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 оптимізації розподілу робочого часу з використанням машинного навчання**»**

Спеціальність «126 Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні системи та технології»

Гарант освітньої програми
к.е.н., доцент _____ (підпис) **Максим МОКРІЄВ**

Керівник бакалаврської кваліфікаційної роботи
к.е.н., доцент _____ (підпис) **Євгеній СТАРИЧЕНКО**

Виконав _____ (підпис) **Максим ГУПАЛИК**

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

Гупалику Максиму Миколайовичу

Спеціальність 126 – Інформаційні системи та технології

ОП «Інформаційні системи та технології»

Тема бакалаврської кваліфікаційної роботи «Інформаційна система оптимізації розподілу робочого часу з використанням машинного навчання» затверджена наказом від «11» лютого 2026 р. № 453 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Дані про працівників, структурні підрозділи та посадові ролі підприємства; табелі обліку робочого часу; заявки користувачів; журнали активності та аудиту дій користувачів; дані зовнішніх сервісів автентифікації та календарів; REST API для інтеграції з ERP-системами та сервісами сповіщень; засоби збереження, оброблення, аналізу та візуалізації інформації про робочий час.

Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області моніторингу технічного стану промислового обладнання та технологій predictive maintenance.
2. Дослідження сучасних IoT-платформ і інформаційних систем моніторингу обладнання.
3. Проєктування інформаційного та програмного забезпечення інтелектуальної системи моніторингу технічного стану обладнання.
4. Розробка механізмів збору, збереження та оброблення телеметричних

даних від IoT-пристроїв.

5. Реалізація алгоритмів машинного навчання для виявлення аномалій і прогнозування можливих відмов обладнання.
6. Розробка засобів візуалізації, аналітики та формування сповіщень про критичні стани обладнання.
7. Тестування функціональних можливостей системи та оцінювання ефективності прогнозування технічного стану обладнання.

Дата видачі завдання «11» лютого 2026 р.

Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Євгеній СТАРИЧЕНКО**
Завдання взяв до виконання	_____ (підпис)	**Максим ГУПАЛИК**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – прикладний програмний інтерфейс для обміну даними між компонентами системи.

БД – база даних.

GUI – графічний інтерфейс користувача.

PyQt6 – бібліотека Python для створення кросплатформених настільних застосунків із графічним інтерфейсом.

Python – мова програмування, використана для реалізації програмної логіки системи.

FastAPI – Python-фреймворк для створення серверної частини та REST API.

REST – архітектурний підхід до побудови вебсервісів на основі HTTP-запитів.

JSON – текстовий формат обміну структурованими даними між клієнтом і сервером.

PostgreSQL – реляційна система керування базами даних, використана як основне сховище даних.

SQLite – вбудована реляційна база даних, що може використовуватися для локального або офлайн-режиму.

Redis – система кешування та брокер повідомлень для швидкої обробки тимчасових даних і черг задач.

Celery – інструмент для асинхронної обробки фонових задач.

JWT – JSON Web Token, формат токена для автентифікації користувачів.

OIDC – OpenID Connect, протокол автентифікації користувачів на основі OAuth 2.0.

SSO – єдиний вхід користувача до системи через корпоративний обліковий запис.

RBAC – модель розмежування прав доступу на основі ролей користувачів.

OAuth 2.0 – протокол авторизації для безпечного доступу до ресурсів.

HTTPS – захищений протокол передавання даних між клієнтом і сервером.

TLS – криптографічний протокол захисту мережевого з’єднання.

SQL – мова структурованих запитів для роботи з реляційними базами даних.

UUID – унікальний ідентифікатор, що використовується як первинний ключ у таблицях бази даних.

CRUD – набір базових операцій із даними: створення, читання, оновлення та видалення.

ER-діаграма – діаграма «сутність–зв’язок», що використовується для моделювання структури бази даних.

UML – уніфікована мова моделювання програмних систем.

DFD – діаграма потоків даних, що відображає рух інформації між процесами, сховищами та зовнішніми сутностями.

CI/CD – безперервна інтеграція та безперервне розгортання програмного забезпечення.

Docker – платформа контейнеризації програмних компонентів.

Docker Compose – інструмент для запуску кількох контейнерів за єдиним конфігураційним файлом.

OpenAPI – стандарт опису REST API.

Swagger – інструмент для перегляду та тестування OpenAPI-документації.

SMTP – протокол надсилання електронної пошти.

ERP – корпоративна інформаційна система для планування та управління ресурсами підприємства.

HR – підрозділ або спеціаліст із управління персоналом.

ВСТУП

Зміна парадигм організації праці, поява гібридних і віддалених форматів зайнятості, а також зростання вимог до прозорості управлінських процесів зумовили необхідність створення автоматизованих систем для точного, надійного й масштабованого обліку робочого часу. У традиційних моделях відстеження присутності персоналу значну частину операцій виконується вручну або напівавтоматично, що призводить до помилок, затримок у звітності, обмеженої спостережуваності й зростання навантаження на адміністративний персонал. Крім того, відсутність адаптивності до реальних сценаріїв (зміни графіків, надурочні, порушення політик) ускладнює інтеграцію таких систем у сучасні IT-ландшафти підприємств.

Актуальність теми зумовлена необхідністю впровадження автоматизованих засобів обліку робочого часу в умовах гібридної, дистанційної та змінної зайнятості, що дедалі частіше використовується у сучасних підприємствах. Традиційні системи табелювання часто базуються на ручному вводі або не мають засобів контролю достовірності, що призводить до неточностей, зловживань і неузгодженості даних. В умовах зростання вимог до продуктивності, прозорості внутрішніх процесів та інтегрованості з ERP-системами постає потреба в інтелектуальному програмному рішенні, яке здатне автоматично фіксувати активність користувача, динамічно формувати табелі, реагувати на відхилення та забезпечувати відповідність трудовим політикам підприємства.

Вихідні положення дослідження базуються на застосуванні об’єктно-орієнтованого підходу до проєктування та впровадження багатокомпонентних програмних систем. Система, що розробляється, реалізується мовою Python із використанням PyQt6 для інтерфейсу, FastAPI для побудови API-шару, PostgreSQL як основного сховища даних, Redis для кешування та Celery для асинхронної обробки подій. Модель побудови системи відповідає шаблону поділу відповідальностей на

рівні компонентів (GUI, API, служби бізнес-логіки, сховища, аудит, політики), що забезпечує масштабованість і контроль якості.

Для розгляду повноцінного системи і його дослідження потрібно вирішити наступні **завдання:**

– виконати системний аналіз існуючих рішень у сфері обліку робочого часу;

– сформувати функціональні, нефункціональні та технічні вимоги до системи;

– побудувати UML-моделі: діаграму варіантів використання, активності, класів, компонентів і розгортання;

– розробити логічну та фізичну модель бази даних;

– реалізувати підсистеми реєстрації, авторизації, фіксації check-in/check-out, генерації табелів, управління політиками, відпустками та надурочними;

– забезпечити механізми безпеки (JWT, RBAC), логування подій і нотифікацій;

– налаштувати процеси тестування, журналювання, спостережуваності та CI/CD-конвеєр розгортання.

Мета дослідження - розробити інтелектуальну систему обліку робочих годин, здатну до автоматизованої фіксації присутності, виявлення аномалій, підтримки трудових політик і інтеграції з внутрішніми сервісами підприємства з урахуванням вимог до безпеки, масштабованості та прозорості.

Об’єкт дослідження - процеси організації обліку та аналізу робочого часу в умовах сучасного підприємства.

Предмет дослідження - інформаційні моделі, алгоритми й архітектурні рішення, що реалізують автоматизований контроль робочої активності, управління запитами та генерацію табелів.

Методи дослідження включають:

– системний аналіз вимог;

– об’єктно-орієнтоване моделювання (UML);

– логіко-семантичне проєктування бази даних;

– реалізацію прототипу з використанням Python-фреймворків (FastAPI, PyQt6, Celery);

– тестування продуктивності, fault-injection, інтеграційне й функціональне випробування.

Наукова новизна роботи полягає в поєднанні автоматизованого обліку робочого часу, політик доступу, аналізу аномалій і гнучких засобів візуалізації в єдиній кросплатформенній системі з відкритою архітектурою. Розроблена система підтримує масштабування, кастомізацію політик, інтеграцію з корпоративними сервісами та забезпечує дотримання принципів інтероперабельності, спостережуваності й відмовостійкості.

Структура роботи відповідає етапам життєвого циклу ПЗ:

Розділ 1 містить теоретичні основи, огляд рішень, постановку задачі, аналіз вимог і побудову UML-діаграм;

Розділ 2 присвячено моделюванню даних, архітектурі компонентів і логіці взаємодій;

Розділ 3 описує реалізацію, особливості коду, сценарії обробки подій і безпекові механізми;

Розділ 4 містить методики тестування, результати випробувань і аналіз досягнення технічних критеріїв якості.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області оптимізації робочого часу

Управління робочим часом є однією з ключових функцій у структурі управління підприємством. Класичні системи табелювання, що ґрунтуються на ручному вводі даних або простих RFID-рішеннях, не забезпечують необхідного рівня адаптивності, масштабованості та інтегрованості з внутрішніми системами підприємства.

Предметна область охоплює широкий спектр інформаційних та управлінських завдань: від реєстрації початку й завершення робочого дня до обробки запитів на відпустку, обрахунку надурочних, виявлення порушень трудової дисципліни та формування аналітичної звітності. Особливе значення має формалізація трудових політик підприємства (тривалість зміни, обідні перерви, обмеження на понаднормову роботу, допустимі відхилення), що мають бути відображені в правилах валідації табелів.

Рисунок 1.1 демонструє узагальнену модель предметної області у вигляді DFD-діаграми першого рівня, яка відображає логіку взаємодії основних підсистем та зовнішніх учасників.

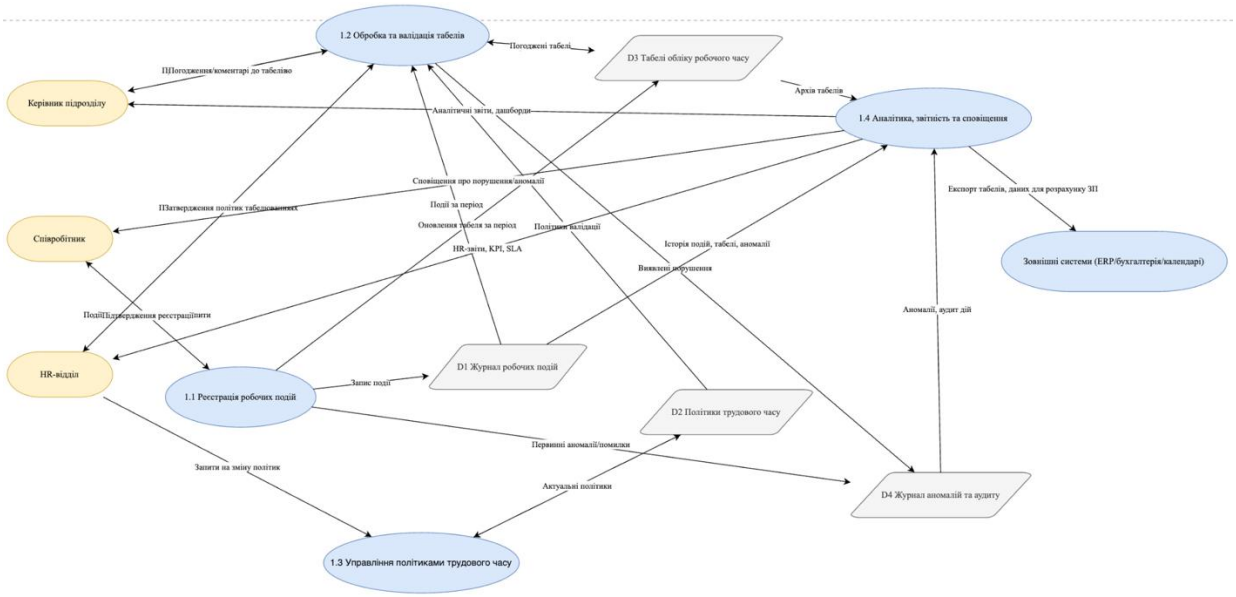

Рис.1.1 – DFD діаграма 1-го рівня оптимізації робочого часу

На схемі показано чотири ключові функціональні блоки: реєстрацію робочих подій, обробку та валідацію табелів, управління політиками трудового часу та модуль аналітики і сповіщень. Співробітники та HR-відділ генерують первинні події про початок, завершення роботи чи перерви, які після збереження у журналі подій формують основу для побудови табелів. Підрозділ керівників виконує погодження табелів і залишає коментарі, а модуль управління політиками надає актуальні правила щодо тривалості зміни, допустимих відхилень та обмежень, що використовуються під час автоматичної валідації. У випадку виявлення порушень система передає аномалії до відповідного журналу та ініціює сповіщення. Аналітичний модуль формує HR-звіти, KPI, SLA, журнали аномалій та агреговані дані для зовнішніх систем, включно з бухгалтерськими модулями. Таким чином, предметна область описує повний цикл обліку робочих подій - від фіксації факту присутності до завершальної аналітики та інтеграційних процедур.

Загальна структура предметної області представлена в таблиці нижче.

Таблиця 1.1 - Ключові компоненти предметної області системи обліку робочих годин

Категорія	Опис
Користувачі системи	Співробітники, керівники відділів, адміністратори, HR-персонал
Робочі події	Початок і кінець робочого дня, обідні перерви, запізнення, відсутність
Табель обліку	Автоматично або вручну згенерована сукупність подій за певний період
Політики трудового часу	Правила щодо тривалості зміни, обідньої перерви, відпусток, овертайму
Аномалії	Відхилення від норм: пропуски, перевищення часу, дублювання записів
Звіти	Формалізовані документи з аналітикою часу, порушень, запитів
Безпека	Автентифікація, ролі доступу (RBAC), аудит дій користувача
Інтеграції	ERP-системи, бухгалтерські модулі, поштові сервіси, календарі

У контексті автоматизації ця предметна область вимагає побудови системи, яка не тільки зберігає історію дій співробітників, а й дозволяє проводити валідацію згідно з політиками, виявляти аномальні шаблони поведінки, формувати зручні візуалізації для менеджменту й оперативно реагувати на відхилення.

Система має підтримувати централізовану модель керування з можливістю делегування повноважень на рівні підрозділів, а також забезпечувати масштабовану архітектуру, придатну до розгортання як у локальному середовищі, так і в хмарній інфраструктурі. Наявність API-шару, механізмів сповіщення (email, push), журналювання дій користувача та засобів аналітики є необхідною умовою для адаптації системи до потреб конкретного підприємства.

1.2 Огляд інформаціних джерел та існуючих рішень

У процесі розробки критично важливим є вивчення сучасного стану галузі, принципів побудови тайм-трекінгових платформ, архітектурних рішень та функціональних можливостей провідних комерційних і відкритих систем. Це дозволяє визначити найкращі практики, виявити архітектурні обмеження та врахувати специфіку реального використання в організаціях різного масштабу.

Одним із таких рішень є Replicon - потужна SaaS-платформа корпоративного рівня для ведення табелів, контролю проєктного часу й інтеграції з фінансовими модулями. Особливістю системи є можливість застосування AI/ML для генерації точних записів часу за активністю користувачів. Основна частина функцій реалізується через вебінтерфейс із багаторівневим доступом та централізованим адмініструванням інтерфейс якої представлений на рис.1.2.

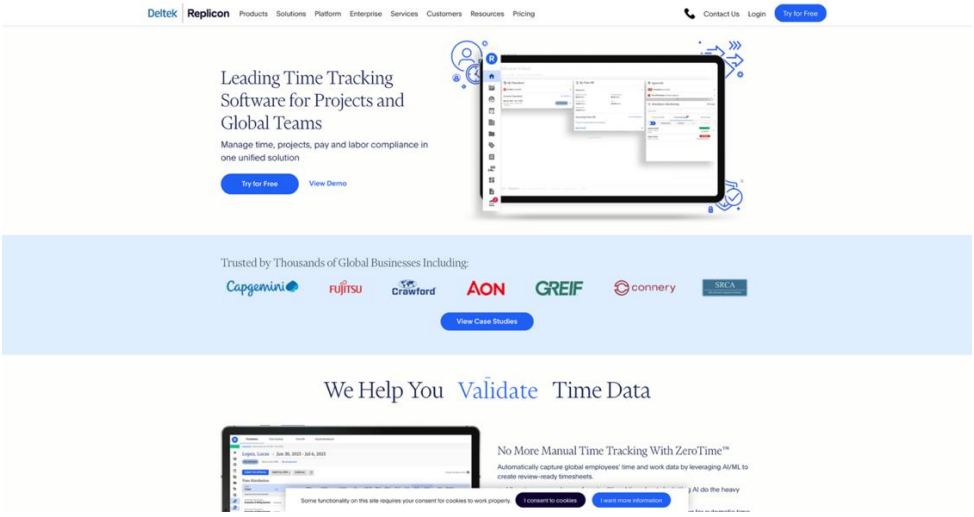

Рис. 1.2 – Вебінтерфейс платформи Replicon з елементами автоматичного трекінгу часу.

Інше популярне рішення - Clockify, яке позиціонується як безкоштовний інструмент для команд із можливістю необмеженої кількості користувачів. Його основна перевага - простота використання, багатомовна підтримка та розгалужені звіти з таймерами, календарями та графіками активності (рис. 1.3). Clockify має плагіни для браузера, інтеграції з Trello, GitHub, Google Calendar та API-інтерфейс.

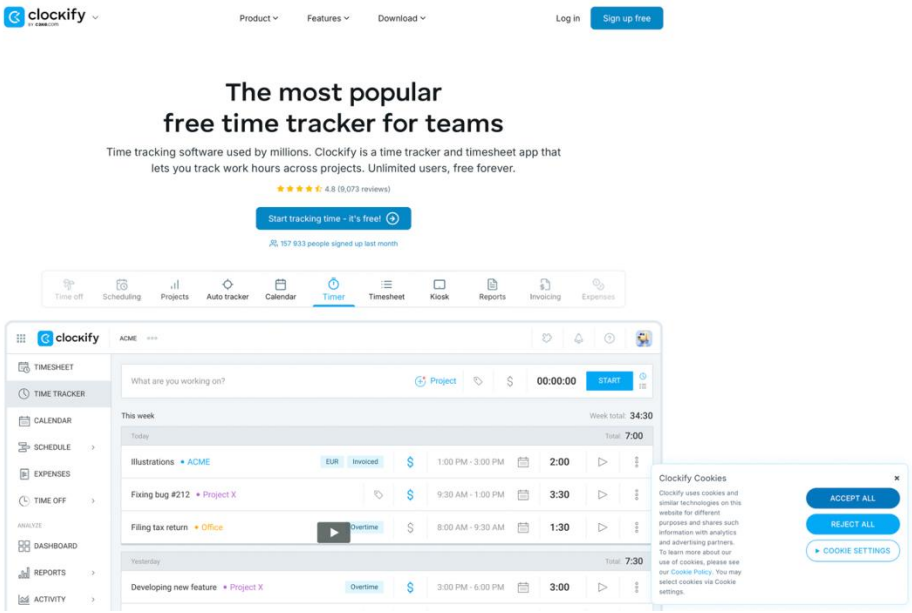

Рис. 1.3 – Таймтрекер Clockify з основним списком активностей працівника.

Система QuickBooks Time надає функціональність не лише для фіксації часу, а й для інтеграції з бухгалтерським обліком та платіжними системами. Сервіс активно використовується для польових працівників і підтримує мобільні пристрої, GPS-трекинг, push-нотифікації та багаторівневу аналітику, що показано на рисунку 1.4.

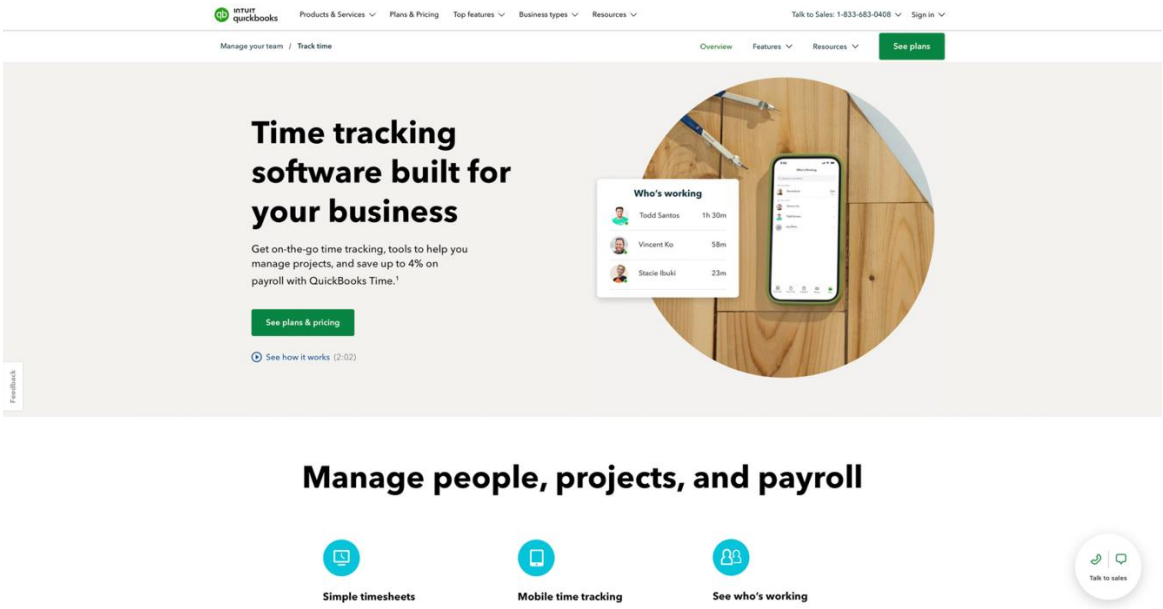

Рис. 1.4 – Головна сторінка QuickBooks Time з модулями обліку, персоналу та проєктів.

Платформа Apploye реалізує функціонал для аналізу продуктивності, керування активністю та автоматизації звітності. Особливістю є можливість збору метрик у реальному часі для віддалених працівників, аналізу неактивності та експортованих табелів (рис. 1.5). Apploye позиціонується як гібридне рішення із хмарним ядром і можливістю агентного розгортання.

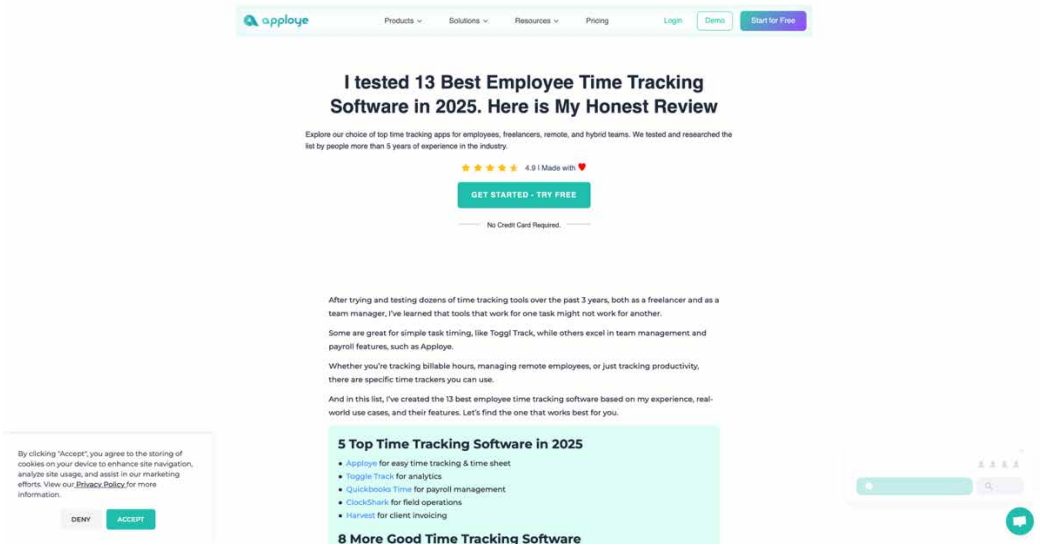

Рис. 1.5 – Огляд Apploye у порівняльному огляді кращих систем тайм-трекінгу.

Для проведення структурного аналізу на основі визначених критеріїв сформовано порівняльну таблицю, яка включає функціональні, технічні та

експлуатаційні характеристики систем і дозволяє візуалізувати переваги проєктованого рішення у таблиці 1.2.

Таблиця 1.2 - Порівняння існуючих рішень з розроблюваною системою

Критерій	Replicon	Clockify	QuickBooks Time	Apploye	Наша система
Тип архітектури	SaaS, корпоративна	SaaS, безкоштовна	SaaS + Payroll	SaaS + Agent	Гібридна (локальний клієнт + API)
Авторизація	LDAP, OAuth2	Email/Google	Email/SSO	Email/Token	OAuth2 + RBAC
Підтримка політик обліку	Гнучка	Обмежена	Базова	Базова	Динамічні політики через конфіг. файли
AI/ML-аналіз	Частково	Відсутній	Відсутній	Обмежений	Аналіз аномалій, ML-шаблони
Генерація табелів	Так	Так	Так	Так	Так + автоматичне формування звітності
Інтеграція з ERP	SAP, Oracle	API	QuickBooks	API	REST API, можливість кастомізації
Локальний режим	Ні	Ні	Ні	Ні	Так (десктоп через PyQt6)
Візуалізація	Web-Dashboard	Web-UI	Web + Mobile	Web + Agent View	PyQt-графіка + інтерактивні таблиці
Ліцензія	Комерційна	Freemium	Платна	Freemium	Відкрита для внутрішнього використання

Як показує аналіз, більшість існуючих рішень працюють у повністю хмарному режимі, не мають розгалуженої підтримки сценаріїв з автономним зберіганням, обмежені в налаштуванні політик обліку та не використовують алгоритмічний аналіз поведінкових аномалій. Запропонована система враховує ці обмеження й орієнтована на підприємства, яким необхідна гнучка, адаптована під політики організації, десктопна система з можливістю централізованого адміністрування, офлайн-режиму.

1.3 Моделювання предметної області

Процес моделювання предметної області є критично важливим етапом, що дозволяє формалізувати функціональну структуру майбутньої системи, визначити межі відповідальності суб'єктів взаємодії, сценарії використання та внутрішню логіку обробки подій. У межах цієї роботи моделювання здійснюється з використанням нотацій UML, які забезпечують уніфіковане представлення різних аспектів системи: взаємодії користувачів, послідовностей обміну повідомленнями та логіки бізнес-процесів.

Діаграма прецедентів (Use Case Diagram) відображає взаємодію користувачів (акторів) із системою та дозволяє формалізувати основні функціональні сценарії, які повинна реалізовувати інтелектуальна система обліку робочого часу. На рисунку 1.6 подано варіанти використання, що охоплюють повний цикл взаємодії - від авторизації користувача до формування звітів, затвердження заявок та взаємодії з зовнішніми службами (SSO, ERP, нотифікації, календарі).

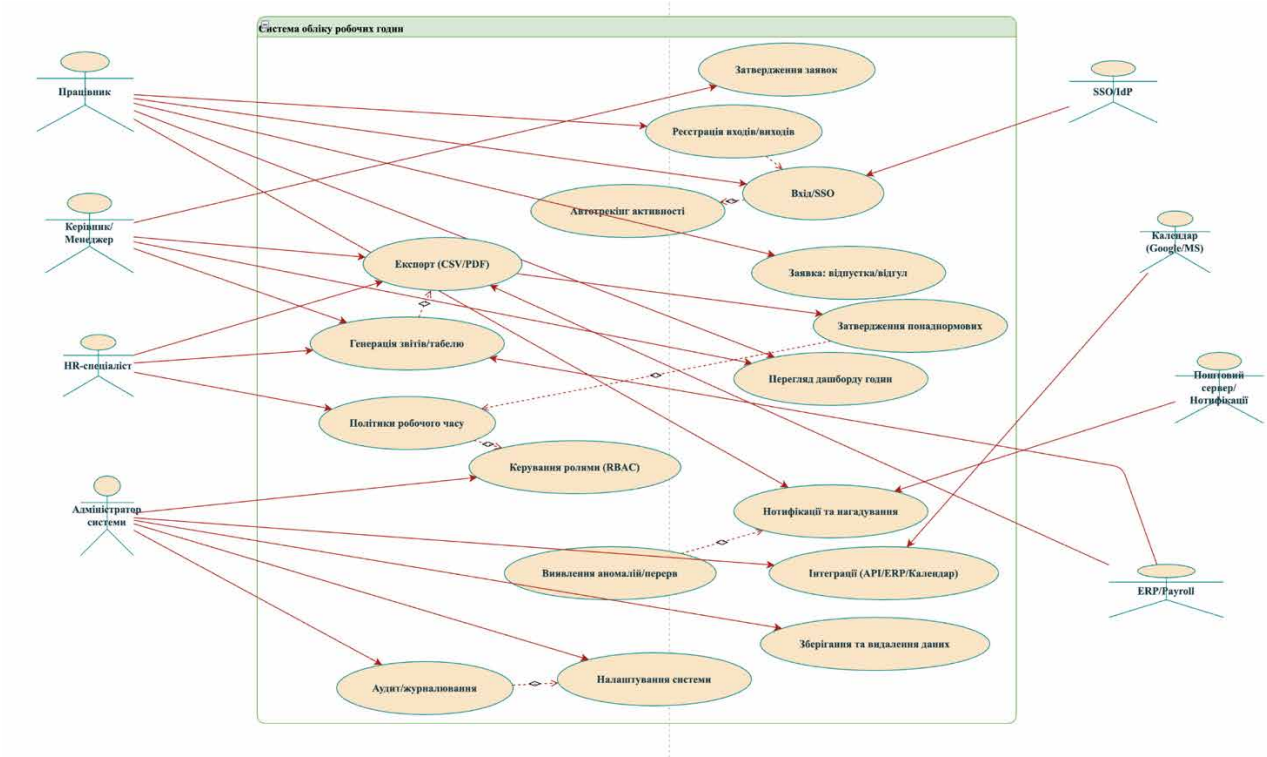

Рис. 1.6 – Діаграма прецедентів для інтелектуальної системи обліку робочих годин.

З діаграми видно, що кожен актор має визначений набір дій. Наприклад, працівник виконує вхід, фіксує присутність (Check-in/Check-out), надсилає заявки на відпустку або вихідний, а також переглядає особисту інформацію у вигляді дашборду. Менеджер - перевіряє та затверджує заявки, а також має доступ до аналітики продуктивності та системи політик. HR-спеціаліст генерує табелі, експортує їх, а адміністратор налаштовує політики, доступи, а також здійснює аудит.

Для деталізації динамічної взаємодії між компонентами було побудовано діаграму послідовності, яка ілюструє основний сценарій авторизації користувача та реєстрації часу початку зміни (рис. 1.7). Взаємодія реалізується через клієнтську частину (GUI), сервер (API), постачальника ідентичностей (IdP, що підтримує SSO/OIDC) та базу даних.

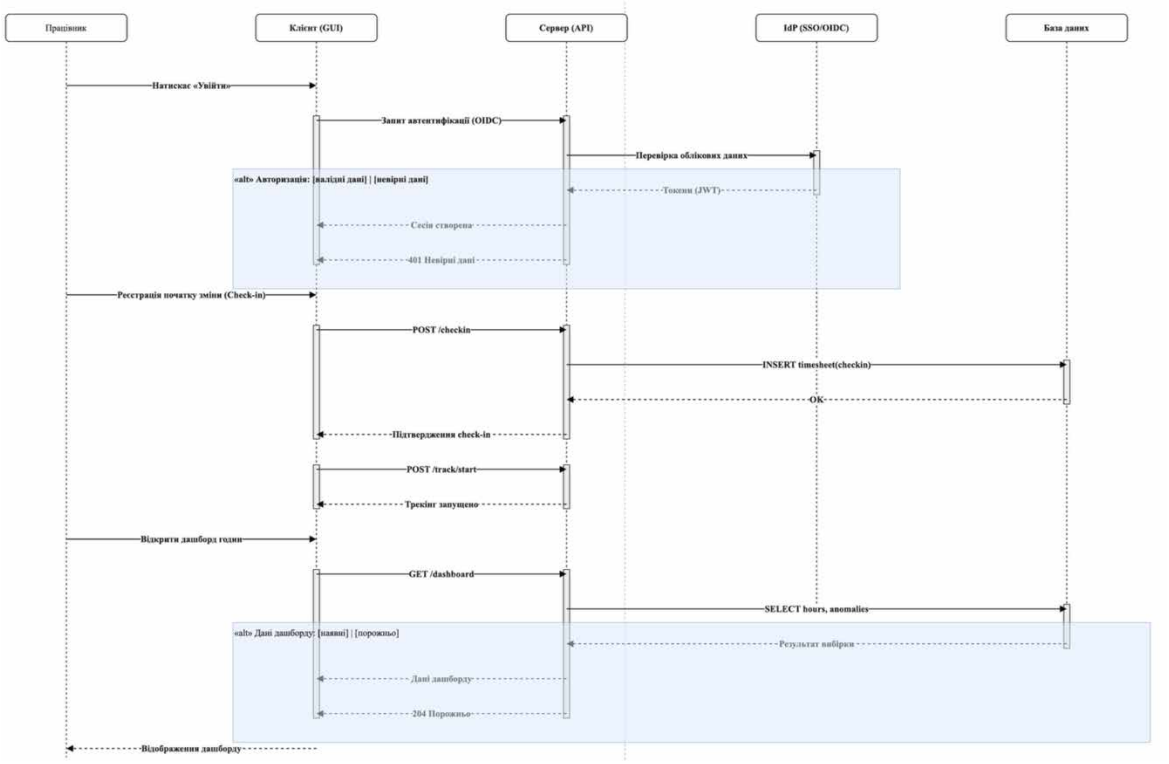

Рис. 1.7 – Діаграма послідовності для авторизації та початку зміни.

У сценарії передбачено використання стандартного протоколу OpenID Connect для автентифікації, після чого система зберігає запис у таблиці обліку часу (таблиця timesheet) та повертає підтвердження. Додатково реалізується перевірка валідності токенів, запуск активного трекінгу й візуалізація дашборду.

Діаграма активності яка представлена на рис.1.8 відображає послідовність дій користувача й системи в процесі роботи. Вона охоплює варіанти вдалого та помилкового входу, запуску трекінгу, періодичного оновлення стану, генерації звіту та завершення робочого дня.

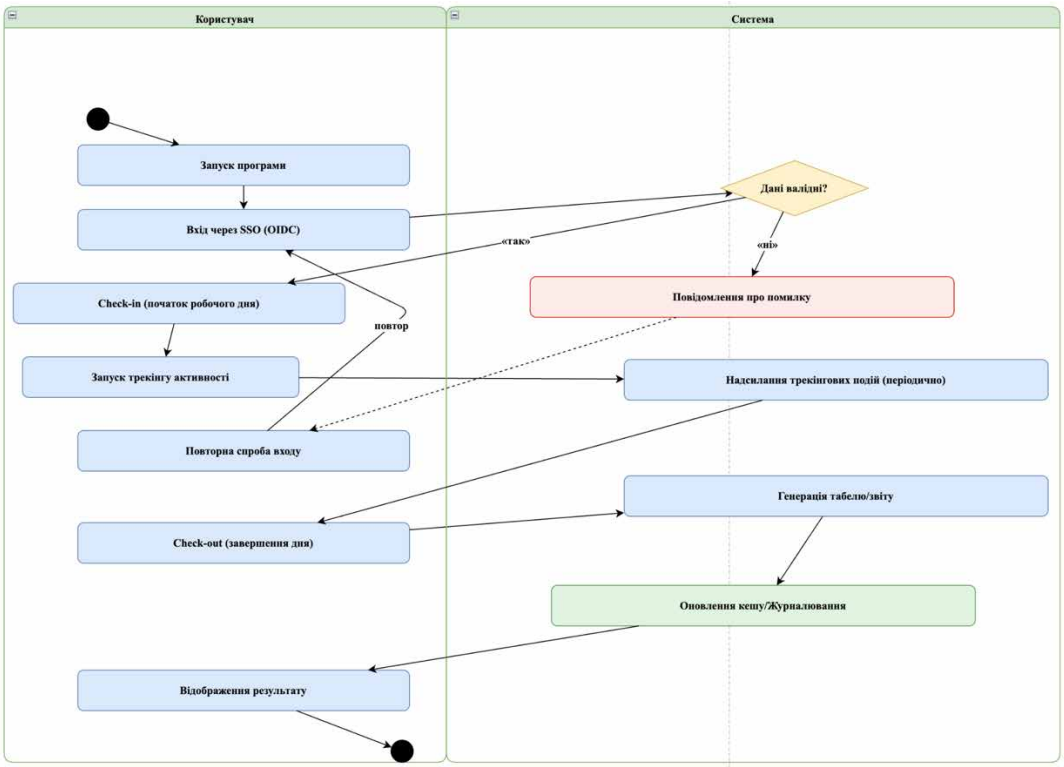

Рис. 1.8 – Діаграма активності користувача в системі обліку робочого часу.

У випадку невірних облікових даних користувач отримує повідомлення про помилку й може повторити спробу. У разі успішної авторизації запускається трекінг, що надсилає події на бекенд, а після завершення робочого дня генерується підсумковий табель з подальшим оновленням журналів активності.

1.4 Аналіз вимог програмної системи

Визначення вимог є критичним етапом, що забезпечує чітке формування функціоналу, очікуваної продуктивності, інтерфейсів, правил доступу та поведінки системи в граничних ситуаціях. Аналіз вимог охоплює три ключові категорії: функціональні, нефункціональні та технічні.

Функціональні вимоги це про те що саме має робити система, нефункціональні - якими властивостями має володіти, *а* технічні - якими засобами буде реалізовано проєктне рішення.

Функціональні вимоги було сформульовано на основі сценаріїв використання, моделі прецедентів (рис. 1.5), а також бізнес-правил керування обліком часу, трекінгу активностей та обробки заявок. Вимоги подано у таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги до системи

№	Назва функції	Опис
F1	Вхід через SSO/OIDC	Підтримка єдиного входу через корпоративний IdP
F2	Check-in/Check-out	Реєстрація початку і завершення робочого дня
F3	Автоматичний трекінг активності	Збір подій, дій та взаємодій користувача з системою
F4	Подання заявок	Надсилання запитів на відпустку/відгул/зміщення графіку
F5	Затвердження заявок	Керівник може погоджувати або відхиляти запити працівників
F6	Генерація табелів і звітів	Побудова звітів по годинах, затримках, аномаліях
F7	Нотифікації і нагадування	Автоматичне інформування через email/вбудовані канали
F8	Інтеграція з ERP/Payroll/Календарями	Синхронізація з корпоративними інструментами
F9	Аудит і журналювання	Запис усіх дій користувачів та адміністраторів
F10	Управління політиками обліку	Налаштування правил підрахунку робочого часу

До нефункціональних вимог належать обмеження і властивості, що визначають якість системи: її продуктивність, масштабованість, безпека та доступність. Ці вимоги критичні для успішного функціонування в умовах реального навантаження. Вимоги систематизовано в таблиці 1.3.

Таблиця 1.3 – Нефункціональні вимоги до системи

№	Категорія	Вимога
N1	Продуктивність	Обробка запиту < 500 мс при 500 одночасних користувачах
N2	Висока доступність	Uptime системи ≥ 99.5%

Продовження таблиці 1.3

N3	Безпека	Відповідність стандарту OWASP Top 10, шифрування JWT, RBAC
N4	Масштабованість	Горизонтальне масштабування бекенду через контейнеризацію
N5	Спостережуваність	Вбудовані метрики (Prometheus), логування (structured JSON)
N6	Портативність	Підтримка Windows/Linux/macOS (через Qt та Python)
N7	Відмовостійкість	Резервне копіювання таблиць БД щогодини, реплікація журналів
N8	Інтероперабельність	REST API з OpenAPI-документацією

Технічні вимоги відображають інструментарій, середовище розробки, протоколи взаємодії, архітектуру даних і способи інтеграції. Вони є відправною точкою для реалізації архітектури на рівні компонентів і взаємодії між підсистемами. Вимоги представлені у таблиці 1.4.

Таблиця 1.4 – Технічні вимоги до системи

№	Категорія	Технології / Платформи / Інструменти
T1	Мова програмування	Python 3.11+
T2	GUI	PyQt6
T3	Backend/API	FastAPI + Uvicorn
T4	База даних	PostgreSQL (primary), SQLite (offline mode)
T5	SSO	OIDC (Google, Azure AD), JWT
T6	Моніторинг	Prometheus + Grafana, логування – structlog
T7	CI/CD	GitHub Actions, Docker, Docker Compose
T8	Документація	OpenAPI (Swagger), MkDocs
T9	API	REST (JSON), Webhook для синхронізації з ERP
T10	Інтеграції	Google Calendar API, SMTP сервер, ERP/Payroll адаптери через REST

Сформульовані вимоги охоплюють весь спектр функціональних і нефункціональних характеристик, необхідних для побудови надійної, безпечної та масштабованої системи обліку робочих годин. Визначені технічні засоби реалізації дозволяють ефективно інтегрувати рішення у вже наявну інфраструктуру підприємства, а також забезпечити його автономність у випадку офлайн-сценаріїв.

Чітка класифікація вимог стане основою для побудови компонентної, логічної та фізичної архітектури системи, що буде реалізована у наступних етапах.

1.5 Постановка завдання

У результаті аналізу предметної області було виявлено основні організаційні та технічні особливості процесів обліку робочого часу на підприємстві, вивчено наявні технологічні рішення та нормативні підходи до побудови подібних систем. Було визначено перелік функціональних сценаріїв, класифіковано типові аномальні ситуації (запізнення, пропуски, перевищення норм), сформовано вимоги до захищеності, доступності, масштабованості та інтеграційної сумісності майбутнього програмного забезпечення.

Здійснено формалізацію варіантів використання на основі UML-нотацій, зокрема побудовано діаграми прецедентів, послідовності та активності, що описують ключові бізнес-процеси: авторизацію через SSO, фіксацію присутності, обробку заявок, формування табелів і звітності, взаємодію з календарями та зовнішніми ERP-модулями. Також було сформовано структуровані таблиці функціональних, нефункціональних і технічних вимог, які охоплюють архітектуру, безпеку, інтерфейси, інструментарій розробки, протоколи, бази даних і механізми розгортання.

На основі визначених функціональних сценаріїв і обраної технологічної стратегії, сформульовано перелік конкретизованих завдань, реалізація яких забезпечує перехід до етапу архітектурного та програмного проєктування:

– реалізувати клієнтську частину системи на основі бібліотеки PyQt6, з підтримкою багатовіконної логіки, адаптивного компонування інтерфейсів і локального зберігання конфігурацій;

– створити серверну частину з використанням FastAPI**,** орієнтовану на REST-архітектуру та асинхронну обробку запитів;

– сформувати архітектуру взаємодії API, що включає маршрути авторизації, обліку часу, генерації звітності, керування заявками та політиками;

– побудувати сховище даних на основі PostgreSQL з логічним проєктуванням таблиць для подій, заявок, табелів, користувачів, політик та журналів дій;

– реалізувати черги задач і асинхронну обробку через Celery, з обробниками для трекінгу, надсилання сповіщень і підготовки звітів;

– застосувати Redis як кешуючий шар, що знижує навантаження на основну БД та оптимізує час відповіді на запити;

– забезпечити автентифікацію через OIDC, з обробкою JWT-токенів і підтримкою єдиного входу (SSO) для корпоративного середовища;

– впровадити механізми інтеграції з Google Calendar API для синхронізації подій і з SMTP-серверами для нотифікацій;

– реалізувати API-документацію у форматі OpenAPI з використанням автоматичної генерації опису через FastAPI;

– підготувати структуру CI/CD-конвеєра з використанням Docker і GitHub Actions для автоматичного тестування, збирання, розгортання та оновлення сервісів.

Формулювання цих завдань відображає перехід до рівня реалізації архітектурних моделей, побудованих на основі проведеного системного аналізу, і забезпечує послідовність у побудові цілісної, керованої та масштабованої системи.

1.6 Висновки до першого розділу

У результаті проведеного системного аналізу предметної області було встановлено, що сучасні підприємства стикаються з істотними обмеженнями при використанні традиційних систем табелювання, які не забезпечують достатнього рівня автоматизації, адаптивності та інтеграції з корпоративною інфраструктурою. Аналіз існуючих рішень показав, що доступні SaaS-платформи орієнтовані переважно на вебсередовище, мають обмежені можливості локальної роботи, недостатню підтримку гнучких політик трудового часу та фактично не застосовують алгоритмічні методи виявлення аномалій, що знижує їх відповідність сучасним вимогам до продуктивності, прозорості та інформаційної безпеки підприємства.

Проведене моделювання предметної області з використанням UML-нотацій дозволило формалізувати основні бізнес-процеси системи, включно з авторизацією, обліком робочої активності, обробкою заявок, генерацією табелів і взаємодією з зовнішніми сервісами. Побудовані діаграми прецедентів, послідовності та активності відобразили структуру взаємодій між користувачами й підсистемами, окреслили межі відповідальності компонентів та забезпечили основу для подальшого архітектурного проєктування.

Структурований аналіз функціональних, нефункціональних і технічних вимог дав змогу сформувати цілісне бачення очікуваної системи, визначити її ключові можливості, обмеження та критерії якості. Було чітко окреслено вимоги до продуктивності, доступності, масштабованості, безпеки, інтероперабельності та інструментального забезпечення, включаючи використання Python, PyQt6, FastAPI, PostgreSQL, Redis, Celery та зовнішніх API.

Перший розділ сформував комплексне теоретичне підґрунтя для проєктування інтелектуальної системи обліку робочого часу, окресливши її проблемний контекст, недоліки існуючих рішень, можливості технічного вдосконалення та необхідні вимоги до архітектури. Отримані результати забезпечують цілісну основу для переходу до моделювання даних, побудови архітектури компонентів і подальшої реалізації системи у наступних розділах.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних сформована за принципами 3НФ з уніфікованою схемою ідентифікації (UUID для всіх первинних ключів), явними зовнішніми ключами, унікальними індексами для бізнес-ідентифікаторів та чітко визначеними кардинальностями. Модель охоплює три групи сутностей: керування доступом (користувачі, ролі, дозволи, токени), облік часу (табелі, записи табелів, заявки, аномалії, політики, зміни/розклади) та транзакційні сервіси (сповіщення, аудит). Узагальнений вигляд структури наведено на рис. 2.1.

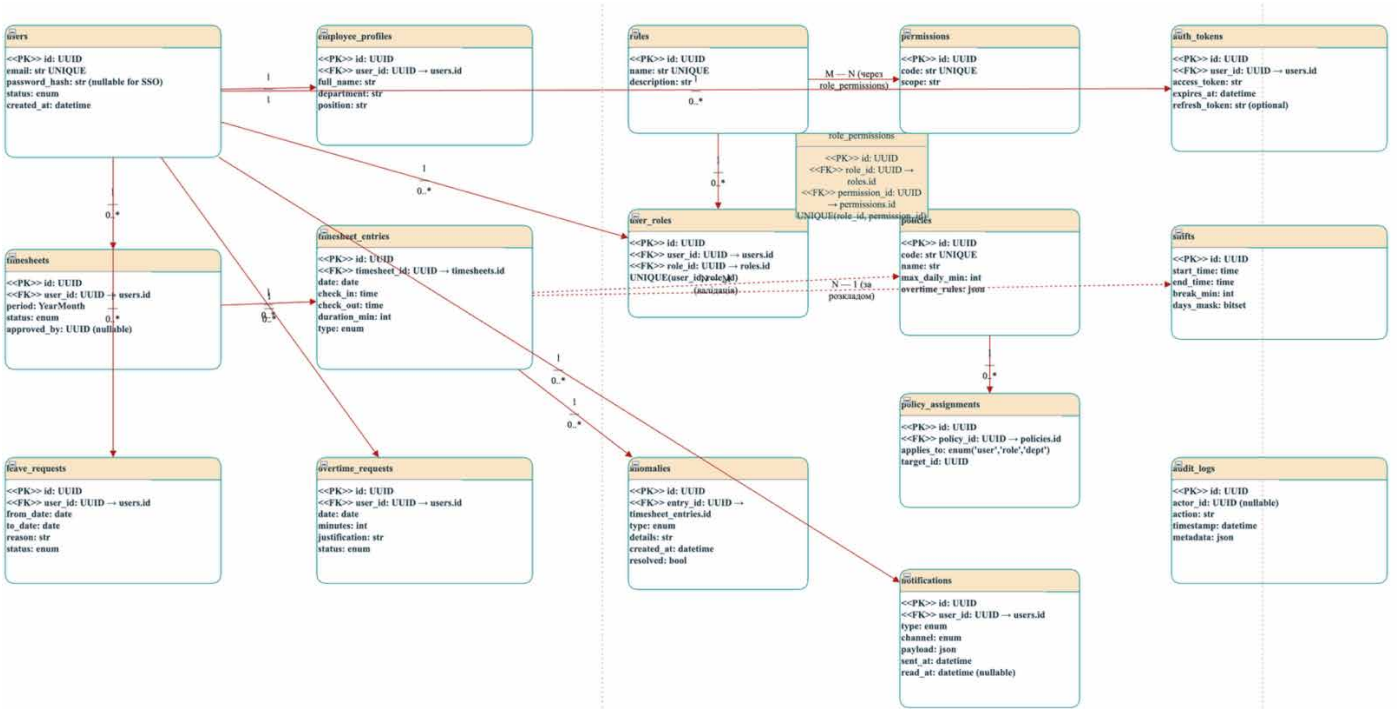

Рисунок 2.1 – ER-діаграма логічної моделі даних інтелектуальної системи обліку робочих годин.

На діаграмі (рис. 2.1) відображено ключові зв’язки:

users 1-1 employee_profiles, users 1-* timesheets, timesheets 1-* timesheet_entries, timesheet_entries 1-* anomalies, users 1-* leave_requests, users 1-* overtime_requests, users 1-* notifications. Багато-до-багатьох реалізовано через асоціативні таблиці

user_roles (користувачі↔ролі) та role_permissions (ролі↔дозволи). Політики обліку описує policies, призначення політик здійснюється через policy_assignments (ціль: користувач / роль / підрозділ). Для сесій та інтеграції з IdP використовується auth_tokens, для спостережуваності - audit_logs. Додатково передбачено логічне посилання записів табеля на довідник змін shifts (розклад) і перевірку відповідності політикам.

Структуровані характеристики сутностей узагальнено у табл. 2.1.

Таблиця 2.1 – Сутності логічної моделі, ключі та призначення

Сутність	Призначення	Первинний ключ (PK)	Зовнішні ключі (FK) / унікальні обмеження
users	Облікові записи користувачів, статус, створення	id: UUID	email UNIQUE; зв’язок 1—1 з employee_profiles
employee_profiles	ПІБ, відділ, посада	id: UUID	user_id → users.id (UNIQUE)
roles	Ролі доступу	id: UUID	name UNIQUE
permissions	Дозволи/області дії	id: UUID	code UNIQUE
user_roles	Зв’язок користувач—роль	id: UUID	user_id → users.id, role_id → roles.id, UNIQUE(user_id, role_id)
role_permissions	Зв’язок роль—дозвіл	id: UUID	role_id → roles.id, permission_id → permissions.id, UNIQUE(role_id, permission_id)
auth_tokens	Сесійні/refresh-токени	id: UUID	user_id → users.id
timesheets	Табелі за період	id: UUID	user_id → users.id, approved_by (nullable)
timesheet_entries	Події дня: check-in/out, тривалість, тип	id: UUID	timesheet_id → timesheets.id
anomalies	Відхилення: запізнення, пропуски, дублікати	id: UUID	entry_id → timesheet_entries.id
leave_requests	Заявки на відпустку/відгул	id: UUID	user_id → users.id

Продовження таблиці 2.1

overtime_requests	Заявки на понаднормові	id: UUID	user_id → users.id
policies	Правила обліку (норма дня, овертайм)	id: UUID	code UNIQUE
policy_assignments	Призначення політики адресату	id: UUID	policy_id → policies.id, applies_to ∈ {user, role, dept}, target_id: UUID
shifts	Розклад змін, перерви, маска днів	id: UUID	—
notifications	Надсилані сповіщення (тип/канал/payload)	id: UUID	user_id → users.id
audit_logs	Аудит дій, метадані	id: UUID	actor_id (nullable)

З технічних міркувань для всіх таблиць, що беруть участь у приєднаннях за часом, застосовуються композитні індекси: timesheet_entries(timesheet_id, date), leave_requests(user_id, from_date, to_date), overtime_requests(user_id, date). Для швидкої авторизації використовуються індекси users(email) та auth_tokens(user_id, expires_at). Посилальна цілісність встановлюється як ON DELETE CASCADE для службових асоціацій (user_roles, role_permissions, auth_tokens, timesheets→timesheet_entries, timesheet_entries→anomalies) і як ON DELETE SET NULL для timesheets.approved_by з метою збереження історичних записів. Політики застосовуються на рівні бізнес-логіки; призначення політики зберігається у policy_assignments, що дозволяє централізовано переобчислювати тривалість і визначати овертайм.

Запропонована логічна модель (рис. 2.1, табл. 2.1) забезпечує повноту представлення предметної області, мінімізує дублювання через нормалізацію, підтримує масштабування за рахунок чітких асоціативних таблиць та індексів, а також забезпечує розширюваність (нові типи аномалій, політик, каналів сповіщень додаються без впливу на існуючі зв’язки). Модель готова до переходу на фізичний рівень з визначенням типів полів БД, індексації та стратегій партиціювання табелів за періодом.

2.2 Діаграма класів і їхні кооперації

Логічні об’єкти домену формалізовано у вигляді класів із чітким розподілом відповідальностей (SRP), явними залежностями та інваріантами. Структуру класів і асоціацій наведено на рис. 2.2, де реалізовано основні агрегації та композиції.

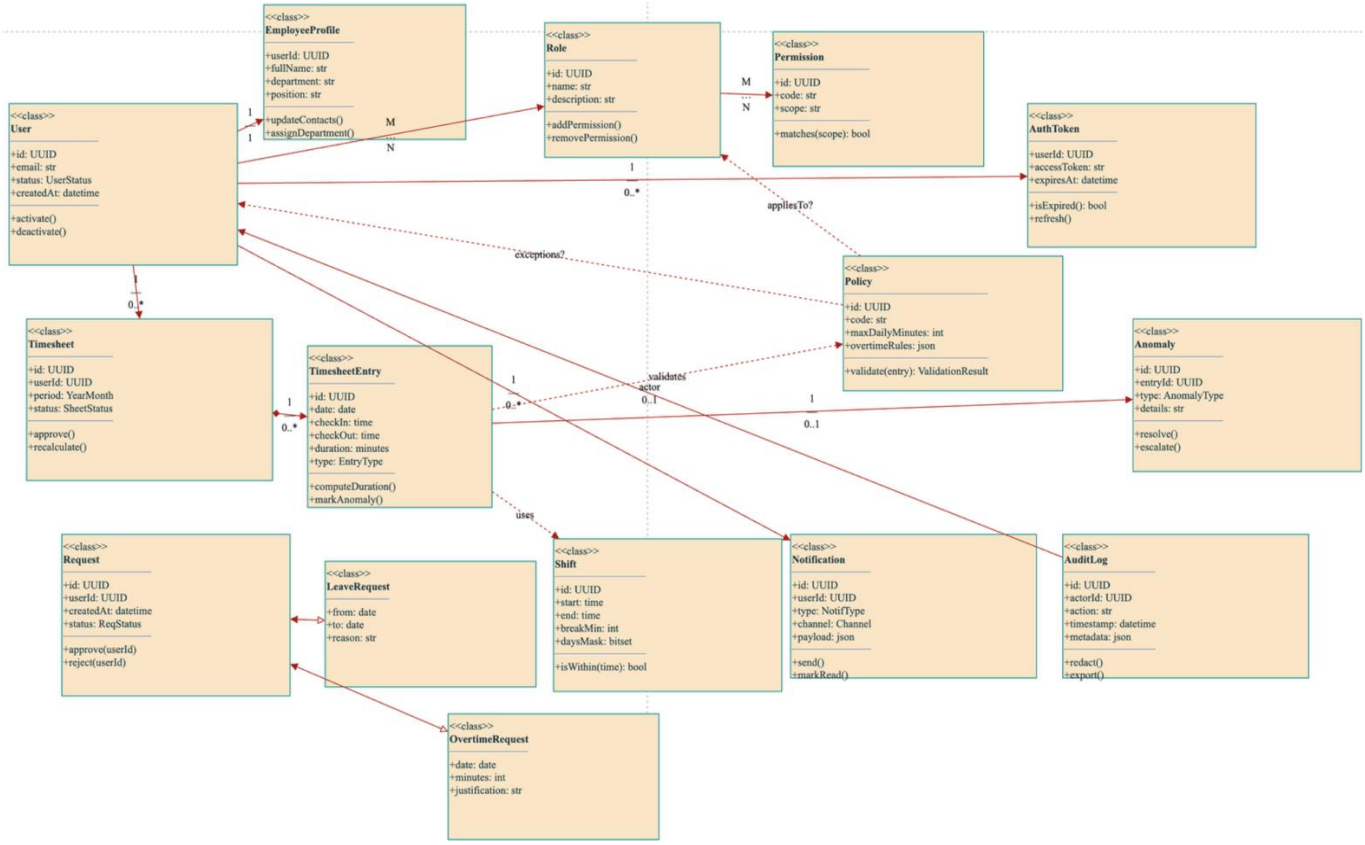

Рисунок 2.2 – Діаграма класів інтелектуальної системи обліку робочих годин.

User-EmployeeProfile (1..1), User-Timesheet (1..*), Timesheet-TimesheetEntry (1..*), TimesheetEntry- Anomaly (0..*), User-Request (1..*) з конкретизаціями LeaveRequest, OvertimeRequest. Контроль доступу реалізовано через ролі та дозволи: User- Role (M..N) і Role-Permission (M..N) за участі асоціативних об’єктів. Політики обліку інкапсульовано у класі Policy, який надає сервісну валідацію validate(entry) для записів табеля, а розклад- у класі Shift (перевірка належності isWithin(time)). Стан сесії та інтеграція з IdP представлені класом AuthToken. Сповіщення та аудит оформлено окремими технічними сутностями Notification і AuditLog.

Операційні сценарії взаємодії між об’єктами подано у вигляді кооперацій. Перший сценарій- реєстрація початку зміни (check-in): GUI ініціює команду, API

делегує дію сервісу табелювання, формується запис у TimesheetEntry, генерується подія аудиту; послідовність повідомлень і учасники показані на рис. 2.3. Другий сценарій - побудова звіту: API створює задачу генерації, сервіс звітів агрегує дані з БД, зберігає артефакт у об’єктному сховищі (S3/MinIO), відправляє посилання через SMTP, API повертає клієнту URL; кооперацію наведено на рис. 2.4. Третій сценарій - виявлення аномалій: сервіс трекінгу подій передає запис активності до аналізатора, який звертається до Policy, формує об’єкт Anomaly та ініціює Notification; взаємодії показані на рис. 2.5.

Семантику класів, ключові атрибути й публічні методи зведено до табл. 2.2 (посилання на таблицю в тексті гарантує коректну інтерпретацію моделей під час реалізації).

Таблиця 2.2 – Класи, відповідальності та інтерфейси

Клас	Відповідальність (R)	Ключові атрибути	Публічні методи / інваріанти
User	Ідентичність користувача, зв’язки з профілем, ролями та артефактами обліку	id:UUID, email, status, createdAt	activate(), deactivate(); інваріант: email UNIQUE
EmployeeProfile	Персональні дані та оргструктура	userId, fullName, department, position	updateContacts(), assignDepartment(); інваріант: userId унікальний
Role	Рольові групи доступу	id, name, description	addPermission(), removePermission(); інваріант: name UNIQUE
Permission	Окремі дозволи/області дії	id, code, scope	matches(scope):bool; інваріант: code UNIQUE
AuthToken	Маркери доступу та їх життєвий цикл	userId, accessToken, expiresAt	isExpired(), refresh()
Timesheet	Табель періоду користувача	id, userId, period, status	approve(), recalculate(); інваріант: період єдиний на userId
TimesheetEntry	Подія дня: check-in/out, тривалість, тип	id, timesheetId, date, checkIn, checkOut, duration, type	computeDuration(), markAnomaly(); інваріант: duration ≥ 0
Policy	Бізнес-правила обліку часу	id, code, maxDailyMinutes, overtimeRules	validate(entry):ValidationResult

Продовження таблиці 2.2

Anomaly	Факт відхилення та ескалація	id, entryId, type, details	resolve(), escalate()
Request	Узагальнена заявка працівника	id, userId, createdAt, status	approve(userId), reject(userId)
LeaveRequest	Конкретизація відпустки/відгулу	from, to, reason	наслідує Request
OvertimeRequest	Конкретизація понаднормових	date, minutes, justification	наслідує Request
Shift	Розклад зміни та перерви	start, end, breakMin, daysMask	isWithin(time):bool
Notification	Канали інформування	id, userId, type, channel, payload, sentAt	send(), markRead()
AuditLog	Трасування дій та подій	id, actorId, action, timestamp, metadata	read(), export()

На рисунках 2.3–2.5 наведено кооперації, що відображають конкретні сценарії взаємодії сервісів у рамках типової архітектури. Рисунок 2.3 моделює взаємодію при події Check-in, де користувач надсилає запит через API, який передається до сервісу табелювання. Після фіксації запису у БД генерується подія аудиту, що реєструє сам факт події.

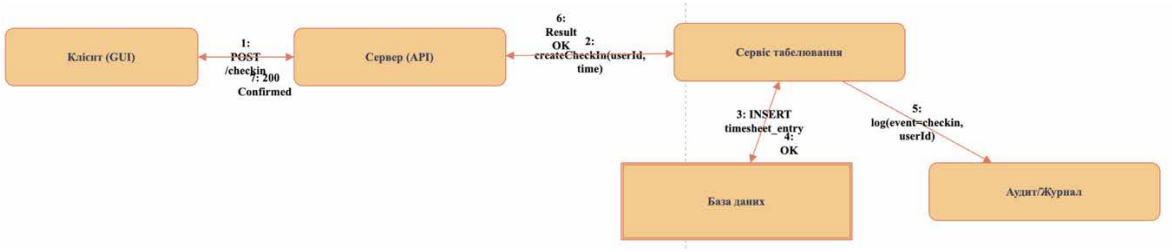

Рисунок 2.3 – Кооперація «Фіксація початку робочого дня».

У другому випадку, на рис. 2.4, зображено взаємодію при побудові звіту - запит надходить до API, звідки делегується сервісу звітів. Дані вибираються з бази, агрегуються, зберігаються у форматі звіту та надсилаються поштовим сервером. Це дозволяє відокремити сервіс формування звітності від основного бізнес-потоку, що позитивно впливає на масштабованість.

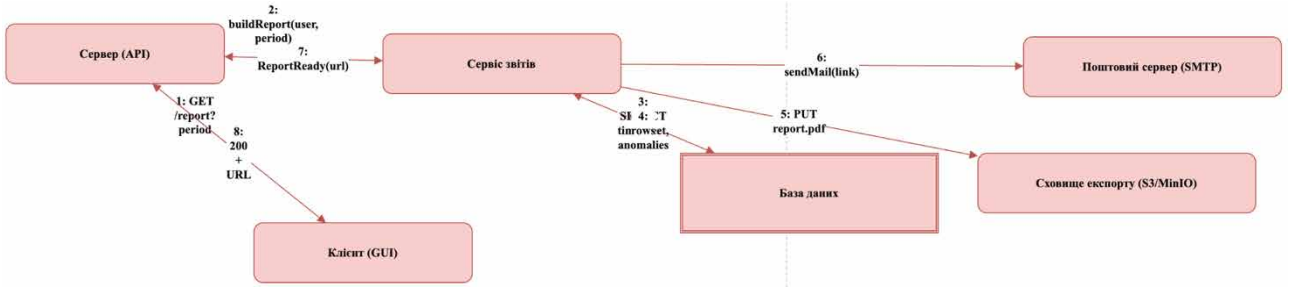

Рисунок 2.4 – Кооперація «Формування та доставка звіту».

На рисунку 2.5 представлено сценарій виявлення аномалій. Сервіс трекінгу активності надсилає події до аналізатора, який у свою чергу звертається до політик (Policy) для перевірки відповідності. У разі відхилень генерується запис аномалії в БД і формується сповіщення відповідальному користувачу.

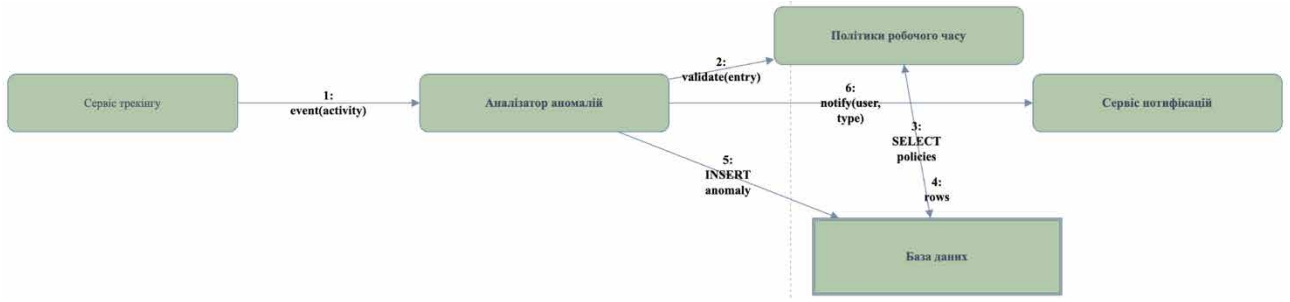

Рисунок 2.5 – Кооперація «Виявлення аномалій у графіку роботи».

Розглянуті кооперації підтверджують логічну цілісність моделі та узгодженість взаємодій між компонентами. Завдяки чіткому поділу на сервіси -табелювання, звітність, аудит, аналіз аномалій, нотифікації - система залишається масштабованою, спостережуваною та такою, що легко тестується. Усі взаємодії мають явні контракти, що дозволяє реалізувати ці сервіси як незалежні мікрокомпоненти.

2.3 Діаграма компонентів

Архітектура програмного забезпечення побудована за принципами сервісної декомпозиції з чітким розмежуванням обов’язків між компонентами на трьох рівнях: клієнтському, сервісному та рівні даних. Такий підхід сприяє масштабованості, модульності, зручності тестування і оновлення окремих частин системи без впливу на інші підсистеми.

На клієнтському рівні розташовується десктопний застосунок, реалізований з використанням бібліотеки PyQt6, що забезпечує взаємодію з користувачем у графічному режимі. Вбудований кеш дозволяє працювати в автономному режимі, синхронізуючи дані після відновлення підключення.

Сервісний рівень включає API-шлюз, який виконує роль BFF (Backend For Frontend), надаючи уніфіковану точку доступу до всіх сервісів. Основні мікросервіси реалізують функціональність трекінгу часу (Timesheet Service), сповіщень (Notification Service), управління ролями й політиками (RBAC / Policy Service), аудиту дій користувачів (Audit & Logging), генерації звітів, а також шлюз інтеграції для взаємодії з зовнішніми ERP/Payroll або календарними сервісами.

На рівні даних зосереджено інтеграцію з базою PostgreSQL, Redis-кешем, сховищем S3/MinIO для експорту звітів, брокером повідомлень RabbitMQ/NATS, а також модулі автентифікації, планування задач і адаптери до зовнішніх API (OIDC, SMTP, Google Calendar тощо). Логування реалізується з використанням систем типу ELK або Loki, що забезпечує спостережуваність поведінки системи.

На рис. 2.6 представлено повну діаграму компонентів системи з візуалізацією каналів зв’язку між підсистемами та зовнішніми інтеграціями.

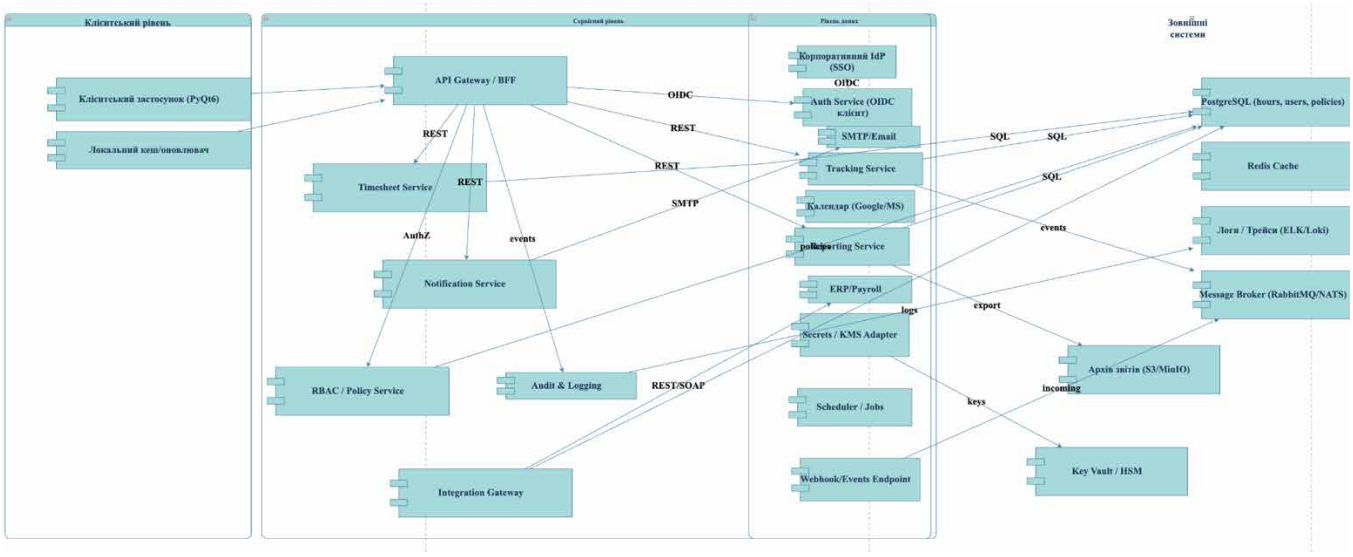

Рисунок 2.6 – Діаграма компонентів системи обліку робочих годин

Компоненти згруповані за рівнями: клієнтським, сервісним, рівнем даних та зовнішніми системами. Відображено всі канали взаємодії: REST, SMTP, SQL, events, logs, export.

Для забезпечення детальної інтерпретації архітектури компонентів, у таблиці 2.3 подано короткий опис основних компонентів системи:

Таблиця 2.3 – Компоненти архітектури системи

Назва компонента	Рівень	Функціональне призначення
Клієнтський застосунок (PyQt6)	Клієнтський	Графічний інтерфейс користувача, взаємодія з API, локальний кеш
API Gateway / BFF	Сервісний	Уніфікований шлюз доступу до мікросервісів, маршрутизація запитів
Timesheet Service	Сервісний	Обробка подій check-in/out, запис у табелі, підрахунок тривалості змін
Notification Service	Сервісний	Генерація повідомлень про події, надсилання email/інших сповіщень
RBAC / Policy Service	Сервісний	Керування ролями, дозволами, політиками робочого часу
Audit & Logging	Сервісний	Журналювання дій користувачів і системних подій
Integration Gateway	Сервісний	Комунікація з зовнішніми ERP/Calendar API, адаптація форматів
PostgreSQL	Зовнішня система	Основне сховище користувачів, записів часу, політик, логів
Redis Cache	Зовнішня система	Тимчасове зберігання сесій, кешування даних
Message Broker (RabbitMQ/NATS)	Зовнішня система	Передача асинхронних повідомлень між сервісами
Сховище експорту (S3/MinIO)	Зовнішня система	Зберігання сформованих звітів у PDF/CSV форматі
Поштовий сервер (SMTP)	Зовнішня система	Надсилання email-нотифікацій
Корпоративний IdP (OIDC)	Зовнішня система	Верифікація користувачів через єдину систему ідентифікації
Secrets / KMS Adapter	Рівень даних	Зберігання ключів, токенів, конфіденційних даних

Обрана архітектура (представлення у діаграмі компонентів) забезпечує горизонтальну масштабованість, модульність, підвищену відмовостійкість і адаптивність до змін у вимогах або інтеграційних API. Використання REST і подій як основних каналів взаємодії дозволяє будувати розширювану екосистему з

можливістю інкапсуляції бізнес-логіки в окремих сервісах без порушення цілісності загального середовища.

2.4 Діаграма пакетів

Архітектура програмної системи структурована за принципами модульності, із суворим розмежуванням відповідальностей між пакетами, що підвищує контрольованість, масштабованість і можливість ізольованого тестування. Побудова пакувальної структури відображає логіку предметної області, застосування шаблонів DDD (Domain-Driven Design), а також вимоги до безпеки, повторного використання компонентів та автоматизованої перевірки.

На рис. 2.7 подано діаграму пакетів, що демонструє ключові залежності між модулями системи, зокрема взаємодію між клієнтським інтерфейсом, сервісами бізнес-логіки, інтеграційними адаптерами, інфраструктурними елементами та модулями налаштувань.

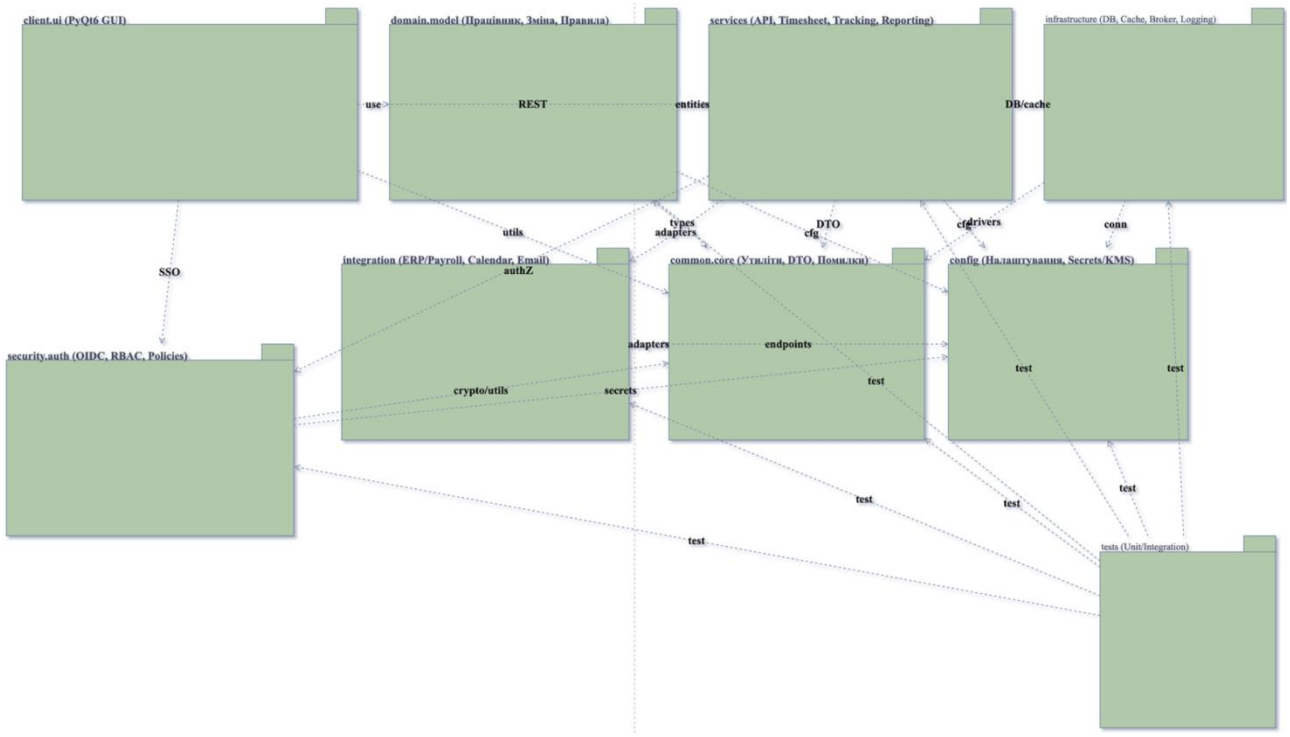

Рисунок 2.7 – Діаграма пакетів системи обліку робочих годин

Основу пакувальної структури становить domain.model, який включає сутності предметної області (користувачі, зміни, політики, ролі), що використовуються

сервісним шаром services. Останній реалізує ключову бізнес-логіку (обробка таблиць обліку, аномалій, трекінгу), оперуючи даними через типізовані DTO-структури, що визначені в пакеті common.core.

Пакет client.ui відповідає за реалізацію клієнтського інтерфейсу на основі PyQt6, з'єднується з backend через REST API, а також взаємодіє з механізмом автентифікації (OIDC) через security.auth.

Всі сторонні інтеграції (електронна пошта, календарі, ERP-системи, сховища) інкапсульовано в модулі integration, що спрощує їхню заміну або доповнення. Інфраструктурний рівень (infrastructure) абстрагує доступ до БД, кешів, брокерів подій і механізмів логування, використовуючи драйвери, які конфігуруються через окремий модуль config. Весь стек підтримує криптографічний захист (через Secrets/KMS), що доступний з будь-якого шару через стандартні інтерфейси.

Окремий акцент зроблено на модуль tests, що забезпечує покриття unit- та інтеграційними тестами на кожному з рівнів, із можливістю тестування конфігурацій, утиліт і зовнішніх викликів.

Відповідна таблиця 2.4 опису пакетів подана нижче для стислого перегляду структури.

Таблиця 2.4 – Тезисний опис пакетів розроблювальної системи

Назва пакету	Призначення
client.ui	Графічний інтерфейс користувача (PyQt6)
domain.model	Сутності предметної області (користувач, зміна, політика)
services	Бізнес-логіка: трекінг, табелювання, звіти
common.core	DTO, утиліти, обробка помилок
integration	Інтеграція з ERP, календарями, поштою, API
infrastructure	Доступ до бази даних, кешу, брокерів, логування
security.auth	OIDC, політики, ролі, RBAC
config	Конфігурації, адаптери секретів, KMS
tests	Unit- та інтеграційні тести, інфраструктурне тестування

Така архітектура дозволяє досягти високого рівня ізольованості компонентів, спрощує CI/CD-процеси та забезпечує гнучкість масштабування як на рівні сервісів, так і всередині окремих пакетів. Поділ на логічні шари гарантує підтримуваність, повторне використання та прозору трасованість залежностей.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій та інструментальних засобів реалізації

Архітектурна побудова інтелектуальної системи обліку робочих годин вимагає узгодженого вибору технологій, які забезпечують не лише функціональну повноту, а й стабільність, масштабованість, безпеку та можливість інтеграції з корпоративною інфраструктурою. Основні критерії вибору включали продуктивність при обробці великої кількості подій, підтримку асинхронних викликів, зручність розробки багаторівневих API, кросплатформеність клієнтського середовища та наявність зрілої екосистеми бібліотек. З урахуванням цих вимог було обрано стек технологій, поданий у таблиці 3.1, який забезпечує повну сумісність компонентів між собою та можливість подальшого розширення системи без втрати її функціональної цілісності.

Таблиця 3.1 – Вибір технологій та інструментальних засобів реалізації

№	Компонент системи	Технологія / інструмент	Призначення
1	Мова програмування	Python 3.11+	Основна мова реалізації логіки, підтримка асинхронності та розвинена стандартна бібліотека
2	Серверна частина	FastAPI + Uvicorn	Реалізація REST API, асинхронна обробка запитів, інтеграція з OpenAPI
3	Клієнтський застосунок	PyQt6	Побудова кросплатформеного графічного інтерфейсу користувача
4	Система керування базами даних	PostgreSQL / SQLite	Зберігання транзакційних даних і підтримка офлайн-режиму
5	Асинхронні черги	Celery + Redis / RabbitMQ	Виконання відкладених і паралельних завдань
6	Автентифікація та безпека	OIDC / JWT	Підтримка єдиного входу (SSO), токен-базована авторизація
7	Контейнеризація та CI/CD	Docker, GitHub Actions	Автоматизація збирання, тестування та розгортання сервісів

Продовження таблиці 3.1

8	Моніторинг і логування	Prometheus, Grafana, ELK	Спостережуваність, збір метрик, централізоване журналювання
9	Документація API	Swagger / MkDocs	Генерація специфікацій та користувацької документації
10	Інтеграційні адаптери	Google Calendar API, SMTP, ERP REST	Обмін даними з зовнішніми сервісами та корпоративними системами

Вибраний стек дозволяє поєднати сучасні підходи до побудови мікросервісної архітектури з високим рівнем ізоляції компонентів і підтримкою DevOps-практик. Використання Python 3.11 і FastAPI забезпечує низьку латентність, просту інтеграцію з бібліотеками машинного навчання та аналітики, а PyQt6 – нативну взаємодію з користувачем без залежності від браузера. Комбінація PostgreSQL і Redis гарантує баланс між транзакційністю та швидкодією кешу, а Docker-орієнтований підхід спрощує масштабування системи у хмарних середовищах. Застосування Prometheus і Grafana створює основу для побудови повноцінної спостережуваності, тоді як CI/CD-конвеєр із GitHub Actions мінімізує людський фактор при оновленнях і тестуванні. Сукупність наведених технологій формує цілісну екосистему, орієнтовану на надійність, гнучкість і тривалу підтримку продукту [10].

3.2 Архітектура системи

Архітектура інтелектуальної системи обліку робочих годин побудована за принципами сервісної декомпозиції та розподілу відповідальностей між логічними рівнями, що забезпечує масштабованість, спостережуваність, відмовостійкість і контрольованість виконання бізнес-процесів. На рисунку 3.1 наведено загальну структурну схему, яка ілюструє взаємодію між клієнтським застосунком, API-шлюзом, мікросервісами та рівнем даних. Кожен із рівнів має чітко визначені функції й комунікує через стандартизовані протоколи REST, асинхронні події або SQL-інтерфейси, що гарантує слабку зв’язність компонентів і можливість незалежного оновлення модулів.

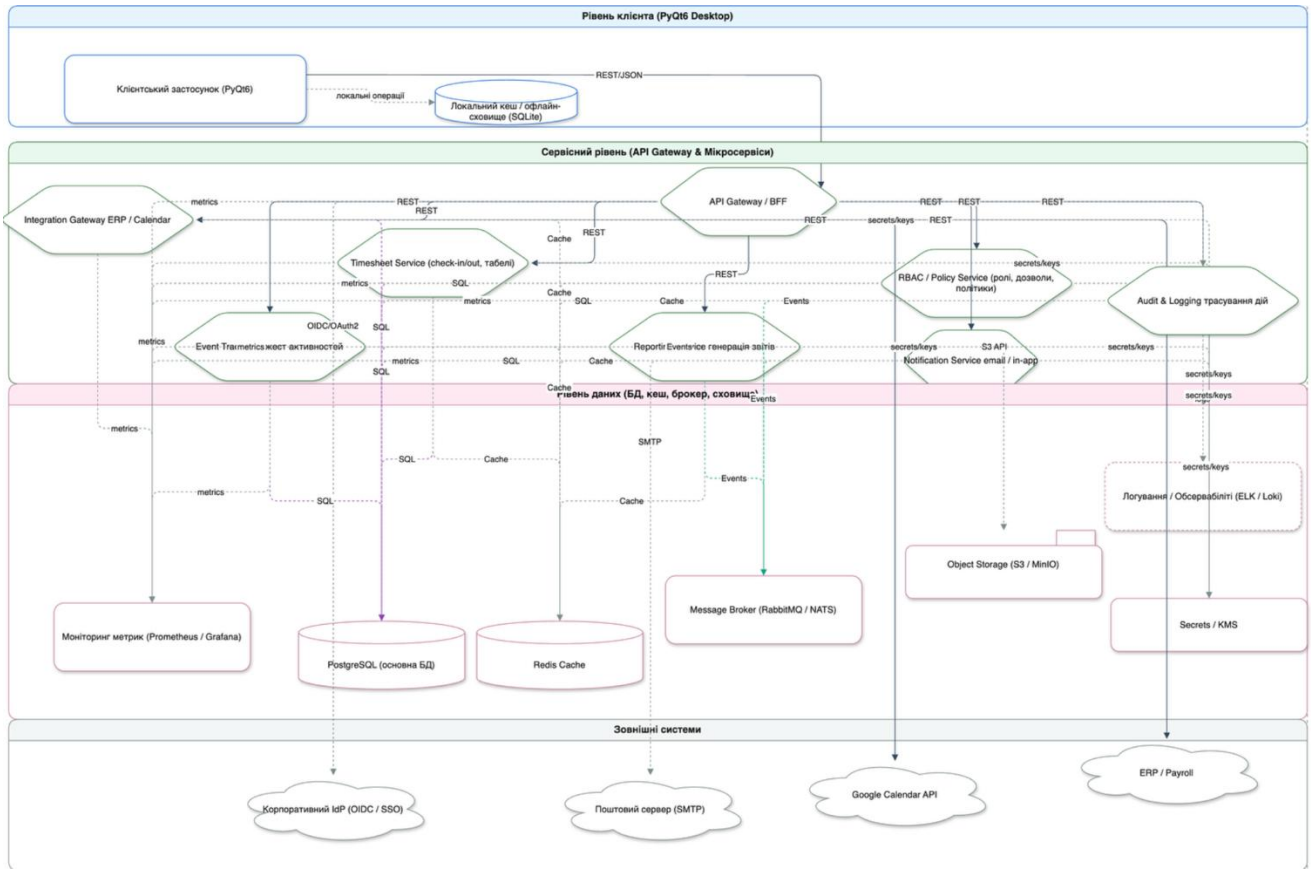

Рисунок 3.1 – Загальна архітектура інтелектуальної системи обліку робочих годин

На клієнтському рівні реалізовано десктопний застосунок на базі PyQt6, який забезпечує візуалізацію даних, авторизацію користувача через OIDC та роботу в автономному режимі завдяки вбудованому кешу SQLite. Комунікація з сервером здійснюється через JSON-запити до API Gateway / BFF, що виконує роль єдиної точки входу. Сервісний рівень включає мікросервіси Timesheet Service (реєстрація подій check-in/out та формування табелів), Reporting Service (генерація звітів і виявлення аномалій), Notification Service (нотифікації email та in-app), RBAC/Policy Service (керування ролями, дозволами, політиками) та Audit & Logging (трасування дій користувачів). Міжсервісна взаємодія відбувається асинхронно через RabbitMQ/NATS і синхронно через REST, що дозволяє масштабувати обробку подій без втрати узгодженості даних [1].

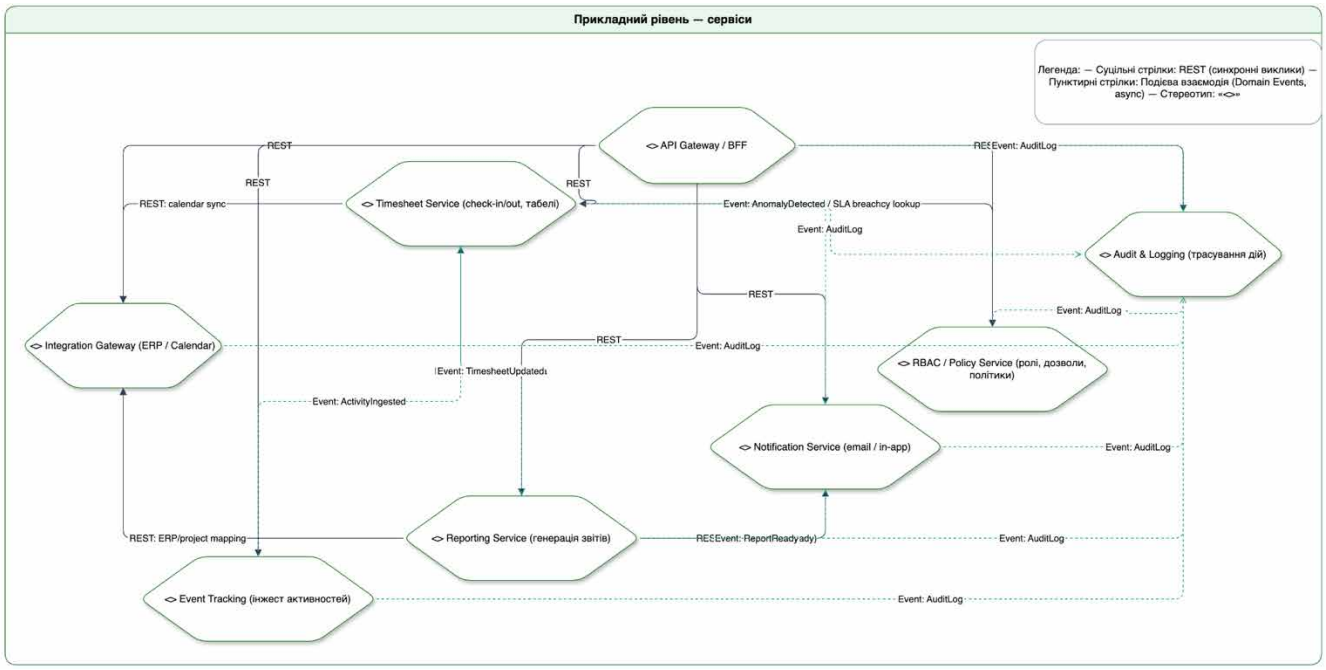

Рисунок 3.2 – Прикладна архітектура сервісного рівня (API Gateway та мікросервіси)

Рівень даних охоплює основну базу PostgreSQL для транзакційних операцій, Redis для кешування, S3/MinIO для зберігання сформованих звітів, а також модуль Secrets/KMS для захисту конфіденційних даних і ELK/Loki для централізованого логування. Інтеграційний шар представлено шлюзом Integration Gateway, який синхронізує події з ERP, Google Calendar та SMTP-серверами. Система підтримує наскрізний моніторинг через Prometheus/Grafana із збором метрик із кожного компонента.

Таблиця 3.2 – Структура архітектурних рівнів системи

Рівень	Основні компоненти	Протоколи взаємодії	Ключові функції
Клієнтський	PyQt6 додаток, SQLite	REST/JSON	Взаємодія з користувачем, автономна робота
Сервісний	API Gateway, Timesheet, Reporting, Notification, RBAC, Audit	REST, Events	Обробка бізнес-логіки, аудит, політики, звітність
Даних	PostgreSQL, Redis, S3/MinIO, KMS	SQL, Cache, S3 API	Зберігання, кешування, шифрування

Продовження таблиці 3.2

Інтеграційний	Integration Gateway, ERP, SMTP, Google Calendar API	REST, Events	Синхронізація із зовнішніми системами
Моніторинг	Prometheus, Grafana, ELK/Loki	metrics/logs	Збір метрик, спостережуваність, аналіз аномалій

Архітектурна модель реалізує концепцію Clean Architecture з розподілом на core (доменні моделі), services (бізнес-логіка) та infrastructure (зовнішні адаптери). Це дозволяє впроваджувати зміни на рівні технологічного стеку без впливу на ядро системи, спрощує модульне тестування та підтримку. Завдяки використанню Docker-контейнеризації кожен сервіс може розгортатися незалежно, а CI/CD-конвеєр забезпечує автоматичне оновлення та перевірку ізольованих компонентів. Таким чином, наведена архітектура поєднує високий рівень модульності, гнучкість розширення та відповідність принципам Domain-Driven Design і DevOps-орієнтованої експлуатації [2].

3.3 Інформаційна база, представлення фізичної моделі даних

Інформаційна база інтелектуальної системи обліку робочих годин побудована відповідно до принципів цілісності, нормалізації (до рівня 3НФ) та безпечного зберігання даних. Структура фізичної моделі забезпечує логічну узгодженість між сутностями, високий рівень референтної цілісності та ефективність виконання запитів при великій кількості транзакцій. Основна схема фізичного рівня подана на рисунку 3.3, де відображено всі таблиці, первинні та зовнішні ключі, індекси та механізми каскадного оновлення.

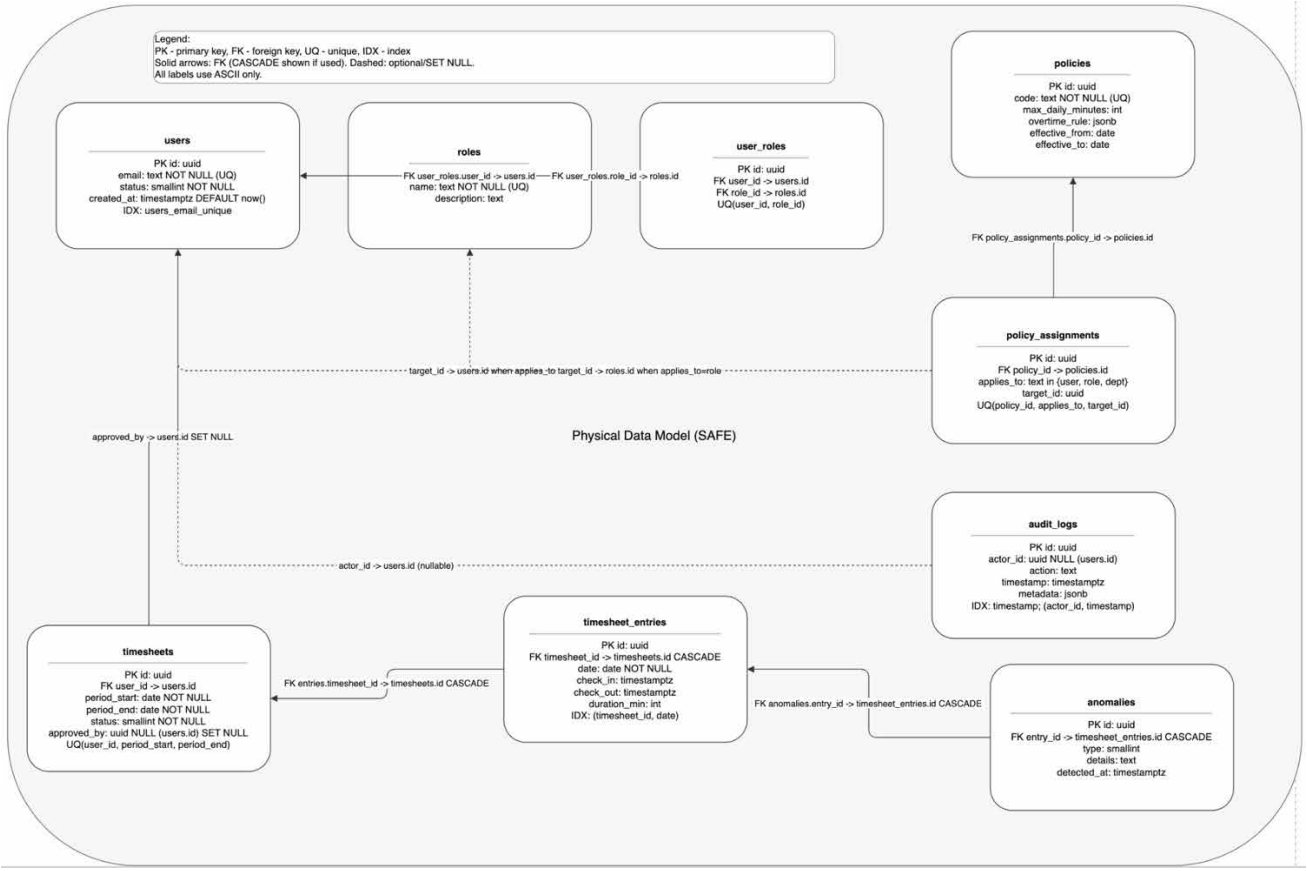

Рисунок 3.3 – Фізична модель даних інтелектуальної системи обліку робочих годин (SAFE)

Модель складається з восьми основних сутностей, що логічно групуються у три домени: керування доступом (users, roles, user_roles, policy_assignments, policies), облік робочого часу (timesheets, timesheet_entries, anomalies) та системні журнали (audit_logs). Згідно з рис. 3.3, для кожної сутності використовується унікальний UUID як первинний ключ, що гарантує розподілену унікальність і незалежність від конкретного сервера бази даних. Всі зовнішні ключі мають параметри ON DELETE CASCADE або SET NULL, що зберігає консистентність під час видалення пов’язаних записів.

На рисунку 3.4 представлено розширену частину моделі, що демонструє механізм зв’язку між сутностями users, roles та user_roles. Вона реалізує класичну many-to-many асоціацію для керування правами доступу, де таблиця user_roles містить комбінації ідентифікаторів користувачів і ролей з унікальним обмеженням (user_id, role_id).

```
INSERT INTO timesheet_entries (id, timesheet_id, date, check_in, type)
VALUES (
    gen_random_uuid(),
    (SELECT id FROM timesheets WHERE user_id = '8b53c0af-22f8-4b1c-bd7b-8e21b1f217a9' AND period = current_date),
    current_date,
    now(),
    'check_in'
)
RETURNING id;
```

Рисунок 3.4 – Логічна зв’язність таблиць керування ролями користувачів

Взаємодія між таблицями timesheets і timesheet_entries (рис. 3.5) реалізує каскадний зв’язок 1:N, що дозволяє зберігати множину записів про події робочого дня в межах одного табеля. Поле approved_by зовнішньо посилається на users.id і встановлюється в NULL у разі видалення керівника, що забезпечує збереження історичної достовірності табелів.

```
SELECT
    u.email AS employee_email,
    t.period,
    SUM(EXTRACT(EPOCH FROM (e.check_out - e.check_in))/3600) AS total_hours,
    COUNT(a.id) AS anomaly_count
FROM users u
JOIN timesheets t ON t.user_id = u.id
JOIN timesheet_entries e ON e.timesheet_id = t.id
LEFT JOIN anomalies a ON a.entry_id = e.id
WHERE t.period BETWEEN '2025-09-01' AND '2025-09-30'
  AND u.department = 'Software Engineering'
GROUP BY u.email, t.period
ORDER BY u.email;
```

Рисунок 3.5 – Фізичний зв’язок між табелями timesheets та timesheet_entries

Механізм фіксації аномалій у роботі системи зображено на рисунку 3.6. Сутність anomalies пов’язана із записами табеля timesheet_entries через зовнішній ключ entry_id з каскадним видаленням. Вона містить тип відхилення, детальний опис і час виявлення, що дозволяє будувати тригерні або аналітичні механізми для прогнозування відхилень.

```sql
SELECT
    e.id AS entry_id,
    e.date,
    e.check_in,
    e.check_out,
    EXTRACT(EPOCH FROM (e.check_out - e.check_in))/60 AS worked_minutes,
    p.max_daily_minutes
FROM timesheet_entries e
JOIN timesheets t ON t.id = e.timesheet_id
JOIN policy_assignments pa ON pa.target_id = t.user_id
JOIN policies p ON p.id = pa.policy_id
WHERE EXTRACT(EPOCH FROM (e.check_out - e.check_in))/60 > p.max_daily_minutes;
```

Рисунок 3.6 – Схема відображення аномалій у зв’язку з подіями табелю

На рисунку 3.7 подано реалізацію зберігання політик обліку часу та їх прив’язки до об’єктів системи через таблицю policy_assignments. Це дозволяє застосовувати політики до користувачів, ролей або підрозділів за гнучкою схемою. Поле applies_to зберігає контекст призначення, а target_id посилається на відповідну сутність, що забезпечує універсальність моделі без необхідності дублювання логіки на рівні коду.

```sql
SELECT
    actor_id,
    action,
    timestamp AT TIME ZONE 'Europe/Kyiv' AS local_time,
    metadata
FROM audit_logs
WHERE actor_id = '8b53c0af-22f8-4b1c-bd7b-8e21b1f217a9'
  AND timestamp BETWEEN now() - INTERVAL '7 days' AND now()
ORDER BY timestamp DESC;
```

Рисунок 3.7 – Механізм призначення політик у базі даних

Окреме місце займає таблиця audit_logs, схема якої показана на рисунку 3.8. Вона зберігає дані про всі критичні дії користувачів, включно з автентифікацією, створенням, модифікацією та переглядом записів. Для забезпечення ефективного пошуку введено композитний індекс (actor_id, timestamp), що оптимізує запити журналів аудиту за ідентифікатором користувача та часовими інтервалами [13].

```sql
SELECT
    date_trunc('day', e.date) AS day,
    COUNT(DISTINCT t.user_id) AS active_users,
    AVG(EXTRACT(EPOCH FROM (e.check_out - e.check_in))/3600) AS avg_work_hours
FROM timesheet_entries e
JOIN timesheets t ON e.timesheet_id = t.id
GROUP BY day
ORDER BY day DESC;
```

Рисунок 3.8 – Структура системного журналу audit_logs для відстеження дій користувачів

Таблиця 3.3 – Основні сутності фізичної моделі даних

№	Таблиця	Ключові поля	Тип зв’язку	Призначення
1	users	id, email, status	1:N → timesheets	Облікові дані користувачів
2	roles	id, name	M:N через user_roles	Ролі доступу
3	user_roles	user_id, role_id	M:N	Зіставлення користувачів і ролей
4	timesheets	id, user_id, period_start	1:N → timesheet_entries	Головна таблиця табелю
5	timesheet_entries	id, timesheet_id, date	1:N → anomalies	Записи подій робочого дня
6	anomalies	entry_id, type, detected_at	N:1 → timesheet_entries	Зберігання виявлених відхилень
7	policies	id, code, max_daily_minutes	1:N → policy_assignments	Опис політик обліку
8	policy_assignments	policy_id, target_id	N:1 → users/roles	Призначення політик
9	audit_logs	actor_id, timestamp	N:1 → users	Логування та аудит операцій

Реалізована фізична модель поєднує нормалізовану структуру, гнучку політику зв’язків і захист даних через суворі обмеження на рівні СУБД. Така організація дає змогу системі ефективно масштабуватись, обробляти паралельні транзакції, зберігати повну історію дій користувачів і забезпечувати відмовостійкість у режимі багатокористувацького доступу [7].

3.4 Алгоритми функціонування програмного забезпечення

Алгоритмічна структура системи визначає логіку обробки подій і даних на рівні прикладного програмного забезпечення, що реалізує сервіси авторизації, обліку робочого часу та формування звітів. Вона спроєктована у відповідності до принципів асинхронної взаємодії мікросервісів, використання токенів доступу (JWT) та кешування результатів для підвищення продуктивності. Кожен алгоритм реалізовано у вигляді незалежного сервісу з чітко визначеними точками інтеграції, які відображені на рисунках 3.9–3.11.

На рис. 3.9 наведено блок-схему алгоритму авторизації користувача через протокол OpenID Connect (OIDC), що забезпечує безпечну автентифікацію в системі за допомогою корпоративного провайдера ідентичності.

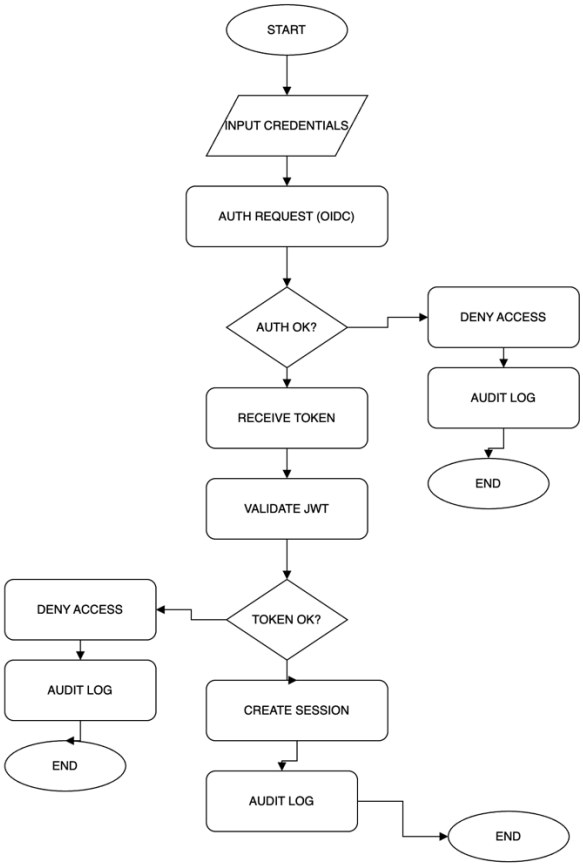

Рисунок 3.9 – Блок-схема алгоритму авторизації користувача через OIDC

Алгоритм реалізує перевірку облікових даних, запит до IdP-сервера, валідацію токена JWT та створення сесії користувача. У разі неуспішної перевірки здійснюється запис у таблицю audit_logs з фіксацією спроби входу. Програмна реалізація подана у

листингу 3.10, де використано бібліотеки FastAPI, Authlib і python-jose для побудови механізму OAuth 2.0 / OIDC-авторизації, а також функцію журналювання подій save_audit_log() [1].

```python
from fastapi import Depends, HTTPException, status
from authlib.integrations.starlette_client import OAuth
from jose import jwt, JWTError
from app.db import save_audit_log

oauth = OAuth()
oidc = oauth.register(
    name="oidc",
    server_metadata_url="https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration",
    client_id="423524352",
    client_secret="#64q237gf2347r6g482468",
    client_kwargs={"scope": "openid profile email"}
)

async def authenticate_user(request):
    token = await oidc.authorize_access_token(request)
    try:
        payload = jwt.decode(token["id_token"], options={"verify_signature": False})
        user_email = payload.get("email")
        if not user_email:
            raise HTTPException(status_code=401, detail="Invalid token payload")
        save_audit_log(user_email, action="LOGIN_SUCCESS")
        return {"email": user_email, "access_token": token["access_token"]}
    except JWTError:
        save_audit_log(None, action="LOGIN_FAILURE")
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid credentials")
```

Рисунок 3.10 – Лістинінг Python-коду реалізації OIDC-авторизації

На рис. 3.11 показано алгоритм реєстрації події Check-in / Check-out, що відповідає за облік початку та завершення робочих змін.

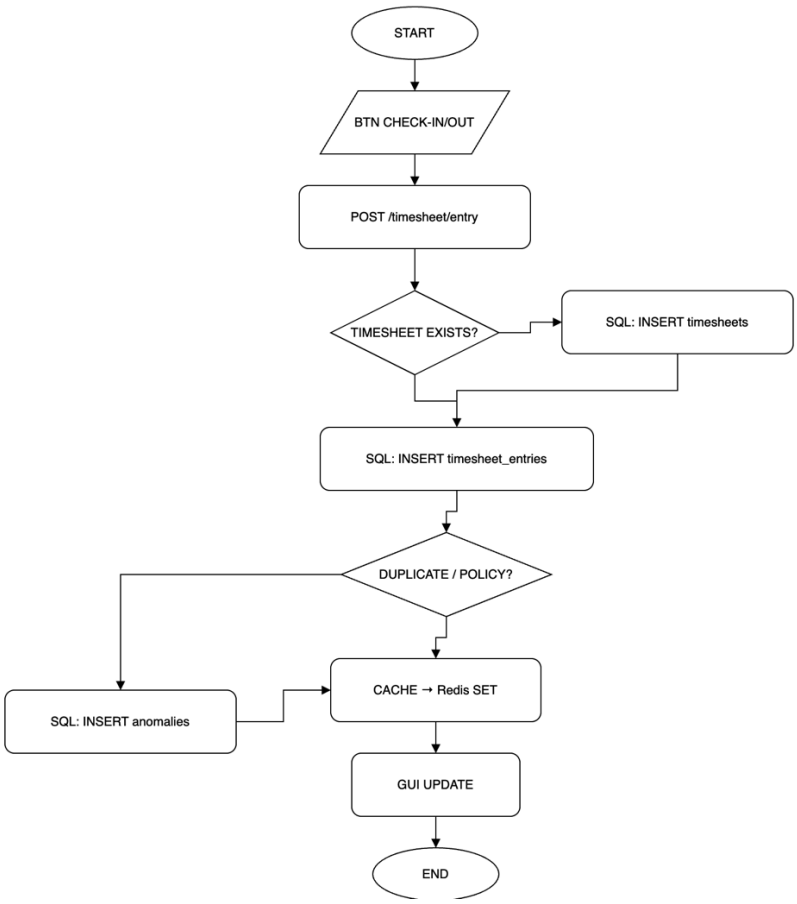

Рисунок 3.11 – Блок-схема алгоритму Check-in / Check-out

Логіка включає перевірку наявності актуального табеля (timesheets), додавання запису у таблицю timesheet_entries, а також виявлення дублювань або порушень політики робочого часу з подальшою реєстрацією аномалії у таблиці anomalies. Після успішного внесення даних кеш оновлюється в Redis, що забезпечує миттєве відображення подій у GUI-інтерфейсі. Відповідна серверна функція наведена у листингу 3.13, який демонструє інтеграцію з PostgreSQL, Redis та модулем audit_log для забезпечення узгодженості транзакцій [2].

```
from datetime import datetime
from app.db import db, redis_cache, insert_anomaly, save_audit_log

def register_event(user_id: str, event_type: str):
    # Перевіряємо наявність активного табеля
    ts = db.fetch_one(
        "SELECT id FROM timesheets WHERE user_id = %s AND period = CURRENT_DATE", (user_id,)
    )
    if not ts:
        ts = db.execute(
            "INSERT INTO timesheets (id, user_id, period) VALUES (gen_random_uuid(), %s, CURRENT_DATE) RETURNING id", (user_id,)
        )

    entry_id = db.execute(
        "INSERT INTO timesheet_entries (id, timesheet_id, date, check_in, type) "
        "VALUES (gen_random_uuid(), %s, CURRENT_DATE, now(), %s) RETURNING id",
        (ts["id"], event_type)
    )

    # Перевірка політики
    duplicates = db.fetch_one(
        "SELECT COUNT(*) FROM timesheet_entries WHERE user_id = %s AND date = CURRENT_DATE", (user_id,)
    )
    if duplicates and duplicates["count"] > 2:
        insert_anomaly(entry_id, reason="DUPLICATE_EVENT")

    redis_cache.set(f"user:{user_id}:last_event", event_type)
    save_audit_log(user_id, action=f"{event_type.upper()}_R  TERED")
    return {"entry_id": entry_id, "status": "ok"}
```

Рисунок 3.13 – Код функції реєстрації події в табелях обліку часу

На рис. 3.14 подано алгоритм формування та відправлення звіту, який реалізує асинхронну агрегацію даних про відпрацьований час і виявлені відхилення.

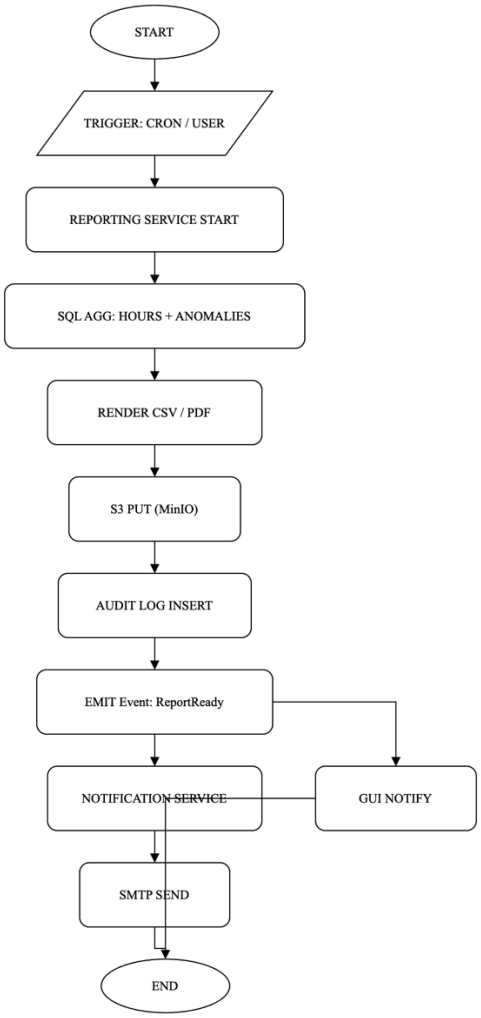

Рисунок 3.14 – Блок-схема алгоритму формування та відправлення звіту

Сервіс запускається за розкладом або користувацьким запитом, виконує SQL-агрегацію з таблиць timesheet_entries та anomalies, формує CSV/PDF-файли і зберігає їх у сховищі S3/MinIO. Далі створюється подія ReportReady, яка ініціює відправлення повідомлення через Notification Service та SMTP-сервер. Код листингу 3.15 реалізує взаємодію з S3-сховищем, механізмом аудиту та електронною поштою, забезпечуючи замкнутий цикл генерації та розповсюдження аналітичної звітності [3].

```python
import csv, io, smtplib
from email.message import EmailMessage
from app.db import db, save_audit_log
from app.s3 import s3_upload
from app.events import publish_event

def generate_report(department: str):
    rows = db.fetch_all("""
        SELECT u.email, SUM(EXTRACT(EPOCH FROM (e.check_out - e.check_in))/3600) AS total_hours
        FROM users u
        JOIN timesheets t ON t.user_id = u.id
        JOIN timesheet_entries e ON e.timesheet_id = t.id
        WHERE u.department = %s AND t.period = CURRENT_DATE
        GROUP BY u.email
    """, (department,))

    buffer = io.StringIO()
    writer = csv.writer(buffer)
    writer.writerow(["Employee", "Total Hours"])
    writer.writerows([(r["email"], r["total_hours"]) for r in rows])

    s3_path = s3_upload(buffer.getvalue().encode(), f"reports/{department}.csv")
    save_audit_log(None, action="REPORT_GENERATED", metadata={"path": s3_path})
    publish_event("ReportReady", {"path": s3_path, "department": department})
    send_email_report(department, s3_path)

def send_email_report(department, s3_path):
    msg = EmailMessage()
    msg["Subject"] = f"Daily Report for {department}"
    msg["From"] = "noreply@system.local"
    msg["To"] = "hr@company.com"
    msg.set_content(f"Report ready: {s3_path}")
    with smtplib.SMTP("smtp.company.com") as s:
        s.send_message(msg)
```

Рисунок 3.15 – Фрагмент Python-коду сервісу генерації звіту

У таблиці 3.4 наведено узагальнення основних алгоритмічних модулів системи та їх призначення, що дозволяє простежити логічну послідовність виконання та інформаційні потоки між модулями.

Таблиця 3.4 – Алгоритмічні модулі та їх функціональне призначення

№	Назва алгоритму	Призначення	Основні сервіси / таблиці
1	Авторизація користувача (OIDC)	Перевірка ідентичності та створення сесії на основі JWT-токенів	users, audit_logs, OIDC Provider
2	Реєстрація Check-in / Check-out	Фіксація подій робочого дня, виявлення аномалій	timesheets, timesheet_entries, anomalies, Redis

Продовження таблиці 3.4

3	Формування звіту	Агрегація даних та генерація аналітичних файлів	timesheet_entries, anomalies, audit_logs, S3 / SMTP

Побудована алгоритмічна архітектура дозволяє забезпечити цілісний життєвий цикл даних у системі - від автентифікації користувача та запису подій до автоматичного створення та доставки аналітичних зведень. Інтеграція між сервісами через єдині інтерфейси REST і подієві механізми забезпечує високу надійність і масштабованість рішення, а реалізація на мові Python з використанням асинхронних парадигм дає можливість досягти оптимальної продуктивності при мінімальному споживанні ресурсів [4].

4 ПРОГРАМНА РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Тестування інтелектуальної системи обліку робочих годин виконується з метою перевірки коректності роботи основних програмних модулів, стабільності взаємодії між клієнтською частиною, API-рівнем, базою даних, кешем, чергами задач і зовнішніми сервісами. Оскільки система реалізує облік робочих подій, авторизацію користувачів, формування табелів, обробку заявок, аналіз аномалій, нотифікації та аудит, тестування має охоплювати як окремі функції, так і повні бізнес-сценарії роботи.

План тестування сформовано відповідно до архітектури системи, що передбачає використання PyQt6 для графічного інтерфейсу, FastAPI для серверної частини, PostgreSQL як основної бази даних, Redis для кешування, Celery для асинхронної обробки подій, а також JWT/OIDC і RBAC для автентифікації та розмежування прав доступу. Основна увага приділяється перевірці достовірності фіксації робочого часу, правильності побудови табелів, коректності застосування трудових політик і надійності збереження журналів дій користувачів.

Тестування проводиться за такими напрямами: модульне, інтеграційне, функціональне, інтерфейсне, безпекове, продуктивнісне та регресійне. Модульне тестування використовується для перевірки окремих компонентів без залежності від інших частин системи. Інтеграційне тестування дає змогу оцінити правильність обміну даними між GUI, API, базою даних, Redis і Celery. Функціональне тестування перевіряє відповідність системи вимогам, сформульованим у першому розділі. Безпекове тестування спрямоване на перевірку авторизації, ролей доступу, захисту токенів і журналювання критичних дій.

Таблиця 4.1 – План тестування програмних модулів системи

№	Модуль системи	Мета тестування	Основні перевірки	Очікуваний результат
1	Модуль авторизації	Перевірити вхід користувача та обробку токенів	Вхід за коректними/некоректними даними, перевірка JWT, завершення сесії	Користувач отримує доступ лише після успішної автентифікації
2	Модуль RBAC	Перевірити розмежування прав доступу	Доступ працівника, менеджера, HR і адміністратора до різних функцій	Кожна роль бачить лише дозволені функції
3	Модуль check-in/check-out	Перевірити фіксацію початку й завершення роботи	Створення події початку зміни, завершення зміни, повторний check-in	Події зберігаються коректно, дублікати блокуються
4	Модуль табелів	Перевірити формування обліку робочого часу	Розрахунок тривалості роботи, перерв, запізнень, відхилень	Табель формується відповідно до політик підприємства
5	Модуль заявок	Перевірити обробку заявок на відпустку, відгул і понаднормові	Створення, редагування, погодження, відхилення заявки	Статус заявки змінюється коректно
6	Модуль політик	Перевірити застосування правил робочого часу	Перевірка тривалості зміни, допустимих відхилень, понаднормових	Система правильно визначає порушення політик
7	Модуль аномалій	Перевірити виявлення нестандартних ситуацій	Пропуск check-out, дублювання подій, перевищення норми часу	Аномалія створюється та відображається в системі
8	Модуль звітності	Перевірити генерацію звітів і табелів	Формування звіту за день, тиждень, місяць, експорт даних	Звіт містить повні та правильні дані
9	Модуль нотифікацій	Перевірити надсилання повідомлень	Повідомлення про заявку, аномалію, завершення звіту	Користувач отримує відповідне сповіщення
10	Модуль аудиту	Перевірити журналювання дій	Запис входу, зміни політик, погодження заявок, помилок	У журналі зберігаються всі важливі дії

Методика оцінювання результатів передбачає порівняння фактичної поведінки системи з очікуваною. Для кожного тестового сценарію визначаються вхідні дані, послідовність дій, очікуваний результат, фактичний результат і статус виконання. Сценарій вважається успішним, якщо система виконує дію без критичних помилок, зберігає дані у правильному форматі, не порушує обмежень доступу та забезпечує коректне відображення результатів у клієнтському інтерфейсі.

Таблиця 4.2 – Критерії оцінювання результатів тестування

Критерій	Метод перевірки	Допустиме значення
Коректність функцій	Порівняння фактичного й очікуваного результату	100 % критичних сценаріїв мають виконуватися без помилок
Час відповіді API	Вимірювання часу виконання запитів	До 500 мс для типових операцій
Цілісність даних	Перевірка записів у PostgreSQL	Відсутність дублювання та порушення зовнішніх ключів
Стабільність GUI	Перевірка роботи форм PyQt6	Відсутність зависань і аварійного завершення
Безпека доступу	Перевірка JWT, RBAC і ролей	Користувач не має доступу до заборонених функцій
Обробка помилок	Введення некоректних даних	Система показує зрозуміле повідомлення та не втрачає дані
Журналювання	Перевірка audit_logs	Критичні дії фіксуються в журналі
Продуктивність	Навантажувальне тестування	Стабільна робота при одночасній роботі багатьох користувачів

Окремо перевіряється робота системи в граничних ситуаціях: повторна спроба check-in протягом активної зміни, відсутність завершення робочого дня, спроба доступу до адміністративних функцій без відповідної ролі, створення заявки з некоректним періодом, втрата з’єднання з API або тимчасова недоступність кешу Redis. У таких випадках система повинна не завершувати роботу аварійно, а формувати повідомлення про помилку, зберігати поточний стан і записувати подію в журнал аудиту.

Для перевірки серверної частини доцільно використовувати автоматизовані тести FastAPI з тестовою базою даних, які охоплюють маршрути авторизації, обліку часу, заявок, політик, звітів і нотифікацій. Для клієнтської частини PyQt6 виконується ручне та сценарне тестування основних форм: вікна входу, панелі працівника, панелі

менеджера, HR-модуля, адміністративного блоку, журналу подій і сторінки звітності. Для перевірки асинхронних операцій тестується робота Celery-задач: генерація звітів, створення сповіщень, обробка аномалій і запис службових подій.

Результати тестування фіксуються у вигляді тестових протоколів, де зазначаються назва сценарію, модуль, вхідні дані, очікуваний результат, фактичний результат і статус виконання. Такий підхід дозволяє не лише підтвердити працездатність системи, а й виявити слабкі місця реалізації, які потребують доопрацювання перед розгортанням у реальному середовищі підприємства.

Отже, запропонований план тестування охоплює всі ключові програмні модулі інтелектуальної системи обліку робочих годин і забезпечує комплексну перевірку її функціональності, продуктивності, безпеки та стійкості до помилок. Методика оцінювання результатів дає змогу об’єктивно визначити відповідність реалізованої системи вимогам, сформульованим на етапі системного аналізу та проєктування.

4.2 Тестування функціональних сценаріїв та інтерфейсних модулів системи

Тестування системи виконувалося для перевірки працездатності основних інтерфейсних модулів інтелектуальної системи обліку робочих годин. Основна увага приділялась не лише відкриттю вікон застосунку, а й перевірці повних користувацьких сценаріїв: входу до системи, перегляду командної аналітики, фіксації робочого дня, створення задач, подання заявок, аналізу аномалій, формування звітів, керування користувачами, політиками та аудитом.

Першим етапом перевірено модуль авторизації. Вікно входу містить поля введення email і пароля, демонстраційні облікові записи, кнопку входу та блок створення нового користувача. Перевірка показала, що система коректно приймає дані адміністратора та звичайного користувача, після чого відкриває робоче середовище відповідно до ролі. Також перевірено введення некоректного пароля: у такому випадку доступ не надається, а користувач залишається на формі входу. Інтерфейс авторизації показано на рис. 4.1.

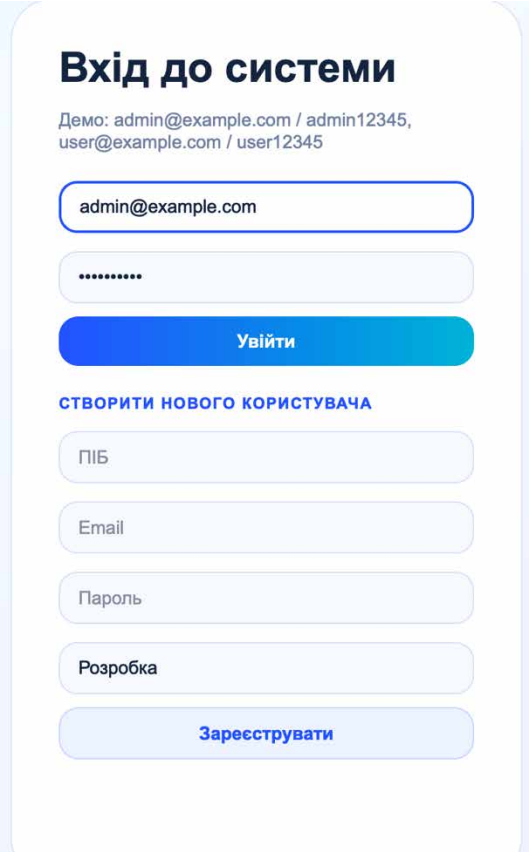

Рисунок 4.1 – Вікно входу до інтелектуальної системи обліку робочих годин

Після успішної авторизації було протестовано головну панель керування. У режимі адміністратора система відображає командний пульс, поточну дату, вибір ролі перегляду, показники робочого часу, фокусу, перерв і кількості аномалій. Також відображається індекс стабільності дня, графік робочих хвилин за останні сім днів і таблиця стану команди. Перевірка підтвердила, що дані на картках, графіках і в таблиці узгоджені між собою та оновлюються відповідно до вибраного користувача або ролі перегляду. Загальний вигляд командного пульсу наведено на рис. 4.2.

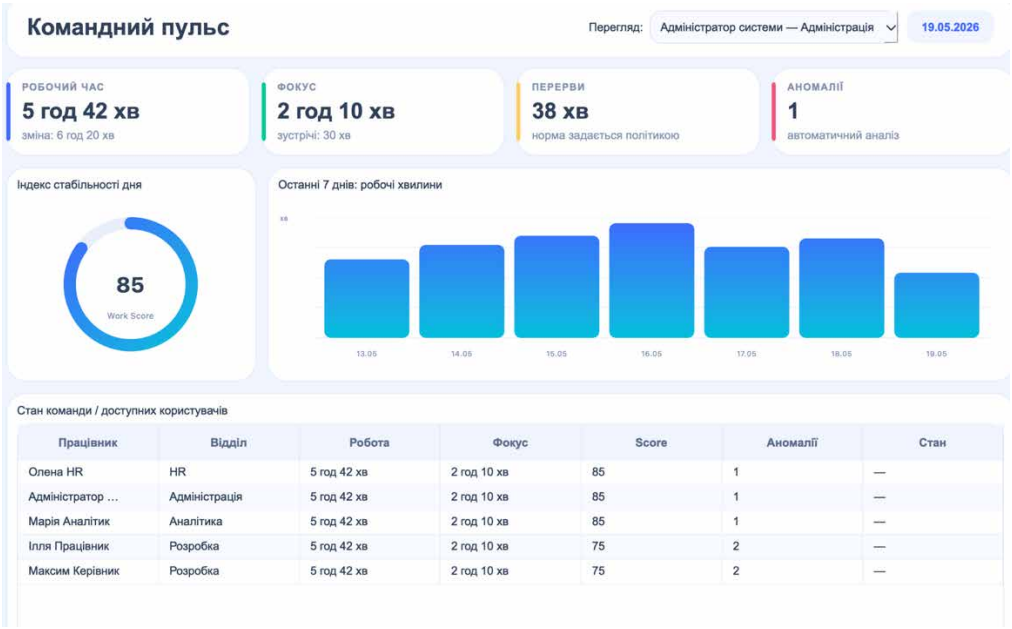

Рисунок 4.2 – Головна панель моніторингу робочого часу та стану команди

Окремо протестовано модуль фіксації робочого дня, який забезпечує виконання основних операцій обліку часу: початок робочого дня, завершення робочого дня, старт і завершення перерви, старт і завершення фокусного режиму. У центральній частині модуля відображається жива стрічка робочого дня з часовими відрізками SHIFT, FOCUS, MEETING, BREAK і CALL. Це дає змогу візуально перевірити, як система інтерпретує різні типи активності протягом доби. Модуль також містить блок швидкого додавання активності та план дня, що дозволяє фіксувати короткі службові події без переходу до інших вікон. Результат тестування модуля наведено на рис. 4.3.

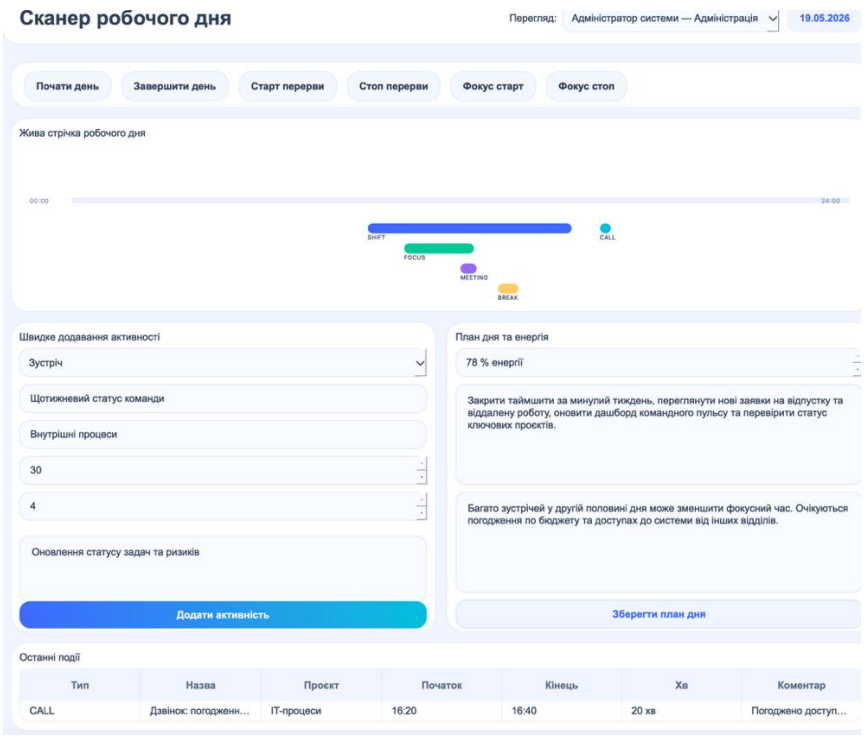

Рисунок 4.3 – Модуль сканера робочого дня та фіксації активностей користувача

Наступним етапом було перевірено модуль проєктів і задач. У лівій частині інтерфейсу реалізовано форму створення нової задачі із зазначенням назви, проєкту, пріоритету, планової оцінки та дедлайну. У правій частині розміщено канбан-реєстр задач із полями ID, назва, проєкт, пріоритет, статус, оцінка, витрачений час і дедлайн. Під час тестування було перевірено створення задачі, зміну статусу, внесення фактично витраченого часу та оновлення вибраного запису. Система коректно відображає задачі з різними статусами: todo, in_progress, review, done, що підтверджує працездатність модуля керування роботою. Вигляд цього модуля подано на рис. 4.4.

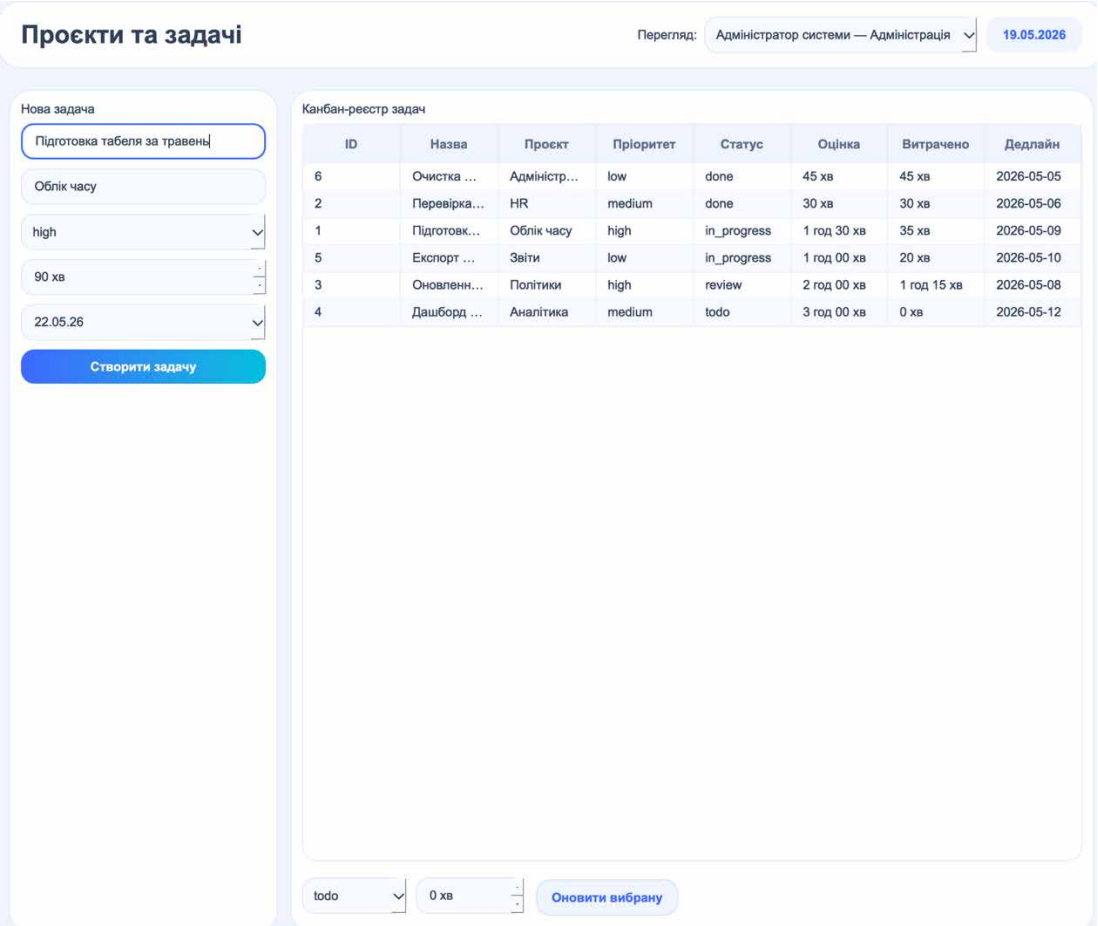

Рисунок 4.4 – Модуль керування проєктами та задачами

Модуль заявок і погоджень перевірявся на сценаріях подання заявки на відпустку, понаднормову роботу, дистанційний режим і корекцію робочого часу. Інтерфейс містить поля вибору типу заявки, дат початку й завершення, тривалості в хвилинах і причини. Журнал заявок відображає працівника, тип, період, кількість хвилин, статус, причину та рішення. Додатково реалізовано кнопки погодження й відхилення заявки. Під час перевірки підтверджено, що нова заявка додається до журналу зі статусом pending, а після обробки керівником змінює стан на approved або rejected. Модуль заявок показано на рис. 4.5.

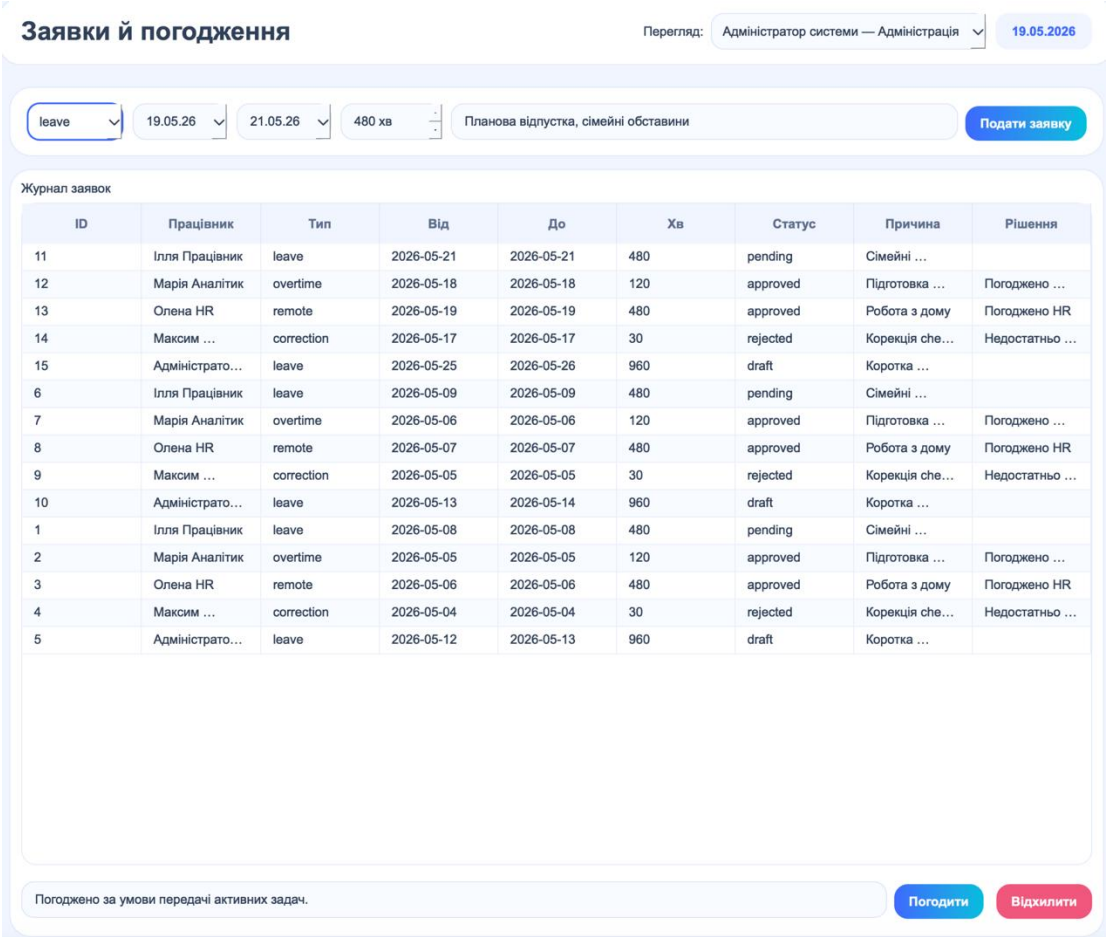

Рисунок 4.5 – Модуль подання, перегляду та погодження заявок

Важливою частиною системи є модуль аномалій та рекомендацій. Під час тестування перевірено виявлення недостатньої тривалості робочого дня, коли фактично нараховано 342 хвилини з нормативних 480 хвилин. Система сформувала попередження рівня `WARNING` і відобразила його в блоці аномалій. Окремо перевірено рекомендаційний блок TimeFlow, який формує короткий висновок щодо поточного режиму роботи та показує Work Score. У тестовому сценарії система визначила поточний Work Score на рівні 85/100 і побудувала графік його динаміки. Результат перевірки наведено на рис. 4.6.

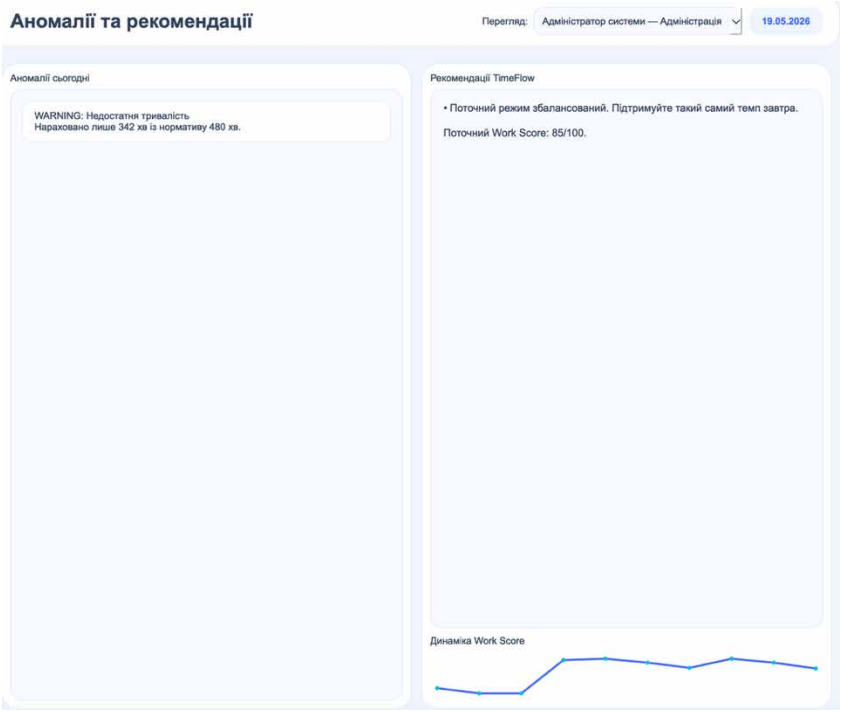

Рисунок 4.6 – Вікно аналізу аномалій, рекомендацій і динаміки Work Score

Модуль аналітичних звітів тестувався шляхом вибору періоду з 12.05.2026 до 19.05.2026, побудови графіка та перевірки табличних даних. Система відображає робочий час, фокусний час, перерви, зустрічі, індекс Score та кількість аномалій за кожен день. Також перевірено наявність кнопок експорту у формати CSV і HTML. Результати показали, що модуль коректно формує аналітичну вибірку за період, відображає її у вигляді графіка та таблиці, а також готує дані для подальшого експорту. Інтерфейс звітності наведено на рис. 4.7.

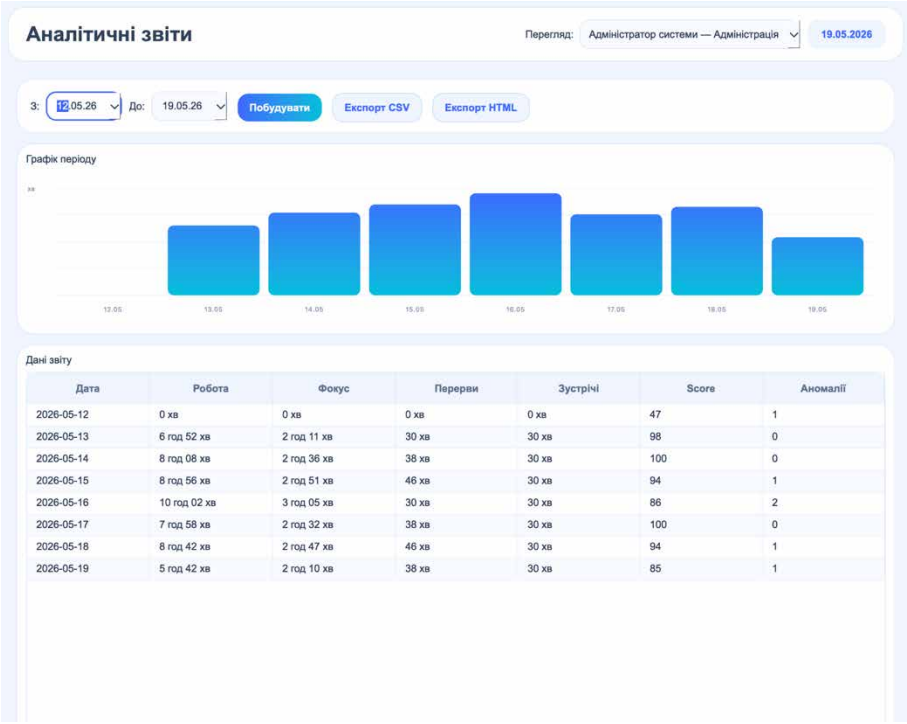

Рисунок 4.7 – Модуль аналітичних звітів за вибраний період

Завершальним етапом перевірено адміністративний модуль користувачів, політик і аудиту. У верхній частині відображено перелік користувачів із полями ID, ПІБ, email, роль, відділ, посада та статус. Нижче розташовано елементи оновлення ролі, статусу, відділу й посади користувача. Окремий блок містить політики робочого часу для різних відділів: HR, адміністрація, аналітика. Для кожної політики зазначено старт, завершення, норму робочого дня, тривалість перерви, фокусного часу та поріг overtime. У нижній частині розміщено журнал аудиту, де фіксуються вхід користувача, системні події та службові зміни. Перевірка підтвердила, що адміністратор має доступ до службових даних, а критичні дії записуються в audit-журнал. Вікно адміністрування показано на рис. 4.8.

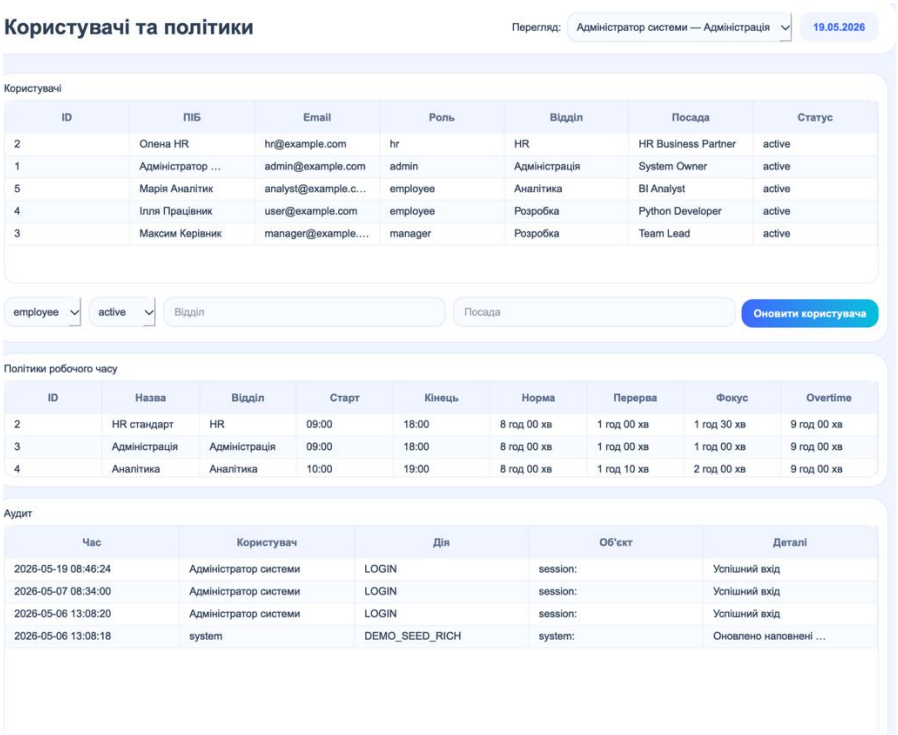

Рисунок 4.8 – Адміністративний модуль користувачів, політик і аудиту

Для узагальнення результатів тестування основних функціональних сценаріїв сформовано таблицю 4.3.

Таблиця 4.3 – Результати тестування основних функціональних сценаріїв системи

№	Сценарій тестування	Вхідні дані / дія	Очікуваний результат	Фактичний результат	Статус
1	Вхід адміністратора	admin@example.com, пароль адміністратора	Відкриття системи з правами адміністратора	Відкрито командний пульс адміністратора	Виконано
2	Вхід звичайного користувача	user@example.com, пароль користувача	Відкриття системи з обмеженим доступом	Користувач отримує доступ до власних функцій	Виконано
3	Перегляд головної панелі	Вибір ролі перегляду	Виведення карток, графіка та таблиці команди	Дані відображено коректно	Виконано
4	Початок робочого дня	Натискання «Почати день»	Створення події SHIFT	Подія відображається у стрічці дня	Виконано
5	Перерва	«Старт перерви» / «Стоп перерви»	Фіксація перерви в таймлайні	Перерва відображається як BREAK	Виконано

6	Фокусний режим	«Фокус старт» / «Фокус стоп»	Облік фокусного часу	Сформовано інтервал FOCUS	Виконано

Під час тестування також оцінювалася зручність інтерфейсу. Вікна мають єдиний візуальний стиль: світлий фон, заокруглені блоки, чіткі таблиці, акцентні кнопки та зрозумілу ієрархію заголовків. Це дозволяє швидко переходити між основними модулями без додаткового навчання користувача. Табличні представлення зручні для HR-персоналу й адміністратора, а графіки та індикатори дають змогу керівнику швидко оцінити стан робочого дня або команди.

За результатами тестування встановлено, що система виконує основні функціональні вимоги: забезпечує авторизацію, фіксацію робочих подій, ведення задач, подання та погодження заявок, аналіз аномалій, побудову звітів, керування користувачами, політиками й аудитом. Виявлені недоліки мають некритичний характер і стосуються переважно відображення довгих текстових значень у таблицях, які скорочуються для збереження компактності інтерфейсу. Це не впливає на логіку роботи системи та може бути усунено шляхом додавання підказок або розширення окремих колонок.

Отже, проведене тестування підтвердило працездатність програмної реалізації інтелектуальної системи обліку робочих годин. Реалізовані модулі відповідають функціональним сценаріям, визначеним на етапі системного аналізу, а інтерфейс забезпечує зручну роботу для працівника, керівника, HR-спеціаліста та адміністратора.

4.3 Розгортання системи та склад інсталяційного пакету

Розгортання інтелектуальної системи обліку робочих годин передбачає встановлення клієнтської частини на робочому місці користувача та запуск серверних компонентів у локальній або корпоративній інфраструктурі підприємства. Система побудована за клієнт-серверним принципом: настільний застосунок PyQt6 відповідає за взаємодію з користувачем, серверна частина FastAPI обробляє запити, база даних

PostgreSQL зберігає основну інформацію, Redis використовується для кешування та черг, а Celery Worker виконує фонові задачі.

Загальну структуру розгортання системи подано на рисунку 4.9.

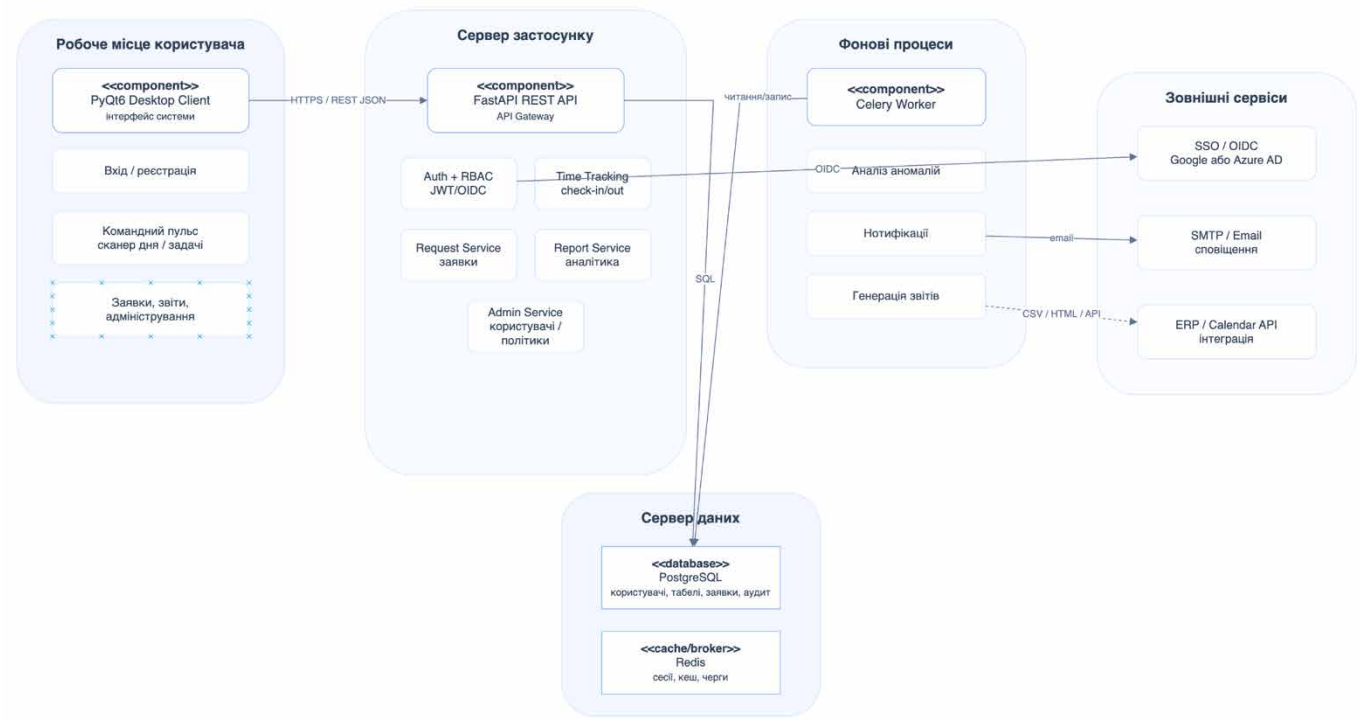

Рисунок 4.9 – Діаграма компонентного розгортання інтелектуальної системи обліку робочих годин

На діаграмі показано п’ять основних зон розгортання: робоче місце користувача, сервер застосунку, блок фонових процесів, сервер даних і зовнішні сервіси. Робоче місце користувача містить PyQt6 Desktop Client, через який виконуються вхід до системи, перегляд командного пульсу, фіксація робочого дня, робота із заявками, звітами та адміністративними функціями. Передавання даних між клієнтом і сервером здійснюється через HTTPS у форматі REST/JSON.

Сервер застосунку виконує роль центрального вузла обробки запитів. У його складі функціонують FastAPI REST API та внутрішні сервіси: Auth + RBAC, Time Tracking, Request Service, Report Service і Admin Service. Модуль Auth + RBAC відповідає за автентифікацію користувачів, перевірку JWT/OIDC-токенів і розмежування прав доступу. Модуль Time Tracking обробляє події check-in/check-out, початок і завершення перерв, фокусний режим та інші робочі активності. Request Service забезпечує створення, перегляд і погодження заявок, Report Service формує

аналітичні звіти, а Admin Service використовується для керування користувачами, ролями, політиками та аудитом.

Окремо винесено блок фонових процесів, який містить Celery Worker. Він використовується для задач, що не повинні блокувати основний інтерфейс користувача: аналіз аномалій, надсилання нотифікацій, генерація звітів, підготовка CSV/HTML-файлів та синхронізація із зовнішніми сервісами. Такий підхід підвищує швидкодію системи, оскільки основний API не витрачає час на тривалі операції, а лише ставить задачу в чергу.

Сервер даних містить два ключові компоненти: PostgreSQL і Redis. PostgreSQL використовується як основне реляційне сховище, у якому зберігаються користувачі, профілі, ролі, табелі, події робочого часу, заявки, політики, аномалії, сповіщення та записи аудиту. Redis виконує функції кешу та брокера повідомлень для Celery, зберігає тимчасові сесійні дані, службові стани й черги фонових задач.

Зовнішні сервіси використовуються для розширення функціональності системи. SSO/OIDC забезпечує інтеграцію з корпоративними обліковими записами, наприклад Google або Azure AD. SMTP/Email відповідає за надсилання повідомлень про заявки, аномалії та сформовані звіти. ERP/Calendar API використовується для експорту даних, синхронізації робочих подій і передавання звітності до корпоративних систем.

Таблиця 4.4 – Основні компоненти розгортання системи

Компонент	Технологія	Призначення
Desktop Client	PyQt6	Графічний інтерфейс користувача
REST API	FastAPI	Обробка запитів клієнта та бізнес-логіки
Auth + RBAC	JWT/OIDC	Авторизація та розмежування доступу
Time Tracking	Python/FastAPI	Фіксація робочого часу та активностей
Request Service	Python/FastAPI	Обробка заявок і погоджень
Report Service	Python/FastAPI	Формування звітів та аналітики
Admin Service	Python/FastAPI	Керування користувачами, ролями й політиками
Celery Worker	Celery	Виконання фонових задач
Cache/Broker	Redis	Кешування, сесії та черги задач
Database	PostgreSQL	Основне сховище даних
External Services	SSO, SMTP, ERP API	Інтеграція з корпоративними сервісами

Інсталяційний пакет системи має містити всі необхідні файли для запуску клієнтської та серверної частини. Для зручності розгортання передбачено окремі директорії для клієнта, API, бази даних, конфігурацій, документації та тестових даних. Така структура дозволяє швидко встановити систему на робочій станції або сервері без ручного пошуку залежностей.

Таблиця 4.5 – Склад інсталяційного пакету системи

Елемент пакету	Призначення
client/	Файли настільного PyQt6-застосунку
server/	Код FastAPI-сервера та бізнес-логіки
workers/	Фонові задачі Celery
database/	SQL-скрипти створення таблиць, індексів і початкових даних
config/	Конфігураційні файли системи
requirements.txt	Перелік Python-залежностей
docker-compose.yml	Опис контейнерів PostgreSQL, Redis, API та Celery
.env.example	Шаблон змінних середовища
README.md	Інструкція встановлення та запуску
tests/	Набір модульних та інтеграційних тестів
docs/	Технічна документація, опис API та структури БД
exports/	Каталог для сформованих CSV/HTML-звітів
logs/	Журнали роботи системи та службові події

Перед запуском системи необхідно налаштувати змінні середовища: параметри підключення до PostgreSQL, адресу Redis, секретний ключ JWT, параметри SMTP, URL SSO/OIDC-провайдера та режим запуску системи. Для демонстраційного або локального використання допускається запуск із тестовими обліковими записами адміністратора і користувача. Для промислового середовища необхідно змінити секретні ключі, увімкнути HTTPS, обмежити доступ до адміністративних маршрутів і налаштувати резервне копіювання бази даних.

Послідовність розгортання системи передбачає такі етапи: встановлення Python і залежностей, підготовка .env-файлу, запуск PostgreSQL та Redis, застосування SQL-міграцій, запуск FastAPI-сервера, запуск Celery Worker, перевірка доступності API та запуск клієнтського PyQt6-застосунку. У разі використання Docker більшість цих дій автоматизується через docker-compose.yml, що спрощує перенесення системи між різними комп’ютерами або серверами.

Після розгортання виконується первинна перевірка працездатності: відкриття вікна входу, авторизація адміністратора, перегляд командного пульсу, створення тестової робочої події, подання заявки, формування звіту та перевірка появи запису в журналі аудиту. Якщо всі ці операції виконуються без помилок, система вважається готовою до експлуатації.

Отже, запропонована схема розгортання забезпечує чіткий поділ відповідальності між клієнтським інтерфейсом, серверною логікою, фоновими процесами, сховищем даних і зовнішніми сервісами. Інсталяційний пакет має достатній склад для локального запуску, тестування та подальшого впровадження системи в інфраструктуру підприємства. Такий підхід спрощує супровід програмного забезпечення, підвищує надійність роботи та дозволяє поступово масштабувати систему відповідно до кількості користувачів і обсягу робочих даних.

4.4 Висновки до четвертого розділу

У четвертому розділі було виконано опис програмної реалізації та тестування інтелектуальної системи обліку робочих годин. Основну увагу приділено перевірці працездатності реалізованих модулів, відповідності функціоналу встановленим вимогам, стабільності інтерфейсу, коректності обробки робочих подій, заявок, задач, аналітичних звітів, аномалій та адміністративних операцій.

Було сформовано план тестування програмних модулів, який охоплює перевірку авторизації, розмежування доступу за ролями, фіксацію check-in/check-out, облік перерв і фокусного часу, формування табелів, створення та погодження заявок, генерацію звітів, роботу модуля аномалій, нотифікацій, аудиту та адміністративного керування. Для оцінювання результатів визначено критерії коректності, стабільності, швидкодії, безпеки, цілісності даних і зручності використання інтерфейсу.

Під час тестування інтерфейсних модулів підтверджено, що система коректно виконує основні користувацькі сценарії. Модуль входу забезпечує авторизацію користувачів і доступ до системи відповідно до ролі. Головна панель відображає ключові показники робочого дня, командний стан, індекс стабільності та графіки

активності. Модуль сканера робочого дня дозволяє фіксувати початок і завершення зміни, перерви, зустрічі, дзвінки та фокусний режим. Модуль проєктів і задач забезпечує створення, перегляд та оновлення задач, а модуль заявок підтримує подання, погодження й відхилення запитів.

Окремо перевірено роботу аналітичних функцій системи. Модуль аномалій виявляє відхилення від установлених політик робочого часу, зокрема недостатню тривалість робочого дня, перевищення норм або проблемні записи. Модуль рекомендацій формує короткі підказки для покращення режиму роботи, а аналітичні звіти дозволяють переглядати динаміку робочого часу, фокусного часу, перерв, зустрічей, Work Score та кількості аномалій за вибраний період. Це підтверджує придатність системи не лише для обліку часу, а й для управлінського аналізу продуктивності.

Було розглянуто схему компонентного розгортання системи, яка передбачає поділ на робоче місце користувача, сервер застосунку, фонові процеси, сервер даних і зовнішні сервіси. Така структура забезпечує зрозуміле розмежування відповідальності між PyQt6-клієнтом, FastAPI API, PostgreSQL, Redis, Celery Worker та інтеграційними сервісами SSO/OIDC, SMTP і ERP/Calendar API. Запропонований склад інсталяційного пакету містить клієнтську частину, серверний код, фонові задачі, SQL-скрипти, конфігураційні файли, документацію, тести, журнали та засоби контейнеризованого запуску.

Результати тестування показали, що реалізована система відповідає основним функціональним вимогам, визначеним у попередніх розділах. Вона забезпечує автоматизований облік робочого часу, підтримку ролей користувачів, обробку заявок, ведення задач, аналіз аномалій, формування звітів, адміністрування користувачів і політик, а також фіксацію службових дій у журналі аудиту. Виявлені недоліки не є критичними й переважно стосуються візуального відображення довгих значень у таблицях, що може бути доопрацьовано шляхом розширення колонок або додавання контекстних підказок.

Отже, у четвертому розділі підтверджено працездатність програмної реалізації інтелектуальної системи обліку робочих годин. Проведене тестування довело, що

система може використовуватися для автоматизації контролю робочого часу, підтримки трудових політик, аналізу активності працівників і підготовки звітності. Запропонована архітектура розгортання та склад інсталяційного пакету створюють основу для подальшого впровадження, супроводу й масштабування системи в умовах підприємства.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено інтелектуальну систему обліку робочих годин, призначену для автоматизації фіксації робочого часу, контролю трудових політик, обробки заявок, виявлення аномалій, формування аналітичних звітів і підтримки управлінських рішень у межах підприємства. Актуальність роботи зумовлена поширенням гібридного, дистанційного та змінного форматів зайнятості, за яких традиційні ручні або напівавтоматизовані засоби табелювання не забезпечують достатньої точності, прозорості й оперативності обліку.

У першому розділі було виконано системний аналіз предметної області. Розглянуто основні процеси обліку робочого часу: реєстрацію початку та завершення робочого дня, фіксацію перерв і фокусного часу, подання заявок, погодження відпусток і понаднормових, формування табелів, виявлення порушень та підготовку звітності. Проведений огляд існуючих рішень, зокрема Replicon, Clockify, QuickBooks Time і Apploye, показав, що більшість таких систем орієнтовані переважно на SaaS-модель, мають обмежений локальний режим роботи, недостатню гнучкість налаштування політик і не завжди підтримують розширений аналіз аномалій. Це підтвердило доцільність розробки власної системи з настільним клієнтом, API-рівнем, централізованою базою даних і підтримкою аналітичних функцій.

У межах моделювання предметної області було побудовано DFD-діаграму, UML-діаграму прецедентів, діаграму послідовності та діаграму активності. Ці моделі дали змогу формалізувати основні сценарії взаємодії користувачів із системою, визначити ролі працівника, керівника, HR-спеціаліста та адміністратора, а також описати логіку авторизації, обліку часу, обробки заявок і формування звітності. На основі проведеного аналізу сформовано функціональні, нефункціональні та технічні вимоги до системи, включаючи вимоги до продуктивності, безпеки, масштабованості, доступності, журналювання та інтеграції з корпоративними сервісами.

У другому розділі було спроєктовано інформаційне та програмне забезпечення системи. Розроблено логічну модель даних у вигляді ER-діаграми, нормалізовану до

третьої нормальної форми. Модель охоплює сутності користувачів, профілів, ролей, дозволів, табелів, записів робочого часу, заявок, політик, аномалій, сповіщень і журналу аудиту. Запропонована структура бази даних забезпечує цілісність даних, мінімізує дублювання та створює основу для подальшого масштабування системи.

Також було побудовано діаграму класів, кооперації, компонентну структуру та модель розгортання. У результаті визначено основні програмні модулі системи: модуль авторизації та RBAC, модуль обліку робочого часу, модуль заявок, модуль звітності, модуль аномалій, адміністративний модуль, модуль сповіщень і модуль аудиту. Такий поділ забезпечує зрозумілу структуру програмного забезпечення, спрощує супровід коду та дозволяє розширювати систему без повної перебудови її архітектури.

У третьому розділі було обґрунтовано вибір технологічного стеку та описано програмну реалізацію системи. Як основну мову програмування обрано Python, що забезпечує швидку розробку бізнес-логіки, роботу з базами даних, API, тестами та аналітичними модулями. Для створення графічного інтерфейсу використано PyQt6, що дозволило реалізувати повноцінний настільний застосунок із вікнами авторизації, командним пульсом, сканером робочого дня, модулями задач, заявок, звітів, аномалій та адміністрування. Для серверної частини обрано FastAPI, який забезпечує побудову REST API, автоматичну OpenAPI-документацію та зручну інтеграцію з іншими сервісами.

Для зберігання даних використано PostgreSQL, для локального режиму передбачено SQLite, для кешування та черг - Redis, а для фонової обробки задач - Celery. Безпекова частина системи базується на JWT/OIDC та RBAC, що дозволяє реалізувати захищену автентифікацію й розмежування доступу за ролями. Такий стек є технічно обґрунтованим, оскільки поєднує кросплатформеність, надійність, масштабованість і можливість подальшої інтеграції з ERP, календарями, SMTP-серверами та корпоративними системами єдиного входу.

У четвертому розділі було виконано тестування системи та описано її розгортання. Сформовано план тестування програмних модулів, який охоплює авторизацію, контроль доступу, check-in/check-out, облік перерв, фокусного часу,

створення задач, подання та погодження заявок, аналіз аномалій, формування звітів, адміністрування користувачів, політик і аудит. За результатами перевірки встановлено, що реалізована система коректно виконує основні функціональні сценарії, відображає дані в інтерфейсі, зберігає службові події та забезпечує зручну роботу для різних категорій користувачів.

Було протестовано ключові інтерфейсні модулі: вікно входу, командний пульс, сканер робочого дня, модуль проєктів і задач, модуль заявок, модуль аномалій і рекомендацій, аналітичні звіти та адміністративну панель. Перевірка підтвердила, що система дозволяє реєструвати робочі події, аналізувати активність, створювати й оновлювати задачі, подавати заявки, виявляти відхилення від політик, формувати звіти та переглядати журнал аудиту. Виявлені недоліки мають некритичний характер і можуть бути усунені під час подальшого вдосконалення інтерфейсу.

Розроблена схема компонентного розгортання передбачає поділ системи на робоче місце користувача, сервер застосунку, фонові процеси, сервер даних і зовнішні сервіси. Така структура забезпечує логічне розмежування відповідальності між PyQt6-клієнтом, FastAPI-сервером, PostgreSQL, Redis, Celery Worker та інтеграційними сервісами. Запропонований склад інсталяційного пакету містить клієнтську частину, серверний код, SQL-скрипти, конфігураційні файли, документацію, тести, журнали та засоби контейнеризованого запуску, що створює основу для подальшого впровадження системи.

Отже, мету кваліфікаційної роботи досягнуто. У результаті виконання роботи спроєктовано та реалізовано інтелектуальну систему обліку робочих годин, яка забезпечує автоматизацію контролю робочого часу, підтримку трудових політик, обробку заявок, аналіз аномалій, формування звітності та адміністрування користувачів. Практична цінність розробки полягає в можливості використання системи для підвищення прозорості обліку робочого часу, зменшення кількості ручних операцій, покращення контролю дисципліни та підтримки управлінської аналітики на підприємстві. Подальший розвиток системи може бути спрямований на створення вебверсії, мобільного клієнта, розширення ML-модуля аналізу

поведінкових аномалій, інтеграцію з додатковими ERP-системами та впровадження розширеної аналітики продуктивності персоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кодекс законів про працю України : Кодекс України від 10.12.1971 № 322-VIII // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/322-08
2. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/2297-17
3. Про електронні документи та електронний документообіг : Закон України від 22.05.2003 № 851-IV // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/851-15
4. Про основні засади забезпечення кібербезпеки України : Закон України від 05.10.2017 № 2163-VIII // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/2163-19
5. ISO/IEC/IEEE 29148:2011. Systems and software engineering — Life cycle processes — Requirements engineering. URL: https://www.iso.org/standard/45171.html
6. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. URL: https://www.iso.org/standard/78176.html
7. Unified Modeling Language, Version 2.5.1 / Object Management Group. URL: https://www.omg.org/spec/UML/2.5.1/About-UML
8. Business Process Model and Notation, Version 2.0.2 / Object Management Group. URL: https://www.omg.org/spec/BPMN/2.0.2/About-BPMN
9. OpenAPI Specification. Version 3.1.0 / OpenAPI Initiative. URL: https://swagger.io/specification/
10. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT) / Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/rfc7519
11. Sakimura N., Bradley J., Jones M., de Medeiros B., Mortimore C. OpenID Connect Core 1.0 incorporating errata set 2. URL: https://openid.net/specs/openid-connect-core-1_0.html
12. OWASP Top Ten Web Application Security Risks / Open Worldwide Application Security Project. URL: https://owasp.org/www-project-top-ten/
13. Python 3.11 Documentation / Python Software Foundation. URL: https://docs.python.org/3.11/
14. PyQt6 Documentation / Riverbank Computing. URL: https://www.riverbankcomputing.com/static/Docs/PyQt6/
15. PyQt Components / Riverbank Computing. URL: https://riverbankcomputing.com/software/pyqt/intro

16. Qt 6 Documentation / The Qt Company. URL: https://doc.qt.io/qt-6/
17. FastAPI Documentation / Sebastián Ramírez. URL: https://fastapi.tiangolo.com/
18. PostgreSQL Documentation / PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/
19. Redis Documentation / Redis Ltd. URL: https://redis.io/docs/latest/
20. Celery Documentation. Distributed Task Queue. URL: https://docs.celeryq.dev/
21. Docker Documentation / Docker Inc. URL: https://docs.docker.com/
22. Docker Compose Documentation / Docker Inc. URL: https://docs.docker.com/compose/
23. GitHub Actions Documentation / GitHub Inc. URL: https://docs.github.com/actions
24. SQLAlchemy Documentation. The Database Toolkit for Python. URL: https://docs.sqlalchemy.org/
25. Pydantic Documentation. Data validation using Python type hints. URL: https://pydantic.dev/docs/
26. Uvicorn Documentation. ASGI web server implementation for Python. URL: https://www.uvicorn.org/
27. pytest Documentation. URL: https://docs.pytest.org/en/stable/
28. Clockify. Time Tracking Software. URL: https://clockify.me/
29. Deltek Replicon. AI-Powered Time Tracking Software. URL: https://www.deltek.com/en/time-tracking/replicon
30. QuickBooks Time. Employee Time Tracking Software / Intuit. URL: https://quickbooks.intuit.com/time-tracking/
31. Apploye. Employee Productivity and Time Tracking Platform. URL: https://apploye.com/
32. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.
33. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York : McGraw-Hill Education, 2020. 705 p.
34. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2003. 208 p.
35. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.
36. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
37. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

38. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O’Reilly Media, 2021. 616 p.

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

40. Beazley D., Jones B. K. Python Cookbook. 3rd ed. Sebastopol : O’Reilly Media, 2013. 706 p.

ДОДАТОК А

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Гупалик Максим Миколайович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Інформаційна система оптимізації розподілу робочого часу з використанням машинного навчання» є 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 (ChatGPT) 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.

«22» травня 2026 р. _____ **Максим ГУПАЛИК**