ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЇ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифрова система корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту на базі Python та JavaScript»

на здобуття освітнього ступеня бакалавра

зі спеціальності 121 Інженерія програмного забезпечення

освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ Дмитро ПОДОЛЬСЬКИЙ

Виконав: здобувач вищої освіти групи ТЦР-42
Дмитро ПОДОЛЬСЬКИЙ

Керівник: доктор техн. наук, професор Ігор СІНІЦИН

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технології цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Подольському Дмитру Олеговичу

1. Тема кваліфікаційної роботи: «Розробка веб-додатку для корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту на базі Python та JavaScript»
керівник кваліфікаційної роботи д.т.н., професор Сініцин Ігор Петрович,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р. № 51.

2. Строк подання кваліфікаційної роботи 18 травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація фреймворків Flask, React, SQLAlchemy та PostgreSQL, наукові джерела з проєктного менеджменту та корпоративних інформаційних систем, матеріали з побудови REST API та клієнт-серверних архітектур, приклади реалізації систем управління бізнес-процесами.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд предметної галузі та аналіз існуючих корпоративних платформ управління проєктами та ресурсами.

2. Аналіз інструментів реалізації та архітектурного проєктування web-застосунку.

3. Програмна реалізація web-застосунку для корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту.

4. Реалізація інтерфейсу та тестування функціональності web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання (гість та користувач).
5. Діаграма варіантів використання (адміністратор).
6. Діаграма класів підбору товарів за типом фігури та кольоротипом.
7. Діаграма структури бази даних.
8. Екранні форми.

6. Дата видачі завдання«20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі та існуючих аналогів (Asana, Monday.com, Wrike)	21.02.2026	Виконано
2	Визначення вимог до системи та постановка завдання на розробку	28.02.2026	Виконано
3	Вибір технологічного стеку та проєктування архітектури (Flask, React, PostgreSQL)	12.03.2026	Виконано
4	Програмна реалізація серверної частини (REST API, автентифікація, модулі)	20.03.2026	Виконано
5	Програмна реалізація клієнтської частини (React, Redux, маршрутизація, UI)	30.03.2026	Виконано
6	Тестування системи (Postman, unit-тести, кросбраузерне тестування)	03.04.2026	Виконано
7	Оформлення пояснювальної записки та підготовка до захисту	08.04.2026	Виконано
8	Передзахист кваліфікаційної роботи	11.05.2026	Виконано

Здобувач вищої освіти _____ Подольський Дмитро

Керівник

кваліфікаційної роботи _____ Ігор Сініцин

“__”_____2026 року

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _50_ стор., __1_ табл., __37_ рис., __26_ джерел.

Мета роботи – покращення процесу корпоративного управління проєктами та ресурсами за допомогою розробленого веб-додатку з інтеграцією функцій фінансового менеджменту.

Об'єкт дослідження – процес організації та контролю корпоративних проєктів, ресурсів і фінансових операцій у малих та середніх компаніях.

Предмет дослідження – програмне забезпечення для автоматизації управління проєктами, складськими запасами, продажами, закупівлями та фінансовим обліком на базі Python та JavaScript.

Короткий зміст роботи: У роботі досліджено проблематику корпоративного управління проєктами та ресурсами, проаналізовано існуючі платформи-аналоги (Asana, Monday.com, Wrike). Спроєктовано та реалізовано веб-додаток Ri Dren, серверна частина якого побудована на Flask (Python) із використанням SQLAlchemy ORM, Flask-JWT-Extended для автентифікації та Flask-RESTful для побудови REST API. Клієнтську частину реалізовано на React із TypeScript, Redux Toolkit для управління станом та React Router для маршрутизації. Система охоплює модулі управління проєктами (пайплайни, задачі, підзадачі), фінансами (рахунки, транзакції, бюджети, рахунки-фактури), складом (склади, інвентар, переміщення, відвантаження), продажами (клієнти, пропозиції, замовлення) та закупівлями (постачальники, замовлення). Реалізовано рольову модель доступу, систему сповіщень, чат, ієрархію співробітників та статистичні дашборди. Проведено функціональне тестування через Postman та модульне тестування серверної логіки.

Сферою використання застосунку є автоматизація бізнес-процесів у компаніях, що потребують єдиної платформи для ведення проєктів, обліку фінансів та управління ресурсами.

КЛЮЧОВІ СЛОВА: WEB-ДОДАТОК, УПРАВЛІННЯ ПРОЄКТАМИ, ФІНАНСОВИЙ МЕНЕДЖМЕНТ, FLASK, REACT, REST API, КОРПОРАТИВНА СИСТЕМА.

ЗМІСТ

ВСТУП 10
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 12
1.1 Характеристика предметної галузі 12
1.2 Аналіз існуючих аналогів корпоративних систем управління 13
1.2.1 Аналіз платформи Asana 13
1.2.2 Аналіз платформи Monday.com 15
1.2.3 Аналіз платформи Wrike 17
1.3 Визначення проблем предметної галузі 21
1.4 Постановка завдання на розробку системи 22
1.5 Визначення вимог до застосунку 24
2 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ 26
2.1 Загальна архітектура програмного забезпечення 26
2.2 Сценарії взаємодії користувача із системою 27
2.3 Сценарії взаємодії адміністратора із системою 29
2.4 Структурна діаграма класів системи персоналізованого підбору 30
2.5 Алгоритм класифікації типу фігури на основі нечітких правил 33
2.6 Модуль фінансового менеджменту 35
2.7 Структура бази даних 36
3 РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ТА ТЕСТУВАННЯ СИСТЕМИ 39
3.1 Структура навігації web-застосунку 39
3.2 Реалізація інтерфейсу системи 41
3.3 Тестування функціональної логіки за допомогою Postman та unit-тестів 51
3.3.1 Тестування модуля реєстрації 51
3.3.2 Тестування API управління проєктами 52
3.3.3 Тестування фінансових операцій 53
ВИСНОВКИ 1
ПЕРЕЛІК ПОСИЛАНЬ 3

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 6
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ 17

ВСТУП

Актуальність: Сьогодні малий і середній бізнес в Україні стикається з тим, що для ведення справ доводиться одночасно користуватися кількома різними програмами: одна – для проєктів, інша – для фінансів, третя – для складу. Дані розкидані по різних сервісах, і зібрати повну картину по компанії буває складно. Особливо це відчувається, коли команда росте і процесів стає більше – ручне перенесення інформації між системами забирає час і породжує помилки.

Ринок корпоративних платформ пропонує чимало рішень. Є Asana і Monday.com, які добре справляються з управлінням задачами, але не мають вбудованого фінансового обліку. Є Odoo – потужна ERP-система, але вона надмірно складна для невеликих команд і потребує значних зусиль на впровадження. Жодне з цих рішень не дає простого поєднання проєктного менеджменту, фінансів, складу, продажів і закупівель в одному інтерфейсі без зайвої складності.

Саме ця прогалина і стала відправною точкою для розробки Ri Dren – веб-додатку, який об'єднує в собі управління проєктами (пайплайни, задачі з підзадачами), фінансовий менеджмент (рахунки, транзакції, бюджети, рахунки-фактури), складський облік (склади, інвентар, переміщення, відвантаження), модулі продажів (клієнти, пропозиції, замовлення) та закупівель (постачальники, замовлення на закупку). Окрім цього, в системі реалізовано рольову модель доступу, внутрішній чат, ієрархію співробітників та статистичні дашборди.

Об’єкт дослідження – процес організації та контролю корпоративних проєктів, ресурсів і фінансових операцій у малих та середніх компаніях.

Предмет дослідження – програмне забезпечення для автоматизації управління проєктами, складськими запасами, продажами, закупівлями та фінансовим обліком на базі Python та JavaScript.

Мета роботи – розробка веб-додатку для корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту.

Для досягнення поставленої мети в бакалаврській роботі виконано наступні задачі:

1. Проаналізувати існуючі корпоративні платформи (Asana, Monday.com, Odoo), виявити їхні сильні та слабкі сторони стосовно комплексного управління бізнес-процесами.

2. Сформулювати вимоги до системи, яка поєднує проєктний менеджмент, фінансовий облік, складське управління, продажі та закупівлі.

3. Обрати технологічний стек і спроєктувати архітектуру додатку – серверну частину на Flask (Python) із SQLAlchemy та Flask-JWT-Extended, клієнтську – на React із TypeScript та Redux Toolkit.

4. Реалізувати серверне API з модулями управління проєктами, фінансами, складом, продажами, закупівлями, а також систему автентифікації, сповіщень і чату.

5. Розробити клієнтський інтерфейс із маршрутизацією, рольовим доступом, статистичними дашбордами та підтримкою локалізації.

6. Провести тестування системи – функціональне через Postman та модульне (unit-тести) для серверної логіки.

Методи дослідження: аналіз наукових та технічних джерел у галузі корпоративних інформаційних систем, порівняльний аналіз існуючих платформ, моделювання архітектури програмного забезпечення, експериментальна розробка та тестування.

Наукова новизна роботи полягає у створенні готового до використання веб-додатку, який дозволяє невеликим компаніям вести проєкти, фінанси та ресурси в єдиному середовищі замість кількох розрізнених інструментів. Система вже пройшла функціональне тестування і може бути розгорнута для реального використання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Характеристика предметної галузі

Корпоративне управління проєктами та ресурсами – це сфера, де компанії намагаються тримати під контролем одразу кілька напрямків: хто над чим працює, скільки грошей витрачається, що лежить на складі, які замовлення в роботі. Для великих корпорацій існують SAP, Oracle, Microsoft Dynamics - але ці системи коштують сотні тисяч доларів і потребують місяців впровадження. Малий і середній бізнес зазвичай не може собі цього дозволити.

На практиці невеликі компанії вирішують проблему так: беруть Trello або Asana для задач, Google Sheets для фінансів, окремий месенджер для комунікації, а складський облік ведуть хто в Excel, хто взагалі на папері. Проблема в тому, що ці інструменти не зв'язані між собою. Менеджер не бачить, як витрати на проєкт співвідносяться з бюджетом. Складський працівник не знає, що під конкретне замовлення потрібно зарезервувати товар. Фінансист не має доступу до статусу проєктів, щоб зрозуміти, коли очікувати оплату.

Предметна галузь, яку охоплює дана робота, знаходиться на перетині кількох дисциплін: проєктний менеджмент, фінансовий облік, управління ланцюгами постачання та CRM-системи. Кожна з цих областей має свої усталені підходи і стандарти, але саме їхнє поєднання в єдиному програмному продукті залишається актуальною задачею, особливо для українського ринку, де багато компаній тільки переходять від ручного управління до цифрових інструментів [1].

Окремо варто згадати про роль комунікації всередині команди. Коли проєктні дані, фінанси і склад зібрані в одному місці, але обговорення відбувається в Telegram або Slack – контекст губиться. Тому сучасні корпоративні платформи все частіше інтегрують чати та систему сповіщень безпосередньо в робочий простір [2].

1.2 Аналіз існуючих аналогів корпоративних систем управління

Перед початком розробки було проаналізовано три платформи, які найчастіше згадуються у контексті управління проєктами та бізнес-процесами: Asana, Monday.com та Odoo. Кожна з них має свою нішу і свої обмеження, які важливо розуміти для формулювання вимог до власного рішення.

1.2.1 Аналіз платформи Asana

Asana – один із найвідоміших інструментів для управління проєктами, запущений у 2008 році колишніми співробітниками Facebook. Платформа орієнтована насамперед на командну роботу над задачами: можна створювати проєкти, розбивати їх на задачі та підзадачі, призначати виконавців, ставити дедлайни і відстежувати прогрес через різні подання – список, дошку (Kanban), таймлайн (діаграму Ганта) та календар [3].

Інтерфейс Asana побудований досить інтуїтивно – новий користувач може розібратися за кілька хвилин без інструкцій. Є вбудована автоматизація: наприклад, коли задача переходить у статус "Завершено", система може автоматично створити наступну задачу або надіслати повідомлення відповідальному. Платформа підтримує інтеграції з понад 200 зовнішніми сервісами – Slack, Google Drive, Zoom, GitHub тощо.

Безкоштовний план обмежений 10 користувачами і базовим функціоналом. Для команд від 15 осіб вартість починається від $10.99 на користувача на місяць (план Premium), а розширені функції – таймлайн, портфелі проєктів, розширена звітність – доступні від $24.99 (план Business). Для невеликої компанії з 20 співробітниками це вже $200-500 на місяць [3].

У платформі немає модуля фінансового обліку – не можна вести рахунки, бюджети, транзакції. Складський облік теж відсутній. Модулі продажів і

закупівель не передбачені. Коментарі до задач є, але повноцінного чату між співробітниками немає - для обговорень потрібен зовнішній месенджер.

Переваги платформи Asana:

– інтуїтивний інтерфейс, який не потребує тривалого навчання;

– кілька подань проєкту – список, Kanban, таймлайн, календар;

– вбудована автоматизація задач (правила, тригери);

– понад 200 інтеграцій із зовнішніми сервісами;

– зручна система підзадач і залежностей між задачами.

Недоліки:

– відсутність модуля фінансового обліку (рахунки, бюджети, транзакції);

– немає складського обліку та управління інвентарем;

– немає модулів продажів і закупівель;

– відсутній вбудований чат між співробітниками;

– рольова модель обмежена базовими ролями (адмін, учасник, гість);

– висока вартість для команд від 15 осіб ($200-500/міс).

Інтерфейс головної сторінки Asana та приклад Kanban-дошки проєкту зображено на рисунках 1.1 та 1.2 відповідно.

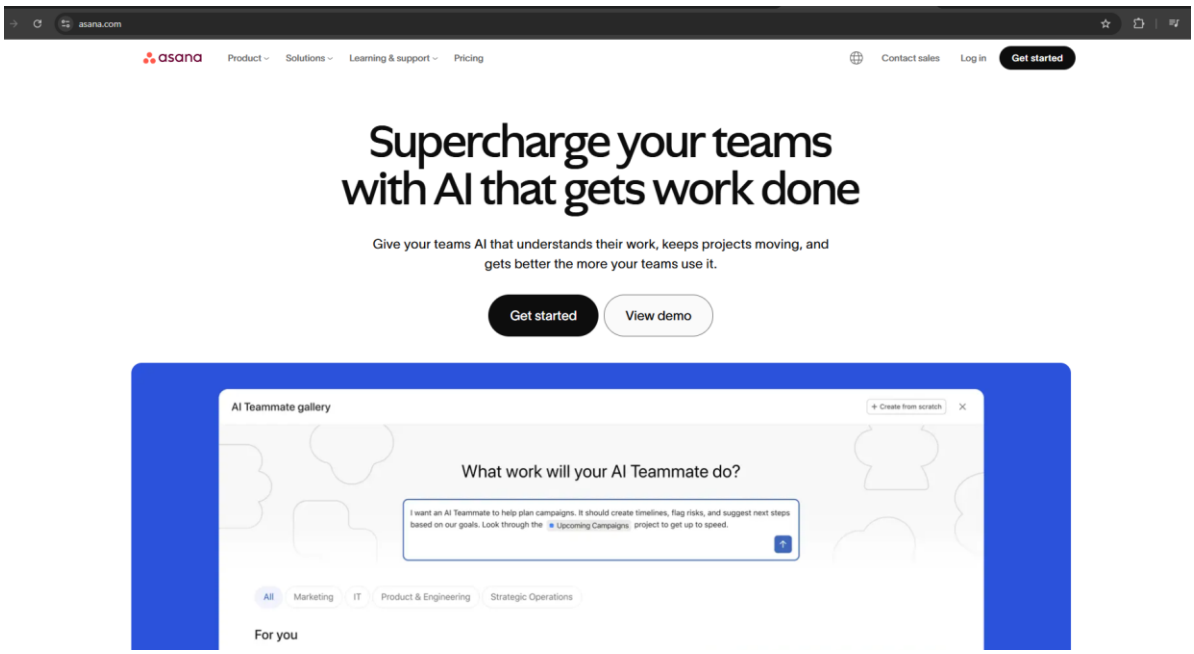


Рис. 1.1. Головна сторінка Asana

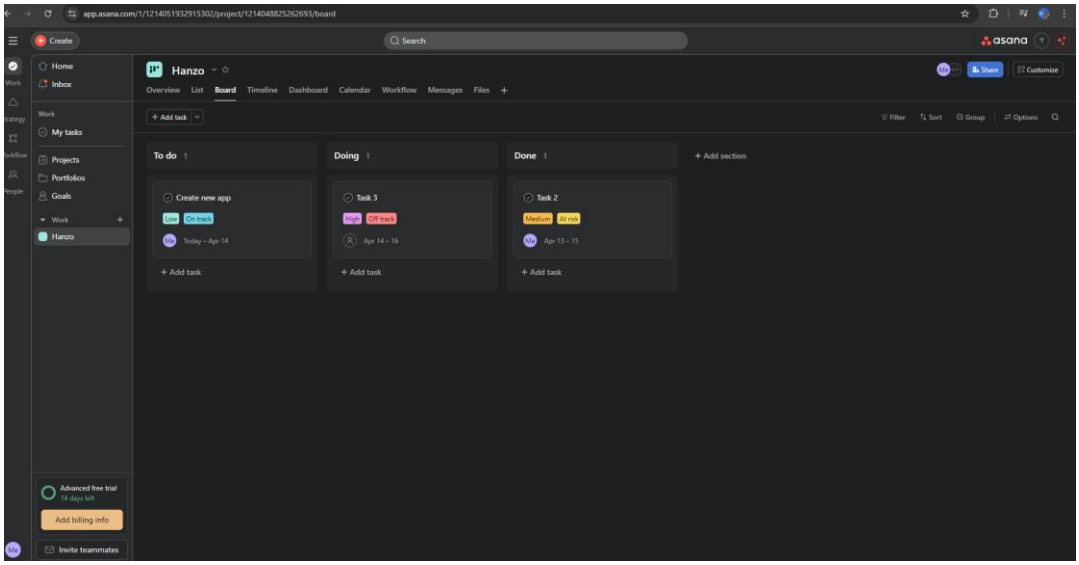


Рис. 1.2. Kanban-дошка проєкту в Asana

1.2.2 Аналіз платформи Monday.com

Monday.com позиціонує себе як Work OS – операційна система для роботи. Платформа намагається бути універсальним інструментом: тут можна вести проєкти, створювати CRM-дошки для відстеження клієнтів, будувати воронки продажів, планувати маркетингові кампанії [4].

Головна особливість Monday.com - гнучкість налаштування. Кожна дошка – це по суті таблиця, де можна додавати стовпці різних типів: текст, число, дата, статус, формула, залежність, файл тощо. Це дає змогу адаптувати платформу під різні процеси без написання коду. Є також автоматизації та інтеграції, хоча їхня кількість залежить від тарифного плану.

Monday.com частково закриває потребу в CRM – можна створити дошку з клієнтами, відстежувати угоди, рахувати суми. Але це не повноцінний фінансовий модуль: немає рахунків, транзакцій, бюджетів з автоматичним перерахунком залишків. Складський облік теж не передбачений – можна зробити дошку з товарами, але без логіки переміщень, відвантажень, прив'язки до замовлень.

Мінімальний платний план (Basic) коштує $9 на користувача на місяць, але він дуже обмежений. Реально працювати можна починаючи з плану Standard ($12) або Pro ($19). При цьому мінімальна кількість користувачів - 3 особи. Для команди з 20 осіб це $240-380 на місяць [4].

При великій кількості дошок і рядків (більше 10 000) інтерфейс починає помітно гальмувати. Це відзначають користувачі у відгуках на G2 та Capterra [5].

Переваги платформи Monday.com:

– висока гнучкість - дошки можна налаштувати під будь-який процес;

– різноманітні типи стовпців (формули, залежності, файли, статуси);

– часткова CRM-функціональність (дошки клієнтів, воронки);

– візуально привабливий інтерфейс з кольоровою індикацією статусів;

– автоматизації та інтеграції з популярними сервісами.

Недоліки:

– немає повноцінного фінансового модуля (рахунки, транзакції, бюджети);

– відсутній складський облік з логікою переміщень та відвантажень;

– продуктивність падає при великій кількості даних (10 000+ рядків);

– немає вбудованого чату – тільки коментарі до елементів дошки;

– права доступу налаштовуються лише на рівні дошок, а не окремих функцій;

– висока вартість для команд ($240-380/міс за 20 осіб).

Приклади інтерфейсу Monday.com – робочий простір та налаштування дошки – зображено на рисунках 1.3 та 1.4.

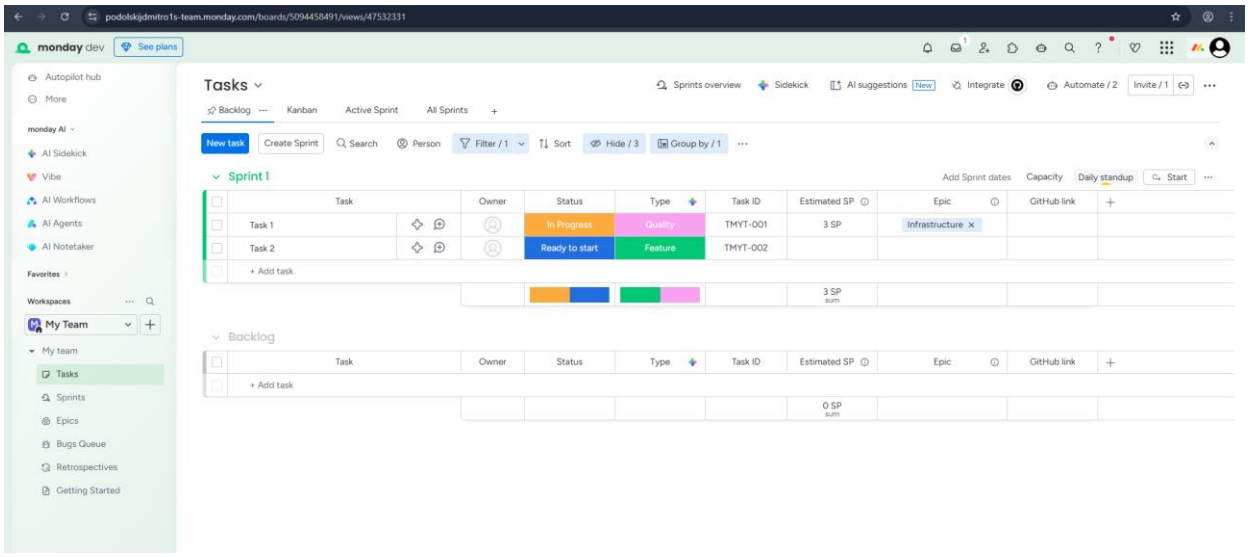


Рис. 1.3. Робочий простір Monday.com

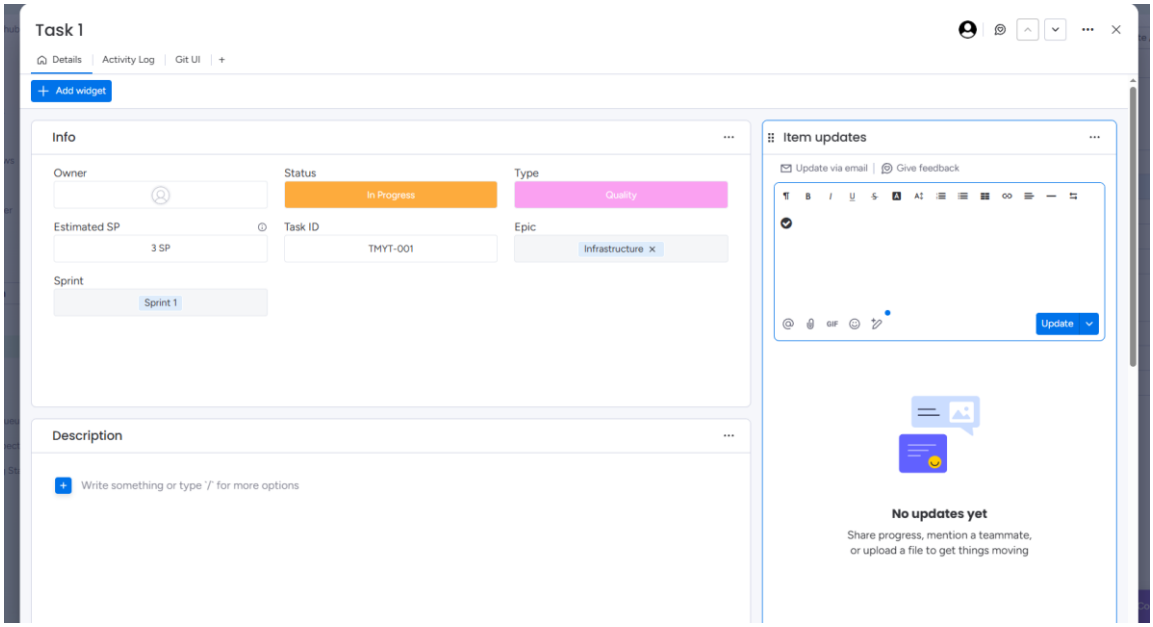


Рис. 1.4. Налаштування дошки Monday.com

1.2.3 Аналіз платформи Wrike

Wrike – платформа для управління проєктами та спільної роботи, яка існує з 2006 року і має понад 20 000 корпоративних клієнтів по всьому світу. Wrike орієнтований на середні та великі команди і пропонує розширений набір інструментів для планування, відстеження та звітності по проєктах [6].

Серед ключових можливостей Wrike – діаграми Ганта з залежностями між задачами, Kanban-дошки, робочі навантаження команди (workload view), крос-тегування задач між проєктами та налаштовувані дашборди. Платформа підтримує вбудоване рецензування файлів (proofing) - можна залишати коментарі прямо на зображеннях або PDF-документах, що корисно для дизайнерських та маркетингових команд.

Wrike має більш розвинену систему звітності порівняно з іншими трекерами задач. Можна будувати кастомні звіти з фільтрацією по проєктах, виконавцях, статусах, датах. Є також модуль обліку часу (time tracking) – співробітники можуть логувати витрачений час на задачі, і на основі цих даних формуються звіти по продуктивності.

Безкоштовний план обмежений базовим функціоналом для невеликих команд. Платні плани починаються від $9.80 на користувача на місяць (Team), а

розширені функції – автоматизації, кастомні поля, звіти - доступні від $24.80 (Business). Для 20 користувачів це $196-496 на місяць [6].

Однак Wrike, як і інші трекери задач, не має вбудованого фінансового обліку. Рахунки, транзакції, бюджети – все це потрібно вести в окремій системі. Складський модуль відсутній повністю. Модулі продажів і закупівель не передбачені. Чат є тільки у вигляді коментарів до задач - окремого месенджера немає.

Переваги платформи Wrike:

− розвинена система звітності з кастомними фільтрами;

− діаграми Ганта з залежностями та критичним шляхом;

− вбудований облік часу (time tracking) на задачах;

− рецензування файлів (proofing) прямо в інтерфейсі;

− workload view для контролю навантаження команди.

Недоліки:

− відсутність фінансового модуля (рахунки, бюджети, транзакції);

− немає складського обліку та управління інвентарем;

− немає модулів продажів і закупівель;

− відсутній повноцінний чат між співробітниками;

− складний інтерфейс – потребує часу на освоєння для нових користувачів;

− висока вартість розширених планів ($496/міс за 20 осіб на плані Business).

Приклади інтерфейсу Wrike – діаграма Ганта та дашборд проєкту – зображено на рисунках 1.5 та 1.6.

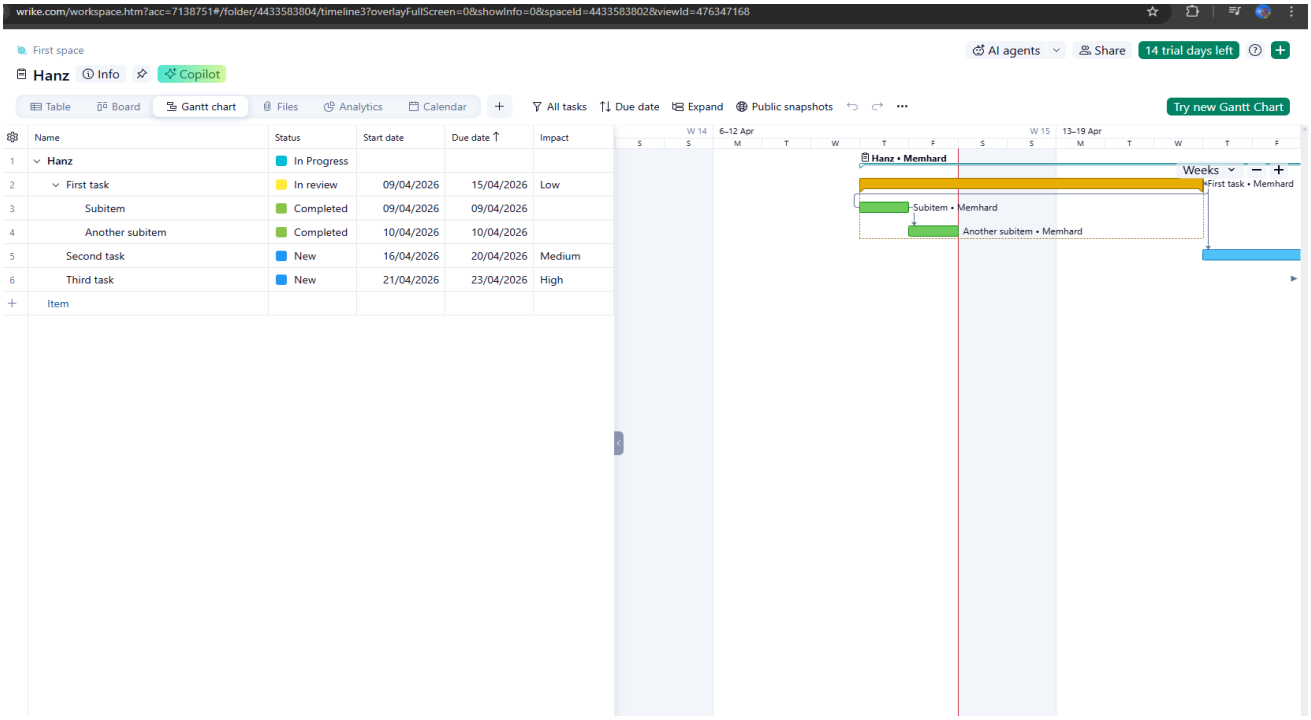


Рис. 1.5. Діаграма Ганта в Wrike

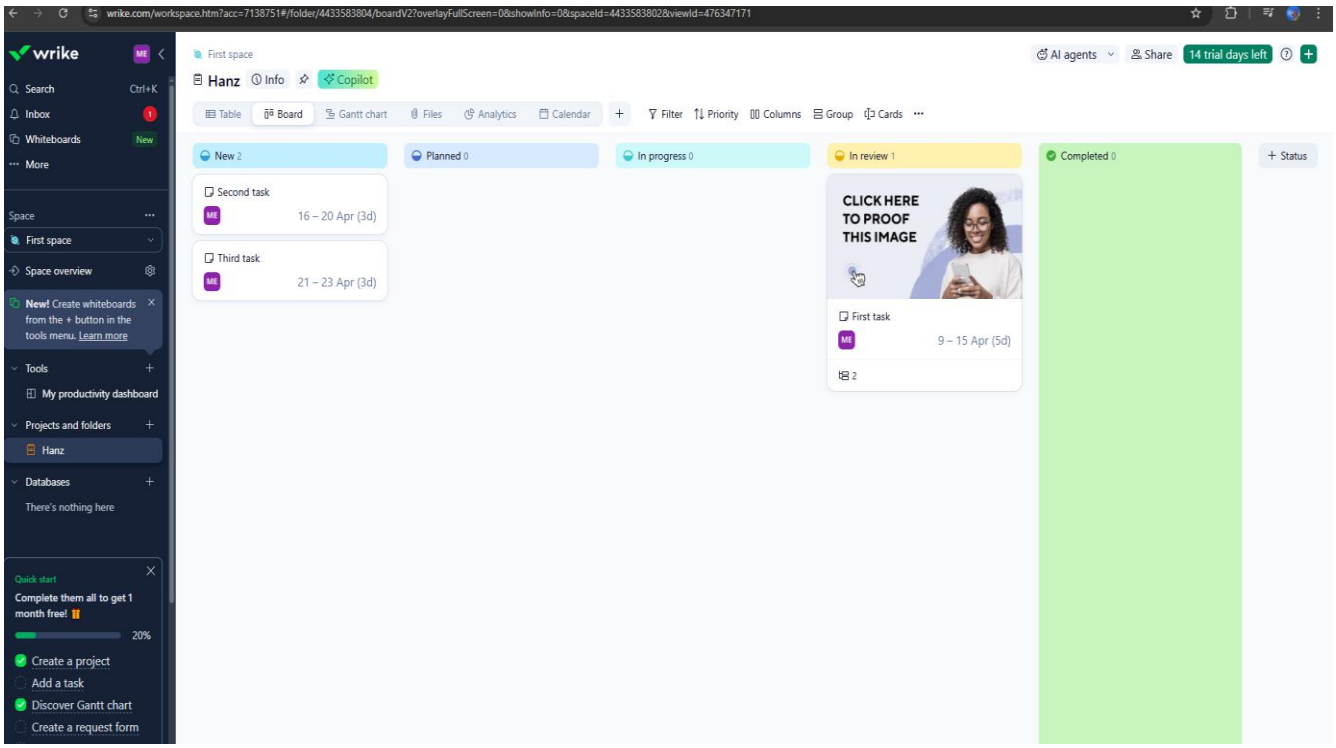


Рис. 1.6. Дашборд проєкту в Wrike

Зведена порівняльна таблиця функціональних можливостей аналізованих платформ наведена у таблиці 1.1.

Таблиця 1.1

Порівняння функціональних можливостей web-додатків

Функціональність	Asana	Monday.com	Wrike
Управління проєктами та задачами	+	+	+
Підзадачі та залежності	+	±	+
Kanban-дошки	+	+	+
Діаграма Ганта / таймлайн	+	+	+
Фінансовий облік (рахунки, транзакції)	–	–	–
Бюджети з контролем витрат	–	–	–
Рахунки-фактури (bills)	–	–	–
Складський облік	–	–	–
Переміщення та відвантаження	–	–	–
CRM / база клієнтів	–	±	–
Модуль закупівель	–	–	–
Вбудований чат	–	–	–
Система сповіщень (in-app + email)	+	+	+
Гнучка рольова модель (на рівні функцій)	–	–	±
Ієрархія співробітників	–	–	–
Статистичні дашборди	±	±	+

1.3 Визначення проблем предметної галузі

У процесі аналізу предметної галузі та вивчення особливостей існуючих платформ для корпоративного управління було виявлено низку типових проблем, які стримують ефективне ведення бізнес-процесів у малих та середніх компаніях і знижують якість організації робочих процесів.

Насамперед, однією з головних проблем є відсутність єдиної платформи, яка б поєднувала управління проєктами з фінансовим обліком та складським управлінням. Переважна більшість існуючих рішень пропонує лише трекінг задач і базову звітність, без можливості вести рахунки, бюджети чи контролювати складські залишки. Компанії змушені використовувати 3-5 різних сервісів одночасно, що призводить до дублювання даних, втрати часу на ручну синхронізацію та підвищеного ризику помилок [1].

Іншою проблемою є відсутність вбудованих засобів комунікації. Жодна з проаналізованих платформ не має повноцінного внутрішнього чату – лише коментарі до задач або елементів дошки. На практиці це означає, що обговорення проєктів відбувається в Telegram, Slack або email, і зв'язок між рішенням та контекстом обговорення втрачається. Співробітник, який підключається до проєкту пізніше, не може відновити історію прийняття рішень без перегляду десятків повідомлень у месенджері.

Також спостерігається обмеженість рольових моделей доступу. Існуючі платформи пропонують базові ролі – адміністратор, учасник, гість – або дозволяють налаштовувати права лише на рівні дошок чи проєктів. Для корпоративної системи, де різні відділи мають бачити різні модулі, цього недостатньо. Потрібна можливість визначати доступ на рівні окремих функцій: хто може додавати рахунки, хто – видаляти товари зі складу, хто бачить фінансову статистику.

Крім цього, до ключових проблем галузі належать:

− висока вартість існуючих рішень: для команди з 20 осіб щомісячна оплата складає $200-500, і це без фінансового та складського модулів;

− відсутність ієрархії співробітників: жодна з платформ не дозволяє побудувати організаційну структуру компанії з підпорядкуванням;

− обмежена локалізація: українська мова інтерфейсу не підтримується в жодному з аналізованих рішень;

− залежність від хмарної інфраструктури провайдера: дані компанії зберігаються на серверах третіх сторін без можливості розгортання на власному сервері.

Отже, можна виділити основні напрями, які потребують вирішення в межах розробки нового web-застосунку:

− створення єдиної платформи, що об'єднує управління проєктами, фінансами, складом, продажами та закупівлями;

− реалізація вбудованого чату та системи сповіщень для збереження контексту обговорень;

− побудова гнучкої рольової моделі з налаштуванням прав на рівні окремих функцій;

− забезпечення можливості розгортання на власному сервері без щомісячної абонплати за користувачів.

Ці проблеми і визначають обґрунтовану потребу в розробці власної корпоративної системи, яка забезпечуватиме якісно новий рівень управління бізнес-процесами для малих та середніх компаній.

1.4 Постановка завдання на розробку системи

На основі проведеного аналізу предметної галузі, вивчення технічних джерел [1–2], а також результатів порівняльного аналізу існуючих аналогів (платформ Asana, Monday.com, Wrike), було виявлено низку проблем, які на сьогодні не вирішуються в межах більшості доступних web-рішень. Ці проблеми стосуються насамперед фрагментації інструментів, відсутності вбудованої комунікації, обмеженої рольової моделі та високої вартості існуючих платформ.

У межах цієї кваліфікаційної роботи ставиться завдання створити web-застосунок Ri Dren для корпоративного управління проєктами та ресурсами, який би забезпечував комплексне ведення бізнес-процесів – від управління задачами до фінансового обліку та складського управління – в єдиному інтерфейсі.

Для реалізації поставленої задачі передбачається:

− реалізація модуля управління проєктами з підтримкою пайплайнів, стадій, проєктів, задач із підзадачами, призначенням виконавців та відстеженням статусів;

− розробка модуля фінансового менеджменту - банківські рахунки з балансами, транзакції (дохід/витрата), бюджети з контролем витрат, рахунки-фактури;

− створення модуля складського обліку – склади, інвентарні позиції з кількістю та ціною, переміщення між складами, відвантаження;

− інтеграція модулів продажів (база клієнтів, комерційні пропозиції, замовлення) та закупівель (база постачальників, замовлення на закупку);

− реалізація системи управління персоналом – ієрархія співробітників, рольова модель доступу з гнучким налаштуванням прав на рівні окремих функцій;

− створення вбудованого чату між співробітниками та системи сповіщень (in-app та email-нотифікації);

− використання сучасного технологічного стеку, зокрема Flask (Python) із SQLAlchemy та Flask-JWT-Extended для серверної частини, React із TypeScript та Redux Toolkit для клієнтської частини;

− забезпечення захисту персональних даних користувачів через хешування паролів та JWT-автентифікацію.

Поставлене завдання передбачає створення системи, яка поєднує інструменти проєктного менеджменту, фінансового обліку та управління ресурсами. Особлива увага буде приділена зручності інтерфейсу, швидкості відповіді системи та можливості розгортання на власному сервері без залежності від хмарних провайдерів.

У результаті реалізації проєкту очікується створення повнофункціонального web-застосунку, здатного якісно підвищити рівень організації бізнес-процесів у малих та середніх компаніях.

1.5 Визначення вимог до застосунку

На основі попереднього аналізу предметної галузі, виявлених проблем та сформульованої мети проєкту, сформульовано вимоги до функціональності та якості розроблюваного web-застосунку для корпоративного управління проєктами та ресурсами. Усі вимоги поділено на функціональні та нефункціональні.

Функціональні вимоги:

1. Реєстрація та авторизація користувачів із підтвердженням email через OTP-код, отриманням JWT-токена та можливістю скидання пароля.

2. Створення компанії після реєстрації із можливістю запрошення співробітників за посиланням.

3. Гнучке налаштування прав доступу для кожного співробітника на рівні окремих функцій (наприклад, "додавати рахунки", "видаляти товари зі складу", "переглядати фінансову статистику").

4. Модуль управління проєктами: пайплайни зі стадіями, проєкти, задачі з підзадачами, призначення виконавців, відстеження статусів.

5. Модуль фінансового менеджменту: банківські рахунки, транзакції (дохід/витрата), бюджети з контролем витрат, рахунки-фактури.

6. Модуль складського обліку: склади, інвентарні позиції, переміщення між складами, відвантаження.

7. Модулі продажів (клієнти, пропозиції, замовлення) та закупівель (постачальники, замовлення на закупку).

8. Внутрішній чат з підтримкою групових бесід та inbox-системою.

9. Система сповіщень – in-app та email-нотифікації при зміні статусу задачі, наближенні дедлайну, створенні замовлення тощо.

10. Статистичні дашборди для кожного модуля з графіками.

11. Підтримка локалізації – українська та англійська мови інтерфейсу.

Нефункціональні вимоги:

1. Контроль доступу до функцій системи, обмежений відповідно до налаштованих прав кожного співробітника.

2. Забезпечення безпеки облікових записів шляхом хешування паролів за допомогою sha256-алгоритму.

3. JWT-автентифікація з access-токеном (термін дії 120 хвилин) та refresh-токеном (від 1 до 30 днів).

4. Оптимізація швидкодії: час відповіді API не повинен перевищувати 500 мс для типових запитів.

5. Кросбраузерна сумісність – коректне відображення та робота інтерфейсу у сучасних версіях браузерів (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge).

Визначені вимоги ляжуть в основу подальшого етапу проєктування системи, побудови її архітектури та реалізації функціональних модулів. Вони спрямовані на забезпечення зручності використання, безпеки та стабільності web-застосунку.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ

2.1 Загальна архітектура програмного забезпечення

Розроблений web-застосунок Ri Dren реалізовано на основі клієнт-серверної архітектури, де клієнтська та серверна частини працюють як окремі додатки і взаємодіють через REST API по протоколу HTTP. Такий підхід дозволяє розробляти, тестувати та масштабувати кожну частину незалежно.

Система складається з трьох основних рівнів:

− Клієнтський рівень (frontend) реалізований як Single Page Application на React із TypeScript. Додаток запускається на порті 3000 і відповідає за відображення інтерфейсу, маршрутизацію між сторінками, управління станом через Redux Toolkit та відправку HTTP-запитів до серверного API. Вся логіка рендерингу відбувається на стороні клієнта – сервер віддає лише дані у форматі JSON.

− Серверний рівень (backend) побудований на Flask (Python) і працює на порті 5000. Він обробляє запити від клієнта, виконує бізнес-логіку, взаємодіє з базою даних через SQLAlchemy ORM та повертає відповіді у форматі JSON. Серверна частина організована модульно – кожен бізнес-домен (проєкти, фінанси, склад, продажі, закупівлі, співробітники) має свою папку з API-ресурсами.

− Рівень даних – PostgreSQL база даних risoftware, до якої сервер підключається через драйвер psycopg2. Усі операції з даними проходять через SQLAlchemy, що забезпечує абстракцію від конкретної СУБД.

Комунікація між клієнтом і сервером відбувається через HTTP-запити. Клієнт надсилає запити на ендпоінти виду /api/company/{id}/pipelines, /api/accounts/{id}, /api/task/{id} тощо. Кожен запит містить JWT-токен у заголовку Authorization. Сервер перевіряє токен, визначає поточного користувача та його права доступу, виконує операцію і повертає результат.

Для захисту системи реалізовано Flask-JWT-Extended для автентифікації та власну систему перевірки прав (check_feature) для авторизації. CORS налаштовано через Flask-Cors для дозволу запитів з порту 3000.

Фонові процеси реалізовано через APScheduler - планувальник, який запускає періодичні задачі: відправку email-сповіщень з черги (кожні 5 секунд) та перевірку дедлайнів задач з надсиланням нагадувань (кожні 24 години).

Архітектура клієнт-серверної взаємодії web-застосунку зображена на рисунку 2.1.

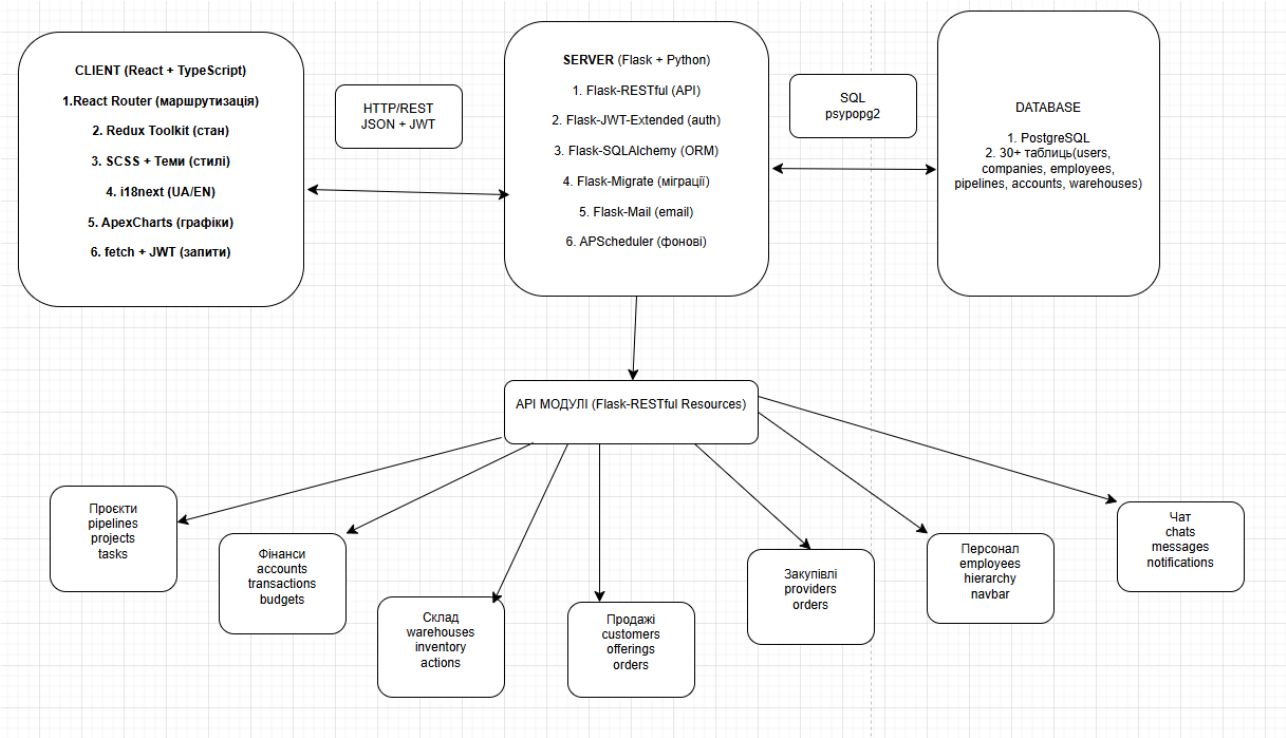


Рис. 2.1. Діаграма архітектури клієнт-серверної взаємодії web-застосунку

2.2 Сценарії взаємодії користувача із системою

У межах розробки web-застосунку Ri Dren була створена діаграма варіантів використання, яка ілюструє основні сценарії взаємодії звичайного користувача (співробітника компанії) з системою.

Після реєстрації та підтвердження email користувач створює компанію або приєднується до існуючої за запрошенням. Далі, залежно від наданих прав доступу, він може працювати з різними модулями системи.

У модулі проєктів користувач переглядає список пайплайнів, відкриває конкретний пайплайн зі стадіями, створює проєкти та задачі, призначає виконавців,

змінює статуси (Open, In Progress, Completed, Accepted, Rejected), додає підзадачі та відстежує час через вбудований time tracker. Також доступний календар задач із фільтрацією по дедлайнах.

У фінансовому модулі користувач створює банківські рахунки, додає транзакції (дохід або витрата), переглядає баланси. Окремо реалізовано бюджети – можна задати ліміт витрат і прив'язати до бюджету рахунки-фактури (bills). Система автоматично відстежує перевищення бюджету.

У складському модулі доступні операції зі складами та інвентарними позиціями: створення складів, додавання товарів із зазначенням кількості та ціни, переміщення між складами, відвантаження. Є також прогнозування потреб на основі поточних залишків.

Модулі продажів і закупівель дозволяють вести базу клієнтів та постачальників, створювати комерційні пропозиції (offerings), оформлювати замовлення на продаж та закупку.

Окрім бізнес-модулів, користувач має доступ до профілю (зміна даних, аватар, пароль), внутрішнього чату (особисті та групові бесіди), системи сповіщень та налаштувань компанії.

Діаграма варіантів використання користувачем наведена на рисунку 2.2.

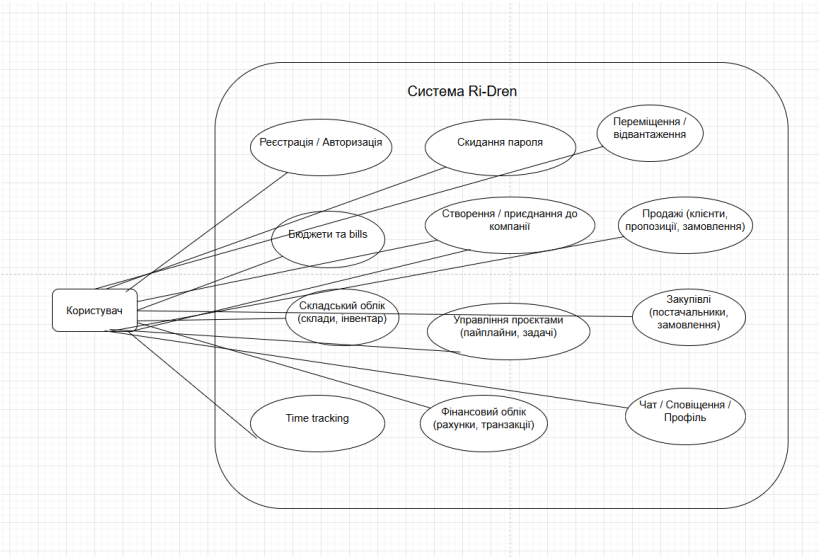


Рис. 2.2. Діаграма варіантів використання актора «Користувач»

2.3 Сценарії взаємодії адміністратора із системою

Адміністратор системи – це користувач, який створив компанію (owner). Він має повний доступ до всіх функцій і додатково може керувати співробітниками та їхніми правами.

Основні сценарії адміністратора включають:

− управління співробітниками – запрошення нових працівників за посиланням, перегляд списку співробітників, переміщення в ієрархії, звільнення;

− налаштування прав доступу – для кожного співробітника адміністратор може увімкнути або вимкнути конкретні функції (наприклад, add_accounts, delete_warehouses, sales_statistics). Це реалізовано через таблицю Navbar, де кожен запис - це пара (employee_id, feature);

− управління позиціями (посадами) та мітками (labels) - створення, редагування та видалення;

− перегляд логів системи – журнал помилок та подій;

− налаштування компанії – зміна назви, контактних даних, аватару.

Адміністратор також має доступ до всіх статистичних дашбордів: по проєктах (кількість задач за статусами, завантаженість команди), по фінансах (доходи/витрати за період, баланси рахунків), по складу (залишки, рух товарів), по продажах та закупівлях.

Діаграма варіантів використання адміністратором наведена на рисунку 2.3.

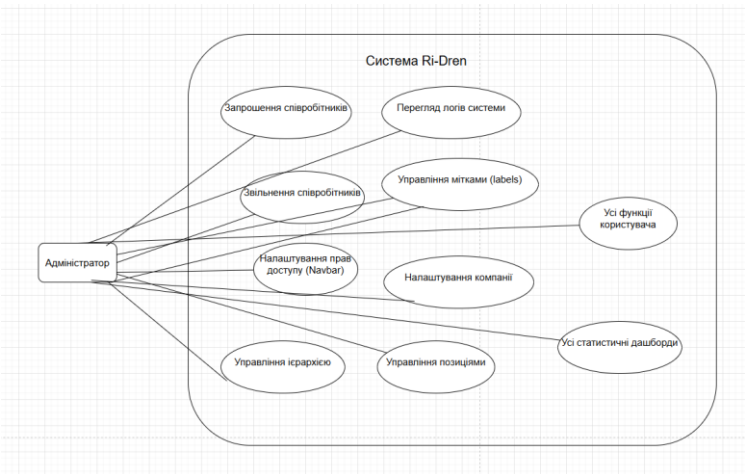


Рис. 2.3. Діаграма варіантів використання актора «Адміністратор»

2.4 Структурна діаграма класів системи персоналізованого підбору

Серверна частина системи побудована на основі моделей SQLAlchemy, кожна з яких відповідає таблиці у базі даних PostgreSQL. Центральними класами є User, Company та Employee.

Клас User зберігає облікові дані: email, хешований пароль (_password), ім'я, прізвище, телефон, аватар (як LargeBinary), стать, освіту, мову інтерфейсу (ua/en), дату реєстрації та статус онлайн. Пароль зберігається як хеш через Werkzeug і не може бути прочитаний напряму - тільки перевірений через метод verify_password().

Клас Company описує компанію: назва, власник (owner_id → User), аватар, дата створення. Компанія є контейнером для всіх бізнес-даних – проєктів, рахунків, складів, клієнтів тощо.

Клас Employee зв'язує User та Company – один користувач може бути співробітником кількох компаній. Employee містить посаду (position), зарплату, дату найму, лідера (leader_id для побудови ієрархії) та зв'язок із таблицею Navbar для прав доступу.

Модуль проєктів включає класи Pipeline (пайплайн зі стадіями), PipelineStage (стадія з порядковим номером), Project (проєкт, прив'язаний до пайплайну та стадії) і ProjectTask (задача з полями title, description, status, priority, due_date, is_closed). Задачі підтримують підзадачі через self-referencing зв'язок (parent_task_id) та виконавців через проміжну таблицю TaskExecutor.

Фінансовий модуль побудований на класах Account (банківський рахунок з available_balance та present_balance), Transaction (операція з типом income/expense та прив'язкою до рахунку), Budget (бюджет з лімітом та поточними витратами) і Bill (рахунок-фактура з прив'язкою до бюджету).

Складський модуль включає Warehouse (склад з назвою та адресою), InventoryItem (товар з кількістю, ціною, категорією) та WarehouseAction (дія: переміщення, відвантаження, поповнення).

Модулі продажів та закупівель побудовані на класах Customer/Provider (клієнт/постачальник з контактними даними), Offering (комерційна пропозиція) та Order (замовлення з позиціями).

Комунікаційний модуль включає Chat (чат з учасниками через ChatMember), Message (повідомлення з текстом, файлами, статусом прочитання), Inbox (канал підтримки) та Notification (сповіщення з типом, заголовком, описом та URL для переходу).

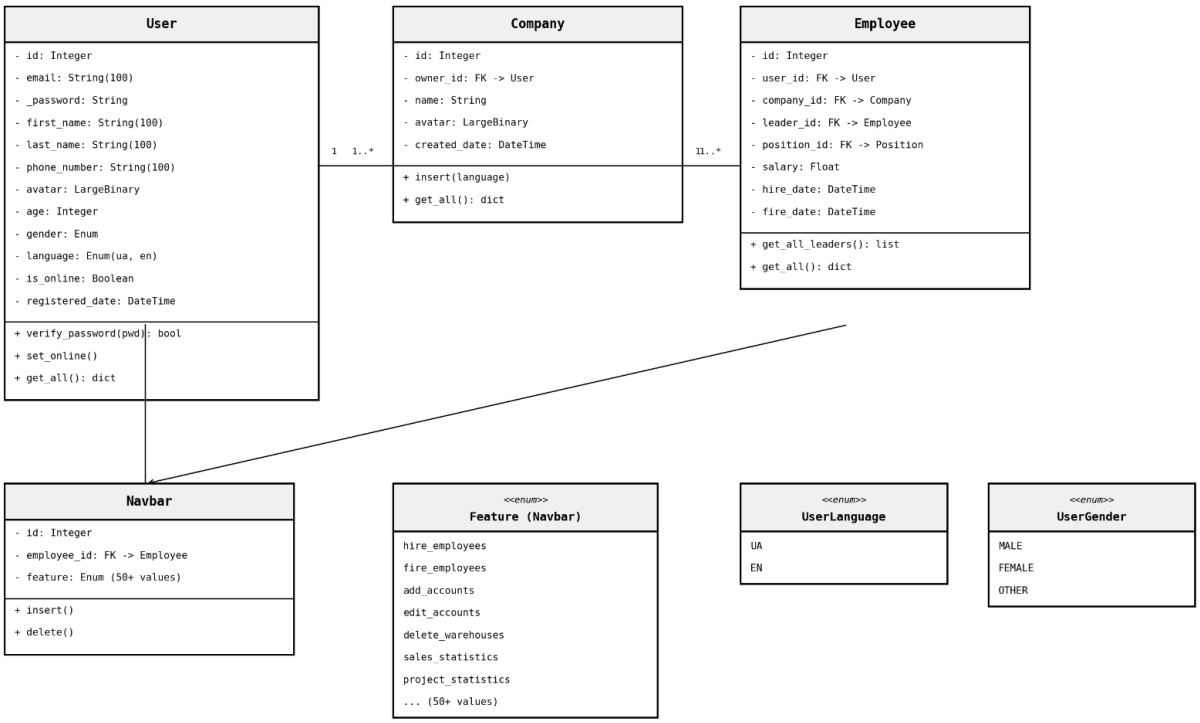


Рис. 2.4. Структурна діаграма класів(User, Company, Employee, Navbar)

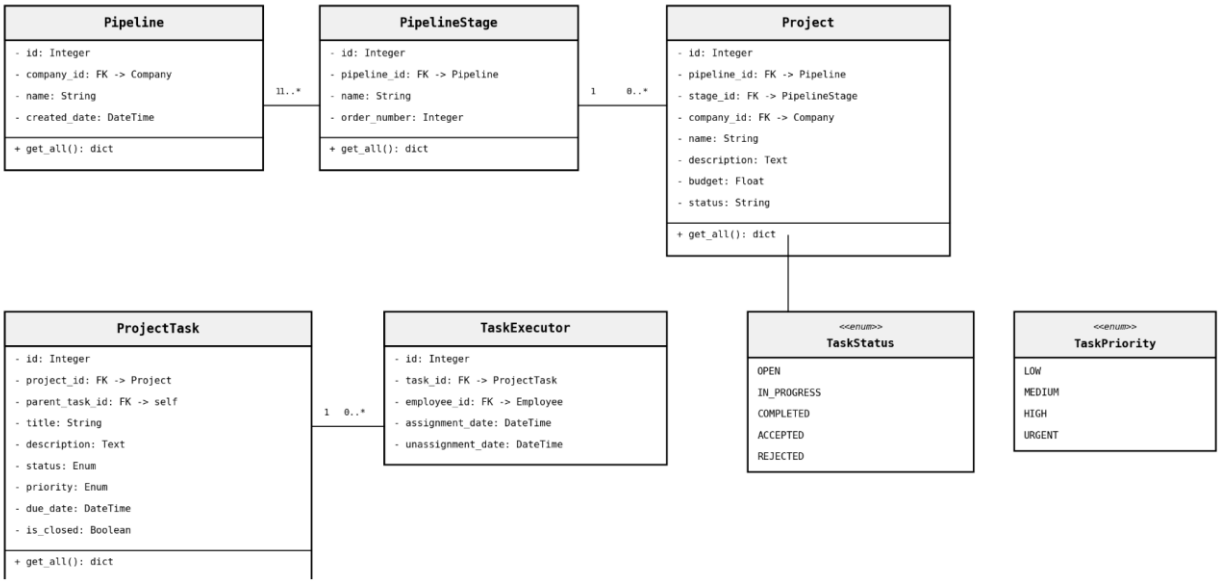


Рис. 2.5. Структурна діаграма класів – модуль проєктів

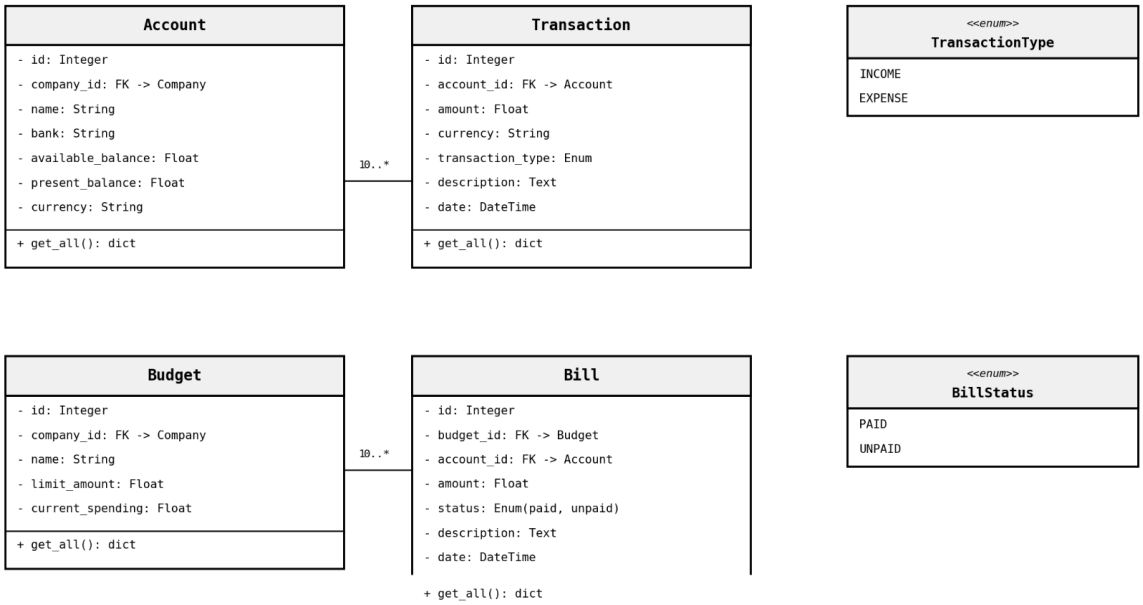


Рис 2.6. Структурна діаграма класів – модуль фінансів

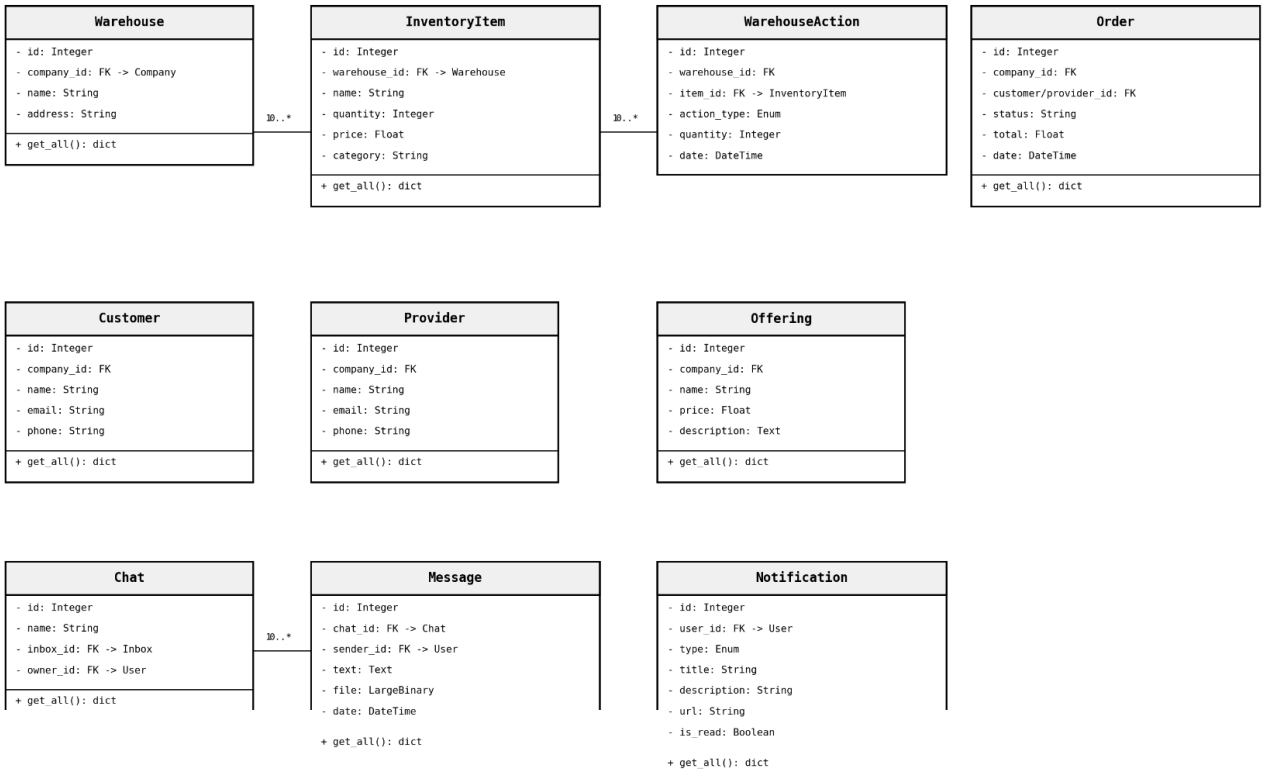


Рис 2.7. Структурна діаграма класів – модуль склад, продажі, чат

2.5 Алгоритм класифікації типу фігури на основі нечітких правил

Модуль управління проєктами є одним із центральних у системі Ri Dren. Він побудований на ієрархічній структурі: Pipeline → Project → Task → Subtask.

Пайплайн (Pipeline) – це верхній рівень організації, який містить стадії (PipelineStage). Кожна стадія має назву та порядковий номер. Проєкти створюються всередині пайплайну і прив'язуються до конкретної стадії. Переміщення проєкту між стадіями відбувається через PATCH-запит на /api/pipelines/{id}/projects.

Серверне API для пайплайнів реалізовано через Flask-RESTful ресурси:

− POST /api/company/{id}/pipelines – створення нового пайплайну;

− GET /api/company/{id}/pipelines – отримання списку пайплайнів компанії;

− GET /api/pipelines/{id} – отримання конкретного пайплайну зі стадіями;

− PUT /api/pipelines/{id} – редагування пайплайну;

− DELETE /api/pipelines/{id} – видалення пайплайну;

− POST /api/pipelines/{id}/stages – додавання стадії;

− GET /api/pipelines/{id}/projects – отримання проєктів пайплайну.

Задачі (ProjectTask) створюються через POST /api/task/create і містять заголовок, опис, пріоритет, дедлайн, статус та прив'язку до проєкту. Статуси задач: Open, In Progress, Completed, Accepted, Rejected. При зміні статусу система автоматично надсилає сповіщення виконавцям та їхнім керівникам через систему нотифікацій.

Підзадачі реалізовані через self-referencing зв'язок - поле parent_task_id вказує на батьківську задачу. Отримати підзадачі можна через GET /api/task/{id}/subtasks.

Для кожної задачі доступний time tracker – співробітник може логувати витрачений час через POST /api/task/{id}/time-tracker. Дані про час використовуються для звітів по продуктивності.

Також реалізовано автоматичні нагадування: APScheduler перевіряє задачі, дедлайн яких настає протягом 24 годин, і надсилає сповіщення виконавцям та їхнім лідерам. Прострочені задачі також отримують окреме сповіщення.

Фрагмент коду створення задачі та приклад API-відповіді зображено на рисунку 2.6.

```
118  class CreateTask(Resource):
119      @jwt_required()
120      def post(self):
121          # Get the data from request
122          data = request.get_json()
123
124          project_id = data.get('project_id')
125          parent_task_id = data.get('parent_task_id')
126          stage_id = data.get('stage_id')
127          title = data.get('title')
128          description = data.get('description')
129          date = data.get('date')
130          due_date = data.get('due_date')
131          executors = data.get('executors')
132          labels = data.get('labels')
133          skills = data.get('skills')
134
135          #warehouse action
136          origin_warehouse_id = data.get("origin_warehouse_id")
137          destination_warehouse_id = data.get("destination_warehouse_id")
138
139          origin_address_id = data.get("origin_address_id")
140          destination_address_id = data.get("destination_address_id")
141
142          items = data.get('items')
143
144          if project_id:
145              # Get project
146              project = Project.query.get(project_id)
147              if not project:
148                  return api_abort(404, f"No projects exist with Id: {project_id}")
149
150              current_employee = is_feature_allowed(project.company_id, "add_tasks")
151
```

Рис. 2.8. Фрагмент коду створення задачі та приклад API-відповіді

2.6 Модуль фінансового менеджменту

Фінансовий модуль системи Ri Dren охоплює чотири основні сутності: рахунки (accounts), транзакції (transactions), бюджети (budgets) та рахунки-фактури (bills). Кожна з них має повний набір CRUD-операцій та додаткові ендпоінти для фільтрації, огляду та статистики.

Рахунки (Account) - це банківські рахунки компанії. Кожен рахунок має назву, банк, валюту, доступний та поточний баланс, опис. API для роботи з рахунками:

− POST /api/company/{id}/accounts – створення рахунку;

− GET /api/company/{id}/accounts – список рахунків з фільтрацією та пагінацією;

− GET /api/company/{id}/accounts/overview – зведена інформація (загальний баланс, кількість рахунків);

− PUT /api/accounts/{id} – редагування рахунку;

− DELETE /api/accounts/{id} – видалення рахунку.

Транзакції (Transaction) прив'язані до конкретного рахунку. При створенні транзакції типу "income" баланс рахунку збільшується, при "expense" – зменшується. Транзакції мають суму, валюту, опис, дату та тип. Список транзакцій доступний як для всієї компанії (GET /api/company/{id}/transactions), так і для конкретного рахунку (GET /api/accounts/{id}/transactions).

Бюджети (Budget) дозволяють задати ліміт витрат на певний період або напрямок. До бюджету прив'язуються рахунки-фактури (bills), і система автоматично підраховує поточні витрати. Якщо витрати перевищують ліміт, спрацьовує механізм overflow resolve – GET /api/budget/{id}/overflow/resolve повертає рекомендації щодо перерозподілу коштів.

Рахунки-фактури (Bill) – це документи на оплату, прив'язані до бюджету. Кожен bill має суму, статус (оплачений/неоплачений), дату та опис. При оплаті bill система створює відповідну транзакцію на прив'язаному рахунку.

Для фінансового модуля реалізовано окремий дашборд статистики (GET /api/company/{id}/finance/statistics) та модуль фінансових звітів (GET /api/company/{id}/finance/reports), які агрегують дані по рахунках, транзакціях та бюджетах за обраний період.

Фрагмент коду фінансового модуля зображено на рисунку 2.6.

```python
class CompanyBudgetFilters(Resource):
    @jwt_required()
    def get(self, company_id):
        can_create = can_access(company_id, "add_budgets")

        # Query to get the minimum and maximum sum of amounts
        min_max_query = db.session.query(
            func.min(Budget.amount_uah).label('min_amount_uah'),
            func.max(Budget.amount_uah).label('max_amount_uah'),
            func.min(Budget.amount_usd).label('min_amount_usd'),
            func.max(Budget.amount_usd).label('max_amount_usd'),
            func.min(Budget.amount_eur).label('min_amount_eur'),
            func.max(Budget.amount_eur).label('max_amount_eur')
        ).filter(
            Budget.company_id == company_id
        ).subquery()

        # Fetch the minimum and maximum amounts
        min_max_amounts = db.session.query(
            min_max_query.c.min_amount_uah,
            min_max_query.c.max_amount_uah,
            min_max_query.c.min_amount_usd,
            min_max_query.c.max_amount_usd,
            min_max_query.c.min_amount_eur,
            min_max_query.c.max_amount_eur
        ).first()

        # Construct filters
        filters = {
            "filters": [
                {
                    "type": "RANGE",
                    "title": "amount_uah",
                    "data": {
                        "min": min_max_amounts.min_amount_uah if min_max_amounts else 0,
                        "max": min_max_amounts.max_amount_uah if min_max_amounts else 0
                    },
                    "key": "amount_uah"
                },
                {
                    "type": "RANGE",
                    "title": "amount_usd",
                    "data": {
                        "min": min_max_amounts.min_amount_usd if min_max_amounts else 0,
                        "max": min_max_amounts.max_amount_usd if min_max_amounts else 0
                    },
                    "key": "amount_usd"
```

Рис. 2.6. Фрагмент коду фінансового модуля

2.7 Структура бази даних

Структура бази даних web-застосунку Ri Dren побудована на реляційній моделі в PostgreSQL та охоплює основні сутності, пов'язані з користувачами, компаніями, проєктами, фінансами, складом, продажами та комунікацією.

Центральними таблицями є users (облікові записи з хешованими паролями, профільними даними та мовою інтерфейсу), companies (компанії з власником та контактними даними) та employees (зв'язок користувача з компанією, посада, зарплата, лідер для ієрархії). Таблиця navbars зберігає права доступу - кожен запис це пара employee_id + feature (наприклад, "add_accounts", "delete_warehouses").

Модуль проєктів представлений таблицями pipelines (пайплайни), pipeline_stages (стадії з порядковим номером), projects (проєкти з прив'язкою до пайплайну та стадії), project_tasks (задачі з полями status, priority, due_date, is_closed, parent_task_id для підзадач) та task_executors (виконавці задач з датами призначення та зняття).

Фінансовий модуль включає таблиці accounts (рахунки з балансами), transactions (транзакції з типом income/expense), budgets (бюджети з лімітом та поточними витратами) та bills (рахунки-фактури з прив'язкою до бюджету та статусом оплати).

Складський модуль побудований на таблицях warehouses (склади), inventory_items (товари з кількістю, ціною, категорією) та warehouse_actions (дії: переміщення, відвантаження, поповнення з зазначенням кількості та дати).

Модулі продажів та закупівель використовують таблиці customers (клієнти), providers (постачальники), offerings (комерційні пропозиції) та orders з order_items (замовлення з позиціями).

Комунікаційний модуль включає таблиці chats (чати), chat_members (учасники), messages (повідомлення з текстом та файлами), message_statuses (статус прочитання), inboxes (канали підтримки) та notifications (сповіщення з типом, заголовком та URL).

Додатково реалізовано таблиці labels (мітки для категоризації), positions (посади), skills (навички співробітників), extra_columns та extra_column_data (кастомні поля для будь-якої сутності – аналог custom fields), loggers (журнал помилок) та users_online (відстеження онлайн-статусу).

Усього модель даних включає понад 30 таблиць із зв'язками через ForeignKey та relationship. Уся структура адаптована для роботи з SQLAlchemy ORM та підтримує автоматичні міграції через Flask-Migrate.

Структура бази даних наведена на рисунку 2.7.

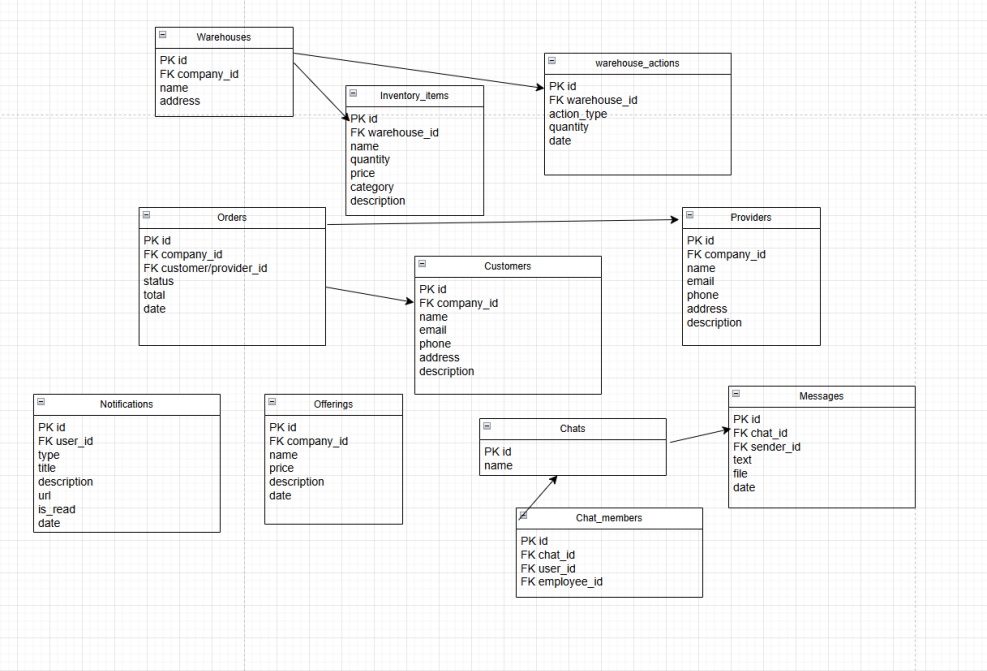


Рис. 2.7. Структура бази даних

3 РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Структура навігації web-застосунку

У процесі розробки web-застосунку Ri Dren одним із перших етапів стала побудова логічної структури навігації. Від того, наскільки зрозуміло організовані переходи між сторінками, залежить швидкість роботи користувача і загальне враження від системи. Тому маршрутизацію було спроєктовано з урахуванням двох основних зон: публічної та захищеної.

Публічна зона доступна без авторизації і включає лендінг (головну сторінку продукту), сторінки "Про нас" та "Опис продукту", форми логіну та реєстрації, підтвердження email через OTP-код, скидання пароля, а також розділ мануалів (документації) по кожному модулю системи. Маршрутизація реалізована через React Router v6 - кожна сторінка відповідає окремому Route-компоненту.

Захищена зона (/ri-business/*) доступна тільки після авторизації. Вона містить головну сторінку кабінету, з якої користувач може перейти до будь-якого модуля: проєкти (пайплайни, проєкт, задача), фінанси (рахунки, транзакції, бюджети, bills, звіти), склад (склади, інвентар, переміщення, відвантаження), продажі (клієнти, пропозиції, замовлення), закупівлі (постачальники, замовлення). Також доступні загальні сторінки: профіль, чат, сповіщення, налаштування, статистика по кожному модулю.

Діаграма навігації головної сторінки зображена на рисунку 3.1.

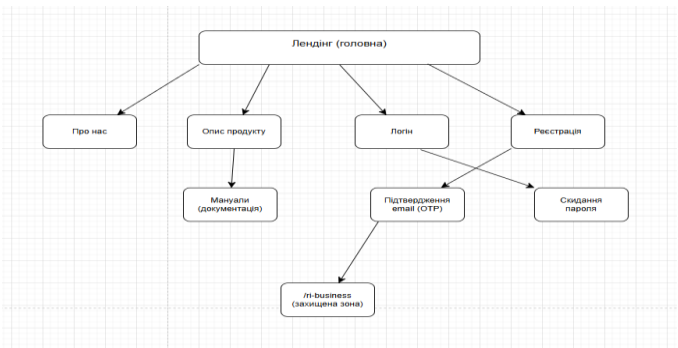


Рис. 3.1. Діаграма навігації головної сторінки

Друга діаграма відображає структуру кабінету користувача. Тут доступні переходи до всіх бізнес-модулів із підсторінками, а також до загальних розділів – профілю, чату, сповіщень та налаштувань.

Діаграма навігації кабінету користувача зображена на рисунку 3.2.

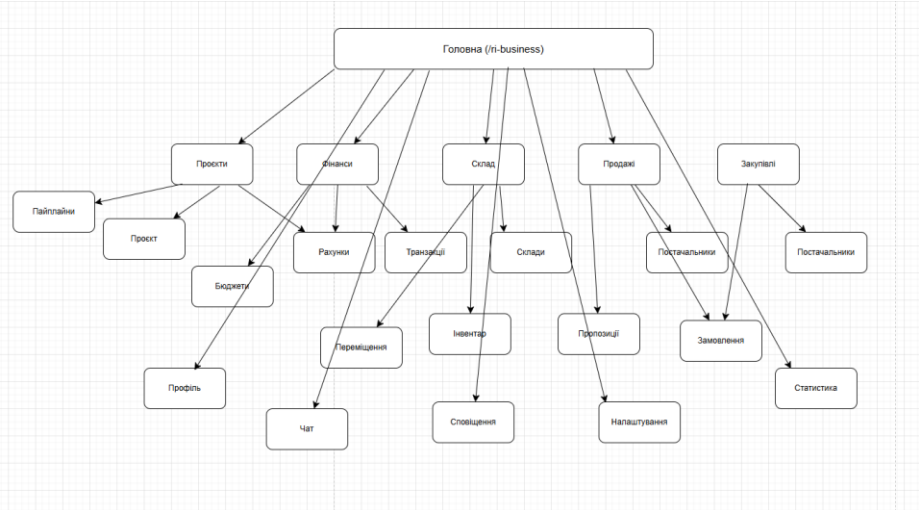


Рис. 3.2. Діаграма навігації особистого кабінету користувача

Третя діаграма показує панель адміністратора, яка має три основні гілки: управління співробітниками (запрошення, ієрархія, пошук), налаштування прав доступу (Navbar, позиції, мітки) та адміністрування компанії (налаштування, логи, статистика по всіх модулях).

Діаграма навігації панелі адміністратора зображена на рисунку 3.3.

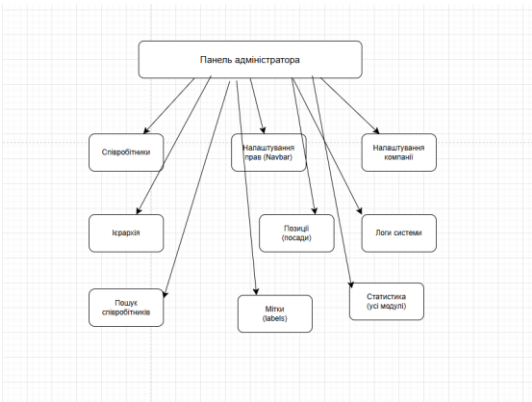


Рис. 3.3. Діаграма навігації панелі адміністратора

Структура навігації реалізована відповідно до принципів юзабіліті: інтерфейс логічно розділений за модулями, користувачі легко орієнтуються між сторінками, а адміністратор має доступ до необхідних функцій з одного централізованого інтерфейсу.

3.2 Реалізація інтерфейсу системи

Інтерфейс web-застосунку Ri Dren розроблено з акцентом на зручність та швидкість роботи. Оскільки система призначена для щоденного використання співробітниками компанії, було важливо зробити інтерфейс таким, щоб основні дії виконувалися за мінімальну кількість кліків. Верстка адаптивна для десктопних екранів, стилізація виконана через SCSS з підтримкою світлої та темної тем.

На наступних рисунках наведено ключові інтерфейсні компоненти, реалізовані у проєкті.

На рисунку 3.4 зображено лендінг – публічну головну сторінку, яка представляє продукт Ri Dren. Тут розміщено опис можливостей системи, кнопки переходу до реєстрації та логіну, а також посилання на документацію.

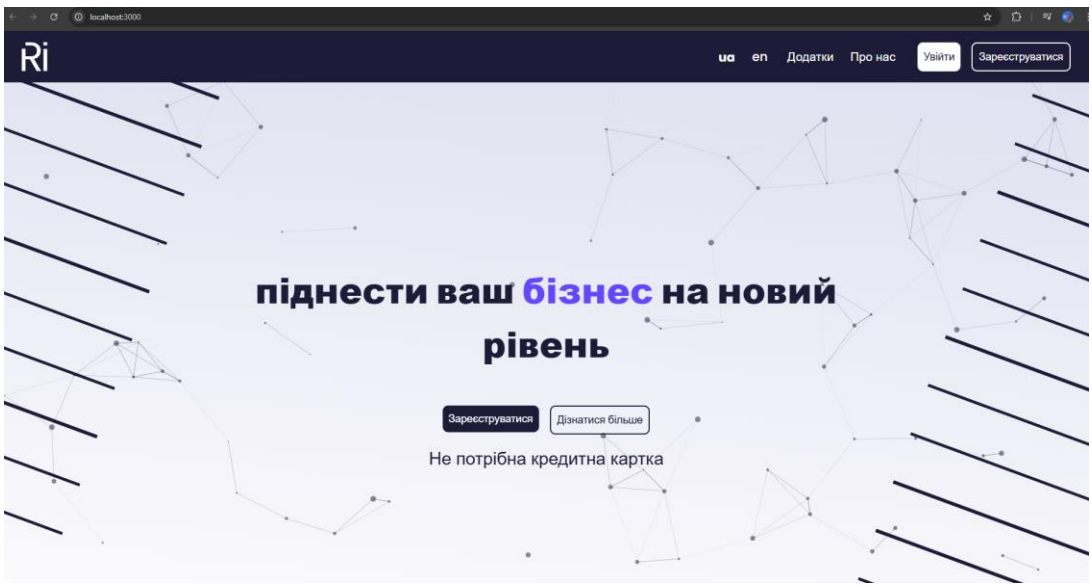


Рис. 3.4 Головна сторінка web-застосунку

На рисунку 3.5 представлено форму авторизації. Користувач вводить email та пароль, після чого система перевіряє дані і видає JWT-токен. Є опція "запам'ятати мене", яка збільшує термін дії refresh-токена з 1 до 30 днів.

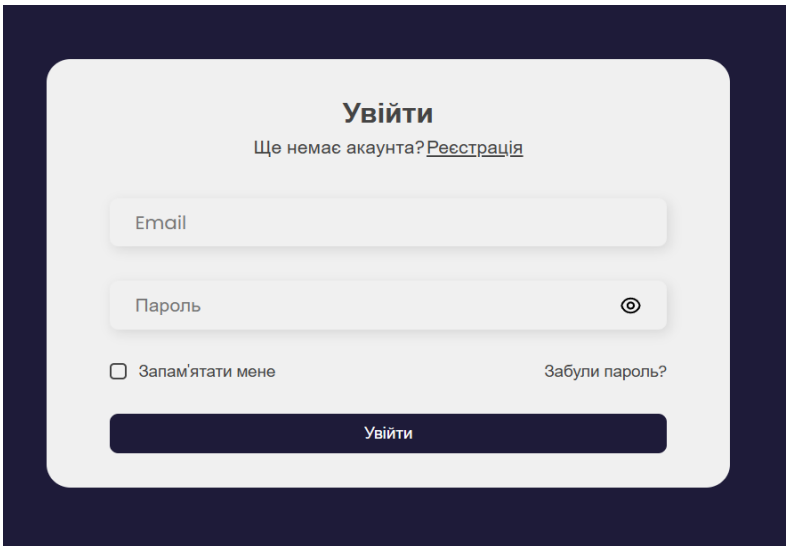


Рис. 3.5. Форма «Увійти»

Рисунок 3.6 ілюструє форму «Зареєструватись», яка дозволяє створити новий обліковий запис. Для завершення реєстрації обов’язково потрібно вказати електронну пошту, пароль, його підтвердження.

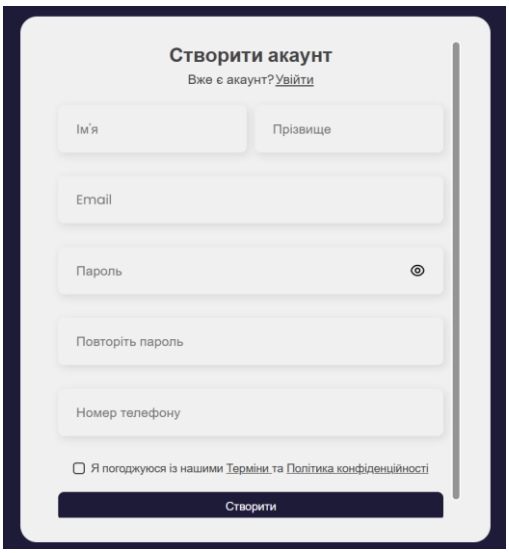


Рис. 3.6. Форма «Зареєструватись»

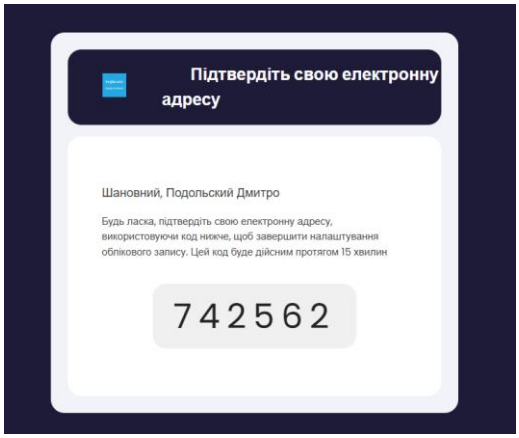


Рис. 3.7. Форма код підтвердження

На рисунку 3.8 зображено головну сторінку кабінету після авторизації. Бічна панель навігації дозволяє швидко переходити між модулями. На головній сторінці відображаються зведені дані - останні задачі, сповіщення, швидкі дії.

Рис. 3.8. Каталог усіх товарів з фільтром

Рисунок 3.9 демонструє модуль управління проєктами - список пайплайнів зі стадіями. Кожен пайплайн відображається як набір колонок (стадій), у яких розміщені картки проєктів. Проєкти можна перетягувати між стадіями.

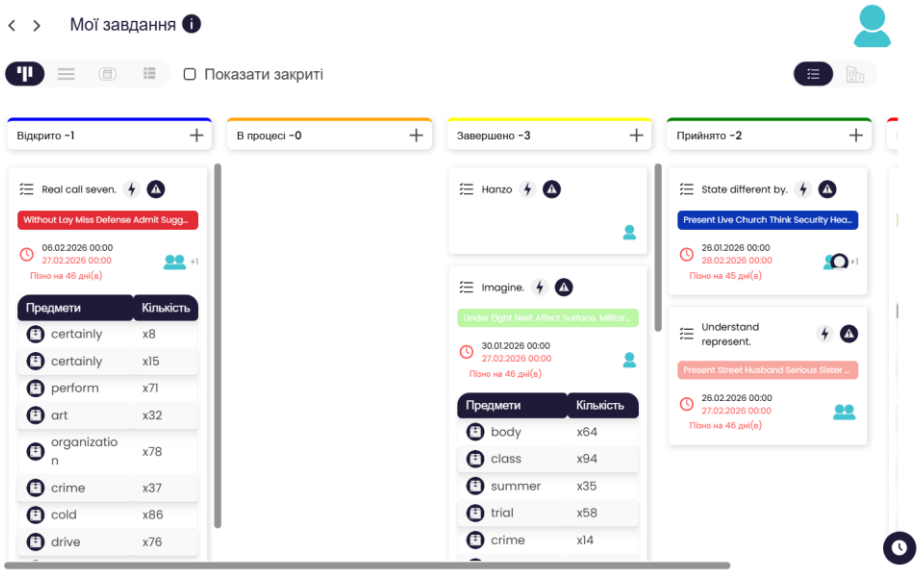


Рис. 3.9. Сторінка деталей товару

На рисунку 3.10-3.11 зображено сторінку конкретної задачі. Тут відображається заголовок, опис, статус, пріоритет, дедлайн, список виконавців, підзадачі та time tracker. Статус можна змінити через випадаючий список.

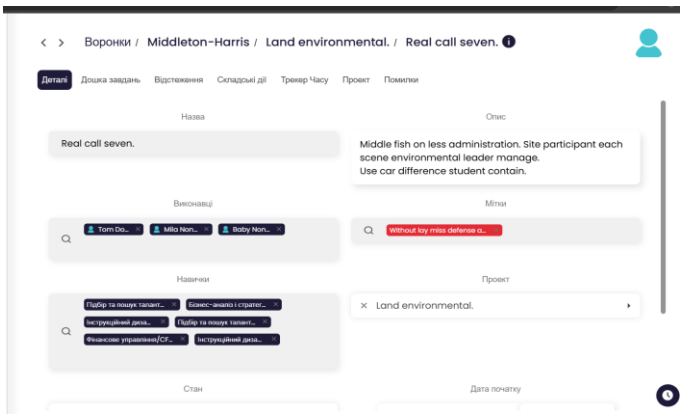


Рис. 3.10. Сторінка задачі

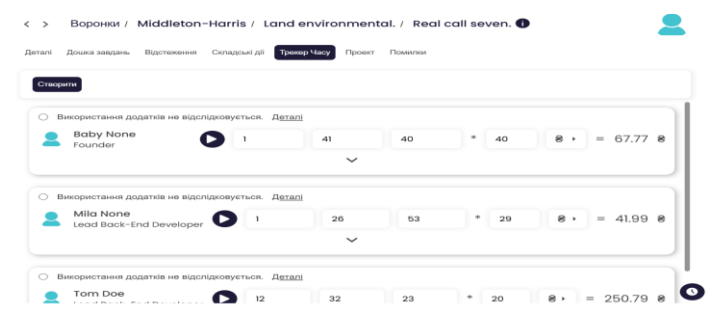


Рис. 3.11. Сторінка задачі з підзадачами та time tracker

Рисунок 3.12 демонструє фінансовий модуль – список банківських рахунків з балансами та можливістю фільтрації. При натисканні на рахунок відкривається сторінка з транзакціями.

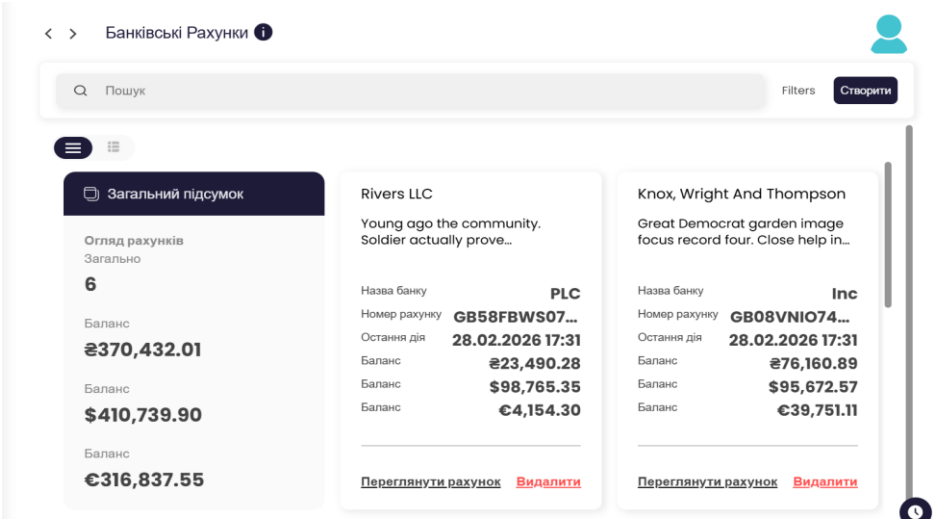


Рис. 3.12. Список банківських рахунків

На рисунку 3.13 зображено сторінку бюджету з транзакціями де можна побачити витрати та дохід від операцій

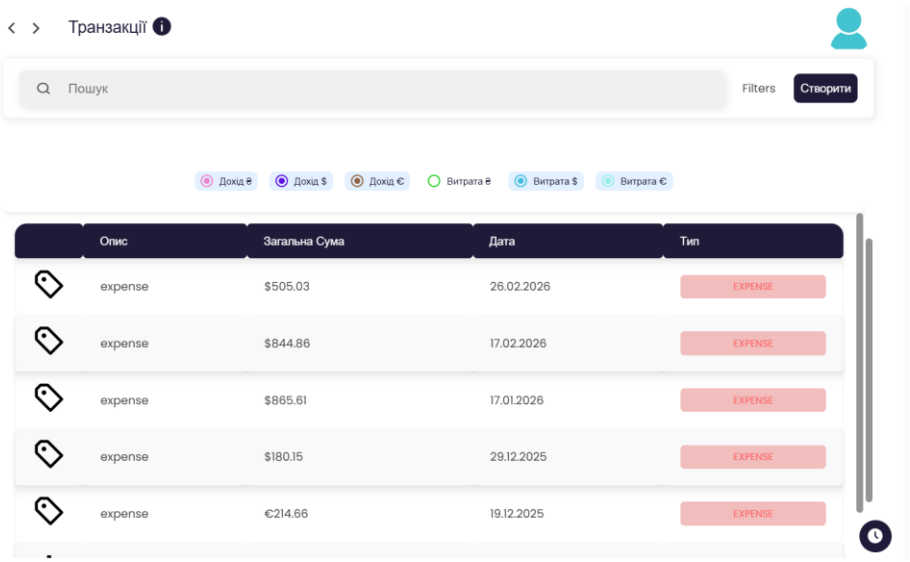


Рис. 3.13. Сторінка транзакцій

Рисунок 3.14 ілюструє складський модуль – список складів з інвентарними позиціями. Для кожного товару відображається кількість, ціна та категорія.

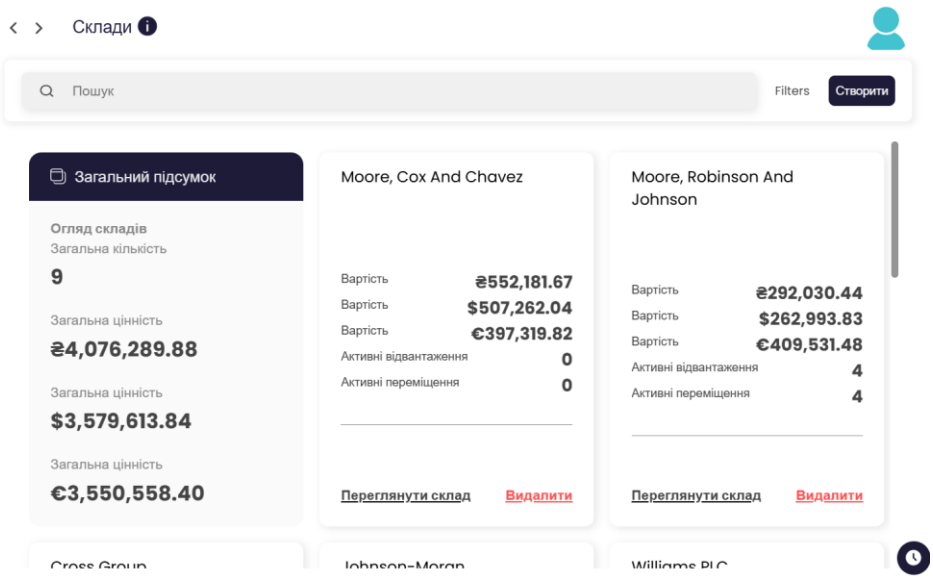


Рис. 3.14. Складський модуль - список складів та інвентар

На рисунку 3.15 зображено модуль продажів – список клієнтів з контактними даними та історією замовлень.

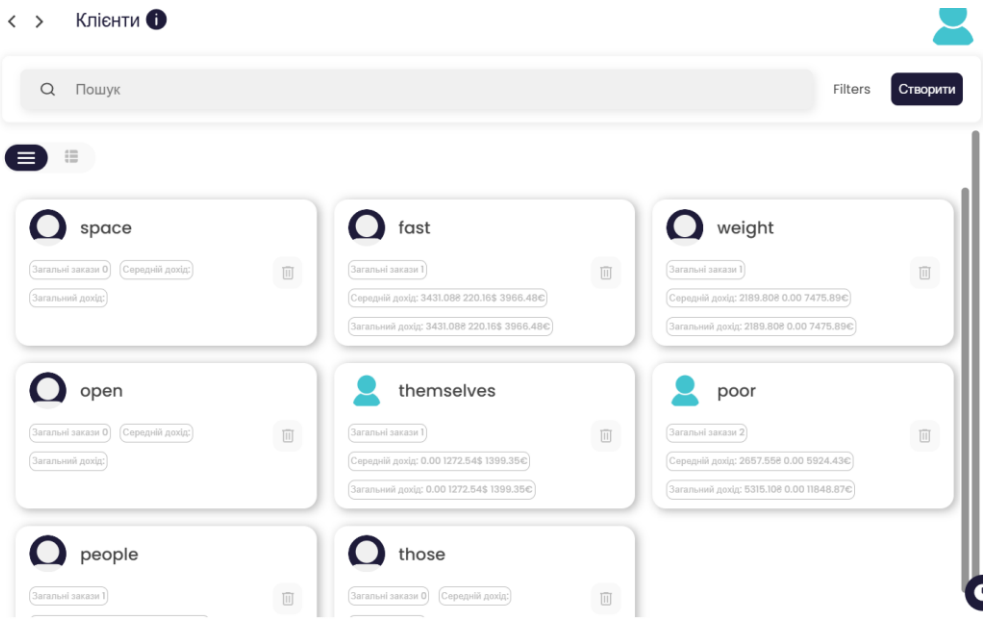


Рис. 3.15. Модуль продажів - список клієнтів

Рисунок 3.16 демонструє внутрішній чат – список бесід зліва та вікно повідомлень справа. Підтримуються особисті та групові чати, а також inbox-канали для підтримки.

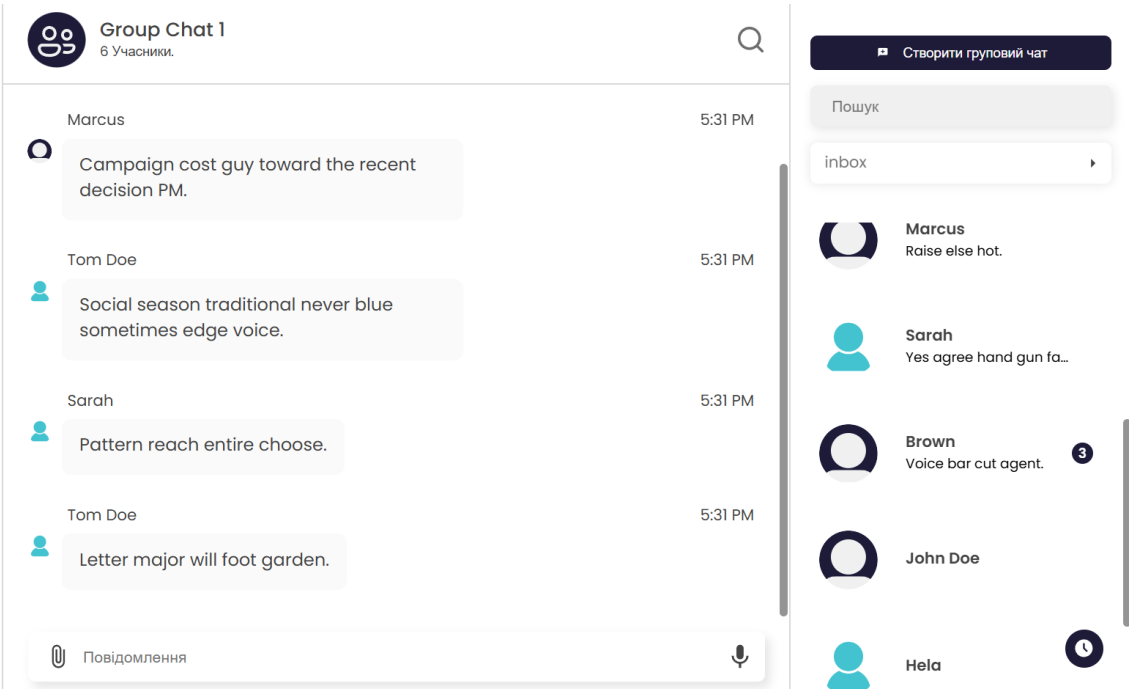


Рис. 3.16. Внутрішній чат

На рисунку 3.17 зображено ієрархію співробітників – деревоподібна структура, де видно підпорядкування між працівниками. Адміністратор може переміщувати співробітників між рівнями.

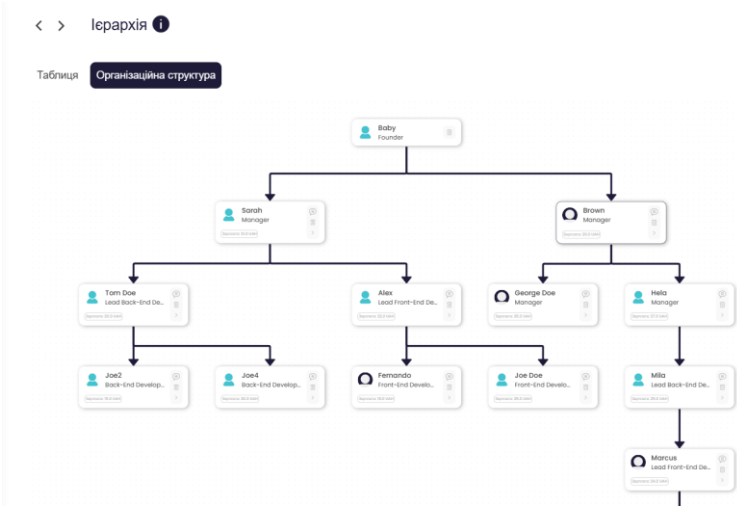


Рис. 3.17. Ієрархія співробітників

Рисунок 3.18 ілюструє сторінку всіх сповіщень від співробітників компаній

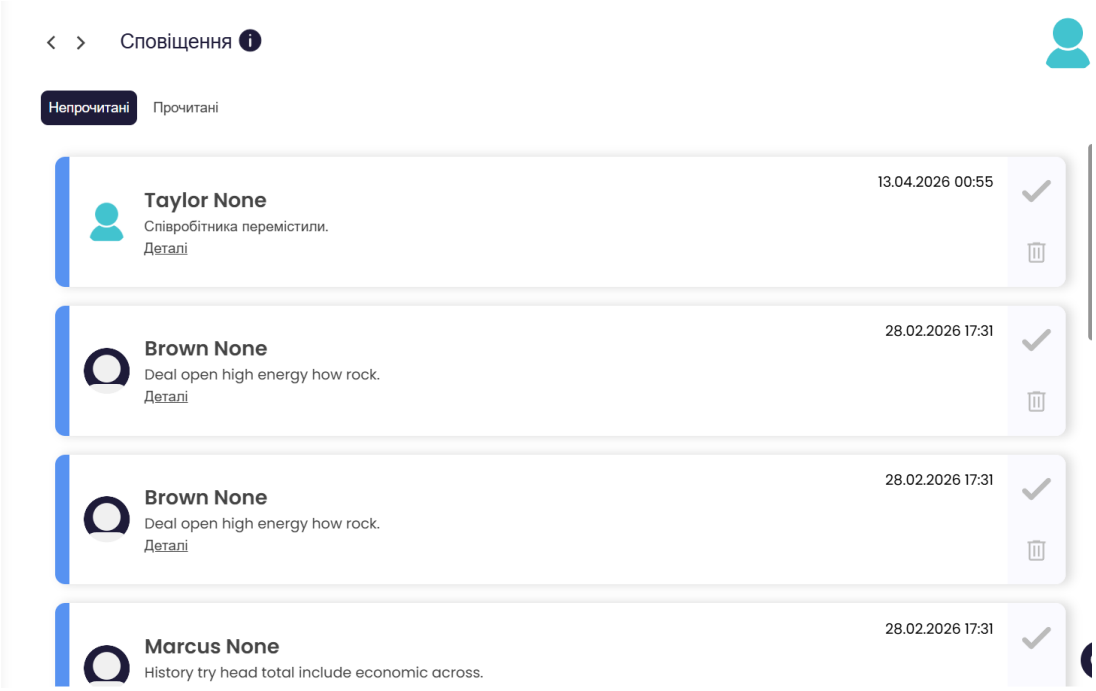


Рис. 3.18. Сторінка всіх сповіщень

3.3 Кросбраузерне тестування

Кросбраузерне тестування проводилося для перевірки стабільності інтерфейсу, коректності стилів, адаптивної верстки та взаємодії з функціональними модулями. Тестування здійснювалося вручну на чотирьох браузерах: Google Chrome (версія 120+), Mozilla Firefox (версія 121+), Opera (версія 105+) та Microsoft Edge (версія 120+).

Тестування охоплювало такі ключові елементи:

– лендінг та форми авторизації/реєстрації;

– головну сторінку кабінету з бічною навігацією;

– модуль проєктів – пайплайни, задачі, time tracker;

– фінансовий модуль – рахунки, транзакції, бюджети;

– складський модуль – склади, інвентар, переміщення;

– внутрішній чат та систему сповіщень;

– налаштування прав доступу та ієрархію співробітників;

– статистичні дашборди з графіками (ApexCharts).

Результати кросбраузерного тестування зображено на рисунках 4.18-4.21.

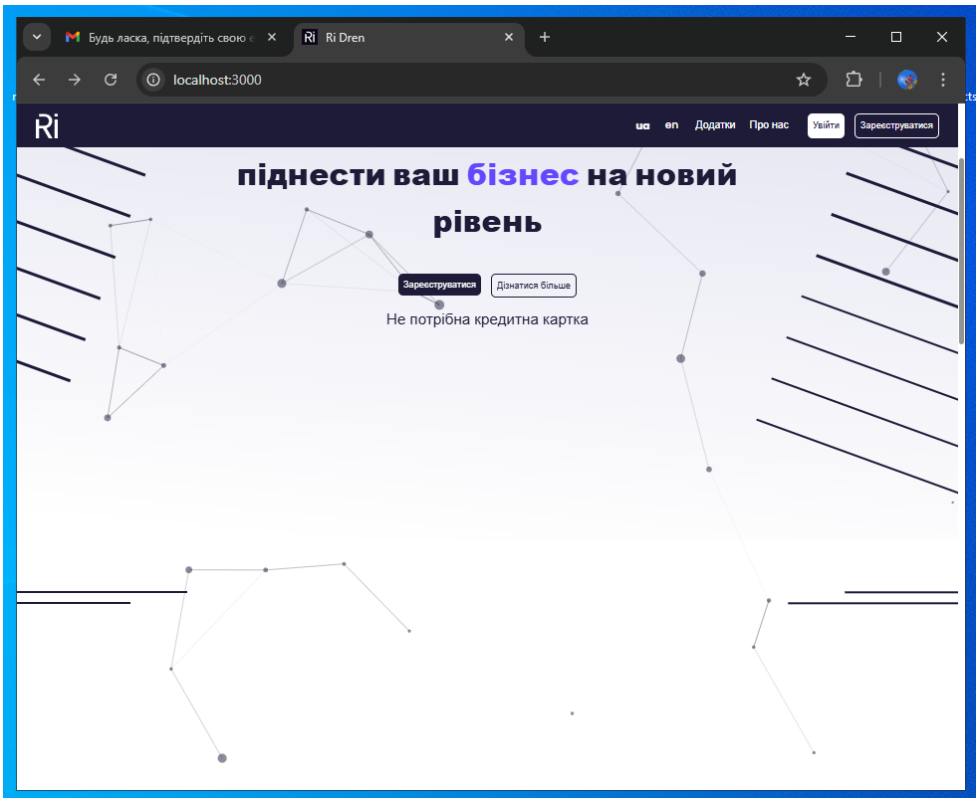


Рис. 3.18. Google Chrome

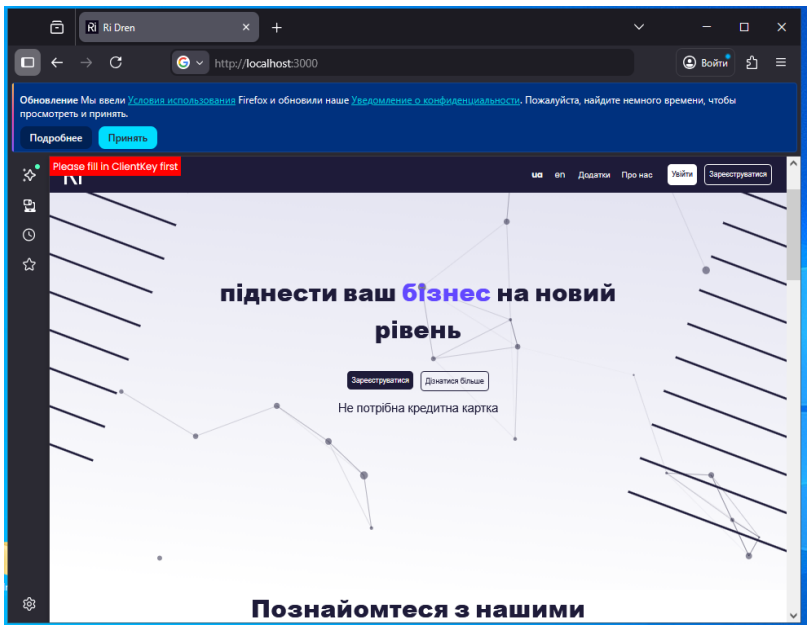


Рис. 3.19. Mozilla Firefox

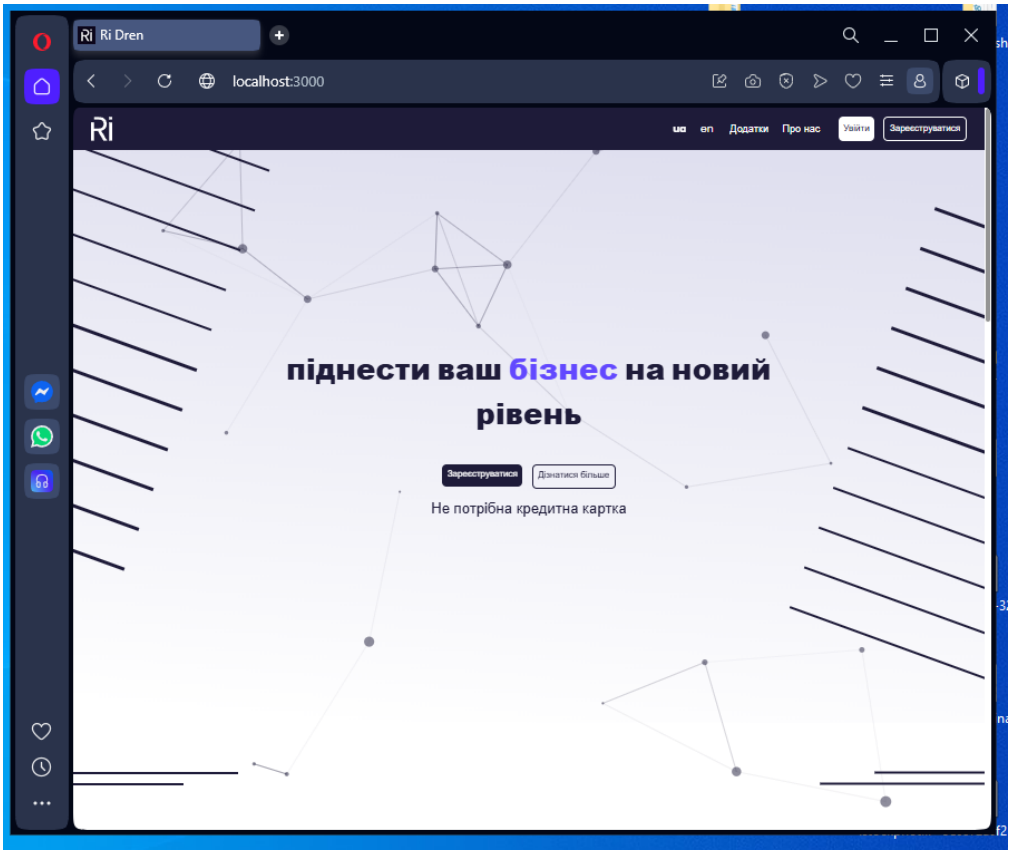


Рис. 3.20. Opera

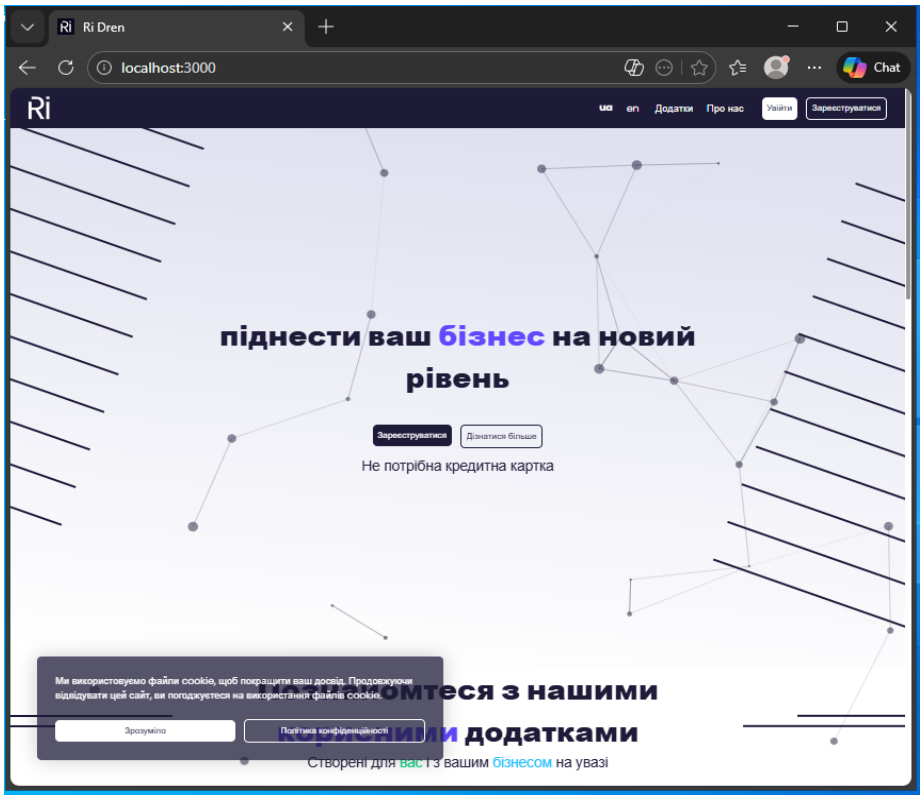


Рис. 3.21. Microsoft Edge

За підсумками тестування встановлено, що web-застосунок відповідає стандартам кросбраузерної сумісності, зберігаючи однакову функціональність і дизайн на всіх перевірених платформах. Незначні відмінності спостерігалися лише у рендерингу шрифтів та тіней – вони не впливають на функціональність і є типовими для різних браузерних движків.

3.3 Тестування функціональної логіки за допомогою Postman та unit-тестів

Для забезпечення надійності ключових елементів бізнес-логіки web-застосунку було проведено два види тестування: функціональне тестування API через Postman та модульне тестування (unit testing) за допомогою фреймворку flask_unittest. Основною метою стало підтвердження коректної роботи найважливіших ендпоінтів та перевірка обробки граничних випадків.

3.3.1 Тестування модуля реєстрації

Реєстрація нового користувача – один із найкритичніших процесів у системі, оскільки від нього залежить подальша робота з усіма модулями. Тестування охоплювало повний цикл: відправка запиту на верифікацію email (POST /api/verification/email/send), підтвердження OTP-коду (POST /api/verification/email/verify) та створення компанії (POST /api/company/create).

Через Postman було перевірено такі сценарії:

− успішна реєстрація з коректними даними – система повертає 200 та дані створеного користувача;

− спроба реєстрації з email, який вже зайнятий – система повертає 409 (Conflict);

− відправка запиту без обов'язкових полів – система повертає 400 з переліком відсутніх полів;

− введення невірного OTP-коду – система повертає 400 з описом помилки.

Результати тестування модуля реєстрації наведені на рисунку 3.22.

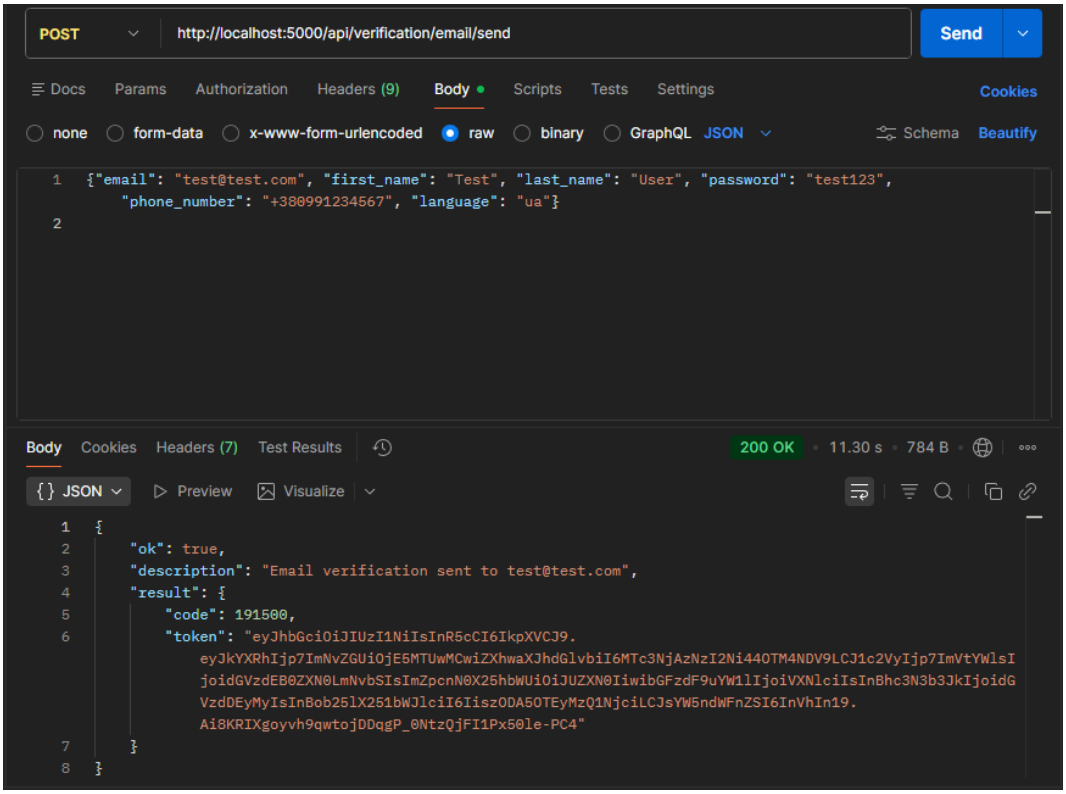


Рис. 3.22. Тестування модуля реєстрації в Postman

3.3.2 Тестування API управління проєктами

Модуль управління проєктами тестувався як через Postman, так і через unit-тести. У класі TestPipelines реалізовано 34 тестових кейси, які покривають повний CRUD-цикл для пайплайнів та стадій.

Тестові сценарії включають:

– створення пайплайну (POST /api/company/1/pipelines) – перевірка статусу 200 та відповідності назви;

– спроба створення без авторизації – очікується 401 (Unauthorized);

– спроба створення в чужій компанії – очікується 403 (Forbidden);

– створення в неіснуючій компанії – очікується 404 (Not Found);

– редагування пайплайну з порожнім тілом – система повертає 200 з повідомленням "Nothing to Update";

– створення стадії з дублюючим order_number – очікується 403;

– видалення пайплайну та перевірка каскадного видалення стадій.

Кожен тест починається з авторизації через POST /api/login і отримання JWT-токена, який передається у заголовку Authorization. Це дозволяє перевірити не тільки бізнес-логіку, а й систему безпеки.

Результати тестування API управління проєктами наведені на рисунку 3.23.

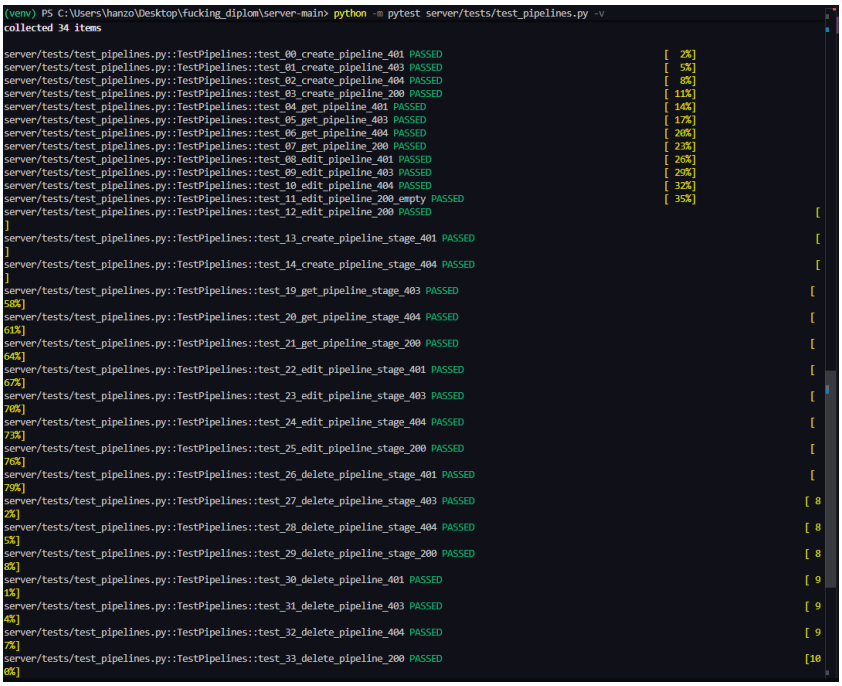


Рис. 3.23. Результати unit-тестів TestPipelines

3.3.3 Тестування фінансових операцій

Фінансовий модуль тестувався через клас TestAccounts, який містить 27 тестових кейсів для рахунків та транзакцій. Тести покривають створення, редагування, видалення рахунків, а також створення та редагування транзакцій на рахунках.

Ключові тестові сценарії:

– створення рахунку з повними даними (name, bank, available_balance, present_balance, currency, description) - очікується 200;

– створення рахунку в неіснуючій компанії - очікується 404;

– створення рахунку в чужій компанії - очікується 403;

– створення рахунку без обов'язкових полів - очікується 400;

– створення транзакції на рахунку (POST /api/accounts/2/transactions) з перевіркою зміни балансу;

– редагування транзакції – зміна суми, валюти, типу;

– видалення рахунку та перевірка каскадного видалення транзакцій.

Тести підтверджують правильність логіки фінансових операцій: при створенні транзакції типу "income" баланс рахунку збільшується, при "expense" – зменшується. Також перевірено, що неавторизовані користувачі не можуть виконувати операції з чужими рахунками.

Результати тестування фінансових операцій наведені на рисунку 3.24.

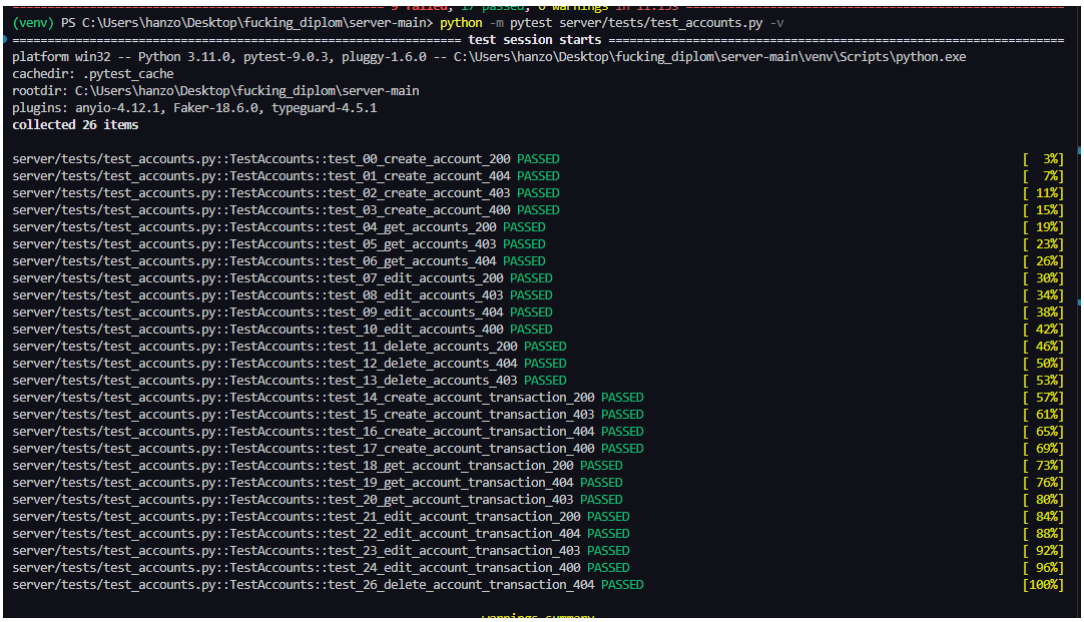


Рис. 3.24. Результати unit-тестів TestAccounts

ВИСНОВКИ

У результаті виконаної кваліфікаційної роботи було створено web-застосунок Ri Dren для корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту. Розроблена система дозволяє малим та середнім компаніям вести проєкти, фінанси, складський облік, продажі та закупівлі в єдиному інтерфейсі, замість використання кількох розрізнених інструментів.

У процесі роботи було виконано такі задачі:

1. Проведено аналіз існуючих корпоративних платформ – Asana, Monday.com та Wrike. Виявлено, що жодна з них не поєднує управління проєктами з фінансовим обліком, складським управлінням та вбудованою комунікацією. Усі три платформи зосереджені на трекінгу задач і не мають модулів для ведення рахунків, бюджетів, складських залишків чи замовлень. Також виявлено обмеженість рольових моделей та високу вартість для команд від 20 осіб ($200-500/міс).

2. Сформульовано вимоги до системи, яка поєднує проєктний менеджмент, фінансовий облік, складське управління, продажі та закупівлі. Визначено 11 функціональних та 5 нефункціональних вимог, включаючи JWT-автентифікацію, гнучку рольову модель на рівні окремих функцій, вбудований чат та підтримку локалізації (UA/EN).

3. Обрано технологічний стек та спроєктовано архітектуру додатку. Серверну частину побудовано на Flask (Python) із використанням Flask-RESTful для REST API, SQLAlchemy ORM для роботи з PostgreSQL, Flask-JWT-Extended для автентифікації та Flask-Mail для email-сповіщень. Клієнтську частину реалізовано на React із TypeScript, Redux Toolkit для управління станом та React Router для маршрутизації.

4. Реалізовано серверне API з модулями управління проєктами (пайплайни, стадії, проєкти, задачі з підзадачами, time tracker), фінансами (рахунки, транзакції, бюджети, bills з overflow resolve), складом (склади, інвентар, переміщення, відвантаження), продажами (клієнти, пропозиції, замовлення) та закупівлями (постачальники, замовлення). Також реалізовано систему автентифікації з OTP-

верифікацією email, внутрішній чат з inbox-каналами та систему сповіщень (in-app та email) з автоматичними нагадуваннями про дедлайни через APScheduler.

5. Розроблено клієнтський інтерфейс із бічною навігацією по модулях, рольовим доступом (50+ функцій через таблицю Navbar), статистичними дашбордами на ApexCharts та Chart.js, підтримкою світлої/темної теми та локалізацією через i18next. Ієрархія співробітників реалізована як деревоподібна структура з можливістю переміщення.

6. Проведено тестування системи. Функціональне тестування через Postman підтвердило коректну роботу ендпоінтів реєстрації, авторизації та основних CRUD-операцій. Модульне тестування через flask_unittest охопило 34 кейси для модуля проєктів (TestPipelines) та 27 кейсів для фінансового модуля (TestAccounts), перевіривши обробку статусів 200, 400, 401, 403, 404 та каскадне видалення. Кросбраузерне тестування підтвердило коректну роботу в Chrome, Firefox, Opera та Edge.

Отриманий результат підтверджує практичну ефективність розробленого рішення. Web-застосунок Ri Dren дозволяє реалізувати комплексний підхід до управління бізнес-процесами, орієнтований на потреби малих та середніх компаній. Система може бути розгорнута на власному сервері без щомісячної абонплати за користувачів, що робить її доступною альтернативою існуючим хмарним платформам.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Подольський Д.О, Сініцин І.П. Розробка веб-додатку для корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація» 2026. Збірник тез. – К.: ДУІКТ, 2026.

2. Подольський Д.О, Сініцин І.П. Рольова модель доступу у веб-системі управління проєктами. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» 2026. Збірник тез. – К.: ДУІКТ, 2026.

ПЕРЕЛІК ПОСИЛАНЬ

1. Raji M. A., Olodo H. B., Oke T. T., Addy W. A., Ofodile O. C., Oyewole A. T. E-commerce and consumer behavior: A review of AI-powered personalization and market trends [Електронний ресурс]. – GSC Advanced Research and Reviews. — 2024. – Vol. 18(3). – P. 66–77. – Режим доступу: https://gsconlinepress.com/journals/gscarr/content/e-commerce-and-consumer-behavior — Дата звернення: 15.05.2025.

2. Slack Technologies. The State of Work: How teams communicate and collaborate [Електронний ресурс]. – Режим доступу: https://slack.com/resources/why-use-slack/state-of-work – Дата звернення: 15.05.2025.

3. Asana – офіційний сайт [Електронний ресурс]. – Режим доступу: https://asana.com/ – Дата звернення: 16.05.2025.

4. Monday.com – офіційний сайт [Електронний ресурс]. – Режим доступу: https://monday.com/ – Дата звернення: 16.05.2025.

5. G2. Monday.com Reviews [Електронний ресурс]. – 2024. – Режим доступу: https://www.g2.com/products/monday-com/reviews – Дата звернення: 16.05.2025.

6. Wrike – офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.wrike.com/ – Дата звернення: 16.05.2025.

7. Stack Overflow. Developer Survey 2024 [Електронний ресурс]. – 2024. – Режим доступу: https://survey.stackoverflow.co/2024/ – Дата звернення: 17.05.2025.

8. Flask Documentation [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/ – Дата звернення: 17.05.2025.

9. SQLAlchemy Documentation [Електронний ресурс]. – Режим доступу: https://docs.sqlalchemy.org/ – Дата звернення: 17.05.2025.

10. Werkzeug Documentation. Security Helpers [Електронний ресурс]. – Режим доступу: https://werkzeug.palletsprojects.com/en/stable/utils/#module-werkzeug.security – Дата звернення: 17.05.2025.

11. Flask-JWT-Extended Documentation [Електронний ресурс]. – Режим доступу: https://flask-jwt-extended.readthedocs.io/ – Дата звернення: 17.05.2025.

12. React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/ – Дата звернення: 18.05.2025.

13. Redux Toolkit Documentation [Електронний ресурс]. – Режим доступу: https://redux-toolkit.js.org/ – Дата звернення: 18.05.2025.

14. React Router Documentation [Електронний ресурс]. – Режим доступу: https://reactrouter.com/ – Дата звернення: 18.05.2025.

15. ApexCharts Documentation [Електронний ресурс]. – Режим доступу: https://apexcharts.com/ – Дата звернення: 18.05.2025.

16. i18next Documentation [Електронний ресурс]. – Режим доступу: https://www.i18next.com/ – Дата звернення: 18.05.2025.

17. React Icons [Електронний ресурс]. – Режим доступу: https://react-icons.github.io/react-icons/ – Дата звернення: 18.05.2025.

18. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ – Дата звернення: 19.05.2025.

19. Flask-Migrate Documentation [Електронний ресурс]. – Режим доступу: https://flask-migrate.readthedocs.io/ – Дата звернення: 19.05.2025.

20. TypeScript Documentation [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/ – Дата звернення: 19.05.2025.

21. Testlio. Cross Browser Testing for Web & Mobile Applications [Електронний ресурс]. – 2025. – Режим доступу: https://testlio.com/blog/comprehensive-guide-to-cross-browser-testing/ — Дата звернення: 20.05.2025.

22. Google Developers. Chrome DevTools Overview [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/devtools/ – Дата звернення: 20.05.2025.

23. Postman Documentation [Електронний ресурс]. – Режим доступу: https://learning.postman.com/docs/ – Дата звернення: 20.05.2025.

24. Flask-RESTful Documentation [Електронний ресурс]. – Режим доступу: https://flask-restful.readthedocs.io/ – Дата звернення: 20.05.2025.

25. Подольський Д.О, Сініцин І.П. Розробка веб-додатку для корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту.

Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація» 2026. Збірник тез. – К.: ДУІКТ, 2026.

26. Подольський Д.О, Сініцин І.П. Рольова модель доступу у веб-системі управління проєктами. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» 2026. Збірник тез. – К.: ДУІКТ, 2026.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЇ ЦИФРОВОГО РОЗВИТКУ

Цифрова система корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту на базі Python та JavaScript

Виконав студент 4 курсу
групи ТЦР-42
Подольський Дмитро Олегович
Керівник роботи
д.т.н. професор Сініцин Ігор Петрович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – покращення процесукорпоративного управління проєктами та ресурсамиза допомогою розробленого веб-додатку з інтеграцією функцій фінансового менеджменту.

Об'єкт дослідження – процес організації та контролю корпоративних проєктів, ресурсів і фінансових операцій у малих та середніх компаніях.

Предмет дослідження – програмне забезпечення для автоматизації управління проєктами, складськими запасами, продажами, закупівлями та фінансовим обліком на базі Python та JavaScript.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати існуючі корпоративні платформи (Asana, Monday.com, Wrike), виявити їхні сильні та слабкі сторони.
2. Сформулювати вимоги до системи, яка поєднує проєктний менеджмент, фінансовий облік, складське управління, продажі та закупівлі.
3. Обрати технологічний стек і спроєктувати архітектуру додатку – серверну частину на Flask (Python), клієнтську – на React із TypeScript.
4. Реалізувати серверне API з модулями управління проєктами, фінансами, складом, продажами, закупівлями, а також систему автентифікації, сповіщень і чату.
5. Розробити клієнтський інтерфейс із маршрутизацією, рольовим доступом, статистичними дашбордами та підтримкою локалізації.
6. Провести тестування системи – функціональне через Postman та модульне (unit-тести) для серверної логіки.

АНАЛІЗ АНАЛОГІВ

Функціональність / Показник	Asana	Monday.com	Wrike	Ri Dren (Власна розробка)
Управлінняпроєктамита задачами	+	+	+	+
Підзадачі та залежності	+	±	+	+
Фінансовий облік (рахунки, транзакції)	-	-	-	+
Бюджети з контролем витрат	-	-	-	+
Складський облік	-	-	-	+
CRM / база клієнтів	-	±	-	+
Модуль закупівель	-	-	-	+
Вбудований чат	-	-	-	+
Гнучка рольова модель (на рівні функцій)	-	-	±	+
Ієрархія співробітників	-	-	-	+
Локалізація (UA/EN)	-	-	-	+

ВИМОГИ ДО WEB-ЗАСТОСУНКУ

Функціональні вимоги:

1. Реєстрація та авторизація з підтвердженням email через OTP-код та JWT-токенами.
2. Створення компанії та запрошення співробітників за посиланням .
3. Гнучке налаштування прав доступу на рівні окремих функцій (50+ features).
4. Модулі : проєкти , фінанси , склад, продажі , закупівлі .
5. Внутрішній чат з підтримкою групових бесід та inbox -системою.
6. Система сповіщень (in-app + email).
7. Статистичні дашборди для кожного модуля.
8. Локалізація – українська та англійська мови.

Нефункціональні вимоги:

1. Контроль доступу відповідно до налаштованих прав.
2. Хешування паролів (sha256).
3. JWT-автентифікація (access 120 хв, refresh 1-30 днів).
4. Час відповіді API ≤ 500 мс.
5. Кросбраузерна сумісність (Chrome, Firefox, Opera, Edge).

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Мови Програмування

1. Python – Використовувся для Back-end розробки, використовув бібліотеку Flask.

2. JavaScript – Використовувся для Front-end розробки, використовув фреймворк React

Бази даних

PostgreSQL
PgAdmin – застосунок для управління даними

Мови Верстки

1. Html
2. CSS

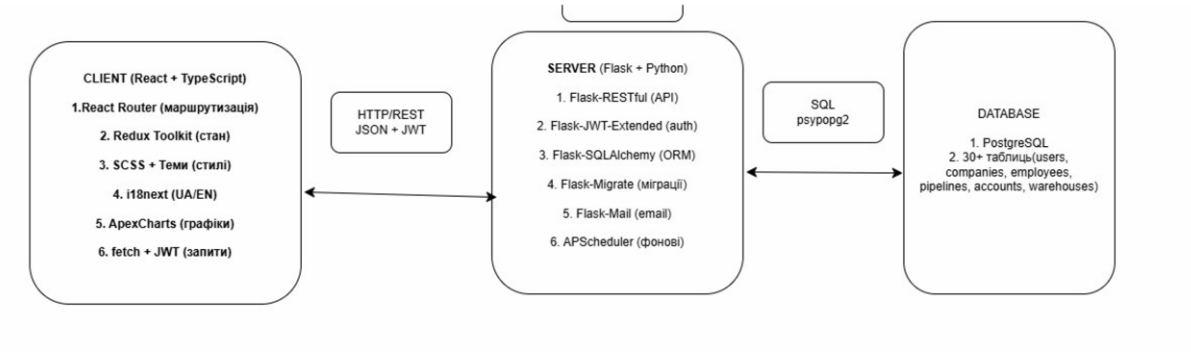
Тестування

Postman

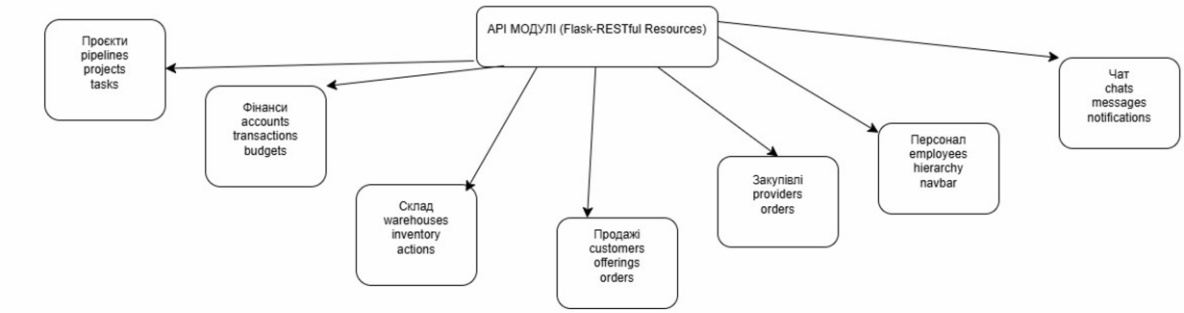
Редактори коду

Visual Studio Code

АРХІТЕКТУРА WEB-ЗАСТОСУНКУ



API Модулі

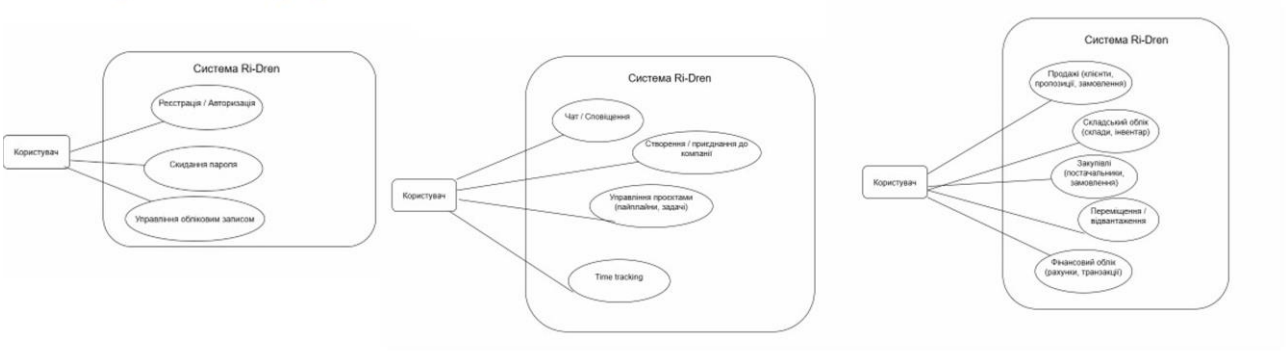


ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ (КОРИСТУВАЧ)

Управління профілем

Робочий процес

Бізнес-операції



ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ (АДМІНІСТРАТОР)

Управління персоналом та доступом

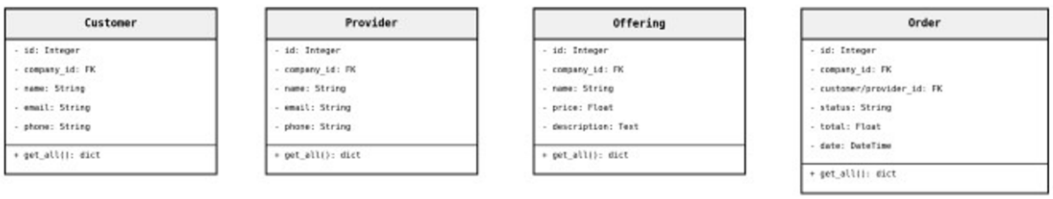
Системне адміністрування

ДІАГРАМА КЛАСІВ ОСНОВНИХ МОДУЛІВ

Модуль комунікацій

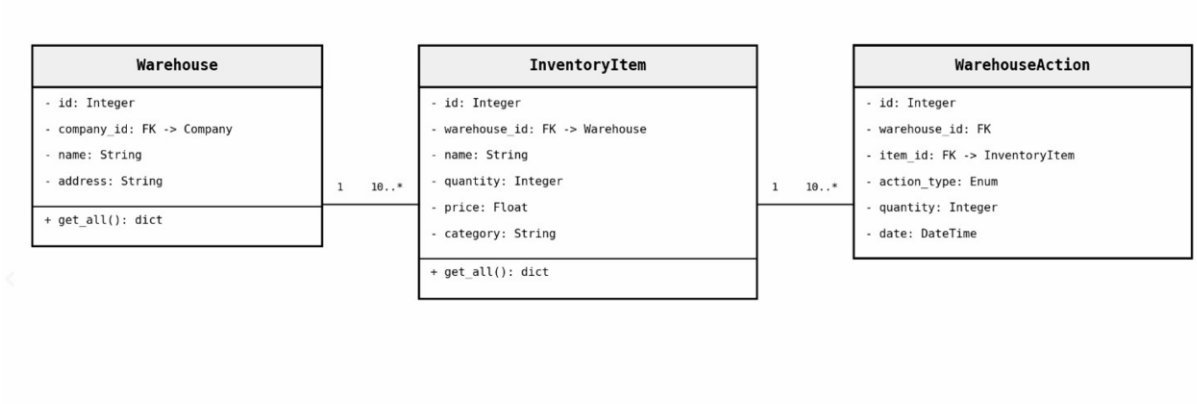
ДІАГРАМА КЛАСІВ ОСНОВНИХ МОДУЛІВ

Модуль торгівлі

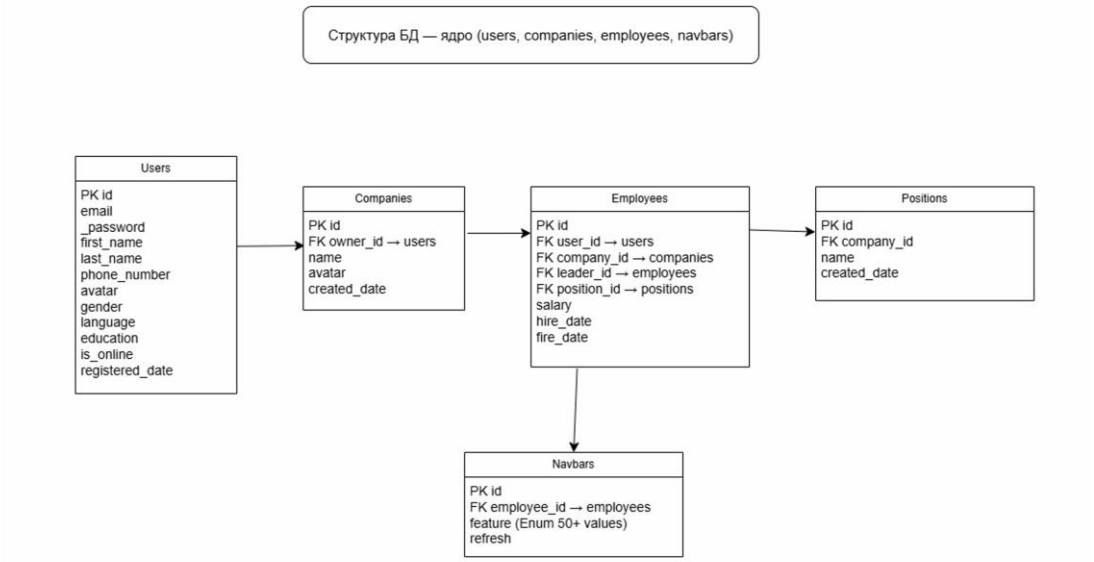


ДІАГРАМА КЛАСІВ ОСНОВНИХ МОДУЛІВ

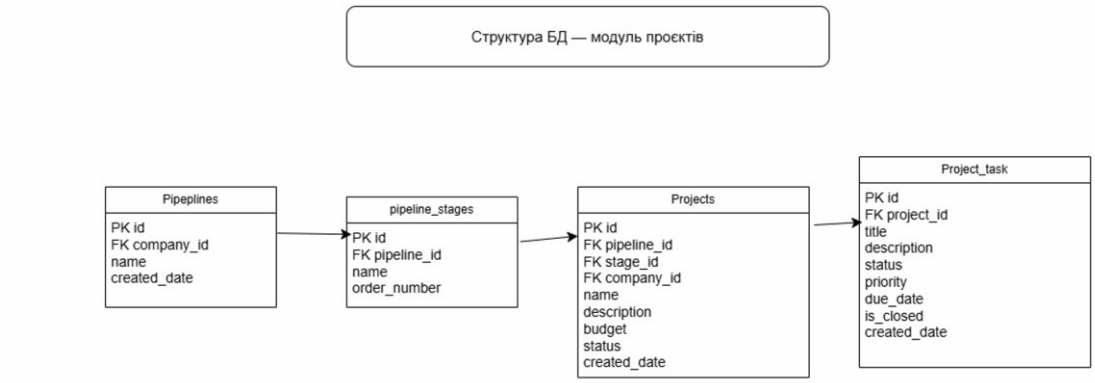
Модуль складу



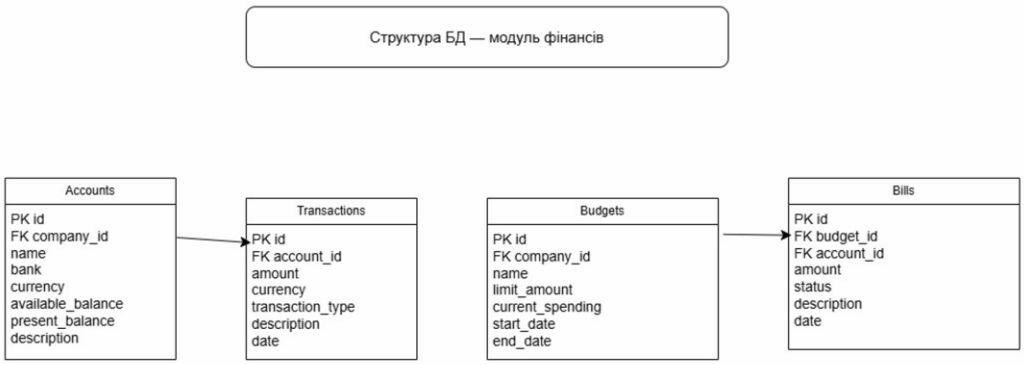
СТРУКТУРА БАЗИ ДАНИХ



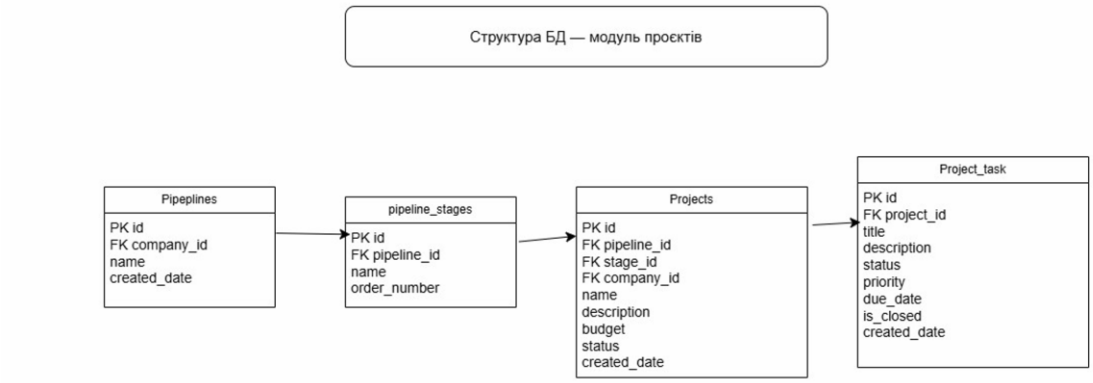
СТРУКТУРА БАЗИ ДАНИХ



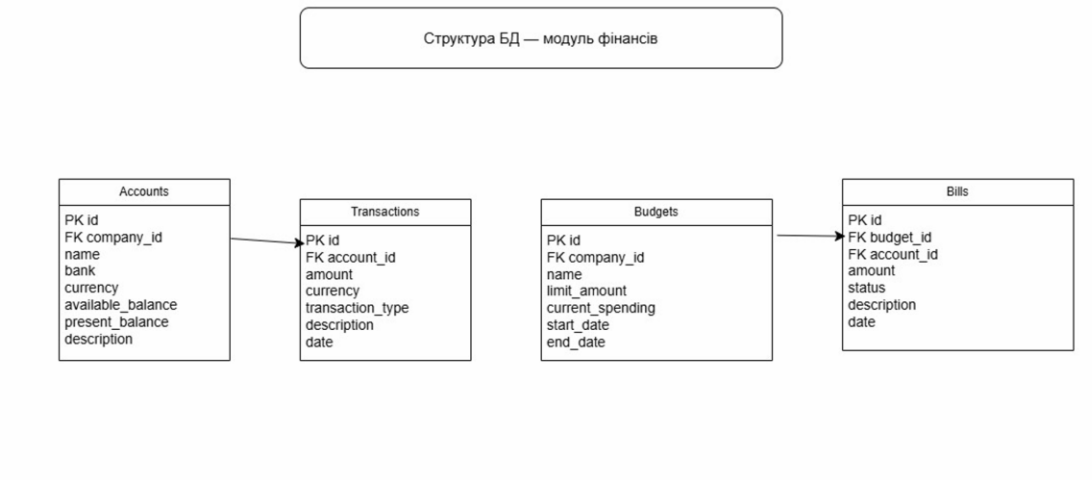
СТРУКТУРА БАЗИ ДАНИХ



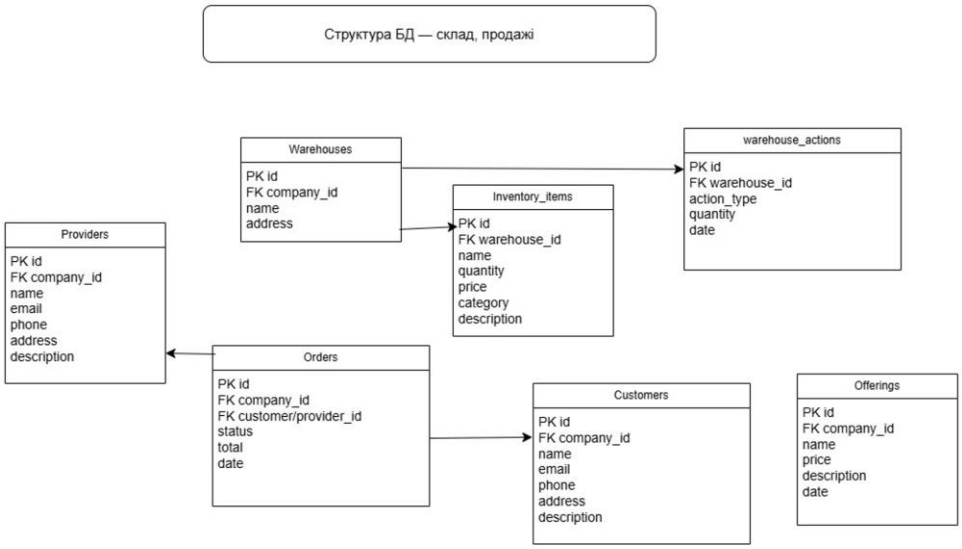
СТРУКТУРА БАЗИ ДАНИХ



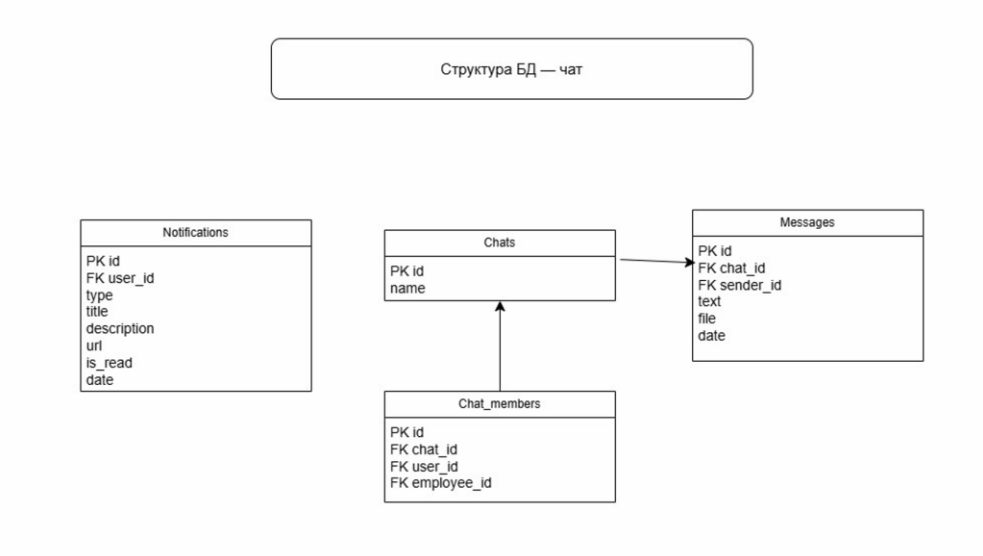
СТРУКТУРА БАЗИ ДАНИХ



СТРУКТУРА БАЗИ ДАНИХ



СТРУКТУРА БАЗИ ДАНИХ



ВИСНОВКИ

1. Проаналізовано існуючі платформи (Asana, Monday.com, Wrike) – жодна не поєднує управління проєктами з фінансами, складом та комунікацією.
2. Спроєктовано та реалізовано web-застосунок Ri Dren на Flask + React із 7 бізнес-модулями.
3. Реалізовано гнучку рольову модель (50+ функцій), вбудований чат, систему сповіщень та локалізацію (UA/EN).
4. Проведено тестування: 61 unit-тест, функціональне тестування через Postman, кросбраузерне тестування.
5. Система може бути розгорнута на власному сервері без щомісячної абонплати за користувачів.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Подольський Д.О, Сініцин І.П. Розробка веб-додатку для корпоративного управління проєктами та ресурсами з інтеграцією фінансового менеджменту. // Матеріали Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація» 2026. Збірник тез. – К.: ДУІКТ, 2026.

2. Подольський Д.О, Сініцин І.П. Рольова модель доступу у веб-системі управління проєктами. // Матеріали VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» 2026. Збірник тез. – К.: ДУІКТ, 2026.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
class User(db.Model):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True)

    _password = Column(String, nullable=False)
    first_name = Column(String(100), nullable=False)
    last_name = Column(String(100))
    phone_number = Column(String(100))
    email = Column(String(100), nullable=False, unique=True)
    avatar = Column(LargeBinary)
    age = Column(Integer, nullable=True)
    gender = Column(Enum("male", "female", "other", name="UserGender"),
nullable=True)
    language = Column(Enum("ua", "en", name="UserLanguage"), nullable=True,
default="en")
    education = Column(Enum("diploma", "bachelor", "master", "phd",
name="UserEducation"), nullable=True)
    registered_date: datetime = Column(DateTime, default=datetime.utcnow)
    confirmed_on = Column(DateTime)
    is_online = Column(Boolean, nullable=False, default=False)
    is_admin = Column(Boolean, nullable=False, default=False)
    last_online_date = Column(DateTime, default=datetime.utcnow)
    created_date = Column(DateTime, default=datetime.utcnow)
    companies = relationship("Company")
    # Temporary User login restriction
    login_allowed = Column(Boolean, nullable=False, default=True)

    @hybrid_property
    def _default_order_by(self):return User.created_date.desc()

    @property
    def password(self):
        print("Password can not be read.")

    @password.setter
    def password(self, password):
```

```
        self._password = generate_password_hash(password, method="sha256")

    def verify_password(self, password):
        return check_password_hash(self._password, password)

    def get_full_name(self):
        return f"{self.first_name} {self.last_name}"

    def set_online(self):
        self.is_online = True
        self.update()

        def make_user_offline():
            with db.app.app_context():
                dbuser = User.query.get(self.id)
                dbuser.is_online = False
                dbuser.update()

        t = Timer(

ApplicationConfig.ENDPOINT_MAKE_USER_OFFLINE_AFTER_N_SECONDS,
            make_user_offline,
        )
        t.start()

    @property
    def name(self):
        if self.last_name:
            return f"{self.first_name} {self.last_name}"
        return self.first_name

    def insert(self):
        self.email = self.email.lower()
        db.session.add(self)
        db.session.commit()

    def update(self):
        db.session.commit()
```

```
def delete(self):
    db.session.delete(self)
    db.session.commit()

def get_all(self):
    companies = [
        Company.query.get(i).get_all()
        for i in [
            i.company_id for i in Employee.query.filter_by(user_id=self.id).all()
        ]
    ]

    return {
        "user_id": self.id,
        "email": self.email,
        "phone_number": self.phone_number,
        "first_name": self.first_name,
        "last_name": self.last_name,
        "age": self.age,
        "gender": self.gender,
        "education": self.education,
        "registered_date": self.registered_date.timestamp(),
        "confirmed_on": self.confirmed_on.timestamp(),
        "is_online": self.is_online,
        "last_online_date": self.last_online_date.timestamp(),
        "companies": companies,
        "avatar": decode_blob(self.avatar),

    }

def __repr__(self):
    return "<User %r>" % self.id

@classmethod
def get_users(cls, company_id=None,
              min_age=None, max_age=None,
              gender=None,
              education=None,
              registered_date_from=None, registered_date_to=None,
```

```
            confirmed_date_from=None, confirmed_date_to=None,
            needle=None, per_page=10, page=1):
    # Create a base query
    query = db.session.query(User).join(Company, Company.id == User.company_id)

    if company_id:
        query = query.filter(User.company_id == company_id)

    if any([min_age, max_age]):
        query = query.filter(and_(User.age >= min_age, User.age <= max_age))

    if any([registered_date_from, registered_date_to]):
        query = query.filter(
            and_(User.registered_date >= registered_date_from, User.registered_date <=
registered_date_to))

    if any([confirmed_date_from, confirmed_date_to]):
        query = query.filter(and_(User.confirmed_on >= confirmed_date_from,
User.confirmed_on <= confirmed_date_to))

    if gender:
        query = query.filter(User.gender == gender)

    if education:
        query = query.filter(User.education == education)

    # Apply search functionality using OR clauses
    if needle:
        looking_for = '%{0}%'.format(needle)

        users = query.filter(
            (Company.name.ilike(looking_for)) |
            (User.first_name.ilike(looking_for)) |
            (User.last_name.ilike(looking_for)) |
            (User.phone_number.ilike(looking_for)))

    # If no search query (needle) provided, return all users listed
    else:
        users = query
```

```
        # Pagination
        users = users.paginate(
            per_page=per_page, page=page, error_out=False)

        # Execute the query and retrieve the items
        user_list = users.items

        # Return a list of user
        return [user.get_all() for user in user_list]

class UsersOnline(db.Model):
    __tablename__ = "users_online"
    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey('users.id'))
    timestamp = Column(DateTime, default=datetime.utcnow)
    user = relationship("User")
    created_date = Column(DateTime, default=datetime.utcnow)

    @hybrid_property
    def _default_order_by(self):return UsersOnline.created_date.desc()
    def insert(self):
        db.session.add(self)
        db.session.commit()

    def update(self):
        db.session.commit()

    def delete(self):
        db.session.delete(self)
        db.session.commit()

    @classmethod
    def online_users(cls):
        return db.session.query(func.count(distinct(cls.user_id))).scalar()

class Navbar(db.Model):
    __tablename__ = "navbars"
    id = Column(Integer, primary_key=True)
```

```
    employee_id = Column(Integer, ForeignKey("employees.id",
ondelete="CASCADE"), nullable=True)
    created_date = Column(DateTime, default=datetime.utcnow)
    feature = Column(
        Enum("hire_employees", "move_employees", "fire_employees",
"edit_employees", "employee_statistics",
             "get_productivity",
             "add_positions", "edit_positions", "delete_positions",
             "add_labels", "delete_labels",

             "add_pipelines", "edit_pipelines", "delete_pipelines", "project_statistics",
             "add_projects", "edit_projects", "delete_projects",
             "add_tasks", "edit_tasks", "delete_tasks",

             "get_customers", "add_customers", "edit_customers", "delete_customers",
"sales_statistics",
             "get_offerings", "add_offerings", "edit_offerings", "delete_offerings",

             "get_providers", "add_providers", "edit_providers", "delete_providers",
"purchase_statistics",
             "get_orders", "add_orders", "edit_orders", "delete_orders",

             "warehouse_statistics",
             "get_warehouses", "add_warehouses", "edit_warehouses",
"delete_warehouses",
             "get_warehouse_actions", "add_warehouse_actions",
"edit_warehouse_actions", "delete_warehouse_actions",
             "get_inventory_items", "add_inventory_items", "edit_inventory_items",
"delete_inventory_items",

             "finance_statistics", "financial_reports",
             "get_accounts", "add_accounts", "edit_accounts", "delete_accounts",
             "get_transactions", "add_transactions", "edit_transactions",
"delete_transactions",
             "get_budgets", "add_budgets", "edit_budgets", "delete_budgets",
             "get_bills", "add_bills", "edit_bills", "delete_bills",
             "train_model", "company_settings",
             name="Feature"),
        nullable=True)
```

```
    refresh = Column(Boolean, nullable=True)

    @hybrid_property
    def _default_order_by(self):return Navbar.created_date.desc()
    def insert(self):
        db.session.add(self)
        db.session.commit()

    def update(self):
        db.session.commit()

    def delete(self):
        db.session.delete(self)
        db.session.commit()




class PipelinesGetFiltersResource(Resource):
    @jwt_required()
    def get(self, company_id):
        create_access = bool(is_feature_allowed(company_id, "add_pipelines"))

        # Get all pipelines
        pipelines = Pipeline.query.all()

        # Initialize min and max projects with the total projects of the first pipeline
        min_projects = float('inf')
        max_projects = 0

        # Iterate through the pipelines, stages, and projects to find the minimum and
maximum projects
        for pipeline in pipelines:
            total_projects = 0
            for stage in pipeline.stages:
                for project in stage.projects:
                    if not project.is_closed:
                        total_projects += 1
```

```
            if total_projects < min_projects:
                min_projects = total_projects
            if total_projects > max_projects:
                max_projects = total_projects

        filters = {
            "filters": [
                {
                    "type": "RANGE",
                    "title": "Number of Projects",
                    "data": {"min": min_projects, "max": max_projects},
                    "key": "no_of_projects"
                },
                {
                    "type": "CALENDAR",
                    "title": "Earliest deadline",
                    "data": None,
                    "key": "date"
                }
            ],
            "create_access": create_access
        }

        return api_response(filters, "Got Filters")

class PipelinesGetAllTasksResource(Resource):
    @jwt_required()
    def get(self, pipeline_id):
        pipeline: Pipeline = Pipeline.query.filter_by(id=pipeline_id).first()
        if not pipeline:
            return api_abort(404, f"Pipeline with id: {pipeline_id} doesn't exist")

        # Check if user has accounts access and is the current company user
        if not is_feature_allowed(pipeline.company_id, ''):
            raise Forbidden

        status = request.args.get("status")

        page = int(request.args.get("page", 1))
```

```
    per_page = int(request.args.get("per_page", 10))
    show_closed = request.args.get("show_closed", 'false') == "true"

    statuses = {"1": "open", "2": "in_progress", "3": "completed", "4": "accepted", "5":
"rejected"}

    stages = PipelineStage.get_stages(pipeline_id)

    stage_ids = [stage["stage_id"] for stage in stages]

    projects = db.session.query(
        Project.id
    ).filter(
        Project.stage_id.in_(stage_ids)
    ).all()

    project_ids = [project[0] for project in projects]

    query = db.session.query(ProjectTask).filter(
        ProjectTask.project_id.in_(project_ids),
        ProjectTask.is_closed == show_closed
    ).order_by(ProjectTask.created_date.desc())

    if status and is_valid_integer(status) and int(status) > -1:
        query = query.filter(ProjectTask.status == statuses[status])

    needle = request.args.get("needle")  # Get the needle parameter
    # Filter tasks based on needle (search term in task titles)
    if needle:
        search_query = f'%{needle}%'
        query = query.filter(or_(
        ProjectTask.title.ilike(search_query),
        ProjectTask.description.ilike(search_query),
    ))

    tasks = query.paginate(page=page, per_page=per_page, error_out=False).items

    return api_response([task.get_all() for task in tasks], "Pipeline created")
```

```
    @jwt_required()
    def patch(self, pipeline_id):
        show_closed = request.args.get("show_closed", 'false') == "true"

        pipeline: Pipeline = Pipeline.query.filter_by(id=pipeline_id).first()
        if not pipeline:
            return api_abort(404, f"Pipeline with id: {pipeline_id} doesn't exist")

        # Check if user has accounts access and is the current company user
        if not is_feature_allowed(pipeline.company_id, '):
            raise Forbidden

        all_statuses = ["open", "in_progress", "completed", "accepted", "rejected"]
        statuses = {"open": 1, "in_progress": 2, "completed": 3, "accepted": 4, "rejected":
5}
        colors = {"open": "#0000ff", "in_progress": "#ffa500", "completed": "#ffff00",
"accepted": "#008000", "rejected": "#ff0000"}

        stages = PipelineStage.get_stages(pipeline_id)

        stage_ids = [stage["stage_id"] for stage in stages]

        projects = db.session.query(
            Project.id
        ).filter(
            Project.stage_id.in_(stage_ids)
        ).all()

        project_ids = [project[0] for project in projects]

        stage_info = db.session.query(
            ProjectTask.status,
            func.count(ProjectTask.id).label("amount_of_objects")
        ).filter(
            ProjectTask.project_id.in_(project_ids),
            ProjectTask.status.in_(all_statuses),
            ProjectTask.is_closed == show_closed
        ).group_by(ProjectTask.status).all()
```

```
    response = [
        {
            "title": status,
            "id": statuses[status],
            "color": colors[status],
            "quantity": 0,
            "create_access": bool(is_feature_allowed(pipeline.company_id, "add_tasks"))
        }
        for status in all_statuses
    ]

    for item in response:
        for status, count in stage_info:
            if item["title"] == status:
                item["quantity"] = count
                break

    return api_response(response, 'got tasks skeletons')
```

```
from flask import request
from flask_jwt_extended import jwt_required, current_user
from flask_restful import Resource
import pandas as pd
import os
import base64
from datetime import datetime
from werkzeug.exceptions import Forbidden
from sqlalchemy import func

from server.apps.apis.general.filters.accounts import get_account_filters,
get_filtered_accounts
from server.apps.apis.general.filters.transactions import get_transaction_filters,
get_filtered_transactions

from server.apps.apis.helpers import api_response, api_abort, create_item, update_item,
delete_item, is_feature_allowed, can_access, format_dates
```

```
from server.db.models import Account, Transaction, Bill, Navbar, db


class CompanyAccountFilters(Resource):
    @jwt_required()
    def get(self, company_id):
        create_access = can_access(company_id, "add_accounts")

        # Query to get the minimum and maximum sum of amounts
        min_max_query = db.session.query(
            func.min(Account.balance_uah).label('min_amount_uah'),
            func.max(Account.balance_uah).label('max_amount_uah'),
            func.min(Account.balance_usd).label('min_amount_usd'),
            func.max(Account.balance_usd).label('max_amount_usd'),
            func.min(Account.balance_eur).label('min_amount_eur'),
            func.max(Account.balance_eur).label('max_amount_eur')
        ).filter(
            Account.company_id == company_id
        ).subquery()

        # Fetch the minimum and maximum amounts
        min_max_amounts = db.session.query(
            min_max_query.c.min_amount_uah,
            min_max_query.c.max_amount_uah,
            min_max_query.c.min_amount_usd,
            min_max_query.c.max_amount_usd,
            min_max_query.c.min_amount_eur,
            min_max_query.c.max_amount_eur
        ).first()

        return api_response(get_account_filters(min_max_amounts.min_amount_uah,
min_max_amounts.max_amount_uah, min_max_amounts.min_amount_usd,
min_max_amounts.max_amount_usd, min_max_amounts.min_amount_eur,
min_max_amounts.max_amount_eur, create_access), "Got Filters")

class CompanyAccountOverview(Resource):
    @jwt_required()
    def get(self, company_id):
        needle = request.args.get("needle")
```

```
        # Check if the feature 'accounts' is allowed for the given company
        if not is_feature_allowed(company_id, 'get_accounts'):
            raise Forbidden

        query =  Account.query.filter_by(company_id=company_id)

        min_balance_uah = request.args.get('min_balance_uah', None)
        max_balance_uah = request.args.get('max_balance_uah', None)
        min_balance_usd = request.args.get('min_balance_usd', None)
        max_balance_usd = request.args.get('max_balance_usd', None)
        min_balance_eur = request.args.get('min_balance_eur', None)
        max_balance_eur = request.args.get('max_balance_eur', None)
        min_last_action = request.args.get('min_last_action', None)
        max_last_action = request.args.get('max_last_action', None)

        accounts = get_filtered_accounts(query, needle, min_balance_uah,
max_balance_uah, min_balance_usd, max_balance_usd, min_balance_eur,
max_balance_eur, min_last_action, max_last_action).all()

        total = len(accounts)
        balance = [{"currency": "UAH", "value": 0}, {"currency": "USD", "value": 0},
{"currency": "EUR", "value": 0}]

        for account in accounts:
            balance[0]["value"] += account.balance_uah
            balance[1]["value"] += account.balance_usd
            balance[2]["value"] += account.balance_eur

        data = {"total_accounts": total, "balance": balance}
        return api_response(data, "Got Accounts Overview")



class CompanyAccounts(Resource):
    # get accounts list filtered by company and paged
    @jwt_required()
    def get(self, company_id):
        page = int(request.args.get("page", 1))
        per_page = int(request.args.get("per_page", 10))
```

```
    needle = request.args.get("needle")

    # Check if the feature 'accounts' is allowed for the given company
    if not is_feature_allowed(company_id, 'get_accounts'):
        raise Forbidden

    query =  Account.query.filter_by(company_id=company_id)

    min_balance_uah = request.args.get('min_balance_uah', None)
    max_balance_uah = request.args.get('max_balance_uah', None)
    min_balance_usd = request.args.get('min_balance_usd', None)
    max_balance_usd = request.args.get('max_balance_usd', None)
    min_balance_eur = request.args.get('min_balance_eur', None)
    max_balance_eur = request.args.get('max_balance_eur', None)
    min_last_action = request.args.get('min_last_action', None)
    max_last_action = request.args.get('max_last_action', None)

    query = get_filtered_accounts(query, needle, min_balance_uah, max_balance_uah,
min_balance_usd, max_balance_usd, min_balance_eur, max_balance_eur,
min_last_action, max_last_action)

    export = request.args.get("export", False)
    export_type = request.args.get("export_type", "csv")

    if export == "":
        export = False

    if export_type == "":
        return api_abort(400, "Need Export type")

    accounts = query.paginate(page=page, per_page=per_page, error_out=False).items

    if export:
        accounts = query.all()
        for account in accounts:
            del account["id"]

        df = pd.DataFrame.from_records(accounts)
```

```
            if export_type == "csv":
                df.to_csv("accounts.csv")
                resp = {"file": base64.b64encode(open("accounts.csv",
"rb").read()).decode('UTF-8')}
                os.remove(os.getcwd() + "/accounts.csv")
            elif export_type == "xlsx":
                df.to_excel("accounts.xlsx", sheet_name='Accounts')
                resp = {"file": base64.b64encode(open("accounts.xlsx",
"rb").read()).decode('UTF-8')}
                os.remove(os.getcwd() + "/accounts.xlsx")
            else:
                resp = {"msg": "Not Implemented Yet"}
        else:
            resp = [account.get_all() for account in accounts]

        return api_response(resp, 'Got Account(s)')

    # add account
    @jwt_required()
    def post(self, company_id):
        # It parses the JSON data sent in the request body and returns it as a Python
dictionary.
        data = request.get_json()

        # Checking if the feature 'add_accounts' is allowed for the given company.
        current_employee = is_feature_allowed(company_id, 'add_accounts')
        if not current_employee:
            raise Forbidden

        accounts = []

        if "data" not in data:
            return api_abort(400, "Need data: [] in the body")

        for account in data["data"]:
            account["company_id"] = company_id
            n_account = create_item(Account, account)
            accounts.append(n_account.get_all())
```

```
# Send the data for the new account to the create_item function and return it
return api_response(accounts, 'Account(s) added!')
```