

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри інформаційних
систем і технологій

_____ Ігор БОЛБОТ
(підпис)

_____ Михайло ШВИДЕНКО
(підпис)

«___» _____ 2026 р.

«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«Інформаційна система моніторингу робочої активності
персоналу»**

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
д-р екон. наук, проф.

_____ (підпис)

Роман РУДЕНСЬКИЙ

**Керівник бакалаврської
кваліфікаційної роботи**
д-р філос., доц.

_____ (підпис)

Валентина КОРОЛЬЧУК

Виконав

_____ (підпис)

Юрій ІЛЬЧЕНКО

Київ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

**Завідувач кафедри інформаційних систем і
технологій**

к.е.н., доцент _____ Михайло ШВИДЕНКО
(підпис)

« ____ » _____ 2026 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Ільченку Юрію Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Інформаційна система моніторингу робочої активності персоналу»

затверджена наказом від «06» січня 2026 року № 46 «С»

Термін подання завершеної роботи на кафедру « ____ » _____ 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи:

вебсистема моніторингу робочого часу «Smart Time-Tracker».

Перелік питань, що підлягають дослідженню:

Провести аналіз предметної області, аналогів та обґрунтувати стек технологій.

Спроектувати архітектуру системи, модель доступу та структуру бази даних.

Розробити бекенд-логіку, клієнтський інтерфейсу. Інтегрувати ШІ-асистента (Gemini API) та налаштувати автоматичну розсилку звітів.

Дата видачі завдання «19» січня 2026 р.

**Керівник бакалаврської
кваліфікаційної роботи**

(підпис)

Валентина КОРОЛЬЧУК

Завдання взяв до виконання

(підпис)

Юрій ІЛЬЧЕНКО

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП	7
РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ПРОЄКТНИХ РІШЕНЬ	10
1.1. Аналіз сучасних підходів до моніторингу робочого часу в ІТ-індустрії	10
1.2. Огляд систем-аналогів та порівняння методів автоматизації звітності	11
1.3. Дослідження можливостей застосування генеративного штучного інтелекту в корпоративних системах	14
1.4. Обґрунтування вибору технологічного стеку: переваги Django, PostgreSQL та інтеграції Google Generative AI (Gemini)	16
РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ	19
2.1. Моделювання бізнес-процесів взаємодії користувачів із застосуванням рольової моделі (RBAC)	19
2.2. Проєктування архітектури системи та схеми інтеграції із зовнішніми API	22
2.3. Інфологічне та даталогічне проєктування бази даних для зберігання сеансів та статистичних показників	25
2.4. Розроблення логіки роботи AI-асистента та алгоритмів візуалізації аналітичних даних	28
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ОРІЄНТОВАНОГО ДОДАТКА	33
3.1. Розроблення бекенд-логіки на базі Django: керування сеансами, фільтрація та пагінація даних	33

3.2. Реалізація клієнтського інтерфейсу за допомогою Bootstrap 5 та інтерактивних графіків Chart.js	37
3.3. Практична реалізація інтеграції з Gemini API для створення інтелектуального помічника	40
3.4. Автоматизація розсилки аналітичних звітів засобами SMTP та експорту даних у формат CSV	43
3.5. Тестування функціональних можливостей та безпеки доступу до даних	48
РОЗДІЛ 4. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ТА ЕФЕКТИВНІСТЬ ПРОЄКТУ	53
4.1. Розрахунок витрат на розроблення та хмарну інфраструктуру системи	53
4.2. Оцінка соціально-економічного ефекту від впровадження системи та автоматизації управлінських функцій	55
ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61
ДОДАТКИ	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІТ – інформаційні технології

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ІІІ – штучний інтелект

AI (Artificial Intelligence) – штучний інтелект

API (Application Programming Interface) – інтерфейс прикладного програмування

BI (Business Intelligence) – бізнес-аналітика, методи та інструменти перетворення сирих даних на зручну для сприйняття інформацію

BPMN (Business Process Model and Notation) – стандартна нотація для моделювання бізнес-процесів

CSV (Comma-Separated Values) – текстовий формат для збереження табличних даних

ER (Entity-Relationship) – модель (діаграма) сутність-зв'язок

ERP (Enterprise Resource Planning) – система планування ресурсів підприємства

HTML (HyperText Markup Language) – стандартна мова розмітки вебсторінок

HTTP/HTTPS (HyperText Transfer Protocol / Secure) – протокол передавання гіпертексту / захищений протокол

JSON (JavaScript Object Notation) – текстовий формат для обміну даними

LLM (Large Language Model) – велика мовна модель (різновид штучного інтелекту)

MVT (Model-View-Template) – архітектурний шаблон проєктування програмного забезпечення (Модель-Представлення-Шаблон)

ORM (Object-Relational Mapping) – об'єктно-реляційне відображення, технологія зв'язування баз даних із концепціями об'єктно-орієнтованих мов

RBAC (Role-Based Access Control) – керування доступом на основі ролей користувачів

ROI (Return on Investment) – фінансовий показник рентабельності (окупності) інвестицій

SaaS (Software as a Service) – програмне забезпечення як послуга (хмарна модель надання ПЗ)

SMTP (Simple Mail Transfer Protocol) – мережевий протокол для передавання електронної пошти

SPACE (Satisfaction, Performance, Activity, Communication, Efficiency) – багатовимірний фреймворк для оцінки продуктивності розробників

SQL (Structured Query Language) – мова структурованих запитів для роботи з реляційними базами даних

UI (User Interface) – інтерфейс користувача

UML (Unified Modeling Language) – уніфікована мова моделювання

UX (User Experience) – користувацький досвід (досвід взаємодії з інтерфейсом)

ВСТУП

Сучасний етап розвитку глобальної економіки характеризується стрімкою цифровою трансформацією, що докорінно змінила підходи до організації робочого простору. Останні роки продемонстрували масовий перехід підприємств, особливо в секторі інформаційних технологій (ІТ), на гібридні або повністю дистанційні моделі функціонування. У таких умовах традиційні механізми контролю діяльності персоналу, що базувалися на фізичній присутності в офісі, виявилися малоефективними. Менеджмент компаній постає перед викликом: як забезпечити високу продуктивність команд, не вдаючись до деструктивного мікроменеджменту, який пригнічує ініціативу та знижує рівень довіри всередині колективу.

Згідно з дослідженнями фреймворку SPACE, розробленого фахівцями GitHub та Microsoft Research [1], продуктивність розробників програмного забезпечення є комплексним поняттям, яке неможливо виміряти лише кількістю відпрацьованих годин. Вона включає задоволеність (Satisfaction), ефективність (Performance), активність (Activity), комунікацію (Communication) та продуктивність (Efficiency). Водночас, як зазначають Лаксанаджаджа та Ов'єдо-Треспаласіос (2024), існуюче програмне забезпечення для моніторингу часто створює «павутину таємниць» та етичних ризиків через надмірну інвазивність та відсутність прозорості, що призводить до професійного вигорання та стресу працівників [2]. Тому виникає гостра потреба у створенні систем нового покоління, таких як «Smart Time-Tracker», які фокусуються на об'єктивному зборі даних та інтелектуальній підтримці персоналу, а не на тотальному нагляді.

Актуальність розроблення підсилюється стрімким розвитком генеративного штучного інтелекту. Згідно зі звітом Menlo Ventures за 2025 рік, корпоративний ШІ став категорією програмного забезпечення, що масштабується найшвидше в історії, охопивши значну частку глобального ринку

SaaS [3]. Інтеграція таких моделей, як Gemini API, дозволяє трансформувати систему обліку часу в інтелектуальне когнітивне середовище. Як підтверджують останні дослідження Страй та ін. (2026), використання ШІ-інструментів не лише підвищує активність користувачів, а й суттєво покращує їхній суб'єктивний досвід продуктивності та задоволеності роботою [4].

Окрім інтелектуальної складової, важливим аспектом є візуалізація даних у реальному часі. Використання сучасних аналітичних інструментів (Business Intelligence) дозволяє перетворювати фрагментовані логи активності на динамічні дашборди, що критично важливо для зростання компаній в умовах високої конкуренції [5]. Впровадження власної системи на базі Django 5.0 та PostgreSQL із використанням фонові обробки завдань (Celery, Redis) та автоматизованої звітності дозволяє забезпечити високу масштабованість та надійність при оптимальних експлуатаційних витратах [6].

Метою дипломної роботи є проєктування та програмна реалізація інтелектуальної веб-орієнтованої інформаційної системи «Smart Time-Tracker», яка забезпечує прозору фіксацію робочого часу, збір аналітичних метрик за допомогою Chart.js та надання інтелектуальних порад персоналу через інтеграцію з Gemini API.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести системний аналіз предметної області, дослідити сучасні метрики продуктивності (фреймворк SPACE) та проаналізувати етичні аспекти моніторингу персоналу.
2. Здійснити порівняльну характеристику існуючих аналогів та обґрунтувати переваги інтелектуальної системи на базі технологічного стеку Python/Django.
3. Спроектувати архітектуру системи та структуру реляційної бази даних, враховуючи вимоги до безпеки та розмежування прав доступу на основі ролей (RBAC).

4. Розробити програмні модулі для фіксації робочих сеансів у реальному часі та інтелектуальний AI-асистент на базі великих мовних моделей.

5. Реалізувати BI-модуль для візуалізації статистики активності та автоматизувати процес генерації аналітичних звітів з розсилкою через Gmail SMTP.

6. Провести тестування функціональних можливостей системи та оцінити її техніко-економічну ефективність.

Об'єктом дослідження є процес управління робочим часом та аналізу професійної активності персоналу в цифровому робочому середовищі.

Предметом дослідження є методи, алгоритми збору статистики та інтелектуальні програмні засоби побудови веб-систем моніторингу з використанням генеративного штучного інтелекту.

Практичне значення отриманих результатів полягає у створенні інноваційного програмного комплексу «Smart Time-Tracker», який пропонує новий підхід до взаємодії в команді. Для робітників система стає персональним асистентом, що надає поради та сприяє розвитку навичок самоорганізації, які є ключовими для успішної кар'єри в IT [7]. Адміністратори отримують інструменти для оперативного контролю активності без втручання в особистий простір підлеглих, а вище керівництво – глибоку аналітику та автоматизовані звіти, що дозволяє приймати стратегічні рішення на основі об'єктивних даних, візуалізованих у реальному часі. Розроблена система є масштабованою та готовою до розгортання в хмарних середовищах, що робить її універсальним рішенням для сучасного бізнесу.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ПРОЄКТНИХ РІШЕНЬ

1.1. Аналіз сучасних підходів до моніторингу робочого часу в ІТ-індустрії

Трансформація глобального ринку праці та остаточне закріплення гібридних моделей роботи висунули нові вимоги до управління людськими ресурсами в ІТ-індустрії. Питання контролю професійної діяльності перемістилося з площини фізичної присутності в офісі у площину цифрової активності. Проте традиційні методи моніторингу, що базуються на кількісних показниках (кількість рядків коду або годин «онлайн»), дедалі частіше демонструють свою неефективність, оскільки не враховують складність інтелектуальної праці та психологічний комфорт фахівців.

Сучасний підхід до оцінювання продуктивності розробників базується на розумінні того, що ефективність є багатогранною метрикою. Згідно з фреймворком SPACE, розробленим провідними фахівцями GitHub та Microsoft Research, продуктивність не може бути зведена до одного показника. Вона включає п'ять взаємопов'язаних вимірів: задоволеність та благополуччя (Satisfaction), результативність (Performance), активність (Activity), комунікацію (Communication) та ефективність робочого процесу (Efficiency) [5]. Такий підхід доводить, що інформаційна система моніторингу має не просто фіксувати час, а створювати цілісну картину робочого дня, сприяючи «потоківому стану» працівника замість створення перешкод постійними перевітками.

Паралельно з розвитком аналітичних інструментів загострюється проблема етичності цифрового нагляду. Значна частина існуючого програмного забезпечення для моніторингу персоналу використовує інвазивні методи, такі як запис натискань клавіш або автоматичне створення знімків екрана. Дослідження етичних аспектів такого ПЗ вказують на формування «павутини таємниць» між

роботодавцем та найманим працівником, що критично знижує рівень довіри в команді, провокує стрес та професійне вигорання [8].

Отже, сучасна тенденція розвитку систем моніторингу полягає у переході від парадигми «нагляду» до парадигми «підтримки та прозорості». Це передбачає розроблення інтелектуальних інструментів, які забезпечують:

- добровільну та зручну фіксацію робочих сеансів у реальному часі;
- аналітичну візуалізацію даних для менеджменту без втручання в особистий простір;
- використання штучного інтелекту як асистента для підвищення особистої ефективності спеціаліста.

Така еволюція систем контролю робочого часу дозволяє компаніям не лише отримувати об'єктивні дані для фінансового обліку, а й створювати середовище, що стимулює самодисципліну та професійне зростання кожного члена команди. Розроблення системи «Smart Time-Tracker» відповідає цим вимогам, поєднуючи класичні методи тайм-трекінгу з новітніми технологіями Business Intelligence та генеративного штучного інтелекту.

1.2. Огляд систем-аналогів та порівняння методів автоматизації звітності

Стрімкий розвиток ринку корпоративного програмного забезпечення призвів до появи великої кількості інструментів для контролю робочого часу. Загалом існуючі на ринку інформаційні системи можна розділити на дві великі категорії: системи жорсткого (інвазивного) моніторингу та системи добровільного (декларативного) трекінгу. Для обґрунтування необхідності розроблення власного продукту «Smart Time-Tracker» було проведено аналіз найбільш популярних рішень у цих категоріях: Time Doctor та Toggl Track.

Time Doctor є одним із найпопулярніших представників систем інвазивного моніторингу. Основний акцент у цій програмі зроблено на детальному відстеженні активності: система періодично робить знімки екрана (скріншоти), фіксує рухи миші, натискання клавіш та аналізує відкриті

вебсайти. Хоча такий підхід надає керівництву вичерпну інформацію про те, чим займається працівник у кожному хвилину робочого часу, він має критичні недоліки психологічного та етичного характеру. Як зазначають Ф. Лаксанаджаджа та О. Ов'єдо-Треспаласіос у своєму дослідженні (2024), надмірне використання подібного програмного забезпечення перетворює робочий процес на «павутину таємниць», порушує межі конфіденційності та стає головним фактором стресу для персоналу [8]. Це, у свою чергу, призводить до зниження загальної продуктивності та високої плинності кадрів в ІТ-командах.

Альтернативним рішенням є Toggl Track – популярний інструмент для декларативного обліку часу. Він базується на принципі довіри: працівник самостійно запускає і зупиняє таймер під час роботи над певним завданням. Програма має зручний інтерфейс та не здійснює фоновий шпигунства за користувачем. Проте для рівня середнього та великого бізнесу Toggl Track часто виявляється недостатньо функціональним. Основною проблемою є відсутність глибокої автоматизації звітності для менеджменту (наприклад, автоматичної генерації звітів про перепрацювання) та відсутність інтелектуальних інструментів для підтримки самого працівника у процесі виконання завдань.

Згідно з дослідженням Н. Шарми та П. Хосейна (2020), традиційні та ручні підходи до оцінки ефективності персоналу часто страждають від суб'єктивності та запізненої реакції менеджменту. Автори доводять, що використання автоматизованих Data-Driven (керованих даними) систем дозволяє об'єктивно розраховувати норми активності та автоматично виявляти аномалії без необхідності тотального візуального нагляду [10].

З огляду на виявлені недоліки, розроблювана система «Smart Time-Tracker» проєктується як гібридне рішення. Вона запозичує етичний (неінвазивний) підхід до трекінгу, але доповнює його потужним аналітичним BI-модулем (на базі Chart.js) та когнітивною підтримкою штучного інтелекту (Gemini API), чого немає у згаданих конкурентів. Детальне порівняння функціональних можливостей систем наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика систем моніторингу робочої активності

Критерій порівняння	Time Doctor	Toggl Track	Smart Time-Tracker (Запропонована)
Підхід до моніторингу	Інвазивний (скріншоти, запис активності)	Декларативний (ручний запуск таймера)	Декларативний + автоматизований аналіз
Інтелектуальний AI-асистент	Відсутній	Відсутній	Присутній (інтеграція з Gemini API)
Візуалізація аналітики (BI)	Стандартні таблиці	Базові графіки (у платній версії)	Інтерактивні дашборди (Chart.js)
Автоматизація звітності	Ручне вивантаження менеджером	Обмежена (потребує налаштувань API)	Повна автоматизація (Celery + SMTP)
Ступінь психологічного тиску	Високий (мікроменеджмент)	Низький	Низький (акцент на самоорганізацію)
Експорт даних	CSV, PDF	CSV, Excel	Розширений експорт у CSV

Аналіз даних таблиці 1.1 свідчить, що існуючі рішення не задовольняють у повній мірі сучасні потреби цифрового бізнесу. Time Doctor є занадто жорстким та неетичним інструментом, тоді як Toggl Track – занадто простим і позбавленим інноваційних ШІ-функцій.

Отже, розроблення інформаційної системи «Smart Time-Tracker» є обґрунтованим рішенням. Використання сучасного стеку технологій (Python,

Django, Celery) у поєднанні з генеративним штучним інтелектом дозволить створити продукт, що автоматизує рутинні управлінські процеси (наприклад, щоденну розсилку звітів) та надасть співробітникам віртуального асистента для підвищення особистої продуктивності, уникаючи при цьому деструктивного мікроменеджменту.

1.3. Дослідження можливостей застосування генеративного штучного інтелекту в корпоративних системах

Успішна реалізація інтелектуальної системи моніторингу робочої активності «Smart Time-Tracker» безпосередньо залежить від правильного вибору інструментарію, який повинен забезпечувати високу швидкість розроблення, надійність збереження даних та можливість інтеграції сучасних моделей штучного інтелекту. Основним критерієм при формуванні технологічного стеку стала здатність системи до масштабування та підтримки складних аналітичних обчислень у реальному часі.

Для розроблення серверної логіки було обрано мову програмування Python та високорівневий веб-фреймворк Django 5.0. Вибір Python обумовлений його статусом стандарту в індустрії штучного інтелекту та обробки даних, що спрощує інтеграцію з ML-сервісами. Django, у свою чергу, реалізує принцип «batteries included» (все включено), надаючи з коробки потужну систему автентифікації, ORM (Object-Relational Mapping) для роботи з базами даних та вбудовані механізми безпеки. Як зазначається у дослідженні Р. Глоснера (2024), використання стандартизованих фреймворків для управління доступом на основі ролей (RBAC) є критичним для захисту корпоративних даних від несанкціонованого втручання, що є ключовою вимогою для систем моніторингу [6].

Фундаментом для збереження інформації було обрано реляційну СУБД PostgreSQL. На відміну від легших аналогів, PostgreSQL пропонує розширені можливості для роботи зі складними аналітичними запитам та забезпечує високу цілісність даних (ACID). Це критично важливо при фіксації часових

логів, де втрата навіть одного запису може призвести до некоректного розрахунку заробітної плати або статистики активності. Застосування PostgreSQL також дозволяє ефективно масштабувати систему при зростанні кількості працівників та обсягів накопиченої статистики.

Інноваційною складовою проєкту є інтеграція з Google Generative AI (Gemini API). Вибір цієї моделі зумовлений її високою продуктивністю при обробці текстових запитів та наявністю зручного API для вбудовування в існуючі веб-додатки. Згідно зі звітом Menlo Ventures (2025), впровадження генеративного ШІ в корпоративне програмне забезпечення (Enterprise AI) наразі є найбільш динамічним трендом у світі SaaS, оскільки це дозволяє автоматизувати когнітивні завдання, які раніше потребували участі людини [12]. У системі «Smart Time-Tracker» Gemini API виконує роль інтелектуального асистента, що аналізує контекст робочих завдань та надає персоналізовані інструкції, трансформуючи трекер із пасивного інструменту обліку в активного помічника.

Для забезпечення безперебійної роботи інтерфейсу та автоматизації рутинних процесів до стеку було включено додаткові інструменти:

Celery та Redis використовуються для обробки фонових завдань, що дозволяє генерувати складні аналітичні звіти та розсилати їх через SMTP без навантаження на основний потік веб-сервера.

Chart.js обрано як основну бібліотеку для візуалізації даних на фронтенді завдяки її здатності будувати легкі, адаптивні та інтерактивні графіки (Bar та Doughnut charts).

Bootstrap 5 забезпечує сучасний вигляд системи та коректне відображення панелей керування на будь-яких пристроях (мобільна адаптивність).

Таке поєднання технологій дозволяє створити цілісну екосистему, де надійність класичних серверних рішень доповнюється гнучкістю штучного інтелекту. Це створює міцний технологічний фундамент для реалізації функціональних вимог, визначених у наступному підрозділі.

1.4. Обґрунтування вибору технологічного стеку: переваги Django, PostgreSQL та інтеграції Google Generative AI (Gemini)

Процес розроблення інтелектуальної інформаційної системи «Smart Time-Tracker» потребує чіткого визначення вимог, які забезпечують її ефективне функціонування та відповідність потребам бізнесу. Формування вимог базується на результатах проведеного системного аналізу предметної області та враховує специфіку взаємодії трьох основних груп користувачів: рядових співробітників, адміністраторів відділів та вищого керівництва компанії. На основі аналізу бізнес-процесів було виділено два ключові блоки: функціональні вимоги, що визначають безпосередні можливості системи, та нефункціональні вимоги, що регламентують її якісні характеристики.

Функціональні вимоги до системи «Smart Time-Tracker» охоплюють повний цикл управління робочим часом та інтелектуальної підтримки персоналу. Основним завданням системи є забезпечення зручного інтерфейсу для фіксації робочої активності, інтеграція з генеративним штучним інтелектом для підвищення особистої ефективності та надання глибокої аналітики для менеджменту.

До переліку ключових функціональних вимог належать:

- модуль аутентифікації та авторизації з автоматичним розпізнаванням ролей користувачів (Employee, Manager, Admin) та перенаправленням на відповідні панелі керування;
- інструментарій для трекінгу часу «в один клік» із можливістю прив'язки сеансу до конкретного проєкту або активного завдання;
- інтелектуальний AI-асистент на базі Gemini API, який автоматично аналізує опис обраного завдання та генерує персоналізовані поради чи інструкції для працівника;
- BI-модуль візуалізації даних, що забезпечує побудову інтерактивних графіків (Chart.js) для аналізу продуктивності та розподілу часу;

- система автоматизованої звітності, що здійснює генерацію звітів у форматі HTML та їх розсилку керівництву за допомогою Gmail SMTP за розкладом;
- функціонал адміністрування, що включає моніторинг працівників у реальному часі (віджет «Зараз онлайн»), керування списком завдань та можливість примусового завершення некоректних сеансів.

Нефункціональні вимоги до системи визначають її надійність, безпеку та продуктивність. Враховуючи використання стеку Django та PostgreSQL, система повинна демонструвати високу стійкість до відмов та здатність обробляти значні масиви даних без деградації швидкодії. Ключовим аспектом є безпека даних, що реалізується через систему декораторів доступу та валідацію введених параметрів. Оскільки система призначена для корпоративного використання, особлива увага приділяється адаптивності інтерфейсу (Bootstrap 5), що гарантує коректну роботу як на десктопних станціях, так і на мобільних пристроях.

Продуктивність системи забезпечується винесенням важких процесів (генерація звітів, запити до AI) у фонові завдання за допомогою Celery та Redis. Це дозволяє підтримувати низький час відгуку інтерфейсу навіть при одночасній роботі великої кількості користувачів. Окремою вимогою є масштабованість, що дозволяє легко інтегрувати нові типи завдань або мігрувати на потужніші хмарні сервери без зміни архітектурного ядра.

Завершення етапу формування вимог дозволяє перейти до безпосереднього проєктування архітектури та бази даних системи. Визначений набір функцій та технічних обмежень є фундаментом для створення інноваційного продукту, який не лише автоматизує облік часу, а й створює нову якість робочого середовища завдяки поєднанню прозорості моніторингу з інтелектуальною підтримкою кожного фахівця.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ

2.1. Моделювання бізнес-процесів взаємодії користувачів із застосуванням рольової моделі (RBAC)

Успішне розроблення будь-якої корпоративної інформаційної системи починається з детального аналізу логіки взаємодії користувачів із програмним середовищем. З огляду на специфіку «Smart Time-Tracker», система оперує конфіденційними даними про робочий час, продуктивність персоналу та внутрішні завдання компанії. Тому фундаментальним етапом проєктування є побудова архітектури безпеки та розмежування прав доступу на основі ролей (Role-Based Access Control, RBAC).

Як зазначається у дослідженні Р. Глоснера (2024), впровадження моделі RBAC є стандартом де-факто для захисту сучасних платформ від несанкціонованого доступу та забезпечення принципу найменших привілеїв (Principle of Least Privilege). Використання цієї моделі дозволяє жорстко прив'язати набір доступних прецедентів використання (Use Cases) до конкретної посадової ролі, а не до окремого користувача, що суттєво спрощує адміністрування та масштабування системи [6].

На етапі інфологічного проєктування в системі було виділено три незалежні рівні доступу, кожен з яких формує власний бізнес-процес:

- Робітник (Employee) базовий рівень доступу, орієнтований на операційну діяльність. Користувач взаємодіє з ізольованим робочим простором, де головними бізнес-процесами є ініціалізація та завершення робочих сеансів, прив'язка відпрацьованого часу до конкретних проєктних завдань та запит до інтелектуального AI-асистента (на базі Gemini) для отримання методичних підказок щодо виконання поточної задачі. Робітник має доступ лише до власної історії активності.

- Адміністратор відділу (Manager) управлінський рівень для оперативного контролю. Його бізнес-процеси охоплюють управління складом команд, створення та редагування довідника доступних завдань (із можливістю їх активації/деактивації) та моніторинг робочого процесу в реальному часі. Адміністратор отримує доступ до віджета «Зараз онлайн», який дозволяє відстежувати поточну зайнятість підлеглих без втручання в їхній робочий процес. У разі виникнення позаштатних ситуацій (наприклад, співробітник забув вимкнути таймер), адміністратор має право примусового коригування сеансів.

- Керівник (Admin/Executive) стратегічний рівень, зосереджений на Business Intelligence. Основний фокус взаємодії керівника з системою полягає в отриманні та аналізі консолідованих даних. До його прецедентів належать: доступ до глобальних дашбордів із візуалізованою статистикою (через графіки Chart.js), експорт агрегованих звітів у форматі CSV для подальшої обробки в зовнішніх ERP-системах, а також налаштування розкладу автоматизованих email-розсилок із результатами роботи відділів.

Для наочного представлення описаної архітектури розмежування прав було побудовано UML-діаграму прецедентів (Use Case Diagram), яка ілюструє межі повноважень кожного актора (рис. 2.1).

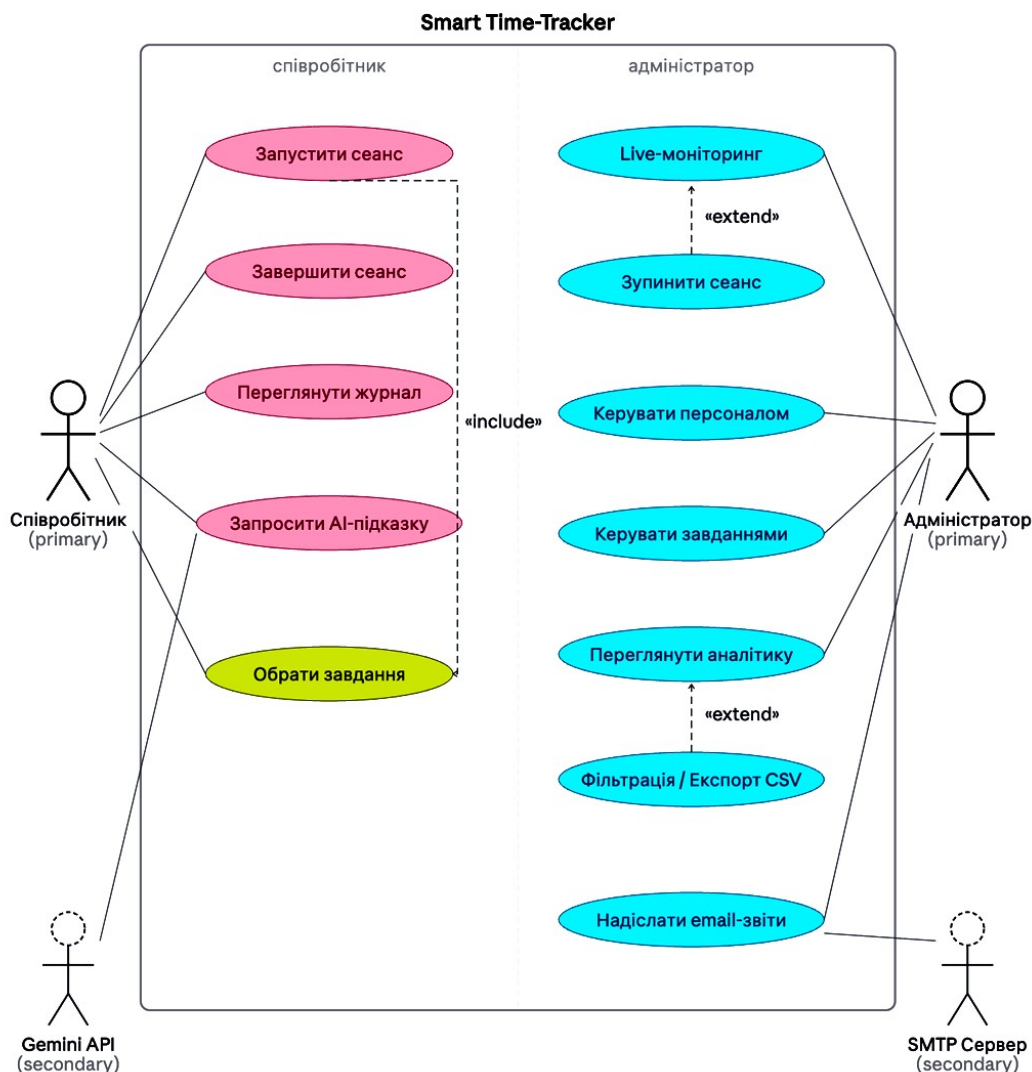


Рис. 2.1. UML-діаграму прецедентів

Окрім статичного розподілу ролей, важливим кроком є моделювання динаміки проходження типового бізнес-процесу від початку робочої зміни до формування звітності. За допомогою нотації BPMN (Business Process Model and Notation) було спроектовано головний щоденний алгоритм роботи платформи.

Процес ініціюється моментом автентифікації користувача. Система здійснює перевірку облікових даних, валідує унікальність сесії та визначає рівень доступу згідно з моделлю RBAC, після чого виконує «розумну маршрутизацію», автоматично перенаправляючи працівника на панель трекінгу, а керівника – на дашборд статистики. Протягом дня система паралельно обробляє два типи потоків: синхронний (запити користувача на створення

сеансів та отримання порад від AI) та асинхронний (фоновий трекінг тривалості та перевірка цілісності з'єднання). О 20:00 за серверним часом ініціюється фінальний етап бізнес-процесу: планувальник задач звертається до бази даних, виконує агрегацію відпрацьованих годин, рендерить HTML-шаблон звіту і передає його брокеру повідомлень для відправки керівництву.

Таким чином, застосування рольової моделі у поєднанні з детальною декомпозицією бізнес-процесів дозволило сформувати чітке бачення архітектури майбутнього додатка. Це мінімізує ризики виникнення логічних помилок на етапі написання програмного коду та створює міцне підґрунтя для переходу до безпосереднього проектування програмної архітектури та бази даних.

2.2. Проектування архітектури системи та схеми інтеграції із зовнішніми API

Архітектура інтелектуальної системи «Smart Time-Tracker» побудована на принципах модульності та асинхронності, що дозволяє поєднувати стабільну роботу веб-додатка з ресурсомісткими операціями обробки даних штучним інтелектом. В основі лежить архітектурний шаблон MVT (Model-View-Template), притаманний фреймворку Django, який забезпечує чітке розділення логіки обробки даних, бізнес-логіки та інтерфейсу користувача.

Центральним вузлом системи є веб-сервер Django, який координує взаємодію між клієнтською частиною та внутрішніми сервісами. Логіка роботи системи передбачає два основні контури обробки запитів:

Синхронний контур обробляє стандартні HTTP-запити користувачів, такі як автентифікація, вибір завдань та ініціалізація сеансів трекінгу. Взаємодія з реляційною базою даних PostgreSQL відбувається через рівень ORM, що гарантує безпеку та цілісність транзакцій.

Асинхронний контур використовується для виконання тривалих або запланованих операцій. За допомогою брокера повідомлень Redis завдання передаються до воркерів Celery. Це стосується процесів агрегації статистики за

великі періоди, формування складних HTML-звітів та їх розсилки через протокол SMTP. Такий підхід дозволяє уникнути блокування інтерфейсу користувача під час виконання важких обчислень.

Особливу увагу при проєктуванні приділено модулю інтеграції з зовнішнім API – Google Generative AI (Gemini). Схема взаємодії побудована на принципі «запит-відповідь» за протоколом HTTPS. Процес інтелектуальної підтримки працівника реалізується за наступним алгоритмом:

- система фіксує ідентифікатор обраного завдання та його текстовий опис;
- серверний модуль формує структурований промпт, додаючи до нього контекст ролі користувача та вимоги до формату відповіді;
- через офіційну бібліотеку `google-generativeai` здійснюється асинхронний виклик моделі Gemini;
- отримана відповідь проходить етап валідації на сервері та передається на фронтенд для динамічного відображення без перезавантаження сторінки (за допомогою Vanilla JS).

Для деталізації часової послідовності взаємодії компонентів системи було розроблено UML-діаграму послідовності (Sequence Diagram), яка ілюструє процес від моменту вибору завдання працівником до отримання ним інтелектуальної поради (рис. 2.3).

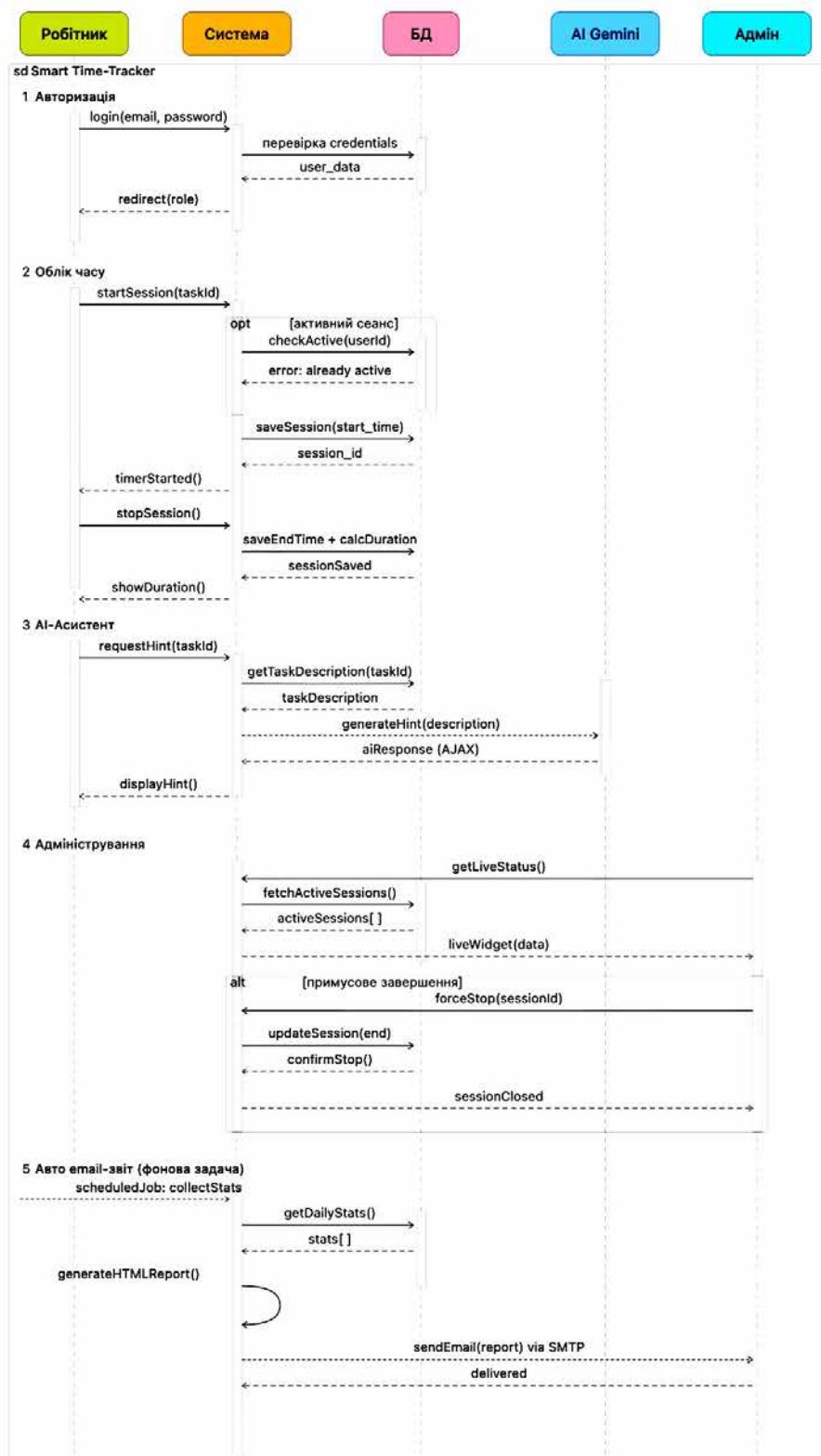


Рис. 2.3. UML Sequence Diagram.

Проектування архітектури також враховує необхідність візуалізації даних. Модуль аналітики витягує агреговані дані з PostgreSQL та готує їх у форматі JSON для бібліотеки Chart.js. Це забезпечує інтерактивність графіків на сторони

клієнта, дозволяючи користувачам перемикатися між різними метриками активності (топ-працівників, розподіл часу за проєктами) без додаткових звернень до сервера.

Запропонована архітектурна схема забезпечує високу операційну видимість системи. Використання Celery Beat як планувальника дозволяє автоматизувати критичні бізнес-процеси, такі як щоденний аудит завершених сеансів та підготовка ранкової аналітики для керівництва. Така структура є масштабованою та дозволяє в майбутньому легко замінювати окремі компоненти (наприклад, переходити на потужніші моделі ШІ) без докорінної зміни логіки всього додатка.

2.3. Інфологічне та даталогічне проєктування бази даних для зберігання сеансів та статистичних показників

Ефективність функціонування інтелектуальної системи «Smart Time-Tracker» безпосередньо залежить від раціональної структури бази даних, яка повинна забезпечувати цілісність інформації про робочий час та швидкий доступ до аналітичних показників. Процес проєктування було розділено на два етапи: створення інфологічної моделі, що описує сутності предметної області та зв'язки між ними, та розроблення даталогічної схеми, адаптованої під особливості реляційної СУБД PostgreSQL. Головним завданням цього етапу стало забезпечення масштабованості системи, що дозволить зберігати мільйони записів про активність без втрати продуктивності при виконанні складних аналітичних запитів.

На етапі інфологічного проєктування було виділено чотири ключові сутності. Перша сутність – «Користувач» – містить персональні дані працівників та їхні системні ролі, що є фундаментом для реалізації моделі доступу RBAC. Друга сутність – «Завдання» – дозволяє структурувати робочий процес за проєктами та типами активності. Третя, найбільш динамічна сутність «Робочий сеанс», фіксує часові мітки початку та завершення праці, а також обчислену тривалість активності. Окремо спроектовано сутність для збереження відповідей

від інтелектуального помічника, що дозволяє системі накопичувати історію порад, згенерованих через Gemini API, та надавати їх користувачу для повторного ознайомлення. При розробленні архітектури цих сутностей було враховано досвід проєктування масштабованих реляційних баз даних для обліку персоналу, що дозволяє ефективно структурувати ієрархічні зв'язки між працівниками та їхньою діяльністю [18].

Даталогічне проєктування полягало у трансформації концептуальної моделі у фізичну структуру таблиць PostgreSQL із визначенням типів даних та обмежень цілісності. Для кожного запису передбачено використання первинних ключів (Primary Keys) типу UUID або BigInt для гарантування унікальності в умовах великих масивів даних. Зв'язки між таблицями реалізовано через зовнішні ключі (Foreign Keys) з каскадним оновленням, що забезпечує автоматичну синхронізацію інформації. Особливу увагу приділено вибору типів даних для часових міток: використання стандарту timestamp with time zone дозволяє системі коректно враховувати географічний розподіл команд та автоматично коригувати зміну часових поясів [20]. Для оптимізації роботи BI-модуля на рівні бази даних створено індекси за полями дати та ідентифікатора користувача, що суттєво прискорює агрегацію статистики при побудові графіків у Chart.js та генерації автоматизованих звітів.

Така багаторівнева структура бази даних дозволяє системі «Smart Time-Tracker» не лише надійно зберігати поточні результати роботи, а й виступати основою для довгострокового аналізу продуктивності. Правильно спроектована реляційна модель мінімізує надмірність даних та забезпечує високу швидкість відгуку інтерфейсу, що є критично важливим для корпоративних систем моніторингу в реальному часі. Визначена схема БД стала фундаментом для подальшої програмної реалізації моделей у фреймворку Django, що описано у наступних розділах роботи.

Для наочного представлення логічної та фізичної структури розробленої бази даних було побудовано ER-діаграму (Entity-Relationship

diagram). На рисунку 2.4 відображено ключові сутності предметної області, їхні атрибути, первинні та зовнішні ключі, а також зв'язки між таблицями. Зокрема, візуалізовано базові ієрархічні зв'язки типу «один-до-багатьох» (1:N): кожен користувач системи може мати безліч створених завдань, а кожне завдання, у свою чергу, містить історію пов'язаних із ним робочих сеансів та згенерованих порад від AI-асистента. Така графічна модель наочно підтверджує, що спроектована база даних відповідає вимогам третьої нормальної форми (3NF), що гарантує відсутність дублювання даних та аномалій при їх оновленні.

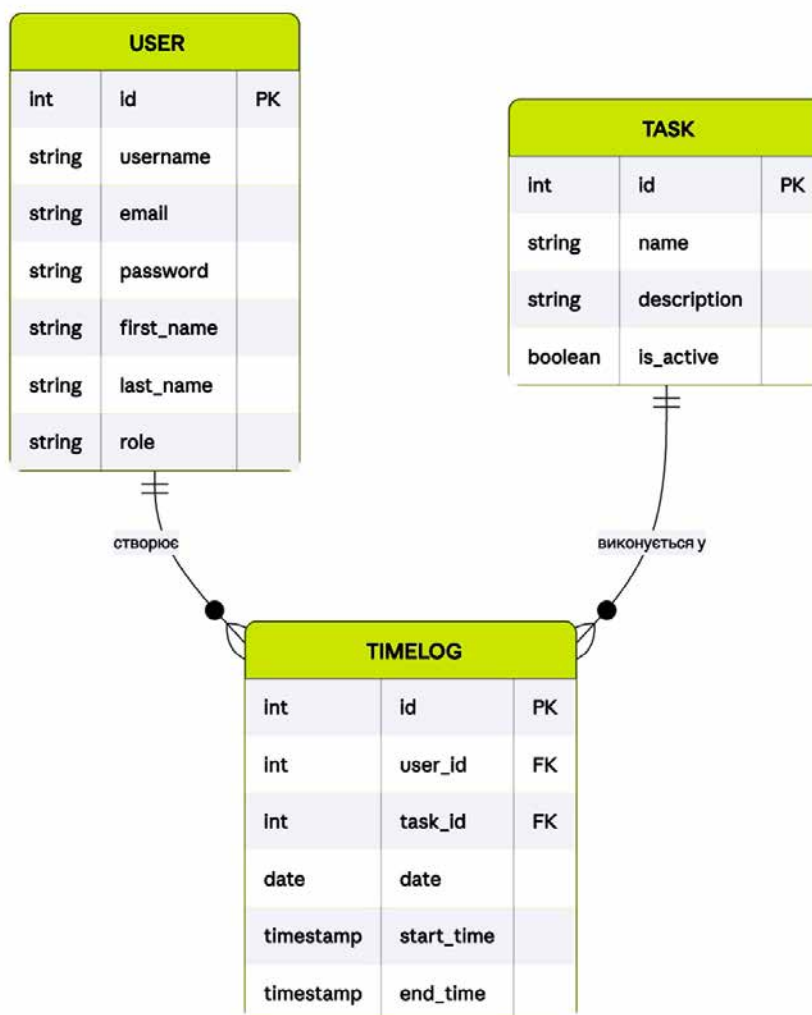


Рис. 2.4 ER-діаграма реляційної бази даних системи «Smart Time-Tracker»

2.4. Розроблення логіки роботи AI-асистента та алгоритмів візуалізації аналітичних даних

Інтеграція генеративного штучного інтелекту та засобів аналітики у систему моніторингу вимагає розроблення чітких алгоритмів обробки даних на серверній стороні. Головним завданням цього етапу є проектування логіки взаємодії між ядром системи на базі Django, зовнішнім інтерфейсом прикладного програмування та бібліотекою клієнтської візуалізації. Відповідно до сучасних досліджень у сфері великих мовних моделей (LLM), якість та безпека роботи корпоративного ШІ-асистента безпосередньо залежать від алгоритму формування контекстуального запиту, що дозволяє уникнути генерації нерелевантної або абстрактної інформації, яка не стосується робочого процесу [3].

Алгоритм роботи інтелектуального помічника ініціюється в момент вибору користувачем конкретного робочого завдання. Процес розпочинається зі збору контекстних даних: система автоматично вилучає з бази даних текстовий опис поточного завдання, посадову роль працівника та його попередню активність. Далі алгоритм виконує конкатенацію цих даних із жорстко заданим системним промптом (System Prompt). Цей прихований системний інструктаж обмежує поведінку мовної моделі, змушуючи її діяти виключно у ролі корпоративного консультанта з продуктивності. Сформований інформаційний пакет передається через захищене HTTPS-з'єднання до хмарного сервісу Google Generative AI. Після отримання відповіді від моделі алгоритм здійснює базову санітизацію тексту, перевіряючи його на наявність некоректних символів форматування, і повертає готовий результат на клієнтську сторону. Такий підхід забезпечує високу релевантність порад та відповідає офіційним рекомендаціям щодо безпечного використання Gemini API [19].

Логіка візуалізації аналітичних даних також базується на алгоритмах попередньої агрегації інформації на рівні бекенду. Оскільки перенесення сирих масивів даних про кожну робочу секунду на клієнтську сторону є вкрай

неефективним з точки зору використання оперативної пам'яті браузера, було розроблено алгоритм серверного зведення показників. Процес візуалізації починається із запиту адміністратора на формування дашборду за певний проміжок часу. Сервер ініціює звернення до реляційної бази даних PostgreSQL, використовуючи математичні функції групування за ідентифікатором працівника або типом проєкту. Алгоритм розраховує загальну суму відпрацьованих секунд для кожної групи, після чого конвертує ці значення у формат годин та хвилин, зручний для людського сприйняття.

Завершальним кроком алгоритму візуалізації є трансформація агрегованих даних у структуру формату JSON, сумісну з вимогами графічної бібліотеки Chart.js. Система автоматично формує два взаємопов'язані масиви: набір текстових міток, що містить назви завдань або прізвища співробітників, та масив відповідних числових значень тривалості роботи. Окрім цього, для забезпечення високої візуальної контрастності на кільцевих та стовпчастих діаграмах, алгоритм динамічно генерує масив кольорів у форматі HEX. Згенерований об'єкт конфігурації передається у HTML-шаблон, де фронтенд-скрипт здійснює лише фінальне відмальовування графіки. Такий архітектурний розподіл обчислювального навантаження між базою даних, сервером та клієнтським пристроєм забезпечує високу продуктивність системи навіть при аналізі річної статистики цілого відділу підприємства.

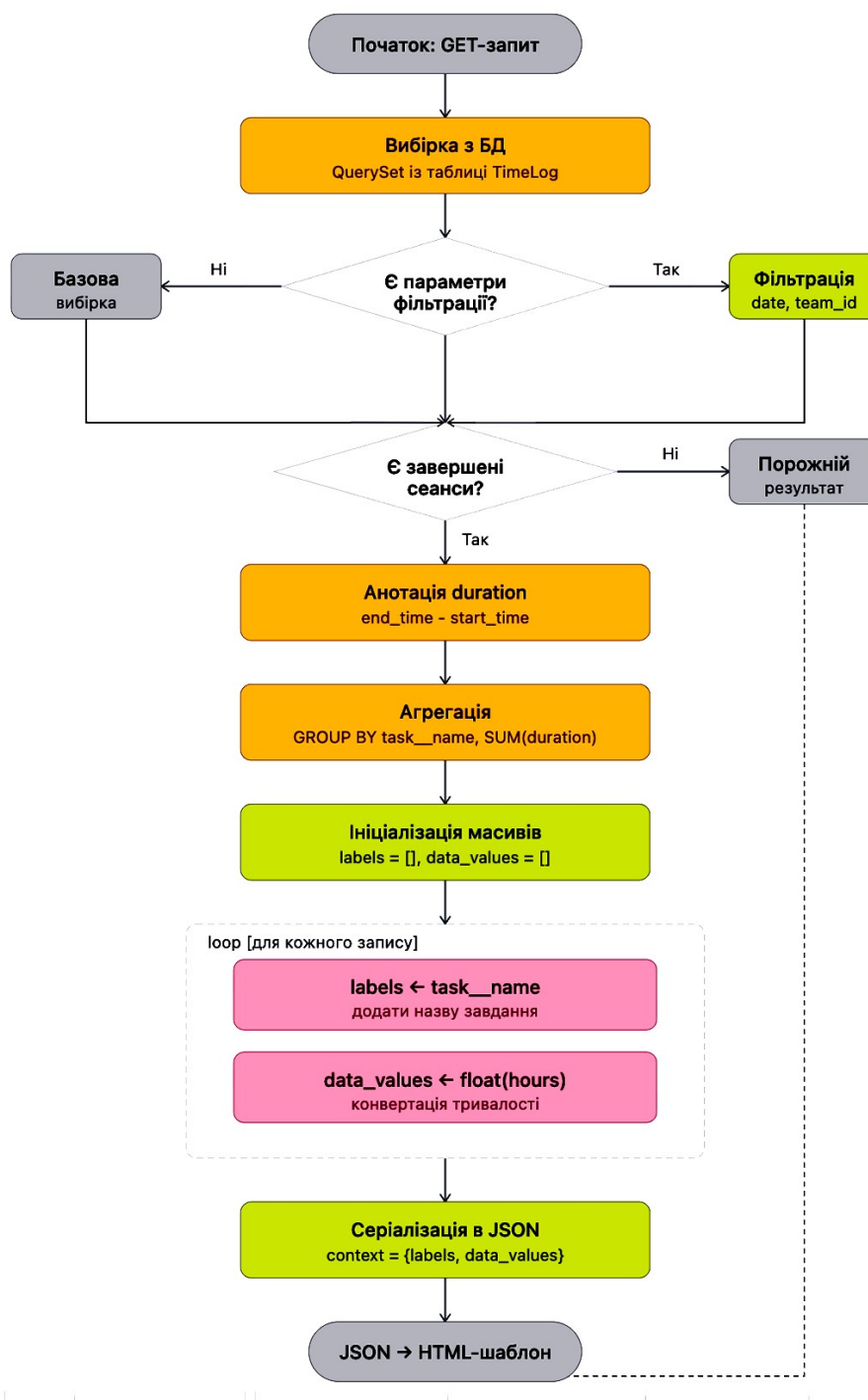
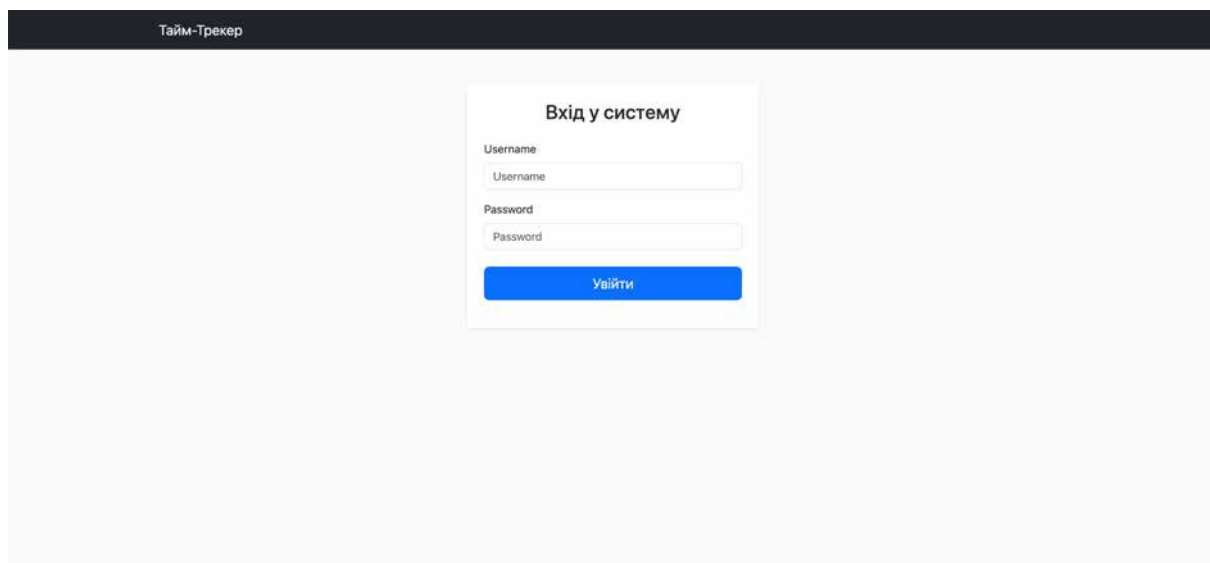


Рис. 2.5. Блок-схема алгоритму агрегації даних для формування конфігураційного об'єкта Chart.js

Важливим етапом проєктування інтерфейсу стала розробка сторінки автентифікації та реєстрації, яка є «вхідними воротами» до системи. Оскільки безпека корпоративних даних є пріоритетом, форму входу було спроектовано з використанням стандартних компонентів Bootstrap 5, що забезпечує чітку

візуальну валідацію введених даних. Дизайн передбачає лаконічне розміщення полів для введення електронної пошти та пароля, що дозволяє користувачеві зосередитися на процесі доступу без зайвих відволікань. На рисунку 2.5 продемонстровано спроектований макет сторінки входу, де реалізовано механізм перевірки облікових даних та автоматичного розподілу користувачів за їхніми ролями в системі.



Тайм-Трекер

Вхід у систему

Username
Username

Password
Password

Увійти

Рис. 2.6. Макет сторінки автентифікації користувачів у системі «Smart Time-Tracker»

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ОРІЄНТОВАНОГО ДОДАТКА

3.1. Розроблення бекенд-логіки на базі Django: керування сеансами, фільтрація та пагінація даних

Програмна реалізація серверної частини інтелектуальної системи «Smart Time-Tracker» базується на використанні високорівневого вебфреймворку Django версії 5.0, що дозволяє забезпечити високу швидкість обробки запитів та безпеку корпоративних даних. В основі розроблення лежить архітектурний патерн MVT (Model-View-Template), де основне обчислювальне навантаження та бізнес-логіка зосереджені у серверних представленнях (views). Важливим аспектом реалізації бекенду є забезпечення надійної автентифікації та розмежування прав доступу. Згідно з дослідженням Р. Глоснера (2024), використання стандартизованих механізмів авторизації на базі фреймворків дозволяє мінімізувати ризики несанкціонованого доступу до персональних логів активності, що реалізовано в системі через вбудовану модель користувачів Django та систему декораторів доступу [6].

Центральним модулем серверної логіки є контролер керування робочими сеансами, який забезпечує точну фіксацію часових інтервалів професійної діяльності. Програмно це реалізовано через обробку POST-запитів, де при старті таймера створюється новий екземпляр моделі сесії з прив'язкою до конкретного завдання та поточної мітки часу сервера. При завершенні роботи алгоритм автоматично розраховує тривалість сеансу як різницю між часом зупинки та старту, враховуючи при цьому можливі розбіжності часових поясів завдяки підтримці timezone-aware об'єктів. На рис. 3.1 продемонстровано фрагмент програмного коду контролера, що відповідає за логіку валідації та збереження сеансу, включаючи перевірку на наявність уже відкритих активних записів для запобігання дублюванню часу.

```

47 @login_required 1 usage
48 def start_work(request):
49     if not TimeLog.objects.filter(user=request.user, end_time__isnull=True).exists():
50         TimeLog.objects.create(user=request.user)
51     return redirect('dashboard')
52
53 @login_required 1 usage
54 def stop_work(request):
55     active_log = TimeLog.objects.filter(user=request.user, end_time__isnull=True).first()
56     if active_log:
57         active_log.end_time = timezone.now()
58         active_log.save()
59     return redirect('dashboard')
60
61
62 @login_required 1 usage
63 def start_work(request):
64     if request.method == 'POST':
65         form = StartWorkForm(request.POST)
66         if form.is_valid():
67             # Перевіряємо, чи немає вже активної сесії
68             if not TimeLog.objects.filter(user=request.user, end_time__isnull=True).exists():
69                 selected_task = form.cleaned_data['task']
70                 TimeLog.objects.create(user=request.user, task=selected_task)
71     return redirect('dashboard')
72

```

Рис. 3.1. Фрагмент програмної реалізації логіки керування сесіями у файлі views.py

Оскільки корпоративна база даних PostgreSQL з часом накопичує значні обсяги інформації, критичним завданням стала реалізація ефективних механізмів вибірки та відображення даних. Для цього було розроблено підсистему динамічної фільтрації на основі об'єктів QuerySet, яка дозволяє адміністраторам та керівникам відсіювати записи за кількома параметрами одночасно: прізвищем працівника, назвою проєкту або конкретним календарним періодом. Для уникнення деградації продуктивності при виконанні записів до пов'язаних таблиць (наприклад, при отриманні назви завдання для кожного сеансу) на програмному рівні застосовано методи оптимізації `select_related`, що дозволяє виконувати складні об'єднання на стороні бази даних одним SQL-запитом, як це рекомендується офіційною документацією Django для високонавантажених систем [17].

Для забезпечення зручності взаємодії з великими масивами статистичних даних в адміністративній панелі та звітах впроваджено механізм пагінації. Програмна реалізація на базі класу Paginator дозволяє розбивати списки сеансів

на окремі сторінки, передаючи на клієнтську сторону лише обмежену кількість об'єктів за один раз. Це суттєво знижує навантаження на оперативну пам'ять сервера та прискорює рендеринг сторінки в браузері користувача. Бекенд-логіка автоматично обробляє GET-параметри запиту, валідує номер поточної сторінки та формує навігаційні посилання для переходу між частинами списку. На рис. 3.2 представлено візуалізацію результатів роботи модулів фільтрації та пагінації в інтерфейсі управління історією сеансів, що забезпечує високу операційну видимість для менеджменту підприємства.

```

74 @user_passes_test(is_manager_or_admin) 2 usages
75 def admin_dashboard(request):
76     date_str = request.GET.get('date')
77     search_query = request.GET.get('search', '')
78     sort_by = request.GET.get('sort', '')
79     order = request.GET.get('order', 'asc')
80
81     # БАЗОВИЙ ЗАПИТ: Беремо всі записи, а не лише за сьогодні
82     logs = TimeLog.objects.all()
83     active_sessions = TimeLog.objects.filter(end_time__isnull=True)
84
85     if request.user.role == 'TEAM_ADMIN':
86         logs = logs.filter(user__team=request.user.team)
87         active_sessions = active_sessions.filter(user__team=request.user.team)
88
89     # 1. Фільтр за датою ТІЛЬКИ якщо вона обрана керівником
90     selected_date = None
91     if date_str:
92         selected_date = datetime.strptime(date_str, format: '%Y-%m-%d').date()
93         logs = logs.filter(date=selected_date)
94
95     # 2. Пошук
96     if search_query:
97         logs = logs.filter(
98             Q(user__first_name__icontains=search_query) |
99             Q(user__last_name__icontains=search_query) |
100             Q(user__username__icontains=search_query)
101         )
102
103     # 3. Анотація для розрахунку тривалості
104     logs = logs.annotate(
105         calc_duration=ExpressionWrapper(
106             Coalesce(*expressions: 'end_time', Now()) - F('start_time'),
107             output_field=fields.DurationField()
108         )
109 )

```

```

111 # 4. Сортування (за замовчуванням: найновіші зверху)
112 if sort_by == 'worker':
113     order_prefix = '-' if order == 'desc' else ''
114     logs = logs.order_by(*field_names: f'{order_prefix}user__first_name', f'{order_prefix}user__username')
115 elif sort_by == 'task':
116     order_prefix = '-' if order == 'desc' else ''
117     logs = logs.order_by(f'{order_prefix}task__name')
118 elif sort_by == 'duration':
119     order_prefix = '-' if order == 'desc' else ''
120     logs = logs.order_by(f'{order_prefix}calc_duration')
121 else:
122     logs = logs.order_by('-start_time')
123
124 paginator = Paginator(logs, per_page: 10)
125 page_number = request.GET.get('page')
126 page_obj = paginator.get_page(page_number)
127
128 get_params = request.GET.copy()
129 if 'page' in get_params:
130     del get_params['page']
131 url_params = get_params.urlencode()
132
133 def get_next_order(target_col):
134     if sort_by == target_col:
135         return 'asc' if order == 'desc' else 'desc'
136     return 'desc' if target_col == 'duration' else 'asc'
137
138 context = {
139     'page_obj': page_obj, # Тепер передаємо сторінку замість всіх логів
140     'active_sessions': active_sessions.order_by('-start_time'),
141     'tasks': Task.objects.all(),
142     'selected_date': selected_date.strftime('%Y-%m-%d') if selected_date else '',
143     'search_query': search_query,
144     'sort_by': sort_by,
145     'order': order,
146     'next_order_worker': get_next_order('worker'),
147     'next_order_task': get_next_order('task'),
148     'next_order_duration': get_next_order('duration'),
149     'url_params': url_params, # Параметри для кнопок пагінації
150 }

```

Рис. 3.2 Реалізація фільтрації та пагінації в інтерфейсі перегляду історії робочих сеансів

Таким чином, розроблена бекенд-архітектура на базі Django 5.0 створює надійний фундамент для роботи всієї системи. Поєднання суворої моделі безпеки, оптимізованих запитів до БД та ефективних засобів представлення великих даних дозволяє системі «Smart Time-Tracker» стабільно функціонувати в умовах постійного зростання обсягу корпоративної інформації. Спроектowana логіка серверних представлень стала основою для подальшої реалізації клієнтського інтерфейсу та інтеграції інтелектуальних модулів підтримки персоналу.

3.2. Реалізація клієнтського інтерфейсу за допомогою Bootstrap 5 та інтерактивних графіків Chart.js

Розроблення клієнтської частини системи «Smart Time-Tracker» базується на принципах створення адаптивного та інтуїтивно зрозумілого інтерфейсу, який забезпечує комфортну роботу користувачів на різних типах пристроїв. Для реалізації візуальної складової було обрано фреймворк Bootstrap 5, що дозволило побудувати гнучку сітку (grid system) та інтегрувати готові компоненти для навігації та управління формами. Основним архітектурним рішенням у дизайні інтерфейсу став поділ на функціональні блоки, що дозволяє працівникам швидко отримувати доступ до таймера та AI-асистента, а керівникам – до аналітичних дашбордів. Як підкреслюють у своєму дослідженні В. Страй та співавтори (2026), якісний інтерфейс програмного забезпечення для розробників безпосередньо впливає на їхній суб'єктивний досвід продуктивності, тому мінімізація візуального шуму та швидкість відгуку елементів UI стали пріоритетними завданнями при верстці фронтенду [11]. На рис. 3.3 представлено загальний вигляд адаптивної головної сторінки додатка, де реалізовано навігаційну панель з автоматичним підсвічуванням поточної ролі користувача.

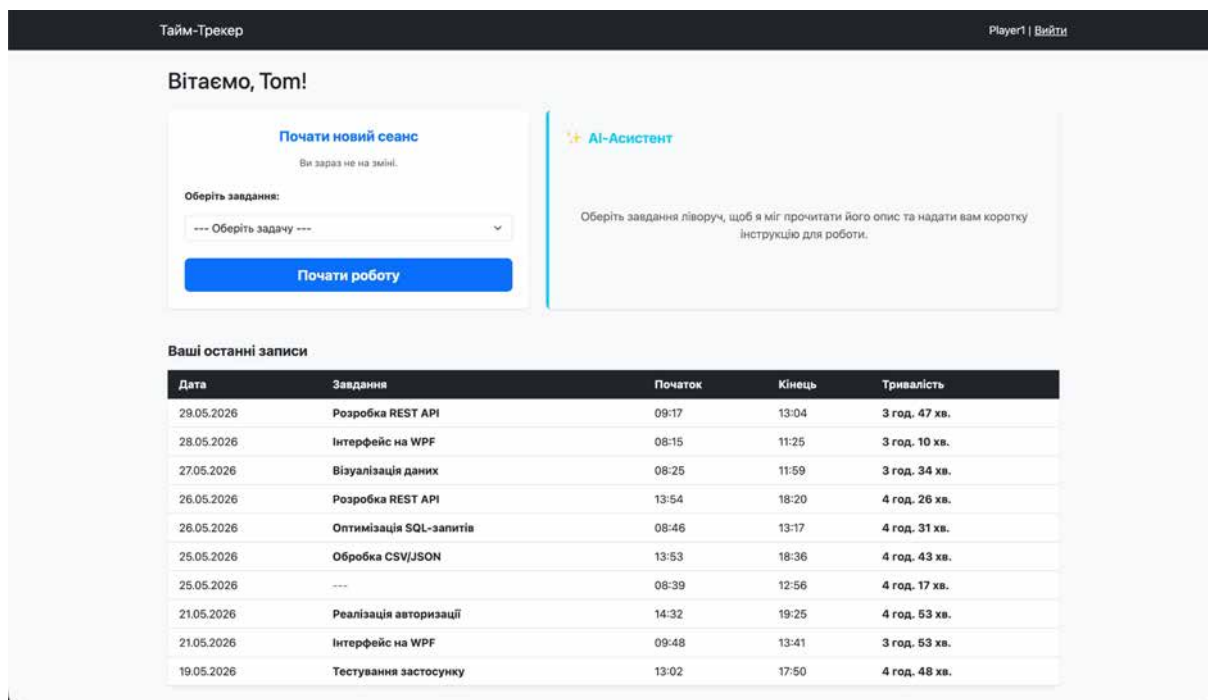


Рис. 3.3. Інтерфейс головної сторінки системи «Smart Time-Tracker» із застосуванням адаптивної верстки Bootstrap 5

Особлива увага в процесі програмної реалізації була приділена ВІ-модулю, який відповідає за візуалізацію результатів моніторингу робочої активності. Для створення інтерактивних звітів було інтегровано бібліотеку Chart.js, яка працює на базі елементів HTML5 Canvas. Процес побудови графіків починається з отримання на стороні клієнта структурованого JSON-об'єкта від Django-контролера, що містить назви проєктів та сумарний відпрацьований час у годинах. Використання такої технології дозволяє реалізувати динамічне перемикання між різними режимами відображення без необхідності перезавантаження всієї вебсторінки. Е. Огбуефі та А. Мгбаме (2024) зазначають, що візуалізація даних у реальному часі за допомогою динамічних дашбордів є критично важливою для оперативного управління в сучасних компаніях, оскільки вона перетворює розрізнені логи в наочні інструменти підтримки прийняття рішень [9]. Приклад реалізованої аналітичної панелі з використанням стовпчастих та кільцевих діаграм наведено на рис. 3.4, де відображено розподіл робочого часу команди за тиждень.

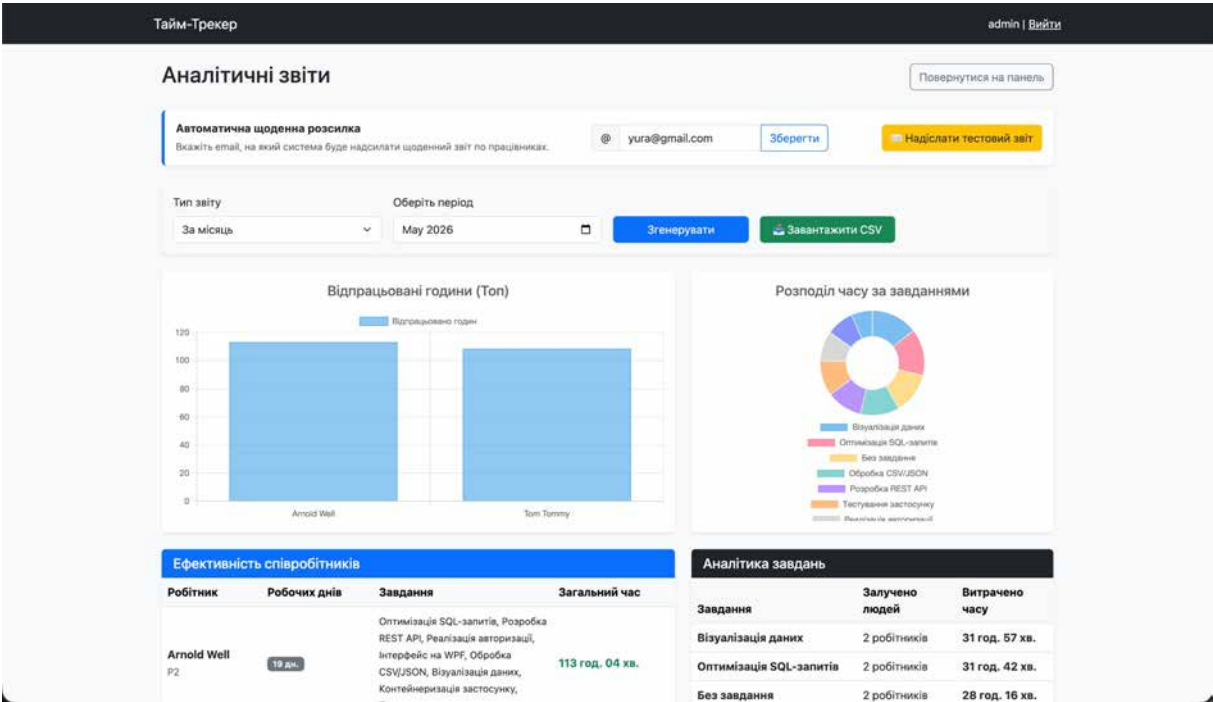


Рис. 3.4. Візуалізація аналітичних даних у реальному часі засобами бібліотеки Chart.js

Інтерактивність інтерфейсу підсилюється за допомогою базового JavaScript (Vanilla JS), який забезпечує зв'язок між діями користувача та фоновими процесами. Зокрема, при запуску таймера скрипт ініціює відлік часу в браузері та паралельно надсилає асинхронний запит на сервер для фіксації початку сеансу в базі даних. Такий підхід дозволяє відображати актуальний стан активності в реальному часі, не створюючи надлишкового навантаження на мережевий канал. Окрім візуальної привабливості, використання Chart.js та Bootstrap 5 дозволило досягти високої швидкості рендерингу, що є важливим для підтримки безперервного робочого процесу та позитивного сприйняття системи як співробітниками, так і адміністративним персоналом підприємства. На рис. 3.5 продемонстровано механізм візуального відгуку системи при отриманні порад від інтелектуального помічника, що реалізовано у вигляді модального вікна з підтримкою Markdown-форматування тексту.

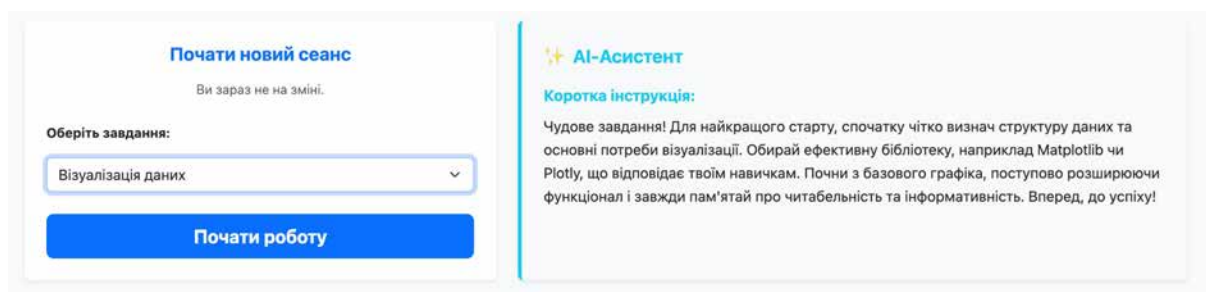


Рис. 3.5. Програмна реалізація блоку відображення відповідей AI-асистента в інтерфейсі користувача

Практична реалізація головного робочого простору працівника (Dashboard) фокусується на зручності управління поточними завданнями. Центральним елементом інтерфейсу є інтерактивна картка завдання, де відображається його назва, опис та поточний статус. Програмно реалізовано динамічну зміну кольору кнопок залежно від стану таймера: при неактивному сеансі кнопка «Почати» має зелене забарвлення, а під час активного відліку вона трансформується у червону кнопку «Зупинити». На рисунку 3.6 можна побачити

реалізацію робочого кабінету розробника, де в реальному часі працює лічильник годин, хвилин та секунд, ініційований JavaScript-скриптом на стороні клієнта.

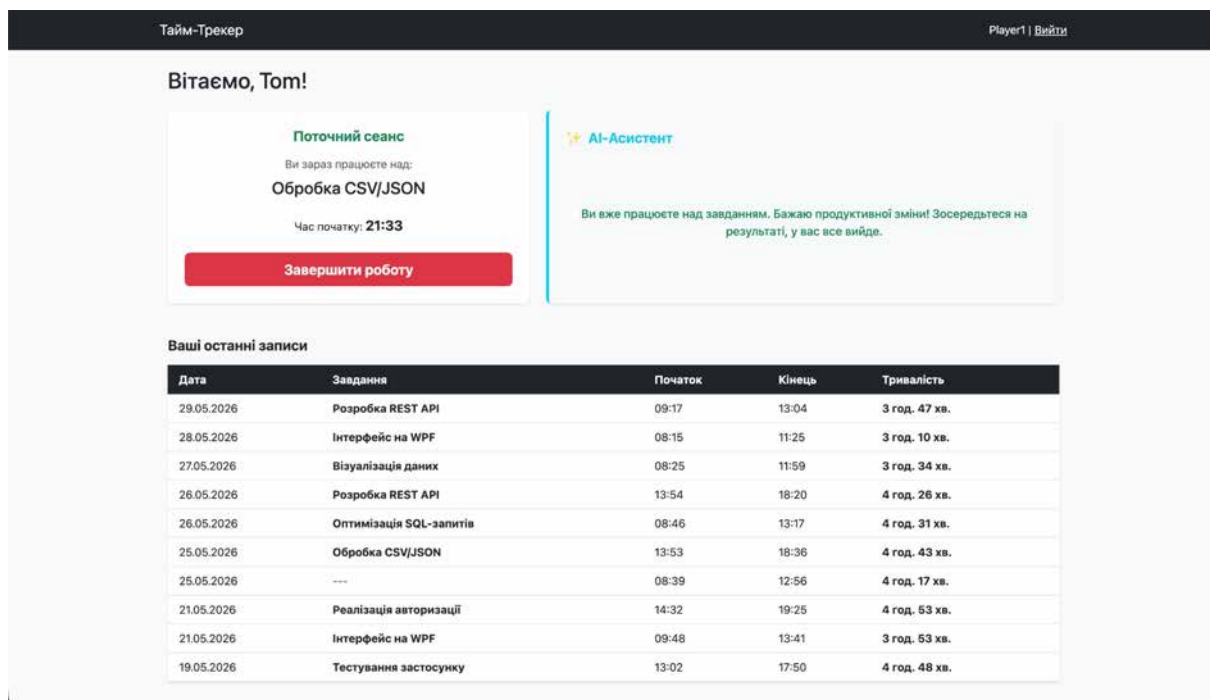


Рис. 3.6. Реалізація робочого простору співробітника з активним таймером сеансу

3.3. Практична реалізація інтеграції з Gemini API для створення інтелектуального помічника

Реалізація віртуального AI-асистента є однією з ключових інновацій системи «Smart Time-Tracker», що перетворює її зі звичайного інструменту фіксації часу на активного помічника. Програмна взаємодія між сервером Django та хмарною інфраструктурою штучного інтелекту базується на використанні офіційного Python-пакета google-generativeai. Оскільки корпоративні системи вимагають суворого дотримання політик конфіденційності, архітектура модуля побудована таким чином, щоб секретні ключі доступу (API Keys) ніколи не зберігалися у вихідному коді. Авторизація запитів до платформи Google здійснюється виключно через безпечне читання змінних оточення (environment variables) сервера. Згідно з дослідженнями впровадження генеративного ШІ в

Enterprise-сегменті, такий підхід є індустріальним стандартом, що мінімізує ризики компрометації доступу до комерційних мовних моделей [12].

Основою програмної логіки асистента є механізм контекстного пром프트-інжинірингу (Prompt Engineering), реалізований на рівні серверних представлень (Views). Коли користувач на клієнтській стороні обирає завдання та натискає кнопку виклику помічника, ініціюється асинхронний POST-запит до відповідного маршруту Django. Сервер за допомогою засобів ORM отримує з бази даних PostgreSQL текстовий опис обраного завдання. Далі алгоритм динамічно формує комплексний запит до моделі Gemini. Цей запит складається з двох частин: користувацького контексту (суть завдання) та прихованого системного інструктажу (System Prompt). Системні інструкції жорстко обмежують поведінку моделі, наказуючи їй генерувати виключно лаконічні, професійні поради щодо підвищення продуктивності та тайм-менеджменту. Як зазначається в офіційній документації Gemini API, надання моделі чіткої рольової установки є критично важливим для уникнення генерації «галюцинацій» та нерелевантного контенту [19].


```

545 @login_required 1 usage
546 def get_task_advice(request):
547     task_id = request.GET.get('task_id')
548     if not task_id:
549         return JsonResponse( data: {'error': 'Не вказано ID завдання'}, status=400)
550
551     try:
552         task = Task.objects.get(id=task_id)
553
554         genai.configure(api_key="AIzaSyABNy3N0eo_RLZRNT_lNdJfvpQiq3XKP9M")
555
556         best_model_name = None
557
558         for m in genai.list_models():
559             if 'generateContent' in m.supported_generation_methods:
560                 # Шукаємо найшвидшу модель (flash)
561                 if 'flash' in m.name:
562                     best_model_name = m.name
563                     break
564
565         if not best_model_name:
566             for m in genai.list_models():
567                 if 'generateContent' in m.supported_generation_methods:
568                     best_model_name = m.name
569                     break
570
571         if not best_model_name:
572             return JsonResponse(
573                 data: {'error': 'Ваш API-ключ не має доступу до жодної текстової моделі. Спробуйте створити новий ключ.'},
574                 status=500)
575
576         print(f"=== AI використовує модель: {best_model_name} ===") # Для перевірки в терміналі
577
578         model = genai.GenerativeModel(best_model_name)
579
580         print(f"=== AI використовує модель: {best_model_name} ===") # Для перевірки в терміналі
581
582         model = genai.GenerativeModel(best_model_name)
583
584         description = task.description if task.description else 'Опис відсутній'
585         prompt = f"""
586         Ти - професійний помічник. Робітник збирається почати роботу над завданням: "{task.name}".
587         Опис завдання від керівника: "{description}".
588         Напиши дуже коротку, чітку та підбадьорливу інструкцію (до 4 речень), як найкраще та найефективніше виконати це завдання.
589         Не використовуй форматування Markdown, пиши простим текстом.
590         """
591
592         response = model.generate_content(prompt)
593         return JsonResponse({'advice': response.text})
594
595     except Task.DoesNotExist:
596         return JsonResponse( data: {'error': 'Завдання не знайдено'}, status=404)
597     except Exception as e:
598         print(f"=== ДЕТАЛЬНА ПОМИЛКА AI: {e} ===")
599         return JsonResponse( data: {'error': f'Детані: {str(e)}'}, status=500)

```

Рис. 3.7. Лістинг серверного коду формування запиту до Gemini API

Особливу увагу в процесі розроблення було приділено механізмам відмовостійкості (Fault Tolerance). Оскільки система залежить від зовнішнього хмарного сервісу, будь-які мережеві затримки або перевищення квот доступу могли б призвести до критичного збою веб-додатка. Для уникнення цього виклик методу генерації тексту (`model.generate_content`) загорнуто у строгий блок обробки винятків (`try-except`). У разі виникнення помилки на боці Google,

серверна логіка перехоплює виняток і повертає на фронтенд коректно відформатований JSON із повідомленням про тимчасову недоступність асистента, не перериваючи при цьому основний процес обліку часу.

У разі успішної генерації поради бекенд здійснює базове очищення отриманого тексту (санітизацію), після чого формує відповідь у форматі JSON. На клієнтській стороні скрипт, написаний на Vanilla JS, перехоплює цю відповідь та динамічно оновлює вміст спеціального модального вікна або віджета у робочій області працівника. Використання асинхронного обміну даними (AJAX) забезпечує миттєву появу III-підказок без необхідності перезавантаження вебсторінки, що створює ілюзію безперервного «живого» спілкування з інтелектуальним ядром системи та значно покращує загальний користувацький досвід.

3.4. Автоматизація розсилки аналітичних звітів засобами SMTP та експорту даних у формат CSV

Сучасні корпоративні інформаційні системи вимагають високого рівня автономності, особливо в питаннях інформування керівництва про поточний стан робочих процесів. Для забезпечення проактивного моніторингу в системі «Smart Time-Tracker» спроектовано та реалізовано підсистему автоматизованої генерації та розсилки аналітичних звітів. Фундаментом для програмної реалізації цієї підсистеми став механізм асинхронних фонових завдань, побудований на базі брокера повідомлень Redis та виконавця черг Celery. Згідно з офіційною технічною документацією інструментарію Celery, використання вбудованого планувальника завдань (Celery Beat) дозволяє ініціювати виконання ресурсомістких скриптів за чітко визначеним розкладом на основі синтаксису crontab, повністю ізолюючи ці процеси від основного потоку вебсервера [15]. Це архітектурне рішення гарантує, що процес важкої агрегації даних за добу для сотень співробітників жодним чином не вплине на швидкість відгуку інтерфейсу для тих користувачів, які в цей момент продовжують взаємодіяти із системою.

Процес формування щоденного звіту ініціюється автоматично о визначеній годині. Серверна логіка виконує складний запит до бази даних PostgreSQL, який консолідує інформацію про загальний відпрацьований час кожного члена команди, розподіл годин за проєктами та виявляє можливі аномалії, такі як критичні перепрацювання. Отримані масиви даних передаються до шаблонізатора Django, який рендерить їх у візуально привабливий HTML-документ із застосуванням інлайнових CSS-стилів, адаптованих для коректного відображення у поштових клієнтах. Після генерації документа система використовує протокол SMTP (зокрема, інтеграцію з серверами Gmail) для безпечної передачі сформованого листа на електронні адреси адміністраторів. Такий підхід до максимальної автоматизації операційних процесів повністю відповідає сучасним індустріальним тенденціям розгортання гібридних хмарних рішень, де зведення до мінімуму ручного втручання в рутинні завдання збору метрик призводить до суттєвого зниження загальних експлуатаційних витрат та підвищення управлінської надійності [2]. На рисунку 3.8 наведено фрагмент коду, що демонструє логіку формування HTML-повідомлення та його передачу через SMTP-бекенд Django всередині асинхронного завдання Celery.

```

619 @shared_task
620 def send_daily_reports_task():
621     today = timezone.now().date()
622
623     # 1. Знаходимо всіх керівників та адмінів, у яких вказано email
624     managers = User.objects.filter(
625         role__in=['MANAGER', 'TEAM_ADMIN'],
626         email__isnull=False
627     ).exclude(email='')
628
629     reports_sent = 0
630
631     # 2. Формуємо персональний звіт для кожного керівника
632     for manager in managers:
633         logs = TimeLog.objects.filter(date=today)
634
635         # Якщо це адмін конкретної команди, фільтруємо дані лише для його команди
636         if manager.role == 'TEAM_ADMIN' and manager.team:
637             logs = logs.filter(user__team=manager.team)
638
639         # Якщо за сьогодні немає записів, пропускаємо розсилку
640         if not logs.exists():
641             continue
642
643         # Обчислення тривалості відпрацьованого часу
644         logs = logs.annotate(
645             duration_val=ExpressionWrapper(
646                 Coalesce(*expressions: 'end_time', Now()) - F('start_time'),
647                 output_field=fields.DurationField()
648             )
649         )
650
651         worker_stats = logs.values(*fields: 'user__first_name', 'user__last_name').annotate(
652             total_time=Sum('duration_val')
653         ).order_by('-total_time')
654
655     # 3. HTML-шаблонізація листа
656     html_content = f"""
657     <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;">
658     <h2 style="color: #0d6efd;">Щоденний звіт за {today.strftime('%d.%m.%Y')}</h2>
659     <p>Вітаємо, {manager.first_name}! Ось зведення по відпрацьованих годинах вашої команди:</p>
660     <table style="width: 100%; border-collapse: collapse; margin-top: 20px;">
661     <thead>
662     <tr style="background-color: #f8f9fa;">
663     <th style="padding: 10px; border: 1px solid #dee2e6; text-align: left;">Робітник</th>
664     <th style="padding: 10px; border: 1px solid #dee2e6; text-align: right;">Відпрацьовано</th>
665     </tr>
666     </thead>
667     <tbody>
668     """
669
670     for w in worker_stats:
671         total_seconds = int(w['total_time'].total_seconds()) if w['total_time'] else 0
672         hours = total_seconds // 3600
673         minutes = (total_seconds % 3600) // 60
674         html_content += f"""
675         <tr>
676         <td style="padding: 10px; border: 1px solid #dee2e6;">{w['user__first_name']} {w['user__last_name']}</td>
677         <td style="padding: 10px; border: 1px solid #dee2e6; text-align: right;"><strong>{hours}</strong> год. {minutes:02d} хв.</td>
678         </tr>
679         """
680
681     html_content += f"""
682     </tbody>
683     </table>
684     <p style="margin-top: 30px; font-size: 12px; color: #6c757d;">Згенеровано автоматично системою Time-Tracker.</p>
685     </div>
686     """

```

```

688 # 4. Формування та відправка email-повідомлення
689 text_content = strip_tags(html_content)
690
691 msg = EmailMultiAlternatives(
692     subject=f"📊 Підсумки дня - {today.strftime('%d.%m.%Y')}",
693     body=text_content,
694     from_email=settings.DEFAULT_FROM_EMAIL,
695     to=[manager.email]
696 )
697 msg.attach_alternative(html_content, mimetype: "text/html")
698
699 try:
700     msg.send()
701     reports_sent += 1
702 except Exception as e:
703     print(f"Помилка відправки для {manager.email}: {e}")
704
705 return f"Успішно надіслано звітів: {reports_sent}"

```

Рис. 3.8. Фрагмент коду асинхронного завдання для автоматичної генерації та відправки email-звітів

Окрім регулярних автоматизованих розсилок, архітектура додатка передбачає можливість вивантаження «сирих» статистичних даних за запитом (On-Demand) для їх подальшого використання у зовнішніх аналітичних програмах або корпоративних ERP-системах. Для вирішення цього завдання реалізовано модуль експорту історії сеансів у формат CSV (Comma-Separated Values). Програмна реалізація базується на використанні стандартної бібліотеки Python csv. При зверненні керівника до відповідного маршруту (URL) сервер генерує об'єкт `HttpResponse` із заданим MIME-типом `text/csv` та HTTP-заголовком `Content-Disposition`, що вказує браузеру на необхідність завантаження файлу на локальний диск клієнта. Алгоритм бекенду ітерує через відфільтрований набір даних, послідовно записуючи кожен робочий сеанс як окремий рядок файлу з урахуванням кодування UTF-8 для коректного збереження кирилических символів. Цей механізм працює за принципом потокової передачі даних (streaming), що робить його максимально економним щодо споживання оперативної пам'яті сервера навіть при експорті багаторічної історії компанії. Результат роботи модуля розсилки у вигляді отриманого корпоративного звіту продемонстровано на рисунку 3.9.

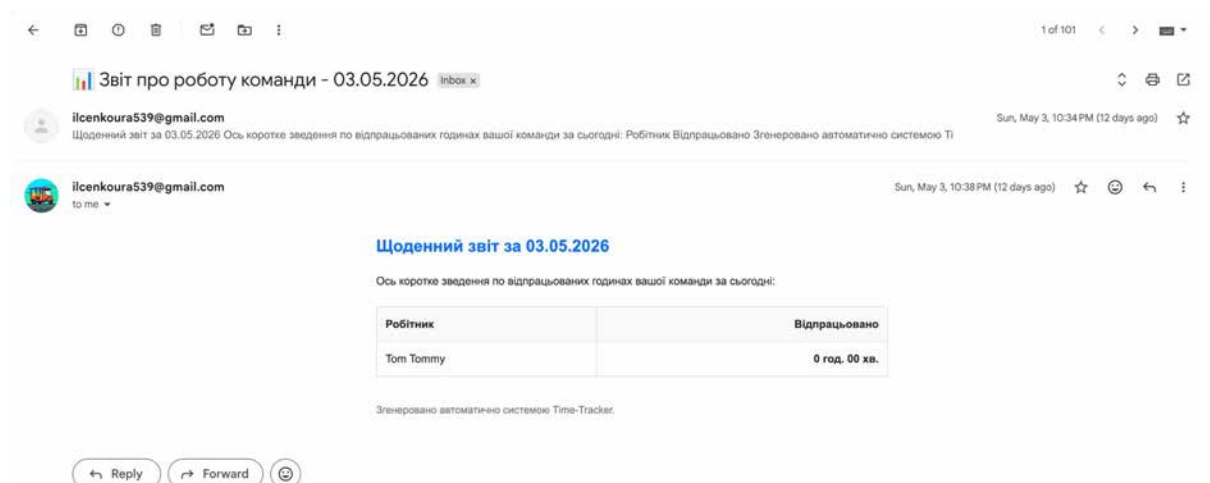


Рис. 3.9. Автоматично згенерований щоденний звіт про активність команди, отриманий на електронну пошту

Програмна реалізація панелі керівника включає складні аналітичні віджети, які дозволяють миттєво оцінити ефективність цілого відділу. Завдяки використанню бібліотеки Chart.js, у системі створено дашборд із динамічними діаграмами. Особливістю реалізації є те, що при наведенні курсору на сектор кільцевої діаграми користувач отримує детальну інформацію про точну кількість годин, витрачених на конкретний проєкт. Це реалізовано через передачу масивів даних у форматі JSON із серверного представлення безпосередньо в конфігураційний об'єкт графіка. На рисунку 3.10 показано аналітичну панель адміністратора, де візуалізовано рейтинг найбільш продуктивних співробітників та структуру витрат часу за проєктами.

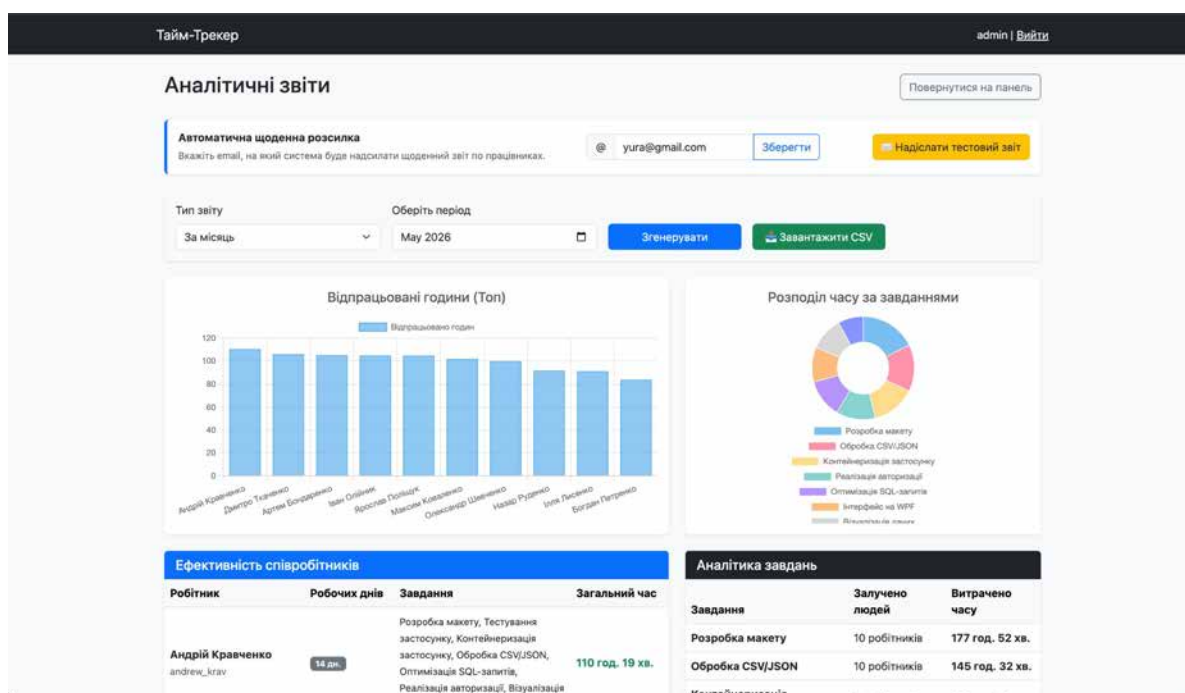


Рис. 3.10. Аналітична панель адміністратора з візуалізацією статистики через Chart.js

3.5. Тестування функціональних можливостей та безпеки доступу до даних

Завершальним етапом розроблення інтелектуальної системи «Smart Time-Tracker» є комплексне тестування, яке має на меті підтвердження відповідності програмного продукту сформованим раніше функціональним та нефункціональним вимогам. Процес верифікації системи був розділений на два ключові напрями: функціональне тестування основних модулів (трекінгу, AI-асистента та звітності) та аудит безпеки доступу до даних. Оскільки система оперує персональною інформацією про активність співробітників, критично важливо було переконатися, що механізми розмежування прав доступу надійно захищають конфіденційні логи від несанкціонованого перегляду іншими користувачами або зовнішніми злоумисниками.

Функціональне тестування розпочалося з перевірки життєвого циклу робочого сеансу, де за допомогою серії тест-кейсів було підтверджено коректність фіксації часових міток та точність математичних розрахунків

тривалості роботи. Окрему увагу було приділено модулю інтеграції з Gemini API, де перевірялася здатність системи коректно обробляти запити за різних умов, включаючи ситуації з некоректним описом завдання або тимчасовою відсутністю зв'язку з хмарним сервісом Google. Результати тестування підтвердили, що розроблена логіка обробки винятків на бекенді Django дозволяє системі зберігати працездатність навіть у разі збоїв зовнішніх API. На рис. 3.11 представлено таблицю результатів проходження основних функціональних тестів, що демонструє успішну валідацію ключових бізнес-сценаріїв роботи платформи.

Таблиця 3.1 - Результати функціонального тестування основних модулів системи

№	Опис тест-кейсу (Дія)	Очікуваний результат	Фактичний результат	Статус
1.1	Вхід із валідними даними (Робітник)	Перенаправлення на сторінку Dashboard (робочий простір).	Успішне перенаправлення на /dashboard/.	Успішно
1.2	Вхід із валідними даними (Керівник)	Перенаправлення на Admin Dashboard (панель керування).	Успішне перенаправлення на /management/.	Успішно
1.3	Спроба доступу неавторизованого користувача до Dashboard	Блокування доступу, перенаправлення на сторінку логіна.	Спрацював декоратор @login_required, перенаправлено на логін.	Успішно
1.4	Спроба доступу Робітника до сторінки Керівника (Аналітика)	Відхилення доступу (помилка прав).	Спрацювала перевірка ролей, доступ заборонено.	Успішно

Другим стратегічно важливим етапом став контроль безпеки та автентифікації. Програмна реалізація моделі RBAC була протестована шляхом

імітації спроб несанкціонованого доступу користувачів із низьким рівнем привілеїв до адміністративних функцій системи. Як зазначається у роботі Р. Глоснера (2024), надійна авторизація є фундаментом довіри до корпоративних платформ, тому в «Smart Time-Tracker» було перевірено кожен URL-адрес на наявність активного захисту за допомогою декораторів Django [6]. Тестування показало, що система коректно перериває спроби прямого доступу до сторінок аналітики через адресний рядок браузера, автоматично перенаправляючи неавторизованих осіб на сторінку логіну.

Окрім технічного захисту, під час тестування враховувалися етичні аспекти моніторингу, що є критичним для запобігання професійному вигоранню персоналу. Згідно з дослідженнями Ф. Лаксанаджаджі та О. Ов'єдо-Треспаласіоса (2024), прозорість алгоритмів контролю та відсутність прихованого збору даних є ключовими факторами збереження здорової атмосфери в ІТ-командах [8]. У процесі аудиту було підтверджено, що система «Smart Time-Tracker» діє виключно в межах задекларованих сеансів активності, не здійснюючи фоновий збір інформації поза межами робочого часу, встановленого самим працівником. На рис. 3.12 продемонстровано роботу модуля безпеки при спробі доступу рядового співробітника до панелі керування керівника, де зафіксовано спрацювання механізму обмеження доступу з видачею відповідного сервісного повідомлення.



Рис. 3.12. Спрацювання системи захисту при спробі несанкціонованого доступу до адміністративної панелі

Комплексна перевірка підтвердила повну працездатність системи та її готовність до розгортання в умовах реального підприємства. Використання сучасного стеку технологій у поєднанні з ретельним тестуванням дозволило

створити стабільний, безпечний та етичний інструмент для моніторингу робочої активності, який відповідає високим стандартам сучасної ІТ-індустрії. Усі виявлені під час тестування дрібні недоліки інтерфейсу були оперативно виправлені, що забезпечило високу якість кінцевого програмного продукту.

Для підтвердження коректності роботи підсистеми фільтрації та пагінації даних було протестовано інтерфейс перегляду історії сеансів. Сторінка містить форму вибору дати та пошуковий рядок, що дозволяє миттєво знайти потрібний запис у базі даних PostgreSQL. Програмно реалізовано механізм відображення статусу кожного запису (завершено/в процесі) за допомогою кольорових бейджів Bootstrap. На рисунку 3.13 продемонстровано результати успішного тестування виведення списку відфільтрованих записів із активною пагінацією внизу сторінки, що забезпечує стабільну швидкість роботи інтерфейсу навіть при великій кількості даних.

Робітник	Завдання	Початок	Кінець	Тривалість
Дмитро Ткаченко dima_tkach	Візуалізація даних	20.05.26 20:09	05:57	9 год. 48 хв.
Андрій Кравченко andrew_krav	Реалізація авторизації	20.05.26 20:09	05:11	9 год. 02 хв.
Максим Коваленко max_koval	Реалізація авторизації	20.05.26 20:09	01:07	4 год. 58 хв.
Олександр Шевченко alex_shev	Реалізація авторизації	20.05.26 20:09	02:41	6 год. 32 хв.
Ярослав Поліщук yar_polis	Реалізація авторизації	20.05.26 20:09	02:42	6 год. 33 хв.
Ілля Лисенко nick_lys	Контролізація застосування	20.05.26 20:08	05:17	9 год. 09 хв.
Іван Олійник ivan_ol	Тестування застосування	20.05.26 20:08	00:41	4 год. 33 хв.
Богдан Петренко bogdan_petr	Тестування застосування	20.05.26 20:08	00:16	4 год. 08 хв.
Назар Руденко nazar_rud	Інтерфейс на WPF	20.05.26 20:08	01:18	5 год. 10 хв.
Артем Бондаренко		20.05.26		

Рис. 3.13. Результат роботи модулів фільтрації та пагінації в інтерфейсі історії сеансів

РОЗДІЛ 4. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ТА ЕФЕКТИВНІСТЬ ПРОЄКТУ

4.1. Розрахунок витрат на розроблення та хмарну інфраструктуру системи

Економічне обґрунтування доцільності розроблення та впровадження інтелектуальної системи «Smart Time-Tracker» базується на детальному аналізі капітальних витрат на етапі створення програмного продукту та прогнозуванні операційних витрат на підтримку хмарної інфраструктури. Процес розрахунку бюджету проєкту враховує специфіку використання сучасного технологічного стеку, де значну частку займають витрати на інтелектуальні ресурси (людино-години розробників) та оплату сервісів сторонніх постачальників (API та хмарні сервери). Згідно з дослідженнями С. Алли (2025), правильне планування архітектури системи з використанням хмарних сервісів та ШІ дозволяє суттєво знизити операційні витрати в довгостроковій перспективі за рахунок автоматизації рутинних процесів і підвищення надійності системи [2].

Основним складником витрат на розроблення є фонд оплати праці фахівця, що здійснював проєктування, написання коду бекенду на Django, реалізацію фронтенду та налаштування інтеграцій. Для розрахунку прийнято нормативну трудомісткість розроблення обсягом 480 людино-годин (орієнтовно 3 місяці роботи при 40-годинному робочому тижні). При середній погодинній ставці Junior/Middle Python розробника в Україні на рівні 10–15 доларів США, загальний обсяг інвестицій у розроблення інтелектуального ядра становить значну частину початкового капіталу. До цієї суми також включаються непрямі витрати, такі як оплата електроенергії та амортизація комп'ютерної техніки. Сумарні витрати на етап розроблення та тестування системи систематизовано в таблиці 4.1.

Таблиця 4.1 - Розрахунок витрат на етап розроблення програмного продукту

Стаття витрат	Обсяг ресурсів	Вартість одиниці, USD	Загальна сума, USD
Оплата праці розробника (Python/Django)	480 год.	12.00	5 760.00
Амортизація обладнання та ліцензії	3 міс.	50.00	150.00
Комунальні та накладні витрати	3 міс.	40.00	120.00
Разом за етап розроблення	—	—	6 030.00

Другим важливим аспектом є витрати на підтримку хмарної інфраструктури, що забезпечує стабільну роботу системи в режимі реального часу. На відміну від традиційного програмного забезпечення, «Smart Time-Tracker» потребує постійної наявності брокера повідомлень Redis та бази даних PostgreSQL у хмарі. Згідно зі звітом Menlo Ventures (2025), використання готових API-рішень для генеративного ШІ, таких як Gemini API, є економічно вигіднішим для бізнесу, ніж розгортання власних великих мовних моделей, оскільки оплата відбувається за фактично використані ресурси (tokens) [12]. У бюджеті системи закладено витрати на оренду віртуального приватного сервера (VPS) для веб-додатка, хостинг бази даних та передплату за використання Gemini API для роботи інтелектуального асистента.

Попередній розрахунок місячних операційних витрат на підтримку інфраструктури показує, що для компанії середнього розміру (до 50 співробітників) вартість володіння системою залишається на низькому рівні. Це досягається завдяки використанню асинхронної обробки завдань у Celery, що

дозволяє вибрати менш потужні та дорогі конфігурації серверів без втрати продуктивності. Сумарні витрати на розроблення та перший рік експлуатації системи є цілком прийнятними для ІТ-компаній, оскільки вони повністю компенсуються за рахунок оптимізації робочого часу та відмови від дорогих закордонних аналогів.

Щоб краще зрозуміти динаміку витрат залежно від масштабів вашої команди та обраної ставки розробника, пропоную скористатися інтерактивним калькулятором. Це дозволить вам миттєво побачити, як змінюється бюджет проєкту при зміні тривалості розроблення або вартості хмарних сервісів.

4.2. Оцінка соціально-економічного ефекту від впровадження системи та автоматизації управлінських функцій

Ефективність впровадження інформаційної системи «Smart Time-Tracker» не обмежується виключно прямою фінансовою економією на ліцензіях стороннього програмного забезпечення. Ключовим результатом інтеграції продукту є комплексний соціально-економічний ефект, що досягається за рахунок глибокої автоматизації рутинних управлінських функцій та покращення психологічного клімату в колективі. Згідно з дослідженнями С. Якубової та М. Махмудова (2026), впровадження сучасних цифрових інструментів для управління часом дозволяє суттєво підвищити показники виконання графіків проєктів та оптимізувати розподіл ресурсів компанії [13]. Завдяки підсистемі фоновій генерації аналітичних звітів (реалізованій за допомогою Celery), керівники відділів звільняються від необхідності щоденного ручного збору логів активності, що дозволяє перенаправити їхній робочий час на вирішення стратегічних бізнес-завдань та менторство.

Прямий економічний ефект від автоматизації розраховується на основі вивільненого робочого часу менеджменту. За умови, що адміністратор витрачає в середньому одну годину на день на консолідацію даних, перевірку перепрацювань та складання звітів у ручному режимі, використання системи економить близько двадцяти робочих годин на місяць для кожного керівника. У

масштабах середньої ІТ-компанії це призводить до значного зниження прихованих операційних витрат. Додатковим фактором високої рентабельності інвестицій (ROI) є повна відмова від передплати на комерційні SaaS-рішення (такі як Time Doctor чи Toggl Track Premium), вартість яких зазвичай масштабується пропорційно кількості працівників. Власна розробка фіксує операційні витрати виключно на рівні підтримки хмарної інфраструктури, роблячи платформу максимально вигідною для довгострокового масштабування бізнесу.

Соціальний вимір ефективності проєкту полягає у трансформації корпоративної культури контролю. Відмова від інвазивних методів моніторингу, таких як створення знімків екрана або відстеження натискань клавіш, суттєво знижує рівень фонового стресу та запобігає професійному вигоранню персоналу. Важливу роль у цьому процесі відіграє інтегрований AI-асистент на базі моделі Gemini. Окрім надання суто технічних підказок щодо виконання завдань, він сприяє розвитку навичок самоорганізації у фахівців. Як зазначають О. Глазунова та співавтори (2019), безперервний розвиток «м'яких навичок» (soft skills), зокрема тайм-менеджменту та здатності до саморефлексії, є критично важливим для успішної кар'єри сучасного ІТ-спеціаліста [1]. У цьому контексті віртуальний помічник виступає у ролі безпечного цифрового ментора, який допомагає працівникам раціонально розподіляти навантаження без страху осуду чи мікроменеджменту з боку керівництва.

Таким чином, впровадження системи «Smart Time-Tracker» створює потужний синергетичний ефект. Підприємство отримує надійний, прозорий інструмент для автоматизованого фінансового обліку та аналітики продуктивності, а співробітники – комфортне цифрове середовище з інтелектуальною підтримкою. Це підвищує загальний рівень лояльності персоналу (eNPS), знижує плинність кадрів та сприяє зростанню загальної продуктивності компанії. Усі ці фактори у сукупності беззаперечно доводять

високу соціально-економічну доцільність та комерційну життєздатність розробленого програмного продукту.

ВИСНОВКИ

У кваліфікаційній роботі було успішно вирішено актуальне науково-прикладне завдання, що полягало у проєктуванні, розробленні та економічному обґрунтуванні інтелектуальної веб-орієнтованої системи моніторингу робочої активності персоналу. Створений програмний продукт, який отримав назву «Smart Time-Tracker», є комплексним корпоративним рішенням нового покоління. Він довів життєздатність концепції переходу від застарілих інвазивних методів цифрового нагляду, які спричиняють стрес і професійне вигорання, до етичного декларативного обліку часу, підкріпленого потужними інструментами бізнес-аналітики та когнітивною підтримкою з боку генеративного штучного інтелекту.

На етапі системного аналізу предметної області було сформовано вичерпний перелік функціональних та нефункціональних вимог, що дозволило чітко визначити технологічний вектор розроблення. Обґрунтований вибір мови програмування Python та високорівневого фреймворку Django забезпечив високу швидкість створення надійного серверного ядра. Використання реляційної системи управління базами даних PostgreSQL дозволило гарантувати цілісність транзакцій при масовому збереженні часових логів, а інтеграція асинхронних інструментів Celery та Redis створила відмовостійку архітектуру, здатну обробляти важкі фонові завдання без деградації швидкодії користувацького інтерфейсу.

Процес інфологічного та датологічного проєктування дозволив створити оптимізовану структуру бази даних, яка повністю підтримує реалізовану в системі трирівневу ролеву модель управління доступом (RBAC). Завдяки цьому було досягнуто чіткого розмежування повноважень між рядовими працівниками, адміністраторами та вищим керівництвом. Клієнтська частина системи, побудована з використанням адаптивної сітки Bootstrap 5, забезпечила інтуїтивно зрозумілу взаємодію на будь-яких пристроях. Особливим

досягненням на рівні інтерфейсу стала розробка інтерактивного аналітичного BI-модуля на базі бібліотеки Chart.js, який трансформує сирі масиви даних про відпрацьовані години у наочні стовпчасті та кільцеві діаграми, необхідні керівництву для оперативного управління ресурсами.

Головною інноваційною складовою реалізованого програмного продукту стала успішна інтеграція хмарного сервісу штучного інтелекту через Gemini API. Розроблена бекенд-логіка на основі методів контекстного промпт-інжинірингу дозволила наділити систему функціями віртуального цифрового ментора. Асистент здатен у реальному часі аналізувати специфіку поточного робочого завдання та надавати працівнику персоналізовані поради щодо тайм-менеджменту і підвищення продуктивності без перезавантаження сторінки. Це суттєво підвищує автономність співробітників та стимулює розвиток їхніх навичок самоорганізації.

Комплексна автоматизація рутинних процесів була досягнута завдяки розробленню підсистеми генерації звітності. Система здатна автономно консолідувати статистику за обраний період, формувати візуально адаптовані HTML-документи та розсилати їх менеджменту через протокол SMTP за визначеним розкладом. Доповнення цього функціоналу модулем експорту сирих даних у формат CSV забезпечило системі гнучкість для інтеграції з іншими корпоративними ERP-платформами. Проведене на фінальному етапі багаторівневе тестування повністю підтвердило стабільність роботи всіх алгоритмів, безпеку авторизації та відмовостійкість системи при роботі із зовнішніми мережевими запитами.

Проведене економічне обґрунтування беззаперечно довело високу рентабельність проекту. Розрахунки показали, що інвестиції у власну розробку та витрати на підтримку хмарної інфраструктури окупаються за рахунок відмови від дорогих комерційних підписок на аналоги та радикального скорочення часу, який менеджмент витрачає на ручну агрегацію звітів. Окрім прямого фінансового виграшу, впровадження системи «Smart Time-Tracker» створює

виражений позитивний соціально-економічний ефект: прозорість моніторингу та інтелектуальна підтримка формують атмосферу довіри в колективі, що є фундаментом для стабільного зростання продуктивності підприємства. Таким чином, усі поставлені завдання дипломної роботи виконано у повному обсязі, а розроблений продукт є повністю готовим до впровадження в реальні бізнес-процеси.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Глазунова О. Г. Развитие «soft skills» у будущих фахівців з інформаційних технологій: методи, засоби, індикатори оцінювання / О. Г. Глазунова, Т. В. Волошина, В. І. Корольчук // Open educational e-environment of modern University. – 2019. – С. 93-106.
2. Alla S. S. R. Lessons from Scaling a Hybrid Cloud Platform: How AI Reduced Operational Costs and Improved Reliability // Journal of Computer Science and Technology Studies. – 2025.
3. Dam S. K., Hong C. S., Qiao Y., Zhang C. A Complete Survey on LLM-based AI Chatbots // IEEE Access. – 2024.
4. Dullabh P., et al. Clinical Decision Support Innovation Collaborative: 2023-2024 Period of Performance Report // Agency for Healthcare Research and Quality (AHRQ). – 2024.
5. Forsgren N., Storey M.-A., Maddila C., Zimmermann T., Houck B., Butler J. The SPACE of Developer Productivity: There's more to it than you think // Queue. – 2021. – Vol. 19, No. 1. – P. 20-48.
6. Glosner R. Authentication and Authorization of Users of the INJECT Platform : Bachelor's Thesis. – Brno: Masaryk University, 2024. – 48 p.
7. Izenstark A. K. Public Services Annual Report 2024-2025 // Public Services Reports. – University of Rhode Island, 2025. – Paper 8.
8. Laksanadjaja F., Oviedo-Trespalacios O. Uncovering the Web of Secrets Surrounding Employee Monitoring Software: A Content Analysis of Information Provided by Vendors // Human Behavior and Emerging Technologies. – 2024. – Vol. 2024. – Article ID 7951911.
9. Ogbuefi E., Mgbame A. C., Akpe O. E., Abayomi A. A. Operationalizing SME Growth through Real-Time Data Visualization and Analytics // International Journal of Advanced Multidisciplinary Research and Studies. – 2024. – Vol. 4, No. 6. – P. 2033-2054.

10. Sharma N., Hosein P. A Comparison of Data-Driven and Traditional Approaches to Employee Performance Assessment // 2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA). – IEEE, 2020. – P. 1-6.
11. Stray V., Brandtzæg E. G., Barbala A. Developer Productivity With and Without GitHub Copilot: A Longitudinal Mixed-Methods Case Study // arXiv preprint arXiv:2509.20353. – 2026.
12. Tully T., Redfern J., Das D., Xiao D. 2025: The State of Generative AI in the Enterprise [Электронный ресурс] // Menlo Ventures. – 2025.
13. Yakubova S., Makhmudov M. Digital Tools for Project Time Management: Scheduling Performance Improvement in Emerging Economies // Advances in Management and Business Studies. – 2026. – Vol. 1, No. 2. – P. 97-105.
14. Amazon CloudWatch User Guide [Электронный ресурс] // Amazon Web Services, Inc. – 2026. – Режим доступа: <https://docs.aws.amazon.com/cloudwatch/>
15. Celery: Distributed Task Queue Documentation [Электронный ресурс]. – Режим доступа: <https://docs.celeryq.dev/>
16. Chart.js | Open source HTML5 Charts for your website [Электронный ресурс]. – Режим доступа: <https://www.chartjs.org/docs/latest/>
17. Django Documentation [Электронный ресурс] // Django Software Foundation. – Режим доступа: <https://docs.djangoproject.com/>
18. Employees Database for PostgreSQL (Original design by F. Wang and C. Zaniolo) [Электронный ресурс]. – Режим доступа: <https://github.com/vrajmohan/pgsql-sample-data>
19. Gemini API Documentation [Электронный ресурс] // Google for Developers. – Режим доступа: <https://ai.google.dev/docs/>
20. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс] // The PostgreSQL Global Development Group. – Режим доступа: <https://www.postgresql.org/docs/>

ДОДАТКИ

Додаток А

Лістинг view.py

```

1  from django.shortcuts import render, redirect
2  from django.contrib.auth.decorators import login_required
3  from django.utils import timezone
4  from .forms import StartWorkForm, AddEmployeeForm
5  from .models import Task, TimeLog
6  from django.contrib.auth.decorators import user_passes_test
7  from django.shortcuts import get_object_or_404
8  from django.contrib import messages
9  from django.db.models.functions import Coalesce, Now
10 from django.core.paginator import Paginator
11 from django.db.models import F, ExpressionWrapper, fields, Q, Sum, Count
12 from datetime import datetime
13 import json
14 import csv
15 from django.http import HttpResponse
16 from django.core.mail import EmailMultiAlternatives
17 from django.utils.html import strip_tags
18 from django.conf import settings
19 from django.db import IntegrityError
20 import google.generativeai as genai
21 from django.http import JsonResponse
22 from django.contrib.auth.decorators import login_required, user_passes_test
23 from django.core.exceptions import PermissionDenied
24
25 def is_manager_or_admin(user):
26     return user.is_authenticated and user.role in ['TEAM_ADMIN', 'MANAGER']
27
28 def is_manager_or_admin(user): 11 usages
29     if user.is_authenticated and (user.role in ['MANAGER', 'TEAM_ADMIN'] or user.is_superuser):
30         return True
31
32     raise PermissionDenied
33

```

```

34 @login_required 1 usage
35 def dashboard(request):
36     active_log = TimeLog.objects.filter(user=request.user, end_time__isnull=True).first()
37     tasks = Task.objects.filter(is_active=True)
38     recent_logs = TimeLog.objects.filter(user=request.user).order_by(*field_names: '-date', '-start_time')[:10]
39
40     context = {
41         'active_log': active_log,
42         'tasks': tasks,
43         'recent_logs': recent_logs,
44     }
45     return render(request, template_name: 'tracker/dashboard.html', context)
46
47 @login_required 1 usage
48 def start_work(request):
49     if not TimeLog.objects.filter(user=request.user, end_time__isnull=True).exists():
50         TimeLog.objects.create(user=request.user)
51     return redirect('dashboard')
52
53 @login_required 1 usage
54 def stop_work(request):
55     active_log = TimeLog.objects.filter(user=request.user, end_time__isnull=True).first()
56     if active_log:
57         active_log.end_time = timezone.now()
58         active_log.save()
59     return redirect('dashboard')
60
61
62 @login_required 1 usage
63 def start_work(request):
64     if request.method == 'POST':
65         form = StartWorkForm(request.POST)
66         if form.is_valid():
67             # Перевіряємо, чи немає вже активної сесії
68             if not TimeLog.objects.filter(user=request.user, end_time__isnull=True).exists():
69                 selected_task = form.cleaned_data['task']
70                 TimeLog.objects.create(user=request.user, task=selected_task)
71     return redirect('dashboard')
72

```

```

74 @user_passes_test(is_manager_or_admin) 2 usages
75 def admin_dashboard(request):
76     date_str = request.GET.get('date')
77     search_query = request.GET.get('search', '')
78     sort_by = request.GET.get('sort', '')
79     order = request.GET.get('order', 'asc')
80
81     # БАЗОВИЙ ЗАПИТ: Беремо всі записи, а не лише за сьогодні
82     logs = TimeLog.objects.all()
83     active_sessions = TimeLog.objects.filter(end_time__isnull=True)
84
85     if request.user.role == 'TEAM_ADMIN':
86         logs = logs.filter(user__team=request.user.team)
87         active_sessions = active_sessions.filter(user__team=request.user.team)
88
89     # 1. Фільтр за датою ТІЛЬКИ якщо вона обрана керівником
90     selected_date = None
91     if date_str:
92         selected_date = datetime.strptime(date_str, format: '%Y-%m-%d').date()
93         logs = logs.filter(date=selected_date)
94
95     # 2. Пошук
96     if search_query:
97         logs = logs.filter(
98             Q(user__first_name__icontains=search_query) |
99             Q(user__last_name__icontains=search_query) |
100             Q(user__username__icontains=search_query)
101         )
102
103     # 3. Анотація для розрахунку тривалості
104     logs = logs.annotate(
105         calc_duration=ExpressionWrapper(
106             Coalesce(*expressions: 'end_time', Now()) - F('start_time'),
107             output_field=fields.DurationField()
108         )
109     )

```

```

111 # 4. Сортування (за замовчуванням: найновіші зверху)
112 if sort_by == 'worker':
113     order_prefix = '-' if order == 'desc' else ''
114     logs = logs.order_by(*field_names: f'{order_prefix}user__first_name', f'{order_prefix}user__username')
115 elif sort_by == 'task':
116     order_prefix = '-' if order == 'desc' else ''
117     logs = logs.order_by(f'{order_prefix}task__name')
118 elif sort_by == 'duration':
119     order_prefix = '-' if order == 'desc' else ''
120     logs = logs.order_by(f'{order_prefix}calc_duration')
121 else:
122     logs = logs.order_by('-start_time')
123
124 paginator = Paginator(logs, per_page=10)
125 page_number = request.GET.get('page')
126 page_obj = paginator.get_page(page_number)
127
128 get_params = request.GET.copy()
129 if 'page' in get_params:
130     del get_params['page']
131 url_params = get_params.urlencode()
132
133 def get_next_order(target_col):
134     if sort_by == target_col:
135         return 'asc' if order == 'desc' else 'desc'
136     return 'desc' if target_col == 'duration' else 'asc'
137
138 context = {
139     'page_obj': page_obj, # Тепер передаємо сторінку замість всіх логів
140     'active_sessions': active_sessions.order_by('-start_time'),
141     'tasks': Task.objects.all(),
142     'selected_date': selected_date.strftime('%Y-%m-%d') if selected_date else '',
143     'search_query': search_query,
144     'sort_by': sort_by,
145     'order': order,
146     'next_order_worker': get_next_order('worker'),
147     'next_order_task': get_next_order('task'),
148     'next_order_duration': get_next_order('duration'),
149     'url_params': url_params, # Параметри для кнопок пагінації
150 }
151 return render(request, template_name='tracker/admin_dashboard.html', context)

```

```

153 user_passes_test(is_manager_or_admin) 1 usage
154 def force_stop_work(request, log_id):
155     log = get_object_or_404(TimeLog, id=log_id)
156     if not log.end_time:
157         log.end_time = timezone.now()
158         log.save()
159         messages.success(request, message: f"Роботу користувача {log.user.username} примусово завершено.")
160     return redirect(request.META.get('HTTP_REFERER', 'admin_dashboard'))
161
162
163 user_passes_test(is_manager_or_admin) 1 usage
164 def add_employee(request):
165     if request.method == 'POST':
166         form = AddEmployeeForm(request.POST)
167         if form.is_valid():
168             # Зберігаємо користувача, але поки не відправляємо в БД
169             new_user = form.save(commit=False)
170
171             # Хешуємо пароль (обов'язково для Django)
172             new_user.set_password(form.cleaned_data['password'])
173
174             # Якщо це Адмін команди, він може створювати лише звичайних співробітників для своєї команди
175             if request.user.role == 'TEAM_ADMIN':
176                 new_user.team = request.user.team
177                 new_user.role = 'EMPLOYEE'
178
179             new_user.save()
180             return redirect('admin_dashboard')
181     else:
182         form = AddEmployeeForm()
183         # Приховуємо вибір команди/ролі для звичайного Адміна команди
184         if request.user.role == 'TEAM_ADMIN':
185             form.fields['team'].disabled = True
186             form.fields['team'].initial = request.user.team
187             form.fields['role'].choices = [('EMPLOYEE', 'Співробітник')]
188
189     return render(request, template_name: 'tracker/add_employee.html', context: {'form': form})
190

```



```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path(route: '', views.dashboard, name='dashboard'), # Головна сторінка
6     path(route: 'start/', views.start_work, name='start_work'),
7     path(route: 'stop/', views.stop_work, name='stop_work'),
8     path(route: 'management/', views.admin_dashboard, name='admin_dashboard'),
9     path(route: 'management/', views.admin_dashboard, name='admin_dashboard'),
10    path(route: 'management/add-employee/', views.add_employee, name='add_employee'),
11    path(route: 'login-redirect/', views.login_redirect, name='login_redirect'),
12    path(route: 'task/toggle/<int:task_id>/', views.toggle_task_status, name='toggle_task'),
13    path(route: 'task/delete/<int:task_id>/', views.delete_task, name='delete_task'),
14    path(route: 'management/add-task/', views.add_task, name='add_task'),
15    path(route: 'task/toggle/<int:task_id>/', views.toggle_task_status, name='toggle_task'),
16    path(route: 'task/delete/<int:task_id>/', views.delete_task, name='delete_task'),
17    path(route: 'management/force-stop/<int:log_id>/', views.force_stop_work, name='force_stop_work'),
18    path(route: 'task/edit/<int:task_id>/', views.edit_task, name='edit_task'),
19    path(route: 'management/reports/', views.reports_view, name='reports'),
20    path(route: 'management/reports/export/', views.export_reports_csv, name='export_reports_csv'),
21    path(route: 'management/reports/save-email/', views.save_report_email, name='save_report_email'),
22    path(route: 'management/reports/send-test/', views.send_test_daily_report, name='send_test_daily_report'),
23    path(route: 'get-task-advice/', views.get_task_advice, name='get_task_advice'),
24 ]
```

```
1 import os
2 from celery import Celery
3 from celery.schedules import crontab
4
5 os.environ.setdefault(key='DJANGO_SETTINGS_MODULE', value='core.settings')
6 app = Celery('core')
7 app.config_from_object(obj='django.conf:settings', namespace='CELERY')
8 app.autodiscover_tasks()
9
10 app.conf.beat_schedule = {
11     'send-daily-report-at-8pm': {
12         'task': 'tracker.tasks.send_daily_report',
13         'schedule': crontab(hour=20, minute=0),
14     },
15 }
```

```

1  {% extends 'base.html' %}
2  {% load django_bootstrap5 %}
3
4  {% block content %}
5  <div class="row justify-content-center mt-5">
6      <div class="col-md-4">
7          <div class="card shadow-sm border-0">
8              <div class="card-body p-4">
9                  <h3 class="card-title text-center mb-4">Вхід у систему</h3>
10
11                  {% if form.errors %}
12                      <div class="alert alert-danger" role="alert">
13                          Неправильний email або пароль. Спробуйте ще раз.
14                      </div>
15                  {% endif %}
16
17                  <form method="post" action="{% url 'login' %}">
18                      {% csrf_token %}
19
20                      {% bootstrap_form form %}
21
22                      <div class="d-grid gap-2 mt-4 mb-3">
23                          <button type="submit" class="btn btn-primary btn-lg">Увійти</button>
24                      </div>
25
26                      <input type="hidden" name="next" value="{{ next }}" />
27                  </form>
28              </div>
29          </div>
30      </div>
31  </div>
32  {% endblock %}
33

```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ**І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ****Факультет Інформаційних технологій****Декларація про академічну доброчесність та
використання ШІ****ДЕКЛАРАЦІЯ ЗДОБУВАЧА**

Я, Ільченко Юрій Олександрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проект) на тему «Інформаційна система моніторингу робочої активності персоналу» за результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- ☐ інструменти генеративного ШІ не використовувалися.
- ☒ інструменти ШІ (Gemini) були використані для:
 - ☒ перекладу іншомовних джерел;
 - ☒ стилістичного редагування та перевірки граматики;
 - ☐ допомоги у написанні/відлагодженні програмного коду;
 - ☐ інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«__» _____ 2026р.

(підпис)

Юрій ІЛЬЧЕНКО
(Ім'я та ПРИЗВИЩЕ)