ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інтелектуальна система управління робочим часом працівників з обробкою запитів природною мовою»

на здобуття освітнього ступеня магістра
зі спеціальності F3 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

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

_____ Ярослав ЖУК
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Ярослав ЖУК
група КНДМ-62

Керівник: Петро КРАВЧУК
науковий ступінь, вчене звання доктор філософії

Рецензент:
науковий ступінь, вчене звання _____
(Ім'я, ПРІЗВИЩЕ)

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність F3 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Жуку Ярославу Руслановичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Інтелектуальна система управління робочим часом працівників з обробкою запитів природною мовою

керівник кваліфікаційної роботи Петро КРАВЧУК, доктор філософії
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна та навчально-методична література з управління, автоматизації бізнес-процесів, обробки природної мови та великих мовних моделей; інструменти для розробки клієнтської частини веб-додатку (React, HTML, CSS, JavaScript, Tailwind CSS); інструменти для розробки серверної частини (Python, FastAPI, SQLAlchemy, PostgreSQL); засоби контейнеризації та розгортання (Docker); сервіс мовної моделі для реалізації модуля обробки природномовних запитів.

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

Дослідження процесів управління робочим часом та відсутностями працівників у сучасних організаціях; аналіз існуючих HRM-систем та програмних рішень, визначення їх функціональних можливостей, переваг і недоліків.
Дослідження сучасних підходів до обробки природної мови у корпоративних інформаційних системах.
Проектування та реалізація інтелектуальної веб-системи управління відсутностями працівників із підтримкою природномовного інтерфейсу та автоматизованого погодження.

5. Перелік графічного матеріалу: *презентація*
   1. Порівняльний аналіз існуючих систем управління відсутностями.
   2. Архітектура системи TimelyMind та схема її основних модулів.
   3. Схема взаємодії модуля обробки природної мови з API.
   4. Екранні форми веб-додатку для управління.
   5. Фрагменти програмного коду серверної та клієнтської частини.

6. Дата видачі завдання«15» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури з HRM та NLP	15.09-27.09.25	виконано
2	Огляд існуючих систем управління відсутностями	28.09-11.10.25	виконано
3	Визначення функціоналу та характеристик HRM-системи	12.10-25.10.25	виконано
4	Вибір інструментів для розробки клієнтської частини	26.10-04.11.25	виконано
5	Вибір інструментів для розробки серверної частини	05.11-12.11.25	виконано
6	Розробка інтелектуальної системи управління відсутностями	12.11-02.12.25	виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-08.12.25	виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.25	виконано

Здобувач вищої освіти _____ Ярослав ЖУК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Петро КРАВЧУК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 75 стор., 19 рис., 34 джерела.

Наукове завдання – дослідження процесів управління робочим часом та відсутностями працівників у сучасних організаціях, аналіз і класифікація програмних засобів управління людськими ресурсами, огляд існуючих програмних рішень, а також вибір інструментів для розробки веб-системи; розробка із підтримкою обробки запитів природною мовою та зручним інтерфейсом.

Мета роботи – створення інтелектуальної веб-системи управління робочим часом та відсутностями працівників, яка забезпечує автоматизацію процесів подання, погодження та моніторингу відсутностей.

Об'єкт дослідження – процеси управління робочим часом та відсутностями працівників в організаціях.

Предмет дослідження – інтелектуальна веб-система управління відсутностями працівників з використанням технологій обробки природної мови.

Короткий зміст роботи: У роботі досліджено призначення та основні підходи, проаналізовано сучасні програмні засоби управління людськими ресурсами, визначено їх функціональні можливості, переваги та недоліки.

На основі проведеного аналізу спроектовано та реалізовано інтелектуальну веб-систему управління відсутностями працівників із зручним та інтуїтивно зрозумілим користувацьким інтерфейсом. Система забезпечує можливість подання та погодження запитів на відсутність, перегляду календаря відсутностей, а також взаємодії з користувачами за допомогою запитів природною мовою.

Ключові слова: інтелектуальна система, управління відсутностями працівників, веб-додаток, обробка природної мови, автоматизація бізнес-процесів.

ABSTRACT

Text part of the master's qualification work: 75 pages, 19 pictures, 34 sources.

The scientific task consists in studying the processes of working time and employee absence management in modern organizations; analyzing and classifying human resource management software tools; reviewing existing software solutions; and selecting tools for the development of a web system, as well as developing it with support for natural language query processing and a user-friendly interface.

The purpose of the work is to create an intelligent web-based system for managing working time and employee absences that ensures automation of the processes of submission, approval, and monitoring of absences.

The object of the research is the processes of working time and employee absence management in organizations.

The subject of the research is an intelligent web-based system for managing employee absences using natural language processing technologies.

Summary of the work: The work investigates the purpose and main approaches, analyzes modern human resource management software tools, and identifies their functional capabilities, advantages, and disadvantages.

Based on the conducted analysis, an intelligent web-based system for managing employee absences with a convenient and intuitive user interface was designed and implemented. The system provides the ability to submit and approve absence requests, view an absence calendar, monitor employee statuses, and interact with users through natural language queries.

Keywords: intelligent system, employee absence management, web application, natural language processing, business process automation.

ЗМІСТ

ВСТУП

Ефективне управління людськими ресурсами є фундаментом успішного функціонування будь-якої сучасної організації. Це пояснюється насамперед тим, що в умовах динамічного ринку та поширення гібридних форматів роботи, контроль за робочим часом трансформується зі звичайної бюрократичної процедури на складний логістичний виклик. У процесі адміністрування відсутностей менеджерам та HR-спеціалістам доводиться опрацьовувати значні масиви інформації, узгоджувати графіки різних команд та враховувати індивідуальні баланси відпусток чи лікарняних. Саме тому, при використанні застарілих підходів, актуальна інформація про доступність працівників далеко не завжди лежить на поверхні, що призводить до помилок у плануванні проєктів.

Досить часто процедура оформлення навіть короткострокової відпустки може перетворюватися на тривалий процес, що вимагає виконання кількох логічних кроків: від перевірки залишку днів до отримання схвалення через ланцюжок керівників. Це надає системам автоматизації HRM (Human Resource Management) особливого значення, однак більшість існуючих рішень страждають від перевантажених інтерфейсів та надмірної складності для кінцевого користувача. Серед сучасних тенденцій у розробці програмного забезпечення особливе місце займають технології штучного інтелекту, зокрема методи обробки природної мови (NLP).

Використання великих мовних моделей (LLM) дозволяє докорінно змінити парадигму взаємодії користувача з системою. На відміну від класичних інтерфейсів, що вимагають навігації через безліч меню, природномовний інтерфейс дозволяє сформулювати запит у довільній формі, покладаючись на «інтелект» системи у розпізнаванні намірів. Такий підхід не лише спрощує процес подачі заявок, а й допомагає працівникам та керівникам економити час, зводячи складні адміністративні процедури до простого діалогу. Розробка інтелектуальної системи, яка поєднує надійність класичних алгоритмів обліку з гнучкістю генеративного штучного інтелекту, є актуальною науковою задачею.

Об'єкт дослідження – процес створення інтелектуальної веб-системи управління робочим часом та відсутностями працівників у організаціях.

Предмет дослідження – інтелектуальна веб-система управління відсутностями працівників із підтримкою обробки запитів природною мовою.

Метою дослідження є створення інтелектуальної веб-системи, що забезпечує автоматизацію процесів подання, погодження та моніторингу відсутностей працівників і підвищує ефективність управління робочим часом в організаціях. Передбачається, що у системі користувачі зможуть подавати запити на відсутність, переглядати статуси колег, отримувати інформацію про доступний баланс днів, а також взаємодіяти з системою за допомогою природних запитів. Завдяки зручному інтерфейсу та використанню інтелектуальних механізмів така система може слугувати ефективним інструментом для оптимізації HR-процесів і зменшення адміністративного навантаження.

Для досягнення мети роботи були поставлені наступні завдання:

- дослідити призначення та класифікацію програмних засобів управління людськими ресурсами, проаналізувати наявні системи управління відсутностями працівників, визначити їх функціональні можливості, переваги та недоліки;
- підібрати інструменти для розробки клієнтської та серверної частин інтелектуальної веб-системи;
- спроектувати та реалізувати систему управління відсутностями працівників із зручним інтуїтивно зрозумілим інтерфейсом та підтримкою обробки запитів природною мовою.

Практичне значення розробленої системи полягає в можливості її використання в організаціях різного масштабу для автоматизації процесів управління робочим часом і відсутностями працівників, підвищення прозорості HR-процесів, скорочення часу на погодження запитів та покращення взаємодії між працівниками, керівниками та відділом кадрів.

1 ОГЛЯД ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Управління робочим часом та відсутностями працівників

В умовах глобалізації економіки та стрімкого розвитку інформаційних технологій, людський капітал залишається найбільш цінним та водночас найскладнішим для адміністрування активом будь-якої організації. Ефективність функціонування сучасного підприємства, особливо у сфері інтелектуальних послуг та IT-секторі, напряму залежить від раціонального використання робочого часу та здатності менеджменту оперативно реагувати на зміни у доступності трудових ресурсів. Управління робочим часом (Workforce Management, WFM) трансформувалося з рутинної облікової функції у стратегічний інструмент, що впливає на фінансові показники компанії, терміни виконання проектів та загальний рівень задоволеності персоналу.

Невід'ємною складовою WFM є управління відсутностями (Absence Management) - системний процес планування, обліку, контролю та аналізу періодів, коли працівник не виконує свої посадові обов'язки. До таких періодів належать щорічні відпустки, лікарняні (тимчасова непрацездатність), відгули, навчальні відпустки та інші види соціальних гарантій. Незважаючи на гадану простоту, цей процес містить значну кількість прихованих ризиків та адміністративних бар'єрів, які стають особливо відчутними зі зростанням штату організації.

Актуальність проблеми ефективного управління відсутностями суттєво зросла з переходом більшості технологічних компаній на гібридний або повністю віддалений режим роботи. В умовах офісної роботи керівник міг візуально фіксувати присутність підлеглих. Натомість, у розподілених командах, де комунікація відбувається асинхронно, відсутність чіткої інформації про статус працівника (наприклад, "чи він сьогодні працює, чи взяв day off") призводить до комунікаційних розривів. Колеги можуть очікувати відповіді на термінові запити, не знаючи, що адресат перебуває у відпустці, що в результаті гальмує робочі процеси та знижує загальну продуктивність команди.

З економічної точки зору, некоректне управління відсутностями призводить до двох діаметрально протилежних, але однаково шкідливих явищ: абсентеїзму та презентеїзму. Абсентеїзм характеризується систематичною відсутністю працівника на робочому місці без поважних причин або зловживанням лікарняними, що тягне за собою прямі фінансові втрати на оплату невикористаного часу та непрямі витрати на пошук заміни. Презентеїзм, навпаки, описує ситуацію, коли працівник виходить на роботу хворим або працює під час відпустки через страх накопичення завдань або тиск з боку керівництва [1]. Останнє явище є особливо небезпечним для IT-галузі, оскільки призводить до професійного вигорання, зниження когнітивних здібностей та, як наслідок, погіршення якості програмного коду та архітектурних рішень.

Традиційні підходи до адміністрування відпусток, що базуються на паперовому документообігу, електронних таблицях (наприклад, Excel або Google Sheets) або розрізнених каналах комунікації (електронна пошта, месенджери), демонструють свою неспроможність у сучасних динамічних умовах. По-перше, такі методи не забезпечують цілісності даних. Інформація про залишок днів відпустки може зберігатися в одній системі, запит подаватися в іншій месенджер, а погодження фіксуватися у третій Це створює ґрунт для помилок, коли працівнику погоджують відпустку, що перевищує його ліміт, або ж забувають нарахувати бонусні дні. По-друге, ручний процес погодження є надзвичайно часозатратним. Типовий workflow передбачає ланцюжок дій: працівник перевіряє баланс, пише заяву, погоджує її з безпосереднім керівником, потім з менеджером проекту або департаменту, і нарешті передає в HR-відділ для фіксації. На кожному етапі виникають затримки, пов’язані з людським фактором.

Окремою проблемою є планування ресурсів. Менеджери проектів, не маючи зручного календарного інструменту з візуалізацією відсутностей команди, ризикують запланувати критичні етапи розробки на періоди масових відпусток ключових спеціалістів. Відсутність автоматизованої системи прогнозування та моніторингу робить процес планування реактивним, а не проактивним. Замість

завчасного перерозподілу навантаження, менеджери змушені "гасити пожежі", коли факт відсутності працівника стає очевидним вже у день початку роботи [2].

Сучасна парадигма "Employee Experience" також висуває нові вимоги до внутрішніх корпоративних систем. Користувачі, які у повсякденному житті звикли до інтуїтивних інтерфейсів та миттєвих реакцій сервісів, очікують аналогічного рівня зручності від робочих інструментів. Бюрократизовані, складні інтерфейси застарілих ERP-систем викликають спротив та знижують лояльність персоналу. Працівник не повинен витрачати час на вивчення інструкцій для того, щоб подати заяву на один день відпустки. Саме тут виникає потреба у застосуванні новітніх підходів до проектування інтерфейсів, зокрема - розмовних інтерфейсів та обробки природної мови (Natural Language Processing, NLP).

Можливість взаємодії з системою обліку часу за допомогою звичайної мови ("Я хочу взяти відпустку з наступного понеділка на три дні") знімає когнітивне навантаження з користувача. Замість заповнення формалізованих полів у веб-формах, працівник делегує задачу інтерпретації свого наміру інтелектуальному агенту. Це не лише пришвидшує процедуру подачі заявки, але й робить систему інклюзивною та доступною. Однак, реалізація такого підходу вимагає складних алгоритмічних рішень для розпізнавання сутностей (дат, типів відсутностей), контексту та валідації бізнес-правил.

Крім того, важливим аспектом є прозорість та доступність інформації про баланси. Часто конфліктні ситуації виникають через те, що працівник та роботодавець по-різному оцінюють кількість доступних днів відпочинку. Автоматизована система, що веде журнал усіх транзакцій і надає доступ до цієї історії в режимі реального часу, виступає об’єктивним арбітром та знижує рівень напруги в колективі.

Варто також зазначити, що законодавче регулювання трудових відносин у більшості країн, включаючи Україну, вимагає суворого обліку робочого часу. Помилки в табелюванні можуть призвести до юридичних наслідків та штрафних санкцій для підприємства. Тому, будь-яка інформаційна система, що претендує на

вирішення задач у цій доменній області, повинна гарантувати точність, надійність зберігання даних та можливість аудиту всіх дій.

Підсумовуючи аналіз предметної області, можна стверджувати, що управління відсутностями є багатогранним процесом, який поєднує в собі адміністративні, фінансові, юридичні та соціально-психологічні аспекти. Існуючі ручні або частково автоматизовані методи управління не задовольняють потреб сучасних організацій через низьку швидкість обробки запитів, високу ймовірність помилок та незадовільний користувацький досвід. Вирішення цих проблем лежить у площині створення інтелектуальних інформаційних систем, які здатні не лише автоматизувати рутинні операції workflow, але й забезпечити новий рівень взаємодії між людиною та системою завдяки використанню технологій штучного інтелекту. Це дозволить перетворити управління відсутностями з бюрократичного тягаря на ефективний сервіс підтримки життєдіяльності компанії.

1.2 Автоматизація HR-процесів в сучасних організаціях

Цифрова трансформація бізнесу докорінно змінила підходи до управління організаційними процесами, і сфера управління людськими ресурсами (Human Resource Management, HRM) не стала винятком. Якщо традиційно функція кадрового адміністрування сприймалася як суто допоміжна та бюрократична, то в умовах цифрової економіки вона набуває стратегічного значення. Автоматизація HR-процесів - це не просто переведення паперових документів у цифровий формат, а комплексна зміна парадигми взаємодії між роботодавцем та працівником, спрямована на оптимізацію використання трудових ресурсів та підвищення загальної ефективності бізнесу.

Сучасні тенденції свідчать про перехід від розрізнених програмних рішень до інтегрованих екосистем управління талантами. Згідно з дослідженнями провідних консалтингових агентств, рівень автоматизації HR-процесів є одним із ключових індикаторів технологічної зрілості компанії. Це зумовлено тим, що витрати на персонал у сфері інтелектуальних послуг можуть складати до 70-80%

операційних витрат компанії. Отже, навіть незначна оптимізація адміністративних процедур, пов’язаних з персоналом, може дати відчутний економічний ефект.

Основним драйвером автоматизації виступає концепція Employee Self-Service (ESS) - самообслуговування працівників [3]. Це модель організації інформаційної системи, за якої працівники отримують прямий доступ до своїх кадрових даних та можуть самостійно ініціювати більшість стандартних процедур без залучення HR-спеціалістів як посередників. Впровадження порталів самообслуговування дозволяє вирішити низку критичних завдань:

Розвантаження HR-відділу. Замість ручного введення даних про відпустки, оновлення контактної інформації чи видачі довідок, HR-менеджери можуть фокусуватися на стратегічних завданнях: розвитку корпоративної культури, навчанні персоналу та рекрутингу.

Підвищення точності даних. Коли працівник самостійно вносить інформацію, ймовірність помилок при "подвійному введенні" нівелюється.

Прискорення прийняття рішень. Автоматизовані системи дозволяють налаштувати тригери та сповіщення, завдяки чому менеджери миттєво дізнаються про запити підлеглих, що суттєво скорочує цикл погодження (workflow approval cycle).

Важливим аспектом сучасної автоматизації є перехід від монолітних "коробкових" рішень (On-premise) до хмарних платформ (SaaS - Software as a Service). Хмарна архітектура забезпечує масштабованість, дозволяючи компаніям швидко адаптувати систему під зростання штату без необхідності закупівлі дороговартісного серверного обладнання. Більше того, SaaS-рішення сприяють мобільності: доступ до корпоративних сервісів стає можливим з будь-якого пристрою та локації, що є критично важливим для забезпечення роботи розподілених команд.

В архітектурі сучасних HR-систем можна виділити кілька ключових функціональних блоків, які підлягають автоматизації:

ATS (Applicant Tracking Systems) - системи управління кандидатами, що автоматизують рекрутинг: від публікації вакансій до надсилання офферу [4].

Onboarding, Offboarding - автоматизація процесів адаптації нових працівників (створення облікових записів, призначення ментора) та звільнення (блокування доступів, розрахунок вихідних виплат).

Performance Management - цифрові інструменти для постановки цілей (OKR/KPI), проведення оцінювання та збору зворотного зв'язку.

Leave, Attendance Management - управління робочим часом та відсутностями, що є предметом дослідження даної роботи.

Особливу увагу слід приділити інтеграційним можливостям сучасних систем. Жодна HR-платформа не функціонує у вакуумі. Для забезпечення безшовності бізнес-процесів необхідна тісна інтеграція з корпоративними месенджерами (Slack, Microsoft Teams), таск-трекерами (Jira, Asana), календарями (Google Calendar, Outlook) та обліковими системами. Використання відкритих API (Application Programming Interface) дозволяє створювати гнучкі сценарії, коли, наприклад, затвердження відпустки в HR-системі автоматично блокує слот у календарі працівника, змінює його статус у Slack на "У відпустці" та переносить призначені на нього задачі в Jira на колег.

Проте, незважаючи на високий рівень розвитку технологій, багато організацій стикаються з проблемою "зоопарку систем". Працівники змушені використовувати різні інтерфейси для різних задач: одну програму для бронювання переговорних кімнат, іншу для запиту відпустки, третю для перегляду зарплатної відомості. Це явище, відоме як "App Fatigue" (втома від додатків), призводить до зниження продуктивності та небажання користуватися впровадженими інструментами. Складні інтерфейси веб-порталів, перевантажені функціоналом, часто стають бар’єром для швидкого виконання простих операцій.

Саме ця проблема стимулює розвиток наступного етапу автоматизації - впровадження розмовного штучного інтелекту (Conversational AI) та чат-ботів. Інтеграція HR-сервісів у звичне середовище спілкування (корпоративний месенджер) дозволяє реалізувати концепцію "Work where you communicate". Замість того, щоб змушувати користувача вивчати інтерфейс нової системи, автоматизація приходить до користувача у вигляді діалогового вікна. Це знаменує

перехід від графічних інтерфейсів користувача (GUI) до розмовних інтерфейсів (CUI), де природна мова стає головним інструментом керування процесами.

Аналіз показує, що хоча ринок пропонує широкий спектр рішень для автоматизації HR-процесів, існує значний розрив між функціональністю потужних Enterprise-систем та потребою користувачів у простоті та інтуїтивності. Більшість існуючих систем фокусуються на потребах адміністраторів, залишаючи досвід рядового працівника на другому плані. Тому актуальним завданням є розробка рішень, які поєднують надійність backend-логіки корпоративних систем з легкістю взаємодії, притаманною сучасним LLM-моделям [5].

Таким чином, автоматизація HR-процесів є незворотним еволюційним етапом розвитку корпоративного управління. Вона створює фундамент для накопичення "великих даних" (People Analytics), які дозволяють переходити від реактивного реагування на проблеми до проактивного прогнозування ризиків та управління ефективністю команд. У контексті даної магістерської роботи, автоматизація розглядається не як кінцева мета, а як засіб створення інтелектуального середовища, де рутинні транзакції делегуються програмним алгоритмам, звільняючи людський інтелект для творчих та аналітичних задач.

1.3 Огляд та порівняльний аналіз існуючих програмних рішень

Ринок програмного забезпечення для управління людськими ресурсами (HRM) та управління робочою силою (WFM) характеризується високим ступенем насиченості та конкуренції. За даними аналітичних звітів Gartner та IDC, глобальний ринок хмарних рішень для HR продовжує зростати, пропонуючи інструменти для компаній будь-якого масштабу - від невеликих стартапів до транснаціональних корпорацій. Для об’єктивного визначення місця розроблюваної системи TimelyMind у загальному ландшафті програмних продуктів необхідно провести детальний аналіз найбільш поширених аналогів, виявити їхні сильні сторони та концептуальні обмеження.

У рамках даного дослідження для порівняльного аналізу було обрано чотири провідні платформи, що широко використовуються як на глобальному ринку, так і в українському IT-секторі: BambooHR, Workday, Factorial та Sage HR.

BambooHR позиціонується як провідне SaaS-рішення для малого та середнього бізнесу (SMB). Система здобула популярність завдяки інтуїтивно зрозумілому інтерфейсу та простоті налаштування. Модуль управління відсутностями дозволяє створювати різні політики нарахування днів (accruals), візуалізувати відпустки команди та генерувати звіти. Переваги: Висока якість UX/UI, наявність мобільного додатку, легка інтеграція з популярними системами (Slack, Google Calendar). Недоліки: Система є закритою хмарною платформою без можливості локального розгортання (On-premise), що може бути критичним для компаній з суворими політиками безпеки даних. Функціонал workflow є досить лінійним і не дозволяє налаштовувати складні, розгалужені ланцюжки погоджень залежно від контексту запиту. Найголовнішим недоліком у контексті даного дослідження є відсутність вбудованого інтелектуального помічника для обробки запитів природною мовою; вся взаємодія відбувається через стандартні форми графічного інтерфейсу.

Workday - це комплексне рішення класу Enterprise, орієнтоване на великі корпорації зі складними організаційними структурами. Workday пропонує найширший спектр функціональності, включаючи фінансовий менеджмент, планування талантів та аналітику. Переваги: Глибока кастомізація бізнес-процесів, потужний аналітичний модуль на базі AI (використовується для прогнозування відтоку кадрів та аналізу навичок), високий рівень безпеки. Недоліки: Надзвичайна складність впровадження та підтримки (вимагає сертифікованих консультантів), висока вартість ліцензії, що робить систему недоступною для малого та середнього бізнесу. Інтерфейс системи часто критикують за перевантаженість. Хоча Workday інтегрує елементи машинного навчання, вони спрямовані переважно на аналітику, а не на спрощення щоденної операційної взаємодії пересічного користувача з системою через діалоговий інтерфейс.

Factorial Європейська HR-платформа, що стрімко набирає популярність завдяки своїй модульності та орієнтації на автоматизацію рутинних процесів. Factorial пропонує зручний інструментарій для управління відсутностями з візуалізацією в календарі. Переваги: Гнучкість налаштувань, відповідність європейським нормам захисту даних (GDPR), відносно доступна цінова політика. Недоліки: Обмежені можливості API для глибокої інтеграції з кастомними внутрішніми сервісами компанії. Функціонал штучного інтелекту знаходиться на початковому етапі розвитку і не пропонує повноцінного розмовного інтерфейсу для управління заявками.

Sage HR (раніше CakeHR) Система фокусується саме на управлінні відсутностями та розкладом змін, хоча згодом розширила функціонал до повноцінної HRIS. Переваги: Один з найкращих на ринку модулів візуалізації графіків та змін, підтримка складних сценаріїв нарахування відпусток (наприклад, перенесення залишків, погодинні відпустки). Недоліки: Інтерфейс взаємодії залишається класичним (point-and-click). Відсутність україномовного інтерфейсу та підтримки специфіки українського трудового законодавства вимагає додаткових зусиль на адаптацію.

Також варто зазначити проблему інтеграції. Великі Enterprise-системи на кшталт Workday є "важковаговиками", інтеграція яких з локальними месенджерами або специфічними внутрішніми інструментами розробки (наприклад, внутрішніми адмін-панелями) вимагає значних ресурсів. Розробка спеціалізованого мікросервісного рішення на Python (FastAPI) дозволяє створити легку, гнучку систему, яка фокусується на вирішенні однієї конкретної проблеми (управління відсутностями) з максимальною ефективністю, не перевантажуючи користувача зайвим функціоналом.

Для систематизації отриманих даних проведемо порівняльний аналіз функціональних можливостей розглянутих систем та проєктованої системи TimelyMind. Результати наведено у Таблиці 1.1.

Таблиця 1.1 - Порівняльний аналіз систем управління відсутностями

Критерій порівняння	**BambooHR**	**Workday**	**Factorial**	**Sage HR**	**TimelyMind (Project)**
Цільова аудиторія	SMB	Enterprise	SMB або Mid-market	SMB	IT-компанії, Mid-market
Інтерфейс взаємодії	GUI (Web/Mobile)	GUI (Complex)	GUI	GUI	Conversational UI (NLP) + GUI
Обробка природної мови (NLP)	Відсутня	Обмежена (пошук)	Відсутня	Відсутня	Основний канал API
Модель розгортання	SaaS (Cloud)	SaaS (Cloud)	SaaS (Cloud)	SaaS (Cloud)	Hybrid / On-premise (Docker)
Гнучкість Workflow	Середня	Висока	Середня	Висока	Висока Configurable
Інтеграція (API)	REST API	SOAP/REST	REST API	REST API	REST API (FastAPI)
Складність впровадження	Низька	Дуже висока	Низька	Низька	Середня
Вартість	Середня	Дуже висока	Середня	Середня	Низька (Open Source base)

Як видно з таблиці 1.1, існуючі на ринку рішення мають спільний недолік - домінування традиційного графічного інтерфейсу (GUI) як єдиного засобу взаємодії. Навіть при наявності мобільних додатків, процес подачі заявки вимагає від користувача навігації по меню, вибору дат у календарних віджетах та ручного заповнення форм. Жодна з розглянутих систем не пропонує повноцінної

можливості керування процесом через природномовні команди (наприклад, через чат-бот, інтегрований з LLM), що могло б суттєво спростити UX та скоротити час на виконання операцій.

Крім того, більшість комерційних рішень розповсюджуються виключно за моделлю SaaS. Це створює бар'єри для організацій, які через вимоги безпеки або внутрішні політики прагнуть зберігати кадрові дані на власних серверах (On-premise) або у приватному хмарному середовищі. Проектована система TimelyMind, завдяки використанню контейнеризації Docker, може бути розгорнута в будь-якому інфраструктурному середовищі, що є конкурентною перевагою.

Отже, проведений аналіз дозволяє зробити висновок про наявність вільної ніші на ринку HRM-систем. Існує потреба у створенні рішення, яке б поєднувало надійність обліку відсутностей з інноваційними методами взаємодії на основі штучного інтелекту. Це дозволить не лише автоматизувати процеси, але й якісно змінити досвід користувача (User Experience), зробивши його більш природним та інтуїтивним.

1.4 Технології обробки природної мови та великі мовні моделі

Стрімкий розвиток штучного інтелекту в останнє десятиліття зумовив якісний стрибок у методах взаємодії між людиною та комп'ютерними системами. Ключову роль у цьому процесі відіграє обробка природної мови (Natural Language Processing, NLP) - міждисциплінарна галузь, що поєднує комп'ютерну лінгвістику, машинне навчання та глибоке навчання. Головною метою NLP є надання комп'ютерам здатності розуміти, інтерпретувати та генерувати людську мову у спосіб, що є змістовним та корисним. Для розроблюваної системи TimelyMind технології NLP виступають не просто додатковою функцією, а фундаментом для побудови користувацького інтерфейсу нового покоління.

Традиційні підходи до побудови діалогових систем базувалися на жорстких правилах та пошуку ключових слів. Такі системи функціонували за принципом сценарних дерев: якщо повідомлення користувача містило слово "відпустка", бот пропонував відповідне меню. Однак такі алгоритми виявлялися безсилими перед

варіативністю людської мови. Синонімія, омонімія, сленг, граматичні помилки або складні синтаксичні конструкції призводили до нерозуміння запиту системою. Сучасний етап розвитку NLP пов'язаний з переходом від статистичних методів до використання нейронних мереж, зокрема глибокого навчання (Deep Learning).

Значним проривом у обробці послідовностей стала поява рекурентних нейронних мереж (RNN) та їх вдосконаленої варіації - мереж довгої короткострокової пам'яті. Ці архітектури дозволили враховувати контекст попередніх слів при обробці поточного елемента послідовності. Проте вони мали суттєвий недолік - проблему зникаючого градієнта, що унеможливлювало ефективну обробку довгих текстів та встановлення залежностей між віддаленими словами у реченні.

Революцією у сфері NLP стала поява у 2017 році архітектури Трансформер (Transformer), запропонованої дослідниками Google. В основі цієї архітектури лежить механізм "уваги". Цей механізм дозволяє моделі оцінювати важливість кожного слова у реченні відносно інших слів, незалежно від відстані між ними. Наприклад, у фразі "Менеджер погодив мою відпустку, оскільки вона була запланована заздалегідь", механізм уваги дозволяє системі зрозуміти, що займенник "вона" стосується слова "відпустка", а не слова "менеджер". Саме архітектура трансформерів стала основою для створення великих мовних моделей, таких як BERT, GPT та Gemini [6].

Великі мовні моделі представляють собою нейронні мережі з мільярдами параметрів, навчені на величезних масивах текстових даних. На відміну від спеціалізованих моделей, які тренуються під одну конкретну задачу (наприклад, класифікація спаму), LLM є фундаментальними моделями. Вони здатні виконувати широкий спектр завдань завдяки здатності до генералізації та навчання "з нуля" (zero-shot learning) або на невеликій кількості прикладів.

У контексті системи управління відсутностями, використання LLM (зокрема Google Gemini) дозволяє вирішувати дві критично важливі задачі:

Розпізнавання намірів (Intent Recognition). Це процес визначення мети користувача на основі його текстового повідомлення. Класичні класифікатори

вимагають тисяч розмічених прикладів для кожного класу. LLM, володіючи семантичним розумінням мови, здатна коректно класифікувати фразу "Мені треба кілька днів відпочити наступного тижня" як намір create_vacation_request, навіть якщо слово "відпустка" не було вжито явно.

Вилучення іменованих сутностей (Named Entity Recognition, NER). Для формування запиту системі необхідні структуровані дані: дати початку та кінця, тип відсутності. LLM здатна виділяти ці сутності з неструктурованого тексту. Особливою перевагою є здатність обробляти відносні часові конструкції. Фраза "наступної п'ятниці" або "через два дні" автоматично конвертується моделлю у конкретну дату формату YYYY-MM-DD на основі поточного контексту [7].

Для повноцінної роботи інтелектуального помічника недостатньо лише розпізнати намір. Необхідно реалізувати механізм заповнення слотів (Slot Filling). Якщо користувач пише "Хочу у відпустку", система розпізнає намір, але їй бракує даних про дати. Сучасні LLM дозволяють реалізувати природний діалог для уточнення інформації. Модель виступає у ролі менеджера діалогу (Dialogue Manager), який зберігає контекст розмови та ставить уточнюючі запитання ("На які саме дати ви плануєте відпустку?") до тих пір, поки всі необхідні параметри (слоти) не будуть заповнені.

Модель Gemini від Google відноситься до класу мультимодальних генеративних моделей. Її архітектура оптимізована не лише для розуміння тексту, але й для складних логічних висновків (reasoning). Це є критично важливим для перевірки бізнес-правил. Наприклад, перш ніж сформувати запит до бази даних, модель може проаналізувати запит користувача на логічну несуперечливість (дата кінця відпустки не може передувати даті початку).

Інтеграція з такими моделями відбувається через API, що дозволяє відокремити логіку "розумного" аналізу тексту від бізнес-логіки додатку. Важливим аспектом є використання техніки Prompt Engineering - розробки спеціальних інструкцій для моделі, які визначають її роль, обмеження та формат вихідних даних. Для системи TimelyMind це означає, що ми можемо інструктувати

модель повертати відповіді виключно у форматі JSON, що робить їх придатними для машинної обробки програмним кодом на стороні серверу.

Попри значні переваги, використання великих мовних моделей несе в собі певні ризики, які необхідно враховувати при проектуванні системи. Основним викликом є явище "галюцинацій", коли модель генерує правдоподібну, але фактично неправдиву інформацію. У контексті HR-системи це неприпустимо (наприклад, модель не може вигадувати залишок днів відпустки). Для нівелювання цього ризику в роботі застосовується архітектурний патерн RAG (Retrieval-Augmented Generation) або функціональний виклик (Function Calling). Суть підходу полягає в тому, що модель не використовує власні "знання" для надання фактологічної відповіді. Замість цього вона формує запит до зовнішньої функції (наприклад, get_balance(user_id)), отримує достовірні дані з бази даних і лише потім генерує текстову відповідь користувачу на основі отриманих фактів. Це дозволяє поєднати красномовність та гнучкість LLM з точністю традиційних баз даних.

Підсумовуючи, можна стверджувати, що інтеграція сучасних технологій NLP та LLM у системи управління персоналом дозволяє подолати розрив між складною внутрішньою логікою програмного забезпечення та природним способом людської комунікації. Замість того, щоб змушувати користувача вивчати інтерфейс системи, система вчиться розуміти мову користувача.

1.5 Постановка задачі дослідження

На основі проведеного аналізу предметної області та існуючих технологічних рішень можна сформулювати постановку задачі даної магістерської роботи. Аналіз показав, що існуючі системи управління відсутностями, незважаючи на широку функціональність, страждають від складності інтерфейсів та відсутності гнучких інструментів взаємодії. Водночас, розвиток технологій NLP відкриває нові можливості для автоматизації рутинних операцій.

Таким чином, науково-технічна задача полягає у створенні інформаційної системи, яка забезпечує автоматизацію процесу управління відсутностями працівників шляхом інтеграції веб-платформи з модулем обробки природної мови.

Для розв'язання цієї задачі необхідно виконати наступні етапи:

1. Розробити архітектуру веб-додатку, що базується на принципах чистої архітектури та мікросервісних підходах, забезпечуючи масштабованість та надійність.
2. Спроектувати реляційну базу даних для зберігання інформації про користувачів, їхні баланси, історію запитів та структуру компанії.
3. Реалізувати REST API для забезпечення взаємодії між клієнтською частиною, сервером та зовнішніми сервісами.
4. Розробити та налаштувати модуль взаємодії з AI API, включаючи розробку системних промптів для коректного розпізнавання намірів та вилучення сутностей.
5. Створити адаптивний користувацький інтерфейс, який поєднує класичні елементи керування (дашборди, календарі) з чат-інтерфейсом.
6. Впровадити механізм рольового доступу та автоматичний workflow для маршрутизації заявок між працівниками та менеджерами.

Вирішення поставлених задач дозволить створити програмний продукт TimelyMind, який поєднає в собі надійність корпоративних систем обліку з простотою та зручністю сучасних комунікаційних платформ.

2 ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ

2.1 Архітектурні рішення для веб-додатків

Проектування архітектури програмного забезпечення є фундаментальним етапом розробки будь-якої інформаційної системи, оскільки саме на цьому етапі закладаються основи її надійності, масштабованості та зручності супроводу. Під архітектурою розуміють сукупність рішень щодо організації програмної системи, які включають вибір структурних елементів, їх інтерфейсів, а також поведінки у рамках взаємодії цих компонентів. Для системи TimelyMind, що поєднує в собі функціонал класичного корпоративного обліку та інтелектуальні компоненти обробки природної мови, вибір архітектурного патерну є критичним фактором успіху.

Враховуючи сучасні вимоги до веб-розробки, для реалізації системи було обрано клієнт-серверну архітектуру з використанням принципів REST. Цей підхід передбачає чітке розмежування відповідальності між стороною, що ініціює запит, та стороною, що обробляє дані. У рамках цієї моделі доцільним є використання трирівневої архітектури, яка дозволяє логічно розділити додаток на три незалежні шари: рівень представлення, рівень бізнес-логіки та рівень даних.

Рівень представлення відповідає за графічний інтерфейс та взаємодію з користувачем. У розроблюваній системі він реалізується як односторінковий веб-додаток, що дозволяє перенести частину обчислювального навантаження, наприклад рендерінг сторінок та валідацію форм, на браузер клієнта. Це забезпечує високу швидкість відгуку інтерфейсу, наближену до нативних програм. Рівень бізнес-логіки виступає ядром системи, де відбуваються основні обчислення, обробка даних, виконання робочих процесів та взаємодія з моделями штучного інтелекту. Цей шар реалізується у вигляді програмного інтерфейсу, який приймає запити, обробляє їх та повертає результат. Третій рівень забезпечує збереження та вилучення інформації, гарантуючи цілісність даних та абстрагуючи логіку від особливостей конкретної системи управління базами даних.

Взаємодія між компонентами системи побудована на основі архітектурного стилю REST. Він накладає ряд обмежень, дотримання яких дозволяє створити продуктивну систему. Ключовим аспектом є відсутність збереження стану клієнта на сервері між запитами. Кожен запит повинен містити всю необхідну інформацію для його обробки, включаючи дані для автентифікації [8]. Це є критично важливим для масштабованості, оскільки дозволяє обробляти запити від одного користувача різними екземплярами серверного додатку без прив'язки до конкретного сервера. Цей стиль накладає ряд обмежень, дотримання яких дозволяє створити продуктивну та масштабовану систему:

- Уніфікований інтерфейс. Всі ресурси системи (користувачі, заявки на відпустку) ідентифікуються за допомогою унікальних URI (Uniform Resource Identifier), а маніпуляції з ними здійснюються стандартними методами протоколу HTTP.
- Відсутність стану (Stateless). Сервер не зберігає інформацію про стан клієнта між запитами. Кожен запит повинен містити всю необхідну інформацію для його обробки, включаючи дані для автентифікації. Це критично важливо для масштабованості, оскільки дозволяє обробляти запити від одного користувача різними екземплярами серверного додатку.
- Кешування. Відповіді сервера можуть бути позначені як такі, що кешуються. Це дозволяє клієнту повторно використовувати отримані дані, зменшуючи навантаження на мережу та сервер.

Розроблена система TimelyMind базується на трирівневій клієнт-серверній архітектурі, що забезпечує чітке розмежування функціональних обов'язків між компонентами. Верхній, клієнтський рівень, представлений веб-додатками для різних типів пристроїв, які взаємодіють із сервером через захищений протокол HTTPS за допомогою REST API. Серверний рівень реалізовано на базі фреймворку FastAPI, який об'єднує модулі бізнес-логіки, включаючи модуль автентифікації, API для роботи з відсутностями та календарем, підсистему workflow, а також клієнт для взаємодії із зовнішнім сервісом API. Нижній рівень відповідає за зберігання

даних і включає реляційну СКБД PostgreSQL для основної інформації та NoSQL-сховище Redis для кешування та управління сесіями. Детальна структурна схема системи, що ілюструє взаємозв'язки між цими компонентами, зображена на рисунку 2.1.

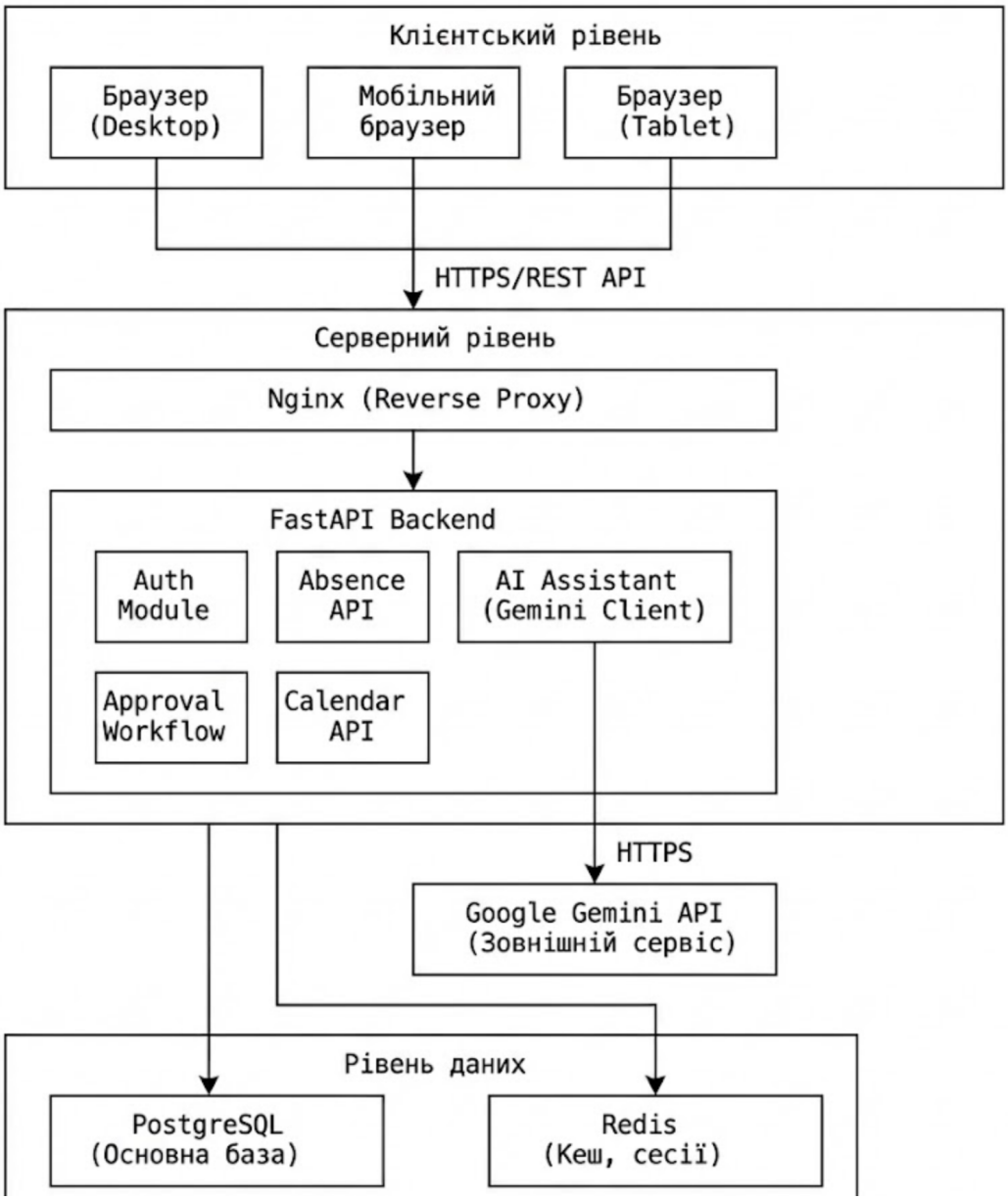


Рисунок 2.1 – Структурна схема додатку

При виборі структурної організації коду було проаналізовано переваги та недоліки монолітного та мікросервісного підходів. Для системи TimelyMind обрано гібридний варіант - модульний моноліт. Це означає, що система розробляється як єдиний модуль розгортання, але всередині код чітко структурований по доменних

областях: автентифікація, управління відсутностями, обробка природної мови, сповіщення. Модулі взаємодіють між собою через чітко визначені інтерфейси, що забезпечує високу зв'язність в межах компонентів та слабку пов'язаність між ними. Такий підхід дозволяє зберегти простоту розробки на початкових етапах, залишаючи можливість легкого виділення окремих частин у незалежні сервіси в майбутньому.

Окрему увагу в архітектурі приділено інтеграції з великими мовними моделями. Оскільки звернення до зовнішнього API є операцією з високою латентністю, архітектура передбачає асинхронну обробку запитів. Серверна частина не блокується під час очікування відповіді від інтелектуального агента, а продовжує обробляти інші запити, використовуючи механізми асинхронного програмування. Це забезпечує ефективне використання обчислювальних ресурсів та стабільну роботу системи навіть при значному навантаженні.

2.2. Інструменти розробки серверної частини

Реалізація серверної складової є ключовим етапом у створенні системи TimelyMind, оскільки саме цей рівень відповідає за обробку бізнес-логіки, безпеку даних та інтеграцію з інтелектуальними сервісами. Вибір технологічного стеку здійснювався на основі критеріїв продуктивності, швидкості розробки, підтримки асинхронних операцій та наявності розвиненої екосистеми бібліотек для роботи зі штучним інтелектом. У наступних підрозділах детально розглянуто обґрунтування вибору мови програмування, веб-фреймворку, системи управління базами даних та допоміжних інструментів, що у сукупності формують надійний фундамент програмного продукту [9].

2.2.1 Мова програмування Python та фреймворк FastAPI

При виборі мови програмування для реалізації серверної логіки було проведено порівняльний аналіз найбільш поширених технологій веб-розробки, серед яких Java, Go, Node.js та Python. Визначальним фактором у контексті даної

магістерської роботи стала необхідність глибокої інтеграції з технологіями обробки природної мови та великими мовними моделями. На сьогоднішній день Python є беззаперечним лідером у сфері наук про дані та машинного навчання. Це інтерпретована об'єктно-орієнтована мова високого рівня, яка вирізняється лаконічним синтаксисом та динамічною типізацією.

Основним аргументом на користь використання Python є наявність найбагатшого набору бібліотек для роботи зі штучним інтелектом. Використання цієї мови дозволяє реалізувати модуль обробки мови нативно, без необхідності створення складних мостів між різномовними середовищами. Крім того, виразний синтаксис дозволяє писати код, який легко читати та підтримувати, що суттєво скорочує час на реалізацію прототипів та внесення змін у бізнес-логіку. Python ефективно справляється як із задачами веб-серверу, так і з фоновими обчисленнями, що дозволяє використовувати єдиний інструмент для всього серверного стеку.

Для побудови програмного інтерфейсу було обрано сучасний фреймворк FastAPI. На відміну від класичних рішень, таких як Django або Flask, FastAPI створений з урахуванням сучасних стандартів веб-розробки та вимог до високого навантаження. Він базується на стандарті ASGI, що забезпечує продуктивність на рівні компільованих мов завдяки неблокуючій обробці запитів. Це є критично важливим показником, оскільки поки система очікує відповідь від зовнішнього API моделі Gemini, сервер може паралельно обробляти запити інших користувачів.

Важливою перевагою фреймворку є вбудована валідація даних завдяки інтеграції з бібліотекою Pydantic. Це дозволяє описувати структуру даних за допомогою стандартних підказок типів, автоматично перевіряти вхідні параметри та генерувати зрозумілі повідомлення про помилки. Також FastAPI автоматично генерує інтерактивну документацію API, що спрощує тестування точок доступу та взаємодію з розробниками клієнтської частини. Підтримка сучасного синтаксису асинхронного програмування дозволяє писати конкурентний код у лінійному стилі, що є необхідним для ефективної роботи з базами даних та зовнішніми сервісами [10].

2.2.2 ORM SQLAlchemy та СКБД PostgreSQL

Надійність зберігання та цілісність даних є критичними вимогами до будь-якої системи кадрового обліку, оскільки втрата або спотворення інформації про відпустки може призвести до фінансових помилок та юридичних наслідків. Саме тому як система управління базами даних була обрана PostgreSQL - потужна об'єктно-реляційна система з відкритим вихідним кодом. Вона зарекомендувала себе як стандарт індустрії завдяки суворому дотриманню принципів ACID, що гарантує атомарність, узгодженість, ізольованість та довговічність транзакцій [11]. Для системи TimelyMind це означає, що операція списання днів з балансу працівника буде або виконана повністю, або не виконана взагалі, виключаючи проміжні некоректні стани навіть у випадку збоїв обладнання.

Важливою особливістю PostgreSQL, що вплинула на її вибір для даного проекту, є вбудована підтримка роботи з неструктурованими даними у форматі JSON. Це дозволяє ефективно поєднувати переваги реляційної моделі (чітка структура таблиць користувачів та заявок) з гнучкістю документо-орієнтованих баз даних. Така функціональність є необхідною для збереження історії діалогів з інтелектуальним агентом, параметрів конфігурації робочих процесів, які можуть змінюватися динамічно, а також для логування відповідей від зовнішніх API.

Для забезпечення взаємодії між програмним кодом на Python та базою даних використовується бібліотека SQLAlchemy. Це інструмент об'єктно-реляційного відображення, який дозволяє працювати з базами даних, використовуючи об'єкти та методи мови програмування замість написання "сирих" SQL-запитів. Такий підхід значно підвищує швидкість розробки та безпеку системи, оскільки автоматично екранує вхідні дані, захищаючи додаток від атак типу SQL-ін'єкцій. SQLAlchemy надає розробнику високий рівень абстракції, що дозволяє у майбутньому змінити систему управління базами даних без необхідності переписування основного коду програми.

Визначальним фактором вибору саме цієї бібліотеки у версії 2.0 стала повноцінна підтримка асинхронного режиму роботи. Оскільки архітектура серверної частини базується на асинхронному фреймворку FastAPI, використання

синхронного драйвера бази даних призвело б до блокування потоку виконання під час очікування відповіді від диска або мережі, що нівелювало б переваги високої продуктивності. Комбінація SQLAlchemy з асинхронним драйвером asyncpg дозволяє виконувати запити до бази даних без блокування основного циклу подій, забезпечуючи максимальну пропускну здатність системи.

Керування структурою бази даних здійснюється за допомогою інструменту міграцій Alembic, який є частиною екосистеми SQLAlchemy. Це дозволяє застосовувати підхід "код як інфраструктура", зберігаючи історію змін схеми бази даних у системі контролю версій. Кожна зміна в моделях даних (наприклад, додавання нового типу відсутності) описується у вигляді окремого скрипта міграції, що забезпечує узгодженість структури бази даних на всіх етапах розробки та розгортання, а також дозволяє автоматизувати процес оновлення продуктивного середовища [12].

2.2.3 Автентифікація JWT та черга задач Celery

Забезпечення безпеки доступу до корпоративних даних є критично важливим аспектом розробки будь-якої інформаційної системи, особливо коли йдеться про персональні дані працівників та графіки роботи. Оскільки обрана архітектура REST передбачає відсутність збереження стану клієнта на сервері, традиційні методи автентифікації на основі сесій та файлів cookie виявляються неефективними для побудови масштабованого програмного інтерфейсу. Тому для реалізації механізму контролю доступу було обрано стандарт JSON Web Token. Цей підхід дозволяє передавати інформацію про користувача та його права у зашифрованому вигляді безпосередньо у заголовку HTTP-запиту [13]. Головною перевагою такої технології є можливість створення системи без збереження сесій на серверній стороні, що дозволяє легко масштабувати додаток, додаючи нові екземпляри серверів без необхідності синхронізації сесій між ними.

Використання токенів також спрощує інтеграцію з мобільними додатками та сторонніми сервісами, оскільки механізм автентифікації залишається уніфікованим незалежно від платформи клієнта. Для підвищення рівня безпеки реалізовано схему

з використанням пари токенів: короткострокового токена доступу, який використовується для авторизації запитів, та довгострокового токена оновлення, що зберігається у захищеному сховищі та використовується для отримання нових ключів доступу без необхідності повторного введення пароля користувачем. Такий підхід мінімізує ризики у випадку перехоплення даних, оскільки час життя основного ключа доступу суворо обмежений.

Окрім обробки синхронних запитів користувачів, сучасна веб-система повинна виконувати низку фонових операцій, які можуть вимагати значного часу для виконання. До таких задач у системі TimelyMind належать надсилання електронних листів із підтвердженням відпусток, генерація складних звітів, регулярне нарахування днів відпустки згідно з графіком та взаємодія з повільними зовнішніми сервісами. Виконання цих операцій у межах основного циклу обробки HTTP-запиту призвело б до суттєвої затримки відповіді сервера та погіршення користувацького досвіду. Для вирішення цієї проблеми використано асинхронну чергу задач Celery.

Цей інструмент дозволяє винести важкі обчислювальні процеси за межі основного веб-додатку, делегуючи їх виконання окремим робочим процесам. В якості брокера повідомлень, який забезпечує комунікацію між веб-сервером та виконавцями задач, обрано систему Redis. Це високопродуктивне сховище даних у оперативній пам'яті, що забезпечує миттєву передачу інформації про нові задачі. Така архітектура гарантує, що інтерфейс користувача залишається чутливим навіть під час виконання ресурсномістких операцій: сервер миттєво приймає запит, ставить задачу в чергу та повертає підтвердження користувачу, тоді як фактичне виконання відбувається у фоновому режимі. Крім того, Celery надає вбудовані механізми для планування періодичних завдань, що дозволяє автоматизувати рутинні адміністративні процедури без втручання людини.

2.3 Інструменти розробки клієнтської частини

Розробка клієнтської частини є визначальним етапом у створенні сучасного веб-продукту, оскільки саме інтерфейс користувача формує перше враження про

систему та визначає ефективність виконання робочих завдань. Для системи TimelyMind, яка передбачає активну взаємодію користувача з динамічними елементами, такими як календарні сітки, інтерактивні графіки та чат-бот, критичними вимогами є висока швидкість відгуку та реактивність інтерфейсу. Реалізація цих вимог здійснюється шляхом побудови однострінокового додатку (Single Page Application), де завантаження сторінки відбувається лише один раз, а подальша взаємодія реалізується через динамічне оновлення контенту без перезавантаження браузера.

Основою для побудови програмного інтерфейсу обрано бібліотеку React. Це технологічне рішення дозволяє застосувати компонентний підхід до розробки, розділяючи складний інтерфейс на незалежні, ізольовані частини, такі як кнопки, форми, картки працівників або елементи календаря. Кожен компонент містить власну логіку та розмітку, що значно спрощує підтримку коду та дозволяє повторно використовувати елементи в різних частинах системи. Ключовою перевагою React, що зумовила його вибір, є використання механізму віртуального DOM [14]. Ця технологія створює легковагову копію структури документу в пам'яті та при зміні стану додатку обчислює мінімально необхідну кількість операцій для оновлення реального інтерфейсу. Такий підхід забезпечує високу продуктивність навіть при роботі з великими масивами даних, що є актуальним для відображення графіків відпусток великих департаментів.

Важливим аспектом розробки на React є декларативний стиль програмування. Розробник описує бажаний стан інтерфейсу для різних умов, а бібліотека автоматично забезпечує приведення відображення до цього стану при зміні даних. Це дозволяє уникнути складних маніпуляцій з елементами сторінки вручну та зменшує кількість потенційних помилок. Для управління станом додатку та обміну даними між компонентами використовуються сучасні можливості бібліотеки, зокрема хуки, які дозволяють інкапсулювати логіку роботи з даними та побічними ефектами.

Для реалізації візуального стилю та адаптивності інтерфейсу використано CSS-фреймворк Tailwind CSS. На відміну від традиційних підходів, що базуються

на написанні семантичних класів, Tailwind пропонує методологію utility-first. Стилізація елементів відбувається шляхом комбінування заздалегідь визначених низькорівневих класів безпосередньо у розмітці компонента [15]. Це дозволяє значно прискорити процес верстки та уникнути проблем із конфліктами стилів у великому проекті. Крім того, фреймворк містить вбудовану систему адаптивності, що дозволяє легко налаштовувати відображення елементів для різних розмірів екранів, забезпечуючи коректну роботу системи як на настільних комп'ютерах, так і на мобільних пристроях.

Взаємодія клієнтської частини з сервером реалізується через асинхронні HTTP-запити до REST API. Це дозволяє відокремити процес отримання даних від їх відображення, забезпечуючи плавний досвід користувача. Під час очікування відповіді від сервера або обробки запиту модулем штучного інтелекту, інтерфейс не блокується, а користувачу відображаються відповідні індикатори завантаження. Така архітектура клієнтського додатку повністю відповідає сучасним стандартам індустрії та забезпечує гнучкість, необхідну для подальшого масштабування системи.

2.4 Контейнеризація та розгортання Docker

Забезпечення відтворюваності середовища виконання є однією з найскладніших задач при розробці та впровадженні розподілених програмних систем. Часто виникають ситуації, коли програмний код коректно функціонує на локальному комп'ютері розробника, але демонструє помилки при перенесенні на сервер через відмінності у версіях бібліотек, налаштуваннях операційної системи або конфігурації мережі. Для вирішення цієї проблеми у проекті TimelyMind застосовано технологію контейнеризації на базі платформи Docker. Вона дозволяє упакувати додаток разом з усіма його залежностями, бібліотеками та конфігураційними файлами у єдиний стандартизований модуль, який називається контейнером.

На відміну від традиційної віртуалізації, яка вимагає розгортання

повноцінної гостьової операційної системи для кожного додатку, контейнери використовують ядро хостової системи, забезпечуючи ізоляцію процесів на рівні операційної системи. Такий підхід гарантує значно вищу ефективність використання обчислювальних ресурсів та швидший час запуску сервісів. Ізоляція гарантує, що конфлікти між залежностями різних компонентів системи виключені [16]. Наприклад, серверна частина може використовувати одну версію системних бібліотек, а модуль аналітики - іншу, і вони функціонуватимуть на одному сервері без взаємного впливу.

Оскільки архітектура системи є багатокомпонентною та включає веб-сервер, базу даних, сховище Redis, клієнтський додаток та воркер черги задач, управління кожним контейнером окремо було б неефективним. Для оркестрації цих сервісів використовується інструмент Docker Compose. У цьому файлі визначаються образи для кожного сервісу, змінні середовища, прокинуті порти та віртуальні мережі, що з'єднують компоненти між собою [17]. Завдяки цьому розгортання всієї системи зводиться до виконання однієї команди в терміналі, що автоматизує процес створення, запуску та налаштування зв'язків між контейнерами.

Розгортання системи TimelyMind виконується з використанням технології контейнеризації Docker, що забезпечує ізольованість компонентів та відтворюваність середовища. Для оркестрації сервісів використовується інструмент Docker Compose, який об'єднує всі необхідні контейнери в єдину інфраструктуру на хост-машині. Як показано на схемі (див. рис. 2.2), всі чотири основні сервіси - frontend (веб-сервер Nginx), backend (сервер додатку FastAPI), db (база даних PostgreSQL) та redis (кеш у пам'яті) - розгорнуті як окремі контейнери у спільній віртуальній мережі timelymind_network. Взаємодія між контейнерами відбувається через внутрішні порти. Для збереження даних бази даних та Redis використовуються іменовані томи Docker. Контейнер backend також має налаштоване HTTPS-з'єднання із зовнішнім сервісом Google Cloud Gemini API для обробки запитів штучного інтелекту. Детальна діаграма розгортання, що ілюструє цю конфігурацію, зображена на рисунку 2.2.

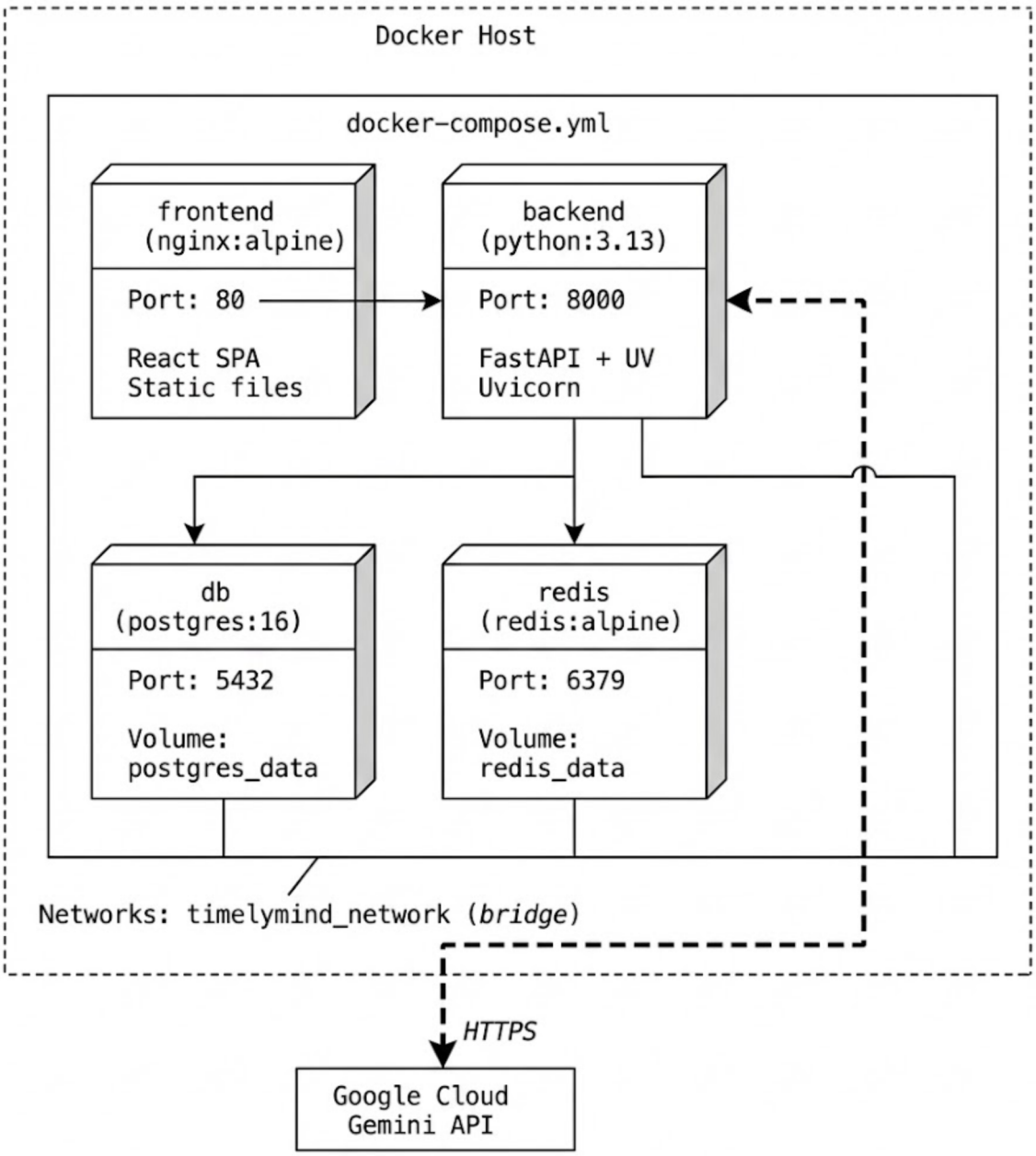


Рисунок 2.2 – Діаграма розгортання

Використання контейнеризації також створює надійну основу для налаштування процесів безперервної інтеграції та доставки. У разі зростання навантаження система дозволяє легко запустити додаткові екземпляри контейнерів обробки на різних серверах, забезпечуючи горизонтальне масштабування без зміни програмного коду. Таким чином, обрана стратегія розгортання забезпечує гнучкість, надійність та портативність розробленого рішення.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Функціональні вимоги до системи

Етап формування функціональних вимог є критично важливим для успішної реалізації програмного продукту, оскільки він визначає чіткі межі поведінки системи та набір операцій, які вона повинна виконувати для задоволення потреб користувачів [18]. Для системи TimelyMind, що позиціонується як інтелектуальний інструмент управління відсутностями, вимоги формуються на основі аналізу бізнес-процесів сучасних IT-компаній та специфіки взаємодії через природномовні інтерфейси.

У процесі проектування було виділено три основні ролі користувачів, кожна з яких має специфічний рівень доступу та набір функціональних можливостей:

- Працівник (Employee) - базовий користувач системи. Основною метою його взаємодії є перегляд власного балансу відпусток, подача запитів на відсутність та відстеження їх статусу.
- Керівник (Manager/Team Lead) - користувач із розширеними правами, відповідальний за управління командою. Окрім функцій працівника, він має повноваження погоджувати або відхиляти запити підлеглих та переглядати календар відсутностей свого відділу для планування ресурсів.
- Адміністратор (HR/Admin) - користувач із повним доступом до системи. Відповідає за налаштування глобальних параметрів, управління обліковими записами користувачів, конфігурацію ланцюжків погодження та перегляд аналітичної звітності по всій організації.

Для візуалізації функціональних можливостей системи та розмежування прав доступу користувачів було розроблено діаграму варіантів використання (див. рис. 3.1). Вона демонструє взаємодію трьох основних акторів: Працівника, Керівника та Адміністратора, з ключовими модулями системи. Працівник має доступ до базових функцій, таких як створення запитів, взаємодія з AI-асистентом та перегляд календаря команди. Роль Керівника успадковує права Працівника та доповнюється можливостями затвердження або відхилення заявок підлеглих. Адміністратор, у

свою чергу, відповідає за управління обліковими записами користувачів та перегляд глобальної статистики по компанії.

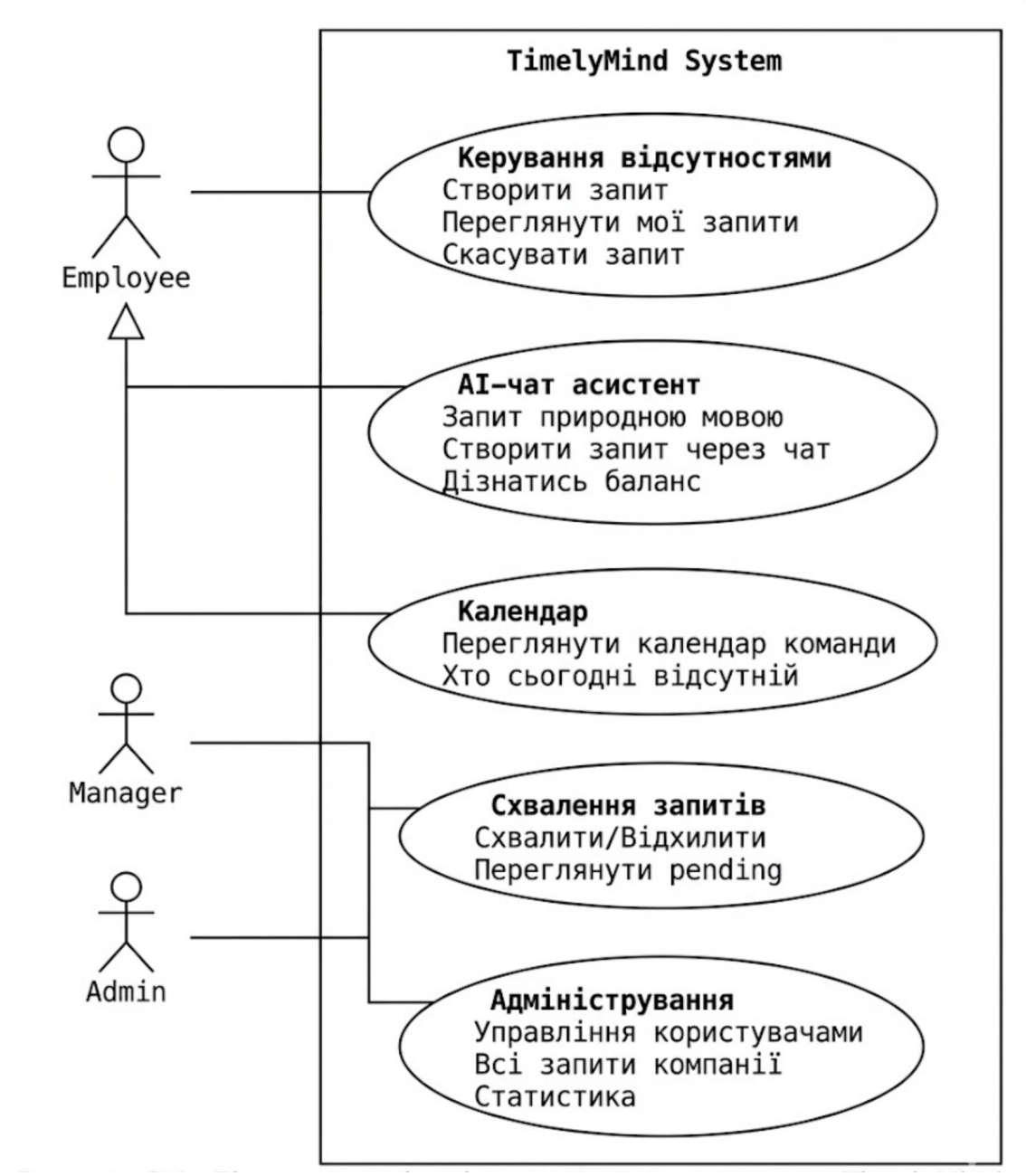


Рисунок 3.1 – Діаграма варіантів використання

На основі визначених ролей сформовано перелік функціональних вимог, згрупованих за логічними модулями системи:

– Система повинна забезпечувати надійний механізм ідентифікації користувачів для захисту корпоративних даних [19].

- Система має підтримувати реєстрацію та вхід користувачів за допомогою електронної пошти та пароля з використанням захищеного протоколу передачі даних.
- Після успішної автентифікації система повинна генерувати токен доступу, який визначає права користувача та час сесії.
- Користувач повинен мати можливість переглядати та редагувати особисту інформацію профілю, змінювати пароль та налаштування сповіщень.
- Система повинна автоматично завершувати сесію користувача після закінчення терміну дії токена безпеки.

Модуль управління відсутностями (Core HRM) це центральний функціональний блок, що забезпечує облік робочого часу [20]:

- Система повинна підтримувати різні типи відсутностей: оплачувана відпустка, лікарняний, відгул за власний рахунок, додаткові вихідні (day off).
- Для кожного типу відсутності система повинна вести окремий баланс доступних днів.
- Система має забезпечувати валідацію запитів на етапі створення: дата закінчення не може передувати даті початку, обраний період не повинен перетинатися з існуючими заявками, а кількість днів не може перевищувати доступний баланс (за винятком спеціальних типів, таких як лікарняний).
- Користувач повинен мати можливість скасувати подану заявку до моменту її фінального затвердження або надіслати запит на скасування вже погодженої відпустки.

Модуль автоматичного погодження (Workflow) це модуль що відповідає за маршрутизацію заявок згідно з ієрархічною структурою компанії:

- При створенні запиту система повинна автоматично визначати безпосереднього керівника ініціатора та надсилати йому сповіщення про необхідність дії.
- Керівник повинен мати можливість затвердити або відхилити запит із додаванням обов'язкового коментаря у випадку відмови.

- Система повинна підтримувати багатоетапне погодження, де запит послідовно проходить через Team Lead, менеджера департаменту та HR-спеціаліста.
- У разі зміни статусу заявки система повинна негайно сповіщати ініціатора через налаштовані канали комунікації (веб-інтерфейс, електронна пошта).

Модуль обробки природної мови (NLP Interface) - ключова функціональна особливість TimelyMind, що відрізняє її від класичних рішень:

- Система повинна надавати чат-інтерфейс для введення запитів у вільній текстовій формі.
- Модуль штучного інтелекту повинен виконувати класифікацію намірів користувача (Intent Recognition), розрізняючи запити на створення відпустки, перевірку балансу, перегляд статусу заявок або отримання довідкової інформації.
- Система повинна вилучати іменовані сутності (Entity Extraction) з тексту повідомлення, зокрема типи відсутностей та дати. Алгоритм має коректно обробляти відносні часові конструкції, такі як "завтра", "наступного тижня", "з понеділка по середу".
- У випадку неповноти даних у запиті (наприклад, користувач вказав "хочу відпустку", але не вказав дати), система повинна ініціювати уточнюючий діалог, запитуючи відсутню інформацію.
- Перед виконанням дії система повинна генерувати підтвердження природною мовою, описуючи параметри створеної заявки для верифікації користувачем.

Модуль візуалізації та моніторингу:

- Система повинна відображати особистий календар користувача зі статусами.
- Керівники повинні мати доступ до зведеного календаря команди для візуалізації перетинів відпусток співробітників та запобігання дефіциту ресурсів.

– Адміністративна панель повинна надавати статистику використання відпусток по відділах та типах за обраний період.

Реалізація повного спектру зазначених вимог дозволить створити цілісну систему, яка не лише автоматизує рутинні операції кадрового обліку, але й суттєво покращить досвід користувачів завдяки впровадженню інтелектуального помічника.

3.2 Проектування бази даних

Проектування схеми бази даних є етапом, що визначає ефективність роботи всієї інформаційної системи, особливо в частині швидкості виконання пошукових запитів та забезпечення цілісності даних. Для системи TimelyMind було обрано реляційну модель даних, реалізовану в СУБД PostgreSQL. Структура бази даних приведена до третьої нормальної форми (3NF), що дозволяє уникнути надлишковості інформації та аномалій при оновленні записів [21]. В основі спроектованої схеми лежить п'ять ключових таблиць, зв'язки між якими забезпечують відображення організаційної структури компанії та логіки процесу погодження відсутностей.

Для моделювання бізнес-процесу погодження заявок на відсутність було розроблено діаграму станів, яка візуалізує життєвий цикл запиту в системі. Діаграма демонструє можливі стани заявки (наприклад, Створено, На погодженні, Затверджено, Скасовано) та переходи між ними, які ініціюються діями різних акторів (працівника, менеджера, HR-адміністратора) або системними подіями. Детальна схема переходів та умов зміни станів для сутності "Запит на відсутність" зображена на рисунку 3.2.

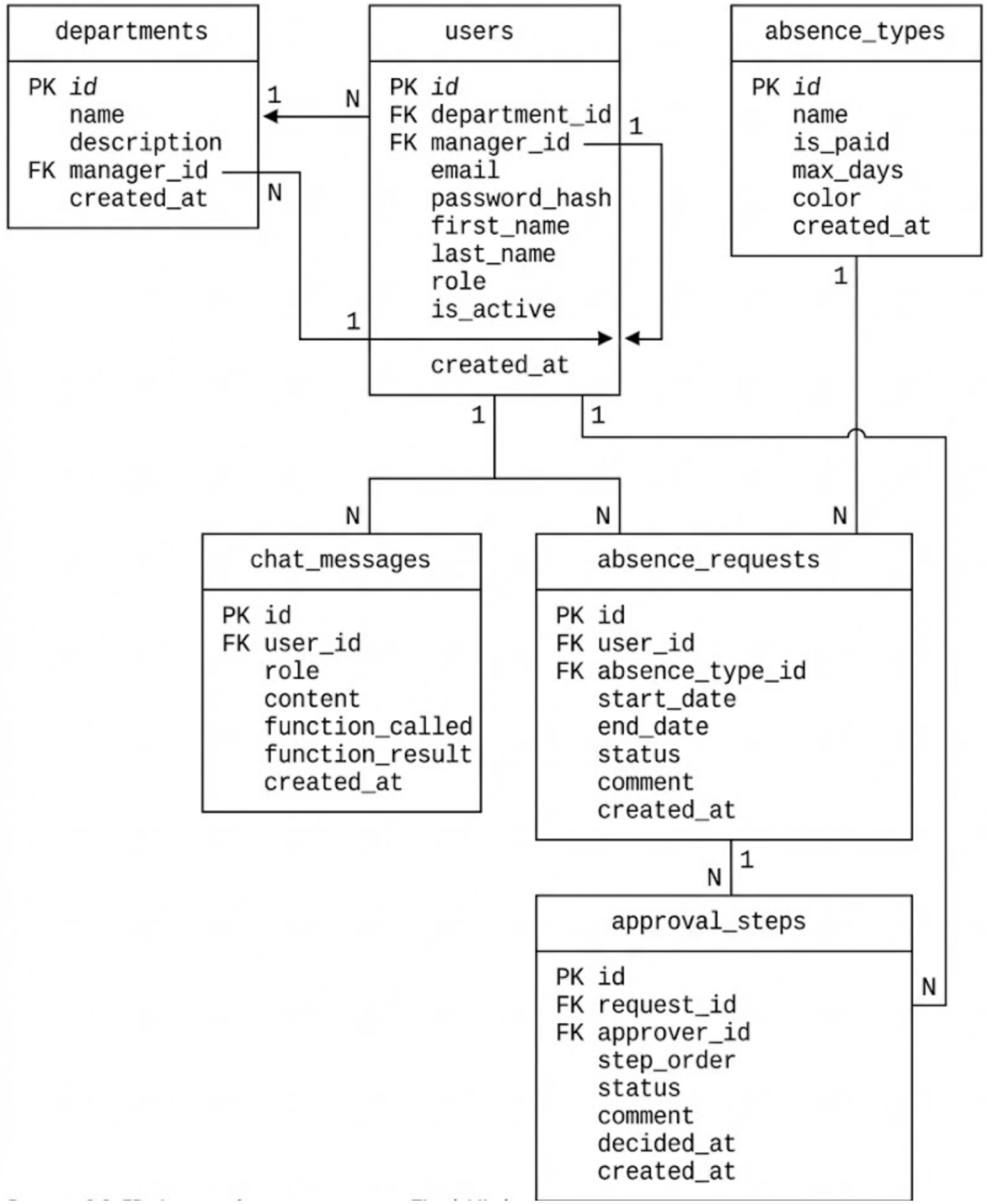


Рисунок 3.2 – ER-діаграма бази даних

Центральним елементом архітектури даних є таблиця users, яка зберігає облікові записи всіх учасників системи. Окрім стандартних атрибутів автентифікації, таких як унікальна електронна пошта та хеш пароля, таблиця містить поле role перелічуваного типу (Enum), що визначає права доступу EMPLOYEE, MANAGER, HR, ADMIN. Важливою особливістю цієї сутності є

реалізація ієрархічних зв'язків: поле manager_id є рекурсивним зовнішнім ключем, що посилається на цю ж таблицю. Це дозволяє будувати дерево підпорядкування співробітників, необхідне для автоматичної маршрутизації заявок [22]. Групування користувачів за структурними підрозділами реалізовано через зв'язок з таблицею departments, яка виконує роль довідника відділів компанії.

Для забезпечення гнучкості бізнес-логіки параметри обліку часу винесено у окрему таблицю absence_types. Вона дозволяє адміністраторам налаштовувати різні категорії відсутностей (лікарняні, відпустки, відгули) без втручання у програмний код. Атрибути цієї сутності включають прапорець оплати (is_paid), річний ліміт днів (max_days_per_year) та колір для візуалізації в інтерфейсі (color). Цілісність даних забезпечується обмеженням унікальності на назву типу, а також забороною видалення типів, які вже використовуються у існуючих заявках (стратегія для зовнішнього ключа).

Безпосередній облік робочого часу ведеться у таблиці absence_requests, де фіксуються дати початку та кінця періоду, поточний статус заявки та посилання на ініціатора. Варто зазначити, що для цієї таблиці налаштовано каскадне видалення записів при видаленні користувача, що спрощує адміністрування системи. Процес погодження заявки, який може включати декілька етапів, моделюється через таблицю approval_steps. Вона реалізує зв'язок "один-до-багатьох" із таблицею заявок, дозволяючи зберігати історію рішень кожного керівника (затвердження або відхилення), їхні коментарі та часові мітки прийняття рішень. Така структура дозволяє системі підтримувати складні ланцюжки workflow та проводити аудит процесу погодження. У всіх таблицях передбачено автоматичне ведення службових полів created_at та updated_at для відстеження хронології змін..

3.3 Проектування та реалізація REST API

Програмний інтерфейс додатку виступає ключовою сполучною ланкою між клієнтською частиною, базою даних та зовнішніми сервісами штучного інтелекту. Реалізація серверної логіки системи TimelyMind виконана відповідно до архітектурного стилю REST, що передбачає роботу з ресурсами через стандартні

методи протоколу HTTP. Для забезпечення високої продуктивності та асинхронної обробки запитів використано фреймворк FastAPI, який дозволяє ефективно реалізувати механізми конкурентності, необхідні для одночасного обслуговування великої кількості користувачів та тривалої комунікації з мовною моделлю [23].

Організація маршрутів програмного інтерфейсу побудована за модульним принципом з використанням механізму маршрутизаторів. Це дозволило логічно розділити точки доступу на групи відповідно до предметних областей: автентифікація, управління користувачами, довідники відділів, запити на відсутність та процес погодження. Усі маршрути згруповані під єдиним префіксом версійності, що забезпечує можливість подальшого розвитку системи без порушення сумісності з існуючими клієнтами. Наприклад, операції з обліковими записами доступні за адресою /users, а робота із заявками здійснюється через шлях /requests.

Критично важливим аспектом реалізації є валідація вхідних даних. Для цього застосовано бібліотеку Pydantic версії 2, яка забезпечує перевірку типів та обмежень на рівні схем даних. Розроблена архітектура чітко розмежовує моделі бази даних SQLAlchemy та схеми передачі даних Pydantic. Такий підхід, відомий як патерн Data Transfer Object, дозволяє контролювати, які саме поля доступні для читання або запису на кожному етапі обробки [24]. Наприклад, при створенні запиту на відпустку клієнт надсилає лише дати та тип відсутності, тоді як система автоматично доповнює об'єкт ідентифікатором користувача та початковим статусом, запобігаючи спробам маніпуляції даними.

Система безпеки API базується на використанні стандарту OAuth2 та JSON Web Tokens. Процес автентифікації реалізовано через спеціальний маршрут, який приймає облікові дані користувача та у разі успішної перевірки повертає підписаний токен доступу. Для захисту приватних маршрутів використано механізм впровадження залежностей. Кожен захищений метод API декларує залежність від функції отримання поточного користувача, яка автоматично вилучає токен із заголовка запиту, перевіряє його валідність та завантажує контекст користувача з бази даних. Це дозволяє реалізувати гнучку систему контролю

доступу на основі ролей, де певні операції, наприклад затвердження заявок, доступні виключно користувачам із правами менеджера або адміністратора.

Обробка помилок у системі стандартизована з використанням відповідних кодів стану HTTP. Успішні операції створення ресурсів супроводжуються кодом 201, запити на отримання даних повертають код 200. У випадку порушення бізнес-правил, таких як спроба взяти відпустку понад ліміт або перетин дат із існуючою заявкою, сервер генерує виняток, який трансформується у відповідь з кодом 400 або 422 та детальним описом проблеми у форматі JSON. Такий підхід спрощує налагодження клієнтського додатку та забезпечує прогнозовану поведінку системи у нештатних ситуаціях. Автоматична генерація документації у форматі OpenAPI дозволяє розробникам клієнтської частини та тестувальникам ефективно взаємодіяти з API без необхідності ручного вивчення вихідного коду серверної частини.

3.4 Модуль обробки природної мови та інтеграція з API

Інтелектуальне ядро системи TimelyMind базується на модулі обробки природної мови, який забезпечує трансформацію неструктурованих текстових запитів користувача у структуровані команди для виконання бізнес-логіки. Для реалізації цього функціоналу обрано велику мовну модель Google Gemini, інтеграція з якою здійснюється через відповідний програмний інтерфейс [25]. Вибір саме цієї моделі зумовлений її високою здатністю до розуміння контексту, підтримкою багатомовності та оптимізацією для виконання інструкцій, що є критичним для коректного розпізнавання намірів у специфічній предметній області управління персоналом.

Архітектурно модуль реалізовано як окремий сервісний шар, який інкапсулює логіку взаємодії із зовнішнім API. Процес обробки запиту розпочинається з формування контексту, який включає системну інструкцію, поточну дату та історію діалогу. Системна інструкція відіграє роль "налаштування" поведінки моделі, чітко визначаючи її роль як асистента з управління відсутностями. У ній закладено правила інтерпретації часових проміжків, перелік

допустимих типів відпусток та формат вихідних даних. Особливу увагу приділено мінімізації ризику галюцинацій моделі шляхом суворого обмеження її функціоналу виключно завданнями класифікації та вилучення сутностей без права генерувати факти, що не містяться у вхідних даних.

Ключовим завданням модуля є розпізнавання наміру користувача. Вхідне повідомлення аналізується на предмет приналежності до одного з попередньо визначених класів: створення запиту, перевірка балансу, скасування заявки або отримання довідкової інформації. Паралельно з визначенням наміру відбувається процес вилучення іменованих сутностей. Система ідентифікує в тексті дати початку та завершення відсутності, тип відпустки та, за потреби, причину. Завдяки семантичним можливостям мовної моделі, модуль успішно опрацьовує складні лінгвістичні конструкції та відносні дати, автоматично конвертуючи вирази на кшталт "наступної п'ятниці" у конкретні календарні дати формату ISO 8601, спираючись на поточний системний час [26].

Для забезпечення надійної програмної обробки результатів генерації використано режим примусового форматування відповіді у формат JSON. Це дозволяє уникнути необхідності складного парсингу текстової відповіді та гарантує, що вихідні дані завжди матимуть очікувану структуру полів. Отриманий від моделі об'єкт проходить додаткову валідацію на стороні сервера: перевіряється коректність дат, існування розпізнаного типу відсутності в базі даних та логічна узгодженість параметрів. У випадку, якщо вхідний запит не містить достатньо інформації для виконання дії, наприклад відсутня дата завершення відпустки, модуль генерує уточнююче запитання, підтримуючи режим діалогу до моменту повного заповнення необхідних параметрів.

Взаємодія з зовнішнім API є операцією з потенційно високою затримкою, тому її реалізовано в асинхронному режимі. Це дозволяє веб-серверу не блокувати обробку інших запитів під час очікування відповіді від моделі. Для підвищення відмовостоійкості передбачено механізм повторних запитів у разі тимчасової недоступності сервісу або отримання некоректної відповіді. Також реалізовано логування всіх транзакцій взаємодії з моделлю, що дозволяє проводити подальший

аналіз якості розпізнавання та вдосконалювати системні промпти для підвищення точності роботи інтелектуального асистента.

3.5 Реалізація workflow погодження відсутностей

Автоматизація процесу погодження заявок є ключовим функціональним елементом системи, що забезпечує перехід від ручного адміністрування до цифрового управління робочими процесами. Реалізація механізму workflow базується на подієво-орієнтованій моделі, де зміна стану заявки ініціює ланцюжок автоматичних дій, таких як визначення наступного погоджувача, надсилання сповіщень та оновлення балансів [27]. Логіка маршрутизації побудована на основі ієрархічної структури організації, закладеної у базі даних, що дозволяє динамічно визначати відповідальних осіб без необхідності жорсткого кодування прізвищ у конфігурації системи.

Процес ініціюється у момент створення працівником нового запиту на відсутність. Система автоматично аналізує профіль користувача для визначення його безпосереднього керівника та, залежно від типу відпустки та політик компанії, формує список необхідних кроків погодження. Технічно це реалізується шляхом створення записів у таблиці кроків погодження, кожен з яких прив'язаний до конкретного менеджера та має початковий статус очікування. Такий підхід дозволяє реалізовувати як прості одноріневі сценарії, де рішення приймає лише тімлід, так і складні багаторівневі процеси, що вимагають послідовного затвердження керівником департаменту та адміністратором кадрів.

Важливим аспектом реалізації є забезпечення транзакційності операцій. Зміна статусу заявки, запис коментаря керівника та перехід до наступного кроку виконуються в межах однієї атомарної транзакції бази даних. Це гарантує узгодженість даних: неможлива ситуація, коли заявка вважається погодженою, але відповідний запис про рішення керівника не збережено. Після успішного виконання транзакції система ініціює асинхронну задачу для сервісу сповіщень. Використання черги задач Celery дозволяє винести процес відправки електронних

листів та push-повідомлень у фоновий режим, забезпечуючи миттєву реакцію інтерфейсу на дії користувача [28].

Для моделювання бізнес-процесу життєвого циклу заявки на відсутність у системі було розроблено діаграму станів. Вона візуалізує всі можливі етапи, через які проходить запит від моменту його створення до фіналізації. Процес починається зі стану PENDING, де заявка очікує на дії з боку погоджувачів або самого користувача. З цього стану можливі три основні переходи: схвалення, відхилення або скасування користувачем. У разі схвалення система перевіряє, чи отримано всі необхідні погодження; якщо так, заявка переходить у стан APPROVED, якщо ні - повертається до PENDING. Дії відхилення або скасування переводять заявку безпосередньо у відповідні кінцеві стани REJECTED або CANCELLED. Усі три кінцеві стани завершують процес. Детальна схема переходів та умов зміни станів для сутності "Запит на відсутність" зображена на рисунку 3.3.

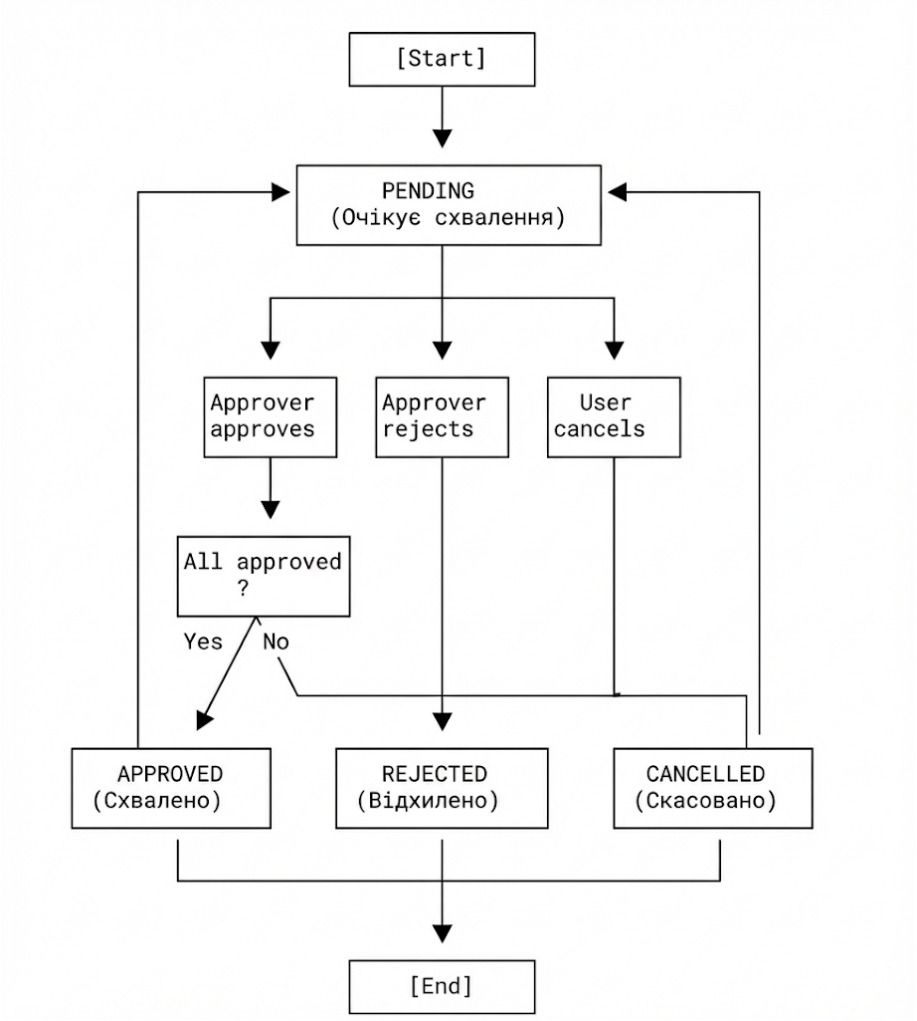


Рисунок 3.3 – Діаграма станів

У системі реалізовано логіку делегування повноважень та обробки виключних ситуацій. Якщо безпосередній керівник сам перебуває у відпустці, система здатна автоматично перенаправити запит на керівника вищого рівня або призначену виконуючу особу. Фіналізація процесу відбувається після отримання позитивних рішень на всіх етапах маршруту. У цей момент статус глобальної заявки змінюється на "Затверджено", а з балансу працівника остаточно списується відповідна кількість днів. Така архітектура забезпечує прозорість процесу прийняття рішень, зберігаючи повний аудиторський слід усіх дій для подальшого аналізу ефективності управління персоналом.

3.6 Реалізація користувацького інтерфейсу

Створення візуальної оболонки системи TimelyMind спрямоване на забезпечення інтуїтивно зрозумілої взаємодії користувачів із функціоналом платформи, мінімізуючи когнітивне навантаження при виконанні рутинних операцій. Клієнтська частина реалізована як односторінковий веб-додаток, що забезпечує миттєву реакцію на дії користувача без необхідності повного перезавантаження сторінки. Архітектура інтерфейсу базується на компонентному підході, де кожен елемент, від кнопки до складного віджету календаря, є ізольованим модулем з власним станом та логікою відображення [29].

Реалізація програмного комплексу TimelyMind виконана з дотриманням принципів модульності та чіткого розмежування відповідальності (Separation of Concerns). Проект організовано у вигляді монорепозиторію, структура якого дозволяє одночасно розробляти та підтримувати як серверну, так і клієнтську частини системи.

У кореневій директорії розміщено файли конфігурації середовища та оркестрації контейнерів (docker-compose.yml), що забезпечує уніфікований запуск усіх сервісів. Серверна частина (backend) побудована на базі фреймворку FastAPI і має чітку ієрархію: окремі пакети для конфігурації (core), моделей бази даних (models), бізнес-логіки (services) та API-маршрутів (api). Клієнтська частина (frontend) містить вихідний код React-додатку, включаючи компоненти інтерфейсу,

контексти станів та сторінки. Повна файлова структура розробленого проекту наведена на рисунку 3.4.

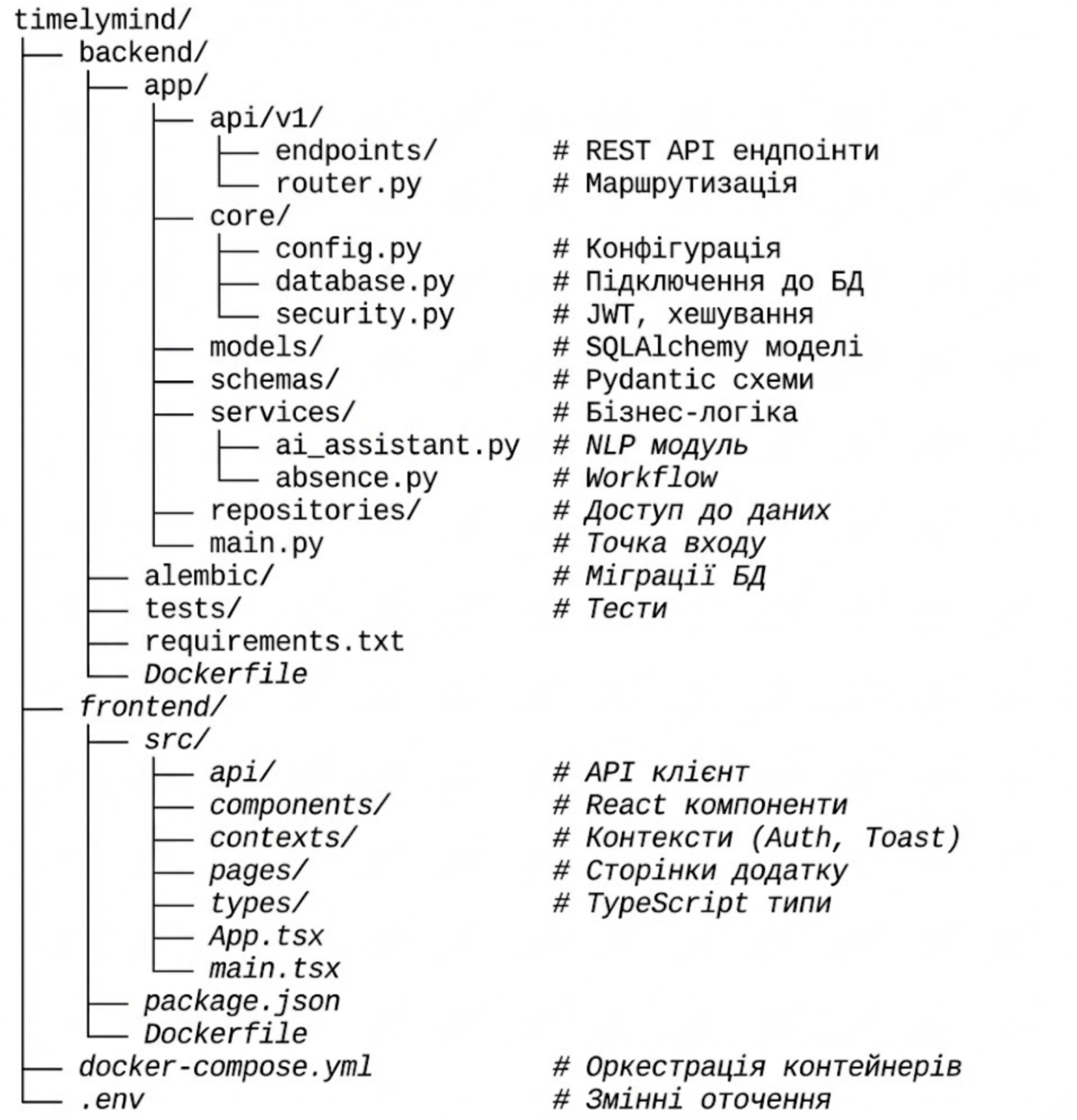


Рисунок 3.4 – Файлова структура додатку

Першою точкою взаємодії користувача із системою є сторінка автентифікації. Розроблений інтерфейс входу, зображений на рисунку 3.5, має лаконічний дизайн, розділений на дві функціональні зони. Ліва частина містить брендинг та короткий опис переваг системи, а права - форму для введення

облікових даних (електронної пошти та пароля). Реалізована клієнтська валідація полів забезпечує миттєвий зворотний зв'язок у разі введення некоректного формату даних, а кнопка "Sign in" ініціює запит до сервера для отримання JWT-токена доступу.

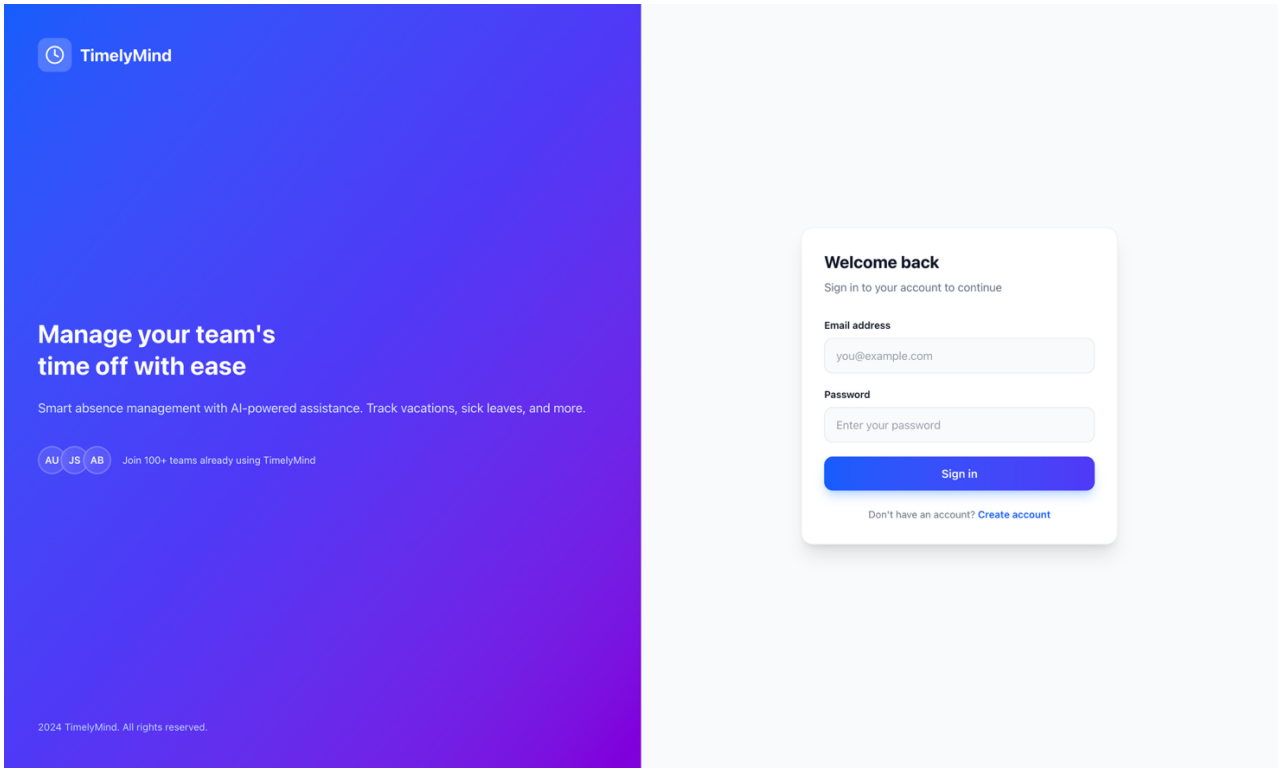


Рисунок 3.5 – Сторінка автентифікації користувача

Для нових користувачів передбачена процедура створення облікового запису. Форма реєстрації, наведена на рисунку 3.6, вимагає введення базової персональної інформації: імені, прізвища, робочої електронної пошти та пароля з підтвердженням. Система автоматично перевіряє складність пароля та збіг введених значень у полях паролів. Крім того, на сторінці акцентується увага на ключових функціях системи - AI-чаті та календарі, що сприяє швидшому залученню нових користувачів до роботи з продуктом.

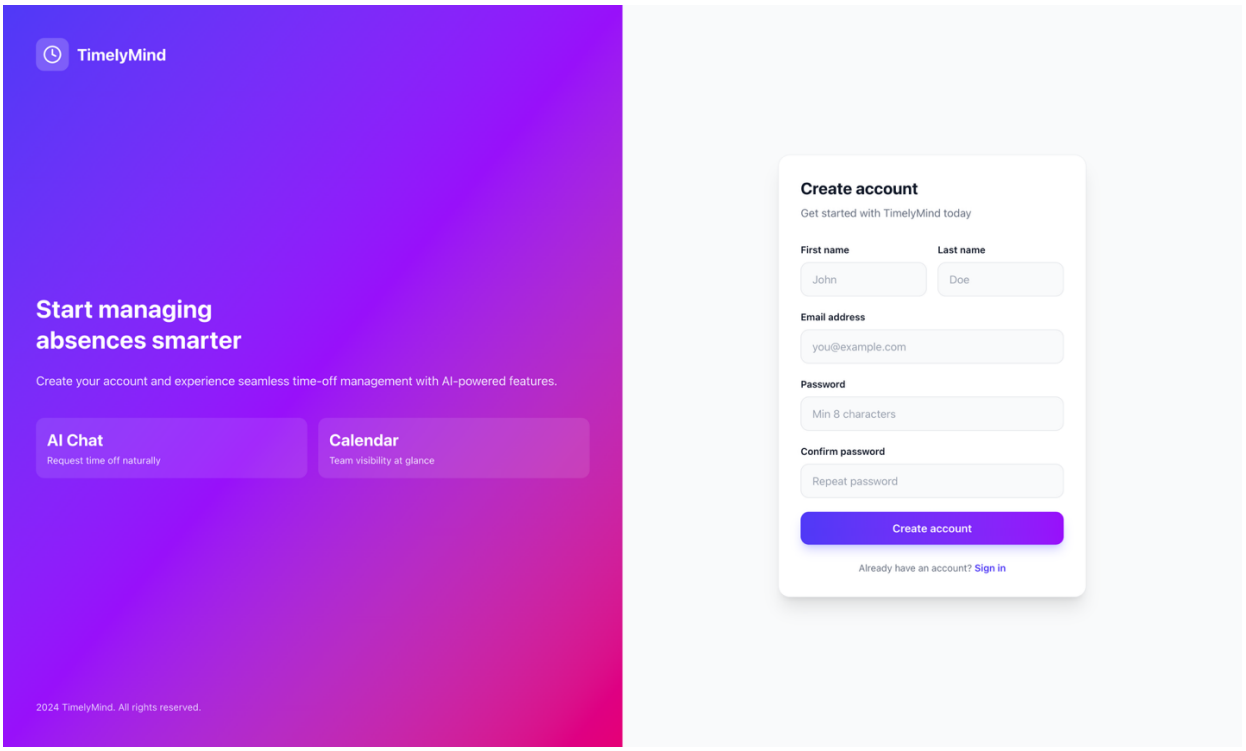


Рисунок 3.6 – Сторінка реєстрації нового користувача

Основним структурним елементом додатку є макет сторінки (Layout), який містить постійну навігаційну панель та динамічну область контенту. Навігація адаптується до ролі поточного користувача: працівники бачать лише розділи власного профілю та календаря, тоді як менеджери отримують доступ до панелі керування командою та журналу погоджень. Головна сторінка, спроектована у вигляді інформаційної панелі (Dashboard), надає користувачу миттєвий доступ до критично важливої інформації: поточного балансу відпусток за категоріями, статусу останніх заявок та найближчих запланованих відсутностей колег (див. рис. 3.7). Для візуалізації статистичних даних використано графічні індикатори прогресу та кольорове кодування, що дозволяє оцінити доступні ресурси за лічені секунди.

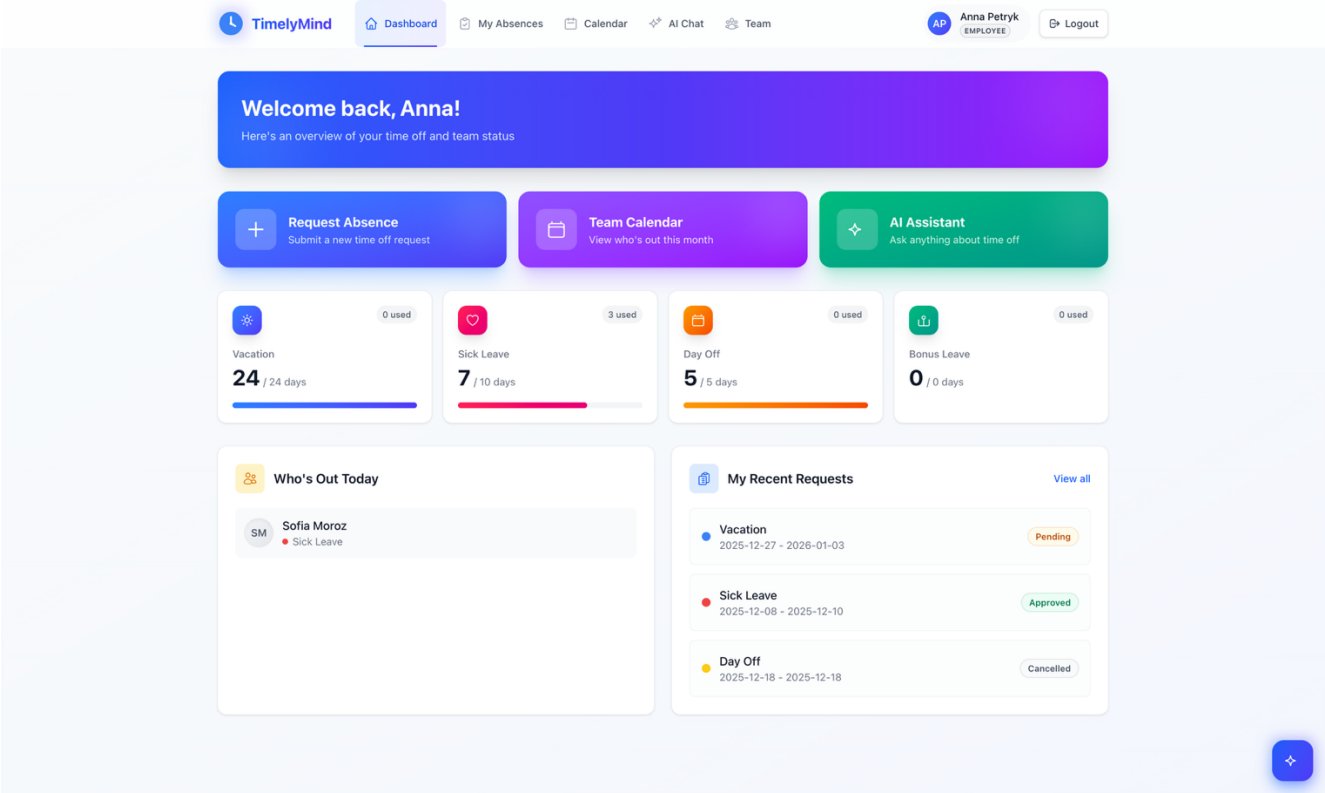


Рисунок 3.7 – Головна інформаційна панель (Dashboard) користувача

Центральним елементом візуалізації часових даних є модуль календаря. Він підтримує перемикання між місячним та тижневим видами, відображаючи відсутності у вигляді кольорових блоків. Колірна схема синхронізується з налаштуваннями бази даних, що дозволяє користувачам швидко розрізняти лікарняні, відпустки та відгули. Для менеджерів реалізовано зведений вигляд календаря, що дозволяє бачити графік усієї команди на одній сітці, полегшуючи процес планування проектів та виявлення потенційних конфліктів ресурсів (див. рис. 3.8). Взаємодія з календарем є інтерактивною: натискання на слот дати ініціює створення нової заявки, а клік на існуючий запис відкриває детальну інформацію про подію.

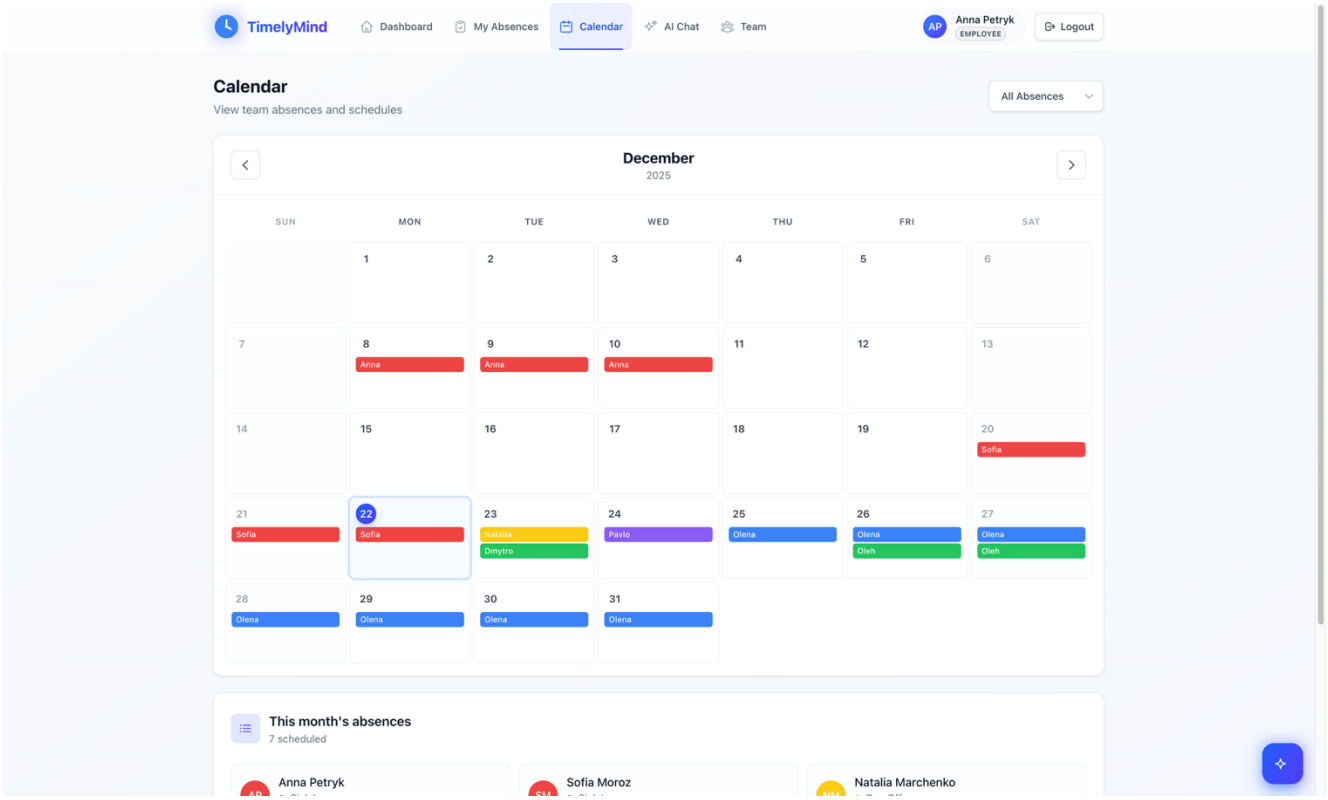


Рисунок 3.8 – Інтерфейс календаря відсутностей команди

Окрему увагу приділено реалізації інтерфейсу інтелектуального асистента. Чат-компонент інтегровано у систему у вигляді плаваючого віджету, доступного з будь-якої сторінки додатку, або як окремий повноцінний розділ (див. рис. 3.9). Візуально він наслідує звичні патерни сучасних месенджерів: історія повідомлень відображається у хронологічному порядку, повідомлення користувача та системи розрізняються за розташуванням та стилем оформлення. Для покращення користувацького досвіду реалізовано індикацію процесу "друкування" під час очікування відповіді від сервера, а також візуалізацію структурованих даних. Якщо система повертає інформацію про створену заявку, вона відображається не просто як текст, а як інтерактивна картка з кнопками підтвердження або редагування.

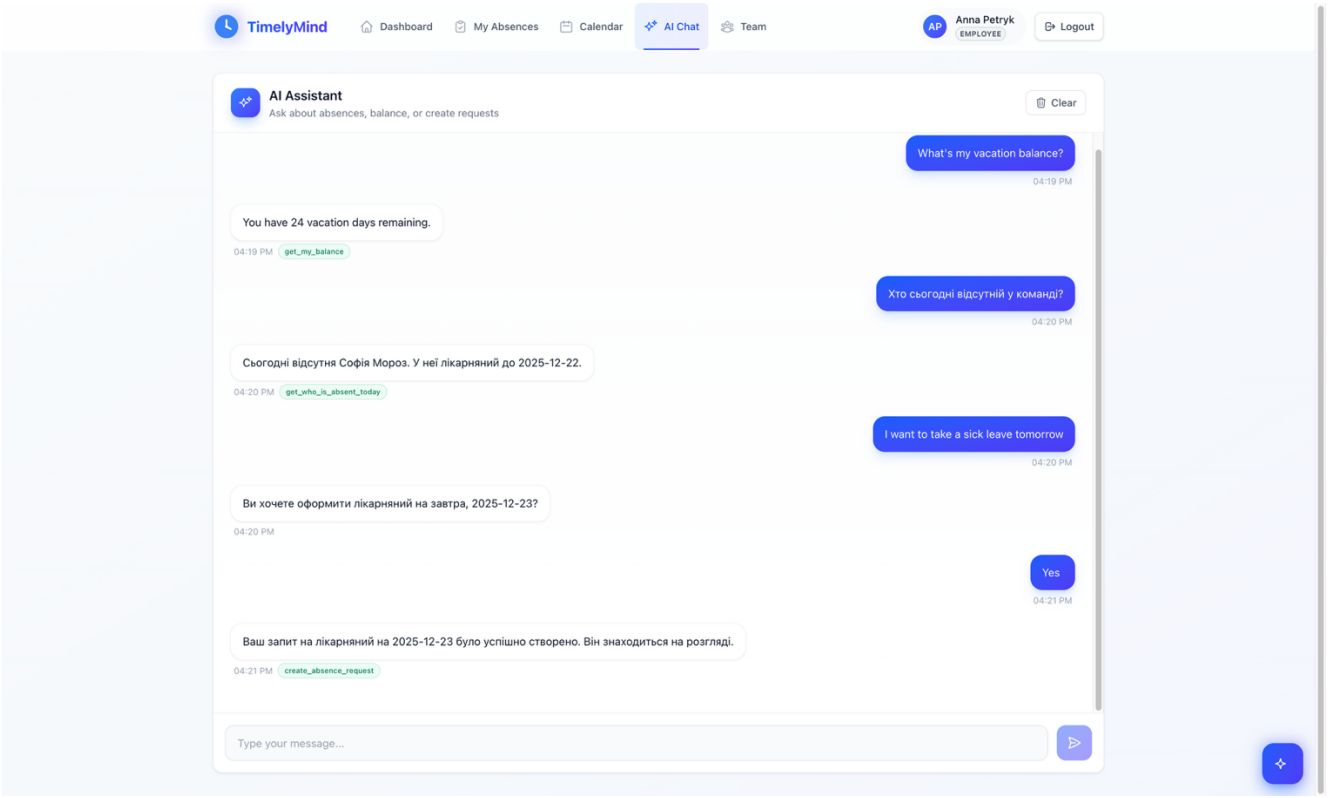


Рисунок 3.9 – Інтерфейс діалогу з AI-асистентом

Для забезпечення безперервного доступу до інтелектуальних функцій системи, незалежно від поточного контексту роботи користувача, розроблено плаваючий віджет AI-асистента. Цей компонент, зображений на рисунку 3.10, доступний у глобальній навігації та дозволяє взаємодіяти з системою без переходу на окрему сторінку чату. Інтерфейс віджету містить набір "швидких пропозицій" (Quick suggestions) для виконання найпопулярніших дій в один клік, таких як перевірка балансу відпустки або перегляд відсутніх колег. Також передбачено поле для введення довільних запитів природною мовою, що дозволяє користувачу вирішувати термінові питання, не відволікаючись від основного робочого процесу.

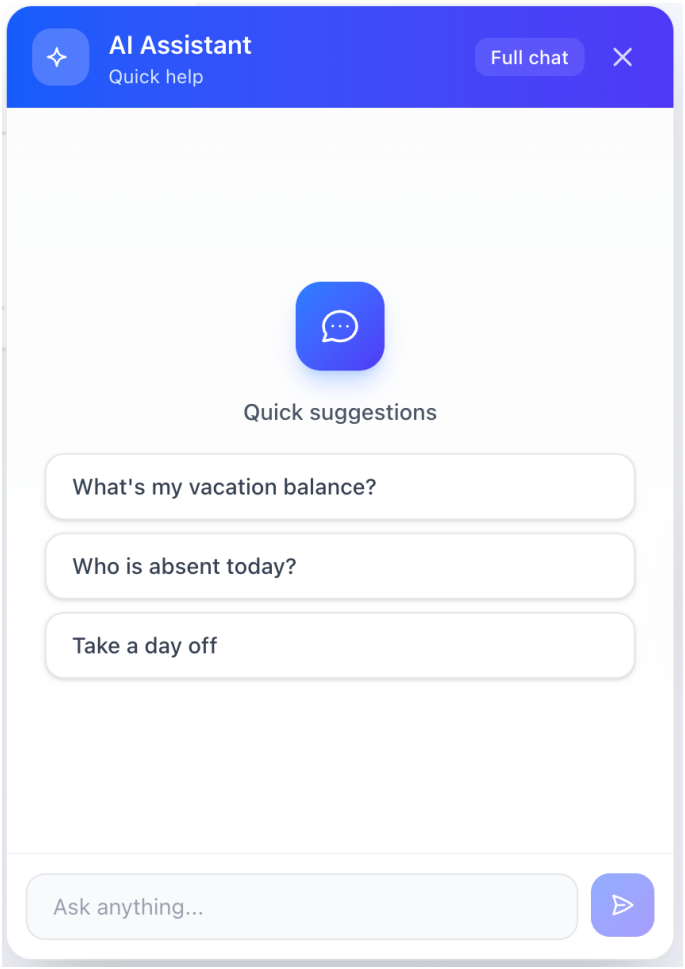


Рисунок 3.10 – Віджет швидкого доступу до AI-асистента

Для контролю за станом власних запитів та перегляду історії відпусток призначено розділ "My Absences". Інтерфейс сторінки, зображений на рисунку 3.11, реалізовано у вигляді структурованої таблиці, яка надає користувачу повну інформацію про кожну подію: тип відсутності (Vacation, Sick Leave, Day Off), обраний часовий проміжок та поточний статус погодження. Система візуально розрізняє статуси за допомогою кольорових індикаторів: помаранчевий для очікування ("pending"), зелений для затверджених ("approved") та сірий для скасованих ("cancelled") заявок. Окрім перегляду, користувач має можливість керувати активними заявками - кнопка "Cancel" дозволяє відкликати запит, який ще не пройшов процедуру фінального затвердження.

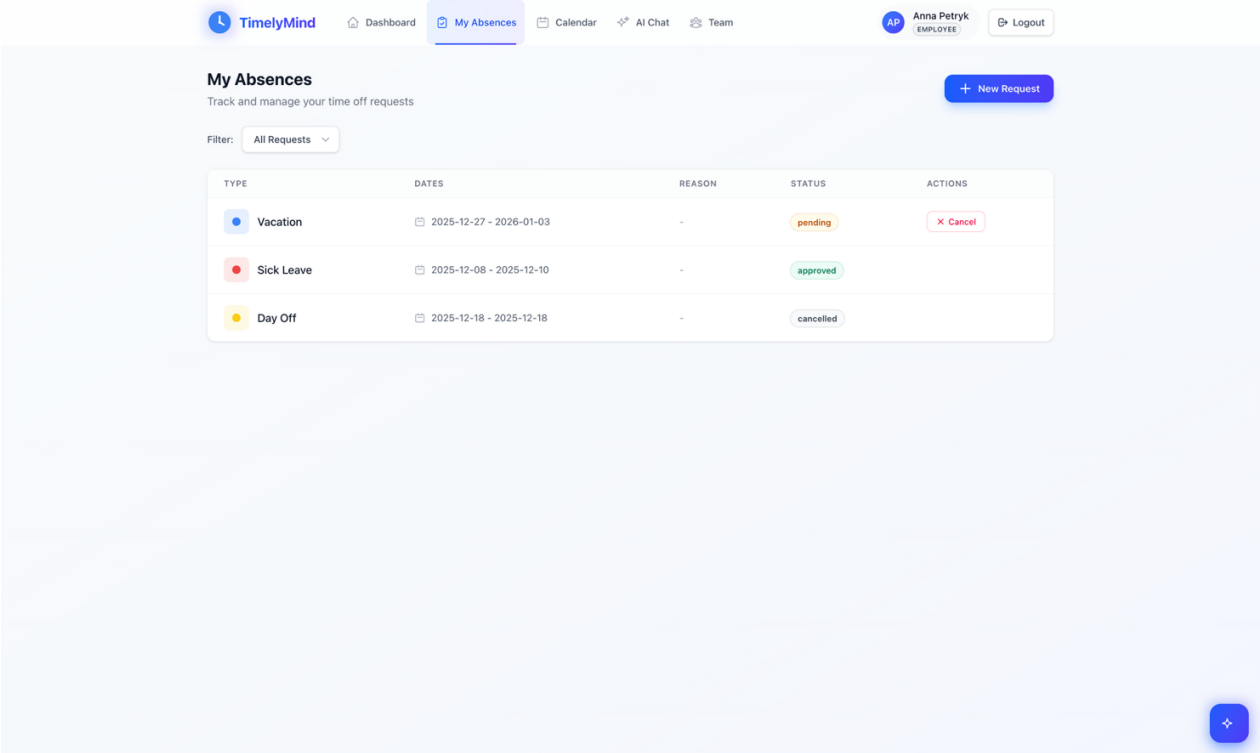


Рисунок 3.11 – Інтерфейс керування особистими заявками на відсутність

Для ініціювання процесу відсутності розроблено спеціалізовану форму, яка відкривається при створенні нового запиту. Інтерфейс, зображений на рисунку 3.12, спроєктовано максимально лаконічно, щоб мінімізувати час на заповнення. Користувачеві необхідно обрати тип відсутності (наприклад, щорічна відпустка або лікарняний) та вказати бажаний діапазон дат у календарі. Також передбачено необов'язкове текстове поле "Reason", де працівник може залишити коментар для свого керівника, пояснюючи причину запиту. Після натискання кнопки "Submit Request" система виконує валідацію введених даних і, у разі успіху, створює запит та надсилає сповіщення відповідним погоджувачам.

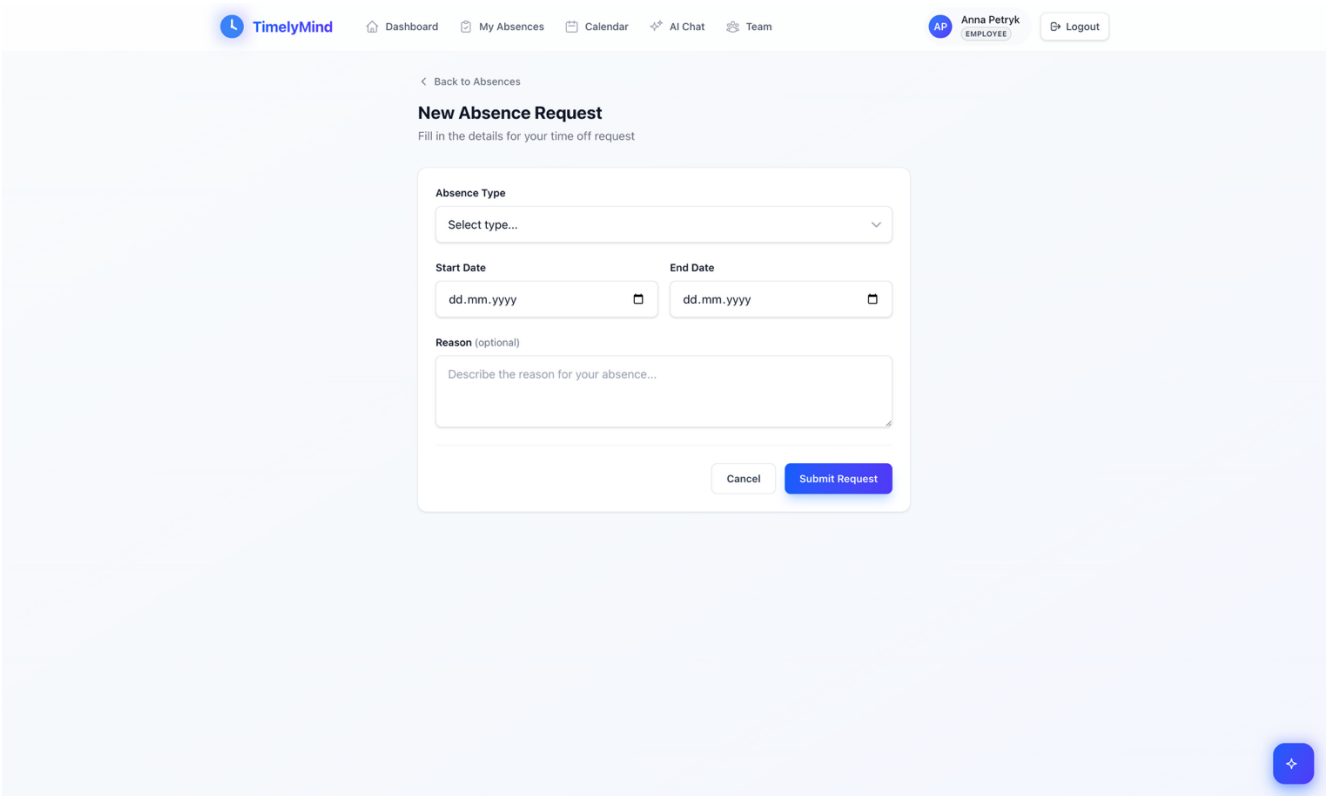


Рисунок 3.12 – Форма створення нової заявки на відсутність

Для забезпечення прозорості організаційної структури та спрощення комунікації всередині підрозділу реалізовано вкладку "My Team". Інтерфейс, представлений на рисунку 3.13, відображає список усіх співробітників поточного департаменту у форматі зручних інформаційних карток. Кожна картка містить візуальний аватар з ініціалами, повне ім'я колеги, корпоративну електронну адресу та позначку ролі в системі (наприклад, "manager" або "employee"). Такий функціонал дозволяє користувачам швидко ідентифікувати склад своєї команди та знаходити необхідні контакти без використання сторонніх довідників.

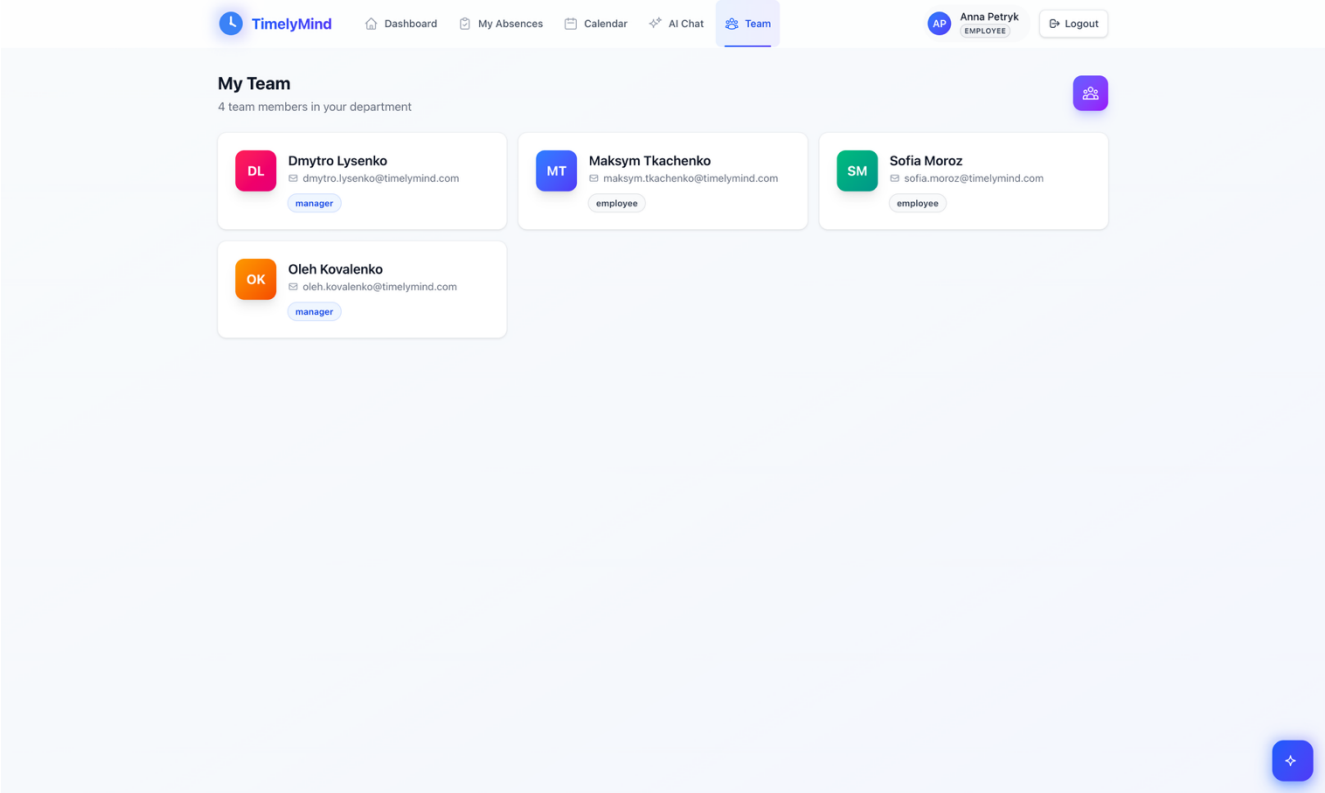


Рисунок 3.13 – Інтерфейс перегляду складу команди

Важливою складовою бізнес-процесу є механізм затвердження заявок, доступ до якого обмежено виключно користувачам із ролями Керівника (Manager) та Адміністратора. Вкладка "Approvals", інтерфейс якої наведено на рисунку 3.14, консолідує всі вхідні запити від підлеглих співробітників, що очікують на розгляд. Кожна заявка відображається у вигляді окремої картки, що містить ключові дані для прийняття рішення: ім'я ініціатора, тип відсутності (з відповідним кольоровим маркуванням), запитуваний період та дату подання. Функціонал сторінки дозволяє керівнику оперативно обробляти чергу запитів, використовуючи кнопки швидкої дії "Approve" (Затвердити) або "Reject" (Відхилити) безпосередньо у списку.

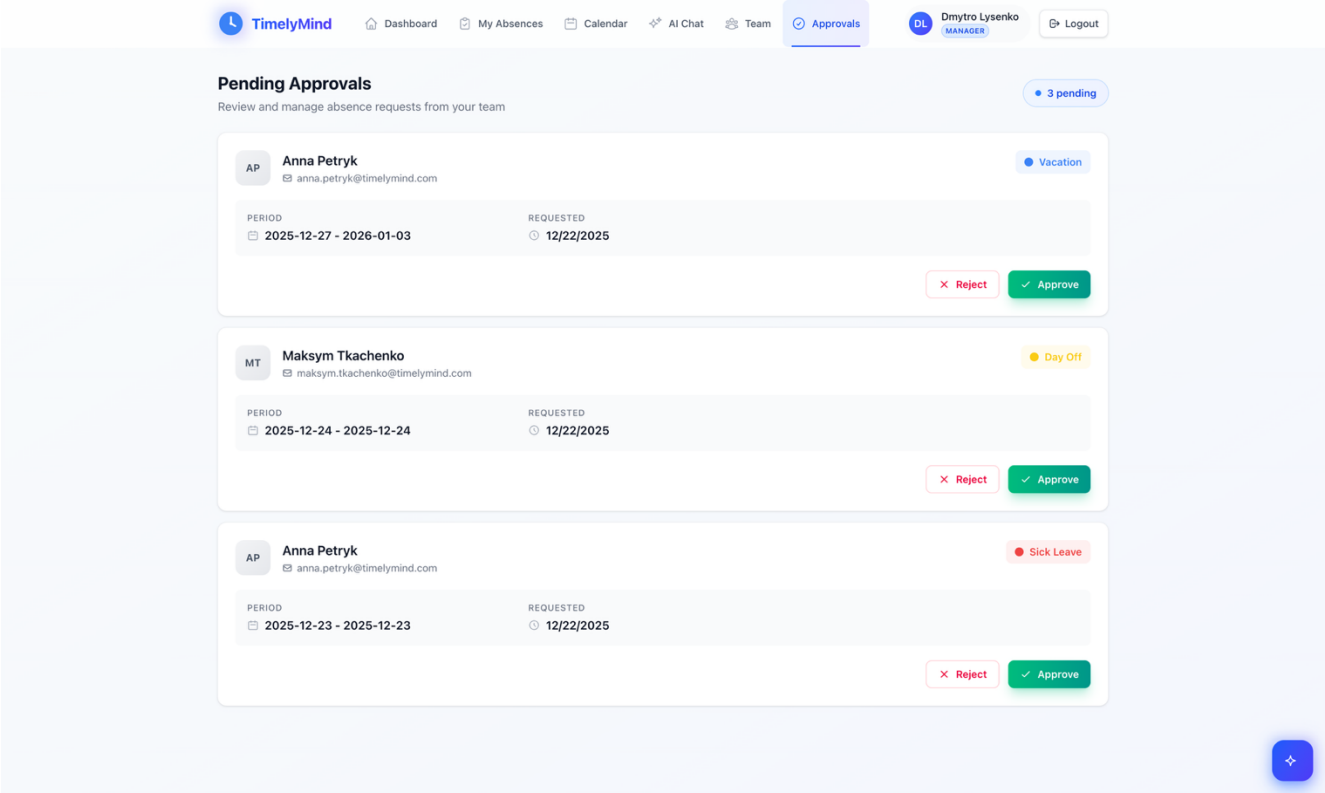


Рисунок 3.14 – Інтерфейс панелі затвердження запитів керівником

Для повного контролю за функціонуванням системи та управлінням організаційною структурою компанії розроблено модуль адміністратора. Головна сторінка адміністративної панелі ("Admin Dashboard"), зображена на рисунку 3.15, надає глобальну статистику по системі. У верхній частині розташовані віджети з ключовими метриками: загальна кількість користувачів, сумарна кількість заявок, кількість активних запитів (Pending) та співвідношення схвалених/відхилених заявок.

Нижче розташовано секцію "All Requests", яка, на відміну від менеджерського інтерфейсу, відображає абсолютно всі запити в компанії незалежно від департаменту. Це дозволяє адміністратору або HR-спеціалісту моніторити загальну ситуацію з відвідуваністю, а за необхідності - втрутитися в процес та виконати примусове затвердження або відхилення будь-якої заявки, використовуючи відповідні елементи керування в таблиці.

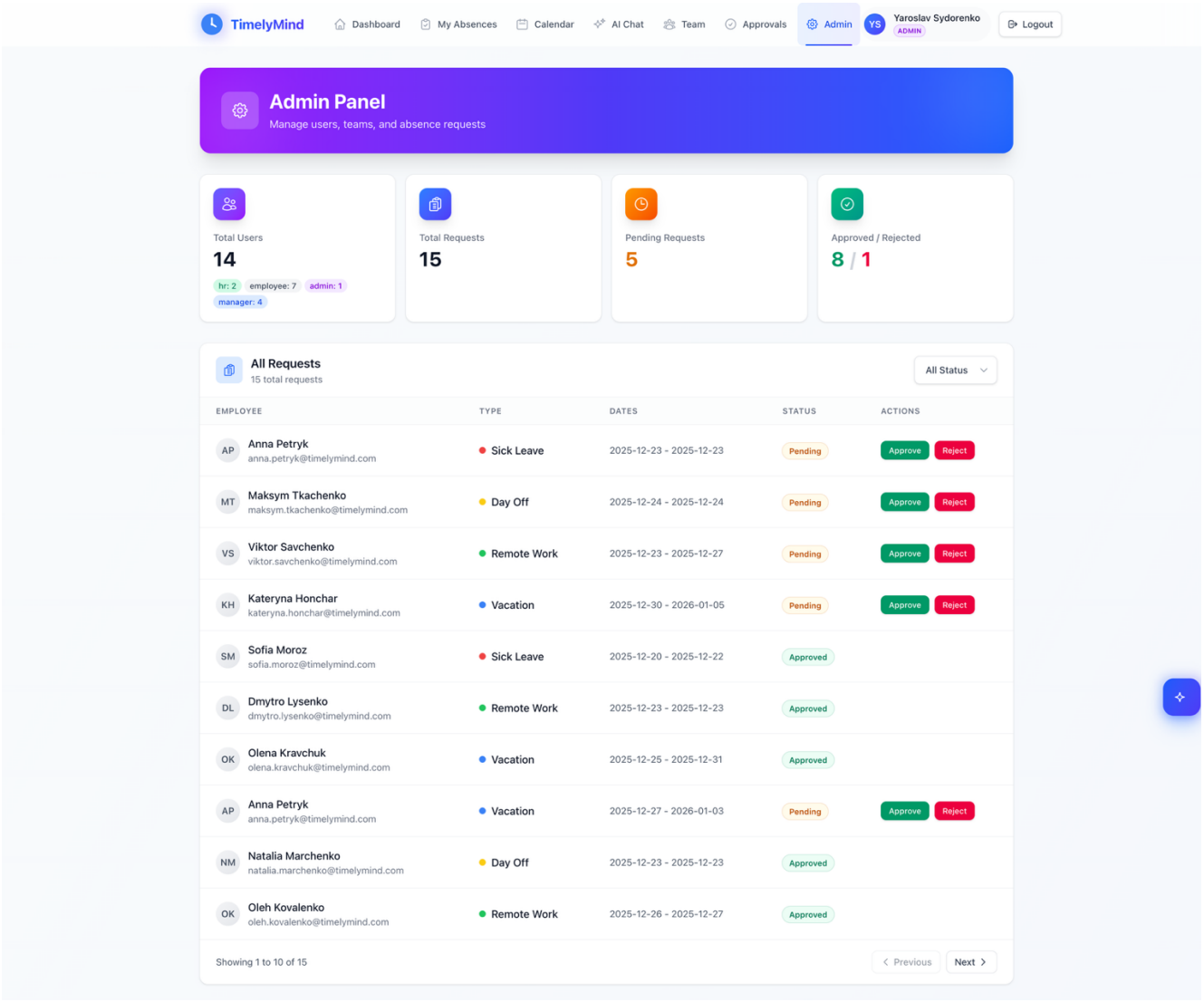


Рисунок 3.15 – Головна панель адміністратора та глобальний реєстр заявок

Критично важливим аспектом безпеки та розподілу прав доступу є підсистема керування користувачами ("Users Management"). Інтерфейс цього розділу, наведений на рисунку 3.16, дозволяє адміністратору переглядати повний список співробітників із зазначенням їхніх ролей (Employee, Manager, Admin, HR) та приналежності до команд.

Функціонал сторінки передбачає можливість редагування профілів користувачів, зміни їхніх ролей, що динамічно впливає на доступний їм функціонал у системі (RBAC), а також деактивацію облікових записів звільнених працівників. Табличне подання даних із кольоровим кодуванням статусів та ролей забезпечує

зручну навігацію та швидкий пошук необхідного співробітника для внесення адміністративних змін.

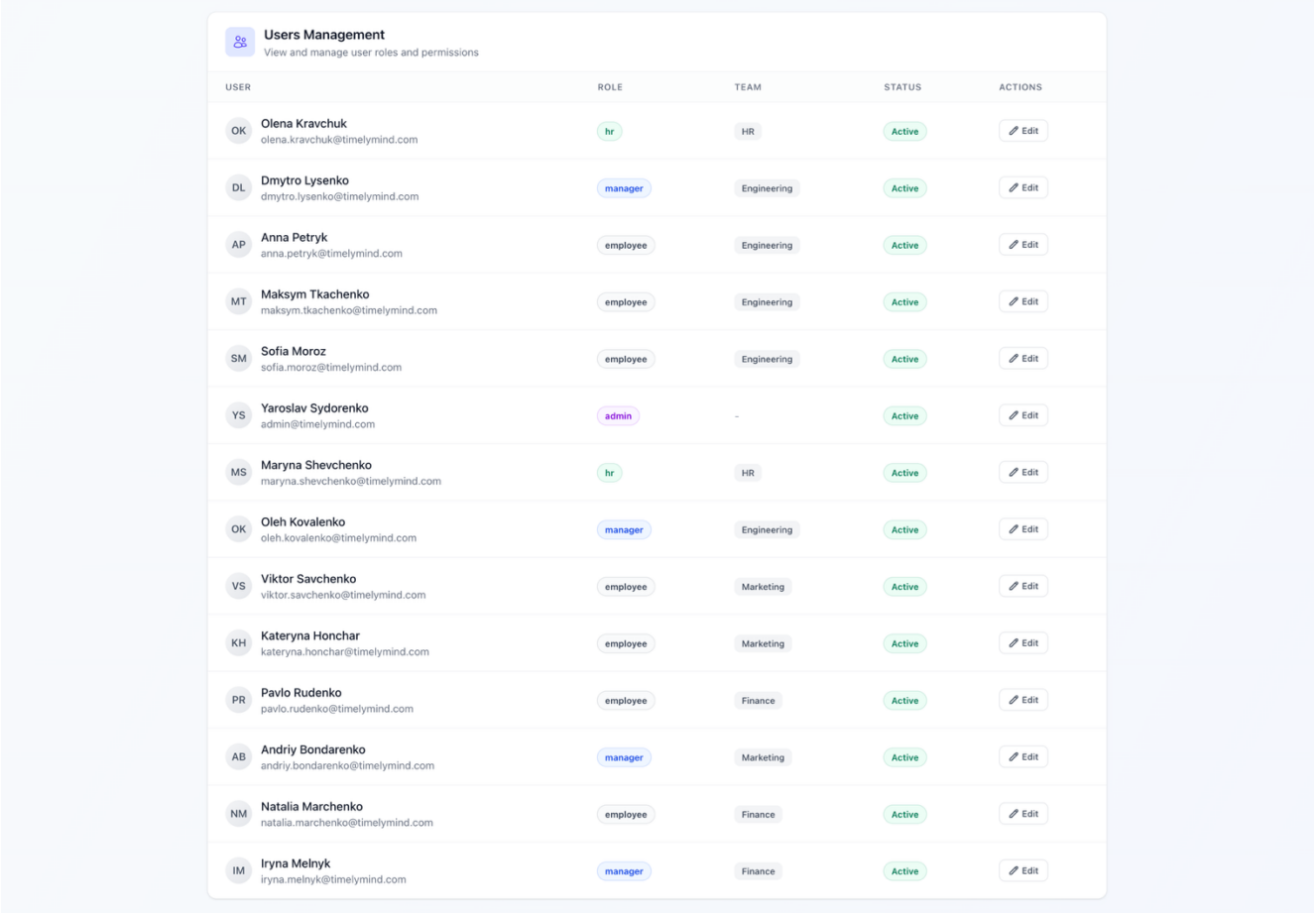


Рисунок 3.16 – Інтерфейс керування обліковими записами та ролями користувачів

Для відображення ієрархічної структури компанії реалізовано розділ управління командами ("Teams Management"). Як показано на рисунку 3.17, система дозволяє створювати окремі підрозділи (наприклад, Engineering, HR, Marketing) та призначати для них відповідальних менеджерів.

Кожна картка команди містить інформацію про назву відділу, його опис та призначених керівників (Manager 1, Manager 2). Саме на основі цих налаштувань працює алгоритм маршрутизації заявок: коли працівник створює запит, система автоматично перевіряє, до якої команди він належить, і надсилає сповіщення саме тим менеджерам, які закріплені за цією командою в даному розділі. Інтерфейс

також дозволяє редагувати склад керівництва або видаляти застарілі організаційні одиниці.

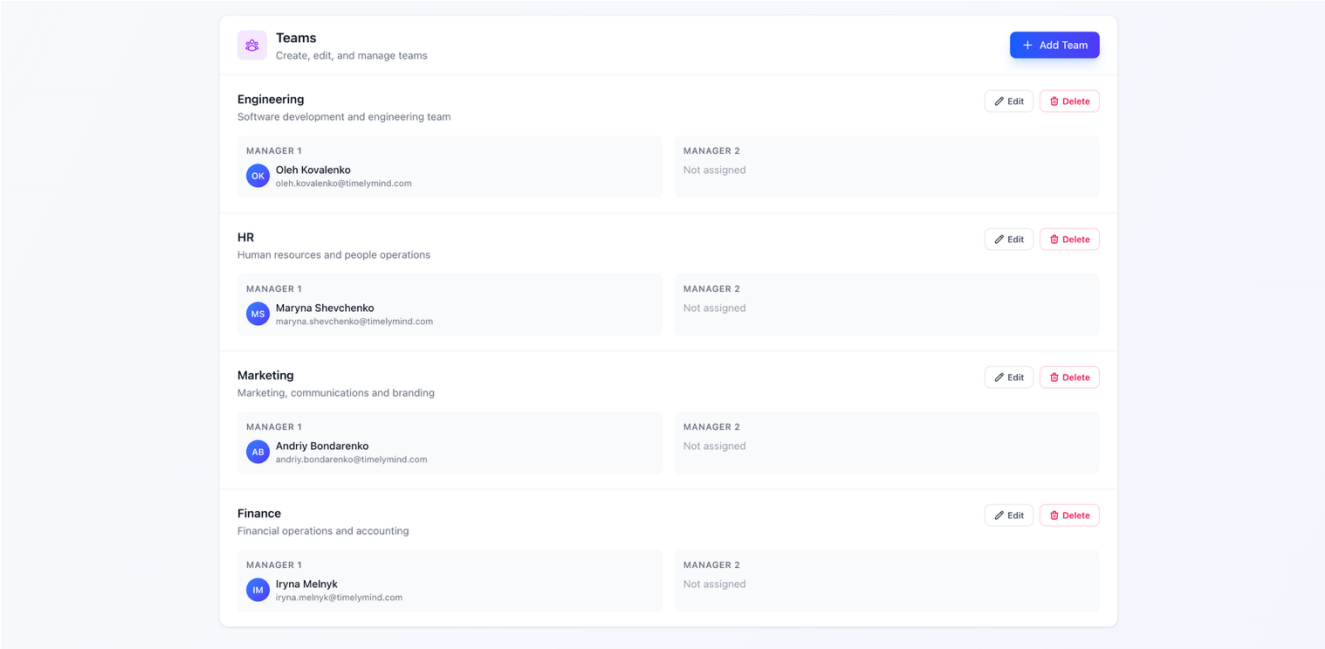


Рисунок 3.17 – Інтерфейс налаштування організаційної структури та команд

Стилізація інтерфейсу виконана з використанням підходу utility-first, що забезпечує єдність дизайну та адаптивність під різні розміри екранів [30]. Система автоматично перебудовує розташування елементів для коректного відображення на мобільних пристроях, трансформуючи таблиці у списки карток та приховуючи другорядні елементи навігації у випадаюче меню. Взаємодія з сервером здійснюється через асинхронні запити, стан виконання яких (завантаження, успіх, помилка) транслюється користувачу через систему спливаючих сповіщень (Toasts) та скелетних екранів завантаження, що робить роботу з системою плавною та передбачуваною.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

4.1 Верифікація відповідності системи функціональним вимогам

Етап верифікації програмного забезпечення є критично важливою складовою життєвого циклу розробки, метою якого є підтвердження того, що створений продукт повністю відповідає специфікаціям та очікуванням зацікавлених сторін, сформульованим на стадії проектування. Для оцінки якості реалізації системи TimelyMind було проведено комплексне функціональне тестування, яке охоплювало перевірку роботи всіх архітектурних рівнів: від цілісності даних у базі PostgreSQL до коректності відображення елементів інтерфейсу у веб-браузері [31]. Верифікація здійснювалася шляхом моделювання реальних сценаріїв використання для різних ролей користувачів, що дозволило пересвідчитися у стабільності роботи бізнес-логіки.

У ході тестування підсистеми безпеки та управління доступом було підтверджено надійність механізму автентифікації на базі JSON Web Token. Система коректно обробляє спроби входу, генерує токени доступу та забезпечує розмежування прав згідно з рольовою моделлю. Експериментально встановлено, що користувачі з роллю працівника не мають доступу до адміністративних панелей та не можуть переглядати дані інших колег, окрім загальнодоступного календаря, що відповідає вимогам конфіденційності [32]. Механізм автоматичного завершення сесії при закінченні терміну дії токена спрацьовує штатно, запобігаючи несанкціонованому доступу.

Ключовим об'єктом аналізу стала підсистема управління відсутностями. Перевірка показала, що алгоритми валідації даних ефективно блокують некоректні запити, такі як створення відпустки "заднім числом" або спроби перевищення доступного балансу днів. Операції створення, редагування та скасування заявок виконуються транзакційно, що гарантує узгодженість даних навіть при виникненні помилок на рівні мережі. Візуалізація даних у модулі календаря відповідає фактичним записам у базі даних: кольорове кодування типів відсутностей

відображається коректно, а фільтрація по відділах дозволяє менеджерам швидко оцінювати доступність ресурсів команди.

Особливу увагу при верифікації було приділено модулю обробки природної мови, який є визначальною особливістю системи. Серія тестів із використанням різноманітних текстових формулювань підтвердила здатність інтегрованої моделі Gemini коректно класифікувати наміри користувача. Система успішно розпізнає запити на створення відпустки, перевірку балансу та отримання довідкової інформації навіть при наявності граматичних помилок або сленгу. Функція вилучення сутностей продемонструвала високу точність при роботі з відносними часовими конструкціями: фрази "на наступний тиждень" або "до кінця місяця" коректно трансформуються у точні календарні дати. Механізм діалогу працює згідно з алгоритмом, ініціюючи уточнюючі запитання у випадках, коли вхідної інформації недостатньо для формування заявки [33].

Тестування автоматизованого робочого процесу погодження підтвердило коректність логіки маршрутизації заявок. При створенні запиту система безпомилково ідентифікує керівника ініціатора на основі ієрархічної структури бази даних та створює відповідні задачі на погодження. Сповіщення про необхідність дії та зміну статусу заявки надходять адресатам без суттєвих затримок завдяки використанню асинхронної черги задач. Сценарії делегування та багатоетапного затвердження відпрацьовують згідно із закладеною логікою, забезпечуючи прозорість процесу прийняття рішень.

Для підтвердження якості розробленого програмного продукту та його повної відповідності технічному завданню було проведено детальну верифікацію реалізованого функціоналу. Результати порівняльного аналізу запланованих вимог та фактичної реалізації узагальнено в Таблиці 4.1. У ній наведено матрицю трасування, де для кожної ключової вимоги зазначено статус її виконання, конкретний програмний модуль, що забезпечує цю функцію, та підсумковий результат тестування. Дані таблиці засвідчують, що система TimelyMind успішно покриває всі критичні бізнес-процеси, закладені на етапі проєктування. Вона показує, що кожна вимога, яку ставили на початку диплому, була реалізована.

Таблиця 4.1 – Результати верифікації функціональних вимог системи

Функціональна вимога	**Статус реалізації**	**Спосіб реалізації (Модуль/Компонент)**	**Результат тестування**
Реєстрація та автентифікація користувачів	Виконано	Auth Module (JWT, Password Hashing)	Успішний вхід, безпечне збереження даних
Створення запитів на відсутність (Vacation, Sick Leave)	Виконано	Absence API, Request Form UI	Запит зберігається в БД, статус "Pending"
Багаторівневе погодження заявок (Workflow)	Виконано	Approval Service, Manager Dashboard	Зміна статусу на "Approved"/"Rejected"
Інтеграція з AI-асистентом для довідки	Виконано	Gemini Client, AI Chat Widget	Коректні відповіді на запити про баланс
Візуалізація календаря команди	Виконано	Calendar API, Calendar View UI	Коректне відображення дат відсутності
Адміністрування користувачів та команд	Виконано	Admin Panel Module	Успішне додавання/редагування ролей

Загалом, результати проведеної верифікації свідчать про те, що розроблена система TimelyMind повністю відповідає функціональним вимогам. Реалізований програмний продукт забезпечує автоматизацію повного циклу управління відсутностями, надаючи користувачам зручні інструменти взаємодії через графічний та природномовний інтерфейси. Виявлені під час розробки незначні

відхилення були усунуті на етапі налагодження, і поточна версія системи демонструє стабільну роботу та готовність до дослідної експлуатації.

4.2 Методологія впровадження системи в організаціях

Ефективність використання розробленої системи TimelyMind у реальних виробничих умовах залежить не лише від якості програмного коду, але й від обраної стратегії її інтеграції в існуючу інфраструктуру підприємства. Процес впровадження є комплексною процедурою, що вимагає узгодження технічних, адміністративних та соціальних аспектів. Розроблена методологія базується на поетапному підході, який дозволяє мінімізувати ризики простою робочих процесів та забезпечити безболісну адаптацію персоналу до нових інструментів взаємодії.

Першим етапом впровадження є технічне розгортання системи в інформаційному середовищі замовника. Завдяки використанню технології контейнеризації Docker, цей процес є уніфікованим незалежно від типу інфраструктури, будь то власні сервери компанії або хмарні платформи. На цьому етапі системні адміністратори здійснюють конфігурацію змінних середовища, встановлюючи параметри підключення до бази даних, секретні ключі для генерації токенів доступу та API-ключі для інтеграції з сервісом Google Gemini. Критично важливим кроком є налаштування мережевої безпеки, включаючи встановлення SSL-сертифікатів для шифрування трафіку та обмеження доступу до адміністративних інтерфейсів [34].

Після успішного запуску серверної частини розпочинається етап початкової конфігурації та міграції даних. Для повноцінної роботи системи необхідно наповнити базу даних актуальною інформацією про організаційну структуру компанії. Адміністратори системи створюють профілі відділів, налаштовують типи відсутностей згідно з локальними політиками (кількість доступних днів, правила нарахування) та формують ієрархію підпорядкування. Особливої уваги вимагає імпорт облікових записів користувачів та їхніх поточних балансів відпусток. Оскільки більшість організацій ведуть такий облік у електронних таблицях або застарілих системах, методологія передбачає використання розроблених скриптів

для автоматизованого перенесення даних, що включає етап валідації для виявлення та виправлення помилок у історичних записах.

Третім, і найбільш тривалим етапом, є організаційна адаптація та навчання користувачів. Впровадження системи з елементами штучного інтелекту часто викликає у співробітників настороженість або завищені очікування. Тому стратегія впровадження включає проведення серії навчальних сесій, спрямованих на демонстрацію можливостей природномовного інтерфейсу. Користувачам пояснюються принципи формулювання запитів до чат-бота, межі його компетенції та алгоритми перевірки згенерованих відповідей. Для керівників проводяться окремі інструктажі щодо роботи з панелями моніторингу та управлінням процесами погодження.

Для мінімізації ризиків рекомендується застосування стратегії пілотного запуску. Система спочатку впроваджується в межах одного департаменту або проектної команди, що дозволяє виявити потенційні проблеми у конфігурації workflow та зібрати зворотний зв'язок від лояльної групи користувачів. У цей період система може працювати у режимі "тіньового копіювання", коли облік ведеться паралельно у новій та старій системах для верифікації розрахунків. Лише після успішного завершення пілотного періоду та стабілізації всіх процесів відбувається масштабування рішення на всю організацію та остаточна відмова від попередніх інструментів обліку. Такий підхід гарантує плавний перехід та високий рівень сприйняття інновації колективом.

4.3 Рекомендації щодо використання та адміністрування

Забезпечення стабільної та ефективної роботи системи TimelyMind у довгостроковій перспективі вимагає дотримання чіткого регламенту адміністрування та дотримання рекомендацій щодо експлуатації. Оскільки система оперує чутливими кадровими даними та залежить від зовнішніх сервісів штучного інтелекту, роль адміністратора набуває критичного значення. Першочерговим завданням технічного персоналу є регулярний моніторинг доступності та продуктивності API-інтерфейсів. Адміністраторам необхідно налаштувати

автоматизовані засоби сповіщення про помилки серверної частини та перебої у роботі зовнішньої моделі Gemini, оскільки це безпосередньо впливає на можливість обробки запитів користувачів. Рекомендується встановити ліміти на кількість запитів до LLM для запобігання перевитрат бюджету та перевантаження системи.

Критично важливим аспектом адміністрування є забезпечення цілісності та збереження даних. Необхідно впровадити політику регулярного резервне копіювання бази даних PostgreSQL. Рекомендований графік включає щоденне інкрементальне копіювання та повний бекап системи щотижня. Копії повинні зберігатися на незалежному фізичному носії або у хмарному сховищі, відокремленому від основного сервера. Крім того, перед застосуванням будь-яких оновлень структури бази даних через механізм міграцій Alembic, слід обов'язково створювати контрольну точку відновлення, щоб мати можливість відкотити зміни у разі виникнення конфліктів даних.

Для ефективного використання функціоналу обробки природної мови користувачам рекомендується дотримуватися певних принципів комунікації з інтелектуальним асистентом. Хоча модель здатна розуміти контекст, найвищу точність розпізнавання забезпечують запити, що містять чіткі часові рамки та однозначні формулювання намірів. Організаціям варто розробити коротку інструкцію з прикладами ефективних запитів для співробітників. Також важливо наголосити на необхідності обов'язкової верифікації користувачем параметрів заявки, які система сформувала на основі тексту. Остаточне рішення про підтвердження дат завжди має залишатися за людиною, а не за алгоритмом, щоб уникнути непорозумінь, викликаних можливою неоднозначністю лінгвістичних конструкцій.

З точки зору налаштування бізнес-логіки, адміністраторам рекомендується підтримувати актуальність довідників типів відсутностей та організаційної структури. При створенні нових типів відпусток слід надавати їм зрозумілі назви та детальні описи, оскільки ця інформація може використовуватися мовною моделлю для надання довідкових відповідей користувачам. Налаштування ланцюжків погодження повинно переглядатися при будь-яких кадрових змінах у

керівному складі компанії для уникнення ситуацій, коли заявки "зависають" на звільнених менеджерах. Для цього в системі передбачено функціонал перепризначення активних задач, яким адміністратори повинні користуватися оперативно.

Безпека системи вимагає регулярної ротації криптографічних ключів, що використовуються для підпису токенів доступу та шифрування сесій. Адміністраторам слід контролювати журнали доступу на предмет виявлення підозрілої активності, такої як масові спроби підбору паролів або аномальна кількість запитів від одного користувача. У разі звільнення співробітника його доступ до системи має бути заблокований негайно. Комплексне дотримання цих рекомендацій дозволить мінімізувати технічні ризики та забезпечити комфортну роботу персоналу з програмним продуктом.

4.4 Перспективи подальшого розвитку системи

Розроблена система TimelyMind на даному етапі є повнофункціональним MVP (Minimum Viable Product), який успішно вирішує поставлені задачі автоматизації обліку відсутностей та спрощення взаємодії користувача з HR-процесами. Однак стрімкий розвиток технологій штучного інтелекту та зростаючі вимоги бізнесу до корпоративних систем відкривають широкі можливості для подальшого вдосконалення та масштабування проекту. Аналіз архітектурних рішень та отриманих результатів дозволяє виділити декілька пріоритетних напрямків розвитку системи.

Одним із найбільш перспективних векторів є поглиблення інтеграції з великими мовними моделями шляхом впровадження технології RAG (Retrieval-Augmented Generation). У поточній реалізації система використовує LLM переважно для аналізу команд користувача. Розвиток цього напрямку передбачає завантаження у векторну базу даних внутрішніх нормативних документів компанії: політик відпусток, інструкцій, трудового розпорядку та довідників. Це дозволить інтелектуальному асистенту не лише виконувати команди ("створи заявку"), але й виступати у ролі консультанта, надаючи точні відповіді на складні запитання,

наприклад: "Скільки днів додаткової відпустки передбачено за весілля?" або "Як оплачується лікарняний у вихідний день?", спираючись на конкретні пункти корпоративних документів.

Важливим напрямком розвитку є розширення екосистеми інтеграцій. Хоча поточна версія системи є самодостатньою, для підвищення зручності користування доцільно реалізувати двосторонню синхронізацію з популярними календарними сервісами (Google Calendar, Microsoft Outlook). Це дозволить автоматично блокувати слоти у робочому календарі працівника при затвердженні відпустки в TimelyMind, запобігаючи призначенню зустрічей на час відсутності. Також перспективною є розробка нативних ботів для корпоративних месенджерів Slack та Microsoft Teams, які дозволять виконувати всі операції безпосередньо в інтерфейсі чату, без переходу у веб-браузер.

Наступним логічним кроком є впровадження предиктивної аналітики на базі машинного навчання. Накопичення історичних даних про відсутності дозволить побудувати моделі прогнозування, здатні виявляти приховані патерни поведінки персоналу. Система зможе завчасно попереджати керівників про ризики вигорання співробітників (наприклад, якщо працівник тривалий час не брав відпустку) або прогнозувати періоди пікового навантаження, коли значна частина команди планує відсутність. Такий перехід від описової аналітики до предиктивної значно підвищить цінність системи для стратегічного планування ресурсів.

Життєвий цикл програмного забезпечення передбачає постійне вдосконалення та адаптацію до нових потреб користувачів. На основі аналізу поточного функціоналу та зворотного зв'язку, отриманого під час тестування, було розроблено стратегічний план масштабування системи. У Таблиці 4.2 представлено дорожню карту розвитку TimelyMind, яка класифікує майбутні оновлення за трьома часовими горизонтами: короткостроковим, середньостроковим та довгостроковим. План охоплює як покращення зручності використання, так і впровадження нових технологічних рішень для автоматизації HR-процесів.

Таблиця 4.2 – План розвитку функціональності системи

Етап розвитку	**Орієнтовний термін**	**Заплановані функції**	**Очікуваний ефект**
Короткостроковий	1 місяць	– Сповіщення в Telegram/Slack – Експорт звітів у Excel/PDF – Темна тема інтерфейсу	Підвищення оперативності реагування на заявки та покращення UX
Середньостроковий	3 місяці	– Нативний мобільний додаток (iOS/Android) – Розширена AI-аналітика (прогнозування вигорання) – Календар державних свят для різних країн	Забезпечення доступності з будь-якого пристрою та поглиблення аналітики
Довгостроковий	6 місяців	– Повна інтеграція з Payroll-системами – Public API для сторонніх сервісів – Модуль Performance Review	Перетворення системи на комплексне рішення для управління талантами

Реалізація окреслених перспектив дозволить трансформувати TimelyMind з інструменту обліку в комплексну інтелектуальну платформу управління досвідом співробітників, що відповідає найвищим стандартам сучасної HR-індустрії.

ВИСНОВКИ

Мета атестаційної роботи полягала у підвищенні ефективності управління робочим часом в сучасних організаціях шляхом розробки інтелектуальної веб-системи TimelyMind. У ході дослідження вирішено науково-практичну задачу поєднання надійності корпоративного обліку з гнучкістю технологій обробки природної мови.

Основні наукові та практичні результати роботи:

- Аналіз предметної області виявив, що перехід до гібридних моделей роботи ускладнив контроль доступності персоналу, а існуючі рішення мають перевантажені інтерфейси. Це обґрунтувало необхідність впровадження розмовних інтерфейсів (Conversational UI) для спрощення рутинних операцій.
- Реалізовано клієнт-серверну архітектуру на базі REST API з використанням стеку Python/FastAPI та PostgreSQL. Застосування Docker забезпечило уніфікацію розгортання, а асинхронний підхід дозволив досягти високої продуктивності при інтеграції з AI-сервісами.
- Спроектовано нормалізовану базу даних, що підтримує складну організаційну ієрархію та забезпечує цілісність даних завдяки каскадним операціям та транзакційності.
- Розроблено NLP-модуль на базі Google Gemini, який виконує розпізнавання намірів (Intent Recognition) та вилучення сутностей (NER). Це дозволило трансформувати неструктуровані текстові запити користувача у валідні команди для системи.
- Впроваджено автоматизований workflow погодження, який динамічно маршрутизує заявки згідно з ієрархією компанії. Використання черги задач мінімізувало адміністративні затримки та забезпечило своєчасне інформування учасників процесу.

- Створено гібридний веб-інтерфейс (React), що поєднує візуалізацію календаря з чат-ботом. Тестування підтвердило покращення користувацького досвіду (UX) та зменшення часу на виконання цільових дій.
- Верифіковано працездатність системи шляхом функціонального тестування, яке підтвердило стабільність роботи бізнес-логіки, коректність обробки природної мови та захист персональних даних.

Наукова новизна роботи полягає в удосконаленні методу людино-машинної взаємодії в системах кадрового обліку шляхом застосування контекстно-залежних мовних моделей для автоматизації заповнення атрибутів транзакцій, що дозволяє нівелювати складність інтерфейсу для кінцевого користувача.

Практичне значення одержаних результатів полягає у створенні повністю функціонального MVP-продукту TimelyMind, готового до впровадження у малих та середніх IT-компаніях. Використання системи дозволяє скоротити час на адміністрування відпусток, зменшити кількість помилок при плануванні ресурсів та покращити загальний рівень задоволеності персоналу внутрішніми сервісами компанії.

Перспективи подальшого розвитку проекту пов'язані з розширенням можливостей AI-асистента за допомогою технології RAG для надання консультацій по внутрішніх документах компанії, а також впровадженням модулів предиктивної аналітики для прогнозування відтоку кадрів та вигорання працівників.

ПЕРЕЛІК ПОСИЛАНЬ

1. Zhang B., Wang H., Liu Y. Natural Language Processing for Human Resources: A Survey. 2024. URL: https://arxiv.org/abs/2410.16498 (дата звернення: 23.10.2025).

2. Workforce Management Market Size, Share, Growth Analysis, By Component, By Deployment, By Organization Size, By End-use, By Region - Industry Forecas. 2024. URL: https://www.skyquestt.com/report/workforce-management-market (дата звернення: 23.10.2025).

3. Market Guide for Workforce Management Applications. Gartner. 2024. URL: https://www.gartner.com/en/documents/5306663 (дата звернення: 25.10.2025).

4. Unlocking multimodal understanding across millions of tokens of context. Google DeepMind. 2024. URL: https://deepmind.google/technologies/gemini/ (дата звернення: 25.10.2025).

5. Strohmeier S. Digital Human Resource Management: A Conceptual Clarification. German Journal of Human Resource Management. 2020. 345–365p.

6. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics. Minneapolis, 2019. 4171–4186p. URL: https://aclanthology.org/N19-1423/ (дата звернення: 27.10.2025).

7. The Role of Natural Language Processing in Predictive HR Insights. International Journal of Research in Engineering and Science. 2024. URL: https://www.researchgate.net/publication/390802301 (дата звернення: 27.10.2025).

8. Voron S. Building Data Science Applications with FastAPI. 2nd ed. Packt Publishing, 2023. 518p.

9. Poulton N. Docker Deep Dive: Zero to Docker in a single book. Westholme Publishing, 2023. 439p.

10. Hattingh C. Using Asyncio in Python: Understanding Python's Asynchronous Programming Features. O'Reilly Media, 2020. 156p.

11. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. O'Reilly Media, 2020. 310p.

12. Obe R., Hsu L. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 3rd ed. O'Reilly Media, 2017. 308p.

13. Richardson L., Amundsen M., Ruby S. RESTful Web Clients: Enabling Reuse Through Hypermedia. O'Reilly Media, 2017. 350p.

14. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. IETF Tools. 2015. URL: https://tools.ietf.org/html/rfc7519 (дата звернення: 14.11.2025).

15. Wathan A., Schoger S. Refactoring UI. New Tailwind Labs Inc., 2018. 250p.

16. Lott S. F. Modern Python Cookbook. 2nd ed. Packt Publishing, 2020. 838p.

17. Bayer M. SQLAlchemy 2.0 Documentation. SQLAlchemy. 2024. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 15.11.2025).

18. Wiegers K., Beatty J. Software Requirements. 3rd ed. Microsoft Press, 2013. 672p.

19. Connolly T. M., Begg C. E. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Addison-Wesley, 2014. 1440p.

20. Geewax J. J. API Design Patterns. Shelter Manning Publications, 2021. 480p.

21. Google AI for Developers. Gemini API Documentation: Prompt design strategies. Google AI. 2024. URL: https://ai.google.dev/docs/prompt_best_practices (дата звернення: 20.11.2025).

22. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 616p.

23. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.

24. OpenAPI Specification v3.1.0. OpenAPI Initiative. 2023. URL: https://spec.openapis.org/oas/v3.1.0 (дата звернення: 20.11.2025).

25. Nielsen J. Usability Engineering. Morgan Kaufmann, 1993. 362 p.

26. Weske M. Business Process Management: Concepts, Languages, Architectures. 3rd ed. Springer, 2019. 417 p.

27. Okken B. Python Testing with pytest: Simple, Rapid, Effective, and Scalable. 2nd ed. Pragmatic Bookshelf, 2022. 256 p.

28. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 240 p.

29. Kotter J. P. Leading Change. Harvard Business Review Press, 2012. 208 p.

30. Ferraiolo D. F., Kuhn D. R., Chandramouli R. Role-Based Access Control. 2nd ed. Artech House, 2007. 400 p.

31. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010. 512 p.

32. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 612 p.

33. Agrawal A., Gans J., Goldfarb A. Prediction Machines: The Simple Economics of Artificial Intelligence. Harvard Business Review Press, 2018. 272 p.

34. Bersin J. HR Technology 2024: The Big Report. The Josh Bersin Company. 2024. URL: https://joshbersin.com/research/hr-technology-market-2024/ (дата звернення: 05.12.2025).

ДОДАТОК А. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

ШІ асистент з обмеженням виконання функцій

backend/app/services/ai_assistant.py:

```
ABSENCE_FUNCTIONS = [
    FunctionDeclaration(
        name="create_absence_request",
        description="Create a new absence request for the current user",
        parameters={
            "type": "object",
            "properties": {
                "absence_type": {
                    "type": "string",
                    "description": "Type of absence: Vacation, Sick Leave, Day Off, or
Remote Work",
                    "enum": ["Vacation", "Sick Leave", "Day Off", "Remote Work"]
                },
                "start_date": {
                    "type": "string",
                    "description": "Start date in YYYY-MM-DD format"
                },
                "end_date": {
                    "type": "string",
                    "description": "End date in YYYY-MM-DD format"
                },
                "comment": {
                    "type": "string",
                    "description": "Optional comment for the request"
                }
            },
```

```
            "required": ["absence_type", "start_date", "end_date"]
        }
    ),
    FunctionDeclaration(
        name="get_my_balance",
        description="Get the current user's absence balance",
        parameters={"type": "object", "properties": {}}
    ),
    FunctionDeclaration(
        name="get_who_is_absent_today",
        description="Get list of people who are absent today",
        parameters={"type": "object", "properties": {}}
    ),
]
class AIAssistantService:
    """AI Assistant for absence management with function calling."""
    async def process_message(
        self,
        message: str,
        history: list[dict] | None = None
    ) -> dict[str, Any]:
        """Process a user message and return response."""
        # Build conversation for Gemini
        contents = []
        if history:
            for msg in history:
                role = "user" if msg["role"] == "user" else "model"
                contents.append({"role": role, "parts": [msg["content"]]})
        contents.append({"role": "user", "parts": [message]})
```

```
# Create model with tools
model = genai.GenerativeModel(
    settings.gemini_model,
    system_instruction=self.system_prompt,
    tools=[self.tools]
)
# Generate response
response = model.generate_content(contents)
# Check for function calls
if response.candidates and response.candidates[0].content.parts:
    for part in response.candidates[0].content.parts:
        if hasattr(part, "function_call") and part.function_call:
            func_name = part.function_call.name
            func_args = dict(part.function_call.args)
            # Execute function and get result
            function_result = await self._execute_function(func_name, func_args)
            # Send function result back to model for natural response
            contents.append({"role": "model", "parts": [part]})
            contents.append({
                "role": "user",
                "parts": [genai.protos.Part(
                    function_response=genai.protos.FunctionResponse(
                        name=func_name,
                        response={"result": function_result}
                    )
                )]
            })
            final_response = model.generate_content(contents)
            return {
                "response": final_response.text,
```

```
                "function_called": func_name,
                "function_result": function_result
            }
    return {"response": response.text, "function_called": None}
```

Автоматичний Workflow схвалення

backend/app/services/absence.py:

```
  async def _setup_approval_workflow(self, request: AbsenceRequest, user: User) ->
None:
    """Setup approval workflow based on department manager and user hierarchy."""
    step_order = 1
    added_approvers = set()
    # First, add department manager as approver
    if user.department_id:
        department = await self.department_repo.get_by_id(user.department_id)
        if department and department.manager_id and department.manager_id != user.id:
            await self.approval_repo.create(
                request_id=request.id,
                approver_id=department.manager_id,
                step_order=step_order,
            )
            added_approvers.add(department.manager_id)
            step_order += 1
    # Then, add user's direct manager if different from department manager
    if user.manager_id and user.manager_id not in added_approvers:
        await self.approval_repo.create(
            request_id=request.id,
            approver_id=user.manager_id,
            step_order=step_order,
        )
```

```
            added_approvers.add(user.manager_id)
            step_order += 1
            # Check for second level manager
            manager = await self.user_repo.get_by_id(user.manager_id)
            if manager and manager.manager_id and manager.manager_id not in
added_approvers:
                await self.approval_repo.create(
                    request_id=request.id,
                    approver_id=manager.manager_id,
                    step_order=step_order)
        # If no approvers added, auto-approve (for top-level users)
        if not added_approvers:
            request.status = RequestStatus.APPROVED
            await self.request_repo.update(request)
```

SQLAlchemy 2.0 моделі з типізацією

backend/app/models/absence.py:

```
  class AbsenceRequest(Base, TimestampMixin):
        __tablename__ = "absence_requests"
        id: Mapped[int] = mapped_column(primary_key=True)
        start_date: Mapped[date] = mapped_column(Date, nullable=False)
        end_date: Mapped[date] = mapped_column(Date, nullable=False)
        status: Mapped[RequestStatus] = mapped_column(
            default=RequestStatus.PENDING, nullable=False
        )
        comment: Mapped[str | None] = mapped_column(Text, nullable=True)
        # Foreign keys
        user_id: Mapped[int] = mapped_column(
            ForeignKey("users.id", ondelete="CASCADE"), nullable=False
        )
```

```
    absence_type_id: Mapped[int] = mapped_column(
        ForeignKey("absence_types.id", ondelete="RESTRICT"), nullable=False
    )
    # Relationships with lazy loading
    user: Mapped["User"] = relationship(
        "User", back_populates="absence_requests", lazy="selectin")
    absence_type: Mapped["AbsenceType"] = relationship(
        "AbsenceType", back_populates="absence_requests", lazy="selectin")
    approval_steps: Mapped[list["ApprovalStep"]] = relationship(
        "ApprovalStep",
        back_populates="absence_request",
        lazy="selectin",
        cascade="all, delete-orphan")
    @property
    def days_count(self) -> int:
        return (self.end_date - self.start_date).days + 1
```

JWT автентифікація

backend/app/core/security.py:

```
  def get_password_hash(password: str) -> str:
      """Hash a password using bcrypt."""
      salt = bcrypt.gensalt()
      hashed = bcrypt.hashpw(password.encode("utf-8"), salt)
      return hashed.decode("utf-8")
  def create_access_token(
      subject: int | str,
      expires_delta: timedelta | None = None
  ) -> str:
      """Create a JWT access token."""
      if expires_delta:
```

```
        expire = datetime.now(timezone.utc) + expires_delta
    else:
        expire = datetime.now(timezone.utc) + timedelta(
            minutes=settings.access_token_expire_minutes
        )
    to_encode = {"exp": expire, "sub": str(subject)}
    encoded_jwt = jwt.encode(
        to_encode, settings.secret_key, algorithm=settings.algorithm
    )
    return encoded_jwt
def decode_access_token(token: str) -> dict | None:
    """Decode and verify a JWT access token."""
    try:
        payload = jwt.decode(
            token, settings.secret_key, algorithms=[settings.algorithm]
        )
        return payload
    except JWTError:
        return None
```

React Chat компонент з Hooks

frontend/src/components/chat/QuickChat.tsx:

```
export function QuickChat() {
  const [messages, setMessages] = useState<ChatMessage[]>([]);
  const [input, setInput] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const messagesEndRef = useRef<HTMLDivElement>(null);
  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);
```

```
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!input.trim() || isLoading) return;
    const userMessage = input.trim();
    setInput('');
    setIsLoading(true);
    // Optimistic update
    const tempUserMsg: ChatMessage = {
      id: Date.now(),
      role: 'user',
      content: userMessage,
      created_at: new Date().toISOString(),
    };
    setMessages((prev) => [...prev, tempUserMsg]);
    try {
      const response = await chatApi.sendMessage(userMessage);
      setMessages((prev) => [
        ...prev.slice(0, -1),
        response.user_message,
        response.assistant_message,
      ]);
    } catch {
      setMessages((prev) => prev.slice(0, -1));
      setInput(userMessage);
    } finally {
      setIsLoading(false);
    }
  };
  // ... render JSX
}
```

ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Комп'ютерних наук

ДИПЛОМНА РОБОТА

на ступінь вищої освіти магістр

із спеціальності F3 Комп'ютерні науки

Інтелектуальна система управління робочим часом з обробкою природної мови

Виконав: студент 6 курсу, групи КНДМ-62

Жук Ярослав Русланович

Керівник: доктор філософії Кравчук П. О.

Київ - 2025

Слайд 1

Проблематика: Бюрократія та втрата часу в управлінні робочим часом

Надмірна бюрократія

Традиційні системи управління часом часто перевантажені складними формами та багатоступеневими процедурами погодження, що сповільнює процеси.

Втрата продуктивності

Значна частина робочого часу витрачається на рутинні адміністративні задачі замість виконання основних обов'язків, що знижує загальну ефективність команди.

Складність комунікації

Відсутність єдиного інструменту для подачі запитів та отримання погоджень призводить до фрагментації комунікації та непорозумінь.

Людський фактор та помилки

Ручне введення даних та обробка запитів підвищує ризик помилок, що може мати негативні наслідки для планування та обліку.

Слайд 2

Аналіз існуючих рішень та їх недоліки

Традиційні HRM-системи	Комплексний функціонал, інтеграція з HR-процесами.	Висока вартість, складність впровадження, обмеженість у NLP-функціях.
Календарні додатки (Google Calendar)	Простота використання, широке розповсюдження, доступність.	Відсутність функціоналу погоджень, не підтримує офіційні запити, немає NLP.
Чат-боти для HR (окремі)	Швидка комунікація, автоматизація частин запитів.	Обмежений функціонал, відсутність інтеграції з системою обліку часу, немає workflow.
TimelyMind (наше рішення)	Інтуїтивний інтерфейс, NLP, автоматизація погоджень, комплексний облік часу.	Новизна на ринку, потреба в адаптації до корпоративних політик.

Слайд 3

Мета та задачі дослідження

1

Мета: Розробка інтелектуальної системи TimelyMind

Створення комплексної системи для ефективного управління робочим часом, що мінімізує бюрократію та підвищує продуктивність за рахунок обробки природної мови.

2

Задача 1: Аналіз та моделювання

Дослідження поточних процесів управління часом та розробка оптимальної моделі даних для системи TimelyMind.

3

Задача 2: Проектування архітектури

Розробка надійної, масштабованої та безпечної архітектури системи із застосуванням сучасних технологій.

4

Задача 3: Розробка NLP-модуля

Впровадження функціоналу обробки природної мови для розпізнавання та обробки запитів користувачів через чат-бот.

5

Задача 4: Реалізація Workflow

Створення гнучкого механізму workflow для автоматизації погодження запитів (відпустки, лікарняні тощо).

Слайд 4

Об'єкт та предмет дослідження

Об'єкт дослідження: Процеси управління робочим часом

Усі аспекти, пов'язані з плануванням, обліком, погодженням та аналізом робочого часу співробітників у корпоративному середовищі, а також вплив бюрократичних процедур на продуктивність.

- Аналіз існуючих методик та інструментів.
- Виявлення больових точок та неефективності.
- Формування вимог до автоматизації процесів.

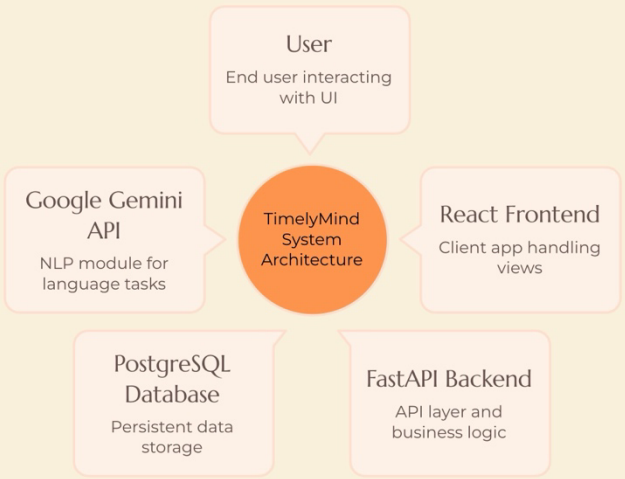
Предмет дослідження: Методи та інструменти інтелектуальної автоматизації

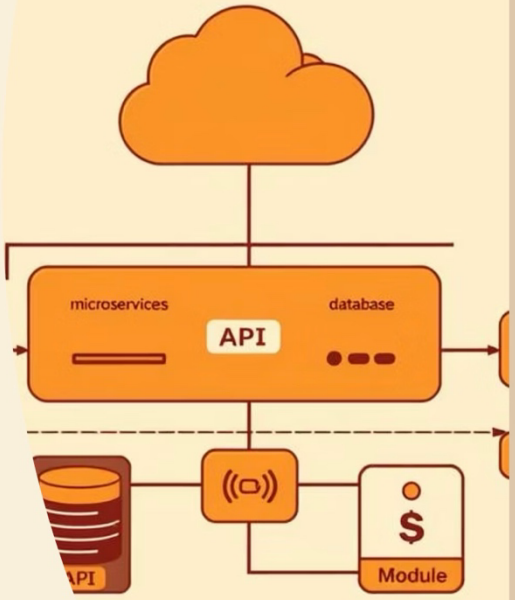
Технології обробки природної мови (NLP) та штучного інтелекту, що застосовуються для розробки автоматизованих систем управління робочим часом з акцентом на взаємодії через чат-бот.

- Вибір оптимального стеку технологій (Python, FastAPI, React, PostgreSQL).
- Розробка архітектури NLP-модуля на базі Google Gemini API.
- Створення алгоритмів для автоматичного формування та погодження запитів.

Слайд 5

Загальна архітектура системи TimelyMind





Архітектура TimelyMind використовує мікросервісну модель для гнучкості, масштабованості та надійності, спрощуючи інтеграцію та додавання нового функціоналу.

Слайд 6

Backend: Швидкість та асинхронність з Python та FastAPI

1

Python: Мова програмування

Обраний через свою гнучкість, велику кількість бібліотек та популярність у сфері обробки даних та AI.

2

FastAPI: Веб-фреймворк

Використаний для створення високопродуктивних асинхронних API завдяки сучасним стандартам та автоматичній генерації документації (OpenAPI).

3

Асинхронність

Забезпечує паралельну обробку запитів, що критично для чат-бота та інтеграції з зовнішніми AI-сервісами без блокування основного потоку.

Вибір цього стеку дозволив створити ефективний та швидкий бекенд, здатний обробляти велику кількість одночасних запитів, що є ключовим для інтерактивної системи з NLP.

Слайд 7

Database: PostgreSQL та схема даних

Вибір PostgreSQL

Реляційна база даних PostgreSQL була обрана за її надійність, потужність, підтримку складних запитів та широкі можливості масштабування. Вона ідеально підходить для зберігання структурованих даних про користувачів, запити та workflow.

- Підтримка ACID-транзакцій.
- Висока продуктивність та безпека.
- Гнучка система індексації та оптимізації запитів.

Ключові сутності схеми

Схема бази даних включає сутності "Користувачі", "Запити на час" (відпустки, лікарняні), "Статуси погодження", "Workflow-шаблони" та "Журнал подій", що забезпечує комплексний облік та відстеження всіх процесів.

- Таблиця `users`: інформація про співробітників.
- Таблиця `time_requests`: деталі запитів користувачів.
- Таблиця `approvals`: ланцюжки погоджень.

Слайд 8

Frontend: Сучасний інтерфейс на React

Інтуїтивний користувацький інтерфейс

Розроблений з урахуванням принципів UX/UI для забезпечення максимально простої та зрозумілої взаємодії користувача з системою.

Адаптивний дизайн

Інтерфейс автоматично підлаштовується під будь-який розмір екрану (десктоп, планшет, мобільний), забезпечуючи комфортну роботу з будь-якого пристрою.

Реактивність та оновлення в реальному часі

Використання React дозволяє створювати динамічні компоненти, які швидко реагують на дії користувача та оновлюють дані в реальному часі.

Frontend-частина TimelyMind є обличчям системи, що забезпечує не лише функціональність, але й приємний та ефективний користувацький досвід.

Слайд 9

Інтеграція AI: Google Gemini API та промпт-інжиніринг

Google Gemini API: Основа NLP-модуля

Для реалізації можливостей обробки природної мови був обраний Google Gemini API. Він забезпечує високу точність розпізнавання намірів користувача та вилучення ключових даних з текстових запитів, що є фундаментом для роботи чат-бота.

- Висока точність розпізнавання.
- Широкі можливості для різних мов.
- Масштабованість та надійність від Google.

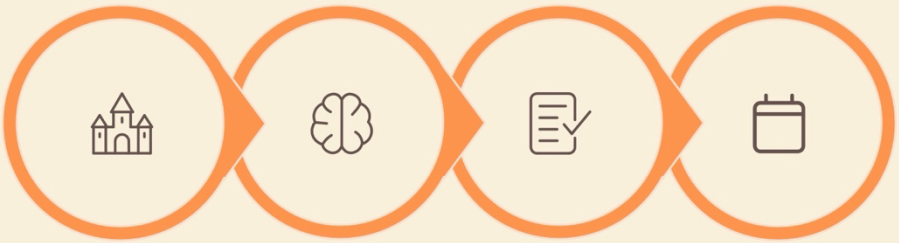
Промпт-інжиніринг для TimelyMind

Розроблено спеціальні промпти, які дозволяють Gemini API точно інтерпретувати запити користувачів, такі як "Хочу у відпустку з вівторка", та перетворювати їх на структуровані дані для системи.

- Ефективне вилучення дати, типу запиту та інших параметрів.
- Обробка нестандартних формулювань.
- Формування чітких інструкцій для системи.

Слайд 10

Шлях користувача: Від запиту до календаря



User Request | AI Processing | Approval Workflow | Calendar Update

Давайте розглянемо спрощений сценарій взаємодії користувача з системою TimelyMind. Цей шлях демонструє ефективність інтеграції NLP та автоматизації процесів.

1. **Користувач робить запит:** Через чат-бот уводить запит природною мовою (наприклад, "відпустка з наступного понеділка").
2. **AI обробка:** Система аналізує текст, витягує необхідні дані та створює проект запиту.
3. **Погодження:** Запит автоматично надсилається на погодження відповідним керівникам згідно з налаштованим workflow.
4. **Оновлення календаря:** Після погодження, подія автоматично додається до персонального та командного календарів.

Слайд 11

Workflow погодження: Гнучкість та контроль

1

Налаштування правил

Можливість створювати власні ланцюжки погодження для різних типів запитів.

2

Багаторівневі погодження

Підтримка послідовних та паралельних етапів затвердження.

3

Сповіщення та статус

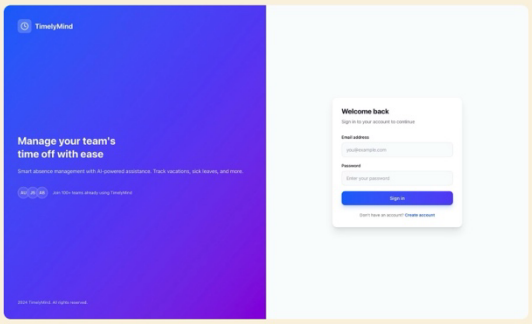
Автоматичні сповіщення учасникам процесу та відстеження статусу кожного запиту.

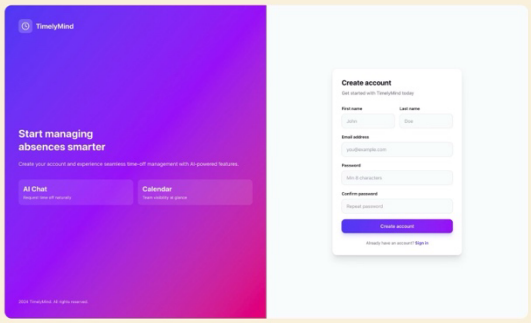
Система дозволяє компаніям адаптувати процеси погодження під свої унікальні потреби, забезпечуючи прозорість та керованість.

Слайд 12

Користувацький Досвід: Екрани Входу та Реєстрації

Для забезпечення безперебійної та інтуїтивно зрозумілої взаємодії, ми приділили особливу увагу дизайну ключових екранів, таких як вхід та реєстрація. Ці інтерфейси розроблені з акцентом на простоту використання, безпеку та адаптивність до будь-яких пристроїв.





Ці скріншоти демонструють початкові етапи взаємодії користувача з TimelyMind, що є основою для подальшого ефективного управління робочим часом.

Слайд 13

Користувацький Досвід: Основні Дашборди та Швидкі Дії

Для максимально ефективної взаємодії, TimelyMind пропонує інтуїтивно зрозумілі дашборди, миттєві дії та інтеграцію з AI-чатботом, що дозволяє користувачам швидко орієнтуватися та управляти своїм робочим часом.

Головний дашборд

Огляд ключових показників, персоналізовані віджети та легкий доступ до необхідних даних для щоденного планування.

Швидкі дії

Миттєве виконання найпоширеніших операцій, таких як запит відпустки або оформлення лікарняного, прямо з головного екрану.

Швидкий AI чат

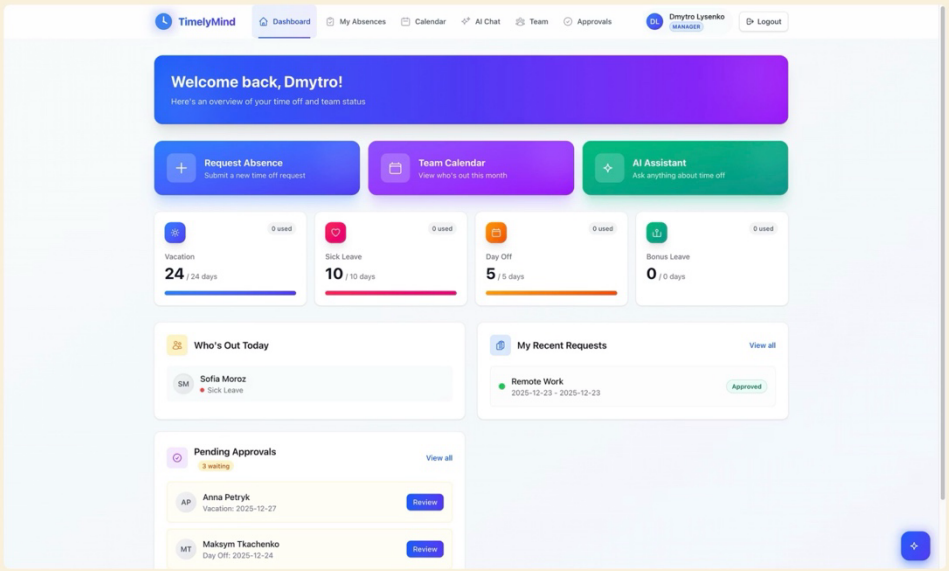
Інтелектуальний помічник, який розуміє запити природною мовою та допомагає управляти завданнями та графіком.

Ці елементи забезпечують швидку та безперешкодну взаємодію з системою, мінімізуючи витрати часу на рутинні операції.

Слайд 14

Демонстрація системи: Головний дашборд

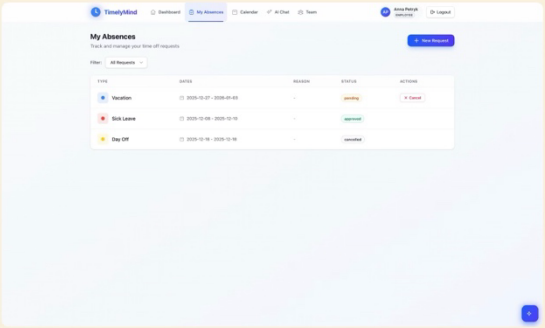
Представляю головний дашборд TimelyMind – ваш центральний пункт управління робочим часом. Тут ви знайдете всю необхідну інформацію з першого погляду.

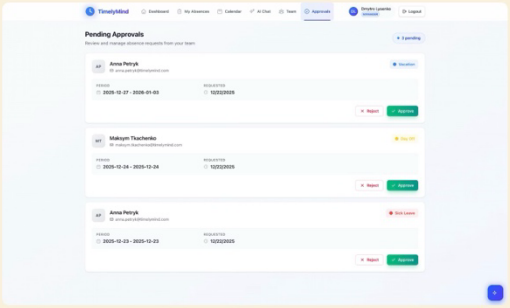


Слайд 15

Мої Запити: Керування Вашими Запитами на Час

Цей розділ детально демонструє інтерфейс сторінки "Мої Запити" в системі TimelyMind, де кожен користувач може легко переглядати статус своїх запитів, історію погоджень та ініціювати нові запити на час. Екран розроблено для максимальної прозорості та контролю.



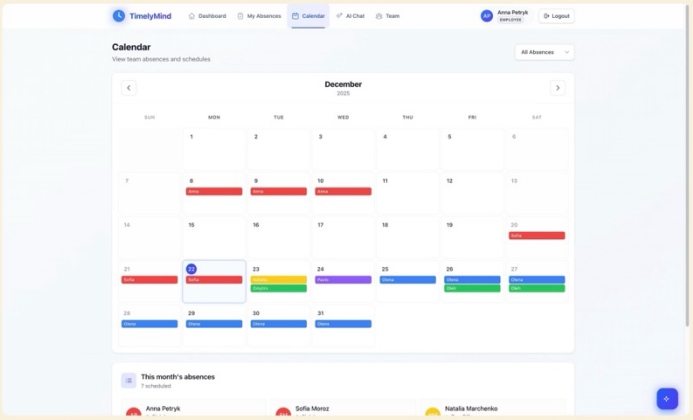


Наведені скріншоти ілюструють, як інтуїтивно зрозумілий дизайн дозволяє користувачам ефективно управляти власним робочим часом, відстежувати прогрес своїх запитів та взаємодіяти з системою без зайвих зусиль.

Слайд 16

Командний Календар: Візуалізація Результатів Workflow

Ця сторінка надає комплексний огляд графіку команди, відображаючи всі запити на час та їхній статус погодження в зручному візуальному форматі. Вона дозволяє швидко оцінити доступність команди та планувати ресурси, забезпечуючи максимальну прозорість.

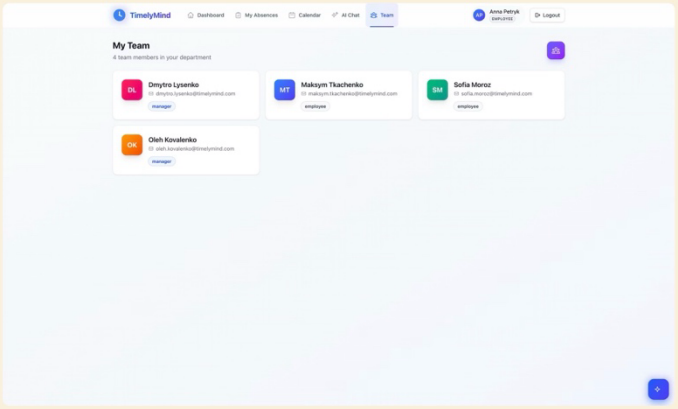


На цих скріншотах ви можете побачити, як TimelyMind трансформує складні процеси управління часом у зрозумілий та інтерактивний інтерфейс, підвищуючи прозорість та ефективність командної роботи.

Слайд 17

Командний календар: сторінка "Моя команда"

Ця картка демонструє візуалізацію результатів workflow на сторінці "Моя команда". Тут керівники та члени команди можуть швидко переглядати графіки відсутностей, погодження та загальну доступність для ефективного планування завдань та ресурсів.

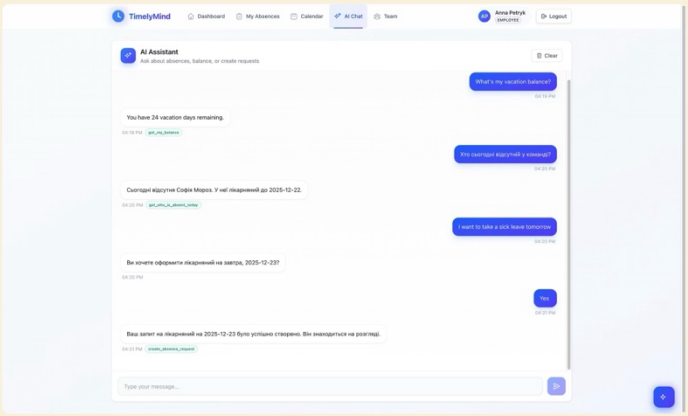


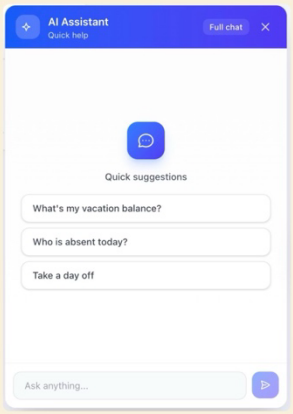
Представлені скріншоти показують, як TimelyMind забезпечує прозорість та ефективність в управлінні командним часом, дозволяючи легко відстежувати статус кожного запиту та планувати ресурси.

Слайд 18

AI-чат: Розумний помічник для командного календаря

AI-чат у TimelyMind не лише відповідає на запитання, а й активно допомагає в інтерпретації результатів workflow та управлінні командним календарем. Він може аналізувати дані, пропонувати рішення та навіть автоматично формувати запити на основі ваших вказівок.



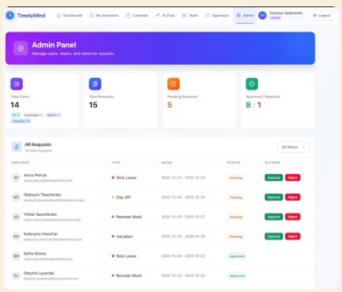


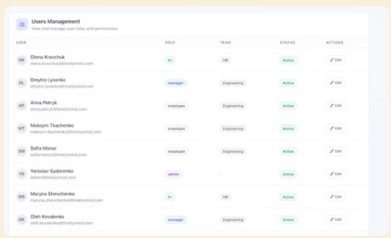
Інтеграція AI-чату значно прискорює та спрощує взаємодію з системою, перетворюючи складні аналітичні задачі на легкий діалог.

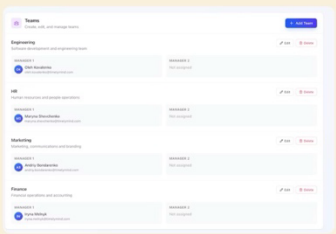
Слайд 19

Admin panel: Адміністрування та суперможливості

Admin panel дозволяє керувати поточним станом у системі. Вона дозволяє мати інформацію про усіх користувачів, керувати створенням та видаленням команд, додаванням працівників до команд та звільнювати їх відбираючи доступ до системи.

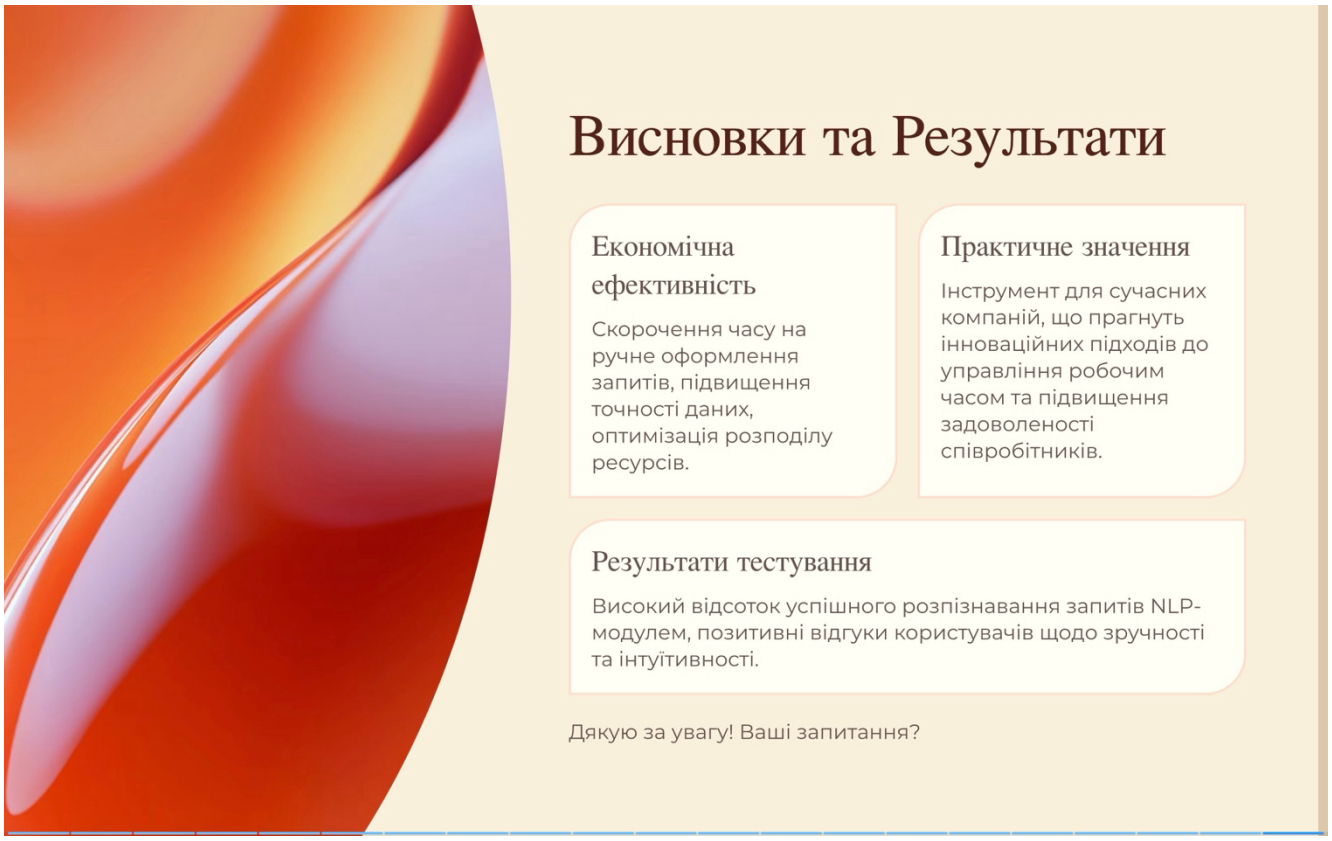






Admin має змогу підтверджувати/відхиляти запити як manager, призначати керівників командам, слідкувати за перебігом усіх запитів.

Слайд 20


Висновки та Результати
Економічна ефективність
Скорочення часу на ручне оформлення запитів, підвищення точності даних, оптимізація розподілу ресурсів.
Практичне значення
Інструмент для сучасних компаній, що прагнуть інноваційних підходів до управління робочим часом та підвищення задоволеності співробітників.
Результати тестування
Високий відсоток успішного розпізнавання запитів NLP-модулем, позитивні відгуки користувачів щодо зручності та інтуїтивності.
Дякую за увагу! Ваші запитання?

Слайд 21