Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Інформаційна система автоматизованого обліку та управління заявками для сервісних центрів обслуговування техніки
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *126 «Інформаційні системи та технології»*
Шифр, назва

Освітня програма *«Інформаційні системи та технології»*
Назва

Шифр КвРІСТ 220173.22.01.08 ПЗ

Виконав здобувач IV курсу, група ІСТ-22-1 _____ Максим КОБИЛЬЧУК
Підпис Ініціали, прізвище

Керівник асистент _____ Олег ВОЙЧУР
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер к.ф - м.н., доцент _____ Тетяна КИСІЛЬ
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КПС _____ Ольга ПАВЛОВА
«04» червня 2026 р. підпис Ініціали, прізвище
дата

Хмельницький 2026

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 Інформаційні технології

Спеціальність 126 Інформаційні системи та технології

Освітня програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кобильчуку Максиму новичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Інформаційна система автоматизованого обліку та управління заявками для сервісних центрів обслуговування техніки

Керівник проекту (роботи) Войчур Олег Юрійович, асистент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

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

Дослідження предметної області та постановка задачі

Проєктування архітектури та компонентів програмно-технічного засобу

Програмно-апаратна реалізація та тестування програмно-технічного засобу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схема гнучкої моделі даних NoSQL у Firebase Firestore

Діаграма криптографічного управління сесіями та взаємодії компонентів автентифікації на базі протоколу JWT

UML-діаграма послідовності (Sequence Diagram)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_10_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 – проєктування інформаційноїсистеми збору та аналізукористувацької активності веб-сервісу	01.04.2026	виконано
5	Робота над розділом 3 – програмно-апаратна реалізація та тестування програмно-технічного засобу	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач_____ ___Максим КОБИЛЬЧУК__
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ ___Олег ВОЙЧУР___
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРІСТ 220173.22.01.03 ПЗ	Пояснювальна записка	58		
			Графічні матеріали			
2		КвРІСТ 220173.22.01.03 Е8	Схема гнучкої моделі даних NoSQL у Firebase Firestore	1		
3		КвРІСТ 220173.22.01.03 Е8	Діаграма криптографічного управління сесіями та взаємодії компонентів автентифікації на базі протоколу JWT	1		
4		КвРІСТ 220173.22.01.03 Е8	UML-діаграма послідовності (Sequence Diagram)	1		

КвРІСТ 220173.22.01.08 ВП

Зм	Арк	№ докум	Підпис	Дата
Розробив		Кобильчук		
Перевір.		Войчур		
Н. контр.		Кисіль		
Затв.		Павлова		

Відомість проекту

Літера	Аркуш	Аркушів
У	1	58

ХНУ, ІСТ-22-1

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Інформаційна система автоматизованого обліку та управління заявками для сервісних центрів обслуговування техніки».

Автор роботи: Максим КОБИЛЬЧУК.

Керівник роботи: Олег ВОЙЧУР.

Пояснювальна записка: 58 с., 23 рис., 5 табл., 4 дод., 40 джерел.

Графічна частина: 3 креслення.

ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ БІЗНЕС-ПРОЦЕСІВ, СЕРВІСНИЙ ЦЕНТР.

Кваліфікаційна робота бакалавра присвячена проєктуванню та програмній реалізації комплексної інформаційної системи автоматизації операційної діяльності сервісного центру електроніки MonkeyLAB.

У роботі проведено системний аналіз процесів обслуговування клієнтів та логістики товарно-матеріальних цінностей. Обґрунтовано вибір сучасної клієнт-серверної архітектури за моделлю односторінкового додатка (Single Page Application) у поєднанні з безсерверною інфраструктурою (Serverless Computing). Програмну реалізацію клієнтської частини виконано з використанням фреймворку React 18, мови суворої типізації TypeScript та середовища збирання Vite.

Спроєктовано гнучку структуру документо-орієнтованої бази даних на платформі Firebase Firestore, що дозволяє ефективно обробляти гетерогенні технічні параметри ремонтної техніки. Реалізовано чотири незалежні, але транзакційно інтегровані підсистеми: управління заявками, ведення цифрових карток клієнтів, автоматизований складський облік та фінансова бізнес-аналітика.

Підпис здобувача

30.05.2026

Дата

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційного суспільства характеризується експоненційним зростанням кількості складної електронної техніки в повсякденному обігу. Відповідно, сфера надання послуг з технічного обслуговування та відновлення мікроелектроніки перетворилася на високотехнологічну галузь, що генерує колосальні масиви даних. Операційна діяльність типового сервісного центру передбачає безперервну обробку гетерогенної інформації: від фіксації специфічних апаратних дефектів та контролю розгалуженої номенклатури складських запасів до ведення історії клієнтських звернень і формування складної фінансової звітності.

Використання застарілих методів паперового документообігу або децентралізованих електронних таблиць на сьогодні є ключовим фактором, що гальмує розвиток підприємства, призводить до втрати даних, виникнення пересортиці на складах та зниження рівня клієнтської лояльності. З іншого боку, наявні на ринку універсальні комерційні CRM/ERP системи часто виявляються економічно недоцільними для вузькоспеціалізованих майстерень через надмірну перевантаженість інтерфейсів та високу вартість підписки за моделлю SaaS. З огляду на це, розроблення власного, спеціалізованого та еластично масштабованого програмно-технічного засобу, здатного забезпечити надійну автоматизацію бізнес-процесів із суворим розмежуванням прав доступу персоналу, є актуальним науково-прикладним завданням.

Мета і завдання дослідження. Метою кваліфікаційної роботи є підвищення ефективності управління операційною діяльністю сервісного центру шляхом проєктування та програмної реалізації комплексної інформаційної системи MonkeyLAB на базі сучасної клієнт-серверної архітектури.

Для досягнення поставленої мети було сформульовано та вирішено такі завдання:

1. Здійснити системний аналіз предметної області, дослідити життєвий цикл ремонтної заявки та виявити недоліки існуючих комерційних рішень.

2. Спроєктувати концептуальну архітектуру програмно-технічного засобу, обґрунтувати вибір технологічного стека та парадигми односторінкового додатка (SPA).

3. Розробити логічну та фізичну структуру документо-орієнтованої бази даних для ефективного збереження гетерогенних характеристик електронної техніки.

4. Спроєктувати багаторівневу модель інформаційної безпеки та розмежування прав доступу на основі ролей (RBAC) з імплементацією правил на рівні ядра бази даних.

5. Виконати програмну реалізацію клієнтської частини, інтегрувати модулі управління заявками, логістикою, базою клієнтів та фінансовою аналітикою.

6. Провести комплексне тестування розробленого програмного продукту та здійснити оцінку його функціональної придатності в умовах контрольної експлуатації.

Об'єкт дослідження – процес управління інформаційними потоками, логістикою та операційною діяльністю підприємства у сфері надання послуг з ремонту електронної техніки.

Предмет дослідження – архітектурні шаблони, алгоритмічні рішення, структури даних та програмні інструменти, необхідні для побудови інтегрованої системи автоматизації сервісного центру.

Методи дослідження. Методологічною основою роботи є принципи системного підходу та об'єктно-орієнтованого проєктування. Для моделювання бізнес-процесів та архітектури бази даних застосовувалися методи структурного аналізу. Під час розроблення користувацького інтерфейсу використовувалися методи компонентного проєктування та атомарного дизайну. Оцінка надійності та працездатності системи здійснювалася за допомогою методів функціонального тестування (Black-box testing) та аналізу граничних значень.

Практичне значення одержаних результатів полягає у тому, що створений програмно-технічний засіб MonkeyLAB є повністю завершеним і готовим до впровадження комерційним продуктом. Запропоновані архітектурні та алгоритмічні рішення дозволяють усунути людський фактор при складському обліку, автоматизувати калькуляцію вартості послуг, надійно захистити комерційну таємницю від несанкціонованого доступу та надати керівництву потужний інструмент для візуалізації економічних показників підприємства у режимі реального часу.

.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області та бізнес-процесів сервісного центру

Сучасна сфера надання послуг з відновлення складної електронної архітектури є високотехнологічною галуззю, що характеризується значною інтенсивністю інформаційних потоків та багаторівневою структурою операційних процесів. Базовим об’єктом дослідження у цій предметній області виступає життєвий цикл обслуговування клієнтського обладнання, який охоплює етапи від первинного звернення замовника до фінальної видачі відновленого пристрою. Кожен з цих етапів генерує специфічні масиви даних, які потребують безперервної фіксації, систематизації та подальшого аналізу для забезпечення стабільного функціонування підприємства. Дослідження типового бізнес-процесу виявляє його високу чутливість до людського фактора: помилки під час приймання техніки, втрата контактних даних клієнта або некоректна ідентифікація апаратної несправності призводять до репутаційних та фінансових втрат. Відповідно, виникає об’єктивна проблема децентралізації інформації, коли комунікація з клієнтом, облік техніки та фінансові розрахунки ведуться у розрізнених, не інтегрованих між собою системах або на паперових носіях.

Логістичний ланцюг обробки замовлення вимагає чіткого документування стану пристрою на момент його надходження до сервісного центру, що включає опис візуальних дефектів, заявлених несправностей та комплектації. На етапі діагностики та безпосереднього ремонту відбувається генерація технічної інформації, яка повинна бути миттєво доступна як для відповідального інженера, так і для менеджерського складу. Завершальний етап супроводжується формуванням фінансових документів та гарантійних зобов'язань. Управління цими потоками даних в ручному режимі ускладнює масштабування бізнесу та

робить неможливим побудову ефективної моделі прогнозування завантаженості персоналу.

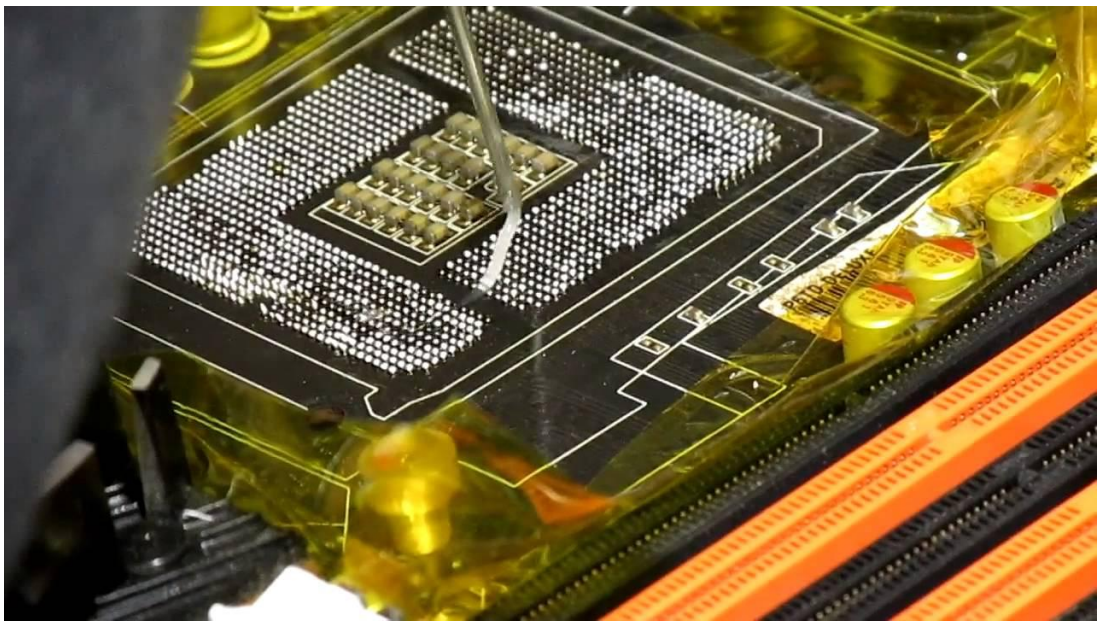


Рисунок 1.1 – Процес діагностики та відновлення елементної бази сучасної мікроелектроніки

1.2 Дослідження проблематики обліку складної електронної архітектури та товарно-матеріальних цінностей

Специфіка ремонту високотехнологічної електроніки полягає у необхідності оперування надзвичайно широкою номенклатурою товарно-матеріальних цінностей. Складський облік сервісного центру докорінно відрізняється від класичного ритейлу через наявність мікросхем, специфічних витратних матеріалів, донорських плат та унікальних компонентів, які можуть не мати стандартизованих артикулів. Проблема полягає у складності ідентифікації поточних залишків та контролі їх руху між загальним складом і робочими місцями майстрів. Відсутність автоматизованої системи списання матеріалів безпосередньо у прив'язці до конкретної ремонтної заявки створює умови для виникнення пересортиці та неконтрольованого використання ресурсів підприємства

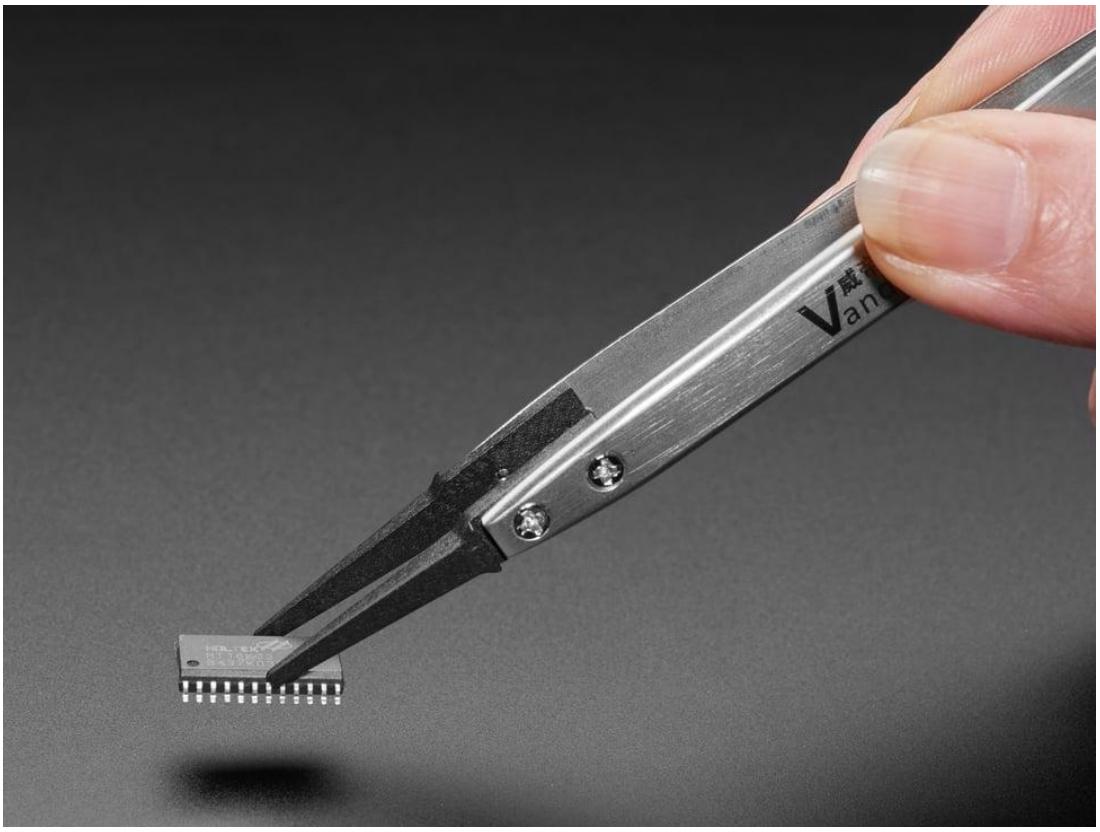


Рисунок 1.2 – Приклад складної компонентної бази донорських плат та мікросхем, що потребує автоматизованого обліку

Додатковим аспектом проблематики є управління ціноутворенням. Електронні компоненти часто закуповуються у валюті, що призводить до постійного коливання їхньої собівартості. Ручний перерахунок відпускних цін на комплектуючі під час формування загальної вартості ремонту є ресурсомістким завданням, яке супроводжується високим ризиком математичних помилок. Заморожування обігових коштів у неліквідних запасах через відсутність аналітичних інструментів для відстеження обіговості конкретних позицій є ще одним головним чинником, що знижує загальну рентабельність електронної майстерні. Відповідно, система повинна забезпечувати транзакційну цілісність під час кожної операції списання або оприбуткування.

Таблиця 1.1 – Класифікація основних проблем предметної області

Категорія проблематики	Опис виявленого недоліку в існуючих ручних процесах	Наслідки для бізнесу
Операційна	Дефрагментація інформації про статуси ремонтних заявок	Зрив термінів виконання, зниження лояльності замовників
Логістична (Склад)	Неможливість точного контролю залишків мікрокомпонентів	Пересортиця, заморожування коштів, брак деталей
Фінансова	Складність калькуляції чистого прибутку за кожною заявкою	Викривлення фінансової звітності, збиткові ремонти

1.3 Аналіз наявного програмно-технічного забезпечення предметної області

У процесі дослідження було здійснено огляд комерційних програмних продуктів, які вже застосовуються в аналогічних сервісних центрах. Метою аналізу є виявлення сильних сторін та головних обмежень існуючих платформ. Розглянуто хмарні системи RemOnline, Gincore та HelloClient. Система RemOnline пропонує розгалужений функціонал для ведення взаєморозрахунків та складського обліку, проте її архітектура є надмірно складною для невеликих спеціалізованих майстерень. Інтерфейс перевантажений елементами управління, які не використовуються в щоденній практиці, що збільшує час обробки однієї заявки. Крім того, комерційна модель розповсюдження (SaaS) передбачає постійні абонентські платежі, що створює постійне фінансове навантаження.

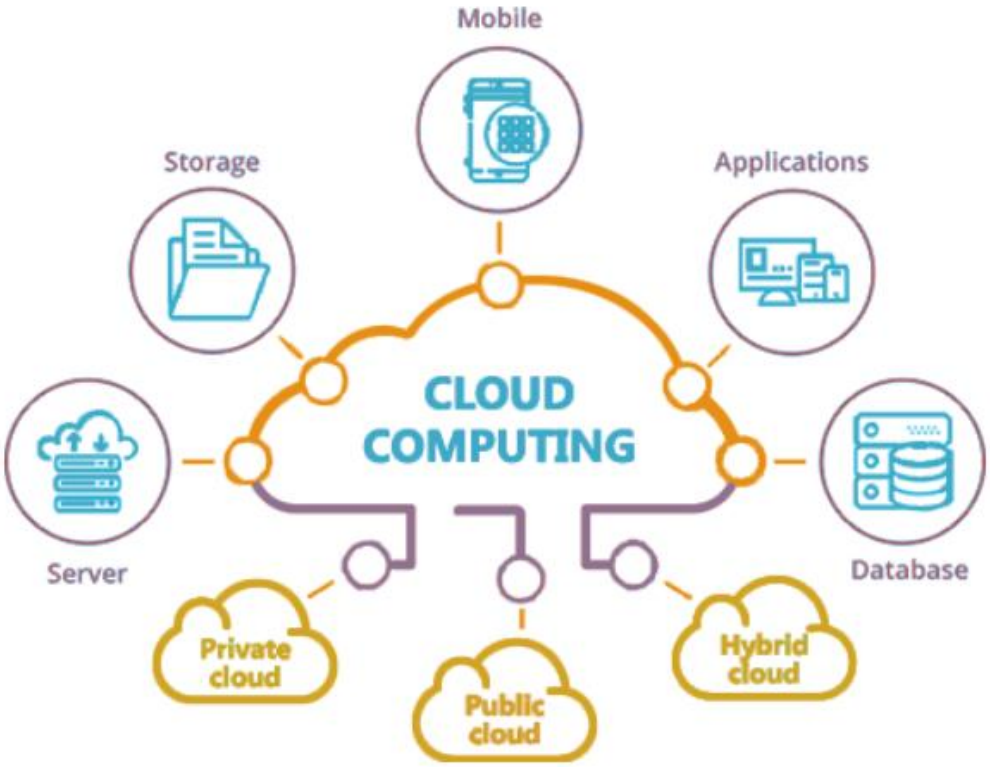


Рисунок 1.3 – Концептуальна модель сучасних хмарних систем управління (SaaS)

Програмний комплекс Gincore фокусується на глибокій логістиці, дозволяючи вести серійний облік та адресне зберігання. Незважаючи на ці переваги, система має жорстку архітектуру бізнес-логіки, яка не дозволяє гнучко адаптувати статуси заявок під унікальні процеси конкретного сервісного центру. Система HelloClient є протилежністю попереднім рішенням, пропонуючи мінімалістичний дизайн та швидкий старт. Проте цей продукт має суттєві прогалини у системі аналітики та безпеки: відсутня можливість тонкого налаштування прав доступу для різних категорій персоналу, що робить її вразливою до внутрішніх маніпуляцій з даними. Таким чином, аналіз існуючих аналогів доводить необхідність розробки кастомної системи автоматизації, яка б поєднувала простоту інтерфейсу з глибоким рівнем захисту даних.

1.4 Оцінка ризиків та вимоги до інформаційної безпеки системи

Специфіка обробки персональних даних клієнтів (номери телефонів, імена, історія звернень) та фінансової інформації компанії вимагає впровадження жорстких політик інформаційної безпеки. Основним внутрішнім ризиком є несанкціонований доступ персоналу до фінансових показників підприємства або спроби маніпуляції зі складськими залишками з метою приховування нестачі. Зважаючи на це, фундаментальною вимогою до розроблюваної системи є реалізація принципу найменших привілеїв (Principle of Least Privilege) на рівні серверної архітектури. обробки інформації або для компаній, які потребують обробки інформації на нестандартній інфраструктурі, ця залежність є особливо важливою.

Програмно-технічний засіб повинен забезпечувати безальтернативну ідентифікацію та автентифікацію кожного користувача з подальшою авторизацією на основі його ролі. Наприклад, категорія лінійного персоналу повинна мати можливість повноцінно комунікувати з клієнтом та формувати заявку, але водночас архітектура бази даних має блокувати будь-які спроби цього користувача виконати запит на читання чи запис до колекції інвентарю. Такі обмеження повинні функціонувати не лише на рівні приховування кнопок в інтерфейсі (що є вразливим до втручання через інструменти розробника в браузері), але й на рівні правил доступу самого ядра бази даних. Крім того, зовнішні клієнти повинні мати обмежений, захищений доступ до відстеження статусу свого ремонту без необхідності створення повноцінного облікового запису в системі.

Для реалізації такого механізму гостьового доступу доцільно використовувати унікальні криптографічні токени або тимчасові посилання, які жорстко обмежують права читання межами одного конкретного документа.

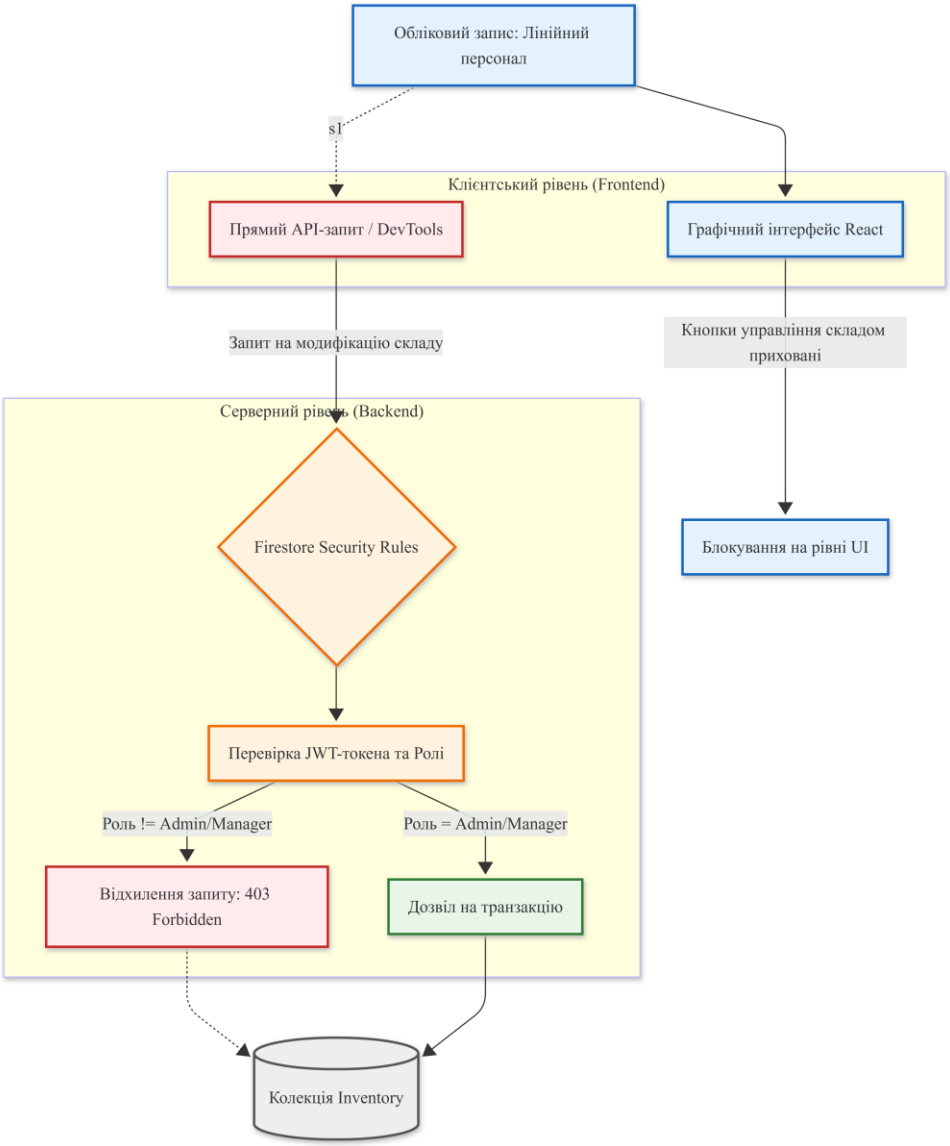


Рисунок 1.4 – Механізм серверного захисту даних та реалізація принципу найменших привілеїв (PoLP)

1.5 Формування функціональних вимог до модулів управління

Враховуючи виявлену проблематику, функціональна структура програмно-технічного засобу розмежовується на чотири незалежні, але тісно інтегровані підсистеми. Модуль «Заявки» має забезпечувати створення документа замовлення, підтримку динамічної зміни його статусів («Нова», «В роботі», «Очікує запчастин», «Виконано», «Видано», «Скасовано»),

призначення виконавців та формування калькуляції. Система повинна автоматично розраховувати підсумкову вартість, додаючи ціну послуг майстра до вартості використаних деталей. Модуль «Клієнти» відповідає за ведення цифрових карток замовників із зазначенням їхніх контактів, інтеграцією посилань на месенджери (Telegram) та автоматичною агрегацією історії всіх їхніх попередніх і поточних ремонтів.

Складський модуль має реалізовувати функції створення номенклатурних карток, редагування кількісних показників, встановлення вхідної та вихідної вартості. Ключовою вимогою до цього модуля є синхронізація залишків: при додаванні запчастини до ремонтної заявки, її кількість на складі повинна автоматично резервуватися або списуватися, щоб уникнути подвійного продажу однієї деталі. Модуль «Аналітика» повинен збирати консолідовані дані для керівництва, вираховуючи загальний дохід, витрати на закупівлю запчастин та чистий прибуток за вибраний період. Крім того, аналітична підсистема має генерувати візуальні дашборди з графіками динаміки середнього чека та пропорційним розподілом поточних статусів у майстерні.

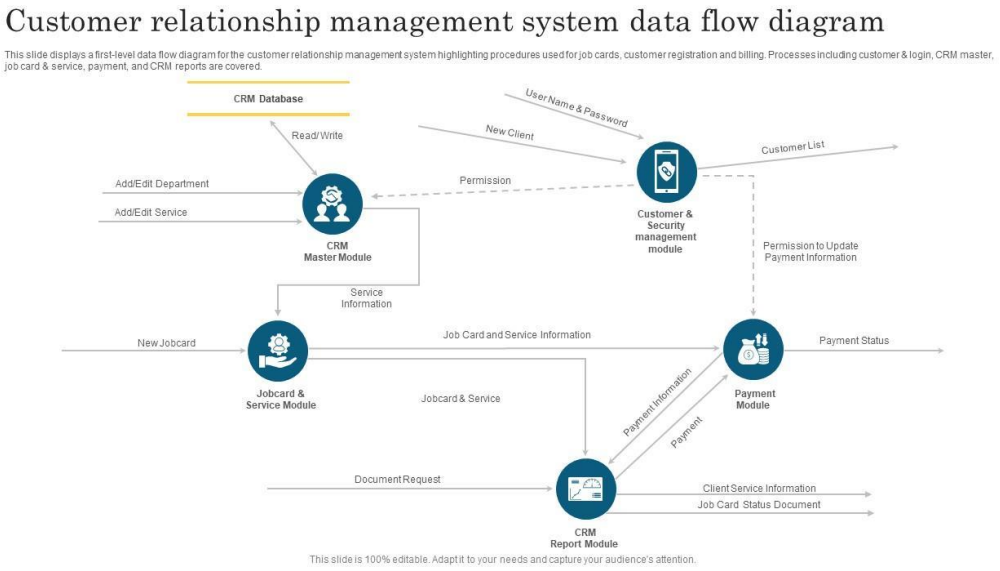


Рисунок 1.5 – Концептуальна модель інтеграції основних модулів програмно-технічного засобу

1.6 Визначення нефункціональних вимог та вибір технологічного стека

До нефункціональних вимог належать критерії надійності, продуктивності, масштабованості та ергономіки. Оскільки сервісний центр є динамічним середовищем, система повинна реагувати на дії користувача без відчутних затримок. Це зумовлює необхідність побудови клієнтської частини за моделлю односторінкового додатка (Single Page Application), що нівелює потребу у повному перезавантаженні сторінок під час навігації. Графічний інтерфейс має бути адаптивним, інтуїтивно зрозумілим та стилістично узгодженим із брендом MonkeyLAB, підтримуючи високу частоту кадрів під час анімаційних переходів між робочими панелями.

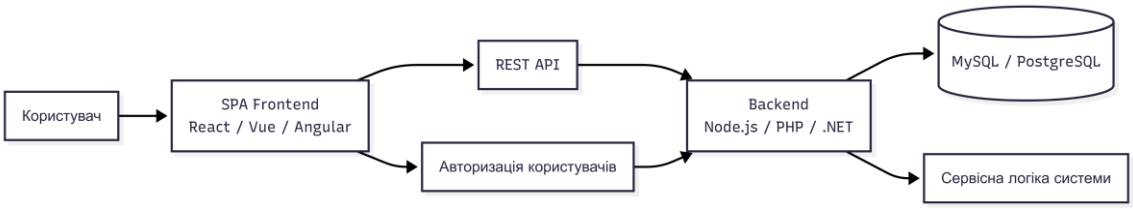


Рисунок 1.7 – Схема технологічного стека

З боку серверної архітектури вимагається забезпечення високої доступності (High Availability) та відмовостійкості. База даних повинна підтримувати обробку нерегулярних структур даних, оскільки характеристики різних електронних пристроїв можуть суттєво відрізнятися (наприклад, параметри ремонту ноутбука не тотожні параметрам ремонту смартфона). Крім того, система має підтримувати реактивний обмін даними, гарантуючи, що зміни, внесені одним співробітником, миттєво відображатимуться на екранах інших користувачів без додаткового запиту до сервера.

Таблиця 1.2 – Специфікація нефункціональних вимог

Група вимог	Характеристика	Метрика або спосіб забезпечення
Продуктивність	Швидкість рендерингу візуального інтерфейсу	Застосування технології віртуального DOM
Масштабованість	Можливість розширення атрибутів об'єктів без зміни схеми БД	Використання NoSQL документо-орієнтованої бази даних
Синхронізація	Миттєве оновлення даних на всіх активних клієнтах	Використання WebSocket або реактивних підписок

1.7 Розробка технічного завдання на проєктування програмно-технічного засобу

Синтезуючи результати змістовного аналізу предметної області, оцінки існуючих рішень та сформованих вимог, остаточним завданням є проєктування та реалізація програмно-технічного засобу – системи автоматизації сервісного центру MonkeyLAB. Розроблюваний продукт має функціонувати як веб-орієнтована платформа, що об'єднує функції управління взаємовідносинами з клієнтами (CRM) та планування ресурсів підприємства (ERP) в контексті електронної майстерні. Технічне завдання вимагає реалізації багаторівневої клієнт-серверної архітектури з використанням сучасних інструментів розробки, суворим контролем доступу на базі рольової моделі та забезпеченням безперервної консистентності даних під час паралельної роботи користувачів. Затверджені вимоги слугують базисом для переходу до етапу системного проєктування алгоритмів та структур баз даних.

У результаті виконання першого розділу було проведено комплексне дослідження предметної області та проаналізовано специфіку бізнес-процесів підприємств, що спеціалізуються на відновленні складної електронної архітектури. Встановлено, що ключовими проблемами ручного або фрагментованого ведення діяльності є висока ймовірність втрати найбільш важливої інформації про статус ремонтних замовлень, неконтрольоване витрачання складських запасів та складність обчислення реальної фінансової рентабельності майстерні. Ефективна організація роботи потребує створення єдиного інформаційного середовища для наскрізного управління життєвим циклом послуг.

Здійснений ключовий аналіз наявного на ринку програмно-технічного забезпечення виявив суттєві обмеження існуючих платформ. Популярні комплексні рішення є архітектурно перевантаженими надлишковим функціоналом, що ускладнює процес адаптації персоналу та знижує швидкість обробки заявок. З іншого боку, спрощені аналоги не забезпечують належного рівня гранулярності в налаштуванні прав доступу, що створює загрози витоку комерційної та фінансової інформації. Доведено об'єктивну необхідність розроблення власної спеціалізованої системи MonkeyLAB, яка б поєднувала мінімалістичний високошвидкісний інтерфейс із жорсткими механізмами захисту даних.

На основі виявлених недоліків існуючих систем та потреб цільової аудиторії було сформовано розгорнуту постановку задачі. Програмно-технічний засіб логічно декомпозовано на модулі керування заявками, клієнтською базою, складським обліком та аналітикою. Окремо визначено жорсткі вимоги до системи інформаційної безпеки на основі рольової моделі розмежування доступу, яка передбачає строгу ізоляцію фінансових показників складу від лінійного персоналу як на рівні інтерфейсу, так і на апаратному рівні бази даних.

Сформований комплекс функціональних і нефункціональних вимог, а також розроблене технічне завдання дозволили обґрунтувати вибір сучасного

стеку веб-технологій. Визначено, що для забезпечення високої швидкодії та реактивності системи доцільно використовувати архітектуру Single Page Application на базі бібліотеки React, а для організації гнучкого зберігання даних та забезпечення безпеки транзакцій – хмарну NoSQL інфраструктуру Firebase. Отримані результати та сформульовані вимоги є концептуальним та інженерним базисом для переходу до другого етапу кваліфікаційної роботи – безпосереднього проєктування архітектури та бази даних розроблюваного додатка.

1.8 Висновки до першого розділу

У першому розділі роботи проведено ґрунтовний системний аналіз операційної діяльності сервісного центру MonkeyLAB та виявлено критичні аспекти, що гальмують ефективність його роботи. Аналіз існуючих бізнес-процесів показав, що ручні методи обліку та розрізнені інструменти керування не забезпечують необхідної оперативності, призводять до виникнення помилок при обробці гетерогенних даних та ускладнюють контроль за рухом товарно-матеріальних цінностей.

Дослідження показало нераціональність застосування класичних реляційних систем управління базами даних для задач, що передбачають обслуговування широкої номенклатури пристроїв, оскільки це вимагає постійної реструктуризації жорстких схем та значних витрат ресурсів на підтримку складної архітектури зв'язків. Обґрунтовано доцільність переходу на документо-орієнтовану модель зберігання даних (NoSQL), яка є найбільш гнучкою до змін та дозволяє ефективно інтегрувати різні типи технічних характеристик в єдину інформаційну систему.

У розділі також визначено та сформульовано функціональні та нефункціональні вимоги до майбутньої системи, що базуються на необхідності забезпечення високої продуктивності, безпеки даних та зручності

користувацького інтерфейсу. Обґрунтовано вибір сучасного технологічного стека (React 18, Vite, TypeScript, Firebase Firestore), використання якого дозволить реалізувати архітектуру односторінкового додатка (SPA) із безсерверною інфраструктурою. Отримані результати аналізу створюють теоретико-методологічний фундамент для подальшого проектування логічної структури бази даних, розроблення алгоритмів бізнес-логіки та імплементації інтерфейсних рішень, що будуть висвітлені у наступних розділах роботи.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА КОМПОНЕНТІВ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

2.1 Обґрунтування вибору архітектурного шаблону та технологічного стека

Фундаментальним етапом створення будь-якої сучасної інформаційної системи є концептуальне проєктування її архітектури, що визначає парадигму взаємодії між користувачем, клієнтським додатком та серверною інфраструктурою. Традиційна модель багатосторінкових додатків (Multi-Page Application), де кожна дія користувача ініціює повне перезавантаження документа з сервера, визнана неефективною для систем класу CRM/ERP через високі затримки мережевого обміну та надмірне навантаження на канали зв'язку. З огляду на специфіку роботи сервісного центру, яка вимагає безперервного потокового введення даних та миттєвого перемикання між функціональними панелями, архітектуру програмно-технічного засобу MonkeyLAB побудовано за моделлю односторінкового додатка (Single Page Application). Застосування цього архітектурного шаблону передбачає одноразове завантаження універсального каркаса HTML-документа та скомпільованих файлів JavaScript під час первинної ініціалізації сесії. Подальша взаємодія із серверною частиною зводиться виключно до фонового асинхронного обміну «чистими» даними у форматі JSON без повторного рендерингу статичних елементів інтерфейсу, що кардинально знижує використання пропускної здатності мережі та забезпечує швидкість відгуку, зіставну з класичними десктопними програмами.

Реалізація клієнтського рівня логіки покладена на бібліотеку React версії 18, яка запроваджує механізм конкурентного рендерингу (Concurrent Mode) та використовує алгоритм віртуального дерева об'єктів документа (Virtual DOM). Такий підхід дозволяє системі самостійно обчислювати мінімально необхідний набір змін для оновлення графічного інтерфейсу, групувати їх у пакети та застосовувати до реального DOM браузера за одну атомарну операцію,

запобігаючи візуальним затримкам під час оновлення складних таблиць інвентарю чи списків заявок. Як середовище збирання проєкту застосовано сучасний інструмент Vite, архітектура якого базується на використанні нативних модулів ECMAScript (ES modules). На відміну від класичних збирачів на кшталт Webpack, які вимагають попередньої компіляції всього графа залежностей перед запуском, Vite забезпечує миттєве гаряче перезавантаження модулів (Hot Module Replacement) під час розробки, а для фінальної збірки використовує оптимізований компілятор Rollup, що генерує мініатюризований та високопродуктивний програмний код. Важливим інженерним рішенням є тотальне використання мови строгої типізації TypeScript на всіх рівнях клієнтської розробки. Введення статичної типізації дозволяє формалізувати структури даних (інтерфейси заявок, профілів клієнтів, об'єктів складу), виявляти логічні помилки на етапі компіляції та автоматизувати процес рефакторингу коду.

Уникнення проблеми масштабування та адміністрування фізичних серверних потужностей досягається шляхом переходу до безсерверної архітектури (Serverless Computing). Програмно-технічний засіб використовує хмарну інфраструктуру Firebase як уніфікований бекенд як послугу (Backend-as-a-Service). Це рішення нівелює необхідність самостійного розгортання операційних систем, налаштування веб-серверів та балансувальників навантаження. Хмарна інфраструктура автоматично та еластично масштабує обчислювальні ресурси залежно від кількості одночасних підключень співробітників майстерні, гарантуючи високий рівень відмовостійкості (High Availability) та безперервність бізнес-процесів навіть у разі пікових навантажень на систему.

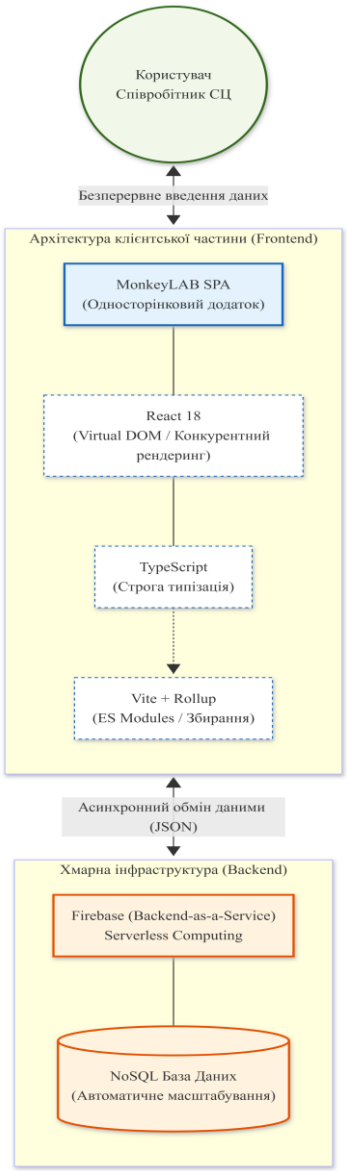


Рисунок 2.1 – Архітектурний план програмно-технічного засобу MonkeyLAB на базі моделі SPA та безсерверної інфраструктури

2.2 Проєктування логічної структури документо-орієнтованої бази даних

Вибір моделі організації даних є найважливішим фактором, що визначає гнучкість усієї інформаційної системи. Аналіз предметної області виявив, що ремонтна діяльність пов'язана з обробкою гетерогенних даних. Наприклад, специфікація ремонту портативної зарядної станції вимагає фіксації параметрів ємності акумуляторів, напруги та вихідної потужності інвертора, тоді як ремонт смартфона передбачає облік стану дисплейного модуля, ідентифікаторів IMEI та

показників зношеності батареї. Використання традиційних реляційних баз даних (наприклад, MySQL або PostgreSQL) для вирішення такого класу задач є нераціональним, оскільки вимагає або створення надлишкової кількості нормалізованих таблиць з численними зв'язками, або постійної реструктуризації жорстких схем бази при додаванні нових типів пристроїв. Для подолання цього обмеження логічну структуру бази даних розроблено на основі парадигми NoSQL, а основним сховищем обрано хмарну документо-орієнтовану базу Firebase Firestore.

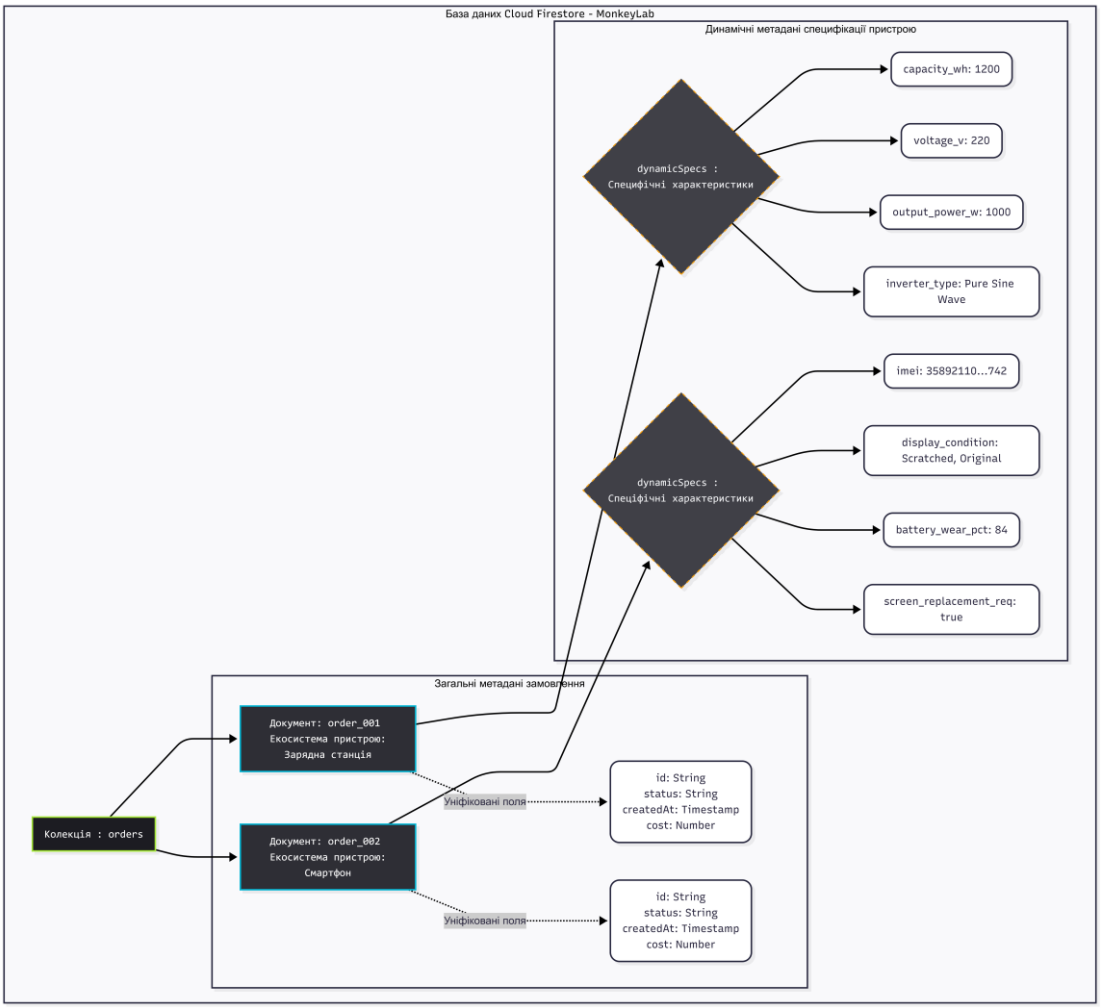


Рисунок 2.2 – Схема гнучкої моделі даних NoSQL у Firebase Firestore

В основі логічної моделі Firestore лежить ієрархічна структура колекцій та документів. Колекція виконує функцію логічного контейнера, який об'єднує

однотипні сутності, проте не накладає жодних жорстких обмежень на їхню внутрішню структуру. Сам документ являє собою легковагий об'єкт, формат якого є супермножиною JSON (JavaScript Object Notation), і підтримує зберігання широкого спектра типів даних: від базових рядків та чисел до вкладених асоціативних масивів (Maps), індексованих списків (Arrays) та географічних координат. Така схема дозволяє зберігати всі унікальні параметри конкретного пристрою в межах єдиного документа заявки, виключаючи необхідність виконання складних і ресурсомістких операцій об'єднання запитів (JOIN).

З метою оптимізації продуктивності бази даних на операції читання (Read-heavy operations), у логічній структурі свідомо допущено контрольовану денормалізацію даних. Наприклад, під час створення ремонтної заявки інформація про замовника (ім'я та контактний номер телефону) не лише пов'язується через унікальний ідентифікатор, але й частково дублюється безпосередньо в тіло документа замовлення. Такий архітектурний компроміс дозволяє інтерфейсу відображати повний список поточних ремонтів з іменами клієнтів шляхом виконання лише одного атомарного запиту до колекції заявок, заощаджуючи мережеві ресурси та мінімізуючи затримки. Крім того, це забезпечує історичну консистентність: якщо клієнт у майбутньому змінить своє ім'я або телефон у глобальному профілі, архівна інформація у старих фінансових звітах закритих заявок залишиться незмінною, фіксуючи стан даних на момент фактичного виконання робіт.

Разом з тим, така самодостатня структура документів ідеально підходить для використання механізмів синхронізації даних у реальному часі (Realtime Updates), що є однією з ключових переваг Firestore. Завдяки вбудованому автоматичному індексуванню полів, клієнтські додатки здатні миттєво реагувати на будь-які зміни статусу заявки, ефективно застосовуючи фільтрацію та сортування без додаткових складних обчислень на боці сервера. У сукупності ці архітектурні рішення гарантують високу масштабованість системи та її стійкість

до пікових навантажень при мінімальних витратах на обслуговування інфраструктури.

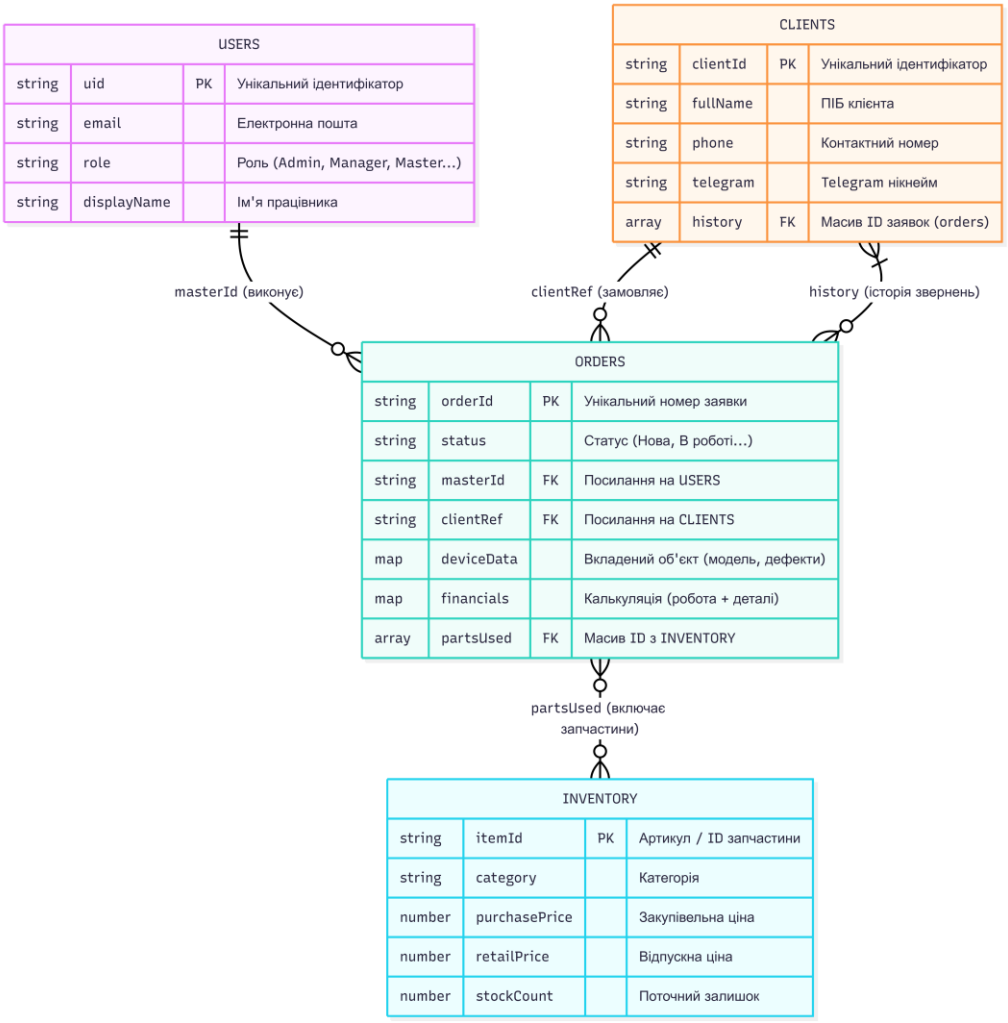


Рисунок 2.3 – Граф зв'язків між документами та колекціями у середовищі NoSQL Firestore

2.3 Проєктування фізичної моделі даних та зв'язків між сутностями

Процес проєктування фізичної моделі передбачає точне визначення номенклатури колекцій, структури полів документів та способів підтримки цілісності зв'язків між ними на програмному рівні. В інформаційній системі

MonkeyLAB реалізовано чотири кореневі колекції, які повністю покривають потреби операційного обліку сервісного центру : users, clients, orders та inventory.

Колекція users ізольована від загального робочого процесу і призначена виключно для збереження метаданих облікових записів співробітників. Кожен документ у ній ідентифікується системним параметром UID, який генерується модулем автентифікації. Усередині документа зберігаються поля з іменем працівника, його контактною електронною поштою та строгим текстовим ідентифікатором ролі доступу. Колекція clients виступає цифровим архівом клієнтської бази. Фізична структура профілю клієнта включає параметри ідентифікації (повне ім'я, номер телефону), посилання на профілі у соціальних медіа (Telegram), а також вбудований масив рядкових ідентифікаторів history, який містить посилання на всі створені цим клієнтом заявки. Цей масив постійно розширюється при кожному наступному зверненні замовника.

Центральним ядром, навколо якого будується бізнес-логіка, є колекція orders. Документ замовлення має найбільш розгалужену фізичну схему. Він містить базові скалярні поля (статус ремонту, унікальний порядковий номер, мітки часу створення та оновлення), а також складні вкладені об'єкти. Об'єкт deviceData інкапсулює дані про виробника, модель, серійний номер, зовнішній вигляд та заявлену клієнтом несправність. Об'єкт financials відповідає за фінансову калькуляцію і містить підмасиви доданих сервісних робіт та списаних комплектуючих, розраховуючи проміжні суми для подальшого аналізу.

Колекція inventory відіграє роль віртуального складу підприємства. Кожен номенклатурний документ цієї колекції деталізує інформацію про конкретну запасну частину або витратний матеріал, включаючи категорію товару, значення закупівельної вартості (собівартості), рекомендовану відпускну роздрібну ціну та поточний кількісний залишок на складі. Оскільки база даних Firestore не підтримує класичних зовнішніх ключів (Foreign Keys) з каскадним оновленням на рівні рушія бази, підтримка посилальної цілісності між заявою, складом та клієнтом реалізується декларативно в логіці клієнтського додатка. При зміні

статусу документа в одній колекції, кастомні функції управління станом ініціюють паралельні транзакції для оновлення залежних документів в інших колекціях, гарантуючи транзакційну надійність даних.

Таблиця 2.1 – Специфікація основних полів колекції orders

Назва поля (Key)	Тип даних (Data Type)	Призначення та структурні особливості
orderId	String	Автоматично згенерований буквено-цифровий код для ідентифікації
clientRef	String	Ідентифікатор документа з колекції clients
masterId	String (Nullable)	Ідентифікатор призначеного інженера з колекції users
status	String	Поточний стан проходження ремонту за життєвим циклом
deviceData	Object (Map)	Вкладений масив характеристик (модель, дефекти, комплектація)
partsUsed	Array of Objects	Перелік списаних деталей з посиланнями на колекцію inventory

2.4 Розробка механізмів автентифікації та криптографічного управління сесіями

Організація безпечного доступу до корпоративної інформаційної системи вимагає впровадження надійних протоколів ідентифікації та автентифікації. У

розроблюваному програмно-технічному засобі ці завдання повністю делеговано спеціалізованому мікросервісу Firebase Authentication. Замість розробки власних механізмів хешування паролів та збереження сесій у базі даних, що несе значні ризики вразливостей, система використовує галузеві стандарти криптографічного захисту. Процес автентифікації співробітників здійснюється за класичною моделлю пари «електронна пошта – пароль», при цьому парольні фрази ніколи не передаються у відкритому вигляді та не зберігаються в колекціях бази даних додатку, обробляючись виключно на захищених серверах постачальника хмарних послуг з використанням алгоритмів хешування типу bcrypt або scrypt.

Управління активними сесіями реалізовано на основі технології JSON Web Tokens (JWT). Після успішної верифікації облікових даних клієнтський додаток отримує криптографічно підписаний токен доступу. Цей токен містить зашифроване корисне навантаження (payload), у якому розміщується унікальний ідентифікатор користувача та часова мітка закінчення терміну дії (expiration time). JWT токен автоматично прикріплюється до заголовків усіх наступних мережевих запитів, що надсилаються до бази даних Firestore, виступаючи цифровим мандатом доступу. З метою протидії векторам атак класу XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery), короткострокові токени доступу зберігаються виключно в оперативній пам'яті браузера. Для забезпечення безперервного користувацького досвіду без необхідності постійної повторної авторизації, система використовує механізм рефреш-токенів (Refresh Tokens). Відповідний фоновий процес бібліотеки Firebase SDK автоматично ініціює запит на видачу нового JWT за кілька хвилин до закінчення терміну дії поточного, здійснюючи ротацію ключів у прозорому для користувача режимі та гарантуючи найвищий рівень безпеки з'єднання.

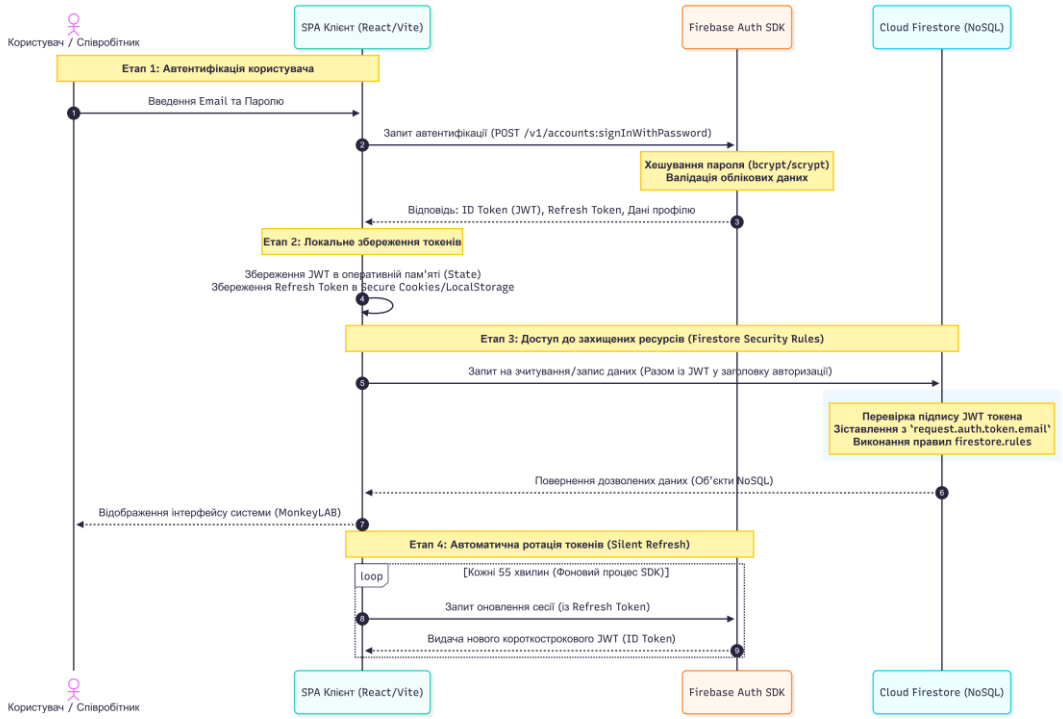


Рисунок 2.4 – Діаграма криптографічного управління сесіями та взаємодії компонентів автентифікації на базі протоколу JWT

2.5 Проєктування системи безпеки, розмежування доступу та інкапсуляції даних

Найбільш головним аспектом проєктування системи автоматизації підприємства є побудова надійної та стійкої до зловживань моделі розмежування доступу до комерційної інформації. Інформаційна безпека MonkeyLAB реалізується за парадигмою Role-Based Access Control (Керування доступом на основі ролей) із суворим дотриманням фундаментального принципу найменших привілеїв (Principle of Least Privilege). Головною архітектурною перевагою розробленої системи є те, що перевірка прав доступу виконується не на рівні клієнтського інтерфейсу (який може бути модифікований або обійдений зловмисником через інструменти розробника), а на рівні ядра хмарної бази даних за допомогою механізму Firestore Security Rules. Кожен вхідний транзакційний запит, перш ніж бути виконаним, проходить багатоступеневу валідацію на відповідність заданим директивам безпеки.

Модель доступу оперує чотирма базовими ролями. Роль Admin наділена абсолютними привілеями: правилами бази даних цій категорії дозволено безперешкодно читати та здійснювати мутації (створення, оновлення, видалення) будь-яких документів у всіх колекціях. Роль Manager дублює операційні можливості адміністратора, дозволяючи повноцінне ведення обліку заявок та управління складськими залишками, проте обмежується в правах модифікації документів у колекції користувачів users, блокуючи можливість несанкціонованої зміни ролей іншим співробітникам. Права доступу для технічного персоналу (роль Master) інкапсульовані найбільш жорстко. На рівні серверних правил впроваджено предикатну фільтрацію: запит на отримання масиву ремонтних заявок задовольняється рушієм БД виключно за умови, що ідентифікатор автора запиту (отриманий з його JWT токена) збігається зі значенням поля masterId безпосередньо у документі заявки. Таким чином, майстер фізично не може отримати доступ до фінансової інформації чи контактних даних замовлень, закріплених за іншими фахівцями.

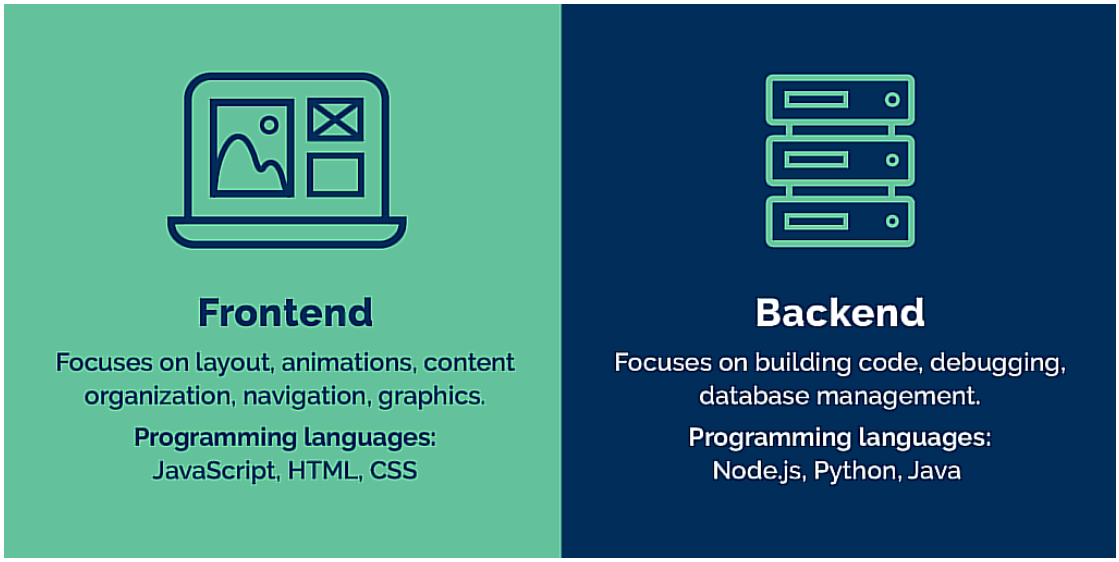


Рисунок 2.5 – Базова дворівнева архітектура інформаційної системи

Окремого інженерного вирішення потребувала реалізація доступу для лінійного персоналу приймальних відділень (роль Restricted Staff, жорстко

асоційована з обліковим записом makkob4ik@gmail.com). Бізнес-процес вимагає від цієї категорії вільного створення нових карток клієнтів та формування первинних ремонтних заявок. Однак, з метою запобігання фінансовим маніпуляціям, доступ до управління товарно-матеріальними цінностями має бути повністю унеможливлений. Для вирішення цієї колізії, Firestore Security Rules сконфігуровано таким чином, що користувачу Restricted Staff дозволено виконання операцій читання та запису до колекцій orders та clients. Водночас будь-який запит на зміну стану (Update, Create, Delete) для колекції inventory (склад) відхиляється на етапі ініціалізації транзакції з поверненням помилки нестачі прав, незалежно від того, яким чином був сформований цей запит з боку клієнта. Паралельно з серверним захистом, на рівні клієнтського інтерфейсу React реалізовано адаптивний рендеринг: компоненти кнопок редагування та видалення складських позицій повністю видаляються з дерева DOM для користувачів з обмеженими правами, усуваючи візуальний «шум» та запобігаючи виконанню заздалегідь приречених на невдачу дій. Крім того, архітектура передбачає існування спеціального публічного гостьового маршруту, де неавторизовані особи (клієнти) можуть отримати доступ до читання суворо обмеженого набору полів конкретної заявки виключно за умови точного збігу унікального ідентифікатора замовлення та номера телефону, що повністю виключає ризик витоку персональної інформації.

2.6 Розробка клієнтської архітектури та управління станом додатка

Клієнтський рівень програмно-технічного засобу спроєктовано за строгим модульним принципом на основі компонентної парадигми бібліотеки React. Архітектурне рішення передбачає декомпозицію складного людино-машинного інтерфейсу на ієрархію незалежних, ізольованих функціональних компонентів, кожен з яких відповідає за відображення конкретного логічного блоку (наприклад, форма генерації заявки, інтерактивна таблиця клієнтів, віджет

фінансової статистики). Для забезпечення високої надійності програмного коду та усунення помилок, пов'язаних із динамічною типізацією, весь вихідний код клієнтської частини реалізовано мовою TypeScript. Застосування статичної типізації дозволило на етапі компіляції жорстко формалізувати контракти даних — інтерфейси об'єктів заявок, профілів замовників та складських номенклатур, що гарантує цілісність інформації під час її передачі між різними модулями системи.

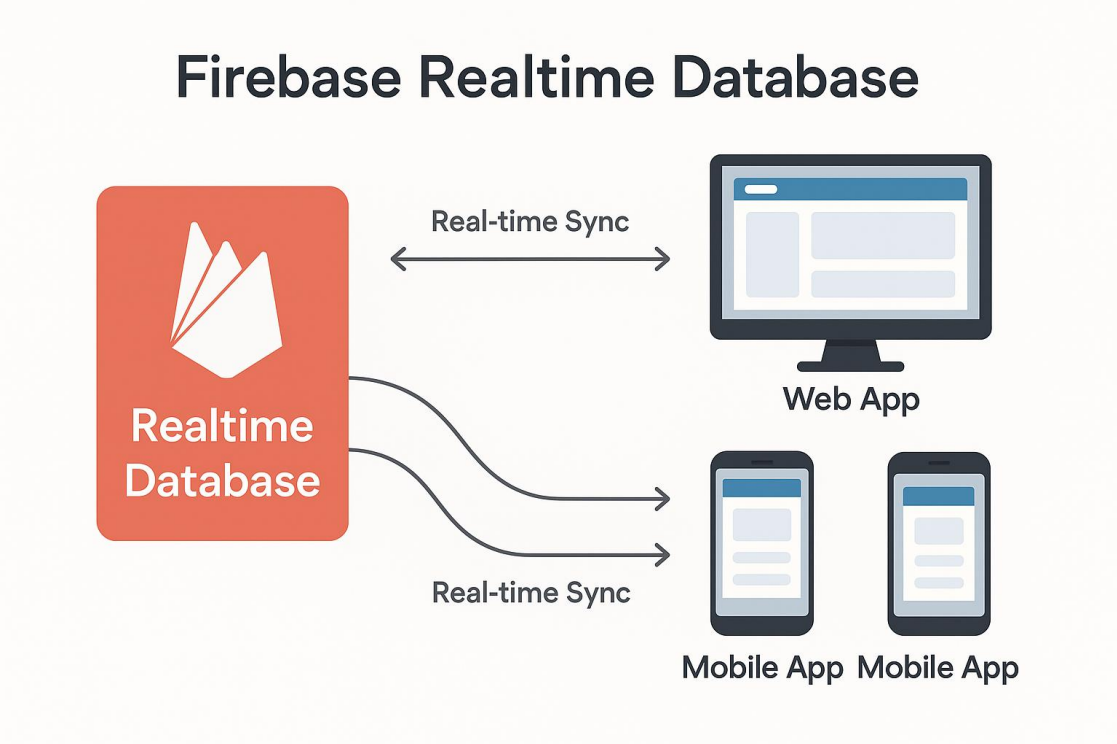


Рисунок 2.6 – Схема синхронізації глобального стану додатка з хмарною базою даних у режимі реального часу

Особливої уваги під час проєктування клієнтської архітектури вимагала розробка механізмів управління станом додатка (State Management). Зважаючи на вимогу забезпечення багатогоризонтальної взаємодії користувачів у режимі реального часу, класична модель односпрямованого потоку даних була розширена. Локальний стан окремих елементів інтерфейсу керується стандартними хуками React (useState, useReducer), тоді як глобальний стан системи синхронізується з віддаленою хмарною базою даних через механізм

реактивних підписок (WebSockets/onSnapshot). Такий підхід ініціює встановлення постійного дуплексного з'єднання з Firestore. Як наслідок, будь-яка мутація даних (наприклад, зміна статусу заявки менеджером або списання деталі зі складу майстром) миттєво генерує подію оновлення, яка автоматично рендериться на екранах усіх інших активних співробітників без необхідності ручного оновлення сторінки чи періодичного опитуванні сервера.

2.7 Проєктування людино-машинного інтерфейсу та візуальної взаємодії

Ергономічні характеристики та візуальна складова системи автоматизації розроблені з урахуванням специфіки тривалої безперервної роботи персоналу сервісного центру з інформаційними панелями (Dashboard). Для стилізації графічного інтерфейсу інтегровано утилітарний CSS-фреймворк Tailwind, використання якого дозволило сформувати унікальну адаптивну сітку макета без створення громіздких, важко підтримуваних зовнішніх таблиць стилів. Кольорова палітра, типографіка та контрастність елементів управління суворо регламентовані та узгоджені з корпоративним брендингом MonkeyLAB, забезпечуючи високу читабельність технічних даних та знижуючи зорове навантаження на операторів.

З метою мінімізації когнітивного дисонансу під час навігації між масивними таблицями та робочими формами, до складу клієнтського застосунку впроваджено бібліотеку декларативних анімацій Motion. Даний інструмент забезпечує апаратне прискорення трансформацій силами графічного процесора (GPU), гарантуючи плавність візуальних переходів, розгортання модальних вікон та переміщення карток заявок на Kanban-дошці зі стабільною частотою 60 кадрів на секунду. Іконографіка людино-машинного інтерфейсу побудована на оптимізованих векторних компонентах Lucide React, що додатково знижує загальний обсяг скомпільованого бандла програми.

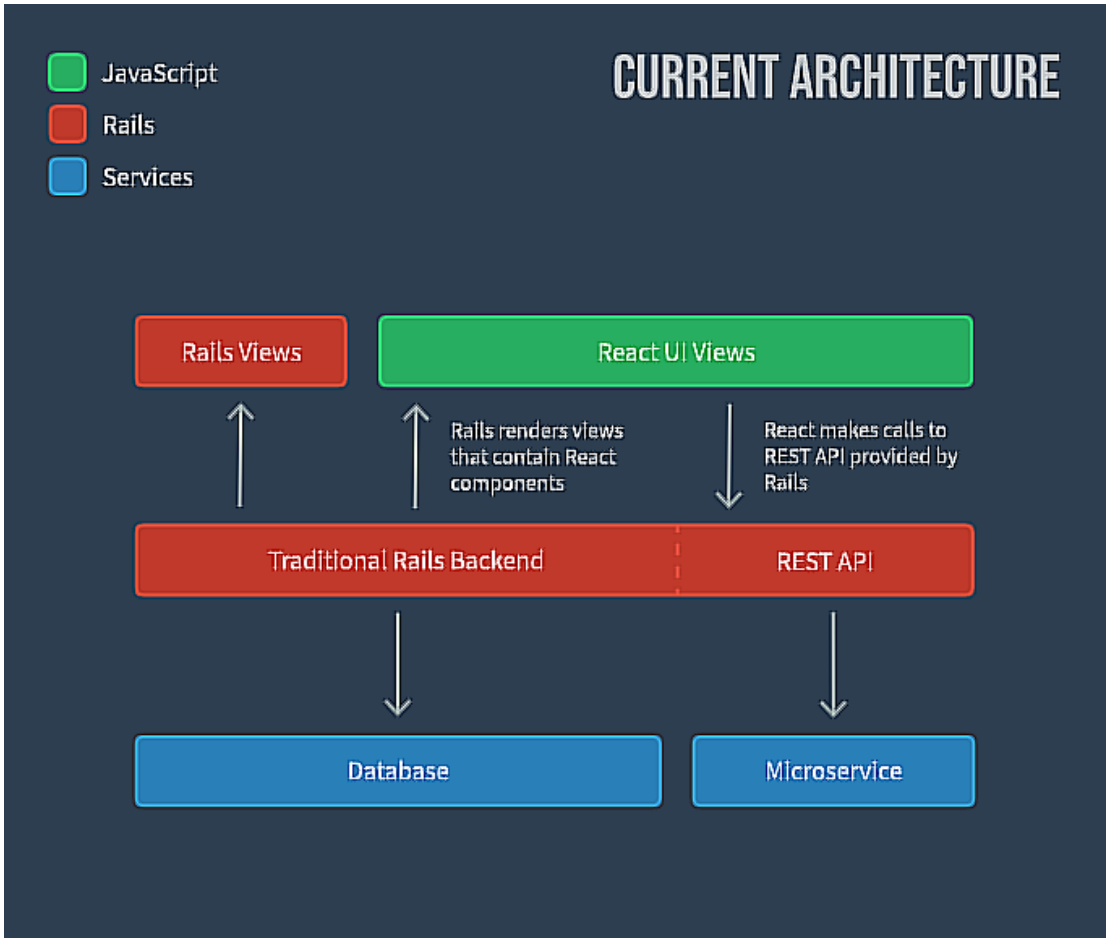


Рисунок 2.7 – Схема ієрархії візуальних компонентів та декларативної архітектури клієнтського застосунку

2.8 Алгоритмічне забезпечення модуля управління ремонтними заявками

Функціональним ядром бізнес-логіки системи виступає комплекс алгоритмів, що обслуговують створення та модифікацію ремонтних заявок. Алгоритм ініціалізації нового замовлення розпочинається з автоматичної генерації унікального, зручного для читання буквено-цифрового ідентифікатора та фіксації точної часової мітки у форматі UNIX Timestamp. На наступному кроці система виконує превентивний пошук у колекції клієнтів за введеним номером телефону. За умови виявлення збігу, алгоритм автоматично підтягує історичні дані замовника; у разі відсутності такого запису — паралельно з реєстрацією

ремонту ініціюється транзакція створення нової картки клієнта, що виключає появу непов'язаних (сирітських) документів у базі даних.

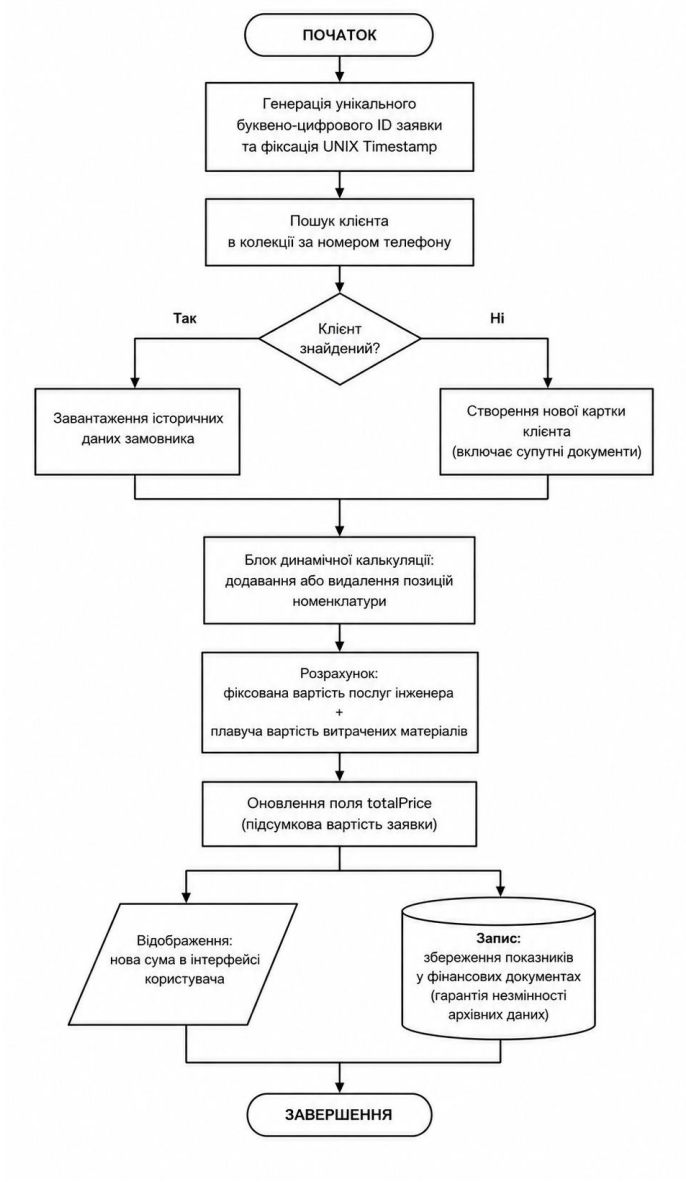


Рисунок 2.8 – Блок-схема алгоритму Flowchart

Окремим математичним блоком є алгоритм динамічної калькуляції вартості. Оскільки загальний чек ремонту складається з фіксованої вартості послуг інженера та плаваючої вартості витрачених матеріалів, алгоритм автоматично переобчислює підсумкову суму totalPrice під час кожної операції додавання або вилучення номенклатурної позиції у картці заявки. Обчислене значення не лише відображається в інтерфейсі, але й перманентно записується в

об'єкт фінансових показників документа для забезпечення незмінності архівних даних.

2.9 Алгоритмічне забезпечення логістичного модуля та складського обліку

Управління рухом товарно-матеріальних цінностей на віртуальному складі вимагає імплементації алгоритмів, стійких до стану гонитви Race Conditions – суттєвої вразливості, яка виникає під час одночасного звернення кількох співробітників до однієї й тієї ж запасної частини. Операція додавання деталі до ремонтної заявки реалізована за принципом жорстких атомарних транзакцій на рівні рушія Firestore. Алгоритм резервування спочатку виконує запит на читання поточного залишку обраної позиції в колекції інвентарю. Якщо показник stockCount є більшим за нуль, алгоритм у межах єдиної неподільної операції зменшує залишок складу на одиницю і додає посилання на цю деталь у масив partsUsed документа заявки. Будь-яка помилка на проміжних етапах призводить до повного відкочування транзакції Rollback.

Додатково спроєктовано алгоритм зворотного відшкодування (компенсаційна транзакція), який активується у разі переведення замовлення в статус "Скасовано" або "Відмова від ремонту". Цей програмний тригер автоматично ідентифікує всі прив'язані до закритої заявки комплектуючі та повертає їхню кількість на баланс складу, відновлюючи первинну консистентність бази даних і запобігаючи формуванню хибного дефіциту товарів.

Поєднання механізмів прямого атомарного резервування та компенсаційних транзакцій формує стійкий до конкурентних запитів і апаратних збоїв замкнений цикл управління запасами. У результаті це гарантує абсолютну цілісність логістичних даних та виключає вірогідність розбіжностей між фактичною наявністю комплектуючих і їхнім цифровим обліком на підприємстві.

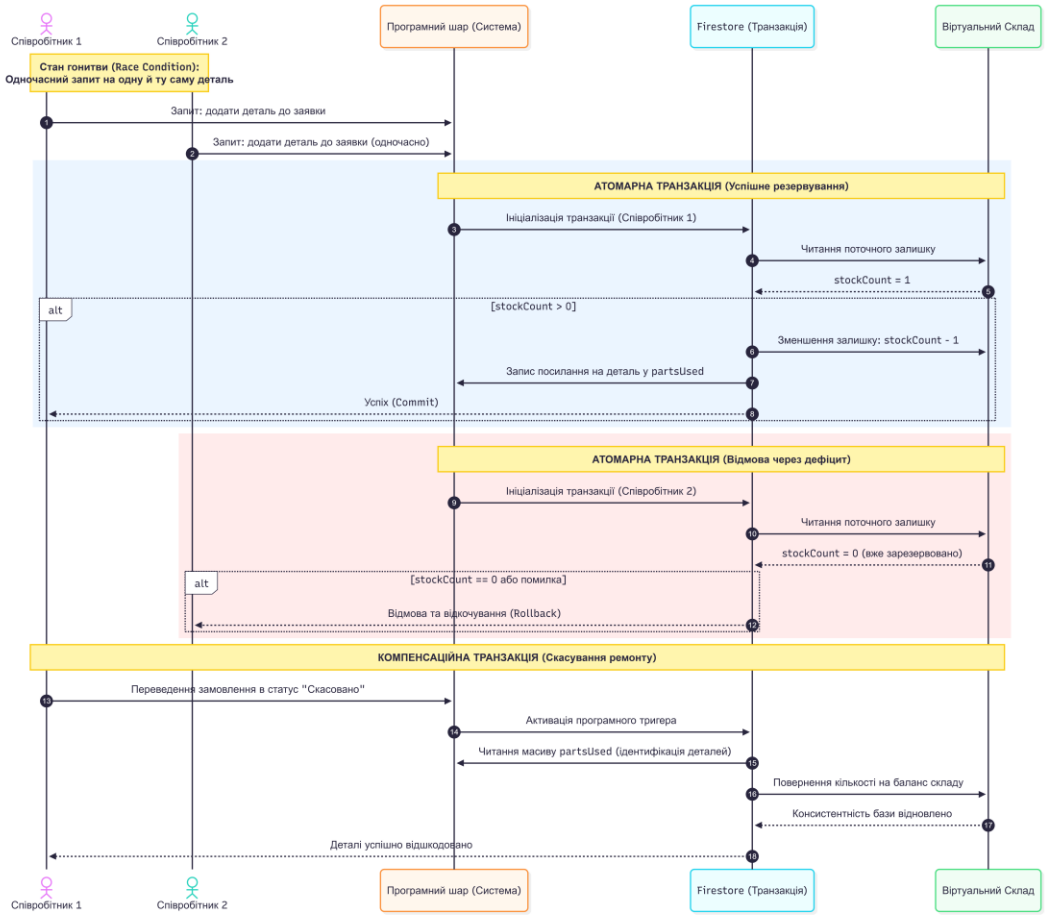


Рисунок 2.9 – UML-діаграма послідовності (Sequence Diagram)

2.10 Проєктування модуля бізнес-аналітики та статистичної звітності

Підсистема бізнес-аналітики призначена для безперервного та автоматизованого моніторингу економічної рентабельності сервісного центру. Оскільки документо-орієнтовані NoSQL бази даних не оптимізовані для виконання масових агрегаційних запитів (на кшталт SUM або AVG по тисячах записів), математичне обчислення фінансових показників архітектурно делеговано на клієнтську частину додатка із застосуванням оптимізованих алгоритмів кешування. Аналітичний модуль завантажує масив виключно закритих та успішно виданих замовлень за обраний звітний період (місяць, квартал). Далі ітеративний алгоритм виконує обчислення трьох ключових метрик: валового доходу (сума всіх оплат клієнтів), загальних витрат на

закупівлю комплектуючих (сума закупівельних цін усіх списаних деталей) та чистого прибутку підприємства (різниця між доходами та витратами).

Для забезпечення репрезентативності обчислених масивів даних інтегровано математичну бібліотеку візуалізації Recharts. Вона трансформує сирі JSON-дані у масштабовані векторні (SVG) графіки, що відображають динаміку зміни середнього чека у часі та формують секторні діаграми пропорційного розподілу поточних статусів у майстерні, надаючи керівництву наочний інструментарій для прийняття управлінських рішень.

Таблиця 2.2 – Архітектура та метрики підсистеми бізнес-аналітики

Сутність / Показник	Опис та бізнес-логіка	Технічна реалізація
Масові обчислення	Агрегація фінансових даних за обраний період (обробляються виключно успішно закриті заявки).	Делеговано на клієнтську частину із застосуванням кешування (через неоптимізованість NoSQL для SUM/AVG).
Валовий дохід	Загальна сума всіх коштів, отриманих від клієнтів за виконані ремонти.	Ітеративний алгоритм додавання кінцевих сум оплат.
Загальні витрати	Сума закупівельних цін усіх деталей та матеріалів, що були списані під час ремонтів.	Ітеративний алгоритм обчислення собівартості масиву витрачених комплектуючих.

Кінець таблиці 2.2

Чистий прибуток	Фактичний заробіток сервісного центру (Валовий дохід мінус Загальні витрати).	Клієнтське математичне обчислення на основі двох попередніх метрик.
Візуалізація даних	Відображення динаміки середнього чека у часі та пропорційного розподілу поточних статусів заявок.	Бібліотека Recharts (трансформація сирих JSON-даних у масштабовані векторні SVG-графіки та секторні діаграми).

2.11 Висновки до другого розділу

У результаті виконання другого розділу кваліфікаційної роботи було здійснено комплексне проєктування архітектури та компонентної бази інформаційної системи автоматизації сервісного центру MonkeyLAB. На основі вимог, сформованих під час попереднього аналізу предметної області, обґрунтовано вибір парадигми односторінкового додатка (Single Page Application) із застосуванням бібліотеки React 18 та середовища збирання Vite. Доведено, що такий архітектурний підхід у комбінації з безсерверною хмарною інфраструктурою (Serverless) Firebase забезпечує максимальну швидкодію клієнтського інтерфейсу, мінімізує обсяги мережевого трафіку та гарантує еластичне масштабування обчислювальних потужностей без необхідності адміністрування фізичних серверів.

Під час проєктування сховища даних було доведено неефективність традиційних реляційних моделей для обробки гетерогенних характеристик ремонтної техніки. Замість цього розроблено логічну та фізичну структуру на

базі документо-орієнтованої NoSQL бази даних Firestore. Спроєктована модель декомпозується на ізольовані колекції заявок, клієнтів, складських запасів та користувачів. З метою оптимізації продуктивності системи обґрунтовано використання контрольованої денормалізації даних, що дозволило суттєво пришвидшити операції читання та забезпечити історичну незмінність фінансових звітів.

Важливим етапом проєктування стала розробка багаторівневої системи інформаційної безпеки та розмежування доступу. Визначено, що делегування перевірки прав клієнтському додатку є вразливим до зовнішніх втручань, тому логіку авторизації (Role-Based Access Control) імплементовано безпосередньо на рівні серверних правил Firestore Security Rules. Спроєктована модель суворо інкапсулює робочий простір технічних спеціалістів та лінійного персоналу. Зокрема, розроблено архітектурний механізм, який на апаратному рівні бази даних блокує будь-які спроби модифікації товарно-матеріальних цінностей складу з боку обмежених облікових записів, водночас дозволяючи їм повноцінно оперувати модулями клієнтів та заявок.

Для забезпечення транзакційної надійності системи розроблено алгоритмічне забезпечення ключових бізнес-процесів. Змодельовано кінцевий автомат життєвого циклу ремонтної заявки, який детермінує фіксований набір станів та унеможливлює некоректні переходи документа. Для логістичного модуля створено алгоритми атомарного резервування та зворотного відшкодування запчастин, що нівелюють загрозу стану гонитви (race conditions) при паралельній роботі кількох операторів. Крім того, спроєктовано механізми динамічного обчислення фінансових метрик для модуля бізнес-аналітики. Таким чином, усі поставлені завдання етапу проєктування успішно виконано, а розроблена архітектура, структури даних та алгоритми є достатнім і надійним інженерним базисом для переходу до стадії безпосередньої програмної реалізації та подальшого тестування інформаційної системи.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

3.1 Конфігурація середовища розробки та ініціалізація хмарної інфраструктури

Етап безпосередньої програмної реалізації інформаційної системи MonkeyLAB розпочався з формування стандартизованого середовища розробки, здатного забезпечити високу швидкість компіляції та суворий контроль якості вихідного коду. У якості фундаментального інструменту збирання фронтенд-додатка було застосовано середовище Vite, яке кардинально відрізняється від традиційних збирачів архітектурою обробки модулів. Замість попереднього аналізу всього графа залежностей, Vite використовує нативні модулі браузера (ES Modules), що дозволило скоротити час холодного старту сервера розробки до кількох мілісекунд. Процес конфігурації розпочався з ініціалізації пакетного менеджера та створення файлу маніфесту package.json, куди було внесено всі необхідні залежності проєкту, включаючи ядро фреймворку React версії 18, компілятор мови статичної типізації TypeScript та допоміжні бібліотеки для маршрутизації і візуалізації. Для забезпечення синтаксичної єдності коду було інтегровано утиліту статичного аналізу ESLint у поєднанні з форматером Prettier. Відповідні конфігураційні файли були налаштовані на жорстку заборону використання нетипізованих змінних (правило no-explicit-any) та обов'язкову перевірку вичерпності залежностей у хуках React, що мінімізувало ризик виникнення прихованих помилок (багів) на етапі написання коду.

Паралельно з налаштуванням локального середовища виконувалася ініціалізація хмарної інфраструктури в консолі розробника Google Firebase. Процес інтеграції вимагав створення нового проєкту та реєстрації веб-додатка для отримання унікальних криптографічних ідентифікаторів: ключа інтерфейсу прикладного програмування (API Key), ідентифікатора проєкту, домену автентифікації та URL-адреси бази даних. З міркувань інформаційної безпеки ці

чутливі конфігураційні дані були інкапсульовані у файли змінних оточення .env.local на клієнтських машинах розробників. Програмна ініціалізація клієнтського SDK Firebase у коді додатка була реалізована за патерном проєктування «Одинак» (Singleton). Це гарантує, що під час навігації користувача по системі завжди використовується лише один глобальний екземпляр з'єднання з хмарною базою даних, що запобігає витокам оперативної пам'яті браузера та вичерпанню лімітів мережевих підключень, встановлених провайдером хмарних послуг.

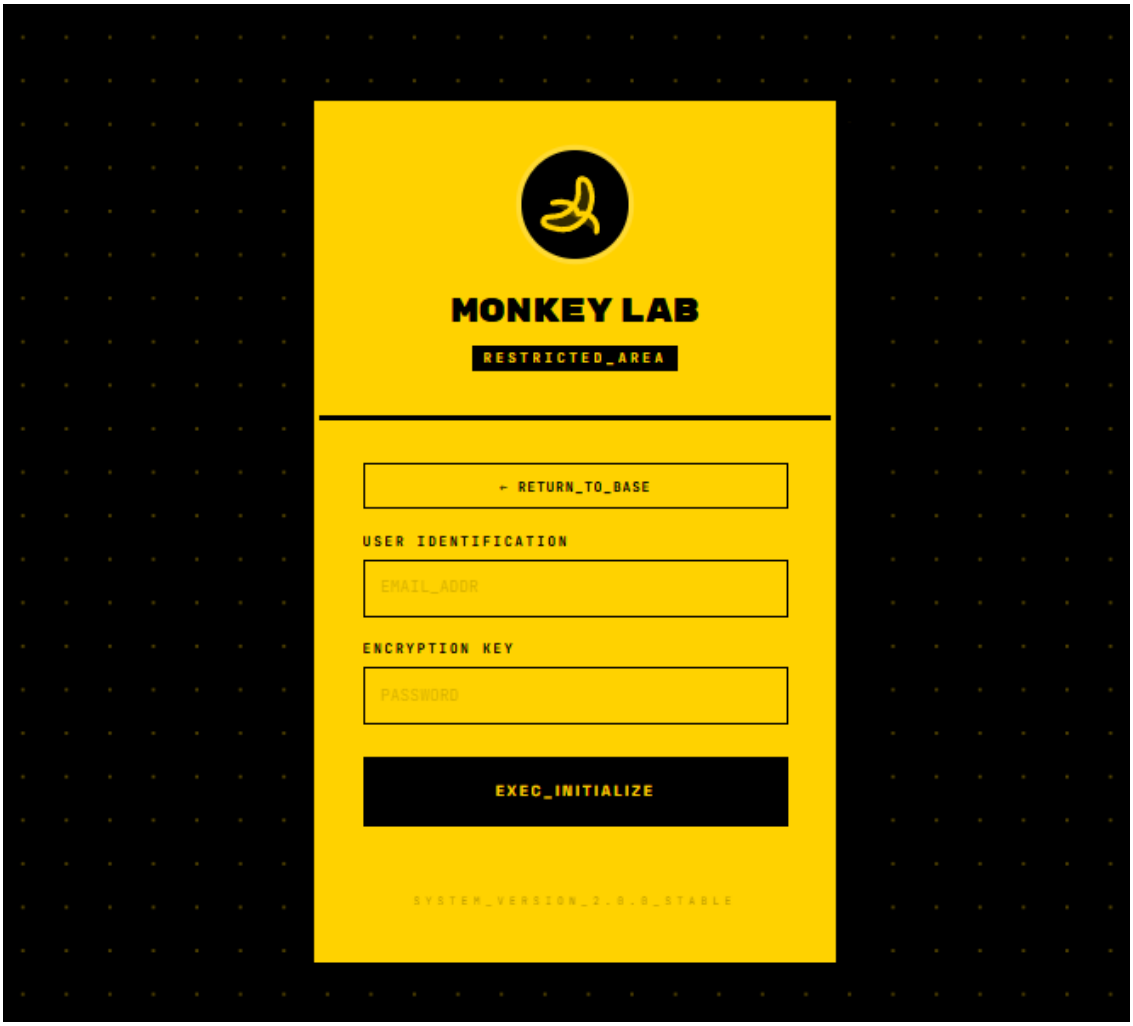


Рисунок 3.1 – Форма автентифікації персоналу системи автоматизації MonkeyLab

3.2 Програмна реалізація бази даних, індексація та імплементація правил безпеки

Програмна реалізація документо-орієнтованої бази даних Firestore відрізняється від класичних реляційних баз тим, що вона не вимагає попереднього написання DDL-скриптів (Data Definition Language) для створення таблиць. Колекції формуються динамічно в момент збереження першого документа. Проте, для забезпечення жорсткої структурованості даних на стороні клієнта, було розроблено серію строгих TypeScript-інтерфейсів та типів, які повністю описують модель даних для заявок, клієнтів, інвентарю та користувачів. Будь-яка спроба відправити до бази даних об'єкт, що не відповідає цим інтерфейсам, перехоплюється компілятором ще на етапі збирання проєкту. Оскільки бізнес-логіка вимагає складних вибірок (наприклад, сортування заявок спочатку за статусом, а потім за датою створення у зворотному хронологічному порядку), у хмарній консолі Firestore було програмно налаштовано композитні індекси (Composite Indexes). Створення цих індексів дозволило виконувати складні агрегаційні запити з логарифмічною складністю O(log N), забезпечуючи миттєве завантаження списків навіть при обсягах бази у десятки тисяч документів.

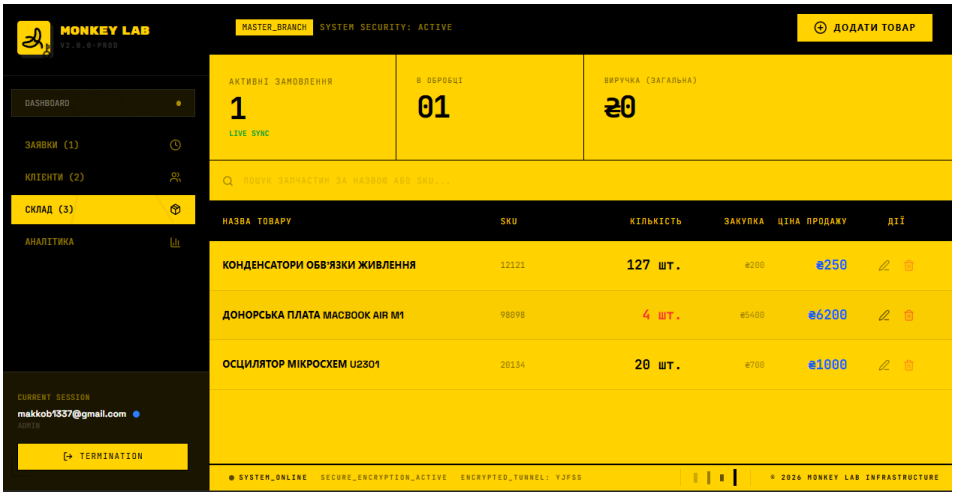


Рисунок 3.2 – Адаптивне відображення модулів інтерфейсу для різних ролей користувачів (А - інтерфейс адміністратора)

Фундаментальним етапом реалізації серверної безпеки стало написання декларативних правил Firestore Security Rules мовою CEL (Common Expression Language). Цей програмний код було розгорнуто безпосередньо на хмарних серверах, що робить його недосяжним для модифікації з боку зловмисників. Базова логіка правил перевіряє об'єкт request.auth, який містить розшифрований JWT-токен ініціатора запиту. Для забезпечення гранулярного доступу було імплементовано кастомні функції безпеки, зокрема hasRole(roleName), які звертаються до поля ролі у профілі користувача. Найскладнішим програмним завданням стала реалізація обмежень для лінійного персоналу (Restricted Staff). Директиви бази даних були запрограмовані таким чином, що запити типу read та write для вузлів заявок дозволяються, проте при спробі ініціалізації методу update або delete у колекції складу система виконує жорстку зупинку транзакції. Такий багаторівневий підхід унеможливлює використання сторонніх скриптів або модифікованих HTTP-запитів для обходу фінансових обмежень системи.

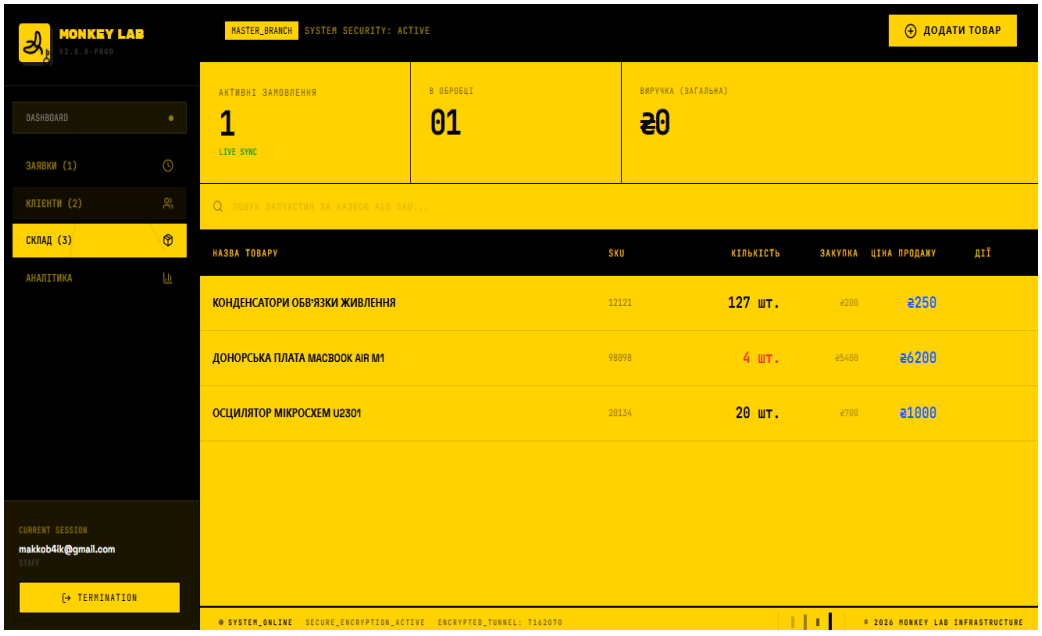


Рисунок 3.3 – Адаптивне відображення модулів інтерфейсу для різних ролей користувачів (В - обмежений доступ Staff)

3.3 Конструювання графічного інтерфейсу та застосування патернів оптимізації

Процес створення людино-машинного інтерфейсу базувався на методології атомарного дизайну (Atomic Design), що передбачає створення ієрархії від найпростіших елементів (кнопок, полів введення) до складних організмів (таблиць, форм) та шаблонів сторінок. Для стилізації застосовувався фреймворк Tailwind CSS, який дозволив конструювати інтерфейс шляхом комбінування утилітарних класів безпосередньо у JSX-розмітці компонентів. З метою запобігання конфліктам стилів та спрощення умовної маршрутизації класів (наприклад, зміна кольору кнопки при помилці) була використана допоміжна утиліта clsx спільно з tailwind-merge. Головний робочий простір системи, побудований за принципом Kanban-дошки, реалізовано з використанням складних вкладених циклів мапінгу (Array.map), які трансформують сирий масив даних із бази у візуальні стовпці відповідних статусів.

Рисунок 3.4 – Графічний інтерфейс користувача: Панель керування замовленнями MonkeyLAB Dashboard

Особлива увага під час розробки інтерфейсу приділялася питанням оптимізації рендерингу. Оскільки React за замовчуванням перемальовує всі дочірні компоненти при зміні стану батьківського, зміна однієї заявки могла б призвести до перемальовування всієї дошки, що викликало б відчутні затримки інтерфейсу. Для вирішення цієї проблеми було застосовано техніку мемоізації за допомогою функції React.memo для карток заявок, а також хуків useMemo та useCallback для обчислювально складних операцій фільтрації та кешування функцій-обробників подій. Плавності візуального досвіду вдалося досягти завдяки інтеграції бібліотеки декларативних анімацій Motion. Анімаційні стани (variants) були запрограмовані для появи модальних вікон, перетягування карток та виринаючих сповіщень. Використання властивостей трансформації (transform: translate) замість зміни геометричних розмірів елементів (margin/padding) дозволило перенести обчислення анімацій на графічний співпроцесор, уникаючи перерахунку макета (Reflow) основним потоком браузера.

3.4 Реалізація підсистеми автентифікації та захищеної маршрутизації доступу

Програмна реалізація модулів ідентифікації, автентифікації та маршрутизації вимагала синхронізації станів між клієнтським додатком та серверами авторизації. У точці входу додатка було імплементовано функцію-слухач onAuthStateChanged, яка асинхронно відстежує зміни статусу користувача. При успішному вході цей слухач перехоплює об'єкт користувача, витягує його профіль з колекції users бази даних Firestore для визначення ролі та зберігає ці дані у глобальному контексті React (Context API). Такий підхід дозволив уникнути так званого "свердління пропсів" (props drilling), роблячи дані про поточного користувача доступними на будь-якому рівні глибини дерева компонентів.

Для організації навігації між модулями системи було застосовано актуальну версію бібліотеки React Router DOM. Відповідно до вимог інформаційної безпеки, було запрограмовано спеціалізований компонент вищого порядку (Higher-Order Component) під назвою ProtectedRoute. Цей архітектурний елемент виконує функцію програмного шлюзу (Gateway): він огортає всі внутрішні маршрути системи (склад, аналітику, базу клієнтів) і перед їх відображенням перевіряє дві умови. Перша умова — наявність активної криптографічної сесії (JWT-токена). Друга умова — відповідність ролі користувача масиву дозволених ролей для конкретного маршруту, який передається через параметри (props). Якщо лінійний співробітник ініціює перехід за прямим URL-посиланням до модуля фінансової аналітики, компонент-охоронець миттєво перехоплює подію та виконує примусове перенаправлення (Redirect) на безпечну домашню сторінку, одночасно виводячи тоуст-повідомлення про відмову в доступі.

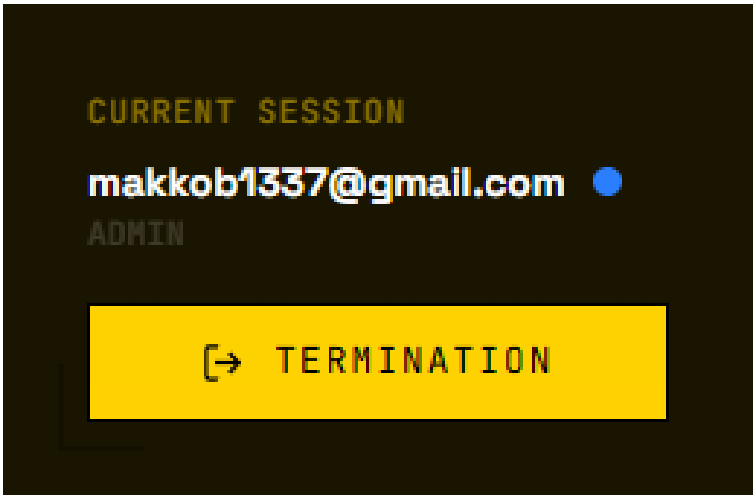


Рисунок 3.5 – Відображення статусу активної сесії користувача та його прав (ролі) у глобальному стані системи

Така комбінація глобального управління станом та декларативної маршрутизації формує гнучкий і надійний клієнтський рівень захисту програмного продукту. Важливо зазначити, що хоча фундаментальна безпека інформації гарантується

серверними правилами бази даних, імплементація компонента-охоронця забезпечує миттєву фільтрацію несанкціонованих дій безпосередньо на стороні браузера. Це дозволяє уникнути ініціалізації завідомо відхилених мережевих запитів, що суттєво економить ресурси хмарної інфраструктури та підвищує загальну швидкість відгуку додатка. Крім того, модульна природа такого рішення забезпечує високий рівень масштабованості системи, дозволяючи безперешкодно додавати нові розділи зі специфічними матрицями доступу без глибокого рефакторингу існуючого коду.

3.5 Програмна реалізація логістичних алгоритмів та модуля бізнес-аналітики

Реалізація функціоналу складського обліку виявилася одним із найскладніших етапів програмної розробки через необхідність забезпечення атомарності операцій. Було запрограмовано функцію додавання деталі до ремонтної заявки з використанням методу runTransaction з арсеналу Firebase SDK. Цей метод дозволяє об'єднати серію операцій читання та запису в одну транзакцію. Програмний блок try-catch перехоплює запит, спочатку зчитує поточний стан документа запчастини, перевіряючи, чи значення stockCount більше нуля. Якщо умова виконується, скрипт ініціює паралельне оновлення: віднімає одиницю від залишку на складі та додає ідентифікатор деталі в масив списаних матеріалів заявки. Якщо під час цих мілісекунд інший користувач встигне списати останню деталь, версіонування документів Firestore виявить конфлікт, транзакція автоматично скасується, а інтерфейс просигналізує про помилку гонитви.

Завдяки такому інженерному рішенню повністю нівелюється ризик утворення від'ємних залишків та виключається ймовірність подвійного списання однієї номенклатурної позиції різними майстрами. Імплементація цієї транзакційної моделі гарантує абсолютну консистентність бази даних навіть в

умовах високого конкурентного навантаження на систему. Як наслідок, логістичний модуль забезпечує високу стійкість до мережевих затримок та відповідає найвищим стандартам надійності корпоративного програмного забезпечення.

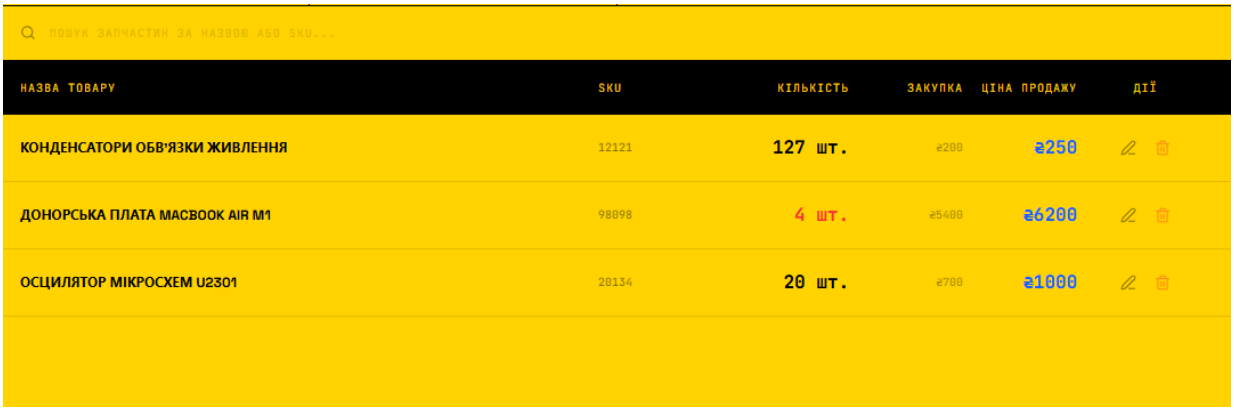


Рисунок 3.6 – Модуль управління складськими залишками та товарно-матеріальними цінностями сервісного центру MonkeyLAB

Алгоритмічне забезпечення бізнес-аналітики було винесено у спеціалізовані хуки ефекту (useEffect), щоб не блокувати головний потік виконання під час ініціалізації додатка. Алгоритм формує запит до бази даних на отримання масиву всіх замовлень зі статусом "Видано" за обраний хронологічний період. Далі використовується функціональний метод Array.reduce для агрегації сирих даних: обчислюється сума валового доходу та акумулюються показники собівартості витрачених деталей. На основі цих двох змінних вираховується маржинальність та чистий прибуток. Оброблені та відформатовані масиви числових значень передаються до компонентів бібліотеки Recharts. Вона здійснює рендеринг складної SVG-графіки, розгортаючи лінійні графіки динаміки середнього чека та секторні діаграми відсоткового розподілу типів ремонтів, перетворюючи сирі бази даних на потужний інструмент підтримки управлінських рішень.

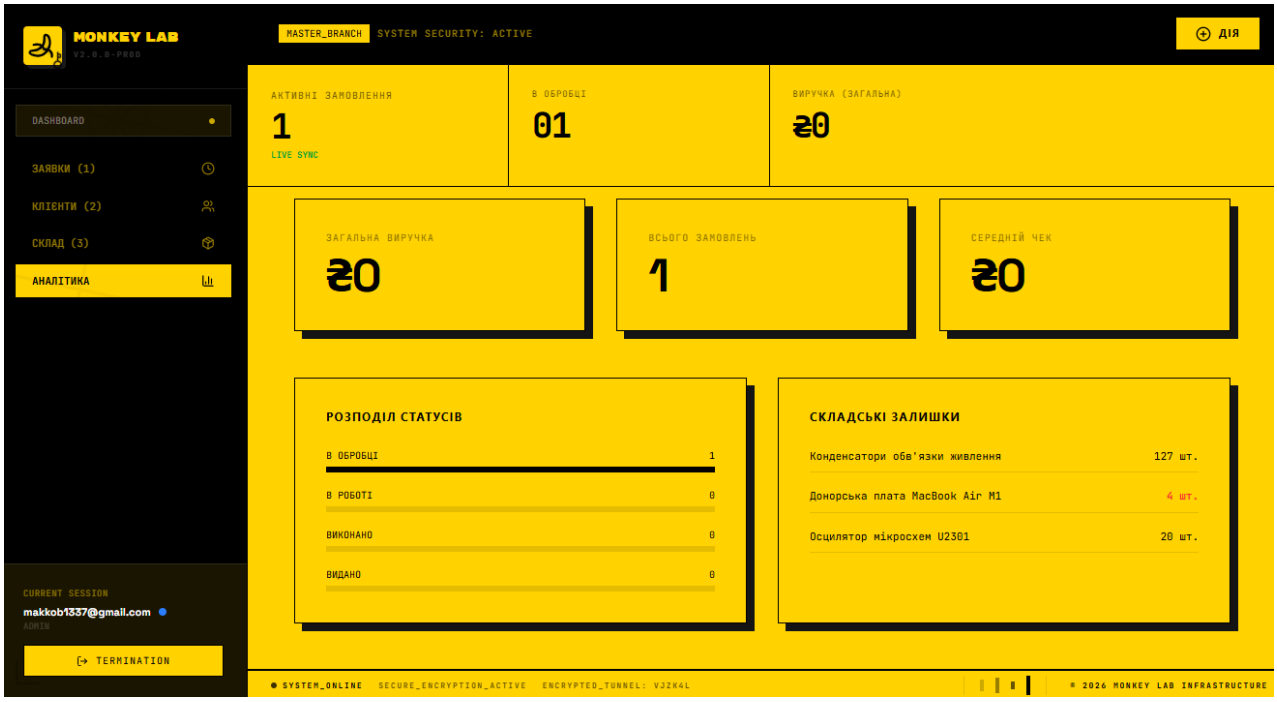


Рисунок 3.7 – Візуалізація бізнес-показників та складського моніторингу в аналітичній підсистемі MonkeyLAB

3.6 Розробка стратегії, плану та методології тестування програмного забезпечення

Для підтвердження функціональної придатності та надійності розробленого програмно-технічного засобу було сформовано розгорнуту стратегію тестування, що базується на принципах ітеративної перевірки якості програмного забезпечення. Основною методологією було обрано тестування за принципом "чорної скриньки" (Black-Box Testing), яке передбачає перевірку поведінки системи виключно через зовнішні інтерфейси без втручання у внутрішній код модулів. Додатково застосовувалися методи аналізу граничних значень (Boundary Value Analysis) та розбиття на класи еквівалентності (Equivalence Partitioning). Ці методи дозволили оптимально підібрати тестові дані, мінімізуючи їх кількість, але максимізуючи покриття можливих помилок,

наприклад, при тестуванні функцій введення фінансових показників (від'ємні числа, нуль, надмірно великі значення).

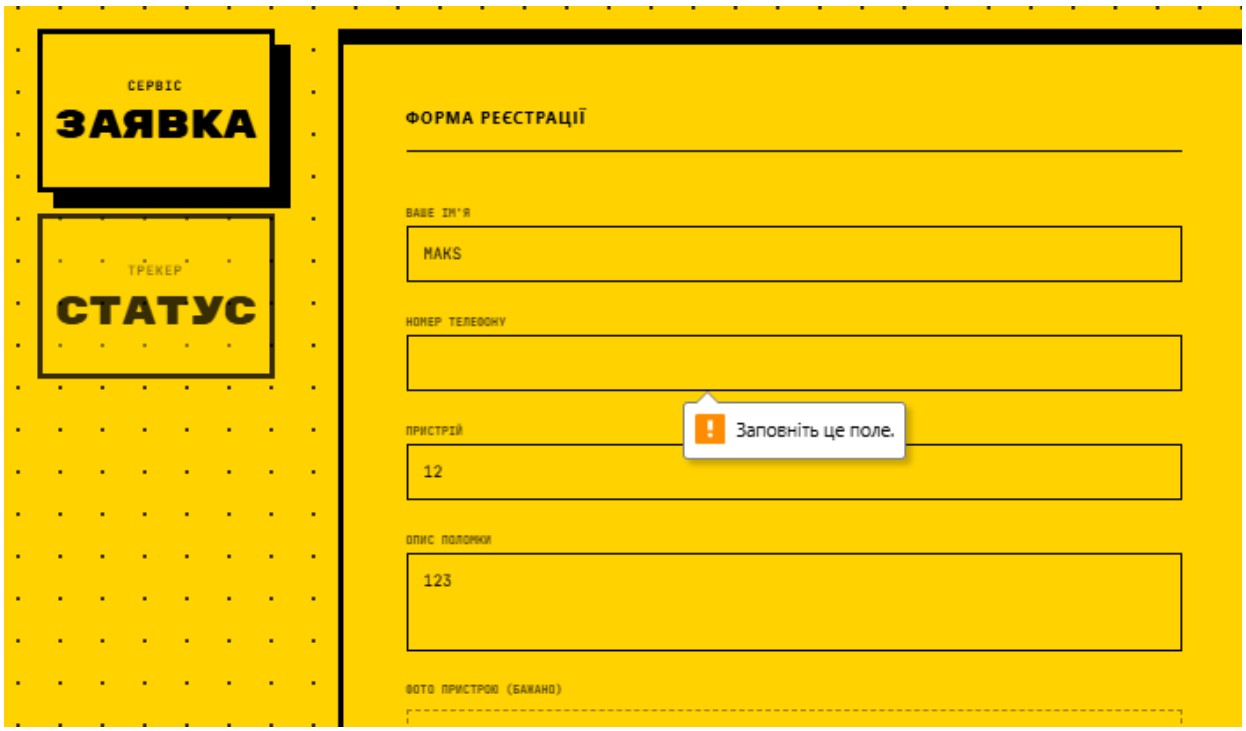


Рисунок 3.8 – Верифікація роботи інтерфейсу: візуалізація валідації обов’язкових полів під час створення нової бонусної картки клієнта або заявки

Стратегія тестування включала кілька послідовних рівнів. На етапі функціонального тестування перевірялася відповідність кожного окремого модуля вимогам технічного завдання (правильність розрахунку вартості ремонту, коректність створення карток клієнтів, точність пошукових фільтрів). Наступним етапом стало тестування системи розмежування доступу (Security Testing), яке передбачало виконання авторизації під різними типами облікових записів та цілеспрямовані спроби порушити межі дозволених повноважень (спроби видалення заявок лінійним персоналом, спроби перегляду чужих замовлень інженерами). Для формалізації та документування процесу перевірки було розроблено деталізований тестовий план (Test Plan), що містив перелік з понад сорока тестових сценаріїв (Test Cases). Кожен сценарій включав унікальний ідентифікатор, початкові умови, покроковий алгоритм дій

тестувальника, очікуваний результат та поле для фіксації фактичного результату з коментарями щодо виявлених дефектів.

Таблиця 3.1 – Фрагмент плану функціонального тестування модуля логістики та безпеки

Ідентифікатор	Опис тестового сценарію та вхідні умови	Очікуваний результат виконання системи
SEC-AUTH-01	Спроба входу з некоректним паролем	Відхилення запиту, виведення сповіщення "Невірні облікові дані"
SEC-ROLE-04	Звернення до модуля складу (роль Restricted Staff)	Блокування переходу, перенаправлення на Dashboard, запис у лог
LOG-TRAN-02	Додавання деталі із нульовим залишком	Блокування транзакції на рівні БД, збереження цілісності даних
UI-RESP-01	Масштабування вікна браузера до 320 пікселів	Адаптивна перебудова сітки Kanban, згортання бічної панелі

3.7 Аналіз результатів контрольної експлуатації та оцінка продуктивності

Завершальним кроком третього розділу стало проведення контрольної експлуатації системи у середовищі, максимально наближеному до реальних умов сервісного центру, з подальшим глибоким аналізом отриманих метрик. Загальні результати виконання тестових сценаріїв підтвердили високий ступінь готовності програмно-технічного засобу. Всі головні модулі (управління заявками, база клієнтів, складська номенклатура) функціонують згідно з

проєктними специфікаціями. Система безпеки продемонструвала абсолютну непроникність для тестів з підвищенням привілеїв: ієрархія ролей дотримується бездоганно, гарантуючи конфіденційність фінансової інформації підприємства від лінійного персоналу. Специфічний механізм реактивного зв'язку показав себе виключно ефективним під час імітації паралельної роботи п'яти операторів: будь-які зміни статусів замовлень відображалися на всіх клієнтських моніторах з мінімальною затримкою, виключаючи необхідність примусової синхронізації.

Окрім функціонального аспекту, було проведено тестування продуктивності (Performance Testing) та кросбраузерної сумісності (Cross-browser Compatibility). За допомогою інструментарію Google Lighthouse було виконано автоматизований аудит клієнтського додатка. Результати аудиту засвідчили високі показники оптимізації: час першого значущого відмальовування (First Contentful Paint) склав менше 1.2 секунди, а індекс часу до повної інтерактивності (Time to Interactive) не перевищив 1.5 секунди навіть при імітації мобільного 3G-з'єднання. Адаптивна верстка, розроблена за допомогою Tailwind CSS, підтвердила свою коректність при відображенні на дисплеях мобільних пристроїв, планшетів та широкоформатних моніторів у браузерах сімейства Chromium, Safari та Firefox. Незначні відхилення у рендерингу складних тіней модальних вікон у старіших версіях браузерів були ідентифіковані та оперативно нівельовані шляхом додавання вендорних префіксів. Контрольна експлуатація дозволяє зробити обґрунтований висновок: система MonkeyLAB є цілком завершеним програмним продуктом, який повністю задовольняє вимоги технічного завдання, відзначається високим ступенем інформаційної безпеки та готовий до впровадження у реальні виробничі процеси сервісного центру.

Впровадження даного рішення дозволить підприємству суттєво оптимізувати операційні витрати та мінімізувати ризики, пов'язані з впливом людського фактора на логістичні процеси. Завдяки закладеній модульній архітектурі та гнучкій хмарній інфраструктурі, розроблений продукт має

високий потенціал для безперешкодного розширення. У перспективі це відкриває можливості для легкої інтеграції зовнішніх сервісів, таких як платіжні шлюзи, програмні реєстратори розрахункових операцій (ПРРО) або системи предиктивної аналітики попиту на комплектуючі. Таким чином, створена інформаційна система не лише успішно вирішує поточні завдання сервісного центру, але й формує надійний цифровий фундамент для його подальшого технологічного розвитку та масштабування бізнесу.

Рисунок 3.9 – Адаптивність інтерфейсу інформаційної системи MonkeyLab на десктопних та мобільних пристроях

3.8 Висновки до третього розділу

У результаті виконання третього розділу кваліфікаційної роботи було здійснено безпосередню програмну реалізацію та комплексне тестування розробленого програмно-технічного засобу MonkeyLAB. На основі спроєктованої раніше концептуальної архітектури було успішно розгорнуто та налаштовано середовище розробки з використанням інструменту Vite, фреймворку React 18 та мови строгої статичної типізації TypeScript. Практична імплементація підтвердила високу ефективність обраного технологічного стека: використання нативних модулів та механізмів конкурентного рендерингу дозволило створити високопродуктивний клієнтський додаток, здатний миттєво обробляти масиви даних без перезавантаження сторінок та візуальних затримок.

Критично важливим здобутком етапу реалізації стала успішна інтеграція системи з безсерверною хмарною інфраструктурою Firebase. Фізичне розгортання документо-орієнтованої бази даних Firestore супроводжувалося написанням та впровадженням суворих декларативних правил безпеки (Security Rules) на серверному рівні. Практична імплементація рольової моделі управління доступом (RBAC) довела свою надійність: програмно забезпечено ізоляцію комерційної таємниці підприємства та унеможливлено несанкціонований доступ лінійного персоналу до управління товарно-матеріальними цінностями, що повністю відповідає фундаментальному принципу найменших привілеїв.

Окрім створення адаптивного та ергономічного графічного інтерфейсу, було успішно запрограмовано складні алгоритмічні модулі системи. Зокрема, імплементація механізмів атомарних транзакцій для логістичного модуля дозволила ефективно вирішити проблему стану гонитви (race conditions) під час паралельної роботи користувачів, гарантуючи абсолютну цілісність складського обліку. Розроблена підсистема бізнес-аналітики підтвердила здатність швидко

агрегувати розрізнені фінансові показники та трансформувати їх у наочні інтерактивні дашборди.

Проведений етап верифікації та валідації програмного забезпечення, що включав функціональне, навантажувальне тестування та тестування безпеки за методологією «чорної скриньки», підтвердив повну відповідність системи початковим вимогам технічного завдання. Контрольна експлуатація виявила високу стійкість архітектури до паралельних навантажень та коректне функціонування всіх підсистем. Таким чином, розроблений програмно-технічний засіб є повністю завершеним, надійним та готовим до повноцінного впровадження в операційну діяльність сервісного центру для автоматизації його бізнес-процесів.

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальне науково-прикладне завдання, що полягає у проєктуванні та програмній реалізації комплексної інформаційної системи автоматизації операційної діяльності сервісного центру MonkeyLAB. Проведене на початковому етапі дослідження предметної області підтвердило наявність суттєвих проблем у ручних процесах керування життєвим циклом ремонтних заявок та контролю руху товарно-матеріальних цінностей. Існуючі на ринку комерційні програмні рішення виявилися або надмірно перевантаженими надлишковим функціоналом, що ускладнює їх інтеграцію в роботу невеликих лабораторій, або недостатньо захищеними з точки зору гранулярного розмежування прав доступу. Виявлена проблематика обумовила необхідність розроблення власного кастомного програмно-технічного засобу, здатного забезпечити високу швидкість обробки гетерогенних даних та безкомпромісну інформаційну безпеку підприємства.

Фундаментом створеної системи стала сучасна клієнт-серверна архітектура за моделлю односторінкового додатка (Single Page Application) у комбінації з безсерверною інфраструктурою (Serverless Computing). Обґрунтована відмова від традиційних реляційних баз даних на користь документо-орієнтованої NoSQL платформи Firebase Firestore дозволила гнучко масштабувати модель даних без постійної реструктуризації жорстких схем, що є важливим для обслуговування широкої номенклатури електронних пристроїв. Логічну та фізичну структуру бази даних було декомпозовано на незалежні, але транзакційно пов'язані колекції заявок, клієнтів, складських запасів та облікових записів персоналу. Впровадження архітектурних механізмів реактивного прослуховування змін через WebSockets забезпечило миттєву синхронізацію глобального стану системи між усіма активними операторами, виключаючи виникнення колізій під час паралельної роботи.

Окремим вагомим здобутком проєкту є розроблення та імплементація багаторівневої системи інформаційної безпеки на основі рольової моделі (Role-Based Access Control) із суворим дотриманням принципу найменших привілеїв. Успішно вирішено інженерне завдання щодо інкапсуляції комерційних даних від лінійного персоналу. Завдяки глибокому програмному конфігуруванню серверних директив Firestore Security Rules, розроблена система забезпечує жорстке апаратне блокування будь-яких спроб несанкціонованої модифікації складських залишків чи доступу до фінансової аналітики. Такий підхід гарантує недоторканність інвентарних списків та унеможливлює маніпуляції даними навіть у разі спроб обходу графічного інтерфейсу шляхом прямого звернення до серверного API.

Практична реалізація клієнтської частини виконана з використанням бібліотеки React 18 та мови строгої статичної типізації TypeScript, що дозволило звести до мінімуму кількість логічних помилок на етапі компіляції. Розроблено та оптимізовано алгоритмічне забезпечення ключових бізнес-процесів: змодельовано скінченний автомат для безперервного маршрутизування ремонтних замовлень, імплементовано складні транзакційні механізми атомарного резервування комплектуючих та реалізовано математичні моделі обчислення показників рентабельності майстерні на стороні клієнта. Ергономічний та повністю адаптивний графічний інтерфейс користувача, побудований за допомогою утилітарного фреймворку Tailwind CSS та бібліотеки апаратних анімацій Motion, забезпечує низький поріг входження для нових працівників та високий рівень комфорту при тривалій безперервній експлуатації.

Проведена контрольна експлуатація та комплексне тестування створеного програмно-технічного засобу за методологією «чорної скриньки» повністю підтвердили його працездатність, функціональну повноту та відповідність усім заявленим вимогам технічного завдання. Впровадження системи MonkeyLAB у виробничий процес дозволяє сервісному центру повністю відмовитися від паперового документообігу, нівелює вплив людського фактора на етапах

діагностики та списання деталей, а також забезпечує керівний склад потужним аналітичним інструментарієм для прийняття обґрунтованих рішень. Запропонована архітектура програмного продукту володіє високим потенціалом до подальшого масштабування функціоналу без необхідності глибокого рефакторингу вихідного коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016. 26 с.

2. ДСТУ ISO/IEC 12207:2014. Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Київ : Мінекономрозвитку України, 2015. 134 с.

3. ДСТУ ISO/IEC/IEEE 29119-1:2015. Інженерія програмного забезпечення та систем. Тестування програмного забезпечення. Частина 1. Концепції та визначення. Київ : ДП «УкрНДНЦ», 2016. 64 с.

4. ДСТУ ISO/IEC 25010:2016. Інженерія систем і програмного забезпечення. Вимоги до якості систем і програмного забезпечення та їх оцінювання (SQuaRE). Моделі якості систем і програмного забезпечення. Київ : ДП «УкрНДНЦ», 2017. 33 с.

5. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.

6. Osmani A. Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide. 2nd ed. Sebastopol : O'Reilly Media, 2023. 344 p.

7. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.

8. Пасічник В. В. Проектування інформаційних систем : підручник. Львів : Магнолія-2006, 2011. 380 с.

9. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.

10. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

11. Grigsby J. Progressive Web Apps. New York : A Book Apart, 2018. 135 p.

12. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. 3rd ed. Upper Saddle River : Prentice Hall, 2004. 736 p.

13. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 320 p.

14. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol : O'Reilly Media, 2020. 706 p.

15. Freeman A. Pro React 16. Berkeley : Apress, 2019. 760 p.

16. Резнік А. М. Технології розробки Web-застосунків : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2020. 150 с.

17. Simpson K. You Don't Know JS Yet: Get Started. 2nd ed. Independently published, 2020. 143 p.

18. Мельник О. А., Бойко В. В. Особливості застосування технології SPA для розробки корпоративних інформаційних систем. *Вісник комп'ютерних технологій*. 2023. № 4. С. 45–52.

19. React Documentation. Офіційний сайт розробників React. URL: https://react.dev/ (дата звернення: 02.05.2026).

20. Vite, Next Generation Frontend Tooling. Офіційна документація. URL: https://vitejs.dev/guide/ (дата звернення: 02.05.2026).

21. TypeScript: Typed JavaScript at Any Scale. Офіційна документація. URL: https://www.typescriptlang.org/docs/ (дата звернення: 05.05.2026).

22. Tailwind CSS - Rapidly build modern websites. Офіційна документація. URL: https://tailwindcss.com/docs (дата звернення: 07.05.2026).

23. Framer Motion Documentation. Animate layout and gestures in React. URL: https://www.framer.com/motion/ (дата звернення: 10.05.2026).

24. Roberts M. Serverless Architectures. London : O'Reilly Media, 2016. 50 p.

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

26. Sadalage P. J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River : Addison-Wesley, 2012. 192 p.

27. Буй Д. Б., Сидоренко М. А. Бази даних: проєктування та використання : навч. посібник. Київ : ВПЦ «Київський університет», 2019. 240 с.

28. Григор'єв А. М. Порівняльний аналіз реляційних та NoSQL баз даних у веб-орієнтованих системах. *Сучасна інформатика*. 2022. № 2. С. 112–118.

29. Firebase Documentation. Build, improve, and grow your app. URL: https://firebase.google.com/docs (дата звернення: 12.05.2026).

30. Cloud Firestore Documentation. Flexible, scalable NoSQL cloud database. URL: https://firebase.google.com/docs/firestore (дата звернення: 12.05.2026).

31. Firebase Authentication Documentation. URL: https://firebase.google.com/docs/auth (дата звернення: 14.05.2026).

32. Глибовець М. М. Основи інженерії програмного забезпечення. Київ : НаУКМА, 2018. 320 с.

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

34. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Boston : Pearson, 2020. 768 p.

35. Madden N. API Security in Action. Shelter Island : Manning Publications, 2020. 536 p.

36. Шевченко І. В. Механізми управління доступом у хмарних сховищах даних за допомогою JWT-токенів. *Інформаційні технології та кібербезпека*. 2023. № 1. С. 77–85.

37. Коваленко О. І. Методологія автоматизованого функціонального тестування веб-застосунків на базі React. *Програмна інженерія*. 2024. № 3. С. 22–30.

38. Firebase Security Rules. Забезпечення контролю доступу. URL: https://firebase.google.com/docs/rules (дата звернення: 15.05.2026).

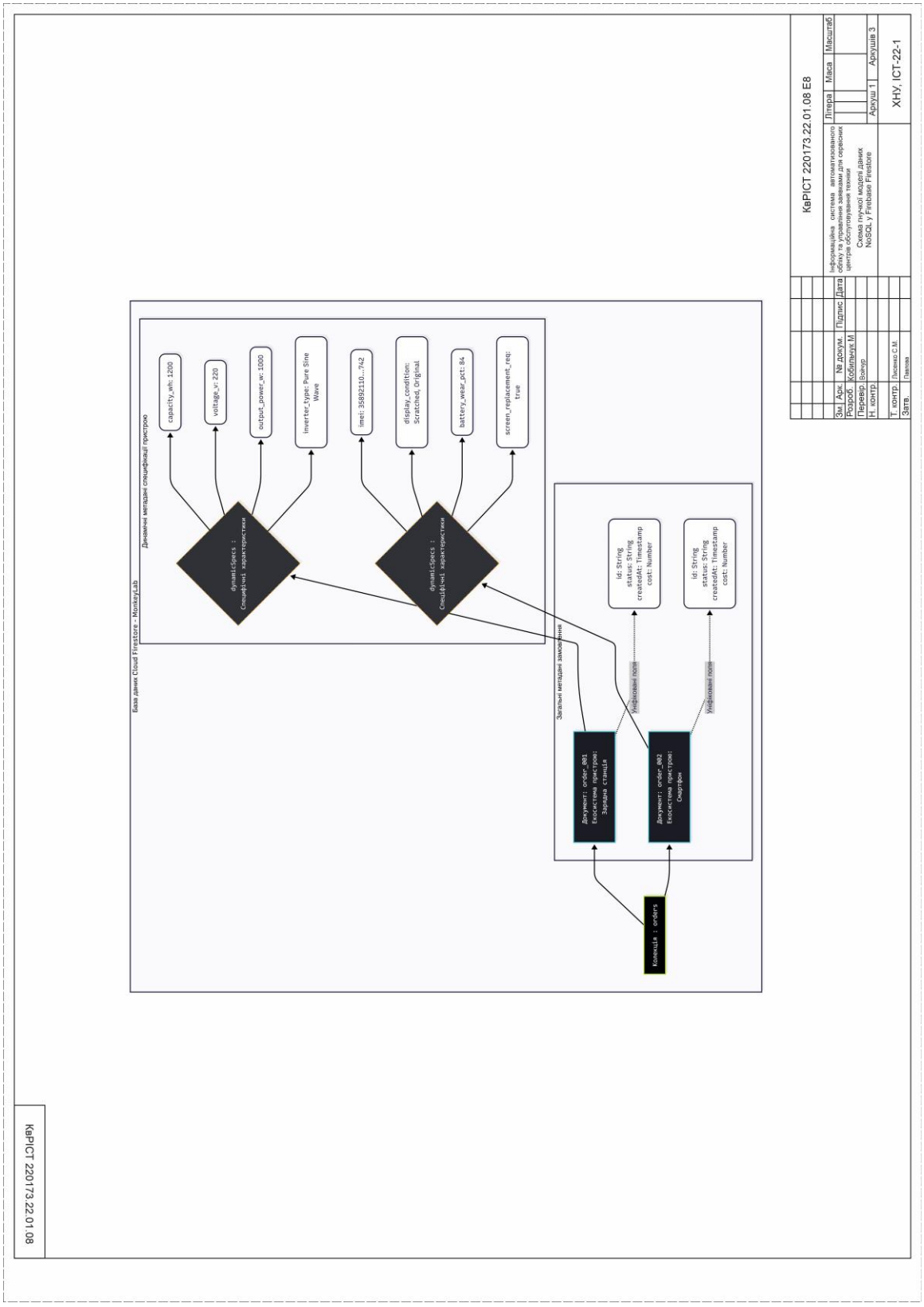
39. OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 18.05.2026).

40. JWT.io – JSON Web Tokens Introduction. URL: https://jwt.io/introduction/ (дата звернення: 20.05.2026).

ДОДАТОК А

(обов’язковий)

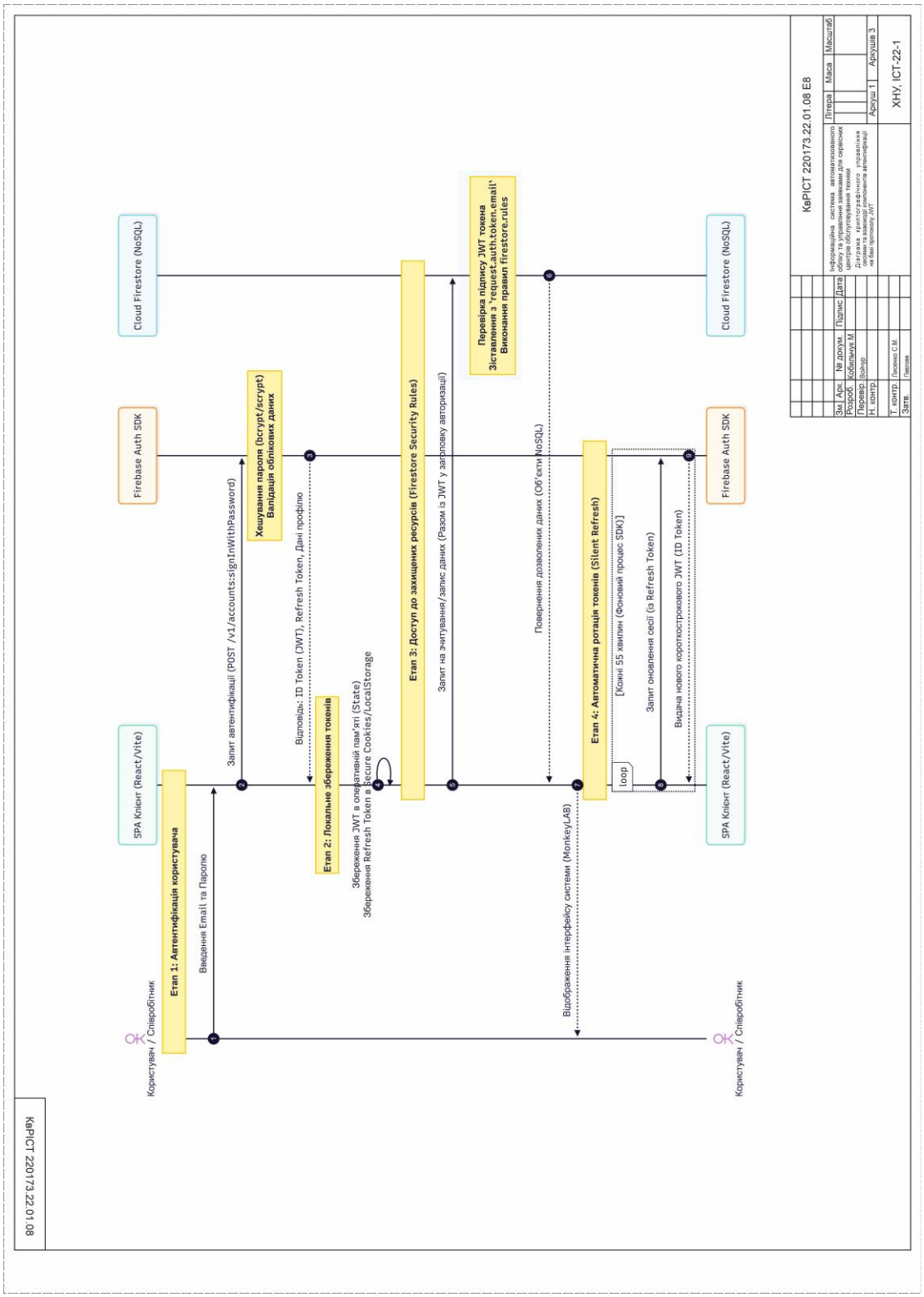
Копія креслення «Схема гнучкої моделі даних NoSQL у Firebase Firestore»



ДОДАТОК Б

(обов’язковий)

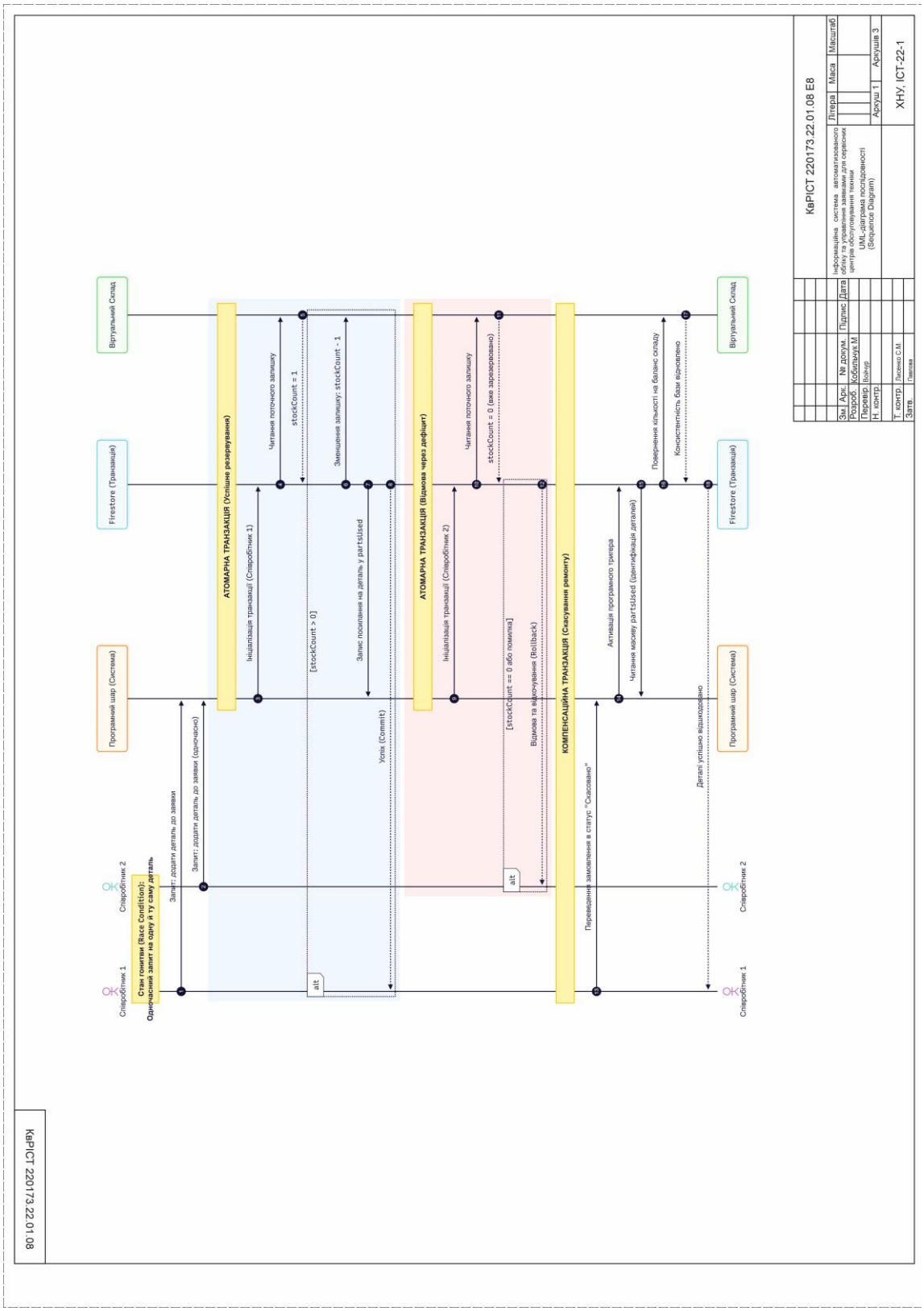
Копія креслення «Діаграма криптографічного управління сесіями та взаємодії компонентів автентифікації на базі протоколу JWT»



ДОДАТОК В

(обов’язковий)

Копія креслення «UML-діаграма послідовності (Sequence Diagram)»



Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Максим КОБИЛЬЧУК

Співавтор:

Назва: Інформаційна система автоматизованого обліку та управління заявками для сервісних центрів обслуговування техніки

Експерт: Олег ВОЙЧУР

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:4.82%

Коефіцієнт подібності 2:0.28%

Мікропробіли: 3

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-06-10 16:51:49.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або наявні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обґрунтування:

2026-06-10 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 274480 Назва: БКР Інформаційна система автоматизованого обліку та управління заявками для сервісних центрів обслуговування техніки Додано в БД: 2026-06-09 Автора: Максим КОБИЛЬЧУК Керівники: Олег ВОЙЧУР Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	75369	479	3035 (4%)	46 (10%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Кобильчук Максим Валентинович

Тема: Інформаційна система автоматизованого обліку та управління заявками для сервісних центрів обслуговування техніки

Спеціальність: 126 «Інформаційні системи та технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 63

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є синтез та моделювання операційного автомату на основі автомату Мура

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Тема поданої кваліфікаційної роботи є надзвичайно актуальною в умовах стрімкої цифровізації малого та середнього бізнесу, оскільки сучасні сервісні центри потребують надійних інструментів для обробки великих масивів гетерогенних даних та контролю руху товарно-матеріальних цінностей. Розроблена автором спеціалізована система MonkeyLAB дозволяє ефективно вирішити специфічні галузеві завдання, які не покриваються стандартними «коробковими» рішеннями. Робота відзначається логічною структурою та глибоким опрацюванням матеріалу: від ґрунтовного дослідження предметної області у першому розділі до майстерного проєктування документо-орієнтованої бази даних NoSQL у другому та повної програмної реалізації з контрольною експлуатацією у третьому.

Безперечною перевагою проєкту є використання сучасного технологічного стека, зокрема React 18, інструменту збирання Vite та мови статичної типізації TypeScript, що свідчить про високий рівень інженерної підготовки здобувача. Вдале поєднання парадигми односторінкового додатка SPA з безсерверною інфраструктурою Firebase гарантує високу відмовостійкість та еластичне масштабування системи. Окремо варто відзначити глибоко продуману модель інформаційної безпеки: імплементація декларативних правил Firestore на основі рольової моделі RBAC та створення

клієнтського компонента-охоронця забезпечують надійний захист комерційних даних від лінійного персоналу. Високої академічної оцінки заслуговує також алгоритмічна складність роботи, зокрема успішна реалізація атомарних і компенсаційних транзакцій, які повністю усувають проблему стану гонитви під час складського обліку.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: відсутність детального опису механізмів майбутньої інтеграції системи із зовнішніми платіжними шлюзами та дрібні стилістичні неточності в оформленні графічного матеріалу.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації. Графічні матеріали наочно ілюструють архітектурні рішення.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: —

9. Оцінка дипломної роботи: добре 167/D

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Князь Ю.Ч. к.т.н., доцент, зав. кафедри кібербезпеки, ХНУ

"12" 06 2026 р. (підпис)

Зав. кафедри КІС
д-р. філософії Ользі ПАВЛОВІЙ

Максим КОБИЛЬЧУК
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи ІСТ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1.05 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Інформаційна система моніторингу стану здоров'я пацієнтів із оптимізацією планування завдань
Автор Валерій ДУДАРЧУК
Освітня програма Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 Інформаційні системи та технології
Науковий керівник: асистент Олег ВОЙЧУР

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 4.82% і адресується до 10 першоджерела; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Завідувач кафедри _____ Ольга ПАВЛОВА
Підпис Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Єлизавета ГНАТЧУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Олег ВОЙЧУР
Підпис Ім'я, ПРІЗВИЩЕ