

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Інформаційна система обліку та моніторингу стану ІТ-
активів підприємства»**

Виконав студент групи ПЗ-22-3
БУРКІВСЬКИЙ Сергій Вадимович

Керівник роботи:
СУГОНЯК Інна Іванівна

Рецензент:
ВЛАСЕНКО Олег Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БУРКІВСЬКИЙ Сергій Вадимович**

Керівник роботи: **СУГОНЯК Інна Іванівна**

Тема роботи: **«Інформаційна система обліку та моніторингу стану ІТ-активів підприємства»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: Об'єкт дослідження є процес збору, систематизації, моніторингу та управління життєвим циклом ІТ-активів і програмного забезпечення на підприємстві; Предмет дослідження є алгоритми контролю станів обладнання, архітектурні рішення веборієнтованої системи та технологічний стек розробки платформи для автоматизації обліку та аудиту ІТ-ресурсів. Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Сугоняк І.І.	01.03.2026р.	01.03.2026р
2	Сугоняк І.І.	01.03.2026р	01.03.2026р
3	Сугоняк І.І.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра на тему «Інформаційна система обліку та моніторингу стану ІТ-активів підприємства» містить 76 сторінок друкованого тексту. Робота ілюстрована 35 рисунками та містить 18 таблиць. Перелік використаних джерел налічує 33 найменування. Пояснювальна записка також містить 14 сторінок додатків. Загальний обсяг роботи становить 93 сторінки.

Об'єкт дослідження – процес збору, систематизації, моніторингу та управління життєвим циклом ІТ-активів і програмного забезпечення на підприємстві.

Предмет дослідження – алгоритми контролю станів обладнання, архітектурні рішення веб-орієнтованої системи та технологічний стек розробки платформи для автоматизації обліку та аудиту ІТ-ресурсів.

Мета роботи – підвищення ефективності управління матеріально-технічною базою підприємства шляхом розробки та впровадження інформаційної системи, що забезпечує автоматизований облік, контроль термінів обслуговування та мінімізацію ризиків втрати ІТ-активів.

У роботі використано методи порівняльного й системного аналізу, об'єктно-орієнтованого проектування та моделювання баз даних.

Практичне значення результатів полягає у розробці готової клієнт-серверної системи обліку ІТ-активів. Вона містить інструменти контролю станів обладнання, управління ліцензіями та формування звітності. Стек вебтехнологій (Next.js, TypeScript, Prisma, PostgreSQL) забезпечує надійність збереження даних і швидкодію системи.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, ОБЛІК ІТ-АКТИВІВ, МОНІТОРИНГ СТАНУ, ВЕБЗАСТОСУНОК, NEXT.JS, PRISMA, БАЗА ДАНИХ.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Інформаційна система обліку та моніторингу стану ІТ-активів підприємства	Літ.	Арк.	Аркуші
Розроб.		Бурківський С.В.						
Керівник		Сугоняк І.І.					3	93
Рецензент		Власенко О.В.				Житомирська політехніка, група ІПЗ-22-3		
Зав. каф		Вакалюк Т.А.						

ABSTRACT

The explanatory note to the bachelor's graduation qualification thesis on the topic "Information System for Accounting and Monitoring IT Assets of an Enterprise" contains 76 pages of printed text. The work is illustrated with 35 figures and includes 18 tables. The list of references comprises 33 sources and occupies 3 pages. The explanatory note also contains 14 pages of appendices. The total volume of the work is 93 pages.

The object of the study is the process of collecting, systematizing, monitoring, and managing the lifecycle of IT assets and software within an enterprise.

The subject of the study is algorithms for equipment state monitoring, architectural solutions of a web-based system, and the technology stack for developing a platform for automating accounting and auditing of IT resources.

The aim of the work is to improve the efficiency of managing the material and technical base of an enterprise by developing and implementing an information system that provides automated accounting, maintenance scheduling control, and minimization of risks related to IT asset loss.

The research methods used include comparative analysis for reviewing existing solutions, system analysis for defining system requirements, object-oriented design methods, and database modeling techniques.

The practical significance of the results lies in the development of a ready-to-use software product. The developed IT asset accounting information system is built on a client-server architecture. The system includes tools for monitoring equipment status, managing licenses, and generating automated reports. A modern web technology stack (Next.js, TypeScript, Prisma, PostgreSQL) is used to ensure reliable data storage and system performance.

The developed platform provides administrators with tools for operational auditing, filtering, and in-depth analysis of the enterprise's technical infrastructure.

KEYWORDS: INFORMATION SYSTEM, IT ASSET ACCOUNTING, STATUS MONITORING, WEB APPLICATION, NEXT.JS, PRISMA, DATABASE.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	8
ВСТУП.....	7
РОЗДІЛ 1. ПРЕДМЕТНА ОБЛАСТЬ ТА ПОСТАНОВКА ЗАДАЧІ.....	10
1.1 Постановка задачі та огляд існуючих систем-аналогів.....	10
1.2. Функціональні та нефункціональні вимоги до системи.	14
1.3. Вибір стеку технологій та апаратного забезпечення	18
1.4. Інструментальні засоби розробки.....	21
Висновки до першого розділу	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ.....	26
2.1 Моделювання варіантів використання та поведінки системи	26
2.2. Проєктування логічної структури бази даних.....	31
2.3. Проєктування компонентної структури та взаємодії модулів	37
2.4. Розробка математичної моделі та алгоритмів обробки даних в системі	43
Висновки до другого розділу	49
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ, ІНТЕРФЕЙС ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ІТ-АКТИВАМИ.....	51
3.1 Порядок встановлення та налаштування параметрів системи	51
3.2 Структура інтерфейсу інформаційної системи моніторингу та управління ІТ-активами	54
3.3 Тестування роботи програмного продукту	69
Висновки до третього розділу	73
ВИСНОВКИ	75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	77
ДОДАТКИ	80

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних.

ІС – інформаційна система.

ІТ – інформаційні технології.

ПЗ – програмне забезпечення.

ПК – персональний комп'ютер.

СУБД – система управління базами даних.

ТО – технічне обслуговування.

API (Application Programming Interface) – інтерфейс програмування застосунків.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

ER-діаграма (Entity-Relationship diagram) – діаграма «сутність-зв'язок».

HTML (HyperText Markup Language) – мова розмітки гіпертексту.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, що використовується в API.

ORM (Object-Relational Mapping) – об'єктно-реляційне відображення.

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

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого застосунку.

SaaS (Software as a Service) – програмне забезпечення як послуга.

SPA (Single Page Application) – односторінковий вебзастосунок.

SQL (Structured Query Language) – мова структурованих запитів.

SSR (Server-Side Rendering) – рендеринг (візуалізація) вебсторінок на стороні сервера.

UML (Unified Modeling Language) – уніфікована мова моделювання.

URL (Uniform Resource Locator) – уніфікований покажчик ресурсу.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується цифровізацією всіх сфер управління ресурсами, і сфера ІТ-менеджменту не є винятком. Щодня в межах діяльності підприємств генеруються та змінюються величезні масиви даних про стан інфраструктури: технічні характеристики обладнання, терміни дії ліцензій, історія сервісного обслуговування та переміщення активів. Традиційні підходи до збору та аналізу такої інформації, які часто базуються на використанні розрізнених джерел або універсальних електронних таблиць, є неефективними. Цей процес вимагає значних витрат часу на ручну обробку, підвищує ризик помилок і не дозволяє оперативно отримувати цілісну картину для прийняття стратегічних рішень щодо модернізації чи закупівель.

В умовах високої потреби в оптимізації витрат ІТ-директорам, системним адміністраторам та менеджерам потрібні швидкі, гнучкі та точні інструменти для роботи з інвентаризаційними даними. Саме тому в останні роки спостерігається стійкий тренд на впровадження спеціалізованих платформ для управління активами (IT Asset Management). Розробка профільних інформаційних систем дозволяє автоматизувати процеси збору, агрегації та моніторингу стану техніки. Надання користувачам зручних інтерактивних дашбордів та автоматизованих звітів значно зменшує час на пошук потрібної інформації та знижує ризики втрати активів.

Проте розробка сучасних систем моніторингу ІТ-активів є складною архітектурною задачею. Вона вимагає проектування надійних баз даних для збереження великих обсягів інформації та реалізації оптимізованих алгоритмів обробки запитів. Побудова системи вимагає впровадження ефективних підходів до структурування коду, наприклад, клієнт-серверної архітектури на основі компонентного підходу та API, що дозволяє логічно розділити предметні області (обладнання, персонал, програмне забезпечення) без зайвого ускладнення інфраструктури. Поєднання сучасного стеку технологій для клієнтської та серверної частин забезпечує необхідну продуктивність, інтерактивність та

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

швидкість відгуку системи. З огляду на це, проектування та розробка спеціалізованої інформаційної системи обліку та моніторингу стану ІТ-активів, що базується на сучасних архітектурних патернах, є актуальним науково-практичним завданням.

Об'єкт дослідження – процес проектування, розробки та функціонування інформаційних систем для обробки, збереження та управління даними про матеріально-технічні ресурси.

Предмет дослідження – методи, архітектурні підходи та програмні засоби створення інформаційної системи обліку та моніторингу стану ІТ-активів.

Мета роботи – проектування та розробка програмної системи для агрегації, зберігання та візуалізації інформації про стан ІТ-інфраструктури, яка забезпечує автоматизацію облікових процесів та надає зручний інструментарій для моніторингу життєвого циклу активів.

Для досягнення поставленої мети було сформульовано та вирішено такі завдання:

- проаналізувати предметну область, дослідити особливості управління ІТ-інфраструктурою та провести огляд існуючих інформаційних систем-аналогів.
- обґрунтувати вибір сучасного стеку вебтехнологій та інструментів розробки, визначити функціональні й нефункціональні вимоги до програмного продукту.
- спроектувати загальну архітектуру системи з використанням клієнт-серверного підходу, побудувати об'єктно-орієнтовану модель та описати структуру взаємодії компонентів.
- спроектувати реляційну базу даних для збереження показників обладнання та ліцензій, розробити алгоритми обробки інформації та розмежування прав доступу.
- реалізувати інформаційну систему моніторингу активів, включаючи розробку серверної частини (API), клієнтського інтерфейсу з дашбордами, модулів обліку та панелі адміністратора.

- провести тестування системи, налаштувати механізми безпеки даних та здійснити перевірку працездатності готового програмного продукту.

У роботі використано методи порівняльного та системного аналізу для огляду існуючих рішень і формування вимог до програмного забезпечення, методи об'єктно-орієнтованого проєктування (UML) для розробки структури системи, а також методи моделювання реляційних баз даних.

Практичне значення одержаних результатів. Створена інформаційна система обліку та моніторингу стану ІТ-активів надає користувачам інтуїтивно зрозумілий інструментарій для інвентаризації, контролю гарантійних термінів та візуалізації стану інфраструктури. Це знижує складність аналізу великих масивів інформації та оптимізує часові й ресурсні витрати системних адміністраторів та менеджерів. Архітектурні рішення, закладені в систему, дозволяють масштабувати платформу для потреб підприємств різного розміру та інтегрувати її з іншими корпоративними сервісами.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ПРЕДМЕТНА ОБЛАСТЬ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі та огляд існуючих систем-аналогів

Предметною областю цього дипломного проекту є сфера управління IT-інфраструктурою (IT Asset Management, ITAM [12, 13]), а саме сегмент спеціалізованих інформаційних систем, які надають можливість збирати, обробляти, зберігати та відстежувати дані про апаратні та програмні ресурси підприємства.

Розвиток цифровізації бізнесу диктує нові правила: використання об'єктивних метрик для оцінки стану парку техніки, контролю ліцензійної чистоти ПЗ та прогнозування витрат на модернізацію стало базовою вимогою для досягнення високої конкурентоспроможності. Проте ефективне управління цими масивами даних традиційними методами стикається з низкою суттєвих перешкод.

Традиційний процес обліку IT-активів є складним, багатоетапним та ресурсомістким циклом. Зазвичай для підготовки детального аналізу стану інфраструктури потрібне залучення фахівців, які вручну збирають інформацію з розрізнених джерел. Спочатку дані заносяться у звичайні табличні процесори (наприклад, Microsoft Excel). Далі системні адміністратори намагаються звести цю інформацію в єдину структуру, проводять розрахунки термінів амортизації та вручну будують звіти для керівництва.

Такий класичний підхід є неоптимальним, дорогим і тривалим. Для сучасних технологічних компаній витрати часу на ручну обробку великих масивів даних про сотні одиниць техніки є критичними. Крім того, будь-яка потреба змінити параметри аналізу, наприклад, відфільтрувати техніку за періодом експлуатації, порівняти показники зносу різних типів обладнання або перевірити терміни дії ліцензій вимагає повторного ручного перерахунку. Це створює затримки в ухваленні рішень, підвищує ризик виникнення помилок через людський фактор і значно ускладнює роботу з інформацією.

Усвідомлення цих бар'єрів призвело до еволюції інструментарію IT-менеджменту та появи концепції інтегрованих платформ. Основною ідеєю цього

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

підходу є абстрагування складності роботи з «сирими» даними: рутинні технічні процеси розрахунків, фільтрації та агрегації перекладаються з людини на програмні алгоритми. Замість роботи з неструктурованими таблицями користувач взаємодіє зі зрозумілим графічним інтерфейсом, візуально оцінюючи ключові метрики через інтерактивні дашборди.

Важливою вимогою є забезпечення швидкодії, інтерактивності та адаптивності застосунку. Робота з великими обсягами технічних даних вимагає оптимізованої архітектури (зокрема, використання сучасної клієнт-серверної компонентної архітектури) для швидкої віддачі інформації. Програмний комплекс повинен самостійно вирішувати проблему масштабування інтерфейсу, надаючи віджетам попередньо налаштовану поведінку, яка гнучко перебудовує контент залежно від пристрою користувача.

Спираючись на проведений аналіз, головним завданням даної роботи є проєктування та розробка програмного продукту «Інформаційна система обліку та моніторингу стану ІТ-активів підприємства». Ця система повинна стати комплексним рішенням, де користувач без глибоких навичок роботи з базами даних зможе через зручний інтерфейс керувати повним життєвим циклом обладнання. Платформа має приховати технічну складність побудови SQL-запитів, надавши інтуїтивно зрозумілий інструментарій для моніторингу активів. Досягнення цієї мети вимагає дослідження архітектурних підходів, методів оптимізації реляційних баз даних та організації ефективної клієнт-серверної взаємодії.

Дослідження концепції візуальної аналітики та архітектурних підходів. Стрімкий розвиток ІТ-технологій спричинив поширення концепції Self-Service BI [32] у сфері корпоративного управління. У контексті вебзастосунків для обліку активів це означає зміну внутрішньої архітектури: процес написання скриптів для вибірки даних замінюється взаємодією користувача з графічним інтерфейсом. Ключовим механізмом тут виступає принцип реактивного відображення даних: оновлення стану активу в базі даних має миттєво

відображатися на всіх пов'язаних аналітичних віджетах без перезавантаження сторінки.

Для забезпечення максимальної гнучкості інтерфейсу доцільно застосувати концепцію Schema-Driven UI [33]. Завдяки цьому підходу конфігурація кожного звіту або картки активу зберігається у форматі JSON [30], що дозволяє динамічно підключати потрібні UI-компоненти залежно від типу обладнання. Це повною мірою відповідає сучасним методологіям фронтенд-розробки, де кожен візуальний елемент (наприклад, графік зносу чи таблиця ліцензій) функціонує як незалежний та ізольований компонент.

З метою обґрунтування доцільності створення власного програмного продукту було проведено аналіз наявних рішень на ринку ITAM-систем [12]. Як базові аналоги для дослідження обрано популярні платформи Snipe-IT та GLPI.

Платформа Snipe-IT – є однією з найпопулярніших систем з відкритим вихідним кодом для обліку IT-активів. Її головною перевагою є розвинений інвентарний функціонал: високодеталізовані картки обладнання, генерація штрих-кодів та інтеграція з каталогами (LDAP). Проте монолітна архітектура платформи орієнтована переважно на статичні табличні звіти. Це суттєво обмежує можливості кастомізації аналітичних дашбордів під специфічні метрики підприємства, оскільки зміна інтерфейсу вимагає безпосереднього втручання у вихідний код.

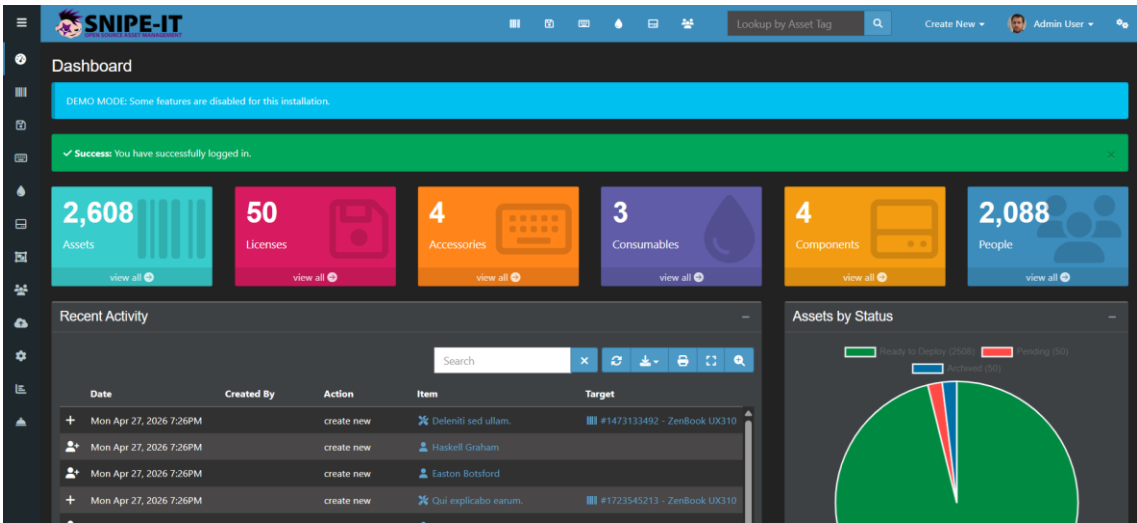


Рис. 1.1 – Інтерфейс вебзастосунку Snipe-IT

Платформа GLPI – це комплексна open-source система, що поєднує облік IT-активів та модуль Service Desk. Незважаючи на високу масштабованість, логіка побудови та інтерфейс GLPI є надмірно складними і перевантаженими десятками додаткових модулів. Більшість цих функцій є надлишковими для малих і середніх підприємств. Впровадження такої громіздкої системи вимагає значних витрат часу на адаптацію персоналу, що підтверджує доцільність розробки більш легкого та інтуїтивно зрозумілого рішення.



Рис. 1.2 – Інтерфейс вебзастосунку GLPI

Для наочної систематизації результатів аналізу існуючих аналогів та обґрунтування функціональних і нефункціональних вимог до власної розробки, основні критерії порівняння зведено у таблицю (табл. 1.1).

Таблиця 1.1

Порівняння інформаційних систем обліку IT-активів

Критерії порівняння	Snipe-IT	GLPI	Платформа, що розробляється
Кастомізація інтерфейсу	Ні	Ні	Так
Формат збереження конфігурацій	Внутрішній	Внутрішній	Відкритий

Конструктор аналітичних звітів	Частково	Ні	Так
Наявність інтерактивних дашбордів	Частково	Ні	Так
Моніторинг життєвого циклу	Так	Так	Так
Архітектурна модульність	Середня	Висока	Висока

Проведений аналіз показує, що існуючі рішення успішно вирішують завдання агрегації даних, проте їхня архітектура передбачає статичне розміщення елементів. Вони не надають користувачеві інструментарію для гнучкого конструювання звітів «під себе». Це підтверджує актуальність розробки власної системи, яка поєднає потужну базу даних на серверній частині та гнучкий, керований конфігураціями інтерфейс.

1.2 Функціональні та нефункціональні вимоги до системи

Аналіз цільової аудиторії та основних сценаріїв використання системи. Одним із ключових етапів проектування програмного продукту є визначення його цільової аудиторії. Розуміння потреб кінцевих користувачів дозволяє оптимізувати користувацький інтерфейс, покращити досвід взаємодії (UX) та сфокусувати зусилля на розробці найбільш затребуваного функціоналу. Аналіз бізнес-процесів управління IT-інфраструктурою дозволяє виділити три основні сегменти цільової аудиторії платформи що розробляється, кожен з яких має власні труднощі та специфічні вимоги до системи.

Першим сегментом є системні адміністратори та IT-інженери. Це користувачі, яким необхідне швидке отримання актуальних технічних даних про кожну одиницю обладнання для оперативного вирішення інцидентів [3, 5]. Основна проблематика полягає в тому, що традиційний облік у ручних таблицях не дозволяє миттєво дізнатися конфігурацію ПК, термін дії гарантії або історію ремонтів без фізичного огляду техніки. Платформа вирішує цю проблему шляхом надання детальних цифрових паспортів активів. Інженер отримує можливість миттєво знаходити потрібний пристрій за серійним номером або QR-

кодом, переглядати його поточний стан та закріпленого користувача через готові інформаційні картки.

Другий сегмент аудиторії – IT-менеджери та керівники відділів. Це професіонали, яким необхідно ухвалювати стратегічні рішення щодо модернізації парку техніки та оптимізації бюджету. Їхньою головною проблемою є відсутність консолідованої аналітики: важко оцінити загальну вартість активів, виявити обладнання, що потребує списання, або відстежити ефективність використання ліцензій на ПЗ. Рішенням, яке пропонує розроблювана платформа, є надання гнучкого візуального конструктора дашбордів, який дозволяє менеджеру за принципом інтерактивних форм та фільтрів самостійно формувати звіти про наповненість складів, витрати на сервісне обслуговування та прогнозовані терміни оновлення застарілої техніки.

Третій сегмент складають бухгалтери та фахівці з логістики. Вони працюють з активами з точки зору фінансового обліку та інвентаризації. Часто вони стикаються з проблемою розбіжностей між фактичною наявністю техніки та записами в документах. На відміну від закритих або надто складних ERP-систем, розроблювана платформа підтримує функції гнучкого експорту та звітності. Користувач може сформувати інвентаризаційну відомість за певним відділом або категорією, а потім вивантажити її у вигляді структурованого CSV або PDF-файлу. При цьому реалізовано принцип «Single Source of Truth» (єдиного джерела істини), що гарантує: будь-яка модифікація даних у звіті не впливає на цілісність глобальної бази даних системи.

На основі профілів цільової аудиторії сформовано базові сценарії взаємодії, які підтверджують здатність платформи покривати різні бізнес-потреби. Сценарій оперативного обліку та закріплення активу розпочинається з надходження нового обладнання на склад. Технічний спеціаліст (комірник) проходить авторизацію, створює нову картку активу, вносить технічні характеристики та завантажує фотофіксацію стану. Далі, використовуючи систему пошуку, він обирає співробітника, за яким закріплюється техніка. Система автоматично генерує запис про переміщення, оновлює статус активу на

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

«В експлуатації» та реактивно перемальовує загальний дашборд наявності техніки на складі.

Сценарій формування аналітичного звіту про життєвий цикл має інший характер. Менеджер відкриває розділ конструктора звітів. За допомогою інтерактивних фільтрів він обирає категорію (наприклад, «Сервери») та параметр фільтрації (термін експлуатації > 5 років). Система миттєво відображає перелік критичного обладнання. Користувач ініціює процедуру експорту, і серверна частина генерує готовий графічний PDF-звіт із діаграмами зносу, який менеджер використовує для обґрунтування бюджету на закупівлю нової інфраструктури.

Функціональні вимоги описують поведінку системи та визначають перелік функцій для задоволення потреб підприємства. Архітектура базується на моделі управління доступом на основі ролей (RBAC). Детальний перелік можливостей наведено у таблиці 1.2.

Таблиця 1.2

Функціональні вимоги до системи за ролями користувачів

Роль користувача	Функціональні можливості у системі
Гість	Перегляд сторінки авторизації, захист від доступу до внутрішніх маршрутів системи (NextAuth).
Співробітник	Перегляд власного профілю та списку техніки, закріпленої за ним. Створення тікетів у службі підтримки (Helpdesk) щодо несправності обладнання.
Технічний спеціаліст	Повний цикл CRUD-операцій над ІТ-активами: додавання, редагування специфікацій, зміна статусів. Проведення операцій видачі/повернення техніки. Обробка та закриття тікетів служби підтримки. Моніторинг ліцензії ПЗ.
Адміністратор	Повний доступ до всіх модулів. Керування ролями користувачів та системними налаштуваннями бази даних. Доступ до дашбордів, конструктора звітів та системних логів (аудит дій та подій безпеки).

Нефункціональні вимоги описують ключові атрибути якості розроблюваної системи, визначаючи суворі інженерні метрики та стандарти її функціонування. На відміну від функціональних вимог, які описують конкретні можливості програми, цей клас вимог фокусується на тому, наскільки швидко,

надійно та безпечно платформа виконуватиме свої завдання в умовах реального корпоративного навантаження. Для забезпечення стабільної та безперебійної експлуатації ІТАМ-системи було сформовано наступний комплекс базових критеріїв.

Вимоги до продуктивності та швидкодії визначають швидкість реакції системи. Цільовий час відгуку серверного API на запити вибірки активів не повинен перевищувати 250 мс. Оскільки інтерфейс побудований на React [24], рендеринг складних таблиць та дашбордів має відбуватися плавно, без підвисань при прокручуванні, а час відгуку користувацького інтерфейсу (UI) на дії користувача не повинен перевищувати 100 мс. Система повинна оптимізувати передачу JSON-даних через стиснення та кешування результатів частих запитів на стороні клієнта.

Вимоги до надійності та відмовостійкості спираються на використання хмарної інфраструктури та сучасної архітектури на базі Next.js [20]. Цільовий показник доступності системи (uptime) 99,9%. База даних PostgreSQL [22] повинна підтримувати автоматичне щоденне резервне копіювання. У разі критичної помилки на сервері (наприклад, у середовищі Vercel [1]), механізм авто-відновлення має перезавантажити сервіс протягом 30-60 секунд без втрати даних.

Вимоги до безпеки є пріоритетними, оскільки система містить дані про внутрішню інфраструктуру. Весь трафік має бути захищений протоколом HTTPS [17]. Паролі користувачів хешуються надійними алгоритмами (bcrypt). Використання ORM (Prisma [23]) на сервері має повністю нівелювати ризики SQL-ін'єкцій при роботі з динамічними фільтрами.

Вимоги до програмного та апаратного забезпечення. Система повинна коректно працювати у сучасних браузерах (Chrome, Firefox, Safari, Edge). Апаратні вимоги до клієнта: мінімум 4 ГБ ОЗП (для комфортної роботи з дашбордами 8 ГБ) та стабільне інтернет-з'єднання від 5 Мбіт/с для швидкого завантаження JSON-конфігурацій та зображень активів.

1.3 Вибір стеку технологій та апаратного забезпечення

Ефективність роботи інформаційної системи обліку та моніторингу ІТ-активів критично залежить від правильно обраного підходу до зберігання даних. Головний архітектурний виклик полягає у необхідності одночасно забезпечувати сувору транзакційну цілісність для базового реєстру техніки (інвентарні номери, серійні коди, фінансові операції) та максимальну гнучкість для збереження технічних характеристик обладнання. Оскільки різні типи активів мають абсолютно несхожі набори параметрів (наприклад, для сервера важливо зберігати кількість ядер та обсяг RAM, а для монітора — діагональ та тип матриці), традиційні підходи до проєктування баз даних [16] з жорсткою схемою потребують перегляду.

Під час проєктування було проаналізовано три основні патерни зберігання динамічних даних. Першим розглянутим підходом став патерн EAV (Entity-Attribute-Value), який історично використовувався в реляційних базах даних для збереження динамічних атрибутів шляхом створення окремих таблиць для сутностей, атрибутів та їхніх значень. Однак його головним недоліком є суттєве падіння продуктивності під час читання та значне ускладнення запитів через велику кількість операцій об'єднання, що створює надмірне навантаження на сервер. Альтернативним підходом є застосування документо-орієнтованих баз даних (NoSQL), таких як MongoDB, де опис активу зберігається як єдиний JSON-документ. Проте відсутність суворої підтримки ACID-транзакцій та механізмів контролю цілісності складних зв'язків робить цей варіант ризикованим для систем фінансового та матеріального обліку. У результаті проєктованої системи було обрано гібридний підхід на базі реляційної СУБД PostgreSQL, яка забезпечує транзакційну цілісність критичних сутностей та одночасно пропонує нативну підтримку типу даних JSONB для збереження варіативних технічних характеристик. Це архітектурне рішення гарантує високу швидкодію читання завдяки виконанню єдиного SQL-запиту, дозволяє уникати складних міграцій під час додавання нових категорій обладнання та забезпечує миттєвий пошук завдяки використанню GIN-індексів.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

Окремим критичним етапом став вибір технологічного стеку для реалізації клієнтської частини (фронтенду). Для забезпечення високої інтерактивності та швидкості відгуку інтерфейсу було проаналізовано популярні JavaScript-фреймворки, такі як Angular, Vue.js та React. Вибір було зроблено на користь бібліотеки React у поєднанні з мета-фреймворком Next.js. Використання інноваційної архітектури Next.js App Router дозволяє вдало поєднати переваги рендерингу на стороні сервера (SSR) та генерації статичних сайтів (SSG). Це суттєво знижує навантаження на клієнтські пристрої, адже браузер отримує вже готову розмітку. Для стилізації користувацького інтерфейсу було інтегровано утилітарний фреймворк Tailwind CSS разом із набором компонентів Shadcn UI. На відміну від традиційних UI-бібліотек з жорстко заданим дизайном (наприклад, Bootstrap), цей підхід надає максимальну гнучкість у створенні унікальної візуальної системи та значно зменшує кінцевий розмір стилів.

Щодо серверної логіки, архітектурне рішення базується на сучасній парадигмі Serverless-обчислень. Замість розгортання монолітного бекенду на базі Express.js чи Nest.js, який вимагає постійно активного сервера, система використовує Next.js Route Handlers. Це дозволяє розбивати серверну логіку на ізольовані безсерверні функції (Serverless Functions), які запускаються лише під час надходження HTTP-запиту. Такий підхід є доцільним для платформ обліку: він мінімізує витрати на холостий хід інфраструктури та забезпечує миттєве горизонтальне масштабування під час пікових навантажень (наприклад, у періоди масової щорічної інвентаризації підприємства). Для надійної та безпечної взаємодії між серверною логікою та базою даних PostgreSQL було інтегровано інструмент об'єктно-реляційного відображення Prisma ORM. На відміну від традиційних рішень, Prisma генерує строго типізований клієнт бази даних на основі декларативної схеми. Це дозволяє виявляти потенційні помилки в SQL-запитах ще на етапі компіляції коду завдяки TypeScript та значно спрощує процеси міграції структури бази даних.

Для забезпечення відмовостійкості, безпеки та безперервного розгортання платформи було обрано сучасні хмарні рішення, що дозволяють побудувати

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

надійну архітектуру для корпоративного використання. Зокрема як платформу для комплексного хостингу (фронтенду та серверних API-маршрутів) обрано Vercel [1]. Цей сервіс оптимізований для розгортання Full-stack-застосунків, розроблених на Next.js. Він надає вбудовану систему CI/CD [11], автоматично налаштовує безпечне HTTPS-з'єднання, роздає статичні файли через глобальну мережу доставки контенту (CDN) та забезпечує швидке завантаження візуальних компонентів системи у користувачів з будь-якого філіалу підприємства. У ролі хмарного провайдера для СУБД PostgreSQL обрано платформу Neon [2] (модель DBaaS). Це дозволяє уникнути витрат часу на ручне адміністрування серверів БД. Платформа бере на себе автоматичне резервне копіювання реєстрів техніки, шифрування даних у спокої та забезпечення високої доступності бази даних.

Важливою інфраструктурною вимогою є наявність власного доменного імені для коректного налаштування політик безпеки CORS та стабільної роботи SSL-сертифікатів. Для виконання цього завдання у ролі реєстратора доменів було обрано сервіс Namescheap. Водночас управління DNS-зоною делеговано потужній глобальній інфраструктурі Cloudflare, що забезпечує системі додатковий рівень безкоштовного мережевого захисту від DDoS-атак.

Окремої уваги заслуговує питання вибору апаратного забезпечення. Історично розгортання корпоративних систем обліку вимагало закупівлі дорогих фізичних серверів, організації безперебійного живлення та найму системних адміністраторів для їхньої локальної підтримки на території підприємства (архітектура On-Premises). Однак обрана хмарна Serverless-архітектура докорінно змінює цю парадигму. Необхідність у власному серверному апаратному забезпеченні повністю відпадає, оскільки всі обчислювальні потужності (CPU, RAM) та фізичні накопичувачі надаються хмарними провайдерами (Vercel та Neon) за моделлю "сервіс як послуга". Це дозволяє підприємству перевести капітальні витрати на обладнання (CAPEX) в операційні (OPEX). Щодо вимог до клієнтського апаратного забезпечення, система є абсолютно невимогливою: завдяки перенесенню важких агрегаційних обчислень

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

на сервер та глибокій оптимізації фронтенду, повноцінна робота з платформою можлива з будь-якого стандартного офісного комп'ютера.

1.4. Інструментальні засоби розробки

Вибір інструментальних засобів розробки є одним із найважливіших етапів проєктування, який безпосередньо впливає на швидкість створення програмного продукту, його надійність, продуктивність, можливості масштабування та зручність подальшої підтримки. Для реалізації проєкту було прийнято рішення використовувати сучасний Full-stack-стек технологій, що базується на єдиній екосистемі TypeScript [10, 29] та фреймворку Next.js для клієнтської та серверної частин. Такий підхід дозволяє поєднати потужні інструменти обробки даних бекенду із сучасними реактивними інтерфейсами фронтенду.

Клієнтська частина системи є надзвичайно складною складовою платформи, оскільки саме вона реалізує логіку візуалізації реєстрів активів, інтерактивних графіків зносу обладнання та редактора аналітичних дашбордів. Як основну бібліотеку для побудови користувацького інтерфейсу було обрано React.js [24] (у складі Next.js). Цей вибір є доцільним для розробки високонавантажених інформаційних рішень з кількох вагомих причин.

По-перше, React [24] базується на суворому компонентному підході. Це ідеально узгоджується з концепцією побудови інтерактивних панелей моніторингу та Schema-Driven UI [33]. Кожен елемент системи (наприклад, картка ноутбука, графік розподілу ліцензій, теплова карта завантаженості складів або панель фільтрів) інкапсулюється в окремий ізольований компонент. Це значно спрощує масштабування бібліотеки віджетів та їх повторне використання у різних модулях системи.

По-друге, ключовою перевагою React [24] є використання віртуальної об'єктної моделі документа Virtual DOM. У системах обліку ІТ-активів, де користувач постійно застосовує складні фільтри до тисяч записів, маніпулює масивами даних або змінює розташування блоків у конструкторі, традиційні маніпуляції з реальним DOM є ресурсомісткими. Завдяки Virtual DOM, алгоритми React [24] швидко обчислюють різницю між станами та точково

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

оновлюють лише ті блоки на екрані, які дійсно змінилися, що забезпечує плавність інтерфейсу.

Як інструмент для збірки та маршрутизації обрано Next.js, який забезпечує швидку заміну модулів (Fast Refresh) та розширені можливості рендерингу (SSR). Це дозволяє не перезбирати весь проєкт під час зміни одного файлу, а миттєво оновлювати лише змінений компонент, а також оптимізувати передачу даних на клієнт.

Серверна частина відповідає за обробку API-запитів, реалізацію складної бізнес-логіки (розрахунок амортизації, агрегацію даних про активи) та безпечну взаємодію з базою даних. У ролі програмної платформи для бекенду обрано інтегровані API-маршрути фреймворку Next.js, які виконуються у середовищі Node.js.

Для забезпечення безпечної авторизації та управління сесіями користувачів було інтегровано бібліотеку NextAuth.js. Цей інструмент надає готові механізми для роботи з JWT-токенами, підтримку рольової моделі (RBAC) та захист маршрутів на рівні серверних Middleware-функцій, що ідеально вписується в екосистему Next.js.

Використання єдиної мови програмування TypeScript [10, 29] для реалізації клієнтської та серверної частин системи є важливою перевагою обраного технологічного стеку. На відміну від класичного JavaScript, TypeScript підтримує статичну типізацію, що дозволяє виявляти значну частину помилок ще на етапі компіляції програмного коду. Це особливо важливо для інформаційних систем, які працюють із великою кількістю взаємопов'язаних сутностей та бізнес-процесів. Крім того, використання спільних інтерфейсів даних для фронтенду та бекенду забезпечує узгодженість структури об'єктів під час передавання інформації між компонентами системи, що зменшує ризик виникнення помилок та спрощує подальшу підтримку програмного продукту.

Окрім засобів збірки та маршрутизації, важливою перевагою Next.js є модель організації програмного коду App Router. Вона дозволяє логічно структурувати програмний код відповідно до предметних областей системи.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдяки цьому модулі управління активами, користувачами, ліцензіями програмного забезпечення та аналітичними звітами можуть розроблятися як відносно незалежні складові єдиного застосунку. Такий підхід підвищує масштабованість проєкту, спрощує супровід програмного забезпечення та створює передумови для подальшого розширення функціональних можливостей платформи без необхідності суттєвої зміни її архітектури.

Для організації доступу до даних використовується сучасна ORM-платформа Prisma [23], яка виконує роль проміжного рівня між прикладною логікою та реляційною базою даних PostgreSQL. Застосування ORM дає можливість працювати з даними через об'єктну модель замість безпосереднього написання великої кількості SQL-запитів. Такий підхід підвищує продуктивність розробки та сприяє зменшенню кількості помилок під час реалізації серверної логіки. Важливою перевагою Prisma є автоматична генерація типів TypeScript [10, 29] на основі схеми бази даних, що забезпечує додатковий контроль цілісності даних та спрощує процес розробки. Крім того, інструмент підтримує механізм міграцій, який дозволяє контролювано змінювати структуру бази даних протягом життєвого циклу програмного продукту.

Для забезпечення високої продуктивності процесу розробки та якісного проєктування архітектури системи було використано низку спеціалізованих програмних засобів. Основним інтегрованим середовищем виступає Visual Studio Code, що відрізняється високою швидкістю та підтримує широкі можливості розширення функціональності. Завдяки модулям для роботи з TypeScript, React і Prisma, це середовище дозволяє автоматизувати рутинні операції, контролювати синтаксис та підвищувати загальну якість програмного коду.

Для проєктування, модифікації структури та комплексного адміністрування реляційної бази даних застосовується графічний клієнт pgAdmin 4. Цей інструмент значно спрощує моніторинг продуктивності SQL-запитів і забезпечує наочне подання зв'язків між сутностями системи.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

Водночас візуалізація архітектурних рішень та побудова UML-діаграм здійснюється за допомогою інструмента PlantUML. Його ключова особливість полягає в текстовому описі моделей, що дає змогу зберігати діаграми в системі контролю версій разом із програмним кодом і прозоро відстежувати всі етапи архітектурних змін.

Враховуючи, що спроектований клієнтський застосунок переносить значну частину обчислювального навантаження на браузер користувача, комп'ютер має відповідати певним апаратним вимогам. Мінімально достатньою конфігурацією є наявність чотирьох гігабайтів оперативної пам'яті, що забезпечує коректне завантаження інвентарних реєстрів та стандартних аналітичних звітів. Для комфортної роботи з великими масивами даних, інтерактивними дашбордами та візуальними конструкторами рекомендовано використовувати не менше восьми гігабайтів оперативної пам'яті.

Стабільна робота клієнтської частини також потребує використання сучасного багатоядерного процесора рівня Intel Core i3, AMD Ryzen 3 або еквівалентного процесора, здатних забезпечити швидке виконання складних JavaScript-сценаріїв [8, 21]. Крім того, оскільки система функціонує за клієнт-серверною моделлю та передбачає постійний обмін даними між браузером і сервером, важливою вимогою є наявність стабільного широкосмугового інтернет-з'єднання зі швидкістю не менше п'яти мегабітів на секунду. Це забезпечує своєчасне завантаження інформації про активи, передачу фотофіксацій обладнання, синхронізацію змін та коректну роботу аналітичних інструментів платформи.

Висновки до першого розділу

У першому розділі було проведено детальний аналіз предметної області та обґрунтовано актуальність створення спеціалізованої інформаційної системи обліку та моніторингу ІТ-активів. Дослідження існуючих на ринку аналогів (наприклад, Snipe-IT, GLPI) дозволило виявити їхні сильні та слабкі сторони, а також сформувати унікальну нішу для розроблюваного продукту – надання користувачам можливості самостійно конструювати інтерактивні аналітичні

звіти та дашборди для контролю життєвого циклу техніки. Окрім цього, було визначено ключові сегменти цільової аудиторії та формалізовано базові сценарії використання системи: від швидкої інвентаризації та закріплення обладнання до глибокого аналізу зносу активів з подальшим експортом даних.

На основі проведеного аналізу було сформульовано головне завдання дослідження та обґрунтовано вибір технологічного стеку і методів зберігання даних. Доведено, що для забезпечення балансу між транзакційною цілісністю (для збереження реєстрів техніки та фінансових операцій) та гнучкістю (для конфігурації динамічних характеристик різних типів пристроїв) оптимальним є гібридний архітектурний підхід: використання реляційної СУБД PostgreSQL [22] із нативною підтримкою формату JSONB. Визначено сучасний Full-stack-стек технологій на базі екосистеми TypeScript [10, 29], React.js [24] у складі фреймворку Next.js [20] для клієнтського інтерфейсу та API-маршрутів серверної логіки, ORM Prisma для роботи з базою даних), а також надійну хмарну інфраструктуру (Vercel [1], Neon [2], Cloudflare).

Також у розділі було чітко визначено функціональні вимоги на основі моделі управління доступом (RBAC), які розмежовують можливості неавторизованих гостей, співробітників, технічних спеціалістів та адміністраторів. Сформульовано конкретні нефункціональні інженерні метрики, що стосуються швидкодії, безпеки, відмовостійкості та апаратного забезпечення. Зібрані вимоги, проаналізовані сценарії та обрані технології створюють міцну концептуальну основу для наступного етапу роботи логічного та фізичного проєктування архітектури програмної системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Моделювання варіантів використання та поведінки системи

Моделювання варіантів використання є одним із ключових етапів об'єктно-орієнтованого проєктування. Воно дозволяє визначити межі програмного комплексу, ідентифікувати основних дійових осіб (акторів) та формалізувати сценарії їхньої взаємодії з інформаційною системою. Цей етап перетворює текстові функціональні вимоги, визначені у першому розділі, у візуальну архітектуру поведінки за допомогою діаграм UML.

На основі розробленої моделі управління доступом на основі ролей (RBAC), в архітектурі платформи було виділено чотири незалежних акторів. Детальний опис кожної ролі наведено у таблиці 2.1.

Таблиця 2.1

Ідентифікація акторів системи

Актор	Опис ролі та зона відповідальності
Гість	Неавторизований відвідувач, має базовий доступ до публічної сторінки входу/авторизації без доступу до захищених NextAuth маршрутів.
Співробітник	Кінцевий користувач платформи (власник техніки), має доступ до особистого кабінету, перегляду закріплених за ним активів та створення сервісних тикетів у службу підтримки (Helpdesk).
Технічний спеціаліст	Співробітник ІТ-відділу, відповідальний за актуальність бази матеріальних активів, має доступ до інвентарних реєстрів, здійснює операції видачі/повернення техніки, проводить ремонти та обробляє сервісні тикети.
Адміністратор	Технічний керівник платформи, має найвищий рівень доступу до всіх підсистем, включаючи управління ролями (RBAC), конструювання аналітичних звітів, повний аудит дій користувачів та керування системними логами.

Зважаючи на масштаб платформи та вимоги архітектури модульного моноліта, усі варіанти використання системи логічно розділені на три функціональні зони: публічний простір, спільний робочий простір (облік, аналітика та Helpdesk) та простір адміністрування.

У межах публічного та особистого просторів платформи реалізовано базові механізми взаємодії для неавторизованих і звичайних користувачів. Процес розпочинається з реєстрації та подальшої безпечної авторизації через встановлення сесії для будь-якого типу облікового запису. Після успішного входу звичайний співробітник отримує можливість переглядати список закріплених за ним ІТ-активів та історію їх переміщень на основі актів прийому-передачі. У разі виникнення несправностей обладнання або необхідності оновлення програмного забезпечення, співробітник може ініціювати подачу сервісної заявки на обслуговування безпосередньо зі свого кабінету.

Робочий простір платформи призначений для технічних спеціалістів та адміністраторів, які виконують основні операції з обліку та аналітики. Інструментарій дозволяє здійснювати повний цикл управління картками активів, що включає створення, редагування специфікацій та видалення записів про обладнання. Для глибокого аналізу даних передбачено використання візуального конструктора, за допомогою якого адміністратори можуть застосовувати багаторівневу фільтрацію активів за категорією, локацією, терміном експлуатації чи гарантійним статусом. Отримані результати моніторингу можна вільно експортувати у форматах JSON, CSV або PDF. По завершенню роботи користувач ініціює безпечний вихід з акаунта.

Простір адміністрування надає інструменти для глобального керування системою та доступний керівному складу (Адміністраторам). Функціонал цієї зони охоплює управління інфраструктурною базою даних, де створюються нові локації, категорії техніки та реєструються постачальники. Адміністратори також розробляють базові шаблони аналітичних звітів, які згодом стають доступними для інших менеджерів. Окрім цього, система забезпечує моніторинг ліцензійної чистоти корпоративного програмного забезпечення та дозволяє здійснювати керування доступом шляхом призначення ролей (RBAC) для всіх облікових записів. Разом з тим, поза межами простору глобального адміністрування, для Технічних спеціалістів виділено окремий функціонал сервісного обслуговування

(Helpdesk). У його контексті технічні спеціалісти здійснюють обробку заявок, змінюючи їхні статуси на різних етапах виконання робіт.

На основі виявлених системних акторів та детально формалізованих прецедентів було спроектовано розширену діаграму варіантів використання. Ця візуальна модель наочно ілюструє архітектуру взаємодії між різними ролями користувачів та ключовими функціональними модулями платформи, що графічно відображено на рис. 2.1.



Рис. 2.1 – Діаграма варіантів використання системи

Наведена візуалізація чітко розмежовує зони відповідальності між різними групами користувачів. Зокрема, актори базового рівня (Гість, Співробітник) взаємодіють переважно з персональними даними, тоді як службові ролі (Технічний спеціаліст, Адміністратор) отримують повноцінний доступ до інструментів контролю та управління базою активів.

Для більш детального розуміння логіки роботи системи було розроблено специфікації для ключових прецедентів платформи. У таблиці 2.2 наведено специфікацію базового варіанта використання «Реєстрація та Авторизація».

Таблиця 2.2

Специфікація прецеденту «Реєстрація та Авторизація»

Атрибут	Опис
Назва прецеденту	Реєстрація та Авторизація
Актор	Гість
Мета	Отримати доступ до функціоналу обліку активів, персональних звітів та управління інвентарем згідно з роллю.
Передумови	Гість знаходиться на сторінці входу, має стабільне інтернет-з'єднання.
Основний сценарій	- Гість вводить облікові дані (email/пароль). - Клієнтський застосунок валідує формат даних. - Сервер перевіряє наявність користувача в БД через NextAuth. - Сервер генерує безпечну сесію та JWT-токен доступу. - Система перенаправляє користувача до Робочого простору.
Альтернативний сценарій	- Невірний пароль: сервер повертає статус 401 Unauthorized, виводиться повідомлення про помилку. - Акаунт заблоковано: система повідомляє про обмеження доступу адміністратором.
Постумови	Користувач отримує доступ до функцій згідно зі своєю роллю (RBAC). Створюється активна сесія.

Одним із ключових інноваційних модулів розробленої системи є гнучкий інструментарій для візуального конструювання аналітичних звітів. Детальний опис бізнес-логіки цього процесу та покроковий алгоритм взаємодії користувача із системою наведено у таблиці 2.3.

Таблиця 2.3

Специфікація прецеденту «Конструювання аналітичного дашборда»

Атрибут	Опис
Назва прецеденту	Конструювання аналітичного дашборда
Актор	Адміністратор
Мета	Створити візуальний звіт про стан ІТ-інфраструктури без написання програмного коду за допомогою інтерактивних форм та фільтрів.
Передумови	Користувач авторизований з правами адміністратора та відкрив редактор аналітичних панелей.

Основний сценарій	1. Система завантажує поточну JSON-конфігурацію дашборда. 2. Користувач обирає потрібний віджет (наприклад, «Діаграма зносу») з панелі інструментів. 3. Через інтерактивні форми та фільтри користувач додає блок у робочу область. 4. Користувач налаштовує параметри фільтрації (наприклад, «Відділ розробки»). 5. Система серіалізує новий стан у JSON та відправляє PUT-запит на сервер.
Альтернативний сценарій	5а. Помилка валідації схеми: сервер відхиляє запит (400 Bad Request), система пропонує відкотити зміни до стабільної версії.
Постумови	Оновлена конфігурація збережена в PostgreSQL (JSONB) та доступна для подальшого моніторингу.

Окрім забезпечення базового клієнтського функціоналу, загальна стабільність та безпека платформи безперервно підтримується суворими адміністративними процесами. Детальна специфікація прецеденту «Призупинення доступу до активу», що наведена у таблиці 2.4, наочно демонструє архітектурні механізми контролю та управління системними ресурсами. Реалізація цього алгоритму гарантує швидке реагування на позаштатні ситуації та своєчасне блокування скомпрометованих записів, що є критично важливим для дотримання корпоративних стандартів ІТ-безпеки.

Таблиця 2.4

Специфікація прецеденту «Призупинення доступу до активу»

Атрибут	Опис
Назва прецеденту	Призупинення доступу до активу
Актор	Технічний спеціаліст, Адміністратор
Мета	Тимчасово або назавжди обмежити використання активу через несправність, сервісне обслуговування або завершення терміну експлуатації.
Основний сценарій	1. Користувач знаходить потрібний актив у загальному реєстрі. 2. Змінює поточний статус на «Ремонт» або «Списано».

Основний сценарій	3. Система вимагає обов'язкового введення причини зміни стану (наприклад, «Вихід з ладу накопичувача»). 4. Сервер через Prisma ORM оновлює запис у БД та автоматично знімає актив із закріпленого співробітника.
-------------------	--

Детальна специфікація інших ключових прецедентів системи, зокрема алгоритмів подачі сервісних тікетів, управління ІТ-активами (CRUD) та обробки заявок на технічне обслуговування, наведена в Додатку А. Усі ці розроблені специфікації фактично визначають базові архітектурні контракти системи. Вони наочно демонструють, що кожна дія актора обов'язково супроводжується глибокою серверною логікою: суворою валідацією прав доступу, управлінням транзакціями та збереженням цілісності станів об'єктів. Така масштабна деталізація бізнес-процесів є необхідними вхідними даними для наступного етапу – проектування реляційної структури бази даних та розробки безпечних API-маршрутів.

2.2. Проектування логічної структури бази даних

Для забезпечення надійного зберігання інформації та підтримки транзакційної цілісності даних про ІТ-інфраструктуру підприємства було обрано реляційну систему управління базами даних PostgreSQL [22]. Ця СУБД гарантує виконання вимог ACID (атомарність, узгодженість, ізолюваність, довговічність), що є критично важливим для збереження об'єктивного реєстру матеріальних активів (інвентарні номери, фінансова вартість, історія переміщень матеріальних цінностей).

Однак, враховуючи специфіку підсистеми моніторингу, де технічні характеристики обладнання є максимально динамічними (кожна категорія техніки від серверів та ноутбуків до периферії й мережевих комутаторів має унікальний набір полів), для зберігання специфікацій активів було використано нативний тип даних JSONB. Цей архітектурний крок дозволяє ідеально поєднати надійність строгих SQL-зв'язків для системних сутностей (користувачі, локації,

логі) із гнучкістю документо-орієнтованих баз для збереження варіативних параметрів обладнання.

Логічна структура бази даних платформи складається з розгалуженої мережі таблиць, які покривають управління користувачами (рольова модель RBAC), категоріями активів, локаціями (склади, офіси), вендорами, ліцензіями ПЗ, сервісними тікетами та безпосередньо збереженими конфігураціями аналітичних звітів.

Для уникнення структурної надлишковості інформації та забезпечення надійної підтримки транзакційної цілісності даних, більшість таблиць спроектованої реляційної моделі було нормалізовано до третьої нормальної форми (3НФ). Такий класичний архітектурний підхід дозволив мінімізувати ризики виникнення аномалій під час виконання базових операцій додавання, оновлення та видалення записів. Крім того, послідовна нормалізація суттєво оптимізувала використання дискового простору та підвищила загальну продуктивність бази даних під час формування складних аналітичних звітів.

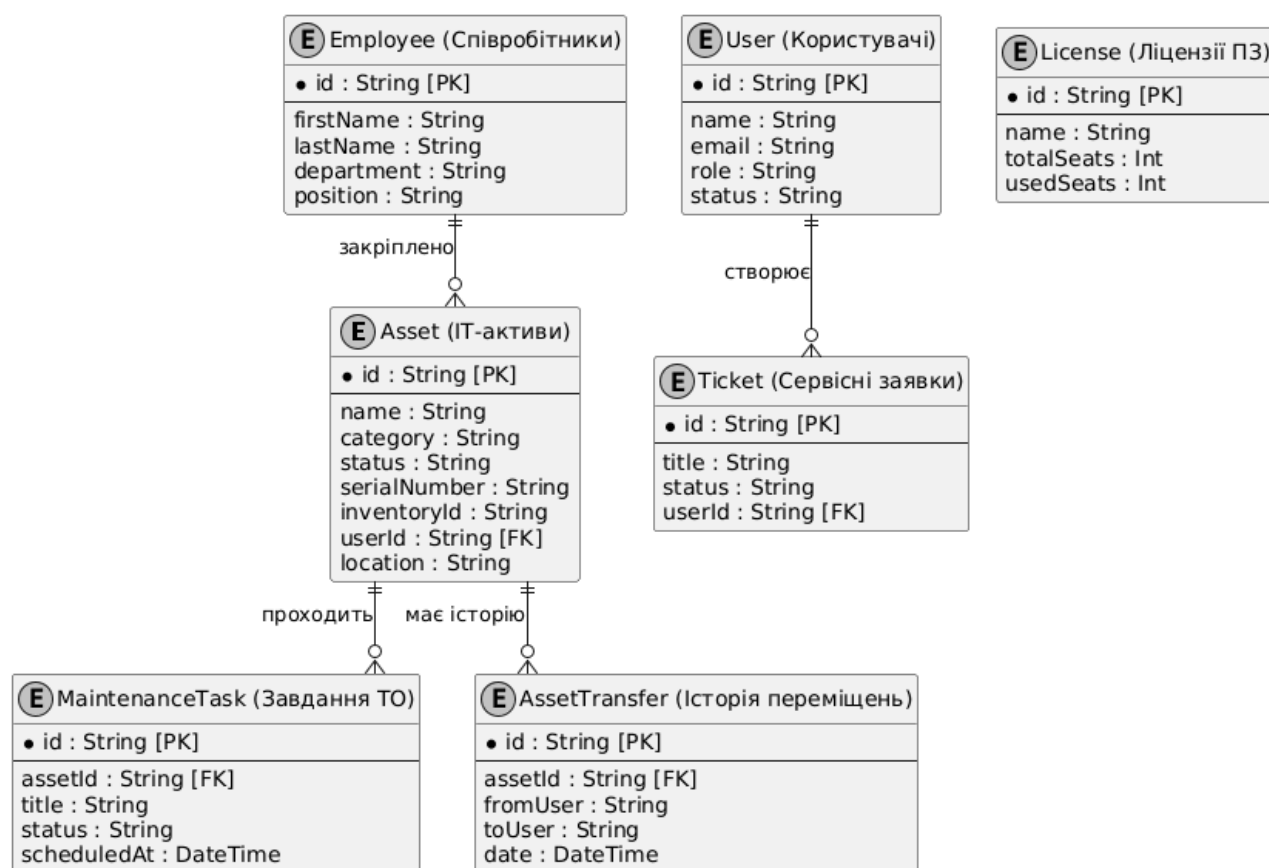


Рис. 2.2 – ER-діаграма бази даних інформаційної системи

Логічна структура реляційної моделі базується на системі чітко визначених зв'язків між ключовими сутностями. Взаємодія між користувачами та обладнанням реалізується через зв'язок типу «один-до-багатьох» (1:N). Ця архітектурна умова гарантує, що хоча один співробітник може бути матеріально відповідальним за декілька одиниць техніки, будь-який конкретний актив у заданий момент часу залишається суворо закріпленим лише за однією особою.

Для організації обліку програмного забезпечення застосовано зв'язок «багато-до-багатьох» (M:N) між таблицями обладнання та ліцензій (Licenses) за допомогою проміжної сутності `asset_licenses`. Такий підхід дозволяє коректно фіксувати випадки, коли на один пристрій встановлено декілька програмних продуктів, або коли єдина корпоративна ліцензія розподіляється між багатьма комп'ютерами чи серверами.

Окремо було спроектовано та реалізовано реляційний зв'язок типу «один-до-багатьох» спеціально для модуля технічного обслуговування (MaintenanceTask). Завдяки такому архітектурному рішення кожна одиниця ІТ-інфраструктури здатна накопичувати детальну історію сервісних робіт та ремонтів протягом усього свого експлуатаційного циклу.

Безпосередня реалізація логічної моделі та створення фізичних таблиць у базі даних здійснювалися за допомогою автоматизованих міграцій Prisma ORM [23]. Детальні специфікації найважливіших сутностей ядра платформи, які описують їх структуру та типи полів, наведено у відповідних таблицях 2.5–2.8. Зокрема, для безпечного зберігання облікових записів користувачів, хешування паролів та гнучкого налаштування прав доступу було розроблено таблицю `users` (табл. 2.5).

Таблиця 2.5

Структура таблиці `users`

Назва поля	Тип даних	Обмеження	Опис
<code>id</code>	String (UUID)	PK, @default(uuid())	Унікальний ідентифікатор користувача.
<code>name</code>	String	NOT NULL	Прізвище та ім'я співробітника.

email	String	UNIQUE, NOT NULL	Корпоративна пошта для авторизації.
password	String	NOT NULL	Хешований пароль (за допомогою bcrypt).
role	String	DEFAULT(«tech»)	Роль у системі (admin, tech, user).
createdAt	DateTime	@default(now())	Дата створення профілю.
updatedAt	DateTime	@updatedAt	Автоматична фіксація дати оновлення.

Фундаментальною сутністю всього модуля інвентаризації виступає ІТ-актив. Для його надійного збереження в базі даних розроблено таблицю assets (табл. 2.6). Ця таблиця структурно поєднує універсальні базові метадані обладнання та гнучку колонку для зберігання унікальних технічних специфікацій. Такий гібридний підхід до проектування моделі даних дозволяє легко масштабувати систему під нові типи пристроїв без необхідності постійної модифікації базової схеми.

Таблиця 2.6

Структура таблиці assets

Назва поля	Тип даних	Обмеження	Опис
id	String (UUID)	PK, @default(uuid())	Унікальний ідентифікатор активу.
name	String	NOT NULL	Коротка назва обладнання.
category	String	NOT NULL	Категорія (наприклад, сервер, ноутбук).
status	String	NOT NULL	Поточний стан (в експлуатації, списано).
serialNumber	String	UNIQUE, NOT NULL	Заводський серійний номер пристрою.

inventoryId	String	UNIQUE, NOT NULL	Унікальний інвентарний штрих-код.
price	Float	NOT NULL	Балансова вартість активу.
location	String	NOT NULL	Поточне місцезнаходження (офіс/склад).
specs	JSONB	NULL	Динамічні характеристики у форматі JSONB.
createdAt	DateTime	@default(now())	Дата додавання активу в систему.

Для фіксації історії планового технічного обслуговування та ремонтних робіт використовується таблиця MaintenanceTask. Кожен запис у цій таблиці представляє окреме сервісне завдання, яке жорстко прив'язане до конкретної одиниці обладнання.

Таблиця 2.7

Структура таблиці MaintenanceTask

Поле	Тип даних (Prisma)	Опис
id	String (UUID)	Унікальний ідентифікатор завдання
assetId	String	Ідентифікатор активу, що ремонтується
title	String	Коротка назва робіт
description	String?	Детальний опис проблеми
scheduledAt	DateTime	Запланована дата ТО
completedAt	DateTime?	Фактична дата завершення робіт
status	String	Статус (Заплановано, В роботі, Вирішено)
type	String	Тип завдання (ТО, Ремонт, Списання)

Для надійної фіксації повної історії планового технічного обслуговування та позапланових ремонтних робіт у базі даних використовується спеціалізована таблиця MaintenanceTask. Кожен окремий запис у цій таблиці логічно представляє конкретне сервісне завдання або інцидент, який жорстко прив'язаний

до відповідної одиниці обладнання через систему зовнішніх ключів. Такий підхід гарантує стовідсоткову простежуваність життєвого циклу кожного ІТ-активу та дозволяє формувати точну статистику щодо частоти його поломок і витрат на утримання.

Таблиця 2.8

Структура таблиці audit_logs

Назва поля	Тип даних	Обмеження	Опис
id	String (CUID)	PK, @default(cuid())	Унікальний ідентифікатор логу.
time	String	NOT NULL	Точний час виконання системної події.
actor	String	NOT NULL	Ідентифікатор користувача, який виконав дію.
type	String	NOT NULL	Тип події (наприклад, STATUS_CHANGE).
source	String	NOT NULL	Модуль системи, звідки ініційовано дію.
text	String	NOT NULL	Детальний текстовий опис модифікації даних.
createdAt	DateTime	@default(now())	Дата і час фіксації транзакції в БД.

Ключовим архітектурним рішенням розробленої платформи є використання формату JSON для зберігання динамічних даних. Такий підхід забезпечує максимальну гнучкість моделі даних при масштабуванні системи. Це дозволяє повністю уникнути виконання складних і ризикованих SQL-міграцій при додаванні нових типів ІТ-обладнання чи інтеграції нових аналітичних віджетів. Стандартний вузол технічних специфікацій у полі assets.specs містить ключі:

- processor / ram / storage для обчислювальної техніки (ноутбуки, сервери).
- display_size / resolution для моніторів.
- ip_address / mac_address / ports для мережевого обладнання.

Під час завантаження картки активу або генерації аналітичного звіту, Backend-частина на базі Next.js [20] виконує єдиний високоефективний запит до

PostgreSQL. Завдяки використанню формату JSONB (Binary JSON), система повноцінно підтримує індексацію вкладених полів за допомогою GIN-індексів. Це дозволяє миттєво фільтрувати тисячі активів за конкретними технічними параметрами без падіння продуктивності, що є типовою проблемою класичних реляційних моделей із великою кількістю «важких» JOIN-операцій.

2.3. Проектування компонентної структури та взаємодії модулів

Діаграма розгортання призначена для моделювання фізичної та інфраструктурної архітектури системи. За допомогою цієї діаграми можна оцінити конфігурацію серверних потужностей та хмарних сервісів, на яких працюватимуть окремі компоненти платформи, а також описати топологію їхньої взаємодії.

Оскільки система проектується як сучасне SaaS-рішення для управління корпоративною інфраструктурою, її архітектура базується на розподіленій хмарній моделі. На рис. 2.3 представлена розроблена діаграма розгортання програмного комплексу.

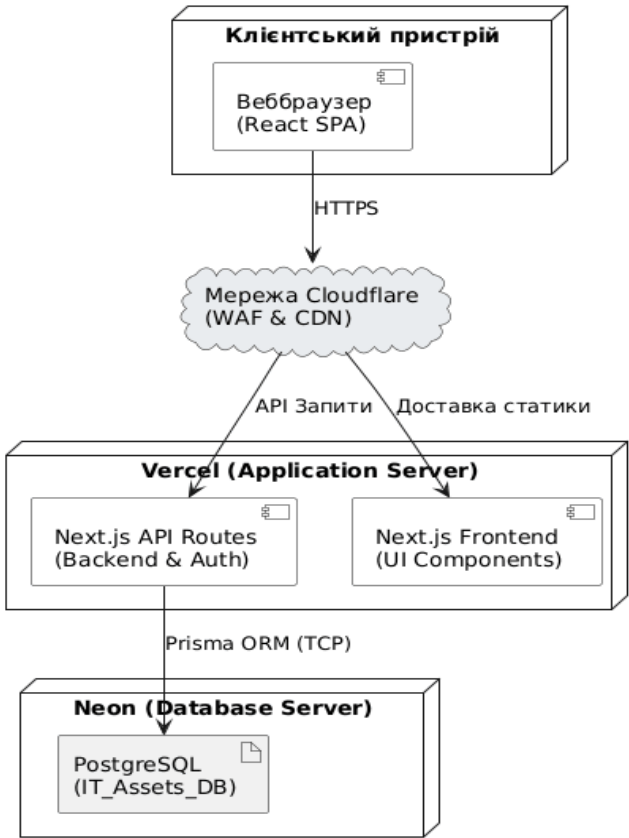


Рис. 2.3 – Діаграма розгортання системи

Фізична інфраструктура платформи є суворо розподіленою та охоплює відразу кілька ключових архітектурних вузлів. Первинною точкою входу в систему виступає локальний пристрій користувача із запущеним сучасним веббраузером. Саме його внутрішнє середовище забезпечує швидке виконання клієнтської частини додатка, яка побудована на базі бібліотеки React [24].

Зовнішній мережевий трафік маршрутизується через глобальний реверс-проксі Cloudflare, який створює безпечний периметр завдяки фільтрації запитів, захисту від DDoS-атак та управлінню SSL-сертифікатами. Обробка проксіюваних запитів здійснюється хмарною платформою Vercel [1], яка виконує роль сервера застосунку. Розгорнутий на ній Full-stack-застосунок Next.js [20] забезпечує швидку доставку фронтенду через CDN та паралельно керує API-маршрутами для виконання бізнес-логіки.

Фізичне збереження, структурування та цілісність даних забезпечує потужний ізольований хмарний вузол PostgreSQL [22]. Цей вузол розгорнутий на базі сучасного керованого сервісу Neon [2], який підтримує автоматичне масштабування обчислювальних ресурсів. Продумана архітектура безпеки передбачає суворе обмеження доступу до бази даних безпосередньо на мережевому рівні. Підключення дозволяється виключно за довіреними IP-адресами внутрішніх серверних функцій Vercel [1]. Такий жорсткий контроль повністю виключає будь-яку можливість несанкціонованого підключення ззовні та гарантує конфіденційність усієї корпоративної інформації підприємства.

Для кращого розуміння роботи системи доменна модель описує всі ключові сутності предметної області та стійкі логічні зв'язки між ними. Цей підхід дозволяє архітектору абстрагуватися від технічних деталей і зосередитися виключно на бізнес-вимогах. На рис. 2.4 детально подано діаграму класів предметної області, яка спеціально розроблена без урахування специфічних інфраструктурних патернів.

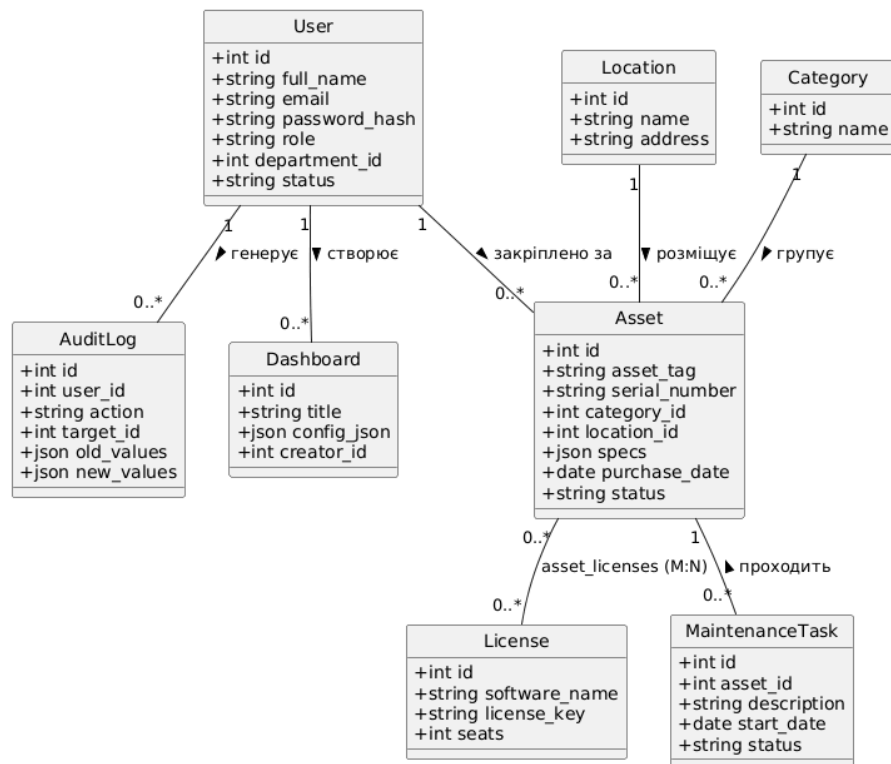


Рис. 2.4 – Діаграма класів предметної області

Центральною доменною сутністю розробленої інформаційної системи виступає основоположний клас User, який відповідає за ідентифікацію та авторизацію співробітників. Цей базовий клас утворює строге відношення композиції з класами Asset (інвентарні активи) та AuditLog (журнал системного аудиту), що забезпечує чітку та прозору ієрархію всіх системних даних. Зі свого боку, багатогранний клас Asset виконує ключову роль базового агрегатора технічних даних про обладнання. Крім того, він логічно пов'язаний із класом Location для точного географічного відстеження фізичного розташування кожного пристрою. Унікальною архітектурною особливістю класу Asset є використання гнучкого атрибута specs формату JSONB, який застосовується для безпечного зберігання динамічних конфігурацій та неструктурованих технічних специфікацій.

Окремої уваги під час розробки також заслуговують обрані патерни проєктування та загальна архітектура програмної реалізації платформи [18]. Головною метою архітектора під час їх вибору було гарантування надвисокого рівня масштабованості вихідного коду та суттєве спрощення його подальшої

технічної підтримки. Для досягнення цієї мети було успішно застосовано перевірені часом сталі інженерні патерни розробки програмного забезпечення [9, 26]. Ці патерни дозволяють ефективно відділити бізнес-логіку від інтерфейсу користувача та абстрактного шару доступу до бази даних. Базова модульна структура та логіка взаємодії цих патернів у рамках розробленої системи наочно продемонстрована за допомогою відповідної UML-діаграми на рис. 2.5.

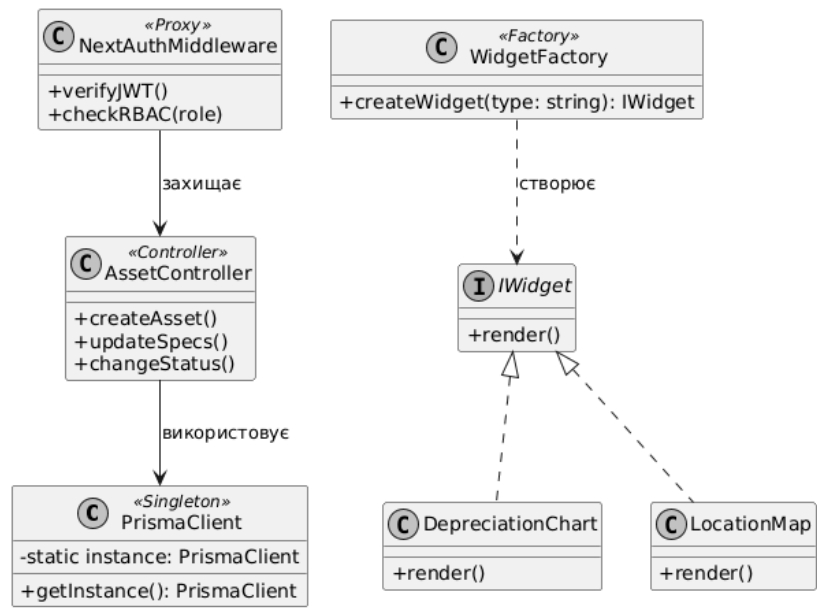


Рис. 2.5 – Діаграма класів реалізації з використанням патернів

Щодо архітектурної реалізації, у системі застосовано низку класичних патернів проєктування. Зокрема, фундаментальним підходом для серверної частини є використання патерну MVC (Model-View-Controller), де API-маршрути Next.js виконують функцію контролерів, тоді як клієнти Prisma ORM виступають у ролі моделей даних. Для динамічного створення різноманітних графічних елементів, таких як діаграми зносу чи мапи локацій, на стороні клієнта реалізовано патерн Factory (Фабрика) на основі JSON-конфігурацій. Крім того, для оптимізації роботи з базою даних застосовано патерн Singleton (Одинак), який гарантує існування єдиного екземпляра пулу з'єднань через PrismaClient. Механізми безпеки, зокрема перевірка сесій JWT-токенів за допомогою NextAuth та контроль прав доступу, побудовані з використанням патерну Proxy/Decorator, що реалізований через Middleware-функції Next.js.

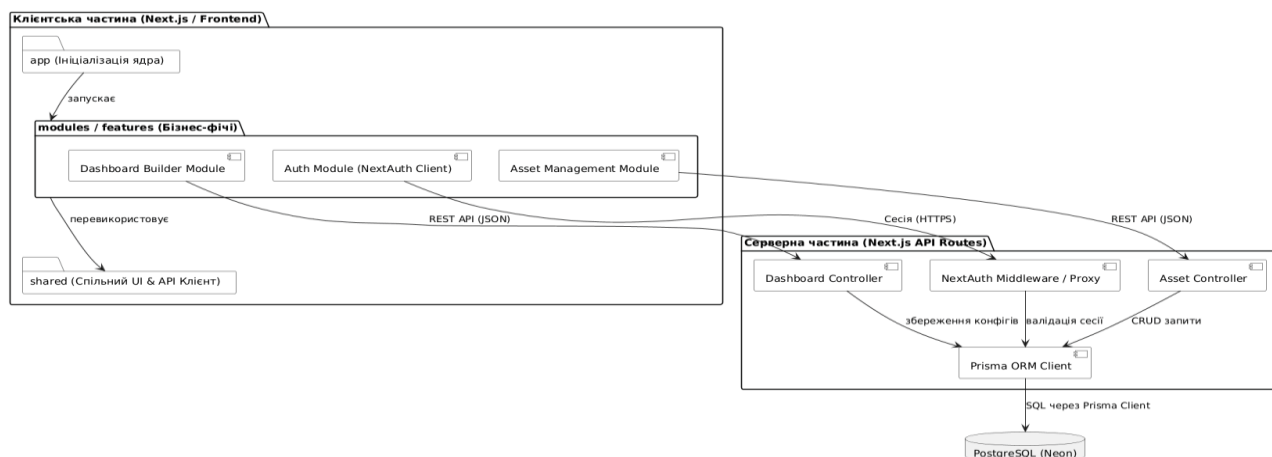


Рис. 2.6 – Діаграма компонентів програмного комплексу

Архітектура клієнтської частини, побудованої на базі бібліотеки React [24], спроектована з використанням сучасної методології Feature-Sliced Design (FSD) [6]. Відповідно до цього підходу, кодову базу чітко структуровано за стандартизованими шарами: рівень `app` відповідає за ініціалізацію ядра та глобальні налаштування, шар `modules/features` містить ізольовані блоки бізнес-логіки (зокрема, `asset_management`, `dashboard_builder` та `auth`), а рівень `shared` об'єднує універсальні перевикористовувані компоненти інтерфейсу. Такий поділ забезпечує високу масштабованість проєкту та суттєво полегшує його подальшу підтримку.

Для детальнішого розуміння динаміки роботи системи використовуються діаграми послідовності, які наочно відображають хронологію та логіку взаємодії програмних об'єктів у часі. Зокрема, на рис. 2.7 проілюстровано покроковий процес створення нової картки ІТ-активу, починаючи від ініціалізації запиту користувачем і закінчуючи успішним збереженням даних у базі.

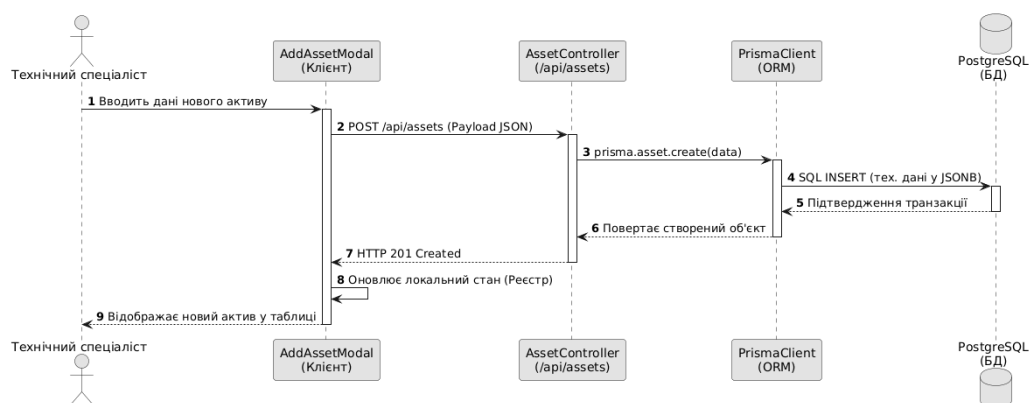


Рис. 2.7 – Діаграма послідовності «Додавання нового активу»

Відповідно до представленої діаграми послідовності (рис. 2.7), базовий алгоритм додавання нового активу в систему складається з таких логічних етапів:

- Технічний спеціаліст вводить необхідні технічні параметри у клієнтську форму AddAssetModal.
- Фронтенд-частина формує та надсилає POST-запит на захищений серверний ендпоінт /api/assets.
- Обробник маршруту (Controller) валідує вхідні дані та викликає метод створення об'єкта через клієнт Prisma.
- Об'єктна модель виконує атомарну транзакцію в базі даних PostgreSQL, записуючи динамічні специфікації у спеціалізоване поле JSONB.
- Після успішного завершення транзакції сервер повертає HTTP-статус 201 Created, а клієнтський інтерфейс динамічно оновлює реєстр техніки без перезавантаження сторінки.
- Наступним важливим системним процесом є процедура призупинення використання активу. Ця складна бізнес-транзакція з умовним розгалуженням логіки детально ілюструється на рис. 2.8.

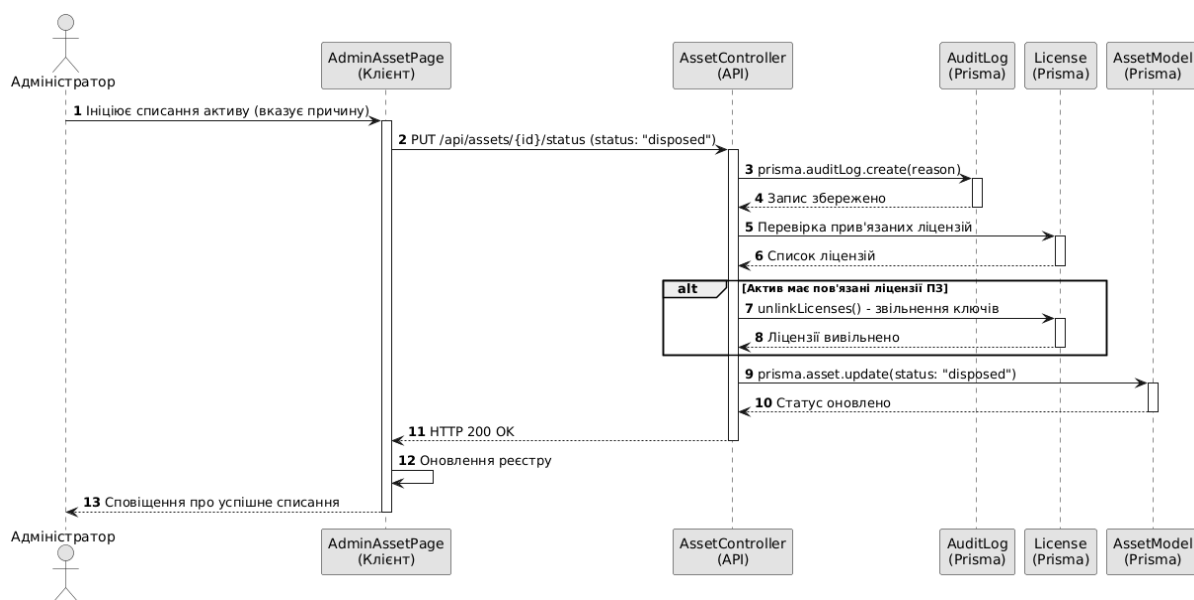


Рис. 2.8 – Діаграма послідовності «Зміна статусу активу»

Адміністратор ініціює зміну статусу через інтерфейс AdminAssetPage. Сервер фіксує причину зміни в моделі AuditLog. Якщо актив має прив'язані ліцензії ПЗ (блок alt), контролер вивільняє ключі через метод unlinkLicenses().

Після цього оновлюється статус об'єкта в AssetModel. Система повертає відповідь 200 OK та динамічно оновлює інтерфейс користувача. Структурні зв'язки об'єктів під час транзакції демонструє діаграма кооперації (рис. 2.9).

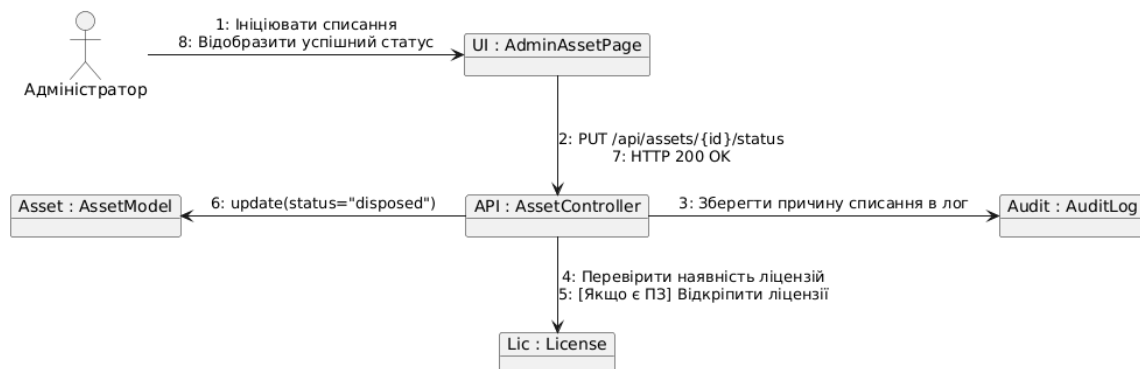


Рис. 2.9 – Діаграма кооперації «Зміна статусу активу»

Діаграма демонструє радіальну топологію: серверний контролер маршруту виступає центральним вузлом, який координує роботу трьох моделей (Asset, AuditLog, License), забезпечуючи повну транзакційну цілісність операції управління IT-інфраструктурою. Така декомпозиція дозволяє незалежно оновлювати окремі частини системи без ризику порушення загальної бізнес-логіки.

2.4. Розробка математичної моделі та алгоритмів обробки даних в системі

Для детального розуміння внутрішньої логіки платформи розроблено алгоритми роботи ключових бізнес-процесів. Візуалізація алгоритмів здійснена за допомогою діаграм активності (Activity Diagrams). Ці діаграми відображають динаміку поведінки системи, показуючи паралельні процеси, розгалуження та чіткий розподіл зон відповідальності між Клієнтом (Менеджером/Співробітником) та Сервером (Backend / БД) за допомогою доріжок (swimlanes).

Алгоритм процесу «Створення звіту про стан активів» описує детальну логіку ініціалізації нового аналітичного звіту або інвентаризаційної відомості. Формування таких документів відбувається на базі заздалегідь підготовленого шаблону структури, що значно прискорює роботу кінцевого користувача. Цей

процес охоплює як клієнтську взаємодію з інтерфейсом, так і складні серверні операції з агрегації необхідних даних.

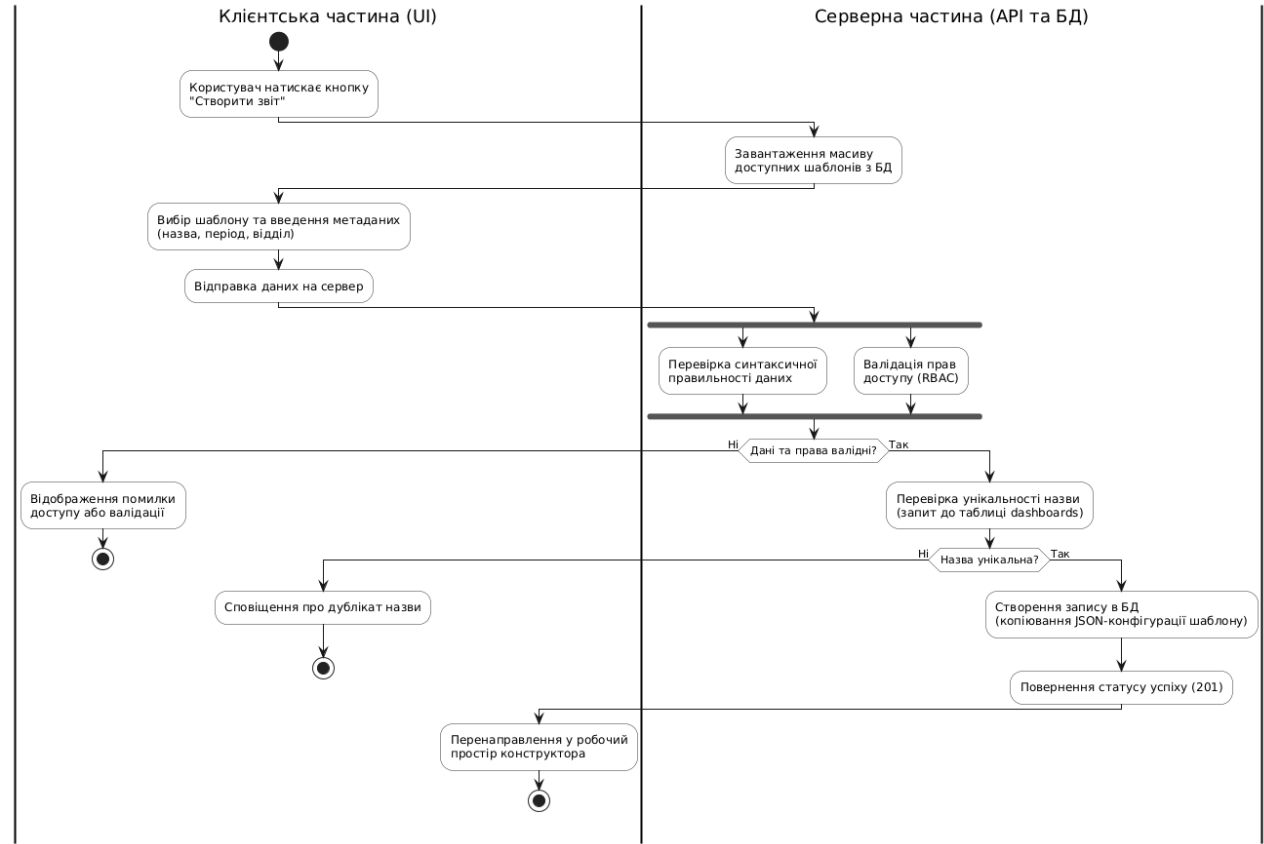


Рис. 2.10 – Діаграма активності «Створення аналітичного дашборда»

Процес створення звіту про стан активів розпочинається з ініціалізації, коли користувач із відповідними адміністративними правами активує функцію конструювання дашборда. На цьому етапі клієнтський застосунок звертається до серверної частини для завантаження масиву доступних шаблонів конфігурацій у форматі JSON, що дає змогу динамічно формувати інтерфейс вибору без потреби жорсткого кодування на фронтенді. Далі відбувається етап введення інформації, під час якого користувач взаємодіє з формою, обираючи базовий шаблон (наприклад, «Дашборд інвентаризації») та заповнюючи ключові метадані, такі як назва панелі, звітний період і цільовий відділ. Ці дані тимчасово зберігаються в локальному стані браузера. Для оптимізації часу обробки серверна частина виконує паралельну валідацію: одночасно перевіряється синтаксична правильність введених даних відповідно до схем валідації та підтверджуються права доступу користувача (RBAC) до інформації обраного підрозділу.

Наступним кроком є миттєва перевірка унікальності назви звіту шляхом звернення до проіндексованої таблиці dashboards, щоб уникнути дублювання в межах робочого простору. За умови успішного проходження всіх перевірок у базі даних створюється новий запис із копіюванням базової JSON-конфігурації віджетів із вибраного шаблону, після чого механізм маршрутизації автоматично перенаправляє користувача у робочий простір візуального конструктора. Алгоритм процесу «Редагування технічних специфікацій активу». Цей процес демонструє роботу з динамічними даними обладнання. Через різноманітність комп'ютерної техніки класичний реляційний підхід є неефективним. Тому основне навантаження щодо перевірки типів лягає на клієнтську частину (використовується суворі валідація JSON-схем через бібліотеку Zod [31]), а сервер бази даних PostgreSQL виступає надійним сховищем, що через нативний тип JSONB гарантує структурну цілісність специфікацій.

Застосування такої архітектури дозволяє повністю абстрагувати рівень бази даних від бізнес-логіки конкретних пристроїв. У разі закупівлі підприємством абсолютно нового типу обладнання (наприклад, графічних планшетів чи спеціалізованих мережевих сховищ NAS), адміністратору не потрібно змінювати жорстку структуру SQL-таблиць або виконувати складні міграції бази даних. Достатньо лише оновити JSON-конфігурацію, і клієнтська частина автоматично згенерує відповідну форму вводу з необхідними новими полями.

Окрім цього, глибока інтеграція Zod-валідації на етапі клієнт-серверної взаємодії гарантує, що до бази даних ніколи не потрапить неформатована або нетипізована інформація. Наприклад, якщо поле «обсяг оперативної пам'яті» очікує виключно числове значення в гігабайтах, спроба користувача ввести туди звичайний текстовий рядок буде миттєво відхилена ще на рівні браузера. Це суттєво зменшує марне навантаження на мережу та серверні потужності. Такий комплексний підхід до управління технічними даними робить платформу надзвичайно гнучкою, стійкою до помилок та повністю готовою до будь-яких змін у корпоративній IT-інфраструктурі.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

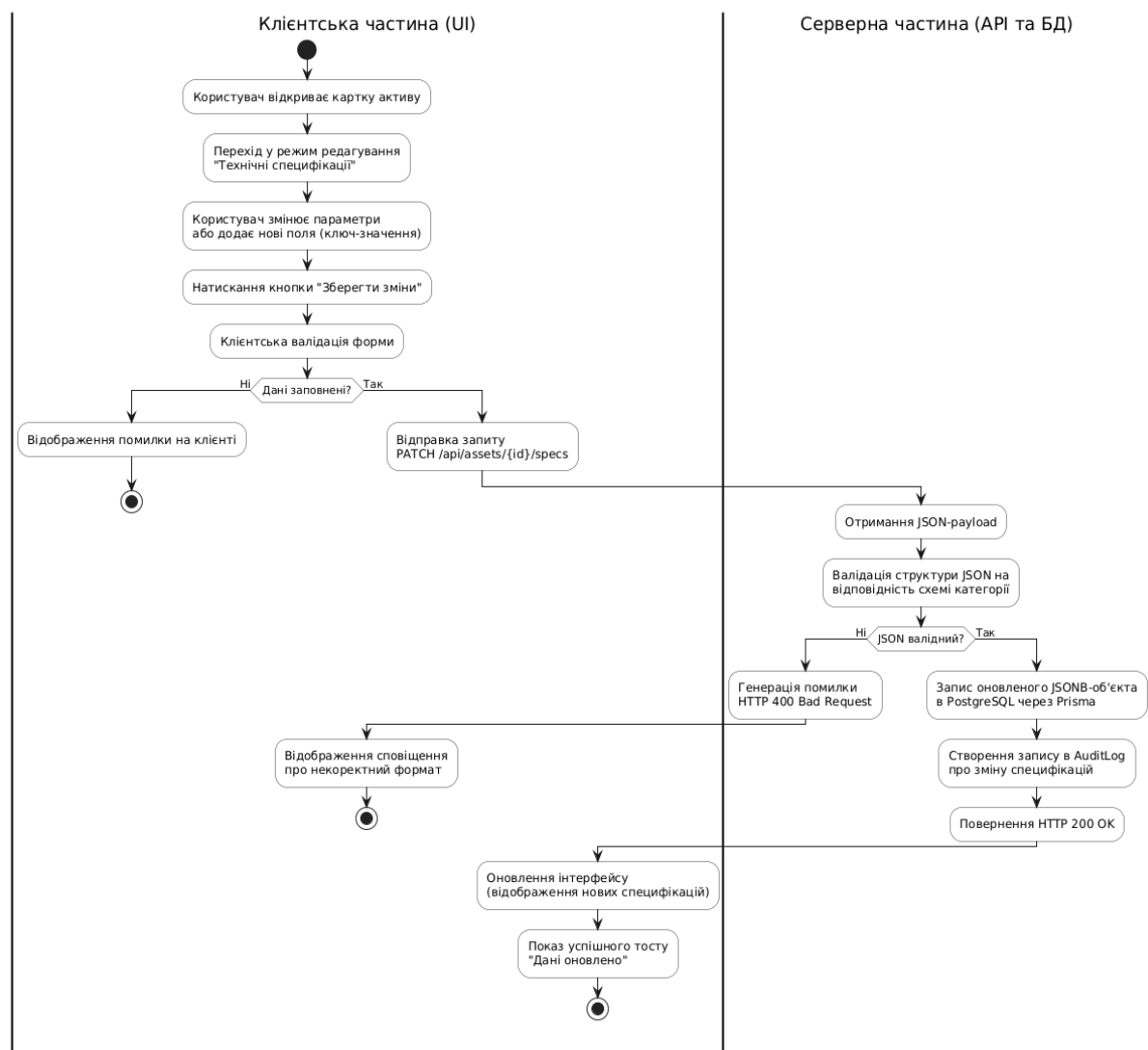


Рис. 2.11 – Діаграма активності «Редагування технічних специфікацій»

Під час редагування технічних специфікацій система спочатку завантажує поточний стан активу, передаючи об'єкт із поля JSONB на клієнтську частину. На основі отриманих даних фронтенд здійснює рендеринг динамічної форми, яка автоматично адаптується під специфіку конкретного пристрою, відображаючи відповідні поля для заповнення. У процесі взаємодії технічний спеціаліст вносить зміни до параметрів, додає нові характеристики або завантажує фотофіксацію безпосередньо в локальній пам'яті браузера. Після завершення редагування ініціюється етап синхронізації, під час якого клієнтська частина формує фінальний оновлений масив даних. Зрештою, сервер приймає цей об'єкт, проводить валідацію на відповідність схемі обраної категорії та виконує оновлення в базі даних, після чого інтерфейс інформує користувача про успішне збереження змін.

Ще одним важливим компонентом системи є алгоритм процесу «Отримання доступу до моніторингу ліцензій». Цей алгоритм детально описує безпекову логіку взаємодії користувача із платформою при спробі перегляду конфіденційних даних, таких як ліцензійні ключі ПЗ або терміни дії корпоративних підписок. Ключовим етапом цієї взаємодії є суворя перевірка прав доступу на основі ролівої моделі (RBAC), що гарантує надійний захист комерційної інформації від несанкціонованого доступу.

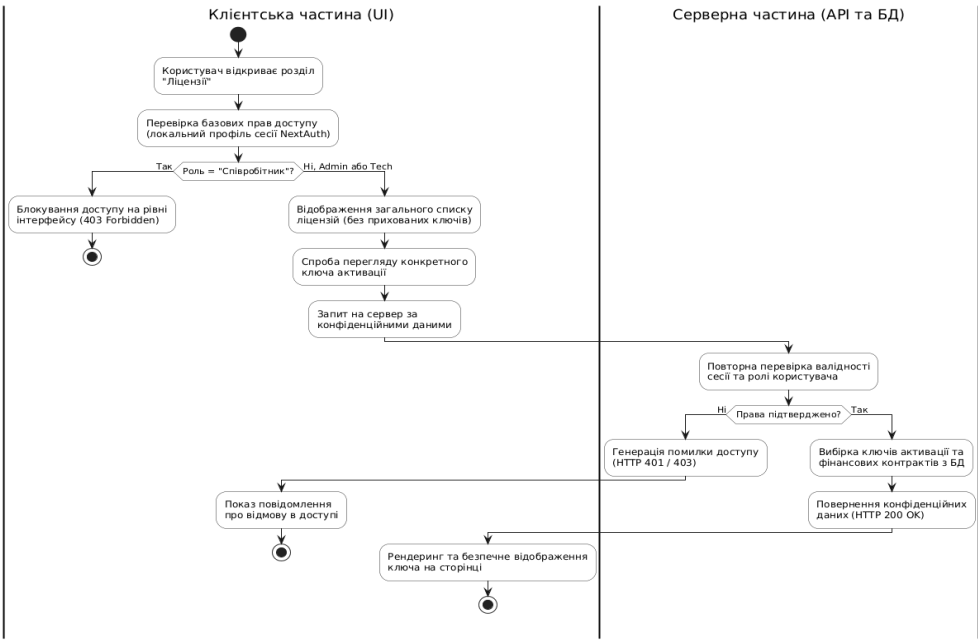


Рис. 2.12 – Діаграма активності «Взаємодія з модулем ліцензій»

Процедура доступу до модуля моніторингу ліцензій супроводжується багаторівневою перевіркою прав користувача. Коли особа намагається переглянути відповідний розділ, клієнтський застосунок спершу перевіряє базові права на основі локального профілю сесії NextAuth. Якщо система ідентифікує користувача як звичайного співробітника, доступ до конфіденційної інформації (ключів активації чи фінансових контрактів) блокується безпосередньо на рівні інтерфейсу з поверненням статусу помилки. У випадку, коли перегляд ініціює технічний спеціаліст або адміністратор, система надсилає запит на сервер для додаткового підтвердження розширених прав доступу та валідації сесії. Тільки після успішного проходження цих серверних перевірок конфіденційні дані передаються клієнту та безпечно відображаються на екрані.

Наступним важливим бізнес-процесом системи є алгоритм «Списання активу Адміністратором». Він відповідає за критичну функцію системної модерації та суворого фінансового контролю за станом корпоративного обладнання. Головна особливість цього алгоритму полягає в демонстрації складного механізму автоматичного оновлення всіх пов'язаних сутностей у базі даних (наприклад, відв'язування ліцензій чи закриття історії обслуговування) під час списання основного активу.

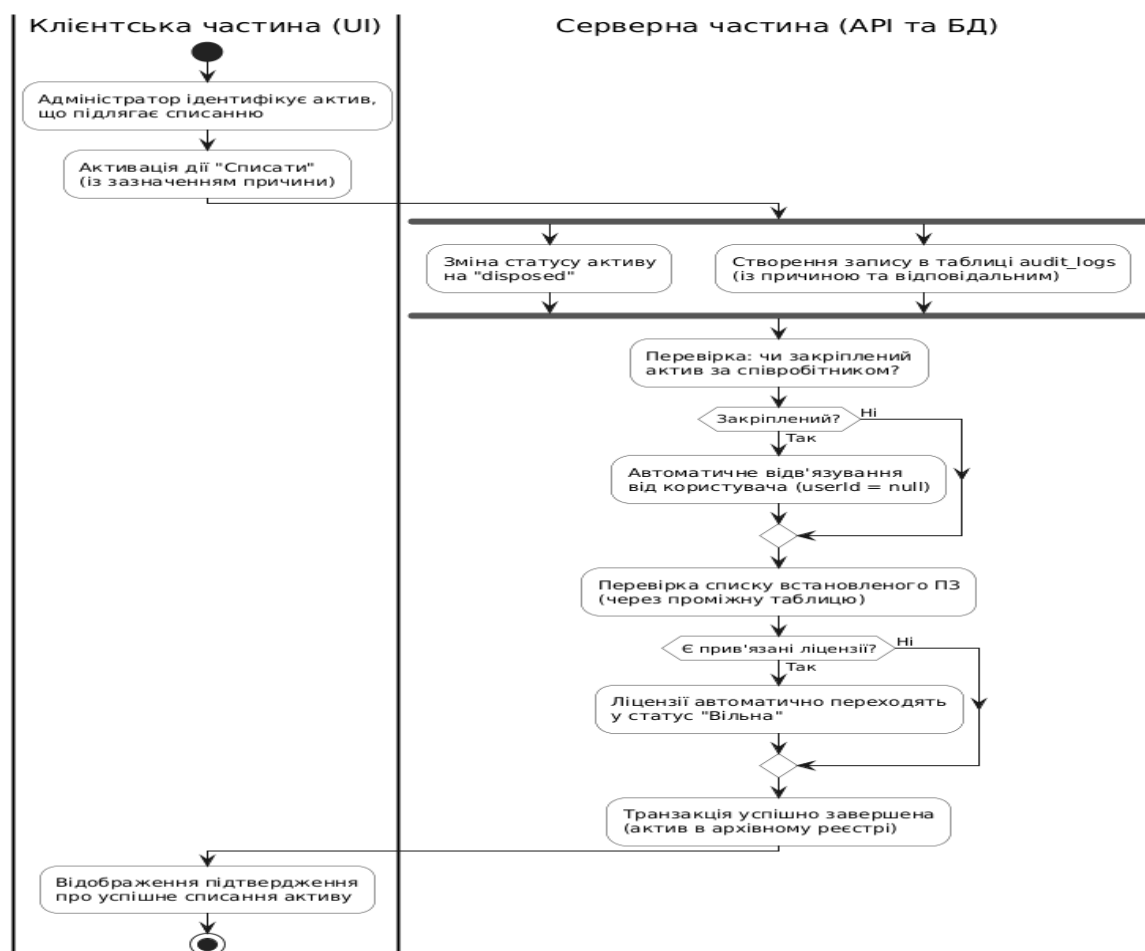


Рис. 2.13 – Діаграма активності «Процес списання активу»

Процес списання обладнання в системі максимально автоматизований та передбачає виконання каскаду пов'язаних операцій. Після того як адміністратор ідентифікує актив, що підлягає списанню (через закінчення терміну експлуатації або критичну поломку), та ініціює відповідну дію, сервер розпочинає транзакцію. Насамперед статус пристрою змінюється на «disposed», а в таблиці audit_logs генерується детальний запис із зазначенням причини та особи, відповідальної за списання.

Далі система самостійно перевіряє наявність зв'язків: якщо актив був закріплений за певним співробітником, він миттєво відв'язується шляхом очищення поля `userId`. Одночасно з цим сервер аналізує реєстр встановленого на пристрої програмного забезпечення і автоматично переводить усі пов'язані ліцензії у статус «Вільна», уможливаючи їх подальше використання на інших робочих станціях. По завершенню цих кроків адміністратор отримує сповіщення про успішне виконання транзакції, а сам актив переміщується до архівного реєстру.

Впровадження такого комплексного алгоритму демонструє високий загальний рівень автоматизації розробленої платформи та безперечну надійність її внутрішніх механізмів контролю. Такий архітектурний підхід практично зводить до мінімуму ризик виникнення критичних людських помилок під час ведення обліку. У результаті компанія отримує гарантію прозорого управління дороговартісною корпоративною ІТ-інфраструктурою та ефективного уникнення зайвих фінансових витрат.

Висновки до другого розділу

У другому розділі було успішно виконано комплексне проєктування програмної архітектури розроблюваної інформаційної системи. Цей багатогранний процес охопив усі найважливіші етапи: від побудови високорівневої моделі хмарної інфраструктури до глибокої деталізації алгоритмів моніторингу життєвого циклу ІТ-активів. Такий системний підхід дозволив сформувати надійний структурний фундамент для подальшої практичної реалізації вебплатформи.

За результатами побудови діаграми розгортання було обґрунтовано безсерверний (Serverless) підхід до організації інфраструктури з використанням хмарних рішень Vercel [1], Neon [2] та Cloudflare. Таке архітектурне рішення гарантує високу відмовостійкість системи, швидку доставку контенту й незалежне масштабування клієнтської та серверної частин у межах технологічного стеку Next.js [20]. У межах проєктування також проведено формалізацію рольової моделі та детальне моделювання варіантів використання

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

системи. У результаті було визначено чотирьох незалежних акторів (Гість, Співробітник, Технічний спеціаліст, Адміністратор), закріплено їхні права доступу згідно з моделлю RBAC та розроблено специфікації для ключових прецедентів, що визначають сувору логіку взаємодії користувача з платформою.

Для програмної реалізації побудовано об'єктно-орієнтовану структуру із застосуванням класичних інженерних патернів проєктування (MVC, Factory, Singleton, Proxy/Decorator). Водночас фізичну організацію коду фронтенд-частини структуровано за методологією Feature-Sliced Design (FSD) [6] для забезпечення ізольованого розвитку модулів обліку та аналітики. Важливим етапом стала розробка архітектури гібридної бази даних на основі PostgreSQL [22] з використанням Prisma ORM [23]. Вона вдало поєднує суворі реляційні зв'язки для обліку матеріальних цінностей і ліцензій із використанням типу даних JSONB для збереження динамічних технічних специфікацій. Це забезпечує високу гнучкість системи без жодної втрати транзакційної цілісності.

Насамкінець було змодельовано динамічну поведінку платформи за допомогою діаграм активності, послідовності та кооперації. Це дало змогу деталізувати бізнес-логіку додавання активів, управління ліцензіями програмного забезпечення й багатоетапного процесу списання обладнання, а також чітко візуалізувати хронологію передачі повідомлень між клієнтом, API-маршрутизаторами та базою даних.

Розроблені в рамках цього розділу логічні та структурні моделі повністю задовольняють усі вимоги затвердженого технічного завдання. Створені UML-артефакти формують вичерпну проєктну документацію, необхідну для успішного переходу до подальших стадій розробки. Зокрема, вони стануть надійною базою для етапу безпосередньої фізичної реалізації програмного продукту, який детально описується у наступному розділі кваліфікаційної роботи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ, ІНТЕРФЕЙС ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ІТ-АКТИВАМИ

3.1 Порядок встановлення та налаштування параметрів системи

Інформаційна система моніторингу та управління ІТ-активами підприємства розгортається у розподіленому хмарному середовищі, а процес від коміту розробника до появи оновленої версії на виробничому сервері (production) є повністю автоматизованим. Архітектура системи базується на Full-stack фреймворку Next.js [20], що дозволяє об'єднати клієнтський інтерфейс та серверні API-маршрути в межах єдиного оптимізованого рішення. Для публікації змін розробнику достатньо інтегрувати перевірений код до основної гілки репозиторію, решту процесів виконує конвеєр безперервної інтеграції та доставки (CI/CD [11, 15]).

Для забезпечення максимальної продуктивності та гнучкості розгортання, розроблена система інтегрує чотири спеціалізовані хмарні служби (табл. 3.1). Кожна з цих сучасних служб відповідає за окремий, суворо ізольований аспект роботи загальної інфраструктури. Такий архітектурний підхід гарантує, що будь-який компонент здатен стабільно функціонувати та масштабуватися абсолютно незалежно від інших модулів під час пікових навантажень.

Таблиця 3.1

Хмарна інфраструктура системи

Служба	Платформа	Призначення
Vercel Cloud Platform	Vercel Inc.	Хостинг Full-stack застосунку Next.js (Рендеринг інтерфейсу та виконання Serverless API)
Neon Cloud Database	Neon	Керована реляційна база даних PostgreSQL з підтримкою гібридних JSONB-сховищ
Cloudflare CDN/WAF	Cloudflare	Проксі-сервер, оптимізація кешування запитів, захист від DDoS-атак та шифрування трафіку

GitHub Actions	GitHub	Реєстр вихідного коду та платформа автоматизації CI/CD конвеєра
----------------	--------	---

Клієнтські інтерфейси функціонують в умовах суворої ізоляції. Будь-які прямі транзакції з браузера користувача до бази даних PostgreSQL [22] фізично заблоковані на рівні хмарної мережі. Єдиною дозволеною точкою взаємодії виступають захищені Serverless API-маршрути (Next.js Route Handlers).

У цій архітектурі саме безсерверні функції беруть на себе всю складну оркестрацію потоків даних та виконання бізнес-логіки. Вони безпечно взаємодіють із реляційними таблицями бази даних, гарантуючи збереження транзакційної цілісності. Крім того, ці функції ефективно обробляють специфічні вкладені об'єкти формату JSONB через потужний абстрактний шар Prisma ORM [23].

Маршрут доставки готового коду проходить через захищений репозиторій GitHub. Під час збірки на платформі Vercel [1] проєкт проходить процес суворої компіляції. Замість важких Docker-образів [7] для виробничого середовища, Next.js використовує процес оптимізованої статичної та серверної компіляції (next build). Вихідний код на TypeScript [10, 29] транспілюється у високоефективний JavaScript-код [8, 21], мінімізується, розбивається на окремі чанки (code splitting) та пакується в ізольовані Serverless-функції AWS Lambda, що керуються платформою Vercel [1]. Це мінімізує час відгуку системи та забезпечує миттєве масштабування при пікових навантаженнях на реєстр активів.

Таблиця 3.2

Стадії конвеєра автоматизації

Стадія	Завдання	Зміст процесу
lint	npm run lint	Статичний аналіз коду (ESLint) та перевірка суворої типізації TypeScript (tsc)
test	npm test	Виконання модульних та інтеграційних тестів бізнес-логіки та React-компонентів

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

build	npm run build	Оптимізована компіляція Next.js застосунку, генерація серверних функцій та статичних сторінок
deploy	Vercel Deployment	Безшовне оновлення робочого оточення на Vercel яке автоматично ініціюється через Webhook після успішних перевірок у GitHub Actions.

Чутливі параметри конфігурації – рядок підключення до бази даних PostgreSQL, секретні ключі авторизації NextAuth [19], токени доступу до хмари Cloudflare категорично не зберігаються у відкритому коді. Вони безпечно фіксуються у вигляді секретів (Variables/Secrets) безпосередньо у панелях керування Vercel [1] та GitHub (табл. 3.3). Під час виконання програми вони стають доступними виключно через змінні оточення process.env.

Таблиця 3.3

Параметри конфігурації виробничого середовища

Параметр	Призначення
DATABASE_URL	Зашифрований рядок підключення з автентифікацією до хмарної бази PostgreSQL на Neon
NEXTAUTH_SECRET	Криптографічна сіль (хеш) для безпечного шифрування сесій користувачів та JWT-токенів
NEXT_PUBLIC_APP_URL	Базова URL-адреса системи для валідації CORS запитів та перенаправлень OAuth
GOOGLE_CLIENT_SECRET	Ключ інтеграції з сервісом корпоративної авторизації співробітників Google OAuth 2.0

Для зручної розробки та безпечного локального тестування система може запускатися повністю автономно без прямої залежності від зовнішніх хмарних провайдерів. Для забезпечення такого режиму роботи розробнику достатньо створити локальний файл конфігурації .env із базовими змінними оточення, які необхідні для повноцінного функціонування бекенду. Після цього ініціюється повністю ізольоване контейнеризоване середовище за допомогою популярного інструменту Docker Compose [7]. Такий підхід гарантує абсолютну ідентичність середовищ розробки та надзвичайно швидке розгортання робочого місця для нових учасників інженерної команди. Детальний опис локальних мікросервісів,

які автоматично піднімаються в рамках цього конфігураційного процесу, наведено у табл. 3.4.

Таблиця 3.4

Сервіси локального середовища розробника

Контейнер / Процес	Технологія / Образ	Порт	Призначення
asset-app	Next.js (Development)	3000	Локальний сервер інтерфейсу та API-роутів
asset-db	postgres:15-alpine	5432	Локальна реляційна база даних PostgreSQL

Процес локального розгортання автоматично запускає скрипти наповнення бази даних тестовими даними (prisma db seed). Створюються базові ролі, категорії ІТ-активів (ноутбуки, монітори, ліцензії) та тестові записи. Локальна конфігурація примусово ізолює розробника від продакшену, спрямовуючи завантажені файли у локальну директорію public/uploads/. Очищення середовища та скидання бази даних до початкового стану виконується однією командою `docker [7] compose down -v && npx prisma db push --force-reset`

3.2 Структура інтерфейсу інформаційної системи моніторингу та управління ІТ-активами

Користувацький інтерфейс системи спроектовано за сучасними архітектурними принципами Single Page Application (SPA) у межах екосистеми Next.js App Router. Візуальна складова платформи побудована із використанням потужної бібліотеки стилів Tailwind CSS [28] та набору готових компонентів Shadcn UI [27]. Такий технологічний стек гарантує високу швидкість відгуку інтерфейсу, уніфікує дизайн-систему та суттєво знижує когнітивне навантаження на кінцевого користувача.

Під час першого входу в систему користувача зустрічає сучасний мінімалістичний екран автентифікації, розроблений з урахуванням найкращих практик UX/UI-дизайну. Його центральним функціональним елементом є зручна форма введення корпоративних облікових даних, яка захищена від

несанкціонованого доступу. Зовнішній вигляд цієї стартової сторінки наочно продемонстровано на рис. 3.1.

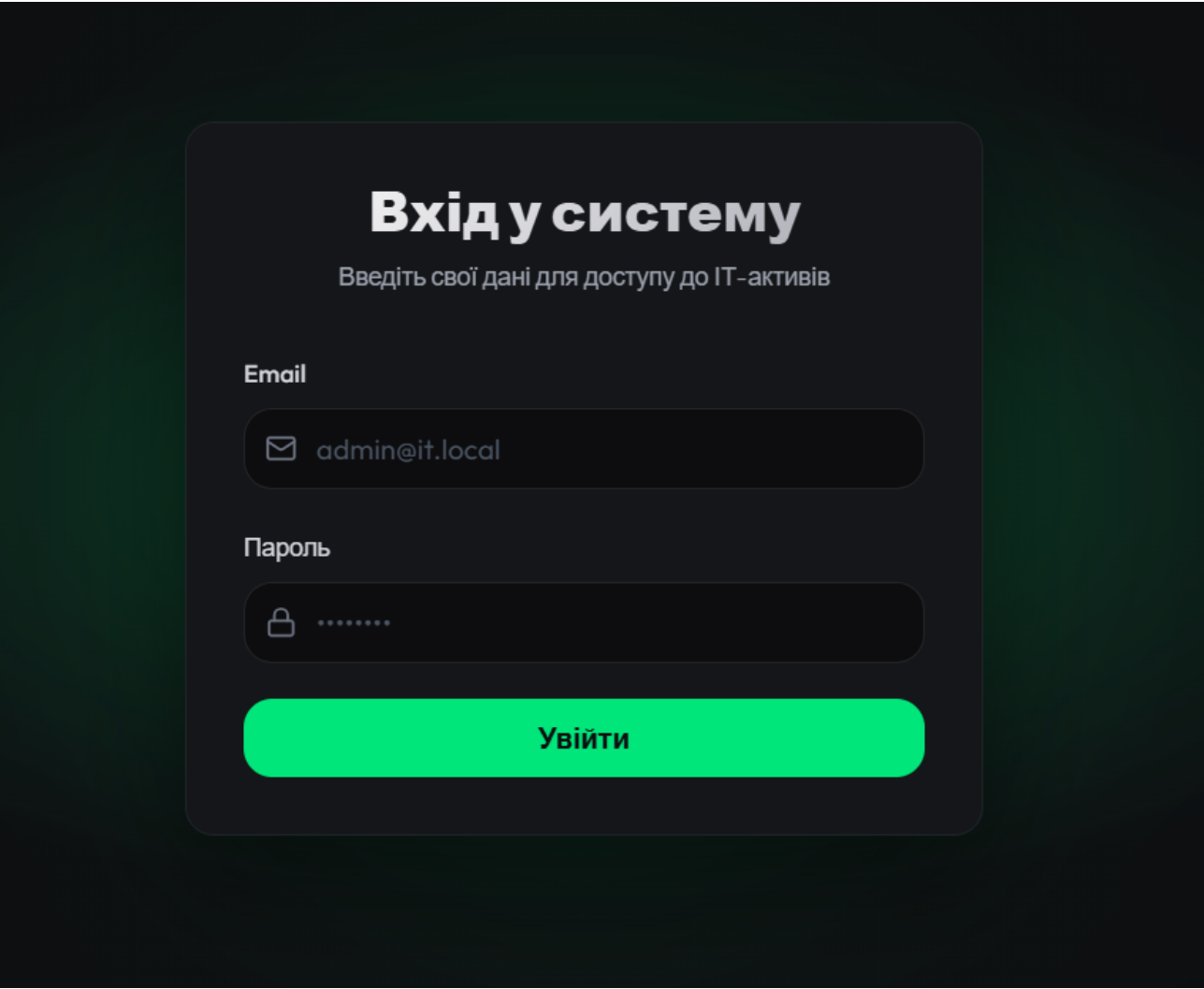


Рис. 3.1 – Сторінка логіну на сайт

Після успішної верифікації токена через модуль NextAuth [19], система автоматично аналізує роль користувача в базі даних згідно з моделлю RBAC. На основі цих авторизаційних даних відбувається захищене перенаправлення (маршрутизація) клієнта до відповідної робочої області. Зокрема, Адміністратор та Технічний спеціаліст отримують повний доступ до глобальної аналітичної панелі, тоді як звичайний співробітник спрямовується в обмежений особистий кабінет для перегляду виключно закріпленої за ним техніки та корпоративних ліцензій.

Головний екран адміністратора виступає центральним аналітичним хабом для моніторингу ІТ-інфраструктури підприємства. У верхній частині головного екрана адміністратора розташовані інтерактивні аналітичні віджети, які за допомогою інтуїтивного кольорового кодування відображають ключові метрики ІТ-фонду підприємства. Зокрема, система виводить загальну кількість обладнання на балансі та частку техніки, що наразі активно експлуатується співробітниками. Водночас окремі віджети дозволяють оперативно відстежувати кількість активів, які передані на технічне обслуговування чи ремонт, а також статистику щодо пристроїв, які були остаточно списані та виведені з експлуатації (рис. 3.2).

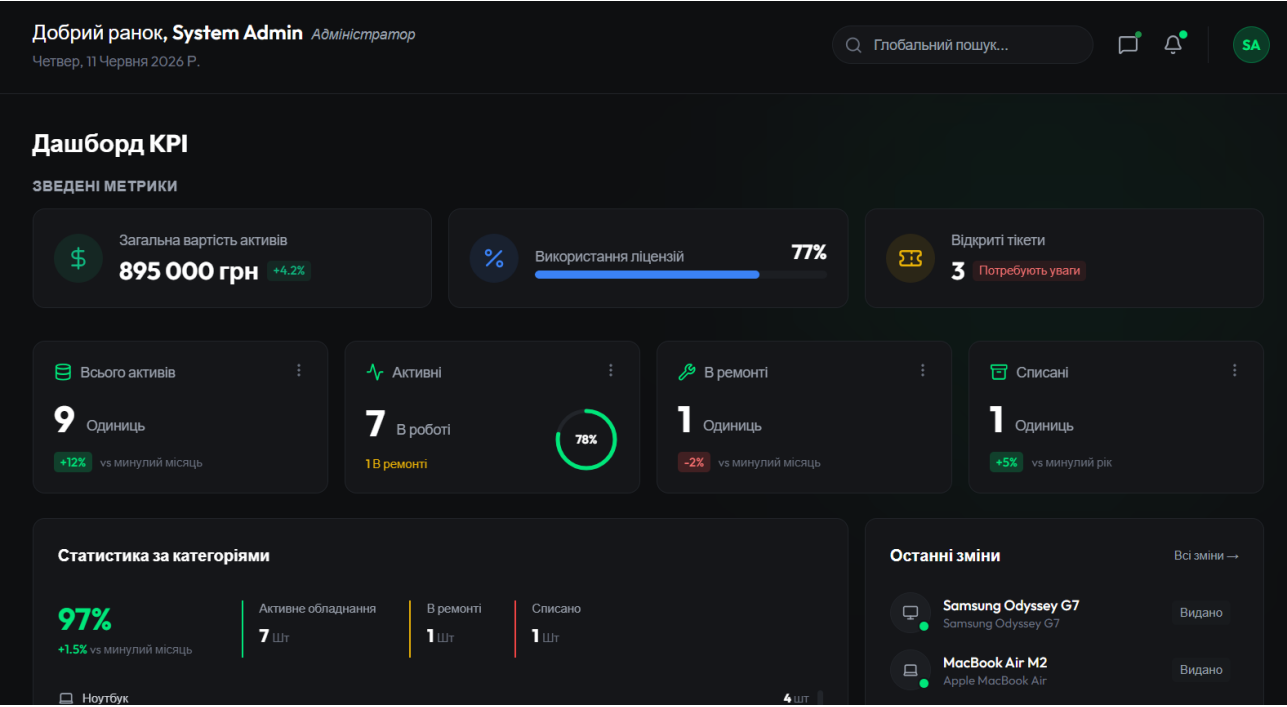


Рис. 3.2 – Головна сторінка

Безпосередньо під аналітичними картками розміщено інтерактивні графіки, які візуалізують розподіл активів за категоріями та поточними статусами експлуатації. Основна навігація системою здійснюється за допомогою фіксованої бічної панелі (Sidebar), що забезпечує швидкий доступ до ключових функціональних розділів платформи: «Дашборд», «Активи», «Ліцензії ПЗ», «Моніторинг», «Служба підтримки», «Планове ТО», «Звіти та відомості», «Співробітники» та «Налаштування».

Водночас глобальне керування сесією користувача винесено у верхню навігаційну панель (Header). Цей елемент інтерфейсу є багатофункціональним: у ньому інтегровано центр сповіщень, вікно виклику чат-бота, а також інтерактивний модуль профілю, який не лише відображає дані поточного облікового запису, а й містить елементи безпечного виходу із системи (рис. 3.3). Таке просторове розмежування навігаційних інструментів гарантує інтуїтивно зрозумілу та ергономічну взаємодію користувача з платформою.

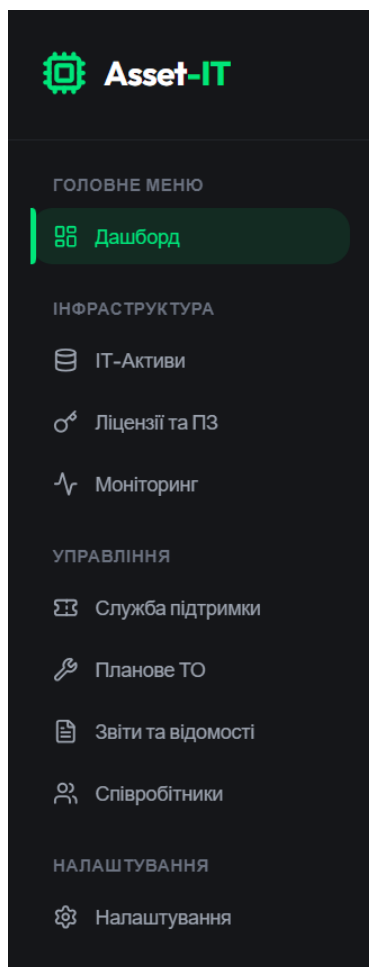


Рис. 3.3 – Бічна панель навігації (Sidebar)

Окрема сторінка «Профіль користувача» слугує багатофункціональним персональним кабінетом для кожного авторизованого співробітника. На цій панелі компактно відображається ключова особиста інформація, поточна системна роль (згідно з моделлю RBAC) та актуальні контактні дані. Окрім базових налаштувань, сторінка містить вичерпний перелік усіх ІТ-активів та програмних ліцензій, які наразі закріплені за даним обліковим записом (рис. 3.4).

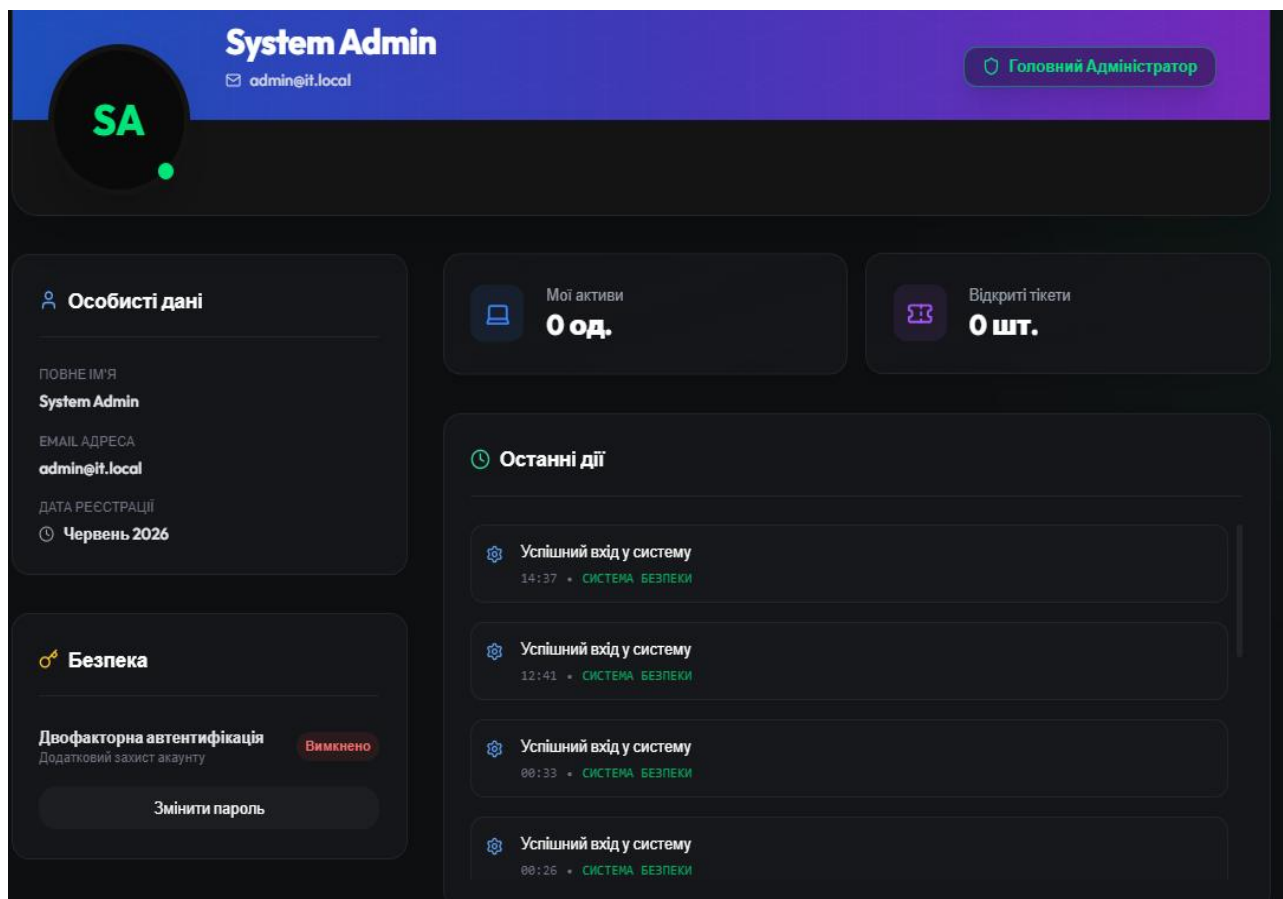


Рис. 3.4 – Профіль користувача

Основний інтерфейс управління наявною комп'ютерною технікою та периферією реалізовано у вигляді сучасної вискоєфективної таблиці даних (Data Table). Вона інтегрує у собі потужні інструменти взаємодії, зокрема підтримує розумну клієнтську пагінацію, багатовимірне сортування колонок та миттєвий швидкий пошук без перезавантаження сторінки. Уповноважений користувач може гнучко здійснювати пошук за унікальним серійним номером, внутрішнім інвентарним кодом або відповідальною матеріальною особою. Крім того, система дозволяє застосовувати складні комплексні фільтри за категоріями та статусами обладнання. Натиснувши цільову кнопку «Додати актив» адміністратор може ініціювати зручний сценарій додавання нового активу, а також безпечно видалення або глибоке редагування вже існуючого запису. Кожен окремий рядок таблиці є високоінформативним: він містить візуальні динамічні бейджі актуальних статусів пристрою та розгорнуту інформацію про поточного закріпленого за ним користувача (рис. 3.5).

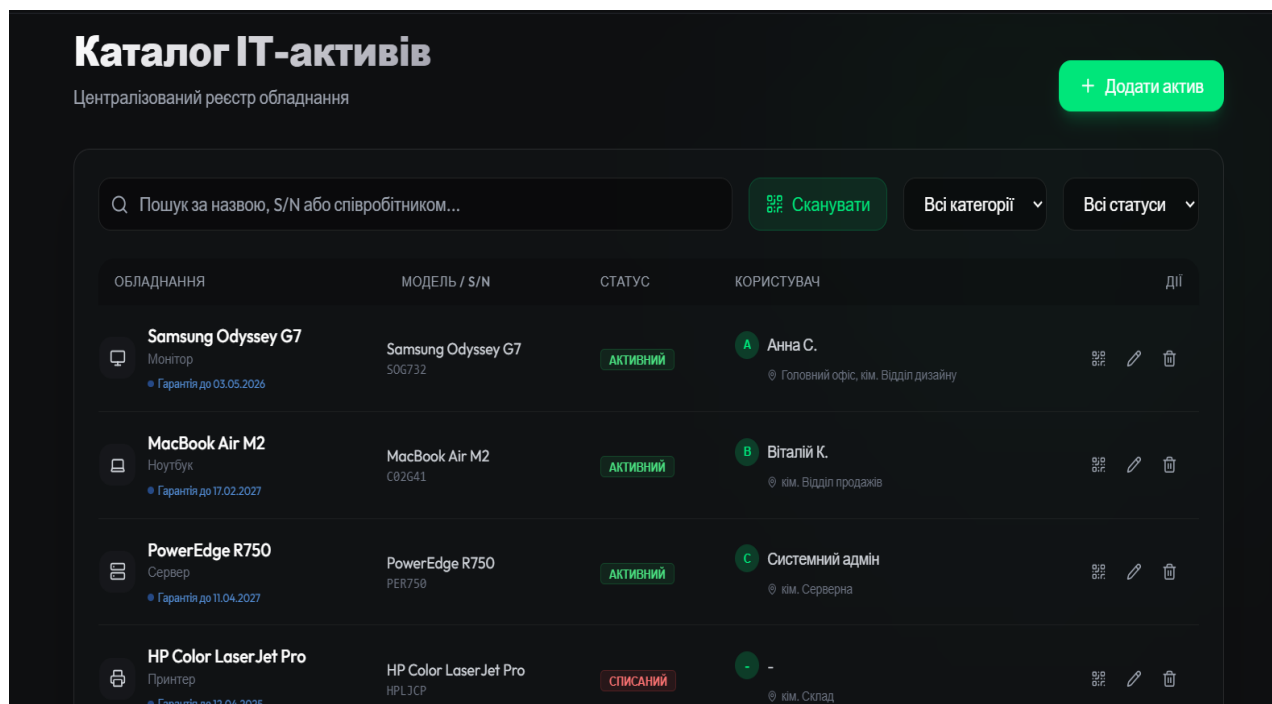


Рис. 3.5 – Сторінка ІТ-активів

Особливістю архітектури інтерфейсу є картка редагування характеристик обладнання. Виклик меню здійснюється натисканням на іконку «олівця». Форма редагування генерується динамічно, підлаштовуючись під тип активу, та дозволяє змінювати як базові параметри, так і специфічні технічні характеристики (JSONB), а також відстежувати історію закріплення за співробітниками.

При натисканні на кнопку «Зберегти зміни» клієнтська частина автоматично валідує введені дані та відправляє PATCH-запит, безпечно оновлюючи структуру в базі даних PostgreSQL. Фрагмент функціонального програмного коду, який реалізує алгоритм валідації та збереження динамічних специфікацій у форматі JSONB наведено в Додатку Б.5.

Інтерфейс ліцензійного модуля спрямований на жорсткий контроль корпоративного програмного забезпечення. Кожна картка ліцензії містить візуальний прогрес-бар (трекер) використання доступних місць, тип підписки, назву продукту та термін його дії. Реалізовано функціонал швидкого додавання нових ліцензійних ключів та деактивації прострочених контрактів (рис. 3.6).

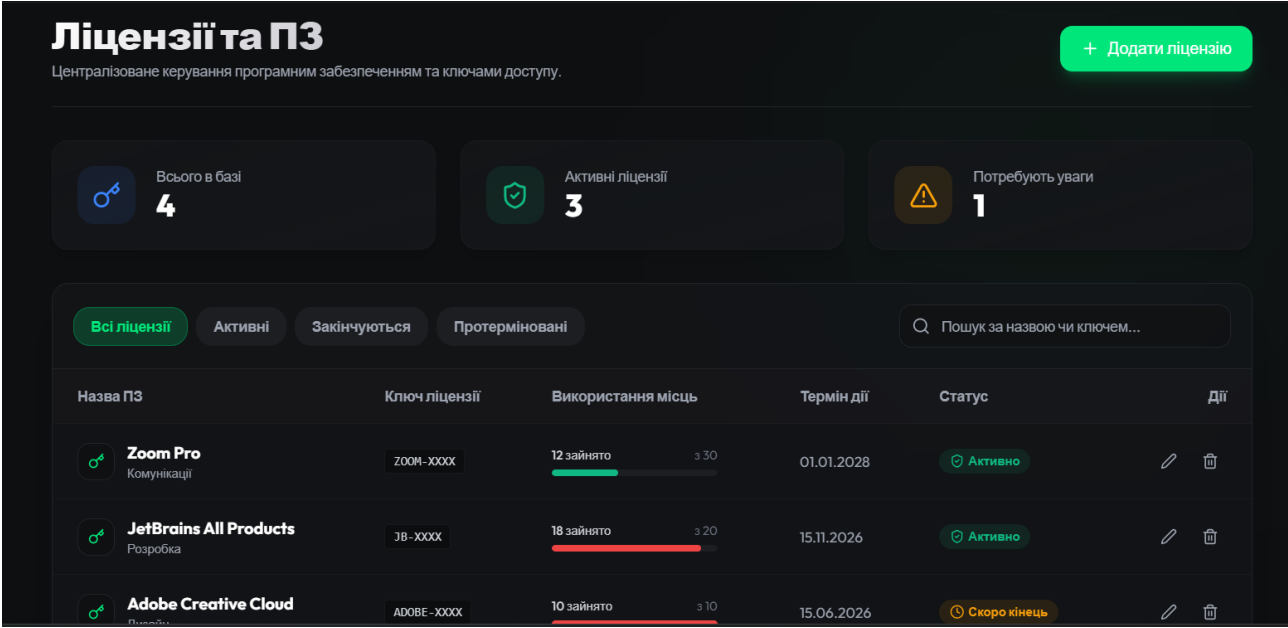


Рис. 3.6 – Сторінка ліцензії та ПЗ

Для задоволення потреб підприємства в офіційній документації було розроблено модуль «Генератор звітів». Його головною задачею є надання зручних інструментів для вивантаження даних системи у загальноприйнятих корпоративних форматах. На рисунку 3.7 зображено користувацький інтерфейс цього модуля з використанням зручної вкладкової структури (Tabs). Такий ергономічний дизайн дозволяє швидко перемикатися між різними типами звітів, не втрачаючи налаштувань.

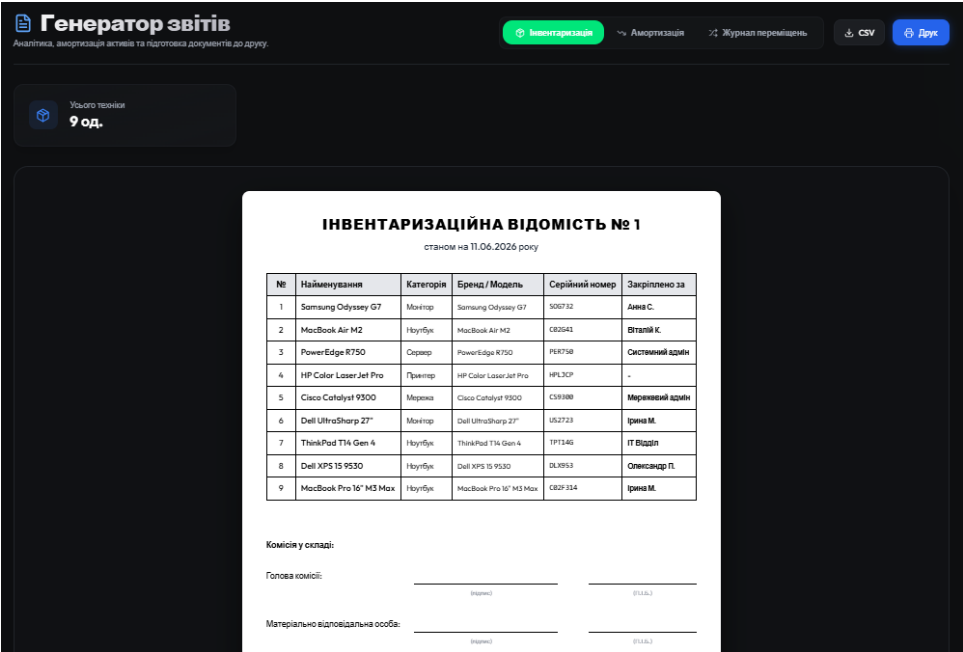


Рис. 3.7 – Сторінка генератора звітів (Інвентаризація)

Після налаштування необхідних параметрів користувач може згенерувати фінальний документ. Цей процес автоматизованого створення офіційної інвентаризаційної відомості відображено на рисунку 3.8. Система успішно формує стандартизований витяг у форматі PDF, який містить перелік обладнання та підходить для виведення на друк і підпису.

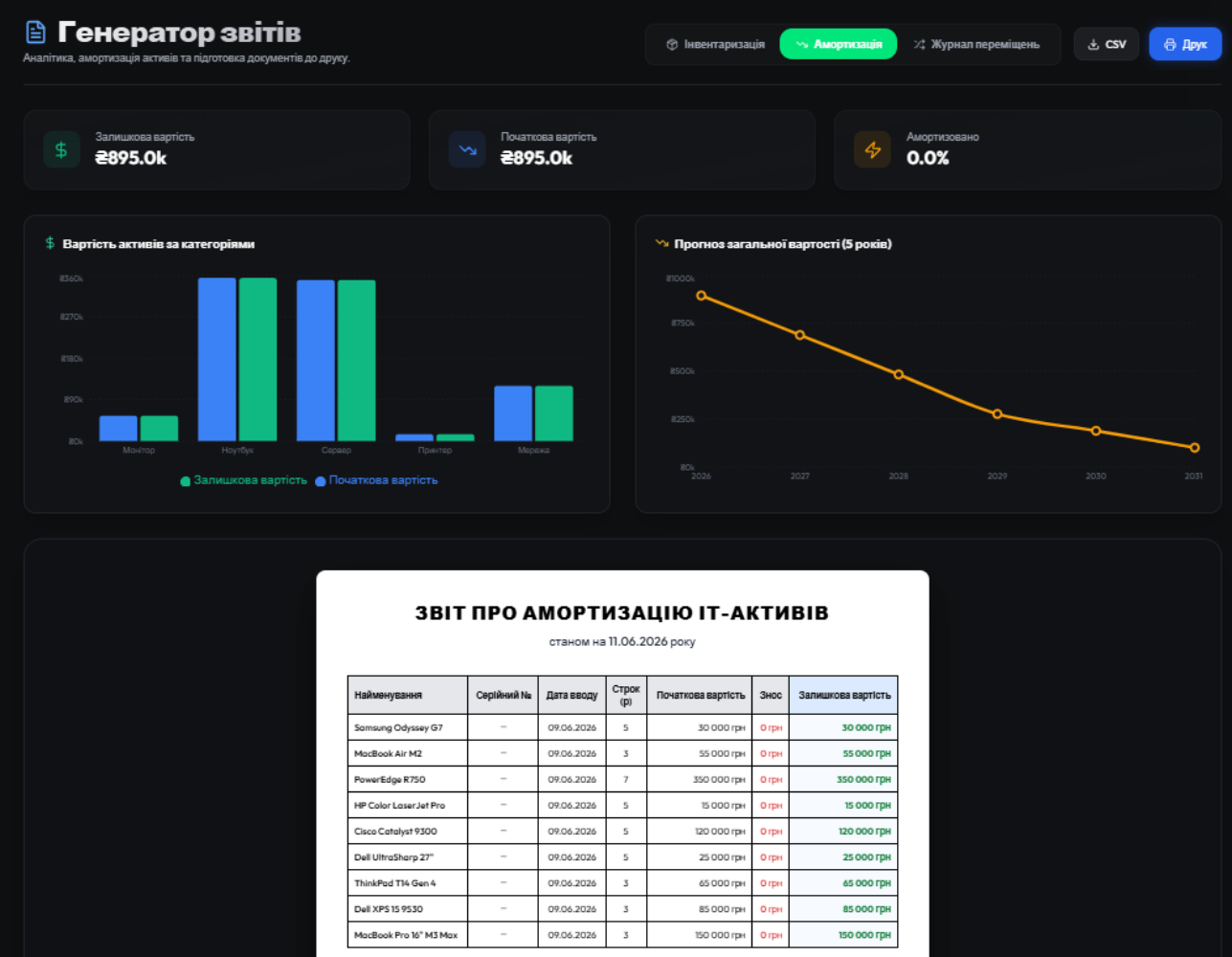


Рис. 3.8 – Сторінка генератора звітів (Амортизація)

Окрім друкованих форм, система підтримує масову роботу з електронними таблицями. На рисунку 3.9 наочно продемонстровано інтерфейс імпорту та експорту масивів даних. Використання формату Excel забезпечує безшовну інтеграцію розробленої платформи з іншими бухгалтерськими програмами та дозволяє швидко оновлювати реєстр активів.

Генератор звітів

Аналітика, амортизація активів та підготовка документів до друку.

Інвентаризація

Амортизація

Журнал переміщень

CSV

Друк

Всього переміщень

6 записів

ЖУРНАЛ ПЕРЕМІЩЕНЬ ІТ-АКТИВІВ

станом на 11.06.2026 року

Дата / Час	Актив	Передано від	Передано до	Зміна локації
10.06.2026 03:16	Samsung Odyssey G7	Олександр П.	Анна С.	Без зміни
10.06.2026 03:16	Samsung Odyssey G7	Анна С.	Олександр П.	Без зміни
09.06.2026 16:57	Samsung Odyssey G7	ІТ Відділ	Анна С.	Без зміни
09.06.2026 16:56	Samsung Odyssey G7	Анна С.	ІТ Відділ	Без зміни
09.06.2026 16:55	Samsung Odyssey Samsung Odyssey	Анна С.	Віталій К.	Без зміни
09.06.2026 16:53	Samsung Odyssey Samsung Odyssey	Не призначено	Анна С.	Без зміни

Рис. 3.9 – Сторінка генератора звітів (Журнал переміщень)

Helpdesk спроектована у вигляді інтерактивної Kanban-дошки для управління інцидентами. Якщо у співробітника виникає апаратна проблема, він створює тикет-заявку, вказуючи опис проблеми, своє ПІБ та рівень критичності. Заявки відображаються у вигляді карток, які технічні спеціалісти можуть перетягувати або переводити між колонками станів: «Відкриті заявки», «В роботі», «Вирішені», що забезпечує прозорість процесу обслуговування техніки (рис. 3.10).

Служба підтримки

Управління інцидентами під контролем: System Admin

+ Створити заявку

Пошук за назвою, користувачем чи ID...

ВІДКРИТИХ 3

В МЕЖАХ SLA 0

ЗАГРОЗА SLA 0

ПРОСТРОЧЕНО 3

ВІДКРИТІ ЗАЯВКИ 2

В РОБОТІ 1

ВИРІШЕНІ 1

27f48528-87ab-4f65-a85d-8ae8d53fbc9c КРИТИЧНИЙ

Оновлення SSL сертифікату

SLA: ПРОСТРОЧЕНО

В роботу →

Сисадмін 02:57

a3f4fb4f-1533-4bb8-851d-e82e474d78d7 КРИТИЧНИЙ

Доступ до бази даних

SLA: ПРОСТРОЧЕНО

Вирішено ✓

Анна С. 02:57

00c385d2-f451-49e4-ba7c-ec72814dbb6c НИЗЬКИЙ

Не працює принтер на 3 поверсі

SLA: Виконано

Перевірити

Ірина М. 02:57

Рис. 3.10 – Сторінка служби підтримки

Розділ «Моніторинг системи» призначений для відстеження стану мережевої інфраструктури та серверів. У реальному часі відображається кількість активних вузлів, їхній мережевий статус (Online/Offline) та телеметричні метрики. Інтерфейс дозволяє динамічно додавати нові вузли для моніторингу або редагувати параметри існуючих.

Окрім базової мережевої перевірки доступності вузлів (класичних ICMP-пінгів), архітектура розробленого модуля моніторингу передбачає глибокий збір розширених телеметричних даних з кожного підключеного сервера. Система періодично агрегує ключові технічні показники, зокрема поточний рівень завантаженості центрального процесора (CPU), відсоток використання вільної оперативної пам'яті (RAM) та загальний час безперебійної роботи системи з моменту останнього перезавантаження (Uptime). Візуалізація цих критичних метрик в інтерфейсі реалізована за допомогою інтуїтивно зрозумілих прогрес-барів та динамічного кольорового кодування. Так, зелений колір сигналізує про стабільну роботу обладнання в межах норми, жовтий заздалегідь попереджає про підвищене навантаження або досягнення порогових значень, а червоний колір миттєво інформує адміністратора про критичні збої або повну втрату зв'язку з віддаленим вузлом.

Для максимальної зручності адміністрування розгалужених корпоративних мереж, до складу модуля моніторингу також тісно інтегровано інтерактивну панель «Системний журнал» (System Log). Цей компонент безперервно фіксує абсолютно всі автоматичні спроби діагностики серверів, що виконуються у фоновому режимі, та виводить точну хронологію виявлених мережевих аномалій. Завдяки цій аналітичній інформації ІТ-інженери можуть не лише сухо констатувати факт падіння певного сервера, але й ретроспективно проаналізувати його технічний стан за кілька хвилин до збою. Такий проактивний підхід суттєво зменшує час реакції технічної підтримки на інциденти та запобігає тривалим простоям корпоративних сервісів. Увесь цей комплексний інструментарій, що гармонійно поєднує візуалізацію телеметрії та

системний журнал, об'єднано на єдиній інформаційній сторінці моніторингу (рис. 3.11).

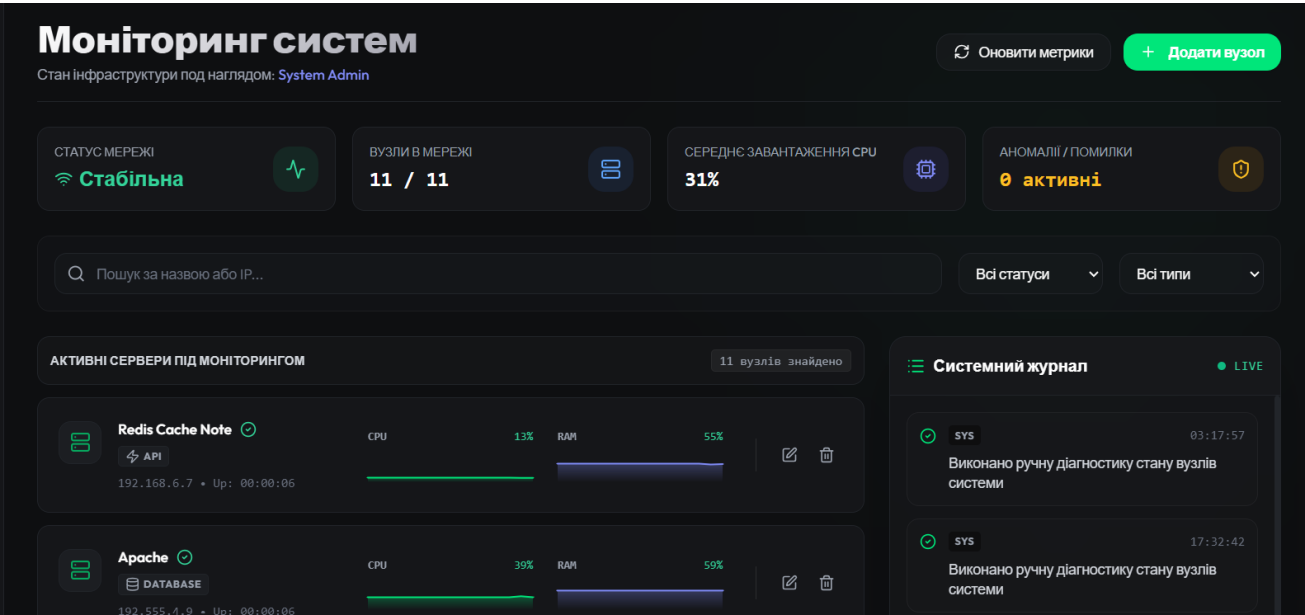


Рис. 3.11 – Сторінка моніторингу системи

Модуль «Співробітники» є директорією персоналу компанії. У вигляді карток або таблиці відображається список працівників із зазначенням їхньої посади, відділу та контактних даних. Після натискання на карту співробітника відкривається детальне зведення про всю апаратну техніку та ліцензії, які за ним наразі закріплені. Інтерфейс підтримує операції додавання нових працівників та їх видалення при звільненні (рис. 3.12).

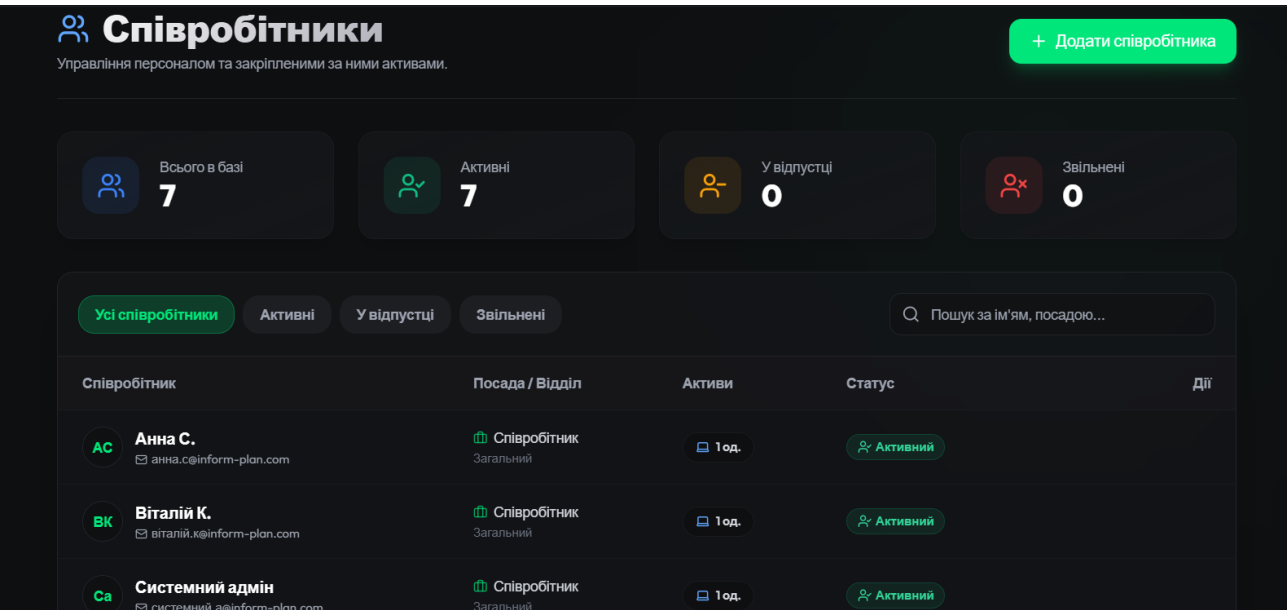


Рис. 3.12 – Сторінка співробітників

Спеціалізована секція «Налаштування» (Settings) використовує сучасну вкладкову (Tab) структуру для забезпечення максимально зручної навігації між ключовими категоріями конфігурації платформи: «Основні параметри», «Безпека та Дані», «Користувачі» та «Журнал аудиту». Такий підхід дозволяє адміністратору миттєво знаходити потрібні інструменти без необхідності переходу між різними сторінками. На першій головній вкладці зосереджені критично важливі глобальні параметри профілю організації, базові регіональні налаштування (включно з основною валютою та префіксом інвентарних номерів), а також розширена конфігурація зовнішніх інтеграцій та автоматичних сповіщень (рис. 3.13).

Рис. 3.13 – Сторінка Налаштування (Основні параметри)

Своєю чергою, у вкладці «Безпека та Дані» глибоко інтегровано потужні інструменти для управління цілісністю системи. Зокрема, адміністратор має можливість створювати резервні копії бази даних або за критичної необхідності виконувати повне скидання налаштувань (Factory Reset). Крім того, передбачено спеціальний перемикач для увімкнення «Режиму технічного обслуговування»

(Maintenance Mode), який тимчасово закриває доступ до платформи всім звичайним працівникам, залишаючи права виключно за системними адміністраторами (рис. 3.14).

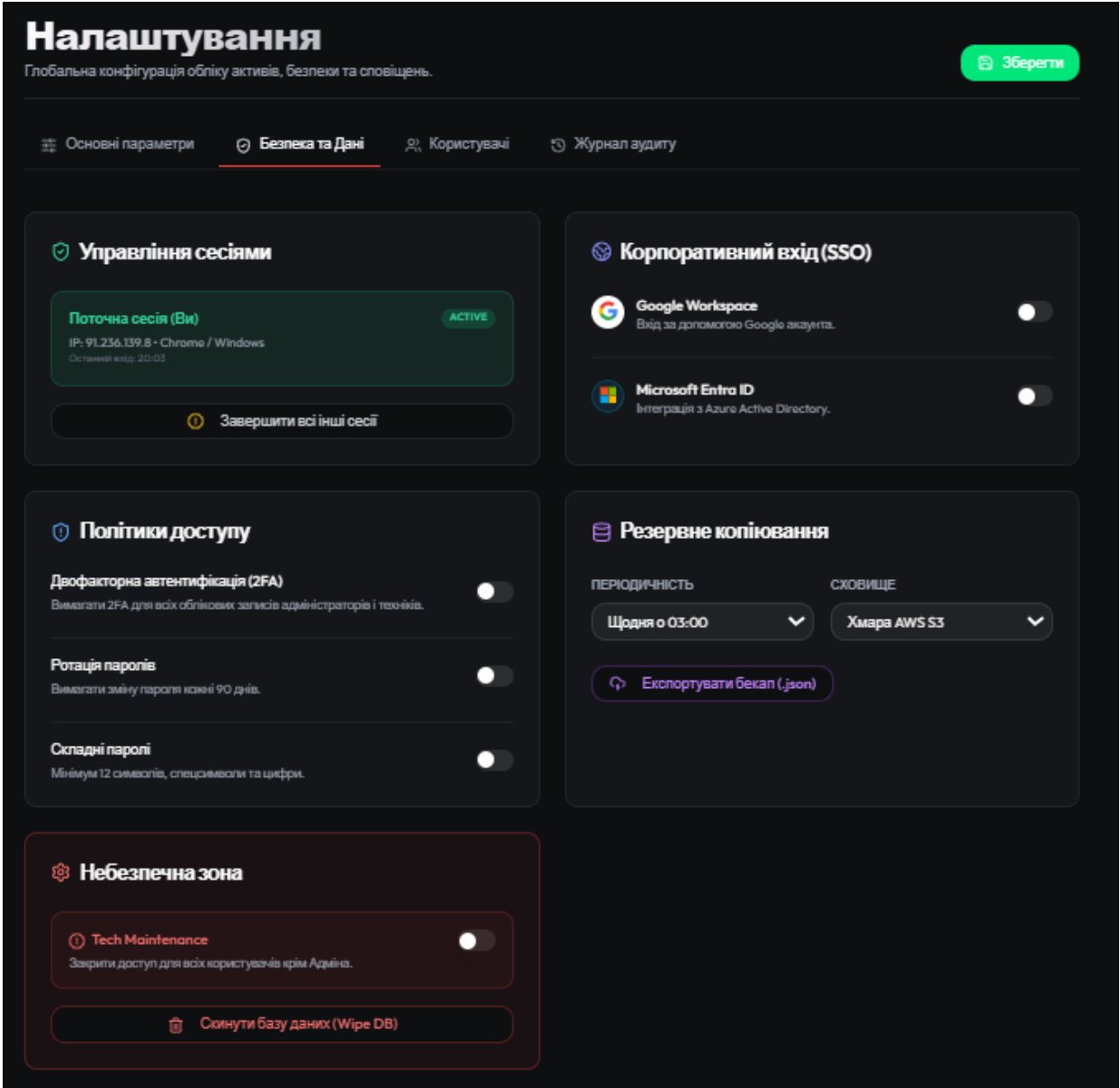


Рис. 3.14 – Сторінка налаштування (безпека та дані)

У вкладці «Користувачі» реалізовано панель управління доступом (Access Management). Адміністратор може переглядати список авторизованих сесій та гнучко змінювати системні ролі (Гість, Співробітник, Технічний спеціаліст, Адмін) для існуючих акаунтів. Додатково передбачено можливість миттєвого призупинення доступу (блокування) для звільнених або підозрілих користувачів, що гарантує високий рівень захисту корпоративних даних (рис. 3.15).

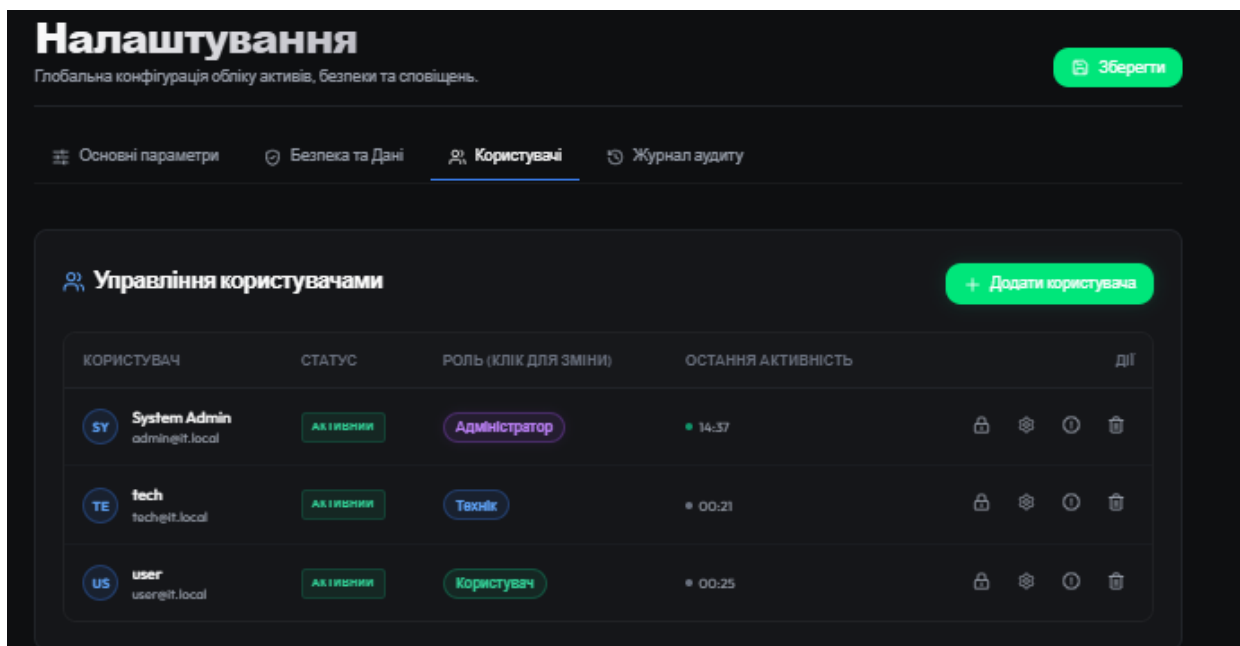


Рис. 3.15 – Сторінка налаштування (користувачі)

Вкладка «Журнал аудиту» (Audit Log) слугує головним інструментом для моніторингу безпеки та містить суворо незмінну хронологічну таблицю всіх критичних подій корпоративної системи. У цьому захищеному реєстрі автоматично та безперервно фіксуються такі важливі операції, як створення нових активів, зміна статусів обладнання, масовий імпорт даних, редагування профілів та незворотне видалення записів. Для кожної зафіксованої події безпомилково відображається точний час за сервером (із точністю до мілісекунд), категорія об'єкта, тип виконаної дії та унікальний ідентифікатор користувача або його системна роль. Крім того, інтерфейс журналу оснащений вбудованими інструментами розширеного пошуку та фільтрації, які дозволяють адміністраторам безпеки миттєво знаходити підозрілі активності за певним працівником або в заданому вузькому діапазоні дат. Такий суворий архітектурний підхід повністю унеможливорює приховування несанкціонованих дій з боку персоналу, відповідає сучасним стандартам корпоративної інформаційної безпеки та гарантує абсолютну прозорість і ретроспективну відстежуваність абсолютно всіх операцій у базі даних (рис. 3.16).

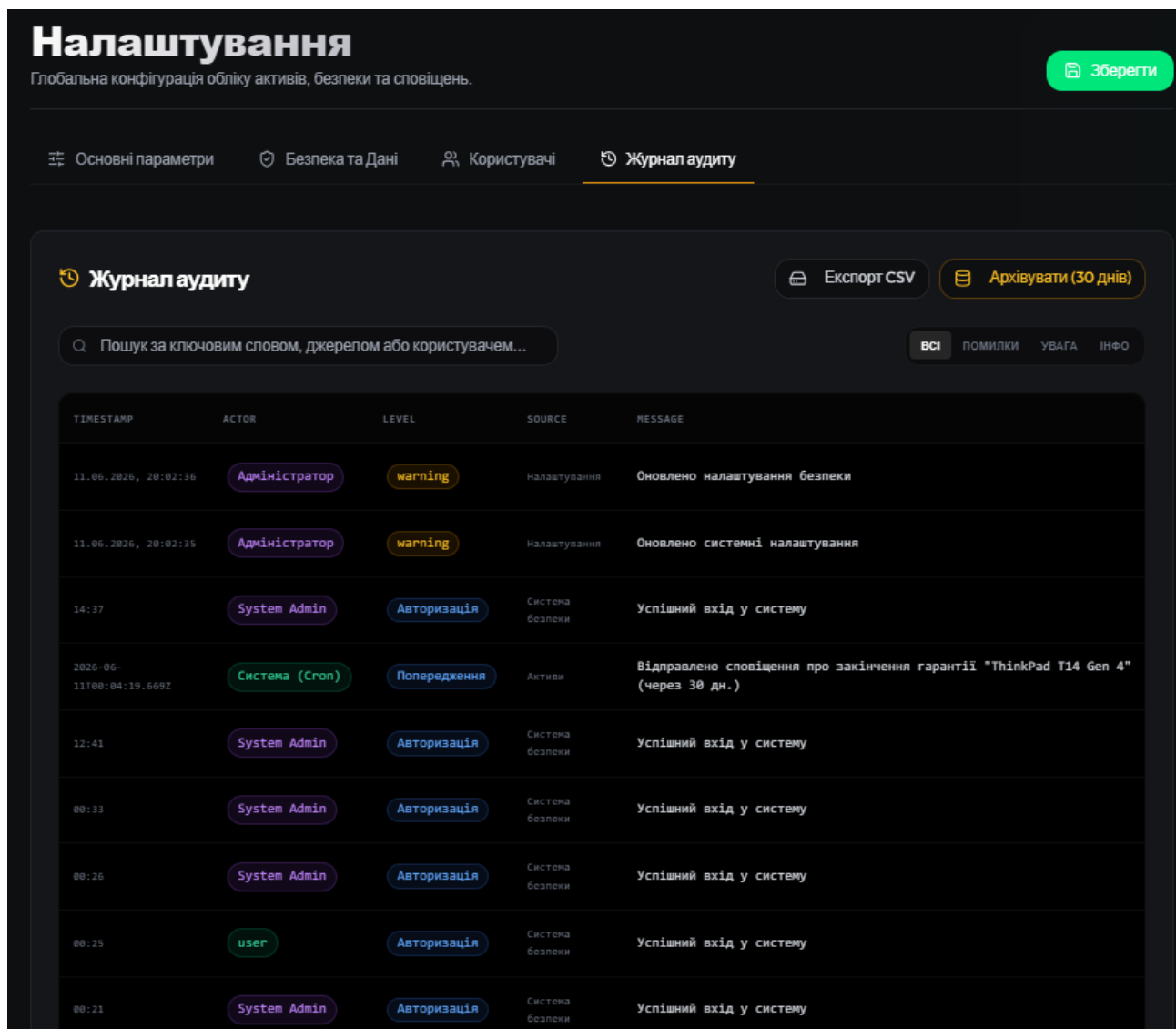


Рис. 3.16 – Сторінка налаштування (журнал аудиту)

Окремий розділ «Планове ТО» спеціалізований для ефективного управління життєвим циклом того обладнання, що потребує регулярного технічного обслуговування або наразі перебуває в активному ремонті. Сучасний інтерфейс цього модуля дозволяє адміністраторам створювати зручні графіки майбутніх профілактичних робіт та призначати відповідальних виконавців. Крім того, функціонал забезпечує можливість відстежувати поточний статус ремонту кожного окремого активу та детально фіксувати всю історію сервісного обслуговування для подальшої технічної та фінансової звітності (рис. 3.17).

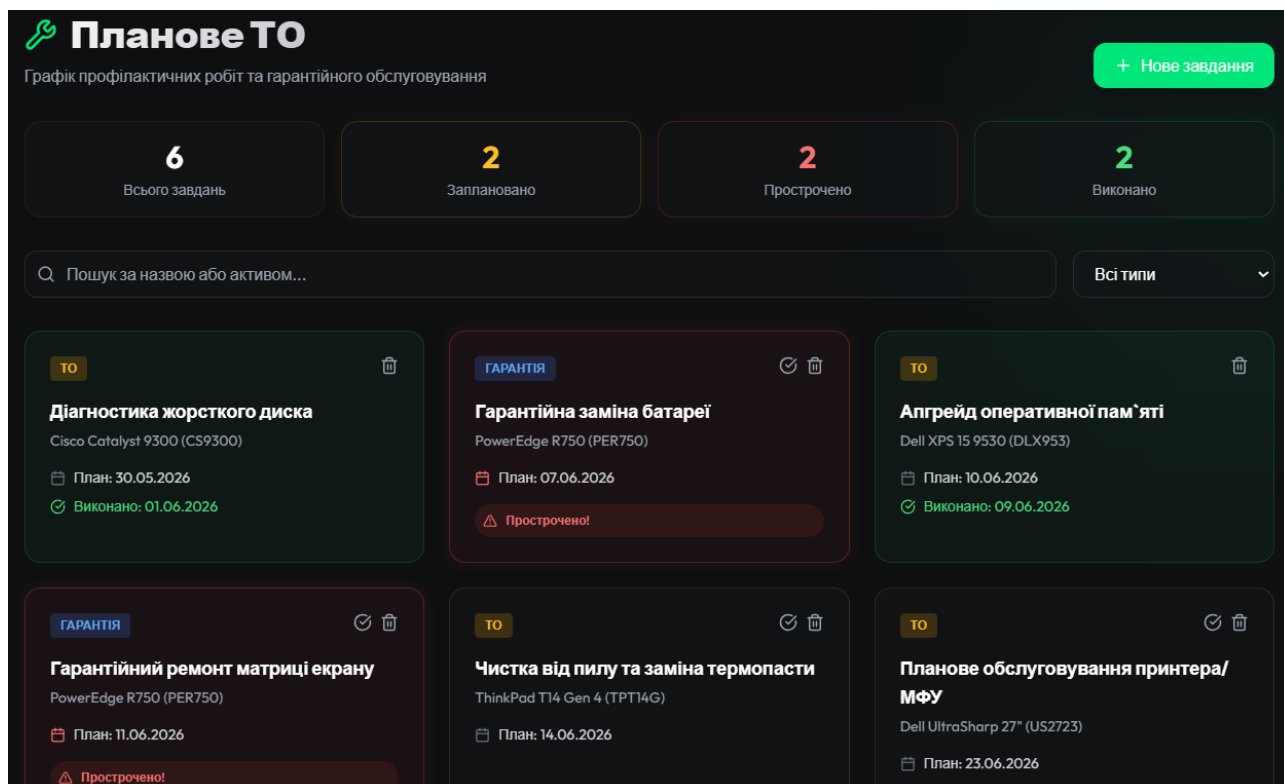


Рис. 3.17 – Сторінка Планове ТО

Таким чином, спроектований та реалізований користувацький інтерфейс повністю відповідає сучасним вимогам до високоякісних корпоративних інформаційних систем. Завдяки використанню компонентного підходу (React) та інтеграції бібліотек Tailwind CSS і Shaden UI, вдалося створити не лише візуально привабливий, але й ергономічний SPA-застосунок із миттєвим відгуком. Наскрізна реалізація моделі управління доступом на основі ролей (RBAC) забезпечила надійну безпеку та гнучку персоналізацію робочого простору для кожного типу користувачів: від звичайного співробітника до системного адміністратора. У комплексі розроблений функціонал дає змогу підприємству ефективно керувати життєвим циклом техніки, здійснювати безперервний моніторинг інфраструктури та формувати точну аналітичну звітність у межах консолідованого інформаційного простору підприємства.

3.3 Тестування роботи програмного продукту

Тестування розробленої системи управління ІТ-активами реалізовано у вигляді класичної багаторівневої піраміди тестування [4]. Цей архітектурний підхід охоплює ізольовані рівні перевірки та ефективно унеможливорює небажану

регресію бізнес-логіки під час майбутнього рефакторингу коду. Загалом інфраструктура покриває три ключові рівні: модульні тести бізнес-інваріантів (Jest [14]), інтеграційні тести API-маршрутів (Supertest) та комплексне поведінкове тестування користувацького інтерфейсу (React Testing Library [24, 25]).

```
Test Suites: 1 passed, 1 total
Tests:      6 passed, 6 total
Snapshots:  0 total
Time:       0.325 s
```

Рис. 3.18 – Результати виконання модульних тестів серверної частини

Доменні тести перевіряють інваріанти сутностей та суворі бізнес-правила системи без запуску HTTP-сервера чи рендерингу інтерфейсу, що забезпечує швидкість їхнього виконання. Вони гарантують, що критичні обмеження цілісності даних виконуються безпомилково на рівні сервісного шару.

Таблиця 3.5

Тести доменного рівня

Клас / Тестовий набір	Назва тесту	Опис перевірки
AssetLifecycleTests	Disposed_WithActiveMaintenance_ShouldThrow	Списання активу неможливе, якщо по ньому є відкрита заявка на ремонт.
LicenseServiceTests	AssignSeat_WhenZeroAvailable_ShouldReject	Система повинна блокувати закріплення ліцензії за активом, якщо всі місця (seats) вичерпано.
SpecValidationTests	ValidateSpecs_InvalidJSONBStructure_Throws	Перевірка валідації схеми: спроба записати невалідні поля в JSONB-специфікацію повертає помилку структури.
AuditLogTests	LogStatusChange_ShouldCreateRecord	Будь-яка зміна статусу активу повинна автоматично створювати незмінний запис у таблиці аудиту.

Для перевірки інтеграційної взаємодії HTTP-контролерів із базою даних використовується Supertest. Для забезпечення ізоляції тестів, перед кожним тестовим набором ініціюється чиста транзакція в тестовій базі даних PostgreSQL, яка повністю відкочується після завершення тесту. Це гарантує відсутність залишкових артефактів.

Розроблені тести комплексно перевіряють коректність повернення відповідних HTTP-статусів при різних сценаріях доступу. Також вони забезпечують сувору валідацію рольової моделі безпеки на серверному рівні. Окремим важливим етапом є тестування роботи складних агрегаційних SQL-запитів, які генеруються через клієнт Prisma (табл. 3.6).

Таблиця 3.6

Тестові класи API-контролерів

Тестовий набір	Кількість тестів	Сфера покриття та перевірки
asset.route.test.ts	22	Перевірка GET /api/assets з урахуванням фільтрації за категоріями, авторизація POST запитів для ролей Технічного спеціаліста та Адміністратора.
license.route.test.ts	16	Перевірка атомарності операцій виділення та відкріплення ключів ПЗ, оновлення лічильників ліцензійних місць.
auth.middleware.test.ts	12	Валідація JWT-сесій NextAuth, примусове блокування доступу користувачів із роллю Співробітника до адмін-панелі.

Тести клієнтської частини написані на базі фреймворку Jest [14] із використанням спеціалізованої бібліотеки React Testing Library [24]. Вони фокусуються виключно на поведінковому тестуванні інтерфейсу з точки зору реального кінцевого користувача. У рамках цих перевірок детально тестується коректність рендерингу динамічних форм, надійність захисту клієнтських маршрутів та загальна реакція інтерфейсу на різноманітні дії користувача (табл. 3.7).

```

Test Suites: 1 passed, 1 total
Tests:      5 passed, 5 total
Snapshots:  0 total
Time:       0.278 s

```

Рис. 3.19 – Результати виконання тестів клієнтської частини

Таблиця 3.7

Тестові файли клієнтської частини

Файл тесту	Що саме підлягає перевірці
AssetTable.test.tsx	Перевірка правильності сортування активів за датою надходження та фільтрації рядків таблиці «на льоту».
SpecDynamicForm.test.tsx	Тестування рендерингу кастомних полів форми введення специфікацій на основі структури метаданих JSONB.
RoleProtectedRoute.test.tsx	Перевірка захисту інтерфейсу: анонімний користувач повинен редиректитись на /login.
useAssetActions.test.ts	Тестування кастомного React-хука, який координує асинхронні запити списання активу та обробляє HTTP-помилки сервера.

```

Test Suites: 1 passed, 1 total
Tests:      6 passed, 6 total
Snapshots:  0 total
Time:       0.276 s

```

Рис. 3.20 – Результати виконання інтеграційних тестів

Для комплексного оцінювання надійності розробленого програмного забезпечення було проведено серію автоматизованих перевірок. У процесі тестування застосовувався сучасний стек технологій для валідації якості коду на всіх рівнях архітектури. Зведені дані щодо використаного інструментарію та загальних результатів тестового покриття детально наведено у табл. 3.8.

Загальне покриття тестами

Рівень тестування	Інструментарій	Статус
Валідація бізнес-логіки	Jest, Zod	Пройдено
Серверні API-маршрути	Supertest, Prisma	Пройдено
Інтерфейси користувача	React Testing Library [25]	Пройдено
Транзакції бази даних	PostgreSQL	Пройдено

Комбінований підхід до тестування дозволив оцінити працездатність системи з усіх боків. Оскільки під час фінального прогону тестових наборів у CI-конвеєрі не було виявлено жодної критичної чи регресійної помилки, поточну програмну реалізацію інформаційної системи моніторингу та управління ІТ-активами можна вважати повністю стабільною, надійною та готовою до промислової експлуатації.

Висновки до третього розділу

Програмний продукт розгортається у сучасній хмарній інфраструктурі. Архітектуру платформи побудовано на базі Full-stack фреймворку Next.js [20], який ефективно об'єднує клієнтський вебінтерфейс (React [24]) та серверні API-маршрути, взаємодіючи з гібридною реляційною базою даних PostgreSQL [22]. Щоб автоматизувати та пришвидшити випуск оновлень, налаштовано CI/CD [11] пайплайн за допомогою GitHub Actions [11]. Кожен коміт у головну гілку репозиторію автоматично ініціює статичний аналіз коду (Linting), перевірку суворості типізації TypeScript [10, 29], запуск тестів та оптимізовану збірку проєкту. Після успішного проходження конвеєра нова версія застосунку безшовно розгортається на хмарній платформі Vercel [1], а інфраструктура бази даних підтримується провайдером Neon [2]. Для комфортної розробки на локальній машині передбачено конфігурацію Docker [7] Compose, яка дозволяє швидко та ізольовано запустити екземпляр бази даних для тестування.

На стороні клієнтського застосунку реалізовано чітке розмежування доступу згідно з моделлю RBAC. Інтерфейс та його функціональні можливості динамічно адаптуються залежно від ролі звичайний Співробітник взаємодіє з

особистим кабінетом виключно для моніторингу закріпленої за ним техніки та ліцензії ПЗ, тоді як Адміністратор та Технічний спеціаліст використовують повноцінну аналітичну вебпанель для глобального управління реєстром активів, моніторингу процесів технічного обслуговування (Maintenance) та списання обладнання. Завдяки цьому екосистему вдалося логічно розділити на зону персонального споживання інформації та зону комплексного адміністрування корпоративної IT-інфраструктури.

Якість коду підтверджено багаторівневим автоматизованим тестуванням. Сервісний шар вкрито модульними тестами (Jest [14]) для суворої перевірки бізнес-правил та ізольованої валідації інваріантів. На клієнтській частині тести контролюють захищені навігаційні маршрути та коректність відмальовування динамічних форм на основі об'єктів JSONB. Найважливіший етап – інтеграційне тестування API за допомогою інструменту Supertest та Prisma ORM [23], що дозволяє перевірити реальну транзакційну роботу контролерів із тестовими екземплярами бази даних PostgreSQL. Увесь тестовий набір виконується успішно, що доводить стабільність системи, готовність до промислової експлуатації та надійність обраних архітектурних рішень.

ВИСНОВКИ

Результатом виконання дипломної роботи є створена інформаційна система моніторингу та управління ІТ-активами підприємства. Система призначена для комплексного обліку комп'ютерної техніки, контролю за ліцензіями програмного забезпечення та управління життєвим циклом обладнання. Вона переводить у сучасний цифровий формат усі етапи роботи з корпоративним ІТ-фондом від реєстрації нового пристрою та закріплення його за співробітником до технічного обслуговування, моніторингу обладнання.

Огляд існуючого програмного забезпечення виявив низку проблем у цій ніші. Наявні на ринку корпоративні системи або є надто громіздкими та дорогими в ліцензуванні, або мають застарілий інтерфейс і базуються на жорстких структурах баз даних, що унеможлиблює зручний облік різноманітної техніки з унікальними специфікаціями. Крім того, більшість розглянутих сервісів не має гнучкої ролізованої моделі з інтуїтивним особистим кабінетом для звичайного працівника та потужною аналітичною вебпанеллю для технічного відділу. Це стало головним аргументом для розробки власного сучасного та модульного рішення.

Програмна архітектура проєкту базується на сучасних інженерних патернах із використанням Full-stack фреймворку Next.js [20]. В основу бізнес-логіки покладено гібридну модель бази даних PostgreSQL [22], яка через об'єктно-реляційне відображення Prisma ORM [23] поєднує суворі реляційні зв'язки для обліку з гнучким форматом JSONB для збереження динамічних технічних характеристик активів. Інфраструктура бази даних надійно контролює транзакційну цілісність, атомарний розподіл ліцензійних місць та автоматично фіксує всі зміни у незмінному журналі аудиту (Audit Log). Завдяки сучасному підходу до розгортання, релізи нових версій системи виконуються повністю автоматично.

Головним практичним результатом є розроблений комплексний інструмент для консолідованого управління корпоративною ІТ-інфраструктурою. Звичайний співробітник отримує зручний кабінет для

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

перегляду доступної йому інформації, тоді як адміністратор та технічний спеціаліст можуть у реальному часі керувати реєстром. Стабільність платформи доведено на практиці: для перевірки коду реалізовано комплекс багаторівневих тестів, які успішно покривають бізнес-правила доменних моделей, інтеграційний рівень API-контролерів бази даних та клієнтські компоненти інтерфейсу користувача.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний портал розробників інфраструктури хмарних рішень Vercel. URL: <https://vercel.com/docs> (дата звернення: 12.05.2026).
2. Neon Serverless Postgres Official Documentation. Neon. URL: <https://neon.tech/docs/> (дата звернення: 15.05.2026).
3. Axelos. ITIL Foundation, ITIL 4 Edition. London: TSO (The Stationery Office), 2019. 212 p.
4. Beck K. Test-Driven Development: By Example. Boston: Addison-Wesley Professional, 2022. 240 p.
5. Brooks P. Metrics for IT Service Management. Zaltbommel: Van Haren Publishing, 2006. 208 p.
6. Feature-Sliced Design. Architectural methodology for frontend projects. URL: <https://feature-sliced.design/docs> (дата звернення: 20.05.2026).
7. Docker Official Documentation. Docker Inc. URL: <https://docs.docker.com/> (дата звернення: 18.05.2026).
8. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol: O'Reilly Media, 2020. 706 p.
9. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley Professional, 2012. 560 p.
10. Freeman A. Essential TypeScript 5: From Beginner to Pro. New York: Apress, 2023. 600 p.
11. GitHub Actions Documentation. GitHub Docs. URL: <https://docs.github.com/en/actions> (дата звернення: 10.05.2026).
12. Hobbs M. IT Asset Management: A Pocket Survival Guide. Ely: IT Governance Publishing, 2020. 154 p.
13. ISO/IEC 19770-1:2017. Information technology – IT asset management – Part 1: IT asset management systems – Requirements. Geneva: ISO, 2017. 34 p.
14. Jest: Delightful JavaScript Testing. Meta Open Source. URL: <https://jestjs.io/docs/getting-started> (дата звернення: 05.05.2026).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland: IT Revolution Press, 2021. 480 p.
16. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017. 616 p.
17. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 8th ed. Pearson, 2020. 864 p.
18. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
19. NextAuth.js Official Documentation. NextAuth.js. URL: <https://next-auth.js.org/getting-started/introduction> (дата звернення: 21.05.2026).
20. Next.js Documentation. Vercel. URL: <https://nextjs.org/docs> (дата звернення: 02.05.2026).
21. Osmani A. Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide. 2nd ed. Sebastopol: O'Reilly Media, 2023. 280 p.
22. PostgreSQL Global Development Group. PostgreSQL 15 Official Documentation. URL: <https://www.postgresql.org/docs/15/index.html> (дата звернення: 11.05.2026).
23. Prisma ORM Documentation. Prisma Data. URL: <https://www.prisma.io/docs/> (дата звернення: 14.05.2026).
24. React Official Documentation. Meta. URL: <https://react.dev/> (дата звернення: 03.05.2026).
25. React Testing Library. Testing Library. URL: <https://testing-library.com/docs/react-testing-library/intro/> (дата звернення: 08.05.2026).
26. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island: Manning Publications, 2018. 520 p.
27. Shadcn UI Components. Shadcn. URL: <https://ui.shadcn.com/docs> (дата звернення: 22.05.2026).

28. Tailwind CSS Documentation. Tailwind Labs. URL: <https://tailwindcss.com/docs> (дата звернення: 04.05.2026).
29. TypeScript Handbook. Microsoft. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 01.05.2026).
30. W3C. JSON (JavaScript Object Notation) Data Interchange Format. URL: <https://www.w3.org/TR/json/> (дата звернення: 25.05.2026).
31. Zod: TypeScript-first schema validation. Zod. URL: <https://zod.dev/> (дата звернення: 26.05.2026).
32. Imhoff C., White C. Self-Service Business Intelligence: Empowering Users to Generate Insights. TDWI Best Practices Report. Renton: TDWI Research, 2011. 36 p.
33. Pezoa F., Reutter J. L., Suarez F., Ugarte M., Vrgoč D. Foundations of JSON Schema. Proceedings of the 25th International Conference on World Wide Web. Montréal: ACM, 2016. P. 263–273.

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця А.1 Специфікація прецеденту «Подача сервісного тікету»

Атрибут	Опис
Актор	Співробітник
Мета	Створити заявку на ремонт або обслуговування техніки, яка вийшла з ладу.
Передумови	Користувач авторизований у системі та має закріплену техніку.
Основний сценарій	1. Користувач переходить у розділ «Служба підтримки». 2. Натискає кнопку «Створити заявку». 3. Обирає свій актив із випадаючого списку та описує проблему. 4. Встановлює пріоритет (Низький, Середній, Критичний). 5. Система зберігає тікет у статусі «Відкрито» і сповіщає техвідділ.
Постумови	Тікет з'являється на Kanban-дошці у технічних спеціалістів.

Таблиця А.2 Специфікація прецеденту «Управління активами (CRUD)»

Атрибут	Опис
Актор	Технічний спеціаліст, Адміністратор
Мета	Додавання нових одиниць техніки на баланс, їх редагування або видалення.
Передумови	Користувач має відповідні права RBAC для редагування реєстрів бази даних.
Основний сценарій	1. Користувач відкриває загальну Data Table активів. 2. Натискає кнопку «Додати актив» або іконку редагування. 3. Заповнює форму (назва, категорія, серійний номер, JSONB-специфікації). 4. Система валідує дані через Zod-схему на сервері. 5. Дані зберігаються в PostgreSQL, а інформація в таблиці оновлюється без перезавантаження сторінки.
Альтернативний сценарій	Помилка унікальності: якщо серійний номер вже існує, сервер повертає статус 409 Conflict.

Таблиця А.3 Специфікація прецеденту «Обробка сервісних тікетів»

Атрибут	Опис
Актор	Технічний спеціаліст, Адміністратор
Мета	Забезпечення процесу розв'язання проблем користувачів через Kanban-дошку.
Передумови	У системі є хоча б один відкритий тікет.
Основний сценарій	1. Технічний спеціаліст відкриває сторінку «Служба підтримки». 2. Бере тікет із колонки «Відкриті» та перетягує (drag-and-drop) у колонку «В роботі». 3. Проводить технічні роботи з обладнанням. 4. Переносить тікет у колонку «Вирішені». 5. Система автоматично записує час вирішення проблеми в базу даних.
Постумови	Користувач, який створив тікет, бачить, що його заявку успішно закрито.

Фрагменти коду

Б.1 Схема бази даних

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Asset {
  id          String    @id @default(uuid())
  name        String
  category    String
  status      String
  serialNumber String    @unique
  inventoryId String    @unique
  price       Float
  location    String
  user        String
  type        String?
  brand       String?
  model       String?
  specs       Json?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  @@map("assets")
}

model SystemSettings {
  id          Int      @id @default(1)
  adminName   String   @default("System Admin")
  adminEmail  String   @default("admin@itasset.com")
  currency    String   @default("UAH")
  assetPrefix String   @default("ITA-")
  emailNotif  Boolean  @default(true)
  slackNotif  Boolean  @default(false)
  slackWebhook String  @default("")
  maintenanceMode Boolean @default(false)
}

model Server {

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    id          String    @id @default(uuid())
    name        String
    ip          String
    type        String
    status      String
    cpu         String
    ram         String
    uptime      String
    createdAt   DateTime @default(now())
    @@map("servers")
}

model Log {
    id          String    @id @default(cuid())
    time        String
    actor       String
    type        String
    source      String
    text        String
    createdAt   DateTime @default(now())
    @@map("logs")
}

model Ticket {
    id          String    @id @default(uuid())
    title       String
    description  String?
    status      String    @default("Відкрито")
    priority    String    @default("Середній")
    user        String
    createdAt   DateTime @default(now())
    updatedAt   DateTime @updatedAt
    @@map("tickets")

model SecurityLog {
    id          String    @id @default(uuid())
    event       String
    ip          String
    device      String
    time        String
    status      String
    createdAt   DateTime @default(now())
    @@map("security_logs")
}

model SecuritySettings {

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    id          Int          @id @default(1)
    twoFactor    Boolean @default(false)
    ldapEnabled Boolean @default(false)
    apiKey       String      @default("")
    @@map("security_settings")
}

model User {
    id          String      @id @default(cuid())
    name        String
    email       String      @unique
    password    String
    role        String      @default("tech")
    createdAt   DateTime @default(now())
    updatedAt   DateTime @updatedAt
}

model License {
    id          String      @id @default(cuid())
    name        String
    softwareType String
    licenseKey   String?
    totalSeats   Int
    usedSeats    Int          @default(0)
    expirationDate DateTime?
    createdAt    DateTime @default(now())
    updatedAt    DateTime @updatedAt
    @@map("licenses")
}

model Employee {
    id          String      @id @default(uuid())
    firstName   String
    lastName    String
    email       String      @unique
    position    String
    department  String
    status      String      @default("Активний")
    createdAt   DateTime @default(now())
    updatedAt   DateTime @updatedAt
    @@map("employees")
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

Б.2 API / Server Actions

```
import { neon } from '@neondatabase/serverless';
import { NextResponse } from 'next/server';
import { sendSlackNotification } from '@lib/slack';
const sql = neon(process.env.DATABASE_URL!);
export async function PUT(req: Request, { params }: { params: any }) {
  try {
    const resolvedParams = await params;
    const id = resolvedParams.id;
    const body = await req.json();
    const { name, status, category, brand, model, serial_number, specs,
assignedTo } = body;
    let userToStore = assignedTo || 'Не призначено';
    if (['retired', 'missing'].includes(status)) {
      userToStore = 'Не призначено';
    }
    const updatedAsset = await sql`
      UPDATE assets
      SET
        name = ${name},
        type = ${category || 'Інше'},
        status = ${status},
        category = ${category},
        brand = ${brand || null},
        model = ${model || null},
        "serialNumber" = ${serial_number || null},
        specs = ${specs || null},
        "user" = ${userToStore},
        "updatedAt" = NOW()
      WHERE id = ${id}
      RETURNING *;
    `;
    if (updatedAsset.length === 0) {
      return NextResponse.json({ error: 'Актив не знайдено' }, { status: 404 });
    }
    const asset = updatedAsset[0];
    try {
      const settingsResult = await sql`SELECT "slackNotif", "slackWebhook" FROM
settings LIMIT 1`;
      const settings = settingsResult[0];
      if (settings && settings.slackNotif && settings.slackWebhook) {
        const payload = {
          blocks: [
            {
              type: 'section',
              text: {
                type: 'mrkdwn',
                text: `🔔 *Оновлено актив:* ${asset.name}\n*Новий статус:*
\`${status}\`\n*Користувач:* ${userToStore}`
              }
            }
          ]
        };
        await sendSlackNotification(settings.slackWebhook, payload);
      }
    } catch (slackError) {
      console.error('Помилка відправки Slack (PUT):', slackError);
    }

    const assetWithFrontendFields = {
      ...asset,
      assignedTo: asset.user === 'Не призначено' ? '' : (asset.user || ''),
    };
  }
}
```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

        serial_number: asset.serialNumber
    };

    return NextResponse.json(assetWithFrontendFields);
} catch (error: any) {
    return NextResponse.json({ error: error.message || 'Failed to update asset'
}, { status: 500 });
}
}

export async function DELETE(req: Request, { params }: { params: any }) {
    try {
        const resolvedParams = await params;
        const id = resolvedParams.id;
        const deletedAsset = await sql`
            DELETE FROM assets
            WHERE id = ${id}
            RETURNING *;
        `;

        if (deletedAsset.length === 0) {
            return NextResponse.json({ error: 'Актив не знайдено' }, { status: 404 });
        }
        const asset = deletedAsset[0];
        try {
            const settingsResult = await sql`SELECT "slackNotif", "slackWebhook" FROM
settings LIMIT 1`;
            const settings = settingsResult[0];

            if (settings && settings.slackNotif && settings.slackWebhook) {
                const payload = {
                    blocks: [
                        {
                            type: 'section',
                            text: {
                                type: 'mrkdwn',
                                text: `🔔 *Увага! Актив видалено з бази!*\\n*Назва:*
${asset.name}\\n*Категорія:* ${asset.category || 'Не вказано'}\\n*ID:* \\`${id}\\``
                            }
                        }
                    ]
                };
                await sendSlackNotification(settings.slackWebhook, payload);
            }
        } catch (slackError) {
            console.error('Помилка відправки Slack (DELETE):', slackError);
        }
        return NextResponse.json({ message: 'Asset deleted successfully' });
    } catch (error: any) {
        return NextResponse.json({ error: error.message || 'Failed to delete asset'
}, { status: 500 });
    }
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		

Б.3 Клієнтська частина / Інтерфейс користувача

```
'use client';

import React, { useState } from 'react';
import { Asset, AssetStatus, AssetType } from '@types/asset';
import { Table, TableBody, TableCell, TableHead, TableHeader, TableRow } from
"@/components/ui/table";
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";

interface AssetTableProps {
  assets: Asset[];
  onDelete: (id: string) => void;
  onEdit: (asset: Asset) => void;
}

const getStatusLabel = (status: AssetStatus) => {
  switch (status) {
    case 'active': return 'Активний';
    case 'maintenance': return 'Обслуговування';
    case 'retired': return 'Списаний';
    case 'missing': return 'Відсутній';
    default: return status;
  }
};

export default function AssetTable({ assets = [], onDelete, onEdit }:
AssetTableProps) {
  const [searchTerm, setSearchTerm] = useState('');
  const [filterType, setFilterType] = useState<AssetType | 'all'>('all');
  const filteredAssets = assets.filter(asset => {
    const matchesSearch = (asset.name ||
'').toLowerCase().includes(searchTerm.toLowerCase()) ||
(asset.owner ||
'').toLowerCase().includes(searchTerm.toLowerCase());
    const matchesFilter = filterType === 'all' || asset.type === filterType;
    return matchesSearch && matchesFilter;
  });

  const handleExportCSV = () => {
    if (filteredAssets.length === 0) return;

    const headers = ['Назва активу', 'Серійний номер', 'Тип', 'Статус',
'Власник', 'Локація'];
    const rows = filteredAssets.map(asset => [
      asset.name,
      asset.serialNumber || asset.id,
      asset.type,
      getStatusLabel(asset.status),
      asset.owner,
      asset.location
    ]);

    const csvContent = '\uFEFF' + [headers.join(','), ...rows.map(e => e.map(val
=> `\"${val}\"`).join(','))].join('\n');
    const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.setAttribute('href', url);
    link.setAttribute('download', `it_assets_export_${new
Date().toISOString().split('T')[0]}.csv`);
    document.body.appendChild(link);
    link.click();
  };
}
```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    document.body.removeChild(link);
  };

  return (
    <div className="glass-panel p-6 overflow-hidden border-none">
      <div className="flex justify-between items-center mb-6 gap-4">
        { /* Пошук */ }
        <Input
          placeholder="Пошук за назвою або власником..."
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
        />
        { /* Кнопка експорту */ }
        <Button onClick={handleExportCSV} disabled={filteredAssets.length ===
0}>>
          Експорт CSV
        </Button>
      </div>

      { /* Таблиця */ }
      <Table>
        <TableHeader>
          <TableRow>
            <TableHead>Назва активу</TableHead>
            <TableHead>Статус</TableHead>
            <TableHead>Власник</TableHead>
            <TableHead>Дії</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {filteredAssets.map((asset) => (
            <TableRow key={asset.id}>
              <TableCell>{asset.name}</TableCell>
              <TableCell>{getStatusLabel(asset.status)}</TableCell>
              <TableCell>{asset.owner}</TableCell>
              <TableCell>
                <Button onClick={() => onEdit(asset)}>Редагувати</Button>
                <Button onClick={() => onDelete(asset.id)}
variant="destructive">Видалити</Button>
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </div>
  );
}

```

Б.4 Компонент навігації з рольовим розмежуванням доступу

```
'use client';

import React, { useState, useEffect } from 'react';
import Link from 'next/link';
import { usePathname } from 'next/navigation';
import { useSession, signOut } from 'next-auth/react';
import { LayoutDashboard, Monitor, Database, Key, Ticket, Users, Settings, Bell,
LogOut } from 'lucide-react';
import { motion, AnimatePresence } from 'framer-motion';

const menuItems = [
  { icon: LayoutDashboard, label: 'Дашборд', href: '/', allowedRoles: ['admin',
'tech', 'user'] },
  { icon: Database, label: 'ІТ-Активи', href: '/assets', allowedRoles: ['admin',
'tech', 'user'] },
  { icon: Key, label: 'Ліцензії та ПЗ', href: '/licenses', allowedRoles:
['admin', 'tech'] },
  { icon: Ticket, label: 'Служба підтримки', href: '/helpdesk', allowedRoles:
['admin', 'tech', 'user'] },
  { icon: Users, label: 'Співробітники', href: '/employees', allowedRoles:
['admin', 'tech'] },
  { icon: Settings, label: 'Налаштування', href: '/settings', allowedRoles:
['admin'] },
];

export default function Sidebar() {
  const pathname = usePathname();
  const { data: session, status } = useSession();
  const [showNotifications, setShowNotifications] = useState(false);
  const [notifications, setNotifications] = useState<any[]>([]);

  useEffect(() => {
    if (status === 'authenticated') {
      fetchNotifications();
      const interval = setInterval(fetchNotifications, 30000);
      return () => clearInterval(interval);
    }
  }, [status]);

  const fetchNotifications = async () => {
    try {
      const res = await fetch('/api/notifications');
      if (res.ok) {
        const data = await res.json();
        setNotifications(data);
      }
    } catch (error) {
      console.error('Не вдалося завантажити сповіщення:', error);
    }
  };

  if (pathname === '/login') return null;

  const userRole = (session?.user as any)?.role || 'guest';
  const filteredMenuItems = menuItems.filter(item =>
item.allowedRoles.includes(userRole));
  const unreadCount = notifications.filter(n => !n.read).length;

  return (
    <aside className="w-[260px] h-[calc(100vh-32px)] flex flex-col sticky top-4
z-50 glass-panel">
      <div className="p-8 flex items-center gap-3">
```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        <Monitor className="text-primary" />
        <span className="text-xl font-extrabold text-transparent">IT-Asset
OS</span>
    </div>

    <nav className="flex-1 px-3 flex flex-col gap-2">
        {filteredMenuItems.map((item) => {
            const isActive = pathname === item.href;
            return (
                <Link key={item.href} href={item.href}>
                    <motion.div className={`flex items-center p-3 rounded-xl gap-3
${isActive ? "text-primary bg-primary/10" : "hover:bg-white/5"}`>
                        <item.icon size={20} />
                        <span className="text-[15px] font-medium flex-
1">{item.label}</span>
                    </motion.div>
                </Link>
            );
        })}
    </nav>

    {/ * Панель профілю та керування сесією */}
    <div className="p-4 border-t border-border flex items-center gap-3">
        <div className="overflow-hidden flex-1">
            <p className="text-sm font-semibold truncate text-white">
                {session?.user?.name || "Завантаження..."}
            </p>
            <p className="text-xs text-muted-foreground truncate">
                {userRole.toUpperCase()}
            </p>
        </div>

        <button onClick={() => setShowNotifications(!showNotifications)}
className="p-2 relative">
            <Bell size={18} />
            {unreadCount > 0 && <span className="absolute top-1 right-1 w-2 h-2
bg-red-500 rounded-full" />}
        </button>

        <button onClick={() => signOut({ callbackUrl: '/login' })} className="p-
2 hover:text-red-400">
            <LogOut size={18} />
        </button>
    </div>
</aside>
);
}

```

Б.5 API-контролер оновлення специфікацій (JSONB)

```
import { NextResponse } from "next/server";
import { db } from "@/lib/db";
import { z } from "zod";

// Схема валідації для динамічного JSONB-об'єкта специфікацій
const specsSchema = z.record(z.union([z.string(), z.number(), z.boolean()]));

export async function PATCH(req: Request, { params }: { params: { id: string } }) {
  try {
    const body = await req.json();

    // 1. Валідація структури JSON-даних (реалізація алгоритму контролю)
    const parsedSpecs = specsSchema.parse(body.specs);

    // 2. Оновлення запису в PostgreSQL з використанням типу JSONB
    const updatedAsset = await db.asset.update({
      where: { id: params.id },
      data: {
        specs: parsedSpecs,
      },
    });

    return NextResponse.json(updatedAsset, { status: 200 });
  } catch (error) {
    if (error instanceof z.ZodError) {
      return NextResponse.json({ message: "Невалідний формат JSONB специфікацій" }, { status: 400 });
    }
    return NextResponse.json({ message: "Внутрішня помилка сервера" }, { status: 500 });
  }
}
```