано ефективну маршрутизацію. Розроблена структура URL-адрес забезпечує логічну навігацію системою та надає API-ендпоінти для асинхронної взаємодії з фронтендом.

Таким чином, програмна реалізація повністю відповідає поставленим технічним вимогам, забезпечуючи стабільну, безпечну та зручну роботу користувачів із різними ролями в системі управління проектами.

ВИСНОВКИ

Розробка інформаційної системи для організації роботи команди розробників дозволила створити сучасне та ефективне рішення, яке поєднує потужний інструментарій для менеджменту та зручність використання. В ході роботи було реалізовано всі етапи проєктування та впровадження системи: від аналізу предметної області та побудови ER-діаграм до програмної реалізації серверної логіки на Django та інтерактивного клієнтського інтерфейсу. Особливу увагу приділено забезпеченню цілісності даних, розмежуванню прав доступу та ергономіці робочого простору, що є критично важливими факторами для продуктивної роботи IT-команд.

Ключовим результатом стало створення гнучкої архітектури на базі патерну MVT, яка дозволяє легко масштабувати систему та адаптувати її до специфічних процесів розробки (Scrum, Kanban). Інтерфейс платформи, реалізований з використанням Jinja2 та динамічних JS-компонентів, орієнтований на інтуїтивність та швидкість реакції, що сприяє підвищенню ефективності виконавців. Удосконалені механізми взаємодії з базою даних PostgreSQL забезпечують надійне зберігання історії проекту та швидкий доступ до технічної документації.

Ще одним важливим досягненням стало впровадження просунутої моделі безпеки, що базується на використанні нативних ролей СУБД та прямого підключення через драйвер. Такий підхід гарантує захист конфіденційної інформації та мінімізує ризики несанкціонованого доступу. Завдяки інтеграції засобів комунікації (чат, заходи) та аналітики, платформа відповідає сучасним вимогам ринку та демонструє високу якість реалізації. Це рішення може стати корисним не лише для невеликих стартапів, але й для розподілених команд розробників у великих компаніях.

Успіх проєкту підтверджує важливість системного підходу до автоматизації процесів розробки, де технічна реалізація тісно пов'язана з потребами бізнес-логіки. Розробка даного проекту має значні перспективи на майбутнє. Архітектура системи

спроектована таким чином, що в неї можна інтегрувати нові модулі, наприклад, автоматизований CI/CD моніторинг або розширену статистику ефективності. Тому я планую продовжувати розвивати цей проект, розширювати його функціонал і доводити його до рівня повноцінного комерційного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Visual Studio Code Documentation. User Guide [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com/docs.

2 Del Sole A. Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux. – Apress, 2019. – 185 p.

3 PostgreSQL Global Development Group. PostgreSQL 16.1 Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/.

4 Riggs S., Ciolino G. PostgreSQL 10 Administration Cookbook. – Packt Publishing, 2018. – 462 p.

5 Holovaty A., Kaplan-Moss J. The Definitive Guide to Django: Web Development Done Right. – Apress, 2009. – 447 p.

6 Django Software Foundation. Django Documentation: Models and Databases [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/en/5.0/topics/db/.

7 Ronacher A. Jinja2 Documentation (Version 3.1.x) [Електронний ресурс]. – Режим доступу: https://jinja.palletsprojects.com/.

8 Spurlock J. Bootstrap: Responsive Web Development. – O'Reilly Media, 2013.

9 Sommerville I. Software Engineering. 10th ed. Pearson, 2015.

10 PostgreSQL Global Development Group. PostgreSQL 16 Documentation: Row Security Policies [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/current/ddl-rowsecurity.html.