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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Веб–застосунок для обліку відділень фінансової компанії, робочих місць AnyDesk і лімітів обороту.»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо–професійної програми «Технології цифрового розвитку»

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

_____ Володимир КИРИЛОВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ТЦР–43

Володимир КИРИЛОВСЬКИЙ

Керівник: Андрій АРОНОВ
канд. техн. наук, доц. каф.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально–науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо–професійна програма «Технології цифрового розвитку»

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кириловському Володимиру Володимировичу

1. Тема кваліфікаційної роботи: «Клієнт–серверна платформа для автоматизації процесів оренди фінансової
компанії з використанням екосистем .NET та React/Next.js»
керівник кваліфікаційної роботи Андрій АРОНОВ
затверджені наказом Державного університету інформаційно–комунікаційних технологій від « 20 » березня 2026 року № 51.
2. Строк подання кваліфікаційної роботи « 01 » травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: Вимоги до розробки веб–застосунку для обліку відділень фінансової компанії, робочих місць AnyDesk та лімітів обороту; технічний стек: TypeScript, Next.js, React, REST API, PostgreSQL, Prisma ORM; функціональні вимоги: CRUD–операції для всіх сутностей, фільтрація лімітів, каскадне видалення, валідація даних; нефункціональні вимоги: надійність, безпека, масштабованість відповідно до ISO/IEC 25010.
4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити):
   1. Аналіз предметної області та існуючих програмних рішень для автоматизації обліку інфраструктури фінансових компаній.
   2. Методологічні засади та принципи проєктування веб–застосунку на основі клієнт–серверної архітектури.

3. Проєктування бази даних та архітектури системи обліку відділень, робочих місць AnyDesk і лімітів обороту.
4. Програмна реалізація веб–застосунку з використанням Next.js, React, REST API та PostgreSQL.
5. Тестування та розгортання розробленої системи.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз існуючих рішень для автоматизації документообігу.
   2. Вимоги до програмного забезпечення.
   3. Програмні засоби реалізації.
   4. Архітектура мікросервісної системи.
   5. Алгоритм генерації договірних документів
   6. Схема бази даних .
   7. Екранні форми інтерфейсу користувача.
   8. Апробація результатів дослідження

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково–технічної літератури	22.01.2026 – 10.02.2026	
2	Аналіз та дослідження існуючих аналогів	11.02.2026 – 15.02.2026	
3	Вибір та обґрунтування технологічного стеку	16.02.2026 – 28.02.2026	
4	Проєктування мікросервісної архітектури та бази даних	01.03.2026 – 08.03.2026	
5	Програмна реалізація застосунку	09.03.2026 – 20.04.2026	
6	Тестування застосунку	21.04.2026 – 29.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	30.04.2026 – 11.05.2026	
8	Розробка демонстраційних матеріалів	12.05.2026 – 17.05.2026	
9	Підготовка презентації та попередній захист	18.05.2026 – 27.05.2026	

Здобувач(ка) вищої освіти _____ Володимир КИРИЛОВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Андрій АРОНОВ
(підпис)

РЕФЕРАТ

Мета роботи – автоматизація процесу обліку відділень фінансової компанії, робочих місць AnyDesk та лімітів обороту шляхом розробки web–застосунку для централізованого управління та моніторингу зазначених даних.

Об'єкт дослідження – процес обліку та управління інфраструктурою фінансової компанії, робочими місцями AnyDesk і лімітами обороту.

Предмет дослідження – web–застосунок для автоматизованого обліку, зберігання та управління даними про відділення фінансової компанії, робочі місця AnyDesk і ліміти обороту.

Методи дослідження – аналіз предметної області, моделювання бізнес–процесів, об'єктно–орієнтоване проєктування, розробка програмного забезпечення з використанням клієнт–серверної архітектури .

Короткий зміст роботи – у роботі проведено аналіз існуючих підходів до автоматизації обліку інфраструктури фінансових компаній, зокрема систем управління активами та корпоративними ресурсами . Розроблено web–застосунок з використанням Next.js/React на фронтенді та REST API на бекенді. Застосунок зберігає дані про відділення компанії, призначені робочі місця з віддаленим доступом AnyDesk та встановлені ліміти обороту у базі даних PostgreSQL. Реалізовано функціонал перегляду, додавання, редагування та видалення записів, а також контроль відповідності поточних показників обороту встановленим лімітам у розрізі відділень.

Галузь використання – фінансові компанії, мікрофінансові організації, кредитні спілки та інші установи фінансового сектору, що мають розгалужену мережу відділень і потребують централізованого обліку робочих місць та контролю лімітів обороту.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ФІНАНСОВА КОМПАНІЯ, ОБЛІК ВІДДІЛЕНЬ, ANYDESK, ЛІМІТИ ОБОРОТУ, REST API, NEXT.JS, REACT, POSTGRESQL, PRISMA ORM, TYPESCRIPT, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, АВТОМАТИЗАЦІЯ ОБЛІКУ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface – програмний інтерфейс застосунку

REST – Representational State Transfer – архітектурний стиль побудови веб–сервісів

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту

JSON – JavaScript Object Notation – формат обміну даними

ORM – Object–Relational Mapping – об'єктно–реляційне відображення

SQL – Structured Query Language – мова структурованих запитів

СУБД – Система управління базами даних

ERP – Enterprise Resource Planning – система планування ресурсів підприємства

CRM – Customer Relationship Management – система управління взаємовідносинами з клієнтами

UML – Unified Modeling Language – уніфікована мова моделювання

DTO – Data Transfer Object – об'єкт передачі даних

URL – Uniform Resource Locator – уніфікований локатор ресурсів

UAH – Ukrainian Hryvnia – українська гривня (код валюти)

USD – United States Dollar – долар США (код валюти)

EUR – Euro – євро (код валюти)

ER – Entity–Relationship – сутність–зв'язок (тип діаграми)

UI – User Interface – користувацький інтерфейс

ACID – Atomicity, Consistency, Isolation, Durability – властивості транзакцій БД

ID – Identifier – ідентифікатор

RDP – Remote Desktop Protocol – протокол віддаленого робочого столу

OpenAPI – специфікація для опису REST API

ВСТУП

В умовах цифровізації фінансового сектору та розширення мереж фінансових компаній виникає необхідність ефективного управління розподіленою інфраструктурою [1]. Відстеження актуального стану відділень, контроль призначених робочих місць із віддаленим доступом AnyDesk та моніторинг лімітів обороту в розрізі кожного підрозділу є критично важливими задачами для операційної діяльності компанії. Традиційний підхід до ведення такого обліку – через електронні таблиці або розрізнені документи – є трудомістким, схильним до помилок і не забезпечує оперативного доступу до актуальних даних [2].

Існуючі універсальні системи обліку активів та ERP–рішення мають широкий функціонал, однак є надмірно складними й дорогими у впровадженні [3], а також не враховують специфіки фінансових компаній із розгалуженою мережею відділень. Комплексна оцінка якості таких систем базується на багатокритеріальних моделях відповідно до міжнародних стандартів [4]. Спеціалізованих рішень, що поєднують облік відділень, управління робочими місцями AnyDesk і контроль лімітів обороту в єдиному інтерфейсі, на ринку практично немає [5]. Таким чином, існує практична потреба у розробці спеціалізованого web–застосунку, який забезпечував би централізоване зберігання та управління зазначеними даними.

Мета роботи: автоматизація процесу обліку відділень фінансової компанії, робочих місць AnyDesk та лімітів обороту шляхом розробки web–застосунку для централізованого управління та моніторингу зазначених даних.

Завдання кваліфікаційної роботи:

- проаналізувати існуючі підходи та інструменти автоматизації обліку інфраструктури фінансових компаній;
- визначити вимоги до функціоналу web–застосунку;
- спроєктувати архітектуру системи на основі клієнт–серверного підходу;
- розробити серверну частину у вигляді REST API;
- розробити клієнтську частину на основі Next.js/React;

- реалізувати функціонал обліку відділень, робочих місць AnyDesk та лімітів обороту;
- протестувати розроблену систему.

Об'єкт дослідження: процес обліку та управління відділеннями фінансової компанії, робочими місцями AnyDesk і лімітами обороту.

Предмет дослідження: інформаційна система для автоматизованого обліку, зберігання та управління даними про відділення фінансової компанії, робочі місця AnyDesk і ліміти обороту.

Методи дослідження: аналіз предметної області, моделювання бізнес–процесів, об'єктно–орієнтоване проєктування, розробка програмного забезпечення з використанням клієнт–серверної архітектури.

Практичне значення одержаних результатів: розроблений web–застосунок може бути впроваджений у фінансових компаніях, що мають розгалужену мережу відділень. Застосування системи дозволяє суттєво скоротити час на пошук та оновлення даних про відділення й робочі місця, забезпечити оперативний контроль відповідності поточних показників обороту встановленим лімітам, а також мінімізувати ризики, пов'язані з некоректним веденням обліку.

Галузь застосування: фінансові компанії, мікрофінансові організації, кредитні спілки та інші установи фінансового сектору, що мають розгалужену мережу відділень і потребують централізованого обліку робочих місць та контролю лімітів обороту.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей [6,7]:

1. Кириловський В.В., Аронов А.О. Веб–застосунок для обліку відділень фінансової компанії, робочих місць anydesk та лімітів обороту// Матеріали VII Всеукраїнської науково–технічної конференції «Застосування програмного забезпечення в інформаційно–комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026, (подано до друку).

2. Кириловський В.В., Аронов А.О. Веб–застосунок для ведення обліку

відділень фінансової компанії, робочих місць AnyDesk та контролю лімітів обороту. // Матеріали VII Всеукраїнської науково–технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026, (подано до друку).

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОБЛІКУ ІНФРАСТРУКТУРИ ФІНАНСОВОЇ КОМПАНІЇ

1.1 Загальна характеристика діяльності фінансових компаній та їх структурних підрозділів

Фінансові компанії є ключовими учасниками сучасного економічного середовища, забезпечуючи широкий спектр фінансових послуг для фізичних і юридичних осіб [8]. Основними напрямами діяльності є кредитування, грошові перекази, валютно–обмінні операції, приймання платежів і фінансове консультування, а також інші послуги, пов'язані з обігом фінансових ресурсів. Ефективність функціонування фінансової компанії значною мірою визначається організацією її внутрішньої структури, швидкістю обробки інформації та рівнем автоматизації бізнес–процесів [9].

Більшість сучасних фінансових компаній характеризується наявністю розгалуженої мережі структурних підрозділів, до складу якої входять центральний офіс, регіональні представництва, філії та окремі відділення. Кожне відділення виконує визначений перелік операцій і взаємодіє з центральною інформаційною системою компанії. Структурні підрозділи мають власні ресурси, персонал, технічне обладнання та фінансові обмеження, які потребують постійного контролю.

Організаційна структура фінансової компанії зазвичай передбачає централізоване управління всіма підрозділами. Центральний офіс контролює діяльність підрозділів, аналізує фінансові показники, встановлює ліміти обороту, контролює доступ до інформаційних ресурсів і координує роботу персоналу. Відділення здійснюють безпосередню взаємодію з клієнтами та проводять фінансові операції відповідно до внутрішніх правил компанії. Інформаційна інфраструктура.

Для безперервної роботи співробітників використовуються комп'ютерні робочі місця, спеціалізоване програмне забезпечення та мережеве обладнання.

Застосовуються також засоби віддаленого адміністрування. Одним із поширених інструментів дистанційного доступу є система AnyDesk. Вона дозволяє адміністраторам оперативно підключатися до робочих станцій працівників, виконувати технічне обслуговування та усувати несправності [10].

Зі збільшенням кількості відділень ускладнюються централізований облік робочих місць, контроль доступу та актуалізація інформації про технічну інфраструктуру [11]. За відсутності автоматизованої системи ці процеси часто виконуються вручну із використанням електронних таблиць, локальних баз даних або паперової документації, що призводить до помилок, втрати інформації та зниження ефективності управління.

Крім технічного контролю, важливим елементом управління є контроль фінансових показників діяльності кожного відділення. Для мінімізації ризиків фінансові компанії встановлюють певні ліміти обороту. Вони визначають допустимі обсяги фінансових операцій за певний період. Контроль таких показників дозволяє зменшити фінансові ризики та забезпечити дотримання внутрішніх політик компанії.

Діяльність сучасних фінансових компаній характеризується складною організаційною структурою та великою кількістю інформаційних потоків, що зумовлює необхідність постійного контролю за технічною та фінансовою інфраструктурою. Це створює передумови для впровадження автоматизованих інформаційних систем, які підвищують ефективність управління структурними підрозділами компанії.

1.2 Особливості обліку відділень та робочих місць у фінансових установах

Ефективність діяльності фінансових компаній значною мірою залежить від належного обліку структурних підрозділів і технічної інфраструктури [12]. У процесі функціонування мережі відділень формується значний обсяг інформації, що потребує постійного оновлення, контролю та систематизації. До цієї інформації

належать дані про місцезнаходження відділень, відповідальних працівників, режим роботи, статус активності підрозділів і перелік технічного обладнання.

Кожне відділення фінансової компанії може містити декілька робочих місць, які співробітники використовують для обслуговування клієнтів і виконання внутрішніх операцій. Робоче місце є сукупністю програмних і технічних ресурсів, необхідних для виконання службових обов'язків працівника. До складу такого робочого місця входять персональний комп'ютер, периферійне обладнання, доступ до внутрішніх інформаційних систем, а також засоби дистанційного адміністрування.

Одним із найбільш поширених інструментів віддаленого доступу є програмне забезпечення AnyDesk. Цей сервіс дозволяє системним адміністраторам оперативно підключатися до робочих станцій співробітників незалежно від їхнього фізичного місцезнаходження. Використання AnyDesk значно спрощує технічне обслуговування обладнання, встановлення програмного забезпечення, усунення технічних несправностей і контроль працездатності робочих місць.

У фінансових компаніях, які мають велику кількість віддалених відділень, адміністрування робочих місць стає складним процесом. Часто інформація про AnyDesk ID, закріплення конкретного обладнання за відділенням або працівником зберігається у таблицях Excel, локальних документах або окремих інформаційних системах [13]. Такий підхід ускладнює пошук необхідної інформації та створює ризик дублювання або втрати даних.

Забезпечення актуальності інформації є ключовим завданням обліку робочих місць. У разі заміни обладнання, зміни працівника або переміщення техніки між відділеннями дані мають оперативно оновлюватися. Відсутність централізованої системи обліку ускладнює цей процес, збільшує витрати часу та підвищує ймовірність помилок [14].

Окремою проблемою є контроль за активністю робочих місць. Частина обладнання може тимчасово не використовуватися або перебувати в неактивному стані. Для керівництва компанії важливо отримувати актуальну інформацію про

кількість активних робочих місць, доступні ресурси та навантаження на технічну інфраструктуру.

Автоматизація обліку відділень і робочих місць забезпечує централізацію інформації, підвищує швидкість доступу до даних і зменшує кількість помилок [15], пов'язаних із ручним веденням документації. Використання сучасних веб–технологій дозволяє отримувати доступ до системи з будь–якого пристрою та значно спрощує адміністрування технічної інфраструктури фінансової компанії. Отже, облік відділень і робочих місць є важливою складовою управління діяльністю фінансової компанії. Автоматизація цього процесу сприяє підвищенню ефективності роботи персоналу, оптимізації використання технічних ресурсів і забезпеченню стабільної роботи всієї мережі відділень.

1.3 Аналіз процесу управління лімітами фінансового обороту

Контроль фінансових операцій у структурних підрозділах є одним із ключових напрямів діяльності фінансових компаній [16]. Для забезпечення стабільності роботи, зниження фінансових ризиків і дотримання внутрішніх нормативів застосовується система лімітів фінансового обороту.

Ліміт фінансового обороту визначає граничне значення фінансових операцій, яке може бути виконане конкретним відділенням протягом певного періоду [17]. Такі обмеження встановлюються на день, тиждень, місяць або інший звітний період залежно від специфіки діяльності підприємства. Основна мета впровадження лімітів полягає у контролі обсягів фінансових операцій і запобіганні перевищенню допустимих показників.

У фінансових компаніях ліміти можуть відрізнятися залежно від типу відділення, регіону його розташування, обсягів клієнтських операцій і рівня фінансового навантаження. Наприклад, великі міські відділення можуть мати значно вищі показники допустимого обороту порівняно з невеликими регіональними представництвами.

Процес управління лімітами складається з кількох основних етапів. Спочатку

відповідальні особи аналізують фінансові показники діяльності відділення та визначають оптимальний розмір допустимого обороту. Далі встановлені значення вносяться до інформаційної системи, після чого здійснюється постійний моніторинг фактичних показників і контроль за перевищенням встановлених меж. Часовий аспект дії фінансових лімітів. Деякі ліміти можуть мати фіксований термін дії, наприклад, протягом певного місяця або кварталу [18]. Інші можуть встановлюватися безстроково до моменту їх зміни або скасування. Саме тому система повинна забезпечувати можливість збереження дати початку дії ліміту та дати його завершення.

У багатьох компаніях контроль лімітів досі здійснюється вручну із використанням електронних таблиць або окремих програмних продуктів. Такий підхід ускладнює пошук актуальної інформації, формування звітності та аналіз ефективності роботи відділень, а також підвищує ризик людських помилок.

Автоматизація управління лімітами фінансового обороту значно підвищує ефективність контролю [19]. Система автоматично перевіряє актуальність лімітів, відображає активні фінансові обмеження, формує звіти та попереджає адміністраторів про перевищення встановлених показників.

Суттєвою перевагою автоматизованого підходу є можливість швидкого аналізу фінансових даних у режимі реального часу. Керівництво компанії отримує доступ до актуальної інформації про діяльність кожного відділення та може оперативно ухвалювати управлінські рішення.

Отже, управління лімітами фінансового обороту є важливою складовою діяльності фінансових компаній. Автоматизація цього процесу мінімізує фінансові ризики, підвищує прозорість операцій і забезпечує ефективний контроль за діяльністю структурних підрозділів.

1.4 Аналіз існуючих програмних рішень для автоматизації обліку

У сучасному бізнес–середовищі фінансові компанії активно впроваджують інформаційні системи для автоматизації внутрішніх процесів. Основна мета

використання таких систем полягає в підвищенні ефективності управління структурними підрозділами, скороченні часу обробки інформації, зниженні кількості помилок та оптимізації використання ресурсів підприємства.

Для вирішення задач обліку відділень, контролю робочих місць і фінансових операцій компанії використовують різні категорії програмного забезпечення [20]. Найпоширенішими є електронні таблиці, CRM–системи, ERP–системи та спеціалізовані програми для віддаленого доступу.

Найпростішим інструментом обліку залишаються електронні таблиці Microsoft Excel і Google Sheets. Вони дозволяють швидко створювати таблиці для збереження інформації про відділення, співробітників, обладнання та фінансові показники [21]. Основною перевагою такого підходу є низька вартість впровадження та простота використання.

Однак зі збільшенням кількості відділень і обсягів інформації використання електронних таблиць стає неефективним. З'являються проблеми дублювання інформації, втрати даних, складності спільної роботи користувачів і відсутність централізованого контролю.

CRM–системи (Bitrix24, Salesforce, HubSpot) дозволяють автоматизувати взаємодію з клієнтами, документообіг і управління бізнес–процесами. Деякі фінансові компанії використовують їх для часткового ведення внутрішнього обліку [22].

Водночас CRM–системи орієнтовані переважно на клієнтські процеси, а не на управління технічною інфраструктурою чи контроль фінансових лімітів.

ERP–системи (SAP ERP, Oracle ERP, Microsoft Dynamics) забезпечують комплексне управління ресурсами підприємства. Вони підтримують фінансовий облік, управління активами, аналітику та планування ресурсів [23].

Незважаючи на широкий функціонал, ERP–системи мають суттєві недоліки:

- висока вартість впровадження;
- складність налаштування;
- потреба в додатковому навчанні персоналу;
- Надмірний функціонал для середніх компаній.

Для дистанційного адміністрування робочих місць використовуються спеціалізовані програми:

- AnyDesk
- TeamViewer
- Remote Desktop Protocol (RDP)

Ці рішення дозволяють здійснювати віддалене підключення до робочих станцій, однак не забезпечують централізованого обліку відділень і фінансового контролю [24].

Таблиця 1.1

Порівняння існуючих програмних рішень

Тип системи	Переваги	Недоліки
Excel / Google Sheets	Простота використання, низька вартість	Ручний облік, помилки, слабка безпека
CRM	Автоматизація бізнес–процесів	Не підходить для технічного обліку
ERP	Комплексний функціонал	Висока вартість
AnyDesk / TeamViewer	Віддалений доступ	Відсутність системи обліку

Аналіз показав, що наявні програмні продукти вирішують лише окремі завдання автоматизації.

Деякі системи орієнтовані на фінансовий облік, інші – на управління клієнтами, а треті забезпечують лише віддалений доступ до технічної інфраструктури.

Комплексного рішення, яке б одночасно забезпечувало:

- облік відділень
- контроль робочих місць
- управління AnyDesk ID
- контроль лімітів обороту

– формування звітності

На сьогодні комплексних рішень практично не існує. У зв’язку з цим виникає необхідність розробки власного вебзастосунку.

Рис. 1.1. Класифікація існуючих програмних рішень для автоматизації обліку

На рисунку 1.1 представлено класифікацію основних програмних рішень, які підприємства використовують для автоматизації облікових процесів. Як видно зі схеми, кожен тип програмного забезпечення орієнтований на розв’язання окремого класу задач.

Електронні таблиці використовуються переважно для базового ведення обліку та зберігання інформації, проте характеризуються низьким рівнем автоматизації й високою ймовірністю виникнення помилок під час ручного введення даних. CRM–системи орієнтовані на управління клієнтськими процесами та документообігом, однак їхній функціонал є недостатнім для повноцінного контролю технічної інфраструктури компанії.

ERP–системи забезпечують комплексне управління ресурсами підприємства, проте їх впровадження потребує значних фінансових витрат і складного налаштування. Системи віддаленого доступу дозволяють здійснювати технічне обслуговування робочих місць, однак не містять функцій централізованого обліку структурних підрозділів і фінансових лімітів.

Аналіз показав, що жодне з наявних рішень не забезпечує одночасного виконання всіх функцій, необхідних для фінансової компанії, зокрема обліку

відділень, контролю робочих місць AnyDesk та управління лімітами обороту. Це обґрунтовує необхідність розробки спеціалізованого вебзастосунку, який буде розглянуто у наступних розділах дипломної роботи.

1.5 Переваги та недоліки існуючих підходів до автоматизації обліку

Після аналізу наявних програмних рішень доцільно детальніше розглянути переваги та недоліки підходів, що використовуються для автоматизації обліку у фінансових компаніях [25]. Різні організації застосовують як універсальні програмні продукти, так і власні локальні рішення, що залежать від масштабів діяльності підприємства, кількості структурних підрозділів та обсягів фінансових операцій.

Найпростішим підходом є використання електронних таблиць для ведення обліку відділень, робочих місць і фінансових показників. Основною перевагою такого підходу є простота використання, доступність і мінімальні фінансові витрати на впровадження. Працівники можуть швидко створювати таблиці та вносити необхідну інформацію без використання спеціалізованого програмного забезпечення.

Однак ручне ведення обліку має чимало недоліків. Збільшення кількості відділень та обсягу інформації підвищує ризик дублювання даних, втрати інформації та виникнення помилок під час ручного введення [26]. Крім того, відсутні автоматична синхронізація між користувачами та механізми контролю доступу.

CRM–системи дозволяють автоматизувати бізнес–процеси підприємства та покращити управління взаємодією з клієнтами. Їхньою перевагою є наявність готового функціоналу, хмарних сервісів і можливостей інтеграції з іншими платформами.

Недоліком CRM–рішень є недостатня адаптація до специфіки технічного обліку робочих місць і контролю фінансових лімітів. Для реалізації необхідного функціоналу компанії часто змушені замовляти додаткові модулі або змінювати

стандартну структуру системи.

ERP–системи характеризуються найширшим функціоналом і дозволяють здійснювати комплексне управління ресурсами підприємства [27]. Вони підтримують фінансовий облік, аналітику, логістику та управління персоналом.

Разом із тим, впровадження ERP–рішень вимагає значних фінансових витрат, тривалого налаштування та спеціального навчання персоналу. Для середніх фінансових компаній такі системи часто є економічно недоцільними.

Системи віддаленого доступу, такі як AnyDesk або TeamViewer, значно спрощують адміністрування технічної інфраструктури компанії. Вони дозволяють оперативно усувати технічні проблеми та здійснювати віддалене обслуговування робочих місць [28].

Основним недоліком таких систем є відсутність функцій централізованого обліку та аналітики. Вони вирішують лише технічну частину задачі, не охоплюючи фінансових аспектів діяльності компанії.

Для більш наочного представлення результатів аналізу переваг і недоліків існуючих підходів доцільно використати порівняльну таблицю.

Таблиця 1.2

Порівняння існуючих підходів автоматизації

Підхід	Переваги	Недоліки
Електронні таблиці	Простота, доступність	Ручний облік, помилки
CRM–системи	Автоматизація бізнес–процесів	Недостатній технічний функціонал
ERP–системи	Комплексне управління	Висока вартість
AnyDesk	Віддалений доступ	Відсутність облікової системи

Проведений аналіз показує, що кожен із розглянутих підходів має власні переваги, проте жоден із них не забезпечує комплексного вирішення задач обліку відділень, контролю робочих місць та управління фінансовими лімітами в межах

єдиної системи.

Таким чином, виникає необхідність у створенні спеціалізованого веб–застосунку, який дозволить об’єднати всі необхідні функції в одному програмному продукті та забезпечить ефективне управління діяльністю фінансової компанії.

1.6 Постановка задачі автоматизації та формування вимог до інформаційної системи

Проведений аналіз предметної області, особливостей обліку відділень, робочих місць та управління фінансовими лімітами показав наявність низки проблем, що виникають під час використання традиційних підходів до ведення обліку. Використання електронних таблиць, окремих програмних продуктів та ручного введення даних значно ускладнює процес управління інформацією, знижує оперативність ухвалення рішень і підвищує ризик виникнення помилок.

Основною проблемою є відсутність єдиного програмного середовища, яке дозволяло б централізовано управляти всіма структурними підрозділами фінансової компанії, контролювати робочі місця співробітників та здійснювати моніторинг фінансових лімітів обороту [29].

У процесі функціонування компанії виникає необхідність постійного оновлення інформації про відділення, реєстрації нових робочих місць, контролю активності обладнання, а також відстеження змін фінансових обмежень. За відсутності автоматизованої системи виконання таких операцій це потребує значних витрат часу та залучення додаткових людських ресурсів.

Для вирішення виявлених проблем було сформовано задачу розробки спеціалізованого веб–застосунку, який забезпечить автоматизацію основних процесів обліку та централізоване управління даними [30].

Основними функціональними вимогами до системи є:

- ведення обліку відділень фінансової компанії;
- збереження інформації про місцезнаходження відділень;
- реєстрація та облік робочих місць;

– збереження AnyDesk ID для кожного робочого місця;
– контроль статусу активності робочих місць;
– створення та редагування лімітів фінансового обороту;
– контроль термінів дії фінансових лімітів;
– формування звітності;
– пошук та фільтрація інформації;
– Управління користувачами системи.

Крім функціональних вимог, система повинна відповідати низці нефункціональних вимог.

До основних нефункціональних вимог належать:

– висока швидкість роботи системи;
– зручний інтерфейс користувача;
– надійність збереження інформації;
– захист даних від несанкціонованого доступу;
– масштабованість;
– можливість подальшого розширення функціоналу;
– кросплатформеність;
– Стабільність роботи за великої кількості користувачів.

Для реалізації зазначених вимог доцільним є використання сучасних веб–технологій, які дозволяють створити централізовану систему з доступом через браузер без необхідності встановлювати додаткове програмне забезпечення.

Рис. 1.2. Загальна схема роботи інформаційної системи

На рисунку 1.2 наведено загальну логіку функціонування майбутньої інформаційної системи. Вхідною інформацією є дані про відділення компанії,

робочі місця та фінансові ліміти. Усі дані зберігаються в централізованій базі даних та обробляються вебзастосунком. Адміністратор системи має можливість контролювати актуальність інформації, редагувати записи та здійснювати моніторинг діяльності структурних підрозділів компанії.

Розробка такої системи дозволить автоматизувати внутрішні бізнес–процеси фінансової компанії, підвищити ефективність управління ресурсами та забезпечити якісніший контроль фінансової й технічної інфраструктури підприємства [31].

Таким чином, у першому розділі було проведено аналіз предметної області, визначено основні проблеми наявних підходів і сформовано вимоги до майбутньої інформаційної системи. Це створює основу для подальшого проєктування архітектури програмного забезпечення у наступному розділі.

2 МЕТОДОЛОГІЧНІ ЗАСАДИ ТА ПРИНЦИПИ ПРОЄКТУВАННЯ ВЕБ–ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ІНФРАСТРУКТУРИ ТА ЛІМІТІВ

2.1. Обґрунтування напрямку розробки та методів розв’язання задач

Динаміка трансформації фінансового сектору диктує жорсткі вимоги до адміністрування розгалужених мереж відділень, де на перший план виходить потреба в оптимізації процесів моніторингу. Специфіка роботи подібних установ невідривно пов’язана з критичною необхідністю тотального контролю операційних лімітів. Водночас виникає супутнє завдання підтримки безперебійного віддаленого доступу до терміналів через спеціалізовані клієнти рівня AnyDesk.

Формування вектора розробки спирається на глибинний аудит вразливих вузлів інфраструктурного управління [32]. Дослідження предметної області вказує на низку системних проблем. Зокрема децентралізація накопичених даних неминуче продукує статистичні похибки під час формування консолідованої звітності. Паралельно з цим постає питання синхронного відстеження часових обмежень фінансових оборотів, ігнорування яких загрожує компанії штрафними санкціями чи блокуванням транзитних рахунків. Нарешті технічна інвентаризація підключень вимагає жорсткої фіксації ідентифікаторів робочих станцій задля оперативного супроводу та підтримки безпеки з'єднань.

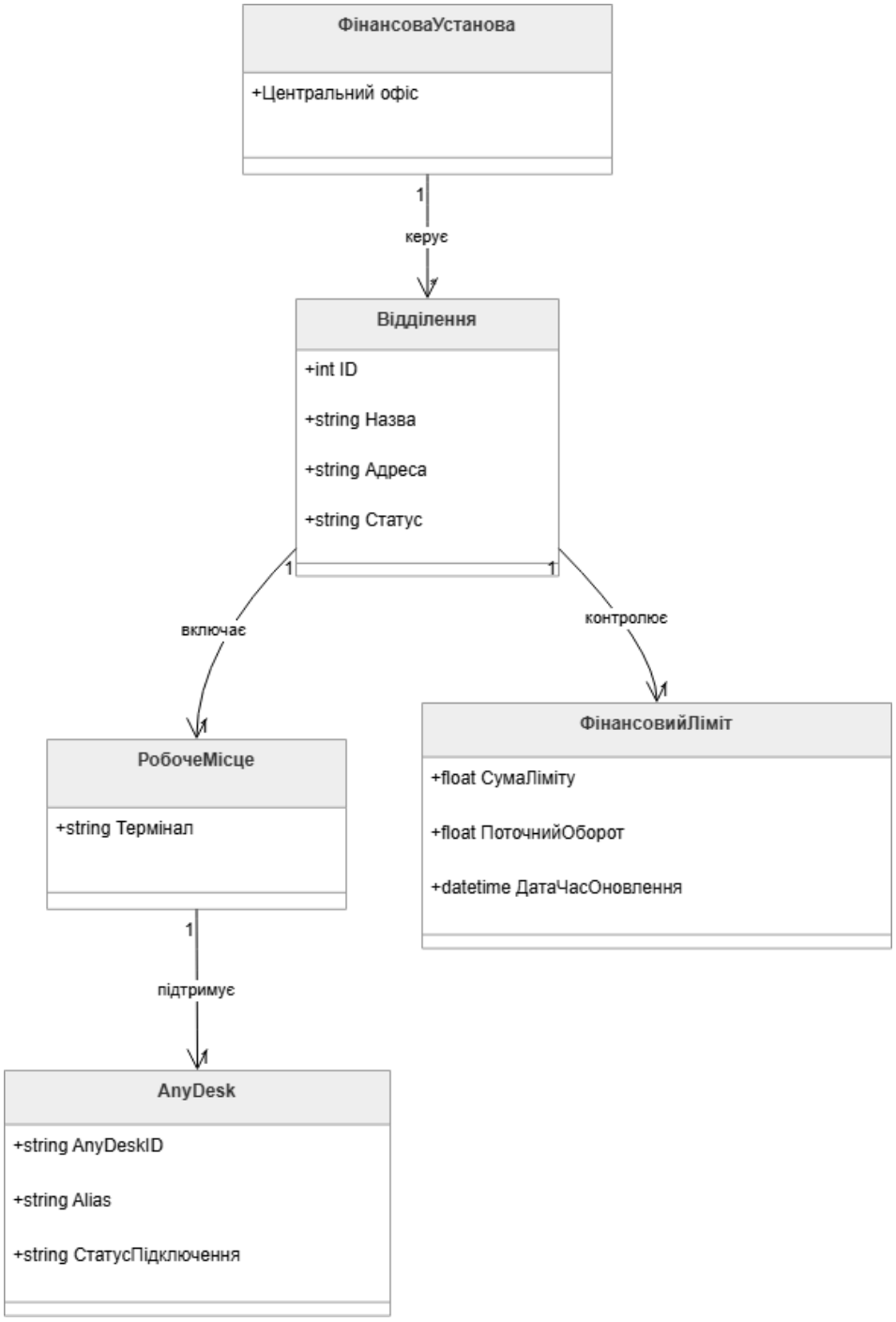

Рис. 2.1. Концептуальна схема об'єктів обліку фінансової компанії

Подолання окреслених бар'єрів вбачається у проектуванні єдиного веб–орієнтованого інформаційного середовища. Ставка на браузерні технології спирається на їхню природну кросплатформенність і відмову від локального розгортання додаткових клієнтів. Цей фактор часто виступає вирішальним тригером збереження високого темпу операційної діяльності установи.

Проектувальна парадигма ґрунтується на засадах системного підходу.

Розмежування логіки збереження даних та клієнтського інтерфейсу досягається шляхом імплементації класичної архітектури клієнт–сервер, гарантуючи тим самим централізоване керування інформаційним пулом відділень і захист цілісності записів. Разом з цим застосування об’єктно–орієнтованого аналізу (ООА) з інтеграцією строгої типізації мови TypeScript зводить до мінімуму ризики виникнення архітектурних аномалій на етапі кодування [33]. Можна припустити, що саме жорстка типізація відіграє запобіжну роль при маніпуляціях з чутливими показниками фінансових оборотів.

Таблиця 2.1.

Порівняльна характеристика методів автоматизації обліку в фінансових установах

Критерій оцінки	Електронні таблиці (Excel)	Десктопні рішення
Централізація даних	Низька; високий ризик десинхронізації	Середня; залежить від топології локальної мережі
Кросплатформенність	Жорстка прив'язка до пропрієтарних екосистем	Потребує портування під різні ОС
Моніторинг лімітів	Статичний; вимагає ручного оновлення	Ускладнений затримками реплікації баз даних
Адміністрування ПЗ	Ручне поширення актуальних файлів	Вимагає втручання технічного персоналу на місцях

Фундаментальною вимогою до майбутньої системи залишається впровадження алгоритмів безперервного контролю лімітів. Відповідна розрахункова методика оперує дискретними часовими інтервалами. Програмний комплекс має не лише акумулювати сумарні показники, але й динамічно

проектуючи їх на межі поточного операційного дня, звіряти транзакції з індивідуальними квотами кожного окремого філіалу.

Потреба у подальшому горизонтальному масштабуванні диктує вибір конкретних архітектурних патернів. Звертаючись до результатів досліджень функціонування СУБД PostgreSQL у розподілених середовищах, логічно дійти висновку про доцільність застосування саме реляційних моделей. Вони оптимально закривають потребу в дотриманні ACID–властивостей під час конкурентної зміни фінансових лімітів [34].

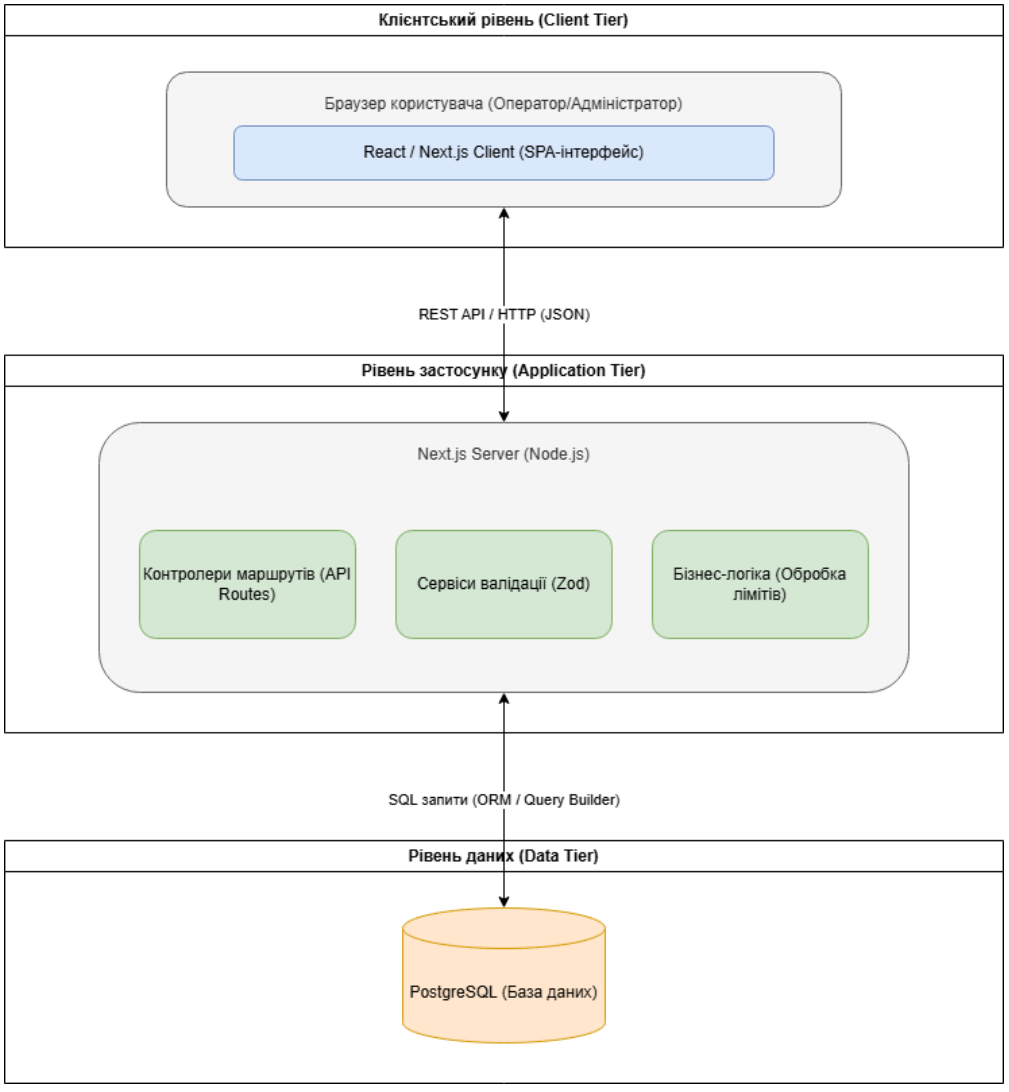

Рис. 2.2. Загальна схема архітектури системи та потоків даних (Data Flow)

Архітектурний стиль Representational State Transfer (REST) обрано стандартом комунікації між клієнтським застосунком та серверним ядром [35].

Залучення текстового формату JavaScript Object Notation (JSON) як основного носія повідомлень спирається на його обчислювальну легковажність і глибоку нативну інтеграцію з екосистемою JavaScript. Цей підхід радикально знижує мережеві затримки під час масового опитування статусів робочих терміналів AnyDesk у межах глобальної інфраструктури компанії.

2.2. Порівняльна оцінка підходів до побудови архітектури та управління даними

Формування технологічного стеку об'єктивно потребує відмови від шаблонних рішень на користь предметно–орієнтованого аналізу. Під час проектування інтерфейсного шару та механізмів транзакційного доступу до даних виникла потреба критично переосмислити усталені парадигми. Розглядаючи моделі рендерингу, доцільно зіставити класичні односторінкові додатки SPA з гібридними рішеннями на базі серверної генерації SSR [36]. Спираючись на результати тестування продуктивності фронтенд–архітектур, спостерігаємо закономірність: висока реактивність клієнтського виконання супроводжується відчутною деградацією метрики першого відтворення контенту.

Оскільки проектований інструментарій оперуватиме масивними наборами інформації, включаючи розширені реєстри філіалів і сотні ідентифікаторів терміналів, доцільним видається застосування фреймворку Next.js [37]. Такий архітектурний маневр переносить вагому частку обчислень на серверні потужності.

Це відчутно знижує вимоги до апаратного оснащення кінцевих вузлів, з якими безпосередньо працюють адміністратори компанії.

Таблиця 2.2.

Порівняльна характеристика підходів до рендерингу

Критерій оцінки	Класичний підхід SPA	Гібридна архітектура Next.js
Швидкість першого відтворення	Низька; зумовлена необхідністю завантаження об'ємних бандлів скриптів	Висока; браузер миттєво отримує сформовану сервером розмітку
Апаратне навантаження на клієнта	Критичне; процеси парсингу та виконання логіки повністю лягають на термінал користувача	Помірне; найважчі обчислювальні процеси заздалегідь делеговані серверу
Пошукова оптимізація SEO	Ускладнена; вимагає інтеграції милиць у вигляді додаткових механізмів пререндеру	Нативна; пошукові роботи індексують готовий текстовий документ

Фундаментом стабільності будь–якого фінансового інструменту виступає шар абстракції бази даних, де нульова толерантність до розбіжностей типів набуває статусу критичної вимоги. Під час пошуку оптимального механізму взаємодії з СУБД розглядалися три полярні стратегії: конструювання прямих SQL–запитів, впровадження класичних об'єктно–реляційних відображень рівня TypeORM та застосування декларативних Data–Mapper рішень. Від прямих звернень до бази довелося відмовитися одразу. Причина криється у високій ймовірності ін'єкцій та абсолютній сліпоті компілятора щодо типізації на етапі збірки. Зрештою вибір консолідувався навколо Prisma ORM. Цей інструмент автоматично генерує клієнт доступу, спираючись на сувору декларативну схему, що органічно доповнює екосистему TypeScript.

Для забезпечення цілісності даних розроблено реляційну схему (Рис. 2.3), де

ключовою сутністю виступає Branch (Відділення). Вона має відношення «один–до–багатьох» із сутностями TurnoverLimit (Ліміти) та Desk (Робочі місця). Такий підхід нормалізації бази даних дозволяє уникнути дублювання інформації та забезпечує каскадне оновлення або видалення пов'язаних записів за допомогою зовнішніх ключів (Foreign Keys), що керуються на рівні Prisma ORM [38].

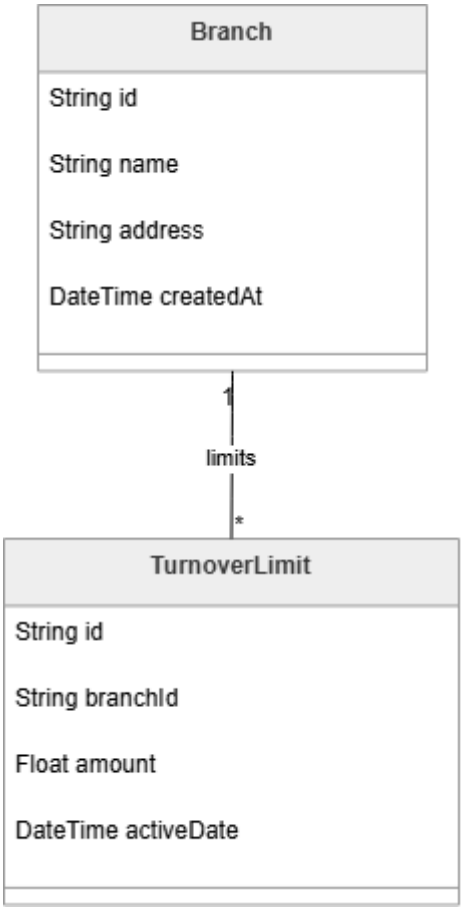

Рис. 2.3. ER–діаграма сутностей бази даних та їх зв'язків

Організація контролю доступу підпорядковується концепції єдиного адміністративного центру. Взявши до уваги ізольованість корпоративного контуру установи, доцільно свідомо редукувати рольову модель до єдиного профілю інфраструктурного адміністратора [39]. Такий крок радикально зрізає обчислювальні витрати на перевірку прав авторизації, форсуючи загальну пропускну здатність програмного інтерфейсу. Сама ж процедура маніпуляції фінансовими квотами спирається на принцип закритого периметра даних. Можна припустити, що на поточному етапі розгортання продукту найменш ризикованим

шляхом імпорту лімітів залишається пряме введення через захищені інтерфейсні форми. Це виключає потребу в потоковій синхронізації із зовнішніми банківськими шлюзами.

Таблиця 2.3.

Порівняння методів синхронізації фінансових лімітів

Критерій оцінки	Пряма інтеграція з API банку	Ручне та пакетне адміністрування
Інженерна складність	Висока; вимагає імплементації складних механізмів автентифікації та обробки таймаутів	Мінімальна; базується на стандартних операціях створення та оновлення записів
Вразливість до зовнішніх збоїв	Критична; падіння банківського шлюзу блокує роботу внутрішньої системи	Відсутня; система функціонує абсолютно автономно
Динаміка впровадження	Повільна; потребує тривалого тестування та узгодження протоколів	Висока; функціонал готовий до експлуатації одразу після розгортання схеми бази

Обрана стратегія архітектурно захищає продукт. Вона формує гарантію того, що будь–яка деградація зовнішніх фінансових сервісів не спровокує каскадне блокування інфраструктурного обліку компанії. За таких умов СУБД PostgreSQL перетворюється на єдине і безумовне джерело істини щодо поточних транзакційних оборотів філіалів. Транзакційна ж цілісність підтримується виключно внутрішніми механізмами обраного Data–Mapper клієнта.

2.3 Принципи дії та загальні характеристики розробленого програмного забезпечення

Фундаментальним базисом створеного програмного продукту для обліку фінансових відділень, лімітних квот та ідентифікаторів терміналів виступає жорстка клієнт–серверна біфуркація [40]. Проєктування складних інформаційних систем диктує об'єктивну необхідність фізичної та логічної фрагментації обчислювального ядра на автономні модулі. З одного боку, серверний бекенд перебирає на себе первинний обчислювальний тягар маршрутизації бізнес–логіки та транзакційних звернень до системи управління базами даних. З іншого – адміністративний клієнтський додаток функціонує виключно для задоволення операторських потреб, проходячи глибоку оптимізацію задля прискореного рендерингу в браузері. Цей архітектурний маневр мінімізує ступінь зв'язності програмного коду, одночасно максимізуючи його внутрішню згуртованість. Модифікація користувацького інтерфейсу жодним чином не вимагає втручання в серверну логіку.

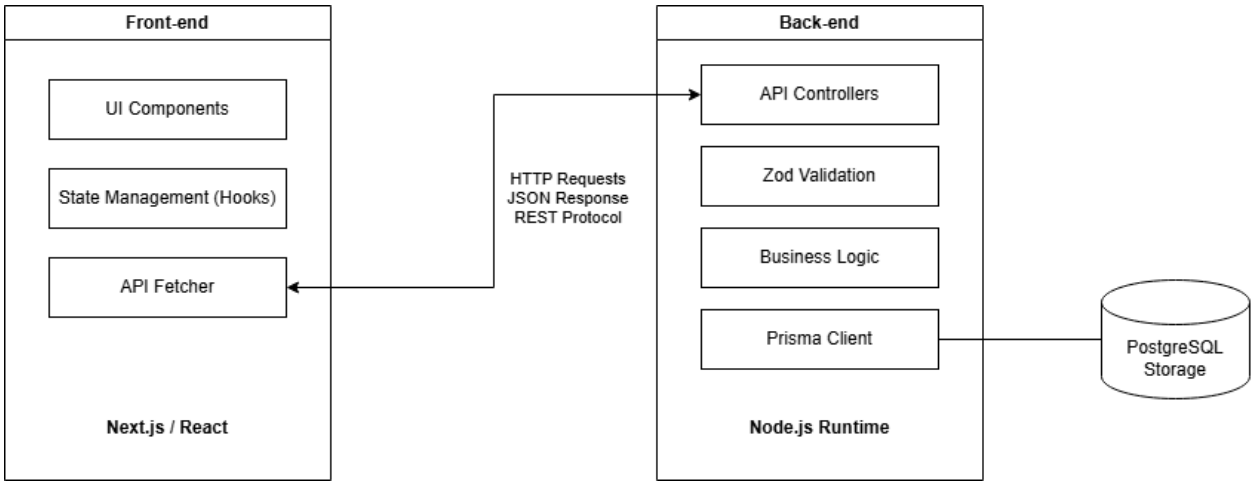

Рис. 2.4. Схема логічної декомпозиції модулів системи

Звернення до структурних зв'язків, проілюстрованих на схемі логічної декомпозиції, чітко вказує на імплементацію фундаментального принципу розділення відповідальності. Клієнтський сегмент, спроєктований на базі

фреймворку Next.js, бере на себе виключно інкапсуляцію логіки представлення разом із керуванням життєвим циклом користувацького сеансу. Відкидаючи монолітну парадигму, цей модуль навмисно позбавлений прямого доступу до фізичного шару зберігання. Будь–які маніпуляції з інфраструктурними об'єктами проходять через абстрагований сервісний прошарок шляхом ініціалізації асинхронних мережевих викликів. Подібна ізоляція видається критично необхідною. Вона форсує швидкість відгуку інтерфейсу, одночасно зрізаючи паразитний трафік завдяки транспортуванню лише строго детермінованих фрагментів JSON.

Своєю чергою, серверний вузол концентрує в собі всю вагу обчислень бізнес–логіки, структурно розпадаючись на кілька жорстко пов'язаних підсистем. Первинним бар'єром виступає шар маршрутизації, який перехоплює HTTP–запити для їхньої подальшої класифікації. Одразу за ним активується підсистема валідації. Вона здійснює агресивний контроль вхідних параметрів, безжально відсікаючи логічно некоректні дані чи аномальні ідентифікатори ще до моменту їхньої інтеграції в транзакційний цикл. Фінальним етапом обробки керує модуль об’єктно–реляційного відображення Prisma ORM, гарантуючи типізовану трансляцію команд до СУБД PostgreSQL.

Можна стверджувати, що подібна архітектура досягає стану абсолютної ізоляції бази даних від агресивного зовнішнього середовища. Сервер фактично редукується до ролі захисного проксі–вузла, підтримуючи непорушність ACID–транзакцій під час мутації фінансових лімітів. Окрім превентивного блокування SQL–ін'єкцій, такий розподіл закладає надійний плацдарм для горизонтального масштабування. Якщо корпоративна мережа відділень зазнає різкого розширення, рознесення клієнтського та серверного компонентів по різних хмарних кластерах відбудеться без жодних модифікацій вихідного коду.

Сама ж міжмодульна комунікація через REST API фіксує перехід до суворого контрактного підходу. Обидва кінці системи спираються на заздалегідь узгоджену структуру обміну повідомленнями. Це розв'язує руки для паралельної імплементації нових функцій. Зворотна сумісність при цьому не порушується,

консервуючи загальну стабільність програмного комплексу.

Розподілена взаємодія ізольованих вузлів підпорядковується принципам стилю передачі репрезентативного стану (REST). Вибір текстового формату обміну даними на базі синтаксису об'єктів JavaScript зумовлений його граничною легковаговістю. Цей крок радикально скорочує накладні витрати на серіалізацію під час транзиту пакетів через корпоративну мережу [41].

Логічним продовженням архітектурного поділу стає оркестрація інформаційними потоками, яка суворо підпорядковується постулатам реляційної парадигми. Підтримка консистентності інфраструктурних масивів спирається на зовнішні ключі, оснащені тригерами каскадного оновлення та видалення. Вони конструюють нерозривний посилальний зв'язок між первинними доменними сутностями: відділеннями, фінансовими квотами та робочими терміналами. Відповідно, генерація "осиротілих" записів виключається на фундаментальному рівні рушія СУБД. Це критична вимога для фінансового сектору.

Можна обережно припустити, що саме двоетапна валідація відіграє вирішальну роль у стабілізації вхідних потоків. Дані, ініційовані клієнтом, проходять жорстку фільтрацію через декларативні схеми безпосередньо на межі програмного інтерфейсу. Сервер категорично відкидає аномальні структури. Аномалії блокуються миттєво. Разом з тим ручне складання специфікацій API поступилося місцем концепції автоматизованої генерації технічної документації, що закладає безперервне підґрунтя для контрактного тестування.

Уся ця прихована механіка зрештою зводиться до єдиної мети – обслуговування матриці користувацької взаємодії, вибудованої навколо ключового актора, яким виступає Оператор інфраструктури. Керівна логіка спрямована на абсолютну мінімізацію когнітивного навантаження на профільного фахівця. Глибинний аналіз прецедентів формує чіткі поведінкові патерни. Передусім фахівець набуває безперервного доступу до агрегованих метрик стану мережі безпосередньо через головну панель. Подальший функціональний простір охоплює повний цикл адміністрування філіалів, диригування життєвим циклом віддалених робочих місць та конфігурацію фінансових квот із жорстким співвіднесенням

транзакційних обсягів з їхнім хронологічним контекстом.

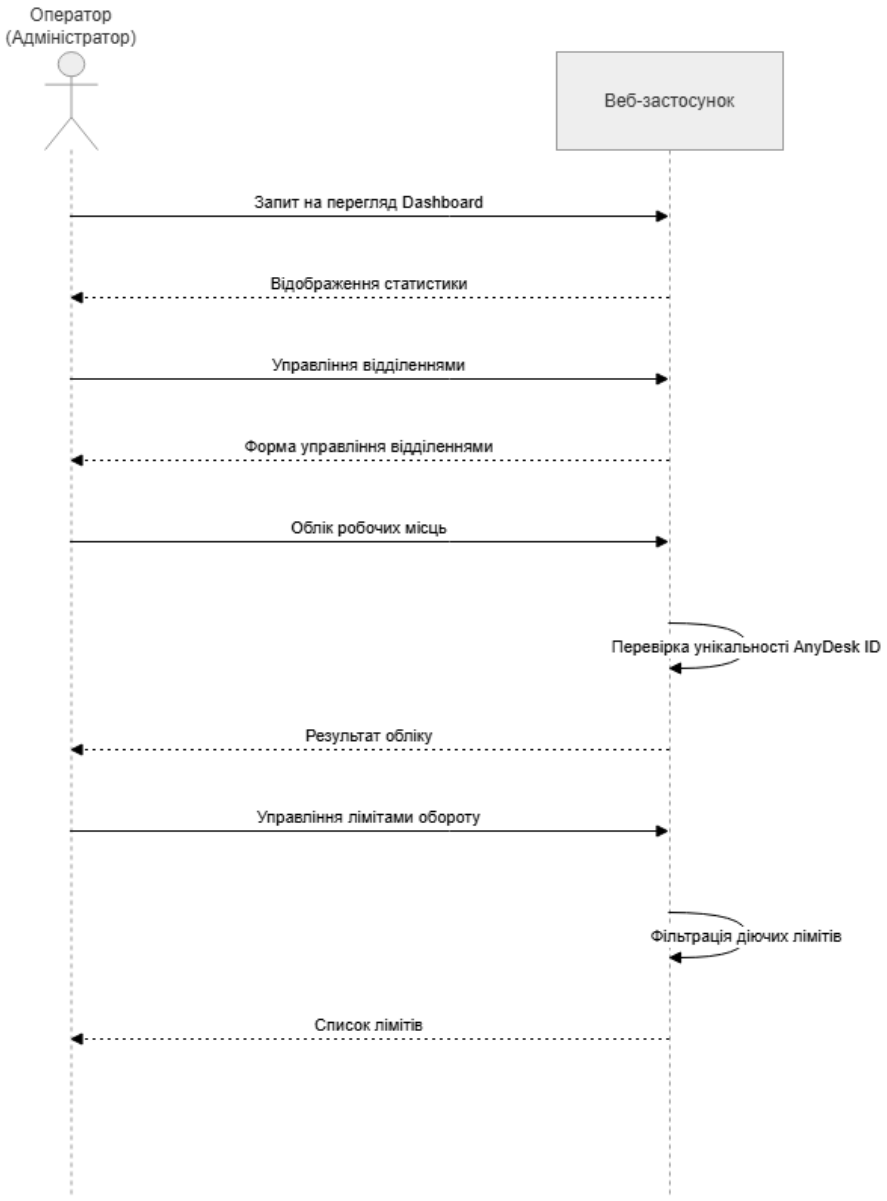

Рис. 2.5. Концептуальна діаграма прецедентів взаємодії оператора з системою

Оцінюючи підсумкові технічні метрики скомпільованого продукту, зібрані дані дозволяють констатувати вичерпне покриття цільової предметної області. Програмний комплекс демонструє підвищені параметри відмовостійкості, де наскрізна статична типізація виступає фундаментальним гарантом нейтралізації непередбачуваної поведінки системи в процесі виконання (Runtime). Звертаючись до питань ергономіки, категорична відмова від повного перезавантаження екранів на користь реактивного асинхронного рендерингу радикально знижує втому

оператора. Динамічна мутація об'єктної моделі документа сприяє миттєвому візуальному відображенню змінених лімітів.

Завершує архітектурний ансамбль розгорнутий захисний периметр, який повністю розриває прямі комунікаційні канали між клієнтським браузером та кластером бази даних. Інтегровані механізми об'єктно–реляційного відображення автономно санітизують усі вхідні потоки. Синтез цих рішень створює непроникний бар'єр проти ін'єкцій шкідливого коду, адаптуючи систему до реальних експлуатаційних навантажень.

2.4. Характеристики надійності, відмовостійкості та безпеки системи

Процедура валідації програмного продукту становить фундаментальний етап проєктування. Звертаючись до специфіки фінансового сектору, варто визнати: ціна технічної аномалії чи компрометації інформації тут набуває катастрофічних масштабів. Спираючись на регламенти міжнародного стандарту ISO/IEC 25010, комплексна оцінка розробленого інструментарію базується на багатокритеріальній моделі якості. Відповідно до цих вимог, архітектура веб–застосунку формувалася з огляду на кілька визначальних векторів.

Аналізуючи показники функціональної придатності та продуктивності, система демонструє вичерпну повноту закладених алгоритмів. Програмний комплекс покриває весь заявлений спектр операцій з адміністрування філіалів, транзакційних лімітів та ідентифікаторів терміналів AnyDesk. Оптимізація обчислювальної ефективності досягається інтеграцією гібридного рендерингу фреймворку Next.js. Завантаження масивів інформації ініціюється асинхронно. Натомість циркуляція повідомлень у легкому текстовому форматі радикально мінімізує експлуатацію пропускної здатності корпоративних каналів зв'язку. Внаслідок цього фахівець миттєво оперує актуальними метриками лімітів незалежно від апаратної деградації локальної мережі.

Окремої уваги потребують параметри надійності та відмовостійкості. Спираючись на метрики згаданого стандарту, можна констатувати високий рівень

зрілості продукту. Фундаментальним архітектурним маневром, який формує цей показник, виступає впровадження наскрізної статичної типізації синтаксису TypeScript. Звертаючись до результатів галузевих досліджень, сувора типізація нейтралізує до 15% прихованих дефектів безпосередньо на стадії компіляції. Це обрізає ймовірність раптових відмов під час активної експлуатації інтерфейсу. Паралельно з цим відмовостійкість інформаційного шару підтримується реляційною СУБД PostgreSQL. Будь–яка ініціація зміни квоти чи переприв'язки термінала інкапсулюється в межах жорстко ізольованої ACID–транзакції. У випадку виникнення мережевої аномалії під час збереження, рушій негайно ініціює відкат процесу. Система автоматично повертається до попереднього узгодженого стану.

Враховуючи маніпуляцію чутливими фінансовими лімітами та точками віддаленого проникнення, питання інформаційної захищеності набуває статусу безальтернативного пріоритету. Архітектурна модель жорстко підпорядковується принципам конфіденційності та цілісності. На практиці це виражається у повній ізоляції кластера бази даних від зовнішнього світу: клієнтський браузер фізично позбавлений можливості прямого звернення до таблиць. Будь–які несанкціоновані модифікації блокуються інтегрованим механізмом багаторівневої валідації. Первинним редутом виступають декларативні схеми на базі бібліотеки Zod. Вони суворо типізують вхідні мережеві запити. Наступним етапом фільтрації керує Prisma ORM. Цей інструмент автономно санітизує всі змінні перед формуванням підготовлених виразів для бази даних. Завдяки такій багатошаровості вектори атак через ін'єкції шкідливого коду нівелюються практично повністю.

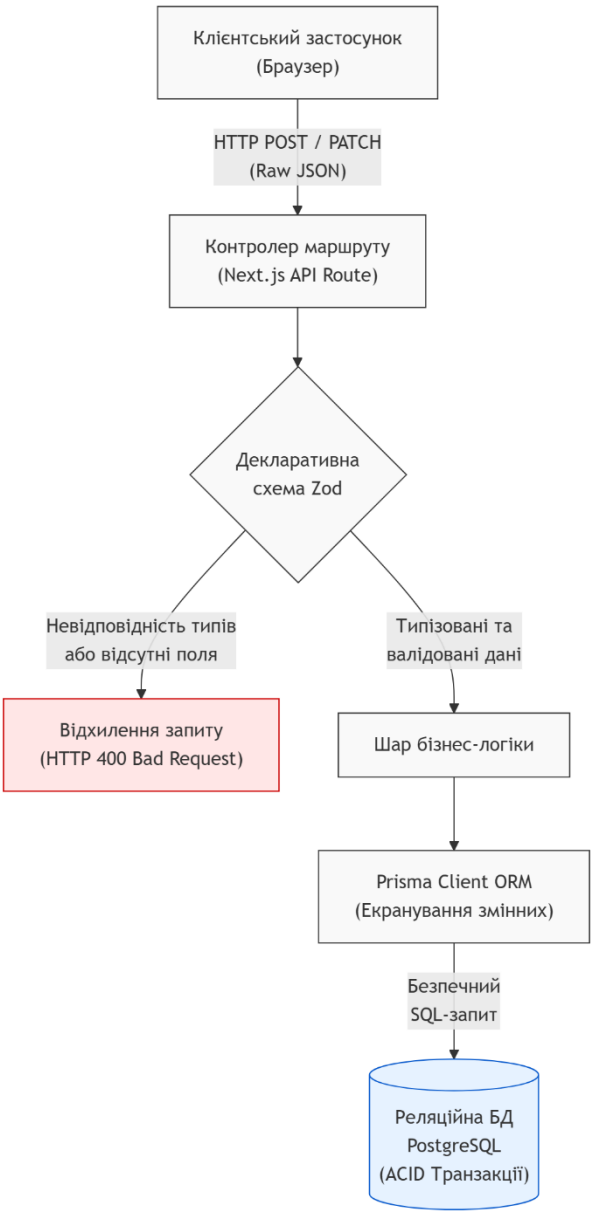

Рис. 2.6. Багаторівнева архітектура безпеки та обробки даних (Security Data Flow)

Структурна логіка, проілюстрована на Рис. 2.6, спирається на концепцію глибокоешелонованого захисту. Процес транзиту інформації навмисно фрагментовано на ізольовані стадії фільтрації. Будь–який вхідний JSON–пакет, ініційований клієнтським вузлом, перехоплюється контролером і відразу спрямовується до валідатора на базі бібліотеки Zod. Останній функціонує як жорсткий пропускний бар'єр. Якщо морфологія даних відхиляється від еталонної схеми – скажімо, фіксується спроба передати текстовий рядок замість числового номіналу ліміту – запит негайно відхиляється з відповідним HTTP–кодом помилки.

База даних при цьому не зазнає жодного паразитного навантаження.

Пакети, які успішно подолали етап суворої типізації, делегуються об'єктно–реляційному відображенню. На цьому рубежі активується другий ешелон безпеки, побудований навколо механізму автоматичного параметризування запитів. Інструментарій Prisma Client примусово розмежовує синтаксис виконуваних SQL–команд та користувацьке корисне навантаження. Аналізуючи цей підхід, логічно припустити, що подібна ізоляція зводить імовірність успішної ін'єкції шкідливого коду до математичного мінімуму. Фінальною ланкою процесу виступає безпосередня фіксація стану в PostgreSQL, де непорушність збережених масивів інформації спирається на фундаментальні властивості ACID–транзакцій

Дослідження ергономічних показників виявляє пряму кореляцію з рівнем когнітивного навантаження на оператора. Категорична відмова від синхронного перезавантаження сторінок, притаманного архаїчним рішенням, на користь реактивного підходу радикально підвищує керованість інтерфейсу. Співробітник здатен безперервно аналізувати масивні таблиці філіалів, не втрачаючи фокусу через візуальні артефакти або мерехтіння дисплея.

Водночас логічна декомпозиція загальної архітектури на автономні модулі формує високий потенціал для подальшого супроводу та переносимості. Інфраструктурний код схем ізольований від візуальної логіки. Цей фактор відкриває шлях до безперешкодного масштабування програмного комплексу та імплементації нових сутностей без загрози колапсу поточного функціонального ядра. Задля систематизації розглянутих архітектурних концептів та підтвердження їхньої відповідності згаданим міжнародним регламентам сформовано зведену матрицю характеристик.

Таблиця 2.4.

Матриця відповідності характеристик системи критеріям стандарту ISO/IEC 25010

Критерій за ISO/IEC 25010	Опис критерію (згідно зі стандартом)	Архітектурна та технічна реалізація у застосунку
Функціональна придатність (Functional Suitability)	Здатність системи надавати функції, що задовольняють встановлені та передбачувані потреби бізнесу за визначених умов.	Імплементація вичерпного циклу операцій управління станом (CRUD) для всіх базових інфраструктурних об'єктів: відділень, фінансових лімітів та терміналів AnyDesk.
Продуктивність (Performance Efficiency)	Забезпечення належної швидкодії та ефективного використання апаратних і мережевих ресурсів під час виконання функцій.	Застосування гібридного підходу до рендерингу та обмін повідомленнями у легковаговому форматі JSON через REST API, що мінімізує мережеві затримки.
Надійність (Reliability)	Здатність системи виконувати задані функції без збоїв протягом певного проміжку часу (відмовостійкість та зрілість).	Використання наскрізної статичної типізації TypeScript для превентивного виявлення помилок під час компіляції.

Таблиця 2.5.

Матриця відповідності характеристик системи критеріям стандарту ISO/IEC 25010

Критерій за ISO/IEC 25010	Опис критерію (згідно зі стандартом)	Архітектурна та технічна реалізація у застосунку
Безпека (Security)	Здатність захищати інформацію від несанкціонованого доступу чи модифікації зі збереженням доступу для легітимних користувачів.	Ізоляція бази даних від клієнтського середовища. Впровадження багаторівневої валідації вхідних даних (Zod) та екранування запитів на рівні Prisma ORM для захисту від SQL–ін'єкцій.
Зручність використання (Usability)	Ступінь, до якого продукт дозволяє користувачам досягати своїх цілей результативно, продуктивно та із задоволенням.	Відмова від синхронного перезавантаження сторінок на користь реактивного асинхронного інтерфейсу (SPA–підхід). Це суттєво знижує когнітивну втому оператора при роботі з великими масивами даних.
Зручність супроводу (Maintainability)	Ступінь ефективності, з яким систему можна модифікувати, оновлювати або адаптувати до нових вимог без руйнування існуючого функціоналу.	Жорстка логічна декомпозиція системи на незалежні модулі (Front–end та Back–end). Використання декларативного підходу до моделювання бази даних (Prisma Schema), що спрощує масштабування.

Підбиття підсумків оцінки параметрів надійності вимагає фокусування на стратегічному потенціалі обраної архітектурної моделі. Злиття жорсткої статичної типізації на рівні мови програмування та суворих реляційних обмежень

безпосередньо в ядрі бази даних формує подвійний контур ізоляції інформаційного середовища. Дані дають підстави обережно судити, що подібний підхід зводить до математичного мінімуму не лише ймовірність успіху цілеспрямованих кібератак. Він радикально купірує ризик каскадних відмов, спровокованих банальним людським фактором під час рутинної операційної діяльності фахівців.

Самостійним параметром життєздатності програмного комплексу виступає його природна схильність до горизонтального масштабування. Імплементація протоколу передачі репрезентативного стану вкупі з тотальною ізоляцією бізнес–логіки від візуального шару готує систему до різкого стрибка кількості підключених філіалів. Час відгуку інтерфейсу при цьому не зазнає деградації. Заявлена стабільність досягається шляхом цілеспрямованого делегування важких процесів агрегації даних на серверний рівень інструментарію Next.js, що негайно вивільняє апаратні ресурси клієнтських терміналів. Паралельно з цим оптимізація мережевого трафіку спирається на використання текстового формату обміну даними у зв'язці з асинхронними запитами. Цей механізм підтримує високу динаміку оновлення статусів підключень навіть за умов критичної деградації корпоративного зв'язку. Будь–яка майбутня модифікація структури збереження інформації не зруйнує архітектурну цілісність уже функціонуючих модулів.

Сформований методологічний базис об'єктивно виходить за тісні межі вирішення локального завдання простого інвентаризаційного обліку. Пропонується концепція стабільного, відмовостійкого та архітектурно прозорого інструменту. Він цілком відповідає жорстким галузевим стандартам фінансового сектору. Закладені на етапі теоретичного обґрунтування проєктні рішення акумулюють вичерпний масив технічних специфікацій. Вони виступають обов'язковим фундаментом для переходу до стадії безпосереднього написання програмного коду та фізичного розгортання компонентів системи.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБ–ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ІНФРАСТРУКТУРИ ТА ЛІМІТІВ

3.1. Вибір та обґрунтування засобів розробки

Проєктування програмних рішень для фінансового сектору та обліку корпоративної інфраструктури диктує жорсткі вимоги до формування технологічного стека. Відмова від ситуативного підбору популярних бібліотек на користь глибокого аналізу архітектурних вимог видається не просто бажаною, а фундаментально необхідною.

З огляду на це, базовою мовою реалізації всього проєкту обрано TypeScript. Цей інструмент формує наскрізну статичну типізацію середовища, зводячи до статистичної похибки ризик виникнення помилок під час виконання. Узгодженість моделей бази даних, об'єктів передачі даних на рівні програмного інтерфейсу та клієнтських компонентів перевіряється ще на етапі компіляції. Динамічний хаос, притаманний класичному JavaScript, унеможливлюється.

Вибудовування загальної архітектури веб–застосунку спирається на можливості фреймворку Next.js. Застосування нової парадигми App Router де–факто імплементує патерн делегування логіки на серверну сторону, залишаючи браузеру виключно завдання рендерингу підготовленого каркаса. Візуальний шар традиційно покладено на бібліотеку React. Компонентна ізоляція інтерфейсу суттєво пришвидшує ітеративне нарощування функціоналу.

Розробник оперує інкапсульованими сутностями замість складного глобального стану.

Зберігання транзакційної інформації, управління лімітами обороту та суворий облік робочих місць вимагає безкомпромісного підходу до цілісності даних. На етапі концептуального проєктування логічно постала дилема вибору між класичними реляційними та гнучкими документо–орієнтованими моделями. Для об'єктивізації фінального рішення було проведено порівняльне зіставлення систем PostgreSQL та MongoDB за низкою критичних критеріїв.

Результати цього аналізу структуровано у таб. 3.1.

Таблиця 3.1

Порівняльний аналіз СУБД для розробки застосунку

Критерій оцінки	PostgreSQL (Реляційна)	MongoDB (Нереляційна)
Модель даних	Строга реляційна схема табличного типу	Гнучка документо–орієнтована схема
Підтримка транзакцій	Повна відповідність вимогам ACID на всіх рівнях	Обмежена підтримка складних транзакційних операцій
Зв'язки між сутностями	Вбудовані зовнішні ключі з каскадним видаленням	Ручне управління зв'язками виключно на рівні коду
Цілісність даних	Висока (контролюється ядром СУБД)	Середня (залежить від якості імплементації в коді)
Доцільність для проєкту	Висока (релевантно для фінансового обліку)	Низька (надлишкова гнучкість, ризик втрати посилань)

Аналіз наведених метрик однозначно свідчить на користь реляційної моделі. Можна обережно стверджувати, що PostgreSQL ідеально закриває потреби предметної області. Наявність вбудованих зовнішніх ключів з механізмами каскадного оновлення робить неможливою появу «сирітських» записів у базі даних. Гнучкість MongoDB у цьому специфічному контексті розцінюється скоріше як архітектурна вразливість, ніж перевага.

Взаємодія ж між серверною бізнес–логікою та реляційною структурою абстрагується за допомогою інструментарію Prisma ORM. Декларативний підхід до опису схем та автоматична генерація типізованого клієнта нівелюють прірву між табличним та об'єктно–орієнтованим світами. Формування захисного периметра

системи логічно завершується імплементацією засобів валідації. Бібліотека Zod бере на себе обов'язок строгої перевірки вхідних потоків даних на межі програмного інтерфейсу. Будь–яка невідповідність очікуваному контракту призводить до негайного відхилення мережевого запиту. Водночас специфікацію самого API автоматизовано через стандарт OpenAPI, знімаючи левову частку рутини з розробника під час подальшого супроводу програмного продукту.

3.2. Проектування архітектури та алгоритмів програмного забезпечення

Формування топології розробленого програмного рішення базується на класичній трирівневій моделі. Такий підхід гарантує жорстку інкапсуляцію відповідальності між ізольованими компонентами системи. Верхній абстрактний рівень – шар представлення, реалізований засобами реактивної бібліотеки React, – відповідає виключно за рендеринг інтерфейсу та перехоплення користувацьких подій. Проміжним етапом виступає шар бізнес–логіки, побудований на базі маршрутизаторів Next.js API. Фундамент системи утворює шар доступу до даних, де абстракція взаємодії з фізичною структурою СУБД делегована інструментарію Prisma. З міркувань захисту внутрішнього периметра, пряма комунікація між клієнтом та базою даних фізично неможлива. Обмін інформаційними потоками здійснюється виключно через внутрішній проксі–маршрут серверної частини.

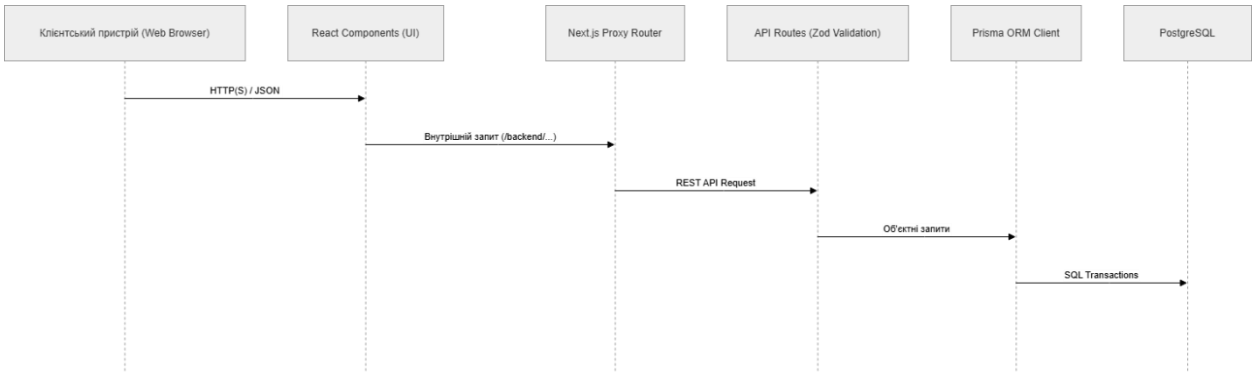

Рис. 3.1. UML–діаграма розгорнутої тришарової архітектури веб–застосунку

Синхронізація станів між клієнтом та сервером вимагає чіткого контракту. Тому комунікацію формалізовано через REST–орієнтований програмний інтерфейс. Проєктування кінцевих точок підпорядковано принципам семантичного використання HTTP–методів та ієрархічної маршрутизації. Специфікацію базових вузлів наведено у таб. 3.2.

Таблиця 3.2

Специфікація основних кінцевих точок (API Endpoints)

HTTP Метод	Маршрут (Endpoint)	Опис операції	Валідація об'єктів передачі даних (DTO)
GET	/api/branches	Отримання консолідованого списку відділень	Звичайна вибірка без складних предикатів
POST	/api/workplaces	Ініціалізація робочого місця AnyDesk	Перевірка цілісності anyDeskId та branchId
GET	/api/turnover–limits	Вибірка встановлених лімітів обороту	Динамічна фільтрація за branchId та activeOnly
DELETE	/api/branches/{id}	Каскадне видалення сутності відділення	Верифікація наявності ідентифікатора у БД

Інтеграція специфікації Swagger знімає необхідність ручного супроводу технічних документів. Система самостійно парсить кодові анотації та генерує візуальне представлення розроблених маршрутів.

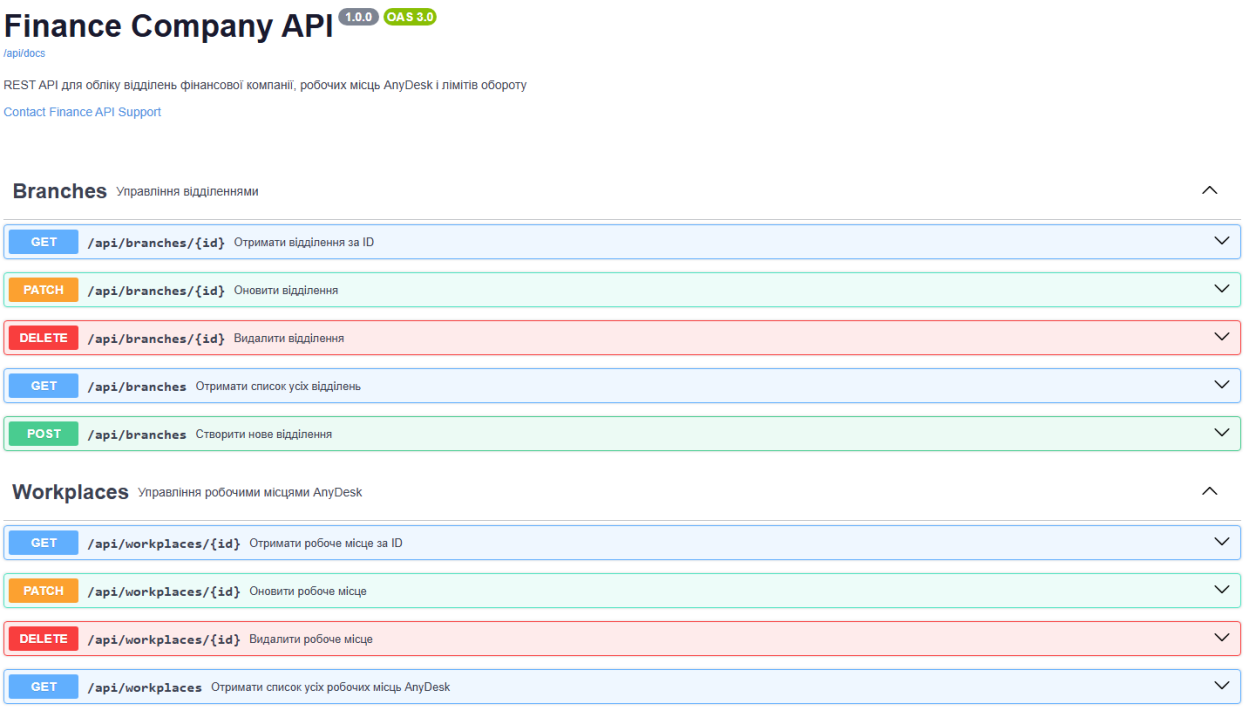

Рис. 3.2. Скриншот інтерфейсу Swagger UI з переліком маршрутів API

Переходячи до рівня збереження даних, варто зазначити, що реляційна схема виступає каркасом всього доменного моделювання. Логіка предметної області продиктувала необхідність виділення трьох ключових сутностей, пов'язаних суворими обмеженнями зовнішніх ключів. Найбільший інтерес з точки зору архітектури становить механізм зберігання лімітів обороту. Управління фінансовими обмеженнями вимагає врахування часового контексту. Фізичну структуру відповідної таблиці відображено у таб. 3.3.

Таблиця 3.3

Фізична структура таблиці TurnoverLimit

Назва поля	Тип даних (Prisma)	Обмеження та властивості	Опис призначення
id	Int	Primary Key, Autoincrement	Унікальний системний ідентифікатор запису
limitAmount	Float	Not Null, > 0	Кількісне значення фінансового ліміту
currency	String	Not Null, Default "UAH"	Стандартизований код валюти
effectiveFrom	DateTime	Not Null	Часова мітка початку дії ліміту
effectiveTo	DateTime	Nullable	Часова мітка завершення дії (безстроковий за умови Null)
branchId	Int	Foreign Key	Жорстке посилання на відділення–власника

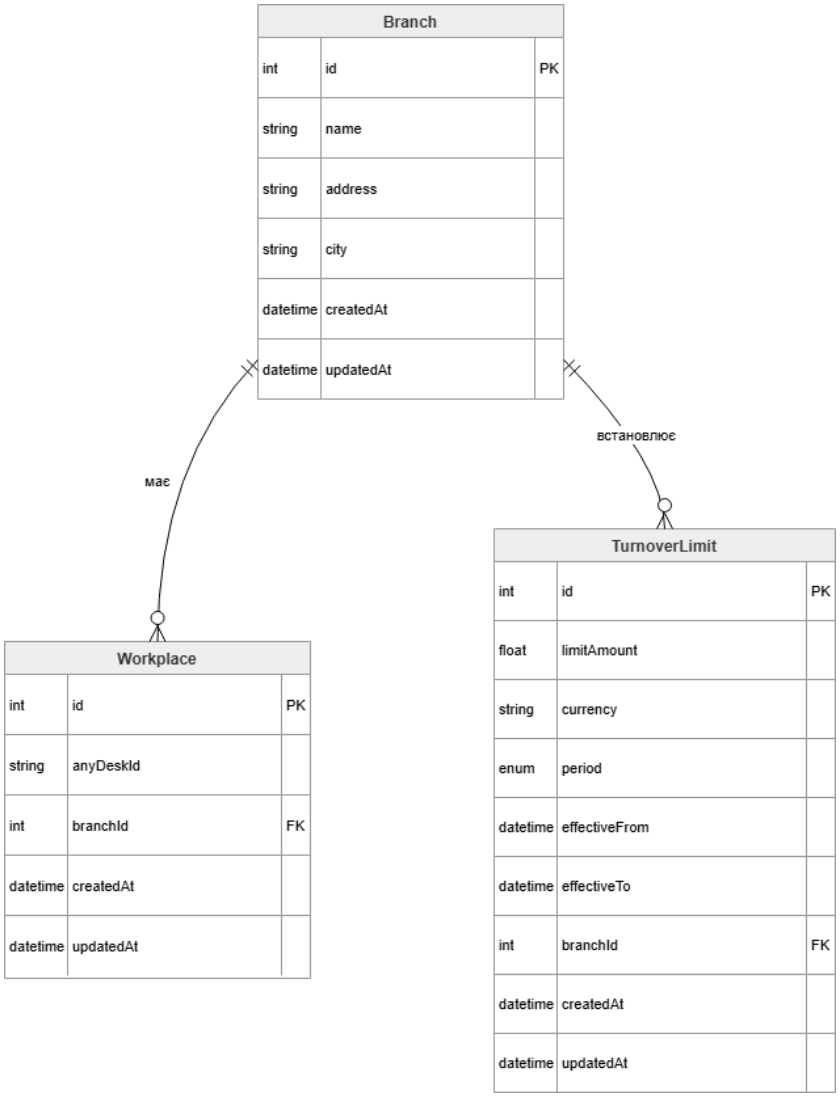

Рис.3.3. ER–діаграма сутностей предметної області бази даних

Програмна реалізація заявлених структур опирається на низку детермінованих алгоритмів. Зокрема, процес створення запису робочого місця AnyDesk жорстко лінієзований. Обробка вхідного POST–запиту починається з десеріалізації JSON–об'єкта. Далі потік управління передається декларативній схемі Zod. Вона формалізує контракт вводу та відтинає некоректні структури даних ще до моменту звернення до бази. Успішне проходження валідації ініціює перевірку консистентності: система запитує наявність вказаного батьківського відділення. Відсутність зв'язку провокує переривання транзакції з кодом помилки 404. Аналогічно обробляється конфлікт унікальності робочого ідентифікатора, генеруючи статус 409. Лише після проходження всіх бар'єрів виконується транзакція запису.

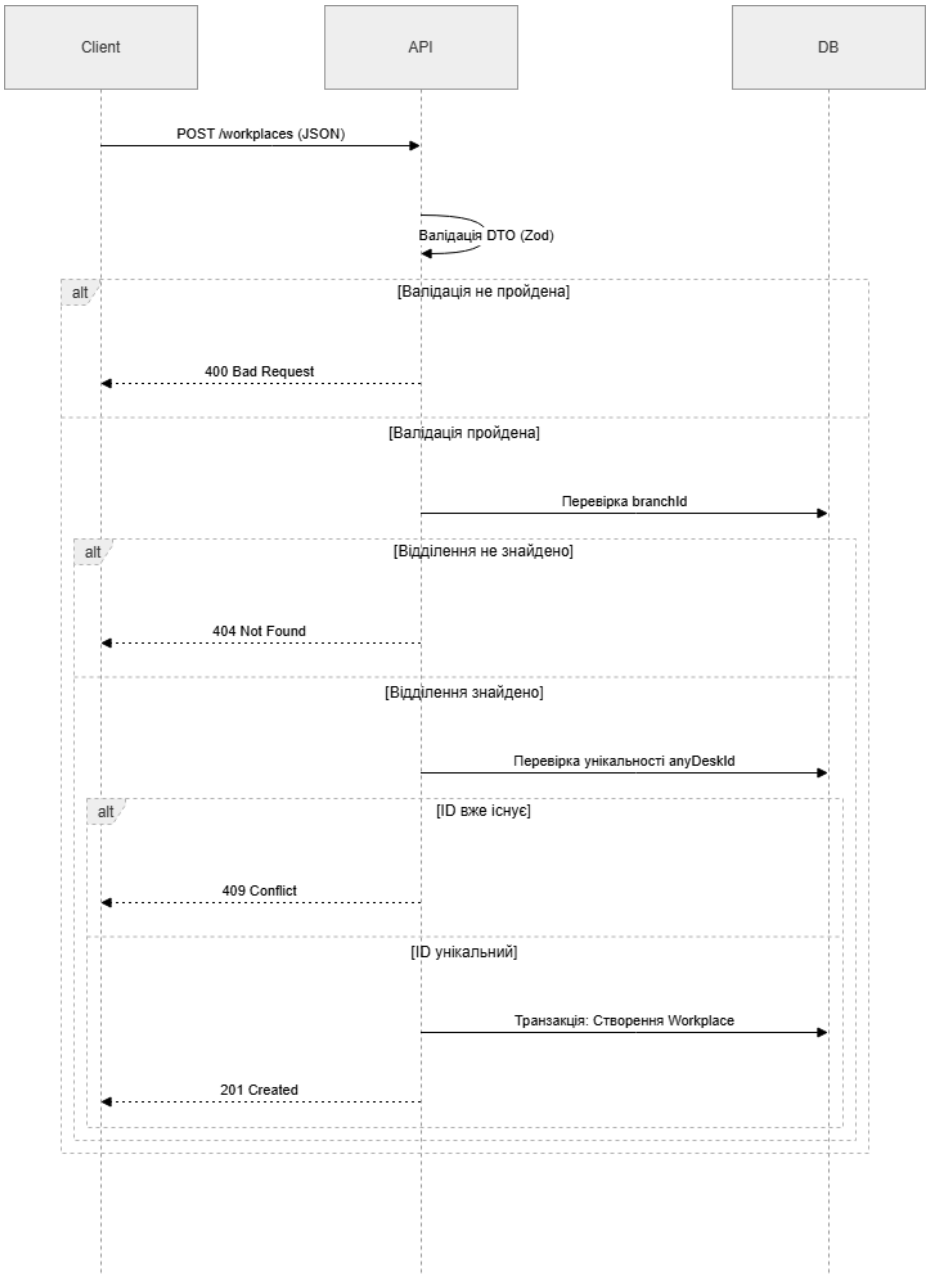

Рис. 3.4. Блок–схема алгоритму створення запису робочого місця AnyDesk

Інший критичний алгоритм відповідає за вибірку та фільтрацію діючих лімітів обороту. Операція спирається на комплексний часовий фільтр. Після нормалізації query–параметрів система оцінює стан прапорця activeOnly. Його активація змушує ядро СУБД шукати специфічний перетин інтервалів: дата початку дії ліміту (effectiveFrom) повинна передувати поточному системному часу, а дата завершення (effectiveTo) – залишатися невизначеною (Null) або належати майбутньому. Сформований набір даних підлягає обов'язковому сортуванню у

зворотному хронологічному порядку.

Абстрактну логіку роботи алгоритму динамічної фільтрації проілюстровано за допомогою блок–схеми (рис. 3.5). Безпосередній вихідний код програмної реалізації даного маршрутизатора винесено до Додатка Б (див. лістинг Б.1).

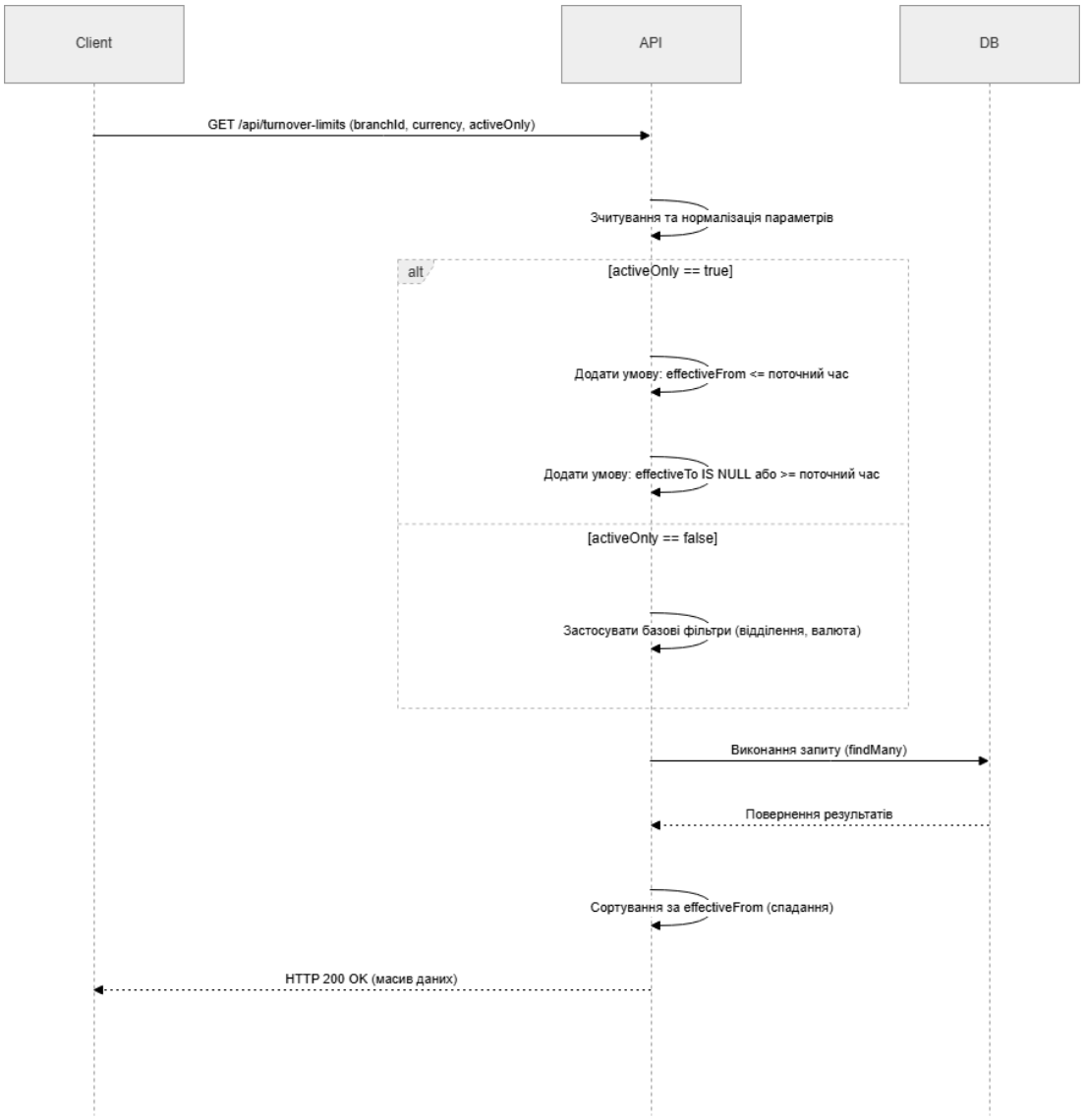

Рис. 3.5. Алгоритм динамічної фільтрації

Окремої уваги заслуговує підхід до формування агрегованих зведень для головної інформаційної панелі (Dashboard). Послідовні мережеві запити до різних таблиць неминуче призвели б до деградації часу відгуку та блокування інтерфейсу. Ця проблема вирішується імплементацією конкурентного завантаження. Паралельний виклик асинхронних операцій (через Promise.all) одночасно ініціює

вибірку відділень, робочих місць та активних лімітів. Агрегація ж отриманих масивів і розрахунок фінальних метрик перенесені на сторону клієнта, розвантажуючи обчислювальні потужності сервера.

Процес обміну повідомленнями між компонентами системи під час конкурентного завантаження агрегованих метрик формалізовано за допомогою UML–діаграми послідовності (див. рис. А.1 у Додатку А). Вихідний код клієнтського компонента інформаційної панелі, що реалізує цей алгоритм, наведено у Додатку Б (див. лістинг Б.2).

3.3. Практична реалізація, розгортання, тестування та методика використання

Практична імплементація розробленого програмного комплексу вимагає жорсткої регламентації середовища виконання. Апаратний та програмний базис спирається на використання кросплатформних операційних систем у поєднанні з середовищем Node.js та екземпляром PostgreSQL. Ініціалізація системи починається з ізольованого конфігурування серверного та клієнтського вузлів. Критично важливі параметри, зокрема реквізити доступу до бази даних та адреси внутрішніх маршрутизаторів, винесено у зовнішні файли конфігурації. Такий підхід виключає компрометацію чутливих даних через системи контролю версій. Перед фактичним запуском бізнес–логіки виконується примусова синхронізація об'єктної моделі Prisma з фізичною структурою реляційної СУБД. Цей процес охоплює валідацію синтаксису схем, компіляцію SQL–міграцій та генерацію типізованого клієнта доступу до даних. Опціонально передбачено механізм автоматизованого наповнення таблиць тестовими зразками, що суттєво спрощує первинне розгортання на нових потужностях.

Розглядаючи питання інформаційної безпеки, варто наголосити на багаторівневому підході до фільтрації вхідних потоків. Проблема класичних SQL–ін'єкцій повністю усувається на рівні ORM–системи. Ядро Prisma автоматично екранує всі параметри перед формуванням кінцевого запиту. Додатковим бар'єром

виступає інструментарій Zod, що відхиляє будь–які аномальні структури даних ще на підступах до серверної логіки. Спроби передати текстові символи замість числових лімітів або порушити формат дат ігноруються системою. Разом з тим архітектурне рішення з використанням внутрішнього проксі–маршрутизатора Next.js повністю ізолює реальну топологію API від зовнішнього спостерігача. Клієнтський браузер комунікує виключно з проміжним вузлом.

Верифікація заявленого функціоналу спиралася на двохетапну методику. Первинному ізольованому тестуванню піддалися кінцеві точки серверної частини з використанням згенерованих специфікацій Swagger. Було відпрацьовано як позитивні сценарії збереження даних, так і реакції системи на свідомо невалідні параметри.

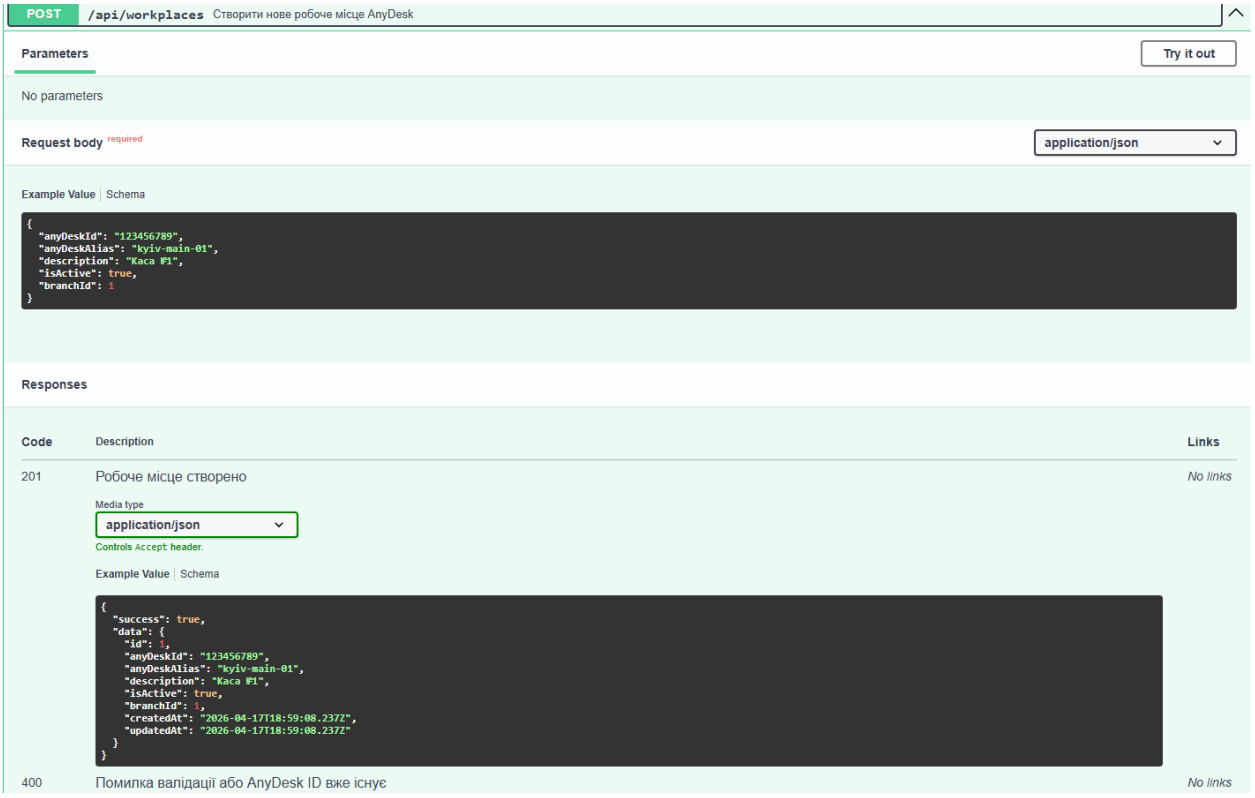

Рис. 3.6. Скриншот інтерфейсу Swagger UI під час виконання тестового запиту POST /api/workplaces

Подальший етап експериментальної перевірки передбачав комплексне функціональне випробування візуального інтерфейсу.

Головним завданням постало підтвердження стабільності бізнес–логіки під час імітації користувацьких сценаріїв. Тестування базового механізму ініціалізації нових філіалів засвідчило коректне збереження валідних наборів даних із відповідною генерацією коду успішного створення. Оновлена інформація миттєво інтегрується у табличний простір екрана. Водночас спроби дублювання унікальних ідентифікаторів терміналів AnyDesk провокують негайне блокування транзакцій.

Ядро системи перехоплює конфлікт станів та транслює помилку через модальне вікно, алгоритмічно унеможливлюючи запис дублікатів. Окрему увагу було приділено перевірці стійкості реляційних зв'язків. Імітація прив'язки апаратного вузла до неіснуючого материнського відділення очікувано завершується відхиленням мережевого запиту на рівні програмного інтерфейсу. Поява незв'язних об'єктів у базі даних фізично виключена.

Наступний блок випробувань охоплював механізми обробки фінансових та часових метрик. Перевірка алгоритмів динамічної фільтрації лімітів обороту підтвердила синхронне відпрацювання логіки на боці клієнта та сервера. Активація критерію актуальності безпомилково відтинає з візуального представлення архівні записи, де фактичний час перевищує встановлену межу дії обмеження. Сама ж процедура створення фінансових лімітів продемонструвала заявлену надійність до аномального вводу. Спроможність системи превентивно блокувати від'ємні суми реалізована дзеркально: через деактивацію візуальних елементів управління та через жорстке відхилення прямих запитів серверним ядром.

Завершальним етапом стала верифікація каскадного видалення. Деструктивний вплив на сутність відділення коректно поширюється на всю ієрархію підпорядкованих об'єктів, автоматично вичищаючи пов'язані робочі місця та ліміти. Вичерпний регламент розгортання програмного комплексу для операторів, а також деталізований протокол проведених випробувань із покроковою фіксацією результатів винесено у Додаток В.

Для ілюстрації механізмів обробки негативних сценаріїв, зокрема порушення унікальності (тест №2), зафіксовано реакцію системи на рівні візуального інтерфейсу.

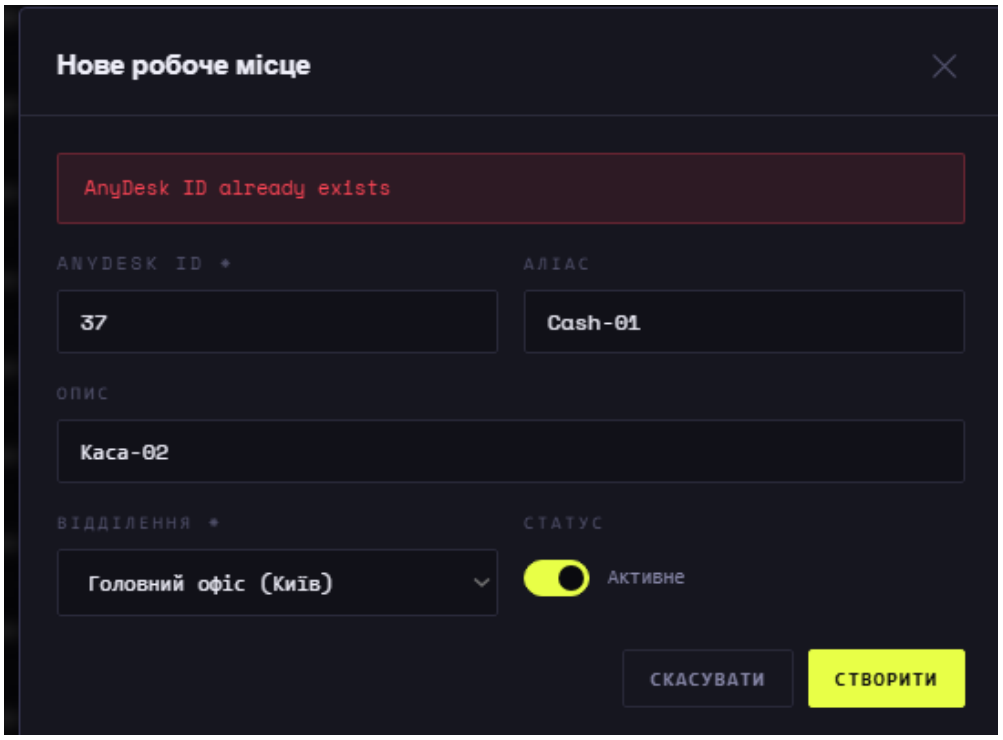

Рис. 3.7. Скриншот модального вікна з помилкою валідації унікальності AnyDesk ID

Опис експлуатаційних характеристик логічно завершує цикл впровадження програмного продукту. Стандартна модель взаємодії оператора із системою починається з головної інформаційної панелі. Після завантаження сторінки ініціюється конкурентний запит, внаслідок якого на екран виводяться агреговані показники стану інфраструктури: загальна кількість активних філіалів, підключених терміналів та налаштованих лімітів. У разі тимчасової відсутності зв'язку з базою даних спрацьовує механізм безпечного відображення помилок, блокуючи руйнування візуального каркаса.

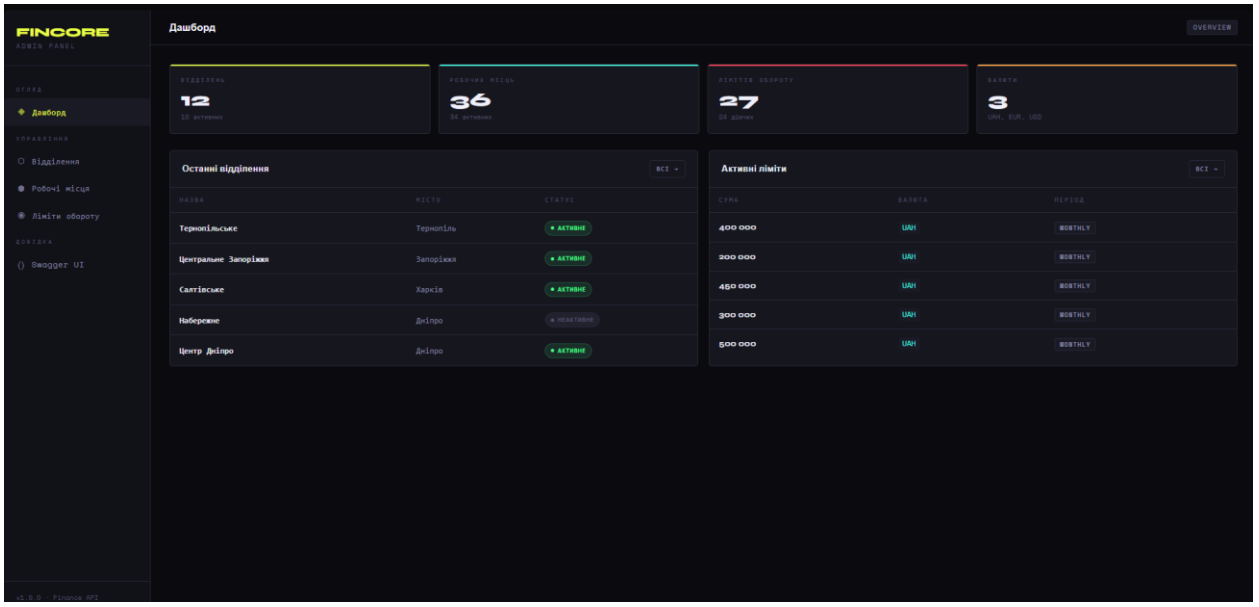

Рис. 3.8. Головна сторінка інформаційної панелі з показниками

Управління топологією мережі та фінансовими обмеженнями здійснюється через відповідні модулі бокової навігації. Реєстрація нових відділень або закріплення ідентифікаторів робочих станцій ініціюється у спеціалізованому обліковому просторі. Візуальне представлення консолідованого переліку підключених терміналів наведено на рисунку 3.9.

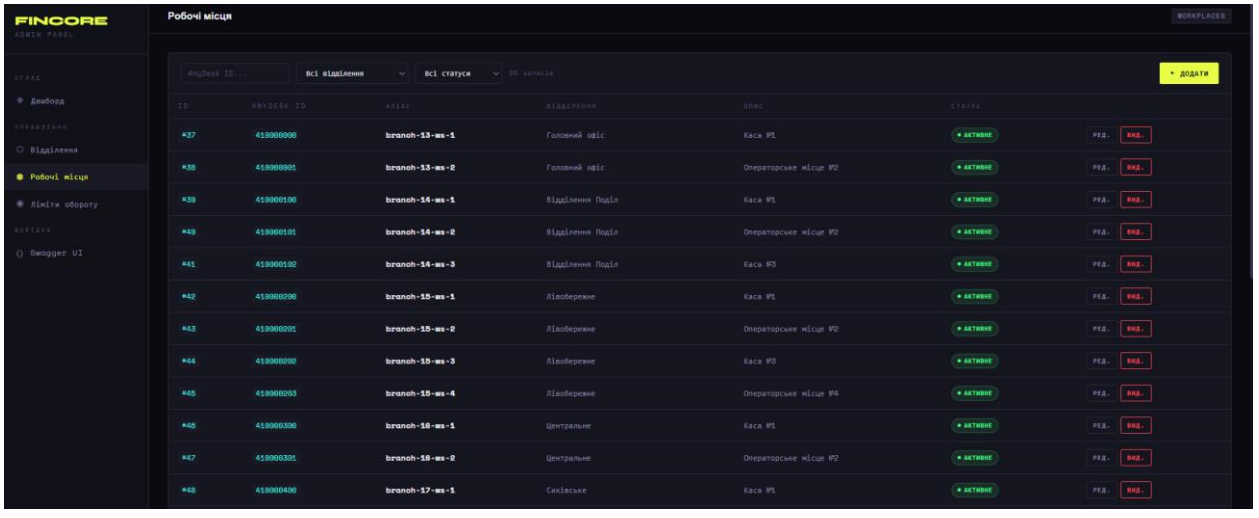

Рис. 3.9. Інтерфейс таблиці "Робочі місця"

Безпосереднє введення нових параметрів інфраструктури реалізовано через виклик інтерактивної модальної форми. Саме на цьому етапі імплементовано механізми жорсткої клієнтської валідації. Алгоритми фронтенду відстежують дії

оператора в режимі реального часу, миттєво ідентифікують синтаксичні аномалії, візуально маркують некоректні поля та фізично блокують трансляцію хибних пакетів до серверної частини (рис. 3.10).

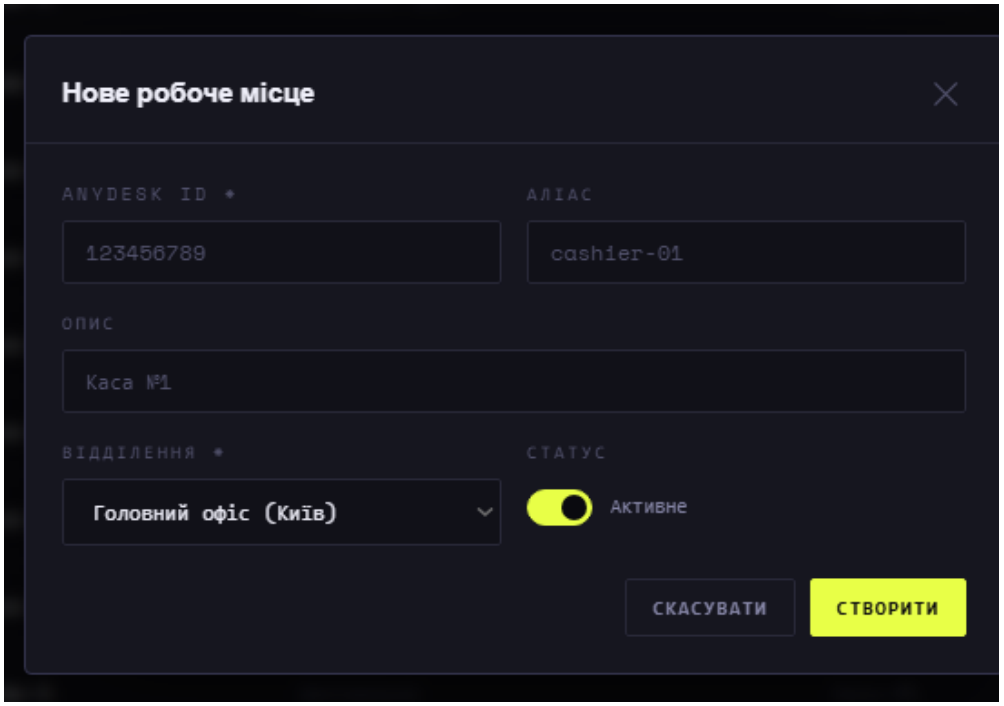

Рис. 3.10. Інтерфейс модальної форми додавання в таблиці "Робочі місця"

Переходячи до фінансового блоку, варто зазначити, що процес встановлення лімітів обороту також суворо регламентований. Оператор має змогу оперувати виключно додатними числовими значеннями. Програмно передбачено можливість створення безстрокових лімітів – для цього достатньо залишити часову мітку завершення дії порожньою. Вбудовані механізми динамічної фільтрації безпосередньо у табличному представленні зводять до мінімуму час пошуку актуальних обмежень серед масиву архівних даних. Можна обережно стверджувати, що спроєктований інтерфейс мінімізує когнітивне навантаження на оператора, повністю делегуючи контроль за цілісністю даних алгоритмам валідації.

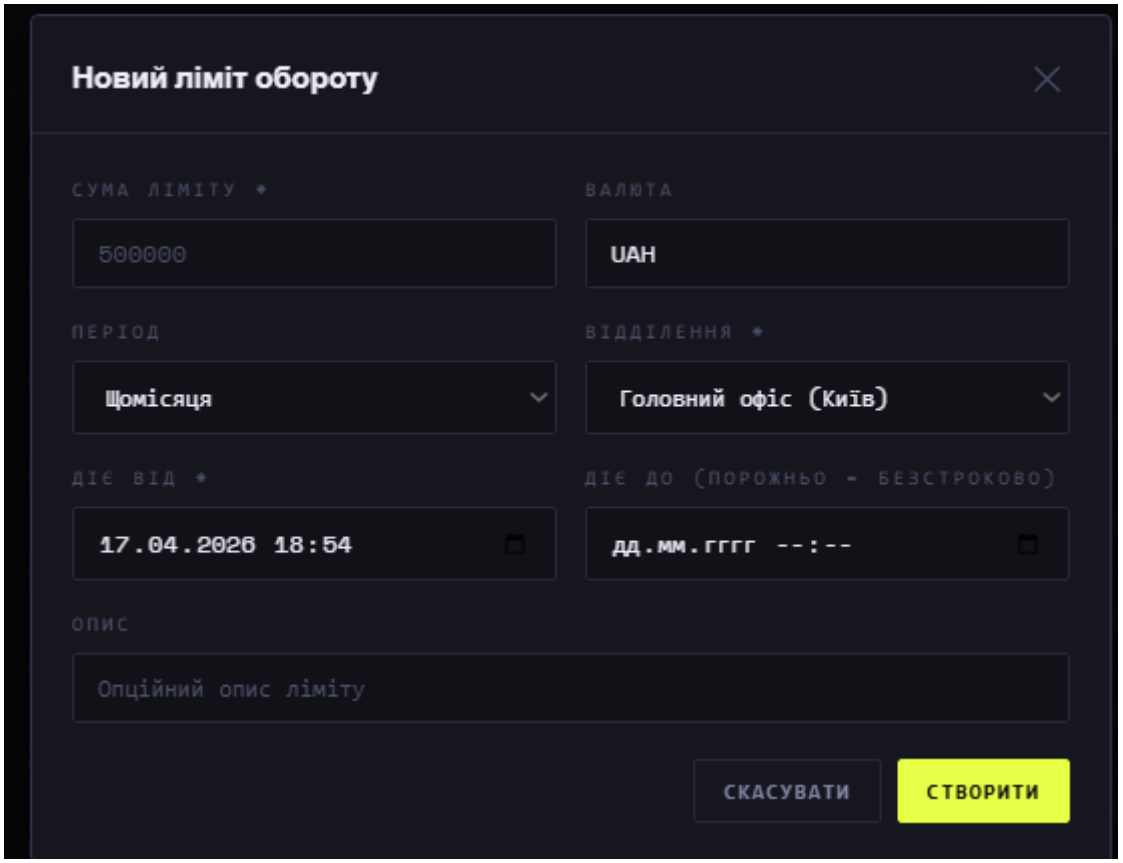

Рис. 3.11. Форма створення та редагування ліміту обороту з вибором дат

Комплексний огляд етапів розгортання та експлуатаційного тестування дає обґрунтовані підстави констатувати досягнення первинних інженерних цілей. Закладені на стадії проєктування архітектурні патерни успішно матеріалізувалися у фізичній реалізації коду. Симбіоз жорстких захисних механізмів серверної частини з проактивною клієнтською валідацією сформував стійку до зовнішніх аномалій інформаційну систему. Відсутність критичних збоїв під час проходження протоколів тестування, помножена на інтуїтивно зрозумілу модель взаємодії, вказує на логічне завершення циклу розробки. Спроєктований веб–застосунок повністю підготовлений до переходу на етап дослідної експлуатації в умовах реального фінансового документообігу.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково–практичне завдання: розроблено спеціалізований веб–застосунок для централізованого обліку відділень фінансової компанії, робочих місць AnyDesk та лімітів обороту. Основні висновки дослідження наведено нижче.

1. Аналіз предметної області показав, що традиційні підходи до обліку інфраструктури фінансових компаній, такі як електронні таблиці, локальні бази даних і розрізнена документація, є неефективними. Вони спричиняють дублювання даних, втрату інформації та значні часові витрати персоналу. Аналіз існуючих програмних рішень (ERP, CRM, системи віддаленого доступу) підтвердив відсутність комплексного інструменту, який би одночасно забезпечував облік відділень, управління робочими місцями AnyDesk і контроль лімітів обороту в одному інтерфейсі.

2. Визначено повний перелік функціональних і нефункціональних вимог до інформаційної системи. Основні функціональні вимоги включають ведення реєстру відділень, збереження AnyDesk–ідентифікаторів, управління часовими лімітами фінансового обороту, а також пошук і фільтрацію даних. До нефункціональних вимог належать надійність, безпека, масштабованість, кросплатформеність і зручність інтерфейсу.

3. Обґрунтовано архітектурні рішення системи на основі клієнт–серверної моделі. Використання гібридного фреймворку Next.js з App Router дозволило перенести обчислювальне навантаження на сервер, що забезпечило високу швидкість рендерингу інтерфейсу. Вибір реляційної СУБД PostgreSQL і Prisma ORM зумовлений необхідністю підтримки ACID–транзакцій і суворої реляційної цілісності даних між сутностями відділень, робочих місць і лімітів.

4. Спроєктовано та реалізовано трирівневу архітектуру застосунку з чіткою декомпозицією на шар представлення (React/Next.js), бізнес–логіки (REST API) та доступу до даних (Prisma ORM + PostgreSQL). Така модель унеможливлює пряму комунікацію між клієнтським браузером і базою даних, що значно підвищує інформаційну безпеку системи.

5. Розроблено та впроваджено ключові алгоритми системи: динамічну фільтрацію лімітів обороту за часовим контекстом із підтримкою безстрокових записів (effectiveTo = null), конкурентне завантаження агрегованих метрик для інформаційної панелі через Promise.all, а також каскадне видалення пов'язаних сутностей.

6. Впроваджено багаторівневу систему захисту даних відповідно до стандарту ISO/IEC 25010. Валідація вхідних потоків за допомогою бібліотеки Zod і автоматичне параметризування SQL–запитів через Prisma Client нейтралізують загрозу ін'єкційних атак. Наскрізна статична типізація TypeScript дозволяє виявляти до 15% потенційних дефектів на етапі компіляції, що підвищує відмовостійкість системи.

7. Проведено комплексне функціональне тестування застосунку за двоетапною методикою: ізольована верифікація кінцевих точок API через Swagger UI та інтеграційне тестування користувацьких сценаріїв. Результати підтвердили коректність роботи всіх реалізованих механізмів: збереження й оновлення записів, контроль унікальності AnyDesk–ідентифікаторів, захист від некоректних фінансових значень, динамічну фільтрацію та каскадне видалення пов'язаних сутностей. Практичне значення програмного продукту полягає в можливості його безпосереднього впровадження у фінансових компаніях, мікрофінансових організаціях та кредитних спілках із розгалуженою мережею відділень. Застосування системи дозволяє суттєво скоротити час на пошук та оновлення інформації про інфраструктуру, забезпечити оперативний контроль відповідності поточних оборотів встановленим лімітам та мінімізувати ризики, пов'язані з некоректним веденням обліку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Avula S. B., Kavuluri H. V. R. Database Engineering for Microservices: PostgreSQL as the Foundation. *International Journal of Artificial Intelligence, Data Science, and Machine Learning.* 2024. Vol. 5, Issue 4. P. 20–28.

2. Kotstein S., Bogner J. Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts. *arXiv preprint arXiv:2108.00033.* 2021. 20 p.

3. Pati S., Zaki Y. Evaluating the Efficacy of Next.js: A Comparative Analysis with React.js on Performance, SEO, and Global Network Equity. *Proceedings of the ACM Web Conference 2025 (WWW '25).* 2025. P. 1–4.

4. ISO/IEC 25010. Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models : веб–сайт. URL: https://iso25000.com/ (дата звернення: 25.04.2026).

5. Blahodelskyi O. S. The Role of Typescript in Scalability and Maintainability of Frontend Code. *International Journal of Scientific Engineering and Science.* 2024. Vol. 8, Issue 8. P. 17–22.

6. Веб–застосунок для обліку відділень фінансової компанії, робочих місць anydesk та лімітів обороту// Матеріали VII Всеукраїнської науково–технічної конференції «Застосування програмного забезпечення в інформаційно–комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

7. Веб–застосунок для ведення обліку відділень фінансової компанії, робочих місць AnyDesk та контролю лімітів обороту. // Матеріали VII Всеукраїнської науково–технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026

8. Prisma ORM Documentation. Data Modeling & Schema Design : веб–сайт. URL: https://www.prisma.io/docs (дата звернення: 20.04.2026).

9. Next.js Official Documentation. App Router Architecture : веб–сайт. URL: https://nextjs.org/docs (дата звернення: 20.04.2026).

10. Fielding R. T. Architectural Styles and the Design of Network–based Software Architectures : дис. ... д–ра філос. / University of California, Irvine. 2000. 180

p.

11. PostgreSQL Global Development Group. PostgreSQL 16 Documentation : веб–сайт. URL: https://www.postgresql.org/docs/16/ (дата звернення: 18.04.2026).

12. Colvin C., Hossain M. A. ACID Properties in Modern Relational Database Systems. *Journal of Database Management.* 2023. Vol. 34, Issue 2. P. 45–61.

13. TypeScript Documentation. TypeScript Handbook : веб–сайт. URL: https://www.typescriptlang.org/docs/ (дата звернення: 19.04.2026).

14. OpenAPI Specification 3.1.0. Swagger / OpenAPI Initiative : веб–сайт. URL: https://swagger.io/specification/ (дата звернення: 21.04.2026).

15. Coloma T., Ramos F. Zod: TypeScript–First Schema Validation with Static Type Inference. *Journal of Open Source Software.* 2023. Vol. 8, Issue 87. P. 1–5.

16. React Documentation. React 18 Overview : веб–сайт. URL: https://react.dev/blog/2022/03/29/react–v18 (дата звернення: 20.04.2026).

17. Wieruch R. *The Road to React.* 2023 Edition. Leanpub. 2023. 320 p.

18. Rowe D. C. Financial Technology and Compliance: Managing Risk in Distributed Branch Networks. *Journal of Financial Regulation.* 2022. Vol. 8, Issue 1. P. 112–134.

19. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object–Oriented Software.* Addison–Wesley Professional. 1994. 395 p.

20. Richardson L., Ruby S. *RESTful Web Services.* O'Reilly Media. 2007. 446 p.

21. Fowler M. *Patterns of Enterprise Application Architecture.* Addison–Wesley Professional. 2002. 533 p.

22. OWASP Foundation. OWASP Top 10 – 2021: The Ten Most Critical Web Application Security Risks : веб–сайт. URL: https://owasp.org/Top10/ (дата звернення: 22.04.2026).

23. Kleppmann M. *Designing Data–Intensive Applications.* O'Reilly Media. 2017. 611 p.

24. Molyneaux I. *The Art of Application Performance Testing.* 2nd ed. O'Reilly Media. 2014. 162 p.

25. Newman S. *Building Microservices: Designing Fine–Grained Systems.* 2nd ed. O'Reilly Media. 2021. 616 p.

26. Hunt A., Thomas D. *The Pragmatic Programmer: Your Journey to Mastery.* 20th Anniversary Edition. Addison–Wesley Professional. 2019. 352 p.

27. Vercel Inc. Edge Runtime and Serverless Functions in Next.js : веб–сайт. URL: https://vercel.com/docs/functions (дата звернення: 21.04.2026).

28. Национальна комісія з цінних паперів та фондового ринку. Вимоги до інформаційних систем фінансових установ : офіційний веб–портал. URL: https://www.nssmc.gov.ua (дата звернення: 15.04.2026).

29. Закон України «Про фінансові послуги та державне регулювання ринків фінансових послуг» від 12.07.2001 № 2664–III. *Відомості Верховної Ради України.* 2002. № 1. Ст. 1.

30. Connolly T., Begg C. *Database Systems: A Practical Approach to Design, Implementation, and Management.* 6th ed. Pearson Education. 2014. 1440 p.

31. Elmasri R., Navathe S. B. *Fundamentals of Database Systems.* 7th ed. Pearson Education. 2015. 1280 p.

32. Silva C., Pereira J. Security Considerations for Financial Web Applications: An Empirical Study. *Computers & Security.* 2023. Vol. 127. P. 103–118.

33. Date C. J. *An Introduction to Database Systems.* 8th ed. Addison–Wesley. 2003. 1024 p.

34. Pressman R. S., Maxim B. R. *Software Engineering: A Practitioner's Approach.* 9th ed. McGraw–Hill Education. 2019. 976 p.

35. Sommerville I. *Software Engineering.* 10th ed. Pearson Education. 2015. 816 p.

36. Bass L., Clements P., Kazman R. *Software Architecture in Practice.* 4th ed. Addison–Wesley Professional. 2021. 464 p.

37. Booch G., Rumbaugh J., Jacobson I. *The Unified Modeling Language User Guide.* 2nd ed. Addison–Wesley Professional. 2005. 496 p.

38. AnyDesk Software GmbH. AnyDesk Remote Desktop Security Whitepaper : веб–сайт. URL: https://anydesk.com/en/whitepapers (дата звернення: 17.04.2026).

39. ISO/IEC 27001:2022. Information Security Management Systems – Requirements : веб–сайт. URL: https://www.iso.org/standard/27001 (дата звернення: 24.04.2026).

40. Tanenbaum A. S., Van Steen M. *Distributed Systems: Principles and Paradigms.* 3rd ed. Pearson Education. 2017. 704 p.

41. Chacon S., Straub B. *Pro Git.* 2nd ed. Apress. 2014. 574 p. URL: https://git–scm.com/book/en/v2 (дата звернення: 19.04.2026).

Додаток А

UML–діаграми взаємодії компонентів системи

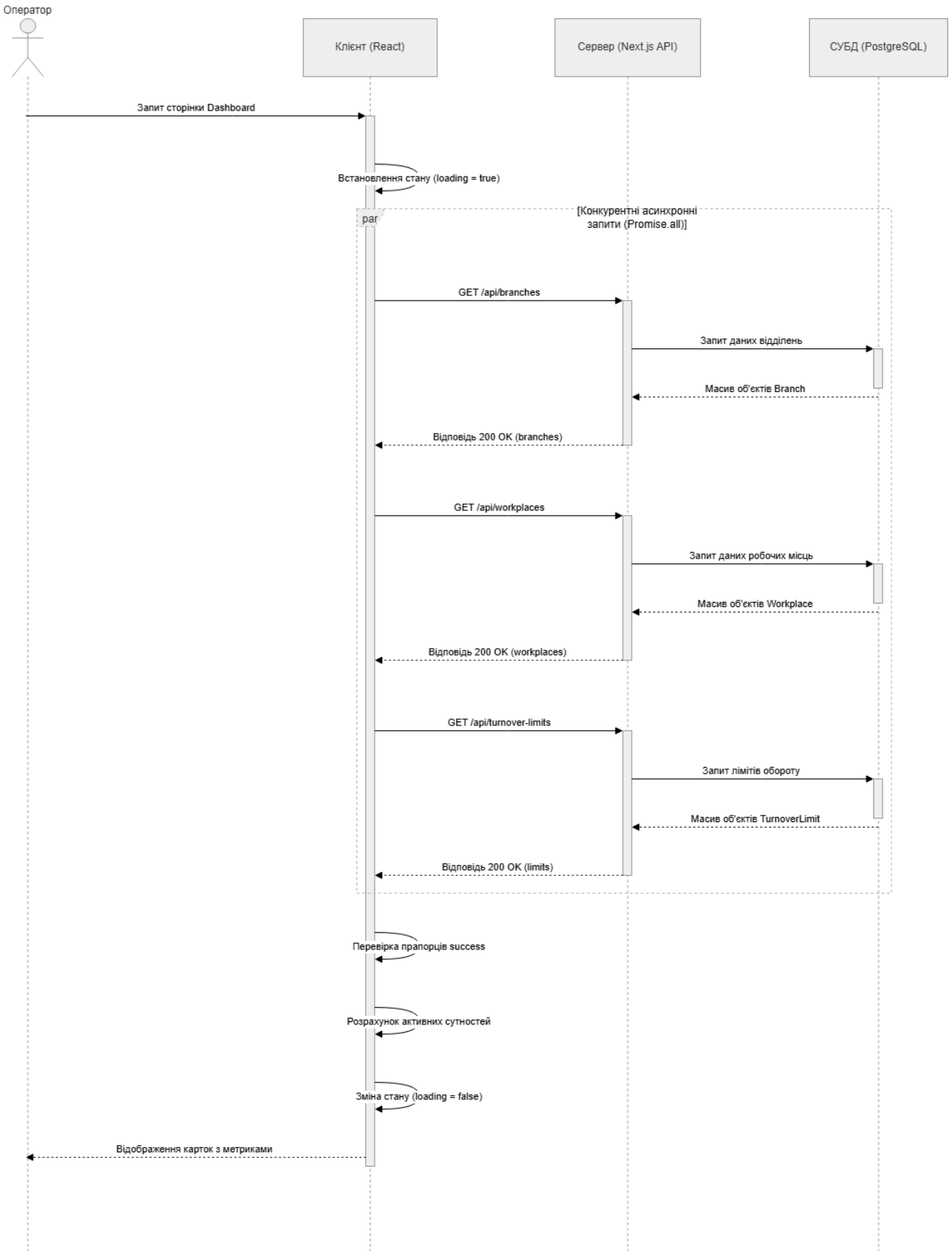

Рис. А.1 Діаграма послідовності (Dashboard)

Додаток Б

Фрагменти вихідного коду розробленого веб–застосунку

Лістинг Б.1. Вихідний код маршрутизатора обробки лімітів обороту (файл route.ts)

```
import { NextRequest } from "next/server";
import { prisma } from "@/lib/prisma";
import { successResponse, errorResponse } from "@/lib/api–response";
import { createTurnoverLimitSchema } from "@/lib/schemas";

/**
 * @swagger
 * /api/turnover–limits:
 * get:
 * summary: Отримати список усіх лімітів обороту
 * tags: [Turnover Limits]
 * parameters:
 * – in: query
 * name: branchId
 * schema:
 * type: integer
 * description: Фільтр по відділенню
 * – in: query
 * name: currency
 * schema:
 * type: string
 * description: Фільтр по валюті (UAH, USD, EUR)
 * – in: query
 * name: period
 * schema:
 * type: string
 * enum: [DAILY, WEEKLY, MONTHLY, QUARTERLY, YEARLY]
 * description: Фільтр по періоду
```

```
 * – in: query
 * name: activeOnly
 * schema:
 * type: boolean
 * description: Тільки активні ліміти (effectiveTo is null або в майбутньому)
 * responses:
 * 200:
 * description: Список лімітів обороту
 * content:
 * application/json:
 * schema:
 * allOf:
 * – $ref: '#/components/schemas/ApiSuccess'
 * – type: object
 * properties:
 * data:
 * type: array
 * items:
 * $ref: '#/components/schemas/TurnoverLimit'
 */
export async function GET(req: NextRequest) {
  try {
    const { searchParams } = req.nextUrl;
    const branchIdParam = searchParams.get("branchId");
    const currency = searchParams.get("currency");
    const period = searchParams.get("period");
    const activeOnly = searchParams.get("activeOnly") === "true";

    const now = new Date();
```

```
    const limits = await prisma.turnoverLimit.findMany({
      where: {
        ...(branchIdParam ? { branchId: parseInt(branchIdParam) } : {}),
        ...(currency ? { currency: currency.toUpperCase() } : {}),
        ...(period ? { period: period as any } : {}),
        ...(activeOnly
          ? {
              effectiveFrom: { lte: now },
              OR: [{ effectiveTo: null }, { effectiveTo: { gte: now } }],
            }
          : {}),
      },
      include: { branch: true },
      orderBy: { effectiveFrom: "desc" },
    });

    return successResponse(limits);
  } catch (error) {
    console.error("[GET /api/turnover–limits]", error);
    return errorResponse("Failed to fetch turnover limits", 500);
  }
}

/**
 * @swagger
 * /api/turnover–limits:
 * post:
 * summary: Створити новий ліміт обороту
 * tags: [Turnover Limits]
 * requestBody:
```

```
         * required: true
         * content:
         * application/json:
         * schema:
         * $ref: '#/components/schemas/CreateTurnoverLimit'
         * responses:
         * 201:
         * description: Ліміт обороту створено
         * content:
         * application/json:
         * schema:
         * allOf:
         * – $ref: '#/components/schemas/ApiSuccess'
         * – type: object
         * properties:
         * data:
         * $ref: '#/components/schemas/TurnoverLimit'
         * 400:
         * description: Помилка валідації
         */
        export async function POST(req: NextRequest) {
          try {
            const body = await req.json();
            const parsed = createTurnoverLimitSchema.safeParse(body);

            if (!parsed.success) {
              return errorResponse("Validation error", 400,
parsed.error.flatten().fieldErrors);
            }
```

```
    const branch = await prisma.branch.findUnique({ where: { id:
parsed.data.branchId } });
    if (!branch) return errorResponse("Branch not found", 404);

    const { limitAmount, effectiveFrom, effectiveTo, ...rest } = parsed.data;

    const limit = await prisma.turnoverLimit.create({
      data: {
        ...rest,
        limitAmount: limitAmount,
        effectiveFrom: new Date(effectiveFrom),
        ...(effectiveTo ? { effectiveTo: new Date(effectiveTo) } : {}),
      },
      include: { branch: true },
    });

    return successResponse(limit, 201);
  } catch (error) {
    console.error("[POST /api/turnover–limits]", error);
    return errorResponse("Failed to create turnover limit", 500);
  }
}
```

Лістинг Б.2. Вихідний код клієнтського компонента інформаційної панелі (файл page.tsx)

```
"use client";

import { useEffect, useState } from "react";
import Link from "next/link";
import { branchesApi, workplacesApi, turnoverLimitsApi } from "@/lib/api";
```

```
import { Branch, Workplace, TurnoverLimit } from "@/lib/types";

export default function DashboardPage() {
  const [branches, setBranches] = useState<Branch[]>([]);
  const [workplaces, setWorkplaces] = useState<Workplace[]>([]);
  const [limits, setLimits] = useState<TurnoverLimit[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    Promise.all([
      branchesApi.list(),
      workplacesApi.list(),
      turnoverLimitsApi.list(),
    ]).then(([b, w, l]) => {
      if (b.success) setBranches(b.data);
      if (w.success) setWorkplaces(w.data);
      if (l.success) setLimits(l.data);
    }).finally(() => setLoading(false));
  }, []);

  const activeBranches = branches.filter((b) => b.isActive).length;
  const activeWorkplaces = workplaces.filter((w) => w.isActive).length;
  const activeLimits = limits.filter(
    (l) => !l.effectiveTo || new Date(l.effectiveTo) > new Date()
  ).length;

  const recentBranches = branches.slice(–5).reverse();

  return (
    <>
```

```
<div className="topbar">
  <span className="topbar–title">Дашборд</span>
  <span className="topbar–badge">OVERVIEW</span>
</div>

<div className="content">
  {loading ? (
    <div className="loading–state">
      <div className="spinner" />
      Завантаження...
    </div>
  ) : (
    <>
      <div className="stats–grid">
        <div className="stat–card">
          <div className="stat–label">Відділень</div>
          <div className="stat–value">{branches.length}</div>
          <div className="stat–sub">{activeBranches} активних</div>
        </div>
        <div className="stat–card accent–2">
          <div className="stat–label">Робочих місць</div>
          <div className="stat–value">{workplaces.length}</div>
          <div className="stat–sub">{activeWorkplaces} активних</div>
        </div>
        <div className="stat–card accent–3">
          <div className="stat–label">Лімітів обороту</div>
          <div className="stat–value">{limits.length}</div>
          <div className="stat–sub">{activeLimits} діючих</div>
        </div>
        <div className="stat–card accent–4">
```

```
            <div className="stat–label">Валюти</div>
            <div className="stat–value">
              {new Set(limits.map((l) => l.currency)).size}
            </div>
            <div className="stat–sub">
              {[...new Set(limits.map((l) => l.currency))].join(", ") || "–"}
            </div>
          </div>
        </div>

        <div style={{ display: "grid", gridTemplateColumns: "1fr 1fr", gap: 16
}}>
          <div className="card">
            <div className="card–header">
              <span className="card–title">Останні відділення</span>
              <Link href="/branches" className="btn btn–ghost btn–sm">
                Всі →
              </Link>
            </div>
            {recentBranches.length === 0 ? (
              <div className="empty–state">
                <div className="empty–state–icon">⬡</div>
                <div className="empty–state–text">Відділень ще немає</div>
              </div>
            ) : (
              <div className="table–wrap">
                <table>
                  <thead>
                    <tr>
                      <th>Назва</th>
```

```
                <th>Місто</th>
                <th>Статус</th>
              </tr>
            </thead>
            <tbody>
              {recentBranches.map((b) => (
                <tr key={b.id}>
                  <td>
                    <Link href={`/branches/${b.id}`} style={{ color: "var(—
text)", fontWeight: 700 }}>
                      {b.name}
                    </Link>
                  </td>
                  <td>{b.city}</td>
                  <td>
                    <span className={`badge ${b.isActive ? "badge–active" :
"badge–inactive"}`}>
                      <span className="badge–dot" />
                      {b.isActive ? "Активне" : "Неактивне"}
                    </span>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
    </div>

    <div className="card">
```

```
<div className="card–header">
  <span className="card–title">Активні ліміти</span>
  <Link href="/turnover–limits" className="btn btn–ghost btn–sm">
    Всі →
  </Link>
</div>
{limits.length === 0 ? (
  <div className="empty–state">
    <div className="empty–state–icon">◉</div>
    <div className="empty–state–text">Лімітів ще немає</div>
  </div>
) : (
  <div className="table–wrap">
    <table>
      <thead>
        <tr>
          <th>Сума</th>
          <th>Валюта</th>
          <th>Період</th>
        </tr>
      </thead>
      <tbody>
        {limits.slice(0, 5).map((l) => (
          <tr key={l.id}>
            <td>
              <span className="amount">
                {Number(l.limitAmount).toLocaleString("uk–UA")}
              </span>
            </td>
            <td>
```

```
                    <span className="mono">{l.currency}</span>
                  </td>
                  <td>
                    <span className="period–badge">{l.period}</span>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
    </div>
  </div>
  </>
 );
}
```

Додаток В

Розширена методика користування програмою та протоколи тестування

В.1. Інструкція з розгортання та ініціалізації програмного забезпечення

Підготовка робочого середовища вимагає ізольованої конфігурації серверного та клієнтського вузлів. Базовим етапом виступає налаштування змінних оточення. У кореневій директорії серверного ядра формується конфігураційний файл, що містить реквізити прямого доступу до бази даних. Дзеркально, на стороні клієнтського додатка, декларується маршрут для комунікації з внутрішнім програмним інтерфейсом. Процедура фізичного запуску серверної частини зводиться до інсталяції системних залежностей через пакетний менеджер та обов'язкового застосування SQL–міграцій абстракції Prisma. Це гарантує синхронізацію програмних моделей з фізичною структурою таблиць. Після успішної генерації клієнта доступу ініціюється локальний сервер розробки. Запуск візуального інтерфейсу відбувається паралельно, в окремому потоці термінала, після розгортання відповідних клієнтських залежностей.

Після успішного розгортання системи ініціюється етап безпосередньої експлуатації користувацького інтерфейсу. Управління топологією мережі філіалів реалізовано через відповідний модуль навігації. Процедура реєстрації нової структурної одиниці ініціюється викликом модального вікна, де оператор зобов'язаний заповнити критичні атрибути сутності, зокрема найменування та фізичну адресу. Фіксація змін призводить до миттєвої транзакції запису. Облік апаратних терміналів спирається на жорстку прив'язку до існуючих відділень. Створення нового запису вимагає введення унікального ідентифікатора системи віддаленого доступу з подальшим вибором материнського філіалу. Спроба реєстрації дублюючого ідентифікатора превентивно блокується алгоритмами перевірки унікальності. Модель управління фінансовими обмеженнями суворо регламентована часовими рамками. Встановлення ліміту обороту вимагає від оператора визначення додатного числового еквівалента, вибору відділення–власника та фіксації дати початку дії. Формування безстрокових лімітів

відбувається шляхом залишення кінцевої часової мітки порожньою. Для пришвидшення роботи з масивом даних імплементовано механізм динамічної фільтрації: активація відповідного тригера автоматично вилучає з візуального представлення неактуальні (архівні) записи.

Логічним завершенням регламенту експлуатації є протоколювання результатів функціонального тестування розробленого інструментарію. З метою оптимізації сприйняття результатів експериментальної перевірки, загальний масив тестових сценаріїв було декомпозитовано. У таблиці В.1 наведено результати тестування базових операцій управління станом сутностей та перевірки реляційної цілісності.

Таблиця В.1

Протокол верифікації створення та управління станом сутностей

№	Об'єкт тестування / Сценарій	Ініційована дія	Очікуваний та фактичний результат	Статус
1	Ініціалізація сутності відділення	Введення валідних даних у форму створення філіалу	Запис успішно збережено (HTTP 201). Метрики оновлено.	Успішно
2	Цілісність реляційних зв'язків	Спроба ініціалізації робочого місця із зазначенням неіснуючого branchId	Транзакцію відхилено (HTTP 404). Захист БД спрацював штатно.	Успішно
3	Каскадне очищення даних	Видалення відділення, що містить пов'язані ліміти та термінали	Фізичне видалення материнської та всіх дочірніх сутностей.	Успішно

Другий етап тестування, спрямований на верифікацію захисних алгоритмів

та механізмів обробки аномального вводу, задокументовано у таблиці В.2.

Таблиця В.2

Протокол тестування механізмів валідації та фільтрації

№	Об'єкт тестування / Сценарій	Ініційована дія	Очікуваний та фактичний результат	Статус
1	Контроль унікальності AnyDesk ID	Повторне введення зареєстрованого раніше ідентифікатора	Блокування запиту (HTTP 409 Conflict). Відображення помилки.	Успішно
2	Захист від від'ємних фінансових значень	Спроба фіксації суми ліміту зі значенням –100	Жорстке відхилення запиту валідатором (HTTP 400 Bad Request).	Успішно
3	Динамічна фільтрація масиву даних	Активація тригеру показу виключно діючих лімітів	Записи з вичерпаним терміном дії (effectiveTo) приховано.	Успішно