

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Вебсистема прогнозування поломок техніки на
основі історії ремонтів»**

Виконала студентка групи ВТ-22-1
ФЕСЕНКО Вікторія Володимирівна

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
НЕРОДА Сергій Іванович

Житомир – 2026

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпе-
чення

Тетяна Вакалюк

«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ФЕСЕНКО Вікторія Володимирівна**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Вебсистема прогнозування поломок техніки на основі історії ремонтів»**,

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

Вихідні дані для роботи: Об'єктом дослідження є автоматизація обліку та аналізу ремонтних замовлень, а предметом – розробка вебсистеми прогнозування поломок техніки на основі Random Forest.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	01.03.2026 р.	01.03.2026 р.
2	Власенко О.В.	01.03.2026 р.	01.03.2026 р.
3	Власенко О.В.	01.03.2026 р.	01.03.2026 р.

РЕФЕРАТ

Кваліфікаційна робота бакалавра містить відомості про розроблену вебсистему прогнозування поломок техніки. Обсяг пояснювальної записки становить 114 сторінок, робота містить 26 рисунків та 29 таблиць. Список використаної літератури налічує 30 найменувань і розміщений на 4 сторінках.

Метою випускної кваліфікаційної роботи є розробка вебсистеми прогнозування поломок техніки для автоматизації роботи сервісного центру, що включає серверну частину з REST API, окремий сервіс машинного навчання та клієнтський вебзастосунок із рольовою моделлю доступу.

У результаті виконання кваліфікаційної роботи розроблено серверну частину на базі Node.js та Express.js з базою даних PostgreSQL, ML-сервіс на Python/Flask із трьома моделями Random Forest для бінарної класифікації наявності ризику поломки, багатокласової класифікації типу несправності та регресійного оцінювання ймовірності відмови пристрою, а також клієнтський застосунок на React/Vite/Ant Design. Реалізовано повний цикл управління замовленнями, рольову модель доступу для чотирьох категорій користувачів (адміністратор, майстер, клієнт, гість), аналітичний дашборд із сімома інтерактивними графіками, вісьмома статистичними картками та таблицею продуктивності майстрів, інтеграцію з Google Gemini для генерації текстових рекомендацій щодо обслуговування пристроїв та експорт звітності у форматах Excel і PDF. Проведено комплексне автоматизоване тестування: 37 тестів серверного API (Jest, Supertest) та 10 тестів ML-сервісу (pytest) – усі пройдено успішно.

Ключові слова: ВЕБСИСТЕМА, ПРОГНОЗУВАННЯ ПОЛОМОК, СЕРВІСНИЙ ЦЕНТР, МАШИННЕ НАВЧАННЯ, RANDOM FOREST.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема прогнозування поломок техніки на основі історії ремонтів	Літ.	Арк.	Аркушів
Розроб.	Фесенко В.В.						3	114
Керівник	Власенко О.В.					ФІКТ, Гр. ВТ-22-1		
Реценз.	Нерода С.І.							
Н. контр.								
Затверд.								

ABSTRACT

The bachelor's qualification thesis contains information about the project developed within this work. The explanatory note comprises 114 pages, includes 26 figures and 29 tables. The list of references contains 30 items and is placed on 4 pages.

The purpose of the qualification thesis is to develop a web-based equipment failure prediction system for automating repair service center operations, including a REST API backend, a dedicated machine learning service, and a client web application with a role-based access control model.

As a result of the qualification thesis, a backend built on Node.js and Express.js with a PostgreSQL database was developed, along with a Python/Flask ML service comprising three Random Forest models for binary failure risk classification, multiclass failure type classification, and device failure probability regression, as well as a client application built on React/Vite/Ant Design. A full order management lifecycle was implemented, covering four user roles (administrator, technician, client, and guest), an analytical dashboard with seven interactive charts, eight statistical cards, and a technician performance table, Google Gemini integration for generating natural language maintenance recommendations, and report export in Excel and PDF formats.

Comprehensive automated testing was conducted: 37 backend API tests (Jest, Supertest) and 10 ML service tests (pytest) – all passed successfully.

Keywords: WEB SYSTEM, FAILURE PREDICTION, SERVICE CENTER, MACHINE LEARNING, RANDOM FOREST.

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

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ПРОГНОЗУВАННЯ ПОЛОМОК ТЕХНІКИ.....	9
1.1 Постановка задачі.....	9
1.2 Аналіз наявних систем-аналогів	13
1.3 Вибір архітектури вебсистеми прогнозування поломок техніки	18
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	23
Висновки до розділу 1	27
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ ПРОГНОЗУВАННЯ ПОЛОМОК ТЕХНІКИ НА ОСНОВІ ІСТОРІЇ РЕМОНТІВ	28
2.1 Варіанти використання та функціональні вимоги системи	28
2.2 Розробка бази даних системи.....	36
2.3 Проєктування алгоритмів роботи системи та математичне забезпечення модуля прогнозування	45
2.4 Реалізація серверної частини та REST API вебсистеми.....	56
2.5 Реалізація клієнтської частини вебсистеми.....	62
Висновки до розділу 2	66
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ ПРОГНОЗУВАННЯ ПОЛОМОК ТЕХНІКИ.....	67
3.1 Порядок встановлення та налаштування параметрів системи	67
3.2 Структура інтерфейсу вебсистеми прогнозування поломок техніки	71
3.3 Тестування роботи програмного комплексу	80
Висновки до розділу 3	85
ВИСНОВКИ.....	86
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	88
ДОДАТКИ.....	92

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – програмний інтерфейс взаємодії застосунків

CORS (Cross-Origin Resource Sharing) – механізм спільного використання ресурсів між різними джерелами

CRUD (Create, Read, Update, Delete) – базові операції роботи з даними

CSP (Content Security Policy) – політика безпеки вмісту

JWT (JSON Web Token) – стандарт передачі даних автентифікації у вигляді підписаного токена

MAE (Mean Absolute Error) – середня абсолютна помилка

ML (Machine Learning) – машинне навчання

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

PDF (Portable Document Format) – формат документів для портативного відображення

RBAC (Role-Based Access Control) – розмежування доступу на основі ролей

REST (Representational State Transfer) – архітектурний стиль побудови програмних інтерфейсів

RMSE (Root Mean Square Error) – середньоквадратична помилка

SKU (Stock Keeping Unit) – артикул для обліку товарних одиниць на складі

СУБД – система управління базами даних

XSS (Cross-Site Scripting) – міжсайтовий скриптинг

R^2 – коефіцієнт детермінації моделі регресії

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

ВСТУП

Сучасні сервісні центри з ремонту побутової та цифрової техніки працюють в умовах зростання кількості звернень, розширення номенклатури пристроїв та підвищення вимог клієнтів до швидкості, прозорості й якості обслуговування. Використання паперового документообігу або розрізнених електронних таблиць для обліку ремонтних замовлень є недостатньо ефективним, оскільки ускладнює контроль статусів, розподіл роботи між майстрами, збереження історії ремонтів і формування звітності [1, 2]. Значна частина накопичених даних про клієнтів, пристрої, типові несправності та повторні звернення при цьому залишається невикористаною для аналітики.

Актуальність теми дипломної роботи зумовлена необхідністю підвищення ефективності діяльності сервісних центрів шляхом автоматизації обліку ремонтних замовлень, централізованого зберігання даних та використання методів машинного навчання для оцінювання ризику майбутніх поломок. Розробка такої вебсистеми є доцільною, оскільки вона поєднує функції управління сервісним центром із можливостями предиктивної аналітики, дає змогу аналізувати історію обслуговування пристроїв і формувати рекомендації щодо профілактичного обслуговування. Використання накопиченої ремонтної історії як джерела даних дозволяє не лише фіксувати результати виконаних робіт, а й виявляти закономірності, що можуть свідчити про підвищений ризик повторних несправностей. Це створює передумови для переходу від реактивної моделі обслуговування, коли ремонт виконується після виникнення проблеми, до проактивного підходу, орієнтованого на попередження можливих поломок.

Метою дипломної роботи є проєктування та розробка вебсистеми прогнозування поломок техніки для автоматизації роботи сервісного центру з інтегрованим модулем машинного навчання для оцінювання ризику виходу пристрою з ладу.

Для досягнення поставленої мети в роботі передбачено проаналізувати предметну область та наявні підходи до автоматизації ремонтних замовлень; ви-

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

значити функціональні й нефункціональні вимоги до системи; розробити архітектуру вебсистеми із серверною частиною, базою даних та окремим сервісом машинного навчання; спроектувати й навчити моделі для класифікації типу поломки та оцінювання ймовірності відмови пристрою; реалізувати клієнтський інтерфейс із рольовою моделлю доступу, аналітичним дашбордом і модулем формування рекомендацій, а також провести тестування основних компонентів системи.

Об'єктом дослідження є процес автоматизації обліку та аналізу ремонтних замовлень у сервісному центрі з обслуговування побутової та цифрової техніки.

Предметом дослідження є методи та засоби розробки вебсистеми прогнозування поломок техніки на основі алгоритмів машинного навчання класу Random Forest та сучасного стеку вебтехнологій.

У роботі використано методи об'єктно-орієнтованого аналізу та проектування, реляційного моделювання даних, REST-підхід до організації взаємодії між компонентами, рольове розмежування доступу, алгоритми Random Forest для задач класифікації та регресійного оцінювання, а також метод синтетичної генерації навчальних даних.

Практичне значення отриманих результатів полягає в тому, що розроблена система може бути використана як основа для автоматизації роботи сервісного центру та підтримки повного циклу управління ремонтними замовленнями. Вона дає змогу централізовано зберігати інформацію про клієнтів, пристрої, ремонти й запчастини, відстежувати статуси замовлень, формувати звітність у форматах Excel та PDF, а також оцінювати потенційний ризик поломки для зареєстрованих пристроїв. Модульна архітектура рішення забезпечує можливість подальшого масштабування та адаптації під потреби конкретного підприємства.

Структура та обсяг роботи. Випускна кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаної літератури та додатків. Повний обсяг пояснювальної записки становить 114 сторінок, робота містить 26 рисунків, 29 таблиць, список використаної літератури із 30 найменувань та додатки.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ПРОГНОЗУВАННЯ ПОЛОМОК ТЕХНІКИ

1.1 Постановка задачі

Сучасний ринок сервісних послуг з ремонту побутової та цифрової техніки характеризується зростанням кількості звернень, розширенням номенклатури пристроїв та підвищенням вимог клієнтів до швидкості, прозорості й якості обслуговування. Сервісні центри, що спеціалізуються на ремонті смартфонів, ноутбуків, планшетів та іншої електронної техніки, щоденно опрацьовують значну кількість замовлень, кожне з яких охоплює прийом пристрою, діагностику, погодження вартості робіт, підбір запчастин, виконання ремонту, контроль якості та видачу пристрою клієнту. За таких умов ручне управління процесами або використання розрізнених електронних таблиць стає недостатньо ефективним, оскільки ускладнює контроль актуального стану замовлень, розподіл навантаження між майстрами, ведення історії ремонтів і формування управлінської звітності.

Однією з основних проблем діяльності сервісного центру є фрагментарність зберігання інформації. Дані про клієнтів, пристрої, ремонтні замовлення, використані запчастини, виконані роботи та попередні звернення часто зберігаються в різних місцях або взагалі не мають єдиної структури. Це призводить до дублювання записів, втрати частини історії обслуговування, складності пошуку потрібної інформації та неможливості швидко отримати повну картину щодо конкретного клієнта або пристрою. Особливо важливою є історія ремонтів, оскільки вона дозволяє оцінювати надійність конкретного екземпляра техніки, виявляти повторювані несправності та формувати обґрунтовані рекомендації щодо подальшого обслуговування.

Аналіз діяльності типового сервісного центру дозволив визначити основні напрями, що потребують автоматизації [1, 2]. Насамперед це стосується управління замовленнями на ремонт, яке має охоплювати повний життєвий цикл заявки — від її реєстрації до видачі відремонтованого пристрою клієнту. У межах

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

такого процесу необхідно забезпечити фіксацію поточного статусу замовлення, відповідального працівника, часу кожної зміни та коментарів, що супроводжують виконання робіт. Це дає змогу уникнути ситуацій, коли клієнт не отримує актуальної інформації про стан ремонту, а працівники сервісного центру не мають єдиного джерела даних щодо виконання заявки.

Важливим напрямом автоматизації є облік клієнтів і пристроїв. Для коректної роботи сервісного центру система повинна зберігати дані про клієнтів, їхні контактні відомості, а також інформацію про зареєстровані пристрої, зокрема тип техніки, бренд, модель, серійний номер, дату придбання та історію попередніх ремонтів. Наявність таких даних дозволяє швидко ідентифікувати пристрій під час повторного звернення, аналізувати частоту виникнення несправностей, оцінювати вік техніки та використовувати накопичену інформацію для прогнозування можливих проблем у майбутньому.

Окремої автоматизації потребує управління складом запчастин. У процесі ремонту сервісний центр постійно використовує комплектуючі, кількість яких має контролюватися в актуальному стані. Система повинна забезпечувати зберігання інформації про назву запчастини, артикул, ціну, кількість на складі, мінімальний залишок і сумісність із типами пристроїв. При використанні запчастини в межах ремонтного замовлення її кількість має автоматично зменшуватися, а у випадку досягнення мінімального залишку система повинна привертати увагу адміністратора або відповідального працівника до необхідності поповнення складу. Такий підхід знижує ризик затримки ремонту через відсутність потрібних деталей.

Не менш важливим є розподіл роботи між майстрами. У реальному сервісному центрі ефективність обробки замовлень значною мірою залежить від правильного призначення виконавця з урахуванням його спеціалізації, поточного навантаження та кількості вже виконаних робіт. Тому система має підтримувати призначення майстра на замовлення, перегляд списку призначених заявок, зміну

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

статусів ремонту та фіксацію виконаних дій. Для адміністратора це створює можливість контролювати продуктивність працівників, а для майстра — отримувати впорядкований перелік власних завдань.

Для керівництва сервісного центру важливими є засоби аналітики та звітності. Система повинна надавати можливість переглядати кількість замовлень за обраний період, співвідношення активних і завершених заявок, сумарний дохід, середній час ремонту, розподіл замовлень за статусами, типами техніки, брендами та видами ремонтних робіт. Такі дані необхідні для прийняття управлінських рішень, оцінювання завантаженості майстрів, планування закупівлі запчастин і виявлення найбільш поширених типів несправностей. Наявність експорту звітності у форматах Excel та PDF додатково спрощує підготовку внутрішніх документів і передавання результатів аналізу керівництву.

Ключовою відмінністю розроблюваної системи є наявність модуля інтелектуального аналізу, призначеного для прогнозування майбутніх поломок техніки на основі накопиченої історії ремонтів. На відміну від звичайних CRM-систем, які лише зберігають інформацію про замовлення, розроблювана вебсистема має використовувати ці дані для оцінювання ризику повторної несправності. Задача прогнозування належить до сфери предиктивної аналітики та потребує застосування методів машинного навчання, оскільки між характеристиками пристрою, його віком, попередніми ремонтами та ймовірністю майбутньої поломки можуть існувати складні залежності.

Для формування прогнозу система має враховувати набір ознак, що характеризують пристрій і його історію обслуговування. До таких ознак належать тип пристрою, бренд, вік техніки, кількість попередніх ремонтів, тип останньої несправності та інтервал між ремонтами. На основі цих даних модуль машинного навчання визначає ймовірність виходу пристрою з ладу в найближчій перспективі, найбільш імовірний тип поломки, рівень ризику та рекомендації щодо профілактичного обслуговування. Це дозволяє сервісному центру не лише реагувати на вже наявні несправності, а й проактивно інформувати клієнтів про потенційні проблеми та пропонувати своєчасне технічне обслуговування.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

Додатковим елементом інтелектуального аналізу є інтеграція з великою мовною моделлю Google Gemini. Якщо модель машинного навчання забезпечує кількісну оцінку ризику та прогнозований тип несправності, то мовна модель використовується для аналізу текстової історії ремонтів і формування природно-мовних пояснень. Завдяки цьому користувач отримує не лише числовий показник і категорію ризику, а й зрозумілі рекомендації щодо подальшої експлуатації пристрою. Такий підхід підвищує практичну цінність прогнозу, оскільки результати аналізу подаються у формі, зрозумілій не лише адміністратору чи майстру, а й звичайному клієнту.

Розроблювана система передбачає роботу з кількома категоріями користувачів. Адміністратор має керувати користувачами, замовленнями, складом запчастин, аналітикою та загальними налаштуваннями процесів. Майстер повинен мати доступ до призначених йому замовлень, інформації про пристрій, історії статусів і можливості оновлення стану ремонту. Клієнт має переглядати власні пристрої, створювати або відстежувати замовлення та отримувати результати прогнозування. Для незареєстрованого користувача передбачається обмежений публічний доступ до відстеження статусу замовлення за його номером. Такий розподіл функцій потребує реалізації рольової моделі доступу, яка забезпечує захист даних і розмежування прав відповідно до ролі користувача.

З урахуванням виявлених проблем доцільним є створення комплексного вебзастосунку, що поєднує функції CRM-системи сервісного центру, засоби управління ремонтними замовленнями, облік клієнтів і пристроїв, контроль складських залишків, аналітичну звітність та модуль інтелектуального прогнозування поломок техніки. Такий програмний комплекс має забезпечити централізоване зберігання даних, прозоре відстеження стану ремонтів, підтримку прийняття управлінських рішень і підвищення якості взаємодії сервісного центру з клієнтами.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2 Аналіз наявних систем-аналогів

Проведений аналіз ринку програмного забезпечення для управління сервісними центрами дозволив виявити низку наявних рішень, що частково охоплюють функціональні потреби, описані в постановці задачі. З метою визначення стандартного функціоналу систем цього класу та обґрунтування доцільності власної розробки, розглянуто чотири найбільш поширені платформи [3].

RepairDesk – хмарна платформа для управління сервісними центрами з ремонту електроніки, що позиціонується як комплексне рішення для автоматизації бізнес-процесів. Інтерфейс системи наведено на рисунку 1.1.

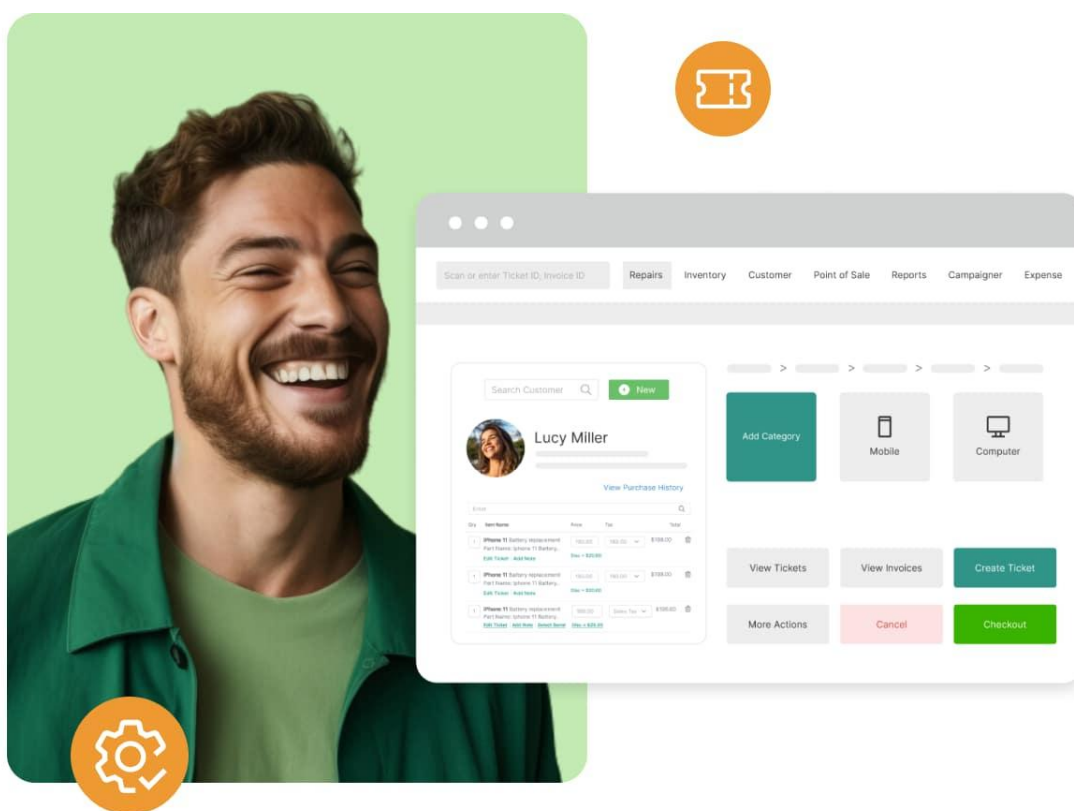


Рис. 1.1 – Інтерфейс системи RepairDesk

Платформа забезпечує управління замовленнями на ремонт із відстеженням їхніх статусів, облік клієнтів та пристроїв, контроль складських залишків запчастин, а також інтеграцію з POS-системами для прийому платежів. Крім того, RepairDesk підтримує формування базових звітів про діяльність сервісного центру та надсилання SMS- і email-сповіщень клієнтам.

До переваг RepairDesk можна віднести розвинену екосистему сторонніх інтеграцій, наявність мобільної версії та орієнтацію на комплексну автоматизацію

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

роботи сервісного центру. Водночас система не містить окремого модуля прогнозування поломок техніки, а її аналітичні можливості здебільшого обмежуються базовою звітністю. Крім того, використання RepairDesk передбачає платну підписку, що може бути суттєвим обмеженням для невеликих сервісних центрів [4].

RepairShopr – платформа для автоматизації діяльності сервісних центрів, що поєднує функції управління ремонтними замовленнями, CRM-системи та білінгового інструменту. Загальний вигляд системи зображено на рисунку 1.2.

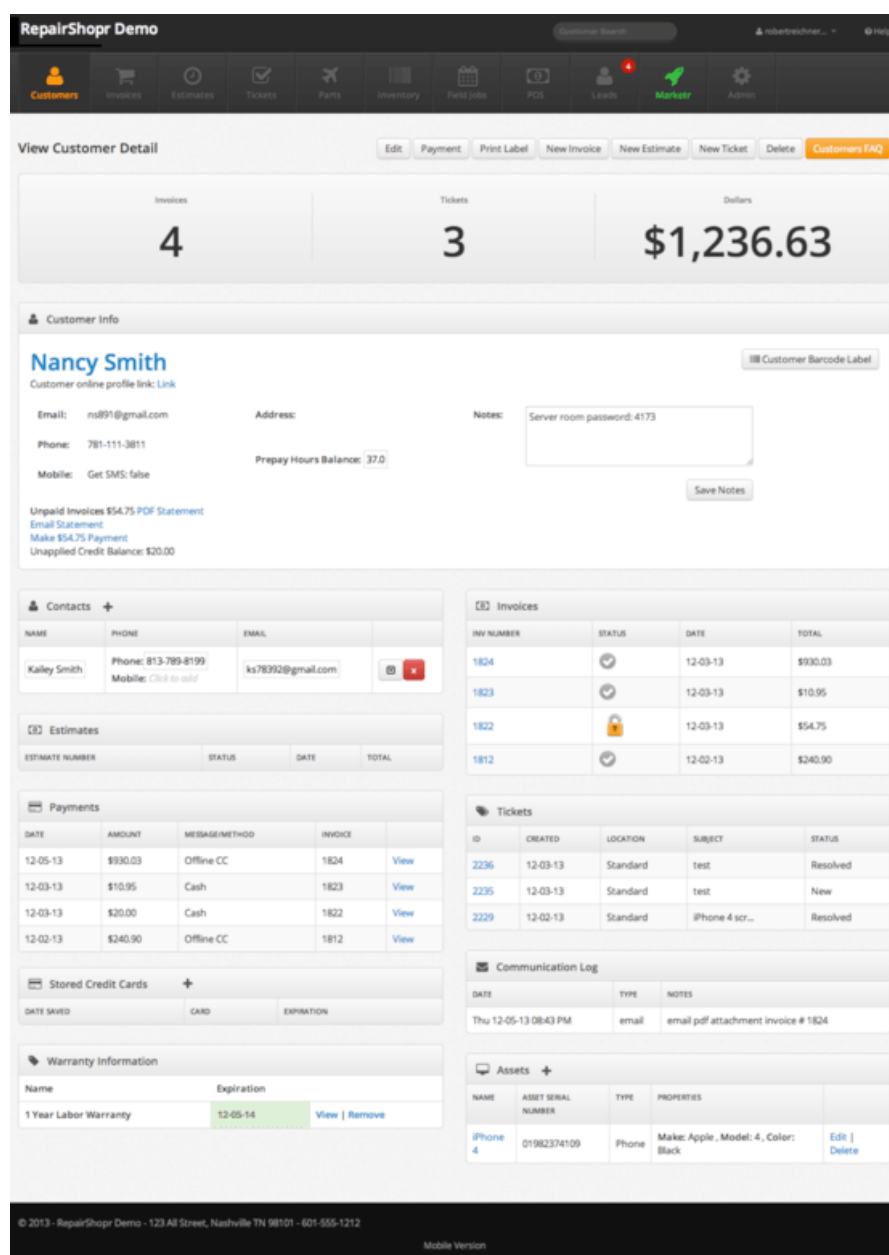


Рис. 1.2 – Інтерфейс системи RepairShopr

Платформа забезпечує прийом і відстеження ремонтних замовлень, ведення клієнтської бази з історією взаємодії, управління складом запчастин, а також формування рахунків та інтеграцію з бухгалтерськими системами. Окремою перевагою RepairShopr є наявність онлайн-порталу для клієнтів, через який вони можуть самостійно відстежувати статус виконання замовлення, що відповідає сучасним вимогам до цифрових сервісів.

Водночас RepairShopr, як і RepairDesk, орієнтована переважно на автоматизацію операційних та облікових процесів сервісного центру. Система не містить окремого модуля інтелектуального аналізу ремонтної історії та не забезпечує прогнозування ймовірних несправностей техніки на основі накопичених даних. Це обмежує її використання для задач предиктивної аналітики та проактивного обслуговування клієнтів.

HelloClient – CRM-система для сервісних центрів, адаптована до потреб ринків країн СНД та частково локалізована для вітчизняного ринку. Вигляд інтерфейсу системи наведено на рисунку 1.3.

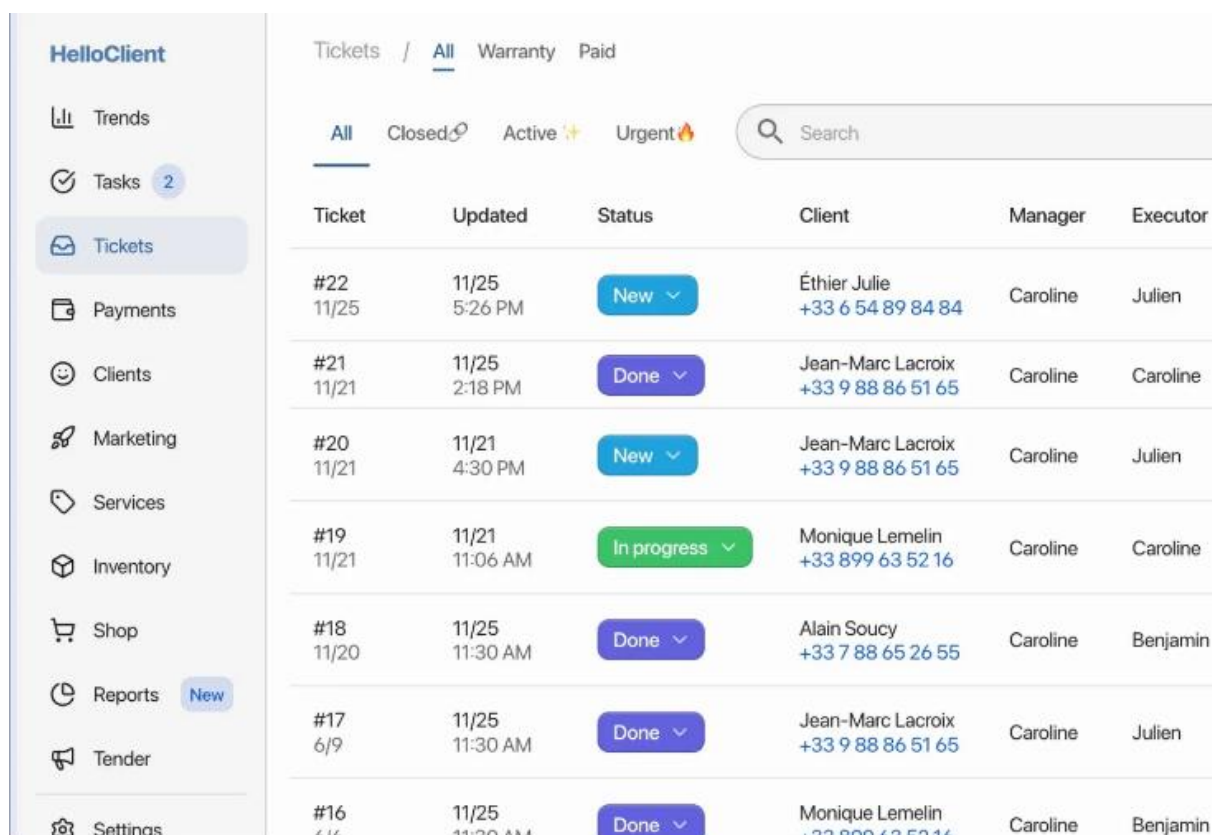


Рис. 1.3 – Інтерфейс системи HelloClient

Функціонал HelloClient охоплює облік ремонтних замовлень і клієнтів, управління запчастинами та складськими залишками, формування базової звітності щодо роботи майстрів, а також підтримку кількох філій в одному акаунті. Завдяки цьому система може використовуватися для автоматизації основних облікових процесів сервісного центру.

До переваг HelloClient можна віднести часткову локалізацію, орієнтацію на потреби локального ринку та відносно невисоку вартість впровадження. Водночас аналітичні можливості системи є обмеженими, а інтерфейс поступається сучасним вебрішенням за зручністю та візуальною організацією. Крім того, HelloClient не містить модуля прогнозування поломок або інтелектуального аналізу даних на основі штучного інтелекту, що обмежує її застосування для задач предиктивної аналітики.

Fixably – фінська SaaS-платформа для управління ремонтними процесами, орієнтована переважно на авторизовані сервісні центри Apple. Інтерфейс системи зображено на рисунку 1.4.

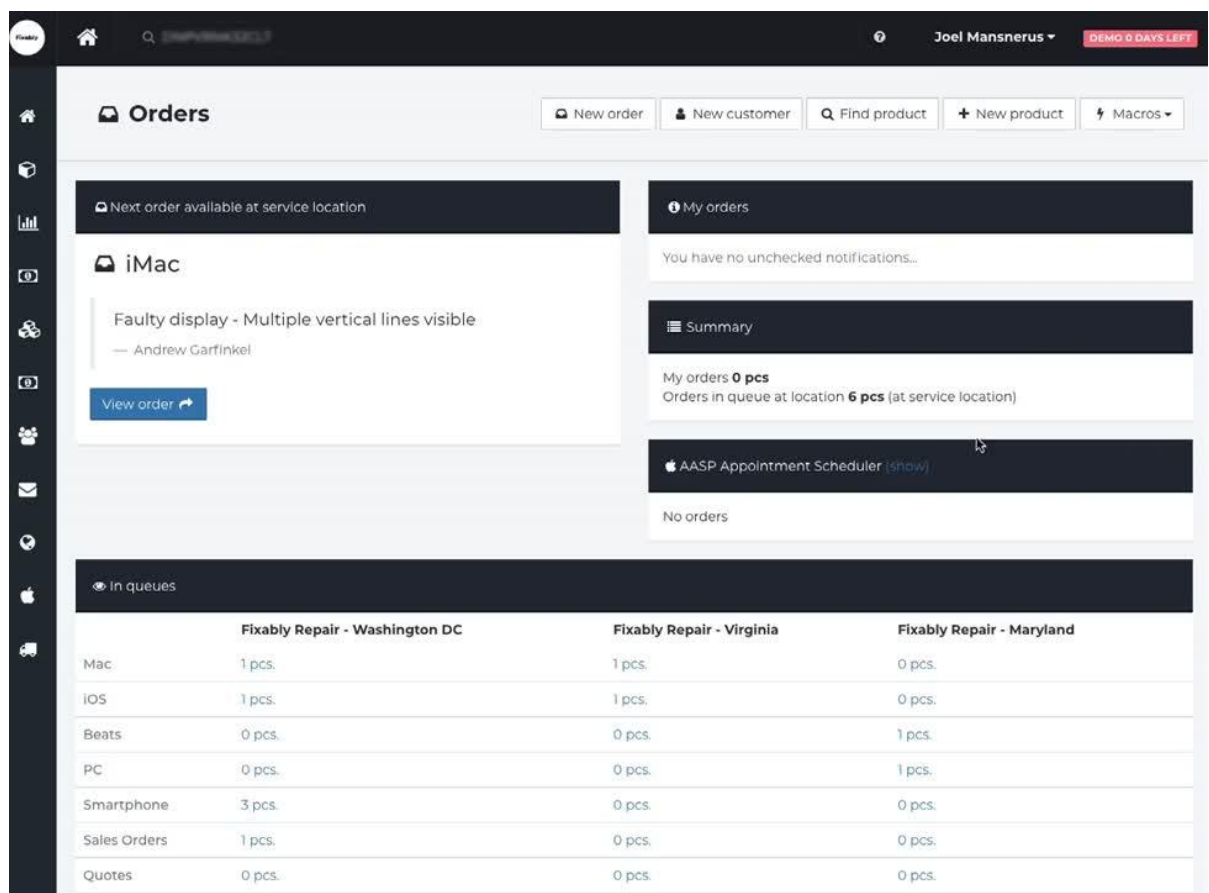


Рис. 1.4 – Інтерфейс системи Fixably

Платформа забезпечує автоматизацію ремонтного процесу відповідно до регламентів Apple, підтримує інтеграцію з GSX (Global Service Exchange), а також містить інструменти для управління гарантійними зверненнями. Крім того, Fixably передбачає використання автоматизованої діагностики на основі кодів помилок Apple та надає розширені засоби аналітики продуктивності сервісного центру.

Серед розглянутих аналогів Fixably вирізняється найбільш розвиненими аналітичними можливостями та глибокою інтеграцією з процесами обслуговування техніки Apple. Водночас орієнтація платформи на екосистему Apple обмежує її застосування для мультибрендових сервісних центрів. Крім того, система не реалізує прогнозування майбутніх поломок техніки на основі методів машинного навчання, що не дозволяє повною мірою використовувати її для задач предиктивної аналітики.

У таблиці 1.1 наведено порівняльний аналіз функціональних можливостей розглянутих аналогів та розроблюваної системи «СервісПро».

Таблиця 1.1

Порівняльний аналіз аналогів програмного продукту

Функціональність	Repair Desk	RepairShopr	HelloClient	Fixably	Сервіс Про
Управління замовленнями	+	+	+	+	+
CRM / облік клієнтів	+	+	+	+	+
Облік пристроїв з історією	+/-	+	+/-	+	+
Управління складом	+	+	+	+	+
Клієнтський портал	+	+	-	+	+
Відстеження статусу (гість)	-	+	-	-	+
Рольова модель доступу	+/-	+	+/-	+	+
Розширена аналітика	+/-	+/-	-	+	+
Прогнозування поломок (ML)	-	-	-	-	+
Інтеграція з LLM (AI-аналіз)	-	-	-	-	+
Рекомендації щодо обслуговування	-	-	-	-	+
Україномовний інтерфейс	-	-	+/-	-	+
Можливість локального розгортання без придбання комерційної ліцензії.	-	-	-	-	+

Як видно з таблиці 1.1, жодна з розглянутих платформ не реалізує прогнозування поломок на основі методів машинного навчання та не інтегрується з великими мовними моделями для інтелектуального аналізу ремонтних даних. Системи RepairDesk та RepairShopr є функціонально повними, однак є комерційними хмарними сервісами із щомісячною оплатою, що унеможлиблює їх використання без придбання комерційної підписки або ліцензії. HelloClient, єдиний частково локалізований аналог, суттєво поступається конкурентам за якістю аналітики. Fixably є найбільш технологічно розвиненою платформою, проте її застосування обмежене вузькою нішею авторизованих центрів Apple.

Серед розглянутих аналогів система «СервісПро» вирізняється поєднанням функцій CRM, модуля прогнозування на основі машинного навчання та інтеграції з Google Gemini, забезпечує повну україномовну локалізацію та може бути розгорнута без придбання комерційної ERP-системи.

1.3 Вибір архітектури вебсистеми прогнозування поломок техніки

Проектування сучасного вебзастосунку вимагає обґрунтованого вибору архітектурного патерну, що визначає спосіб організації, взаємодії та масштабування компонентів системи. Архітектурне рішення безпосередньо впливає на продуктивність, простоту супроводу, надійність і вартість розгортання програмного комплексу. Нижче розглянуто три основні архітектурні підходи, що застосовуються при розробці вебсистем.

Монолітна архітектура передбачає реалізацію всієї функціональності у вигляді єдиного застосунку, що збирається та розгортається як одне ціле. Усі компоненти – рівень представлення, бізнес-логіка та доступ до даних – тісно пов'язані між собою та виконуються в одному процесі. Такий підхід є найпростішим з точки зору первинної розробки та налагодження, оскільки не потребує налаштування міжсервісної комунікації. Проте зі збільшенням обсягу кодової бази монолітна архітектура суттєво ускладнює незалежне оновлення окремих модулів, унеможлиблює вибіркове масштабування компонентів і підвищує ризик

каскадних відмов – ситуацій, коли збій в одному модулі призводить до відмови всієї системи.

Мікросервісна архітектура передбачає декомпозицію системи на набір дрібних, слабо пов'язаних між собою сервісів, кожен з яких відповідає за окрему бізнес-функцію та може бути розгорнутий, оновлений і масштабований незалежно. Перевагою цього підходу є висока технологічна гнучкість – різні сервіси можуть бути реалізовані різними мовами програмування та розгортатися на окремих серверах. Водночас мікросервісна архітектура характеризується значною складністю оркестрації, потребує розвиненої DevOps-інфраструктури (контейнеризація, сервісний реєстр, балансування навантаження), що робить її надлишковою для систем невеликого та середнього масштабу.

Клієнт–серверна архітектура з виділеними рівнями є компромісним підходом, при якому система структурується на три логічні рівні: клієнтський (рівень представлення), серверний (рівень бізнес-логіки) та рівень даних. При необхідності окремі обчислювально інтенсивні задачі виносяться у вигляді спеціалізованих підсервісів, не порушуючи загальної цілісності системи. Такий підхід забезпечує чіткий розподіл відповідальності, технологічну гнучкість та керований рівень складності.

У таблиці 1.2 наведено порівняльний аналіз розглянутих архітектурних підходів за ключовими критеріями.

Таблиця 1.2

Порівняльний аналіз архітектурних підходів

Критерій	Монолітна	Мікросервісна	Клієнт-серверна (багаторівнева)
Простота розробки	+	-	+
Простота розгортання	+	-	+
Незалежне масштабування	-	+	+/-
Технологічна гнучкість	-	+	+
Ізоляція відмов	-	+	+/-
DevOps-інфраструктура	Не потрібна	Обов'язкова	Мінімальна
Доцільність для MVP	+	-	+

Як видно з таблиці 1.2, для систем рівня MVP (Minimum Viable Product) та систем середнього масштабу найбільш обґрунтованим є використання клієнт-

серверної багаторівневої архітектури з можливістю виділення окремих спеціалізованих підсервісів.

Для розроблюваної вебсистеми обрано модульну клієнт-серверну архітектуру з виділеним сервісом машинного навчання. Таке архітектурне рішення зумовлене необхідністю розділення відповідальності між основними компонентами системи, забезпечення технологічної гнучкості, можливості незалежного масштабування та спрощення процесу розгортання.

Клієнтська частина системи реалізована як SPA-застосунок і відповідає за представлення даних, побудову інтерфейсу та взаємодію з користувачем. Основний сервер забезпечує реалізацію бізнес-логіки, автентифікацію, авторизацію, обробку запитів і взаємодію з базою даних. Окремий ML-сервіс відповідає за виконання прогностичних моделей, що використовуються для оцінювання ризику поломки пристрою. Такий розподіл дозволяє зменшити взаємний вплив компонентів під час оновлення, тестування або можливої відмови окремих частин системи.

Технологічна гнучкість архітектури полягає в можливості використання різних мов програмування та фреймворків для окремих компонентів. Сервіс машинного навчання реалізовано мовою Python, яка має розвинену екосистему бібліотек для аналізу даних і побудови моделей машинного навчання, зокрема scikit-learn, pandas і numpy. Основний backend реалізовано на Node.js/Express.js, що забезпечує ефективну обробку HTTP-запитів і зручну побудову REST API [5, 6, 18].

Виділення ML-сервісу в окремий компонент також створює можливість його незалежного масштабування. У разі зростання навантаження на модуль прогнозування його можна масштабувати окремо від основного сервера, не впливаючи на доступність інших функцій системи. Водночас запропоноване рішення не є надмірно складним, оскільки система складається з обмеженої кількості основних компонентів і може бути розгорнута на одному хості за допомогою Docker Compose без використання складної оркестраційної інфраструктури [27].

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

Узагальнена архітектура системи включає клієнтський вебзастосунок React SPA, який забезпечує інтерфейс користувача та взаємодіє з API через HTTP/REST; основний сервер Express.js API Server, що реалізує бізнес-логіку, JWT-автентифікацію, RBAC-авторизацію та маршрутизацію запитів; сервіс машинного навчання Flask ML Service, призначений для виконання прогнозування поломок на основі навчених моделей; реляційну базу даних PostgreSQL, у якій зберігаються основні дані системи; а також зовнішній API Google Gemini, що використовується для генерації природномовних рекомендацій та аналізу ремонтної історії.

На рисунку 1.5 представлено узагальнену структурну схему вебсистеми із зазначенням основних компонентів, їхнього технологічного стеку та протоколів взаємодії.

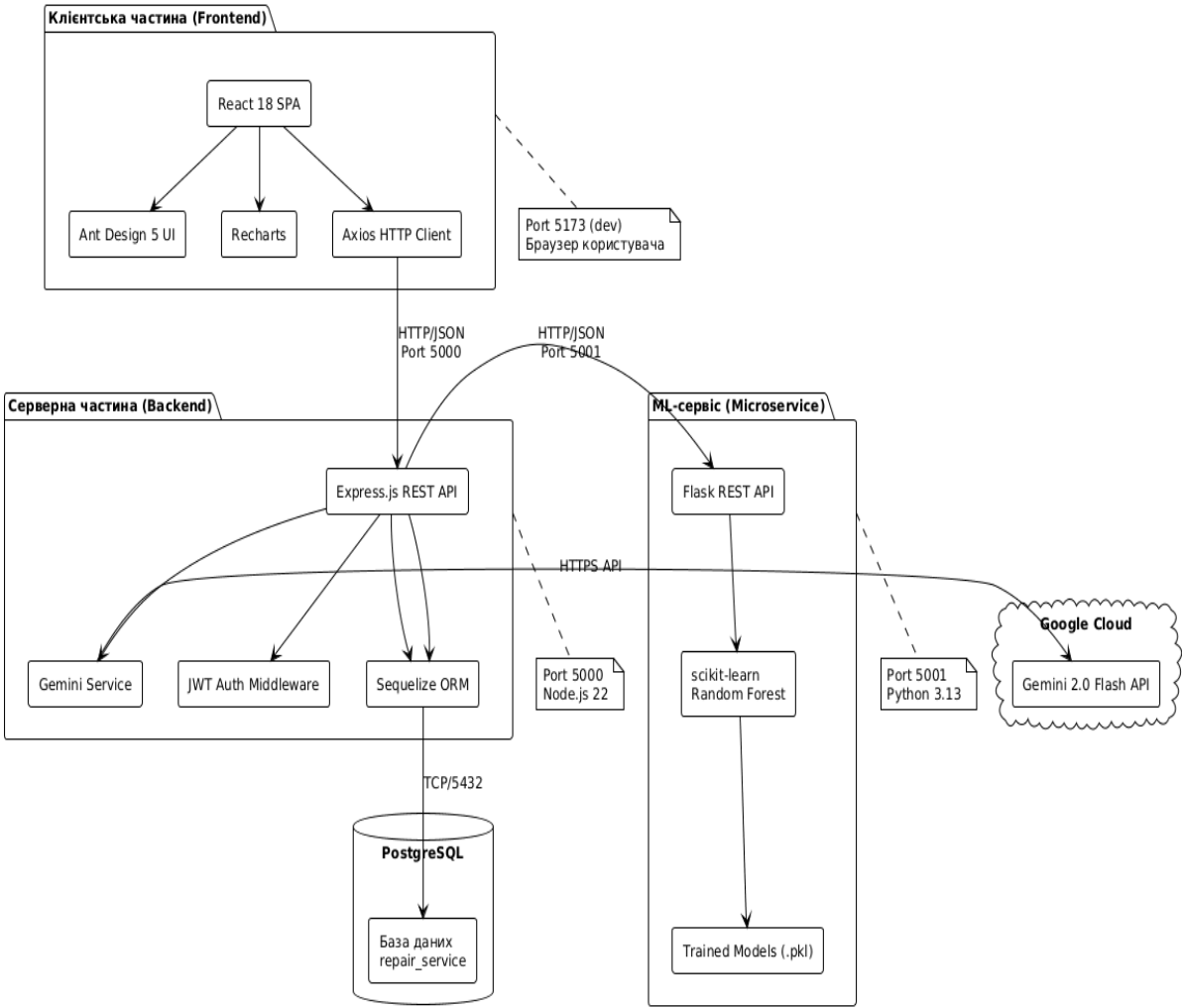


Рис. 1.5 – Узагальнена архітектура вебсистеми

Взаємодія компонентів системи побудована за патерном REST (Representational State Transfer) [21]. Клієнтський застосунок надсилає HTTP-запити до Express.js API, який виступає єдиною точкою входу (API Gateway). При необхідності виконання прогнозу Express-сервер формує запити до Flask ML-сервісу та/або Google Gemini API, агрегує результати та повертає їх клієнту.

Автентифікація реалізована за допомогою JWT (JSON Web Token) з двома типами токенів: access token (час життя – 1 година) для авторизації запитів та refresh token (час життя – 7 днів) для оновлення access token без повторного введення облікових даних [22].

Авторизація побудована на рольовій моделі RBAC (Role-Based Access Control) із чотирма ролями: адміністратор, майстер, клієнт та гість. Кожна роль має чітко визначений набір дозволених операцій, що контролюється middleware-компонентом на рівні API-маршрутів.

Розподіл прав доступу між ролями системи деталізовано у таблиці 1.3.

Таблиця 1.3

Розподіл прав доступу за ролями системи

Операція	Гість	Клієнт	Майстер	Адміністратор
Перегляд статусу замовлення	+	+	+	+
Реєстрація та управління профілем	-	+	+	+
Перегляд списку замовлень	-	Свої	Призначені	Усі
Створення заявки на ремонт	-	+	-	+
Редагування службових даних замовлення	-	-	+	+
Зміна статусу замовлення	-	-	Призначені	+
Управління складом запчастин	-	-	+	+
Призначення майстра на замовлення	-	-	-	+
Перегляд прогнозів поломок	-	Свої	+	+
Перегляд аналітики та звітів	-	-	-	+
Управління обліковими записами	-	-	-	+

Запити від неавтентифікованих користувачів (гостей) обробляються лише в межах публічних маршрутів – перегляду статусу замовлення за унікальним ідентифікатором. Усі захищені маршрути перевіряють наявність та коректність access token перед виконанням будь-якої операції.

Обрана модульна клієнт-серверна архітектура з виділеним сервісом машинного навчання відповідає вимогам до розроблюваної системи, оскільки забезпечує чіткий розподіл відповідальності між компонентами, технологічну гнучкість під час реалізації ML-модуля засобами Python та прийнятний рівень складності розгортання. Реалізація JWT-автентифікації у поєднанні з RBAC-авторизацією забезпечує необхідний рівень захисту даних та диференційований доступ до функціоналу системи відповідно до ролі користувача. Детальний вибір конкретних технологій, бібліотек та фреймворків для реалізації кожного компонента описано у підрозділі 1.4.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір інструментальних засобів реалізації визначається архітектурними рішеннями, прийнятими в підрозділі 1.3, та функціональними вимогами до системи. Нижче наводиться обґрунтування вибору технологій для кожного компонента розроблюваного програмного комплексу.

Node.js – середовище виконання JavaScript на стороні сервера, побудоване на рушії V8 від Google. Ключовою перевагою платформи є асинхронна подієво-орієнтована модель обробки запитів (Event Loop), що забезпечує неблокуючі I/O-операції та ефективну обробку великої кількості одночасних підключень за мінімального споживання ресурсів. Додатковою перевагою є використання єдиної мови програмування JavaScript – для клієнтської та серверної частин, що спрощує супровід кодової бази [5].

Express.js (версія 4.21) – мінімалістичний вебфреймворк для Node.js, що забезпечує гнучку систему middleware для обробки запитів, маршрутизацію з параметрами та легку інтеграцію із СУБД PostgreSQL. Фреймворк обрано завдяки його стабільності, широкій поширеності в індустрії та мінімальним накладним витратам порівняно з більш об'ємними альтернативами (NestJS, Fastify) [6].

Sequelize (версія 6.37) – ORM (Object-Relational Mapping) для Node.js з підтримкою PostgreSQL, MySQL та інших СУБД. Використання ORM дозволяє

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

описувати структуру бази даних у вигляді JavaScript-моделей, виконувати версіоновані міграції схеми та будувати запити через об'єктний API без написання сирого SQL-коду [17].

Для реалізації безпеки застосовано такі бібліотеки:

jsonwebtoken – генерація та верифікація JWT-токенів згідно зі стандартом RFC 7519 для забезпечення безстанової (stateless) автентифікації [22];

bcryptjs – хешування паролів за алгоритмом bcrypt із сіллю, що захищає облікові дані від атак перебором [7].

PostgreSQL (версія 15+) – об'єктно-реляційна СУБД з відкритим вихідним кодом, що відповідає стандарту SQL та забезпечує ACID-сумісність транзакцій. Порівняно з альтернативою MySQL, PostgreSQL краще підтримує складні аналітичні запити з віконними функціями та агрегаціями, необхідними для модуля звітності, а також нативно підтримує тип даних JSONB для зберігання неструктурованих результатів прогнозування [25].

React (версія 18.3) – JavaScript-бібліотека для побудови інтерактивних користувацьких інтерфейсів, розроблена компанією Meta. Компонентний підхід до побудови UI із можливістю повторного використання компонентів, механізм Virtual DOM для оптимізації рендерингу та розвинена екосистема доповнень визначили вибір React як основи клієнтської частини [9].

Vite (версія 5.4) – інструмент збірки нового покоління, що використовує ES Modules для миттєвого запуску середовища розробки та Rollup для оптимізованої збірки продуктивної версії. Порівняно зі стандартним Create React App, Vite забезпечує на порядок швидший запуск завдяки відсутності попередньої компіляції в режимі розробки.

Ant Design (версія 5.21) – бібліотека UI-компонентів корпоративного рівня для React, що надає понад 60 готових компонентів із підтримкою інтернаціоналізації, зокрема української локалізації. Наявність комплексних компонентів (таблиці з пагінацією та сортуванням, форми з валідацією, модальні вікна, системи навігації) суттєво прискорює розробку інтерфейсу адміністративної частини.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

Recharts (версія 2.12) – бібліотека для побудови інтерактивних графіків на базі D3.js, інтегрована у вигляді React-компонентів. Застосовується для візуалізації аналітичних даних на дашборді: динаміки замовлень, розподілу поломок за типами техніки та графіків продуктивності майстрів [10].

React Router (версія 6.26) – стандартна бібліотека маршрутизації для React-додатків, що забезпечує декларативну навігацію між сторінками SPA-застосунку без перезавантаження браузера.

Python версії 3.10+ використано для реалізації мікросервісу машинного навчання, оскільки ця мова програмування є одним із найпоширеніших інструментів у сфері аналізу даних та побудови ML-моделей. Її перевагою є розвинена екосистема спеціалізованих бібліотек, що забезпечують обробку даних, навчання моделей і виконання прогнозування. Виокремлення ML-функціоналу в окремий Python-сервіс дозволяє використовувати можливості цієї платформи незалежно від основного backend-компонента системи [11].

Для побудови REST API мікросервісу використано Flask – легкий мікрофреймворк для Python, який дає змогу швидко реалізувати маршрути для обробки запитів прогнозування. На відміну від більш комплексних фреймворків, Flask не накладає жорстких структурних обмежень на організацію коду, що є доцільним для вузькоспеціалізованого сервісу з обмеженою кількістю API-маршрутів [12].

Для реалізації моделей машинного навчання застосовано бібліотеку scikit-learn версії 1.5.2, яка містить готові інструменти для розв’язання задач класифікації, регресії та кластеризації [30]. У межах задачі прогнозування поломок техніки обрано алгоритм Random Forest, оскільки він добре працює з табличними структурованими даними, характеризується стійкістю до перенавчання завдяки ансамблевому підходу, дає змогу аналізувати важливість ознак і не потребує обов’язкової нормалізації вхідних даних [19].

Google Gemini API, зокрема модель Gemini 2.5 Flash, використовується для аналізу текстової історії ремонтів пристрою та генерації розгорнутих природно-мовних рекомендацій українською мовою [8]. Вибір цієї моделі зумовлений

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

поєднанням відносно низького часу відповіді та достатньої якості генерації тексту для прикладних задач аналізу ремонтної історії.

У таблиці 1.4 узагальнено обраний технологічний стек за компонентами системи.

Таблиця 1.4

Технологічний стек розроблюваної системи

Компонент	Технологія	Версія	Призначення
Серверна частина	Node.js + Express.js	20 / 4.21	REST API, бізнес-логіка
ORM	Sequelize	6.37	Робота з БД
СУБД	PostgreSQL	15+	Зберігання даних
Клієнтська частина	React + Vite	18.3 / 5.4	SPA-інтерфейс
UI-бібліотека	Ant Design	5.21	Компоненти інтерфейсу
Графіки	Recharts	2.12	Аналітична візуалізація
ML-мікросервіс	Python + Flask	3.10+ / 3.0.3	Прогнозування поломок
ML-алгоритм	scikit-learn (Random Forest)	1.5.2	Класифікація та регресійне оцінювання ризику
Генеративний ШІ	Google Gemini API	2.5 Flash	Текстові рекомендації
Автентифікація	jsonwebtoken + bcryptjs	—	Безпека

Мінімальні та рекомендовані вимоги до апаратного забезпечення серверного вузла визначені з урахуванням одночасної роботи чотирьох компонентів: Express.js API, Flask ML-сервісу, PostgreSQL та вебсервера статичних файлів [13]. Відповідні характеристики наведено у таблиці 1.5.

Таблиця 1.5

Вимоги до апаратного забезпечення серверного вузла

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2 ядра, 2.0 ГГц	4 ядра, 2.5 ГГц
Оперативна пам'ять	4 ГБ	8 ГБ
Дисковий простір	20 ГБ SSD	50 ГБ SSD
Операційна система	Ubuntu 22.04 / Windows 10	Ubuntu 22.04 LTS
Мережевий інтерфейс	10 Мбіт/с	100 Мбіт/с

Вимоги до попередньо встановленого програмного забезпечення серверного вузла наведено у таблиці 1.6.

Вимоги до програмного забезпечення серверного вузла

Компонент	Мінімальна версія
Node.js	18.0
Python	3.10+
PostgreSQL	15.0
npm	9.0
pip	23.0

Для кінцевих користувачів системи не висувається жодних вимог до встановленого програмного забезпечення, окрім сучасного веббраузера з підтримкою стандарту ES2020+ (Chrome 90+, Firefox 90+, Safari 15+) та стабільного інтернет-з'єднання. Адаптивний дизайн на базі компонентів Ant Design забезпечує коректне відображення інтерфейсу як на настільних комп'ютерах, так і на мобільних пристроях.

Висновки до розділу 1

У першому розділі проведено аналіз предметної області, визначено шість основних напрямів автоматизації діяльності сервісного центру та здійснено постановку задачі розробки вебсистеми прогнозування поломок техніки; порівняльний аналіз чотирьох наявних аналогів показав, що жодне з розглянутих рішень не забезпечує прогнозування несправностей засобами машинного навчання та інтеграцію з генеративними мовними моделями, що підтверджує актуальність розробки.

Обґрунтовано вибір гібридної клієнт-серверної архітектури з виділеним мікросервісом машинного навчання та визначено технологічний стек: Node.js з Express.js – для серверної частини, React з Ant Design – для клієнтської, PostgreSQL – як СУБД, Python з scikit-learn (Random Forest) – для модуля прогнозування, Google Gemini API – для генерації природномовних рекомендацій; встановлено вимоги до апаратного та програмного забезпечення серверного вузла й клієнтських пристроїв.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ ПРОГНОЗУВАННЯ ПОЛОМОК ТЕХНІКИ НА ОСНОВІ ІСТОРІЇ РЕМОНТІВ

2.1 Варіанти використання та функціональні вимоги системи

Розроблювана вебсистема передбачає чотири ролі користувачів, кожна з яких має чітко визначений набір функціональних можливостей. Рольова модель побудована за принципом RBAC (Role-Based Access Control), що реалізує принцип найменших привілеїв – кожна роль отримує лише ті права, що необхідні для виконання її функцій [14].

Роль адміністратора передбачає повний доступ до всіх функцій системи та є найбільш привілейованою. Роль майстра надає доступ до замовлень, призначених конкретному спеціалісту, а також до інструментів прогнозування для пристроїв, що перебувають у роботі. Роль клієнта забезпечує доступ до власних замовлень, зареєстрованих пристроїв та AI-прогнозів щодо їхнього стану. Роль гостя призначена для неавтентифікованих користувачів і надає доступ виключно до відстеження статусу замовлення за його номером.

У таблиці 2.1 наведено повний перелік функціональних вимог до системи із розподілом за ролями користувачів та унікальними ідентифікаторами для забезпечення трасованості вимог.

Таблиця 2.1

Функціональні вимоги до вебсистеми

Код	Роль	Функціональна вимога
FR-01	Адміністратор	Перегляд аналітичного дашборду з ключовими показниками, графіками динаміки замовлень та розподілу за типами техніки, статусами і брендами
FR-02	Адміністратор	Управління замовленнями на ремонт: створення, редагування, зміна статусу, призначення відповідального майстра
FR-03	Адміністратор	Управління обліковими записами користувачів: створення, редагування, активація та деактивація
FR-04	Адміністратор	Управління складом запчастин: додавання, редагування, видалення позицій, контроль мінімальних залишків

FR-05	Адміністратор	Запуск комбінованого AI-прогнозування поломок для будь-якого пристрою в системі
FR-06	Адміністратор	Експорт звітів та вибірок даних у форматах Excel (.xlsx) та PDF
FR-07	Адміністратор	Перегляд статистики продуктивності майстрів за кількістю виконаних та активних замовлень
FR-08	Майстер	Перегляд списку призначених замовлень із фільтрацією за статусом
FR-09	Майстер	Зміна статусу замовлення з можливістю додавання коментаря
FR-10	Майстер	Введення діагнозу та додавання використаних запчастин до замовлення
FR-11	Майстер	Запуск AI-прогнозування для пристроїв, що перебувають у роботі
FR-12	Майстер	Перегляд хронології зміни статусів замовлення
FR-13	Клієнт	Перегляд своїх замовлень та їхніх поточних статусів
FR-14	Клієнт	Реєстрація нових пристроїв із зазначенням типу, бренду, моделі, серійного номера та дати придбання
FR-15	Клієнт	Створення нових заявок на ремонт для зареєстрованих пристроїв
FR-16	Клієнт	Перегляд детальної інформації про пристрій з повною історією ремонтів
FR-17	Клієнт	Перегляд AI-прогнозу стану пристрою з оцінкою ризику та рекомендаціями
FR-18	Гість	Відстеження статусу замовлення за його номером без автентифікації
FR-19	Гість	Перегляд хронології зміни статусів замовлення із зазначенням дат та коментарів майстра

На основі аналізу предметної області та потреб цільових груп користувачів визначено нефункціональні вимоги до вебсистеми, що охоплюють аспекти продуктивності, безпеки, масштабованості та зручності використання. Конкретні кількісні показники вимог наведено у таблиці 2.2.

Нефункціональні вимоги до вебсистеми

Категорія	Код	Вимога
Продуктивність	NFR-01	Час відповіді API на запити CRUD-операцій не повинен перевищувати 500 мс
Продуктивність	NFR-02	Час виконання ML-прогнозу — не більше 2 секунд
Продуктивність	NFR-03	Час отримання відповіді від Gemini API — не більше 10 секунд
Безпека	NFR-04	Паролі зберігаються виключно у хешованому вигляді з використанням алгоритму bcrypt із сіллю
Безпека	NFR-05	Автентифікація реалізована через пару JWT-токенів (access + refresh)
Безпека	NFR-06	Перевірка авторизації виконується на кожному захищеному API-маршруті до передачі запиту обробнику
Безпека	NFR-07	Забезпечено захист від XSS-атак засобами HTTP-заголовків і валідації вхідних даних
Масштабованість	NFR-08	Система підтримує одночасну роботу до 50 користувачів без деградації продуктивності
Сприйняття	NFR-09	Інтерфейс є інтуїтивно зрозумілим для користувачів без спеціальної технічної підготовки
Сприйняття	NFR-10	Інтерфейс повністю локалізовано українською мовою з використанням uk_UA-локалі Ant Design
Сприйняття	NFR-11	Забезпечено адаптивний дизайн для коректного відображення на екранах різних розмірів

Варіанти використання описують типові сценарії взаємодії акторів із системою та є основою для проектування API-маршрутів і компонентів інтерфейсу. Діаграму варіантів використання наведено на рисунку 2.1 [15]. Вона відображає основні ролі користувачів системи та функції, доступні кожній з них відповідно до визначених прав доступу. Такий підхід дозволяє на ранньому етапі проектування уточнити межі відповідальності акторів і забезпечити узгодженість між функціональними вимогами, структурою інтерфейсу та серверною логікою.

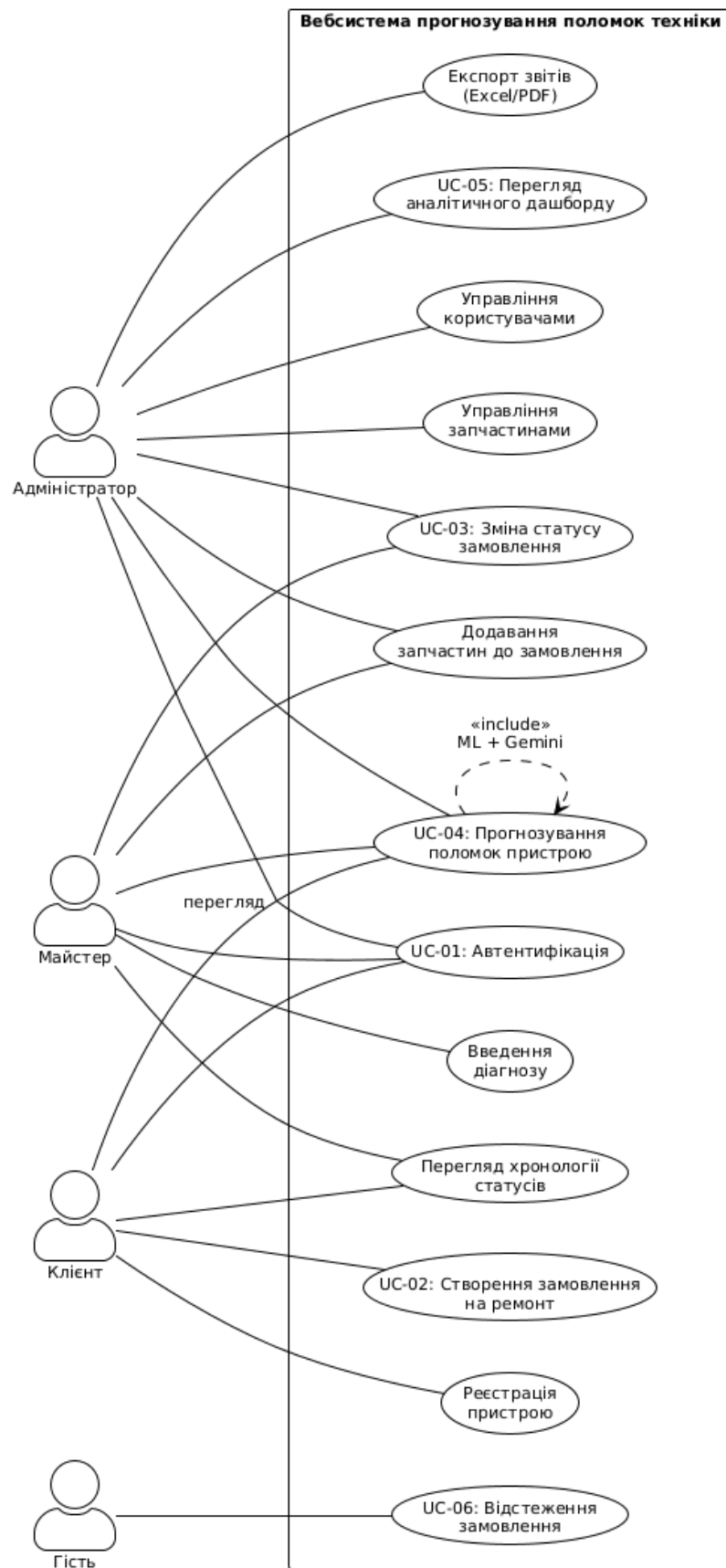


Рис. 2.1 – Діаграма варіантів використання вебсистеми

Нижче наведено опис ключових варіантів використання

Змн.	Арк.	№ докум.	Підпис	Дата

UC-01 «Автентифікація користувача» передбачає участь таких акторів, як клієнт, майстер та адміністратор. Передумовою виконання цього варіанта використання є наявність зареєстрованого облікового запису в системі. Основний сценарій полягає в тому, що користувач вводить email та пароль, після чого система перевіряє облікові дані, генерує пару JWT-токенів – access і refresh – та перенаправляє користувача на головну сторінку відповідно до його ролі. У разі введення неправильних облікових даних система відображає повідомлення про помилку автентифікації без деталізації її причини.

UC-02 «Створення замовлення на ремонт» виконується клієнтом за умови, що він автентифікований у системі та має зареєстрований пристрій. У межах основного сценарію клієнт обирає пристрій зі свого списку, вводить опис проблеми та надсилає заявку на ремонт. Після цього система генерує унікальний номер замовлення у форматі SR{YYMM}-{NNNN}, створює запис зі статусом «Нова» та додає початковий запис до хронології зміни статусів.

UC-03 «Зміна статусу замовлення» передбачає участь адміністратора або майстра. Передумовою є наявність відповідного замовлення в системі. Користувач обирає новий статус замовлення та, за потреби, додає коментар. Система оновлює статус, створює запис у хронології із зазначенням часу та автора змін, а у випадку встановлення статусу «Готово» додатково фіксує дату завершення ремонту та формує сповіщення для клієнта.

UC-04 «Прогнозування поломок пристрою» може ініціюватися адміністратором або майстром, тоді як клієнт має можливість переглядати отримані результати. Передумовою виконання цього сценарію є наявність збереженої історії ремонтів пристрою. Спочатку система збирає характеристики пристрою, зокрема його тип, бренд, вік, кількість попередніх ремонтів, тип останнього ремонту та час, що минув від останнього звернення. Після цього сформовані дані передаються до Flask ML-сервісу, де модель Random Forest виконує прогнозування та повертає бінарну класифікацію, тип імовірної поломки, числову ймовірність відмови та рівень ризику.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

Паралельно повна текстова історія ремонтів передається до Google Gemini API, який формує природномовний аналіз, перелік можливих несправностей і рекомендації щодо технічного обслуговування [8]. На завершальному етапі система агрегує результати ML-моделі та мовної моделі, зберігає прогноз у базі даних і повертає користувачу об'єднану відповідь.

UC-05 «Перегляд аналітичного дашборду» виконується адміністратором. У межах цього сценарію система здійснює серію аналітичних запитів до бази даних і формує набір ключових показників діяльності сервісного центру. До них належать динаміка замовлень за обраний період, розподіл замовлень за типами техніки, статусами та брендами, найпоширеніші типи ремонтів, середня вартість ремонту, а також показники продуктивності майстрів. UC-06 «Відстеження замовлення гостем» передбачає участь незареєстрованого користувача, який має номер замовлення. Гість вводить номер замовлення у відповідну форму, після чого система виконує пошук за публічним маршрутом і повертає обмежений набір даних. Користувачу відображаються поточний статус замовлення, тип пристрою, дата створення заявки та хронологія зміни статусів із коментарями майстра.

Описані варіанти використання охоплюють основні сценарії взаємодії користувачів із системою та відображають функціональні потреби кожної ролі. На їх основі визначено набір серверних модулів, які мають забезпечувати автентифікацію, управління замовленнями, роботу з пристроями, формування прогнозів, аналітику та сповіщення. Таким чином, специфікація варіантів використання є основою для подальшого проєктування структури серверної частини та API веб-системи.

Серверна частина системи організована за шаровою архітектурою (Layered Architecture), що передбачає розділення відповідальності між окремими рівнями та однонаправлений характер залежностей. Такий підхід дає змогу структурувати код, спростити його супровід і запобігти безпосередньому зверненню нижніх шарів до верхніх. Архітектура серверної частини охоплює рівні маршрутизації, middleware, контролерів, моделей та сервісів.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

Рівень маршрутизації (Routes) відповідає за визначення HTTP-ендпоінтів API та їх зв'язування з відповідними контролерами. Для кожної групи ресурсів передбачено окремий файл маршрутів, зокрема auth.js, users.js, devices.js, orders.js, parts.js, analytics.js, predictions.js і notifications.js. Такий підхід забезпечує логічне групування API-маршрутів і полегшує подальше розширення функціональності системи.

Рівень middleware містить проміжні компоненти, які обробляють запити до їх передавання контролерам. До цього рівня належать механізми перевірки JWT access token, контролю відповідності ролі користувача вимогам конкретного маршруту, а також уніфікованої обробки помилок із формуванням стандартизованих відповідей. Завдяки цьому повторювана логіка автентифікації, авторизації та обробки виняткових ситуацій винесена за межі контролерів.

Рівень контролерів (Controllers) реалізує логіку обробки HTTP-запитів і взаємодії з відповідними доменними сутностями. Кожен контролер відповідає за окрему частину предметної області, зокрема автентифікацію, користувачів, пристрої, ремонтні замовлення, запчастини, аналітику, прогнозування та сповіщення. Така організація дозволяє зберігати чітку структуру серверної частини та розділяти логіку обробки різних типів запитів.

Рівень моделей (Models) описує структуру даних і зв'язки між сутностями за допомогою Sequelize ORM. У межах системи визначено моделі User, Device, RepairOrder, OrderStatusHistory, Part, OrderPart, RepairType, OrderRepair, Prediction і Notification, які відображають основні об'єкти предметної області та забезпечують взаємодію серверної частини з реляційною базою даних.

Рівень сервісів (Services) інкапсулює логіку взаємодії із зовнішніми системами та допоміжними компонентами. Зокрема, GeminiService відповідає за формування промπτу на основі ремонтної історії пристрою, виклик Google Gemini API та структуровану обробку отриманої відповіді. Винесення цієї логіки в окремий сервіс дозволяє не перевантажувати контролери та спрощує підтримку інтеграції із зовнішнім API. Діаграму компонентів серверної частини, що ілюструє

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

шарову архітектуру та напрямки залежностей між рівнями, представлено на рисунку 2.2.

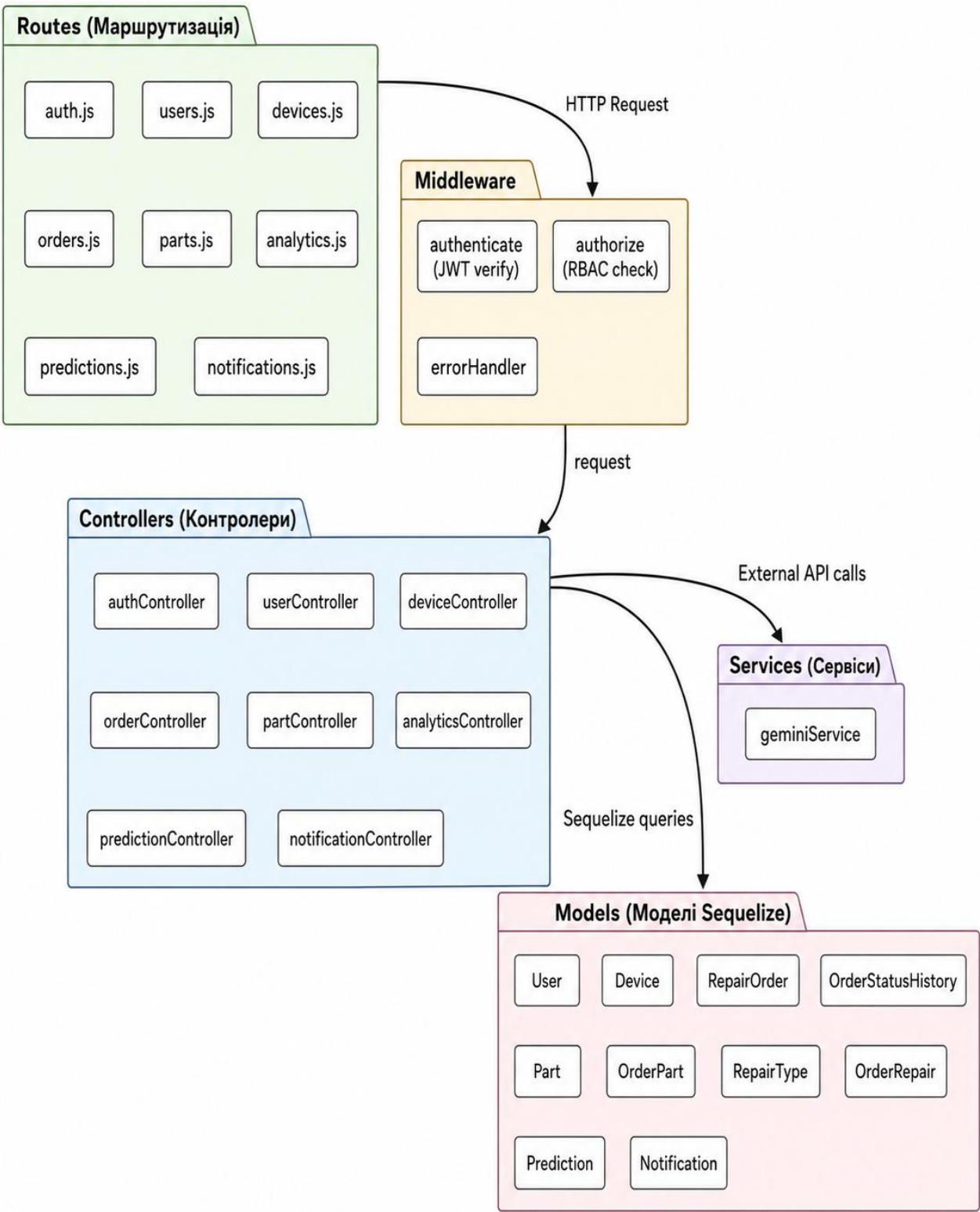


Рис. 2.2 – Діаграма компонентів серверної частини системи
Описана шарова організація серверної частини забезпечує однонаправлену

залежність між рівнями: рівень маршрутизації звертається до middleware, middleware – до контролерів, контролери – до моделей та сервісів. Такий підхід унеможливорює циклічні залежності, спрощує модульне тестування окремих компонентів та полегшує подальше розширення функціоналу без зміни суміжних рівнів. Визначені функціональні й нефункціональні вимоги та специфіковані варіанти використання є основою для проєктування схеми бази даних та детального опису API, що розглядається в наступному підрозділі.

2.2 Розробка бази даних системи

На основі аналізу предметної області та визначених функціональних вимог було виокремлено десять основних сутностей системи та встановлено зв'язки між ними [16]. Сутність User призначена для зберігання даних користувачів системи, зокрема адміністраторів, майстрів і клієнтів, із розмежуванням доступу відповідно до ролей. Сутність Device описує електронний пристрій, що належить клієнту, а RepairOrder є центральною сутністю бази даних, яка відображає заявку на ремонт. Для фіксації змін стану замовлення використовується сутність OrderStatusHistory, що зберігає інформацію про автора зміни, час її виконання та новий статус. Облік складських позицій реалізовано за допомогою сутності Part, тоді як OrderPart виконує роль проміжної таблиці для встановлення зв'язку «багато до багатьох» між ремонтними замовленнями та використаними запчастинами. Для опису видів ремонтних робіт використовується довідкова сутність RepairType, а OrderRepair забезпечує зв'язок між замовленнями та типами виконаних ремонтів. Результати прогнозування поломок зберігаються в сутності Prediction, яка пов'язана з відповідним пристроєм і містить дані AI-аналізу. Сутність Notification використовується для зберігання системних повідомлень, адресованих користувачам.

ER-діаграму бази даних, що відображає всі десять таблиць, їхні поля та зв'язки між ними, представлено на рисунку 2.3

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

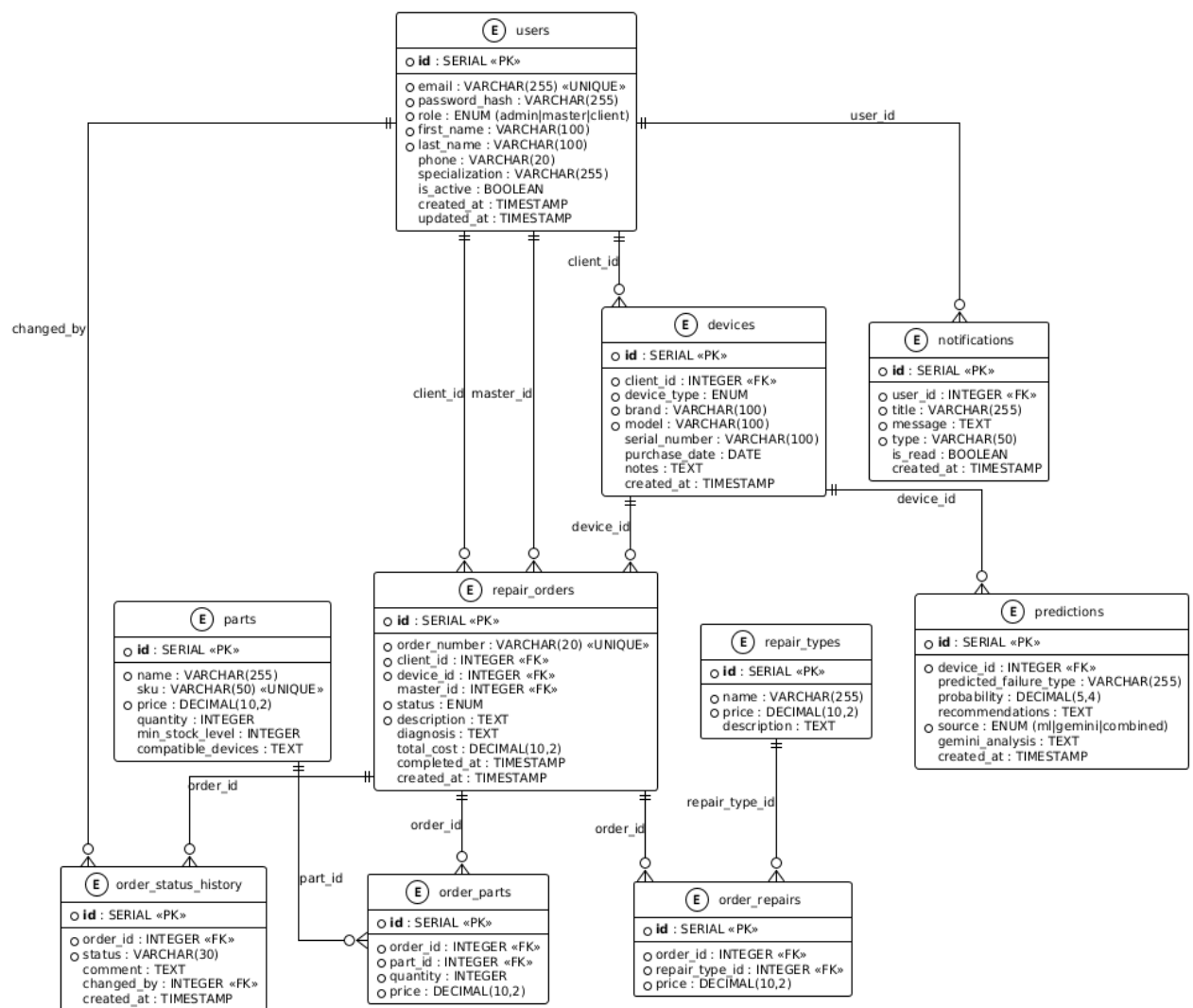


Рис. 2.3 – ER-діаграма бази даних системи

Нижче наведено структуру основних таблиць бази даних.

Таблиця 2.3

Структура таблиці users

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифіка- тор
email	VARCHAR(255)	UNIQUE, NOT NULL	Електронна пошта
password_hash	VARCHAR(255)	NOT NULL	Хеш пароля (bcrypt)
role	ENUM	NOT NULL, DEFAULT 'client'	Роль: admin, master, client
first_name	VARCHAR(100)	NOT NULL	Ім'я
last_name	VARCHAR(100)	NOT NULL	Прізвище
phone	VARCHAR(20)	-	Телефон

продовж. табл. 2.3

specialization	VARCHAR(255)	-	Спеціалізація (для майстрів)
is_active	BOOLEAN	DEFAULT true	Статус активності
created_at	TIMESTAMP	DEFAULT NOW()	Дата створення
updated_at	TIMESTAMP	DEFAULT NOW()	Дата оновлення

Таблиця users є базовою для реалізації механізмів автентифікації, авторизації та рольового розмежування доступу. Вона зберігає облікові дані користувачів, їхні персональні відомості, роль у системі та ознаку активності облікового запису. Наявність поля role дозволяє визначати доступний функціонал для адміністратора, майстра або клієнта, а поле is_active використовується для тимчасового блокування облікового запису без його видалення з бази даних.

Таблиця 2.4

Структура таблиці devices

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор
client_id	INTEGER	FK → users(id), NOT NULL	Власник пристрою
device_type	ENUM	NOT NULL	Тип: phone, laptop, tablet, other
brand	VARCHAR(100)	NOT NULL	Бренд виробника
model	VARCHAR(100)	NOT NULL	Модель
serial_number	VARCHAR(100)	-	Серійний номер
purchase_date	DATE	-	Дата придбання
notes	TEXT	-	Додаткові примітки
created_at	TIMESTAMP	DEFAULT NOW()	Дата створення

Таблиця devices використовується для зберігання інформації про пристрої, що належать клієнтам сервісного центру. Зв'язок із таблицею users через поле client_id дозволяє однозначно визначити власника кожного пристрою. Дані цієї таблиці мають важливе значення не лише для створення ремонтних замовлень, а й для роботи модуля прогнозування, оскільки тип пристрою, бренд, модель і дата

придбання використовуються під час аналізу ремонтної історії та оцінювання ризику майбутніх несправностей.

Таблиця 2.5

Структура таблиці repair_orders

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор
order_number	VARCHAR(20)	UNIQUE, NOT NULL	Номер замовлення (SR{YYMM}-{NNNN})
client_id	INTEGER	FK → users(id), NOT NULL	Клієнт
device_id	INTEGER	FK → devices(id), NOT NULL	Пристрій
master_id	INTEGER	FK → users(id)	Призначений майстер
status	ENUM	NOT NULL, DEFAULT 'new'	Статус замовлення
description	TEXT	NOT NULL	Опис проблеми клієнта
diagnosis	TEXT	-	Діагноз майстра
total_cost	DECIMAL(10,2)	DEFAULT 0	Загальна вартість ремонту
completed_at	TIMESTAMP	-	Дата завершення
created_at	TIMESTAMP	DEFAULT NOW()	Дата створення

Таблиця repair_orders є центральною таблицею бази даних, оскільки саме вона відображає основний бізнес-процес сервісного центру — обробку ремонтного замовлення. У ній зберігаються дані про клієнта, пристрій, призначеного майстра, опис заявленої проблеми, поточний статус, вартість ремонту та часові характеристики виконання робіт. Наявність унікального номера замовлення дає змогу використовувати його для внутрішнього обліку та публічного відстеження стану ремонту.

Таблиця 2.6

Структура таблиці order_status_history

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор
order_id	INTEGER	FK → repair_orders(id), NOT NULL	Замовлення
status	VARCHAR(30)	NOT NULL	Встановлений статус
comment	TEXT	-	Коментар до зміни
changed_by	INTEGER	FK → users(id)	Автор зміни
created_at	TIMESTAMP	DEFAULT NOW()	Дата та час зміни

Таблиця order_status_history доповнює основну таблицю repair_orders і забезпечує збереження повної хронології зміни статусів замовлення. Завдяки цій таблиці система не лише фіксує поточний стан ремонту, а й дозволяє переглядати послідовність усіх етапів обробки заявки із зазначенням автора зміни, часу виконання дії та коментаря. Такий підхід підвищує прозорість роботи сервісного центру та дає можливість відстежувати відповідальність працівників на кожному етапі ремонту.

Таблиця 2.7

Структура таблиці parts

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор
name	VARCHAR(255)	NOT NULL	Назва запчастини
sku	VARCHAR(50)	UNIQUE	Артикул
price	DECIMAL(10,2)	NOT NULL	Ціна одиниці
quantity	INTEGER	DEFAULT 0	Кількість на складі
min_stock_level	INTEGER	DEFAULT 5	Мінімальний залишок
compatible_devices	TEXT	-	Сумісні типи пристроїв

Таблиця parts призначена для організації складського обліку запчастин, які використовуються під час виконання ремонтних робіт. У ній зберігаються назва запчастини, артикул, ціна, фактична кількість на складі, мінімальний залишок і

перелік сумісних пристроїв. Поля quantity та min_stock_level використовуються для контролю залишків і виявлення позицій, які потребують поповнення. Це дозволяє зменшити ризик затримки ремонту через відсутність необхідних комплектуючих.

Таблиця 2.8

Структура таблиці predictions

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор
device_id	INTEGER	FK → devices(id), NOT NULL	Пристрій
predicted_failure_type	VARCHAR(255)	-	Прогнозований тип поломки
probability	DECIMAL(5,4)	-	Числова ймовірність (0–1)
recommendations	TEXT	-	JSON з переліком рекомендацій
source	ENUM	NOT NULL	Джерело: ml, gemini, combined
gemini_analysis	TEXT	-	Текстовий аналіз Gemini
created_at	TIMESTAMP	DEFAULT NOW()	Дата виконання прогнозу

Таблиця predictions використовується для збереження результатів прогнозування поломок техніки. Вона пов'язана з таблицею devices, оскільки кожен прогноз формується для конкретного пристрою. У таблиці зберігаються прогнозований тип несправності, числова ймовірність поломки, рекомендації щодо профілактичного обслуговування, джерело прогнозу та текстовий аналіз, сформований за допомогою Google Gemini. Поле source дозволяє розрізняти результати, отримані від ML-сервісу, Gemini-аналізу або комбінованого прогнозу.

Структури допоміжних таблиць order_parts, repair_types, order_repairs та notifications побудовані за аналогічним принципом і забезпечують підтримку додаткових бізнес-процесів системи. Таблиця order_parts реалізує зв'язок між ремонтними замовленнями та використаними запчастинами, зберігаючи їх кількість і вартість на момент додавання до замовлення. Таблиця repair_types виконує роль довідника видів ремонтних робіт, а order_repairs пов'язує конкретне замовлення з виконаними типами ремонту. Таблиця notifications призначена для зберігання системних повідомлень користувачів, зокрема сповіщень про зміну статусу замовлення або готовності пристрою до видачі.

Між таблицями бази даних визначено зв'язки, що відповідають бізнес-логіці предметної області. У таблиці 2.9 наведено опис усіх зв'язків із зазначенням їхньої кратності.

Таблиця 2.9

Зв'язки між таблицями бази даних

Батьківська таблиця	Дочірня таблиця	Кратність	Опис
users	devices	1 : M	Клієнт може мати кілька пристроїв
users	repair_orders (client_id)	1 : M	Клієнт може мати кілька замовлень
users	repair_orders (master_id)	1 : M	Майстер може виконувати кілька замовлень
devices	repair_orders	1 : M	Пристрій може мати кілька замовлень на ремонт
repair_orders	order_status_history	1 : M	Замовлення має повну хронологію зміни статусів
repair_orders	order_parts	M : N	Замовлення включає кілька запчастин (через order_parts)
parts	order_parts	M : N	Запчастина може бути використана в кількох замовленнях
repair_orders	order_repairs	M : N	Замовлення може включати кілька типів ремонту

repair_types	order_repairs	M : N	Тип ремонту може застосовуватись у кількох замовленнях
devices	predictions	1 : M	Для пристрою може бути виконано кілька прогнозів
users	notifications	1 : M	Користувач може отримувати кілька повідомлень

Наведені зв'язки забезпечують цілісність даних і відображають реальні залежності між об'єктами предметної області. Зв'язки типу “один до багатьох” використовуються для моделювання ситуацій, коли один клієнт може мати кілька пристроїв або кілька ремонтних замовлень, а один пристрій може багаторазово передаватися до сервісного центру. Зв'язки типу “багато до багатьох” реалізовано через проміжні таблиці `order_parts` та `order_repairs`, оскільки одне замовлення може містити кілька запчастин і кілька видів ремонтних робіт, а одна запчастина або один тип ремонту можуть використовуватися в різних замовленнях. Така структура бази даних забезпечує гнучкість, нормалізацію даних і можливість подальшого розширення функціональності системи.

Для всіх полів зовнішніх ключів створено індекси з метою прискорення JOIN-операцій. Додаткові індекси визначено на полях `status` таблиць `repair_orders` та `order_status_history` для оптимізації аналітичних запитів із фільтрацією за статусом.

Застосування Sequelize ORM дає змогу відобразити реляційну структуру бази даних у вигляді набору моделей, що використовуються безпосередньо в серверній частині системи. Кожна модель відповідає окремій таблиці бази даних, а асоціації між моделями відображають зв'язки, визначені на рівні зовнішніх ключів. Завдяки цьому серверна частина може працювати з даними через об'єктний інтерфейс, зберігаючи при цьому цілісність реляційної структури та коректність взаємодії між сутностями.

Діаграму класів моделей даних, що відображає об'єктну структуру

Sequelize ORM-моделей із їхніми полями та асоціаціями, представлено на рисунку 2.4 [17].

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

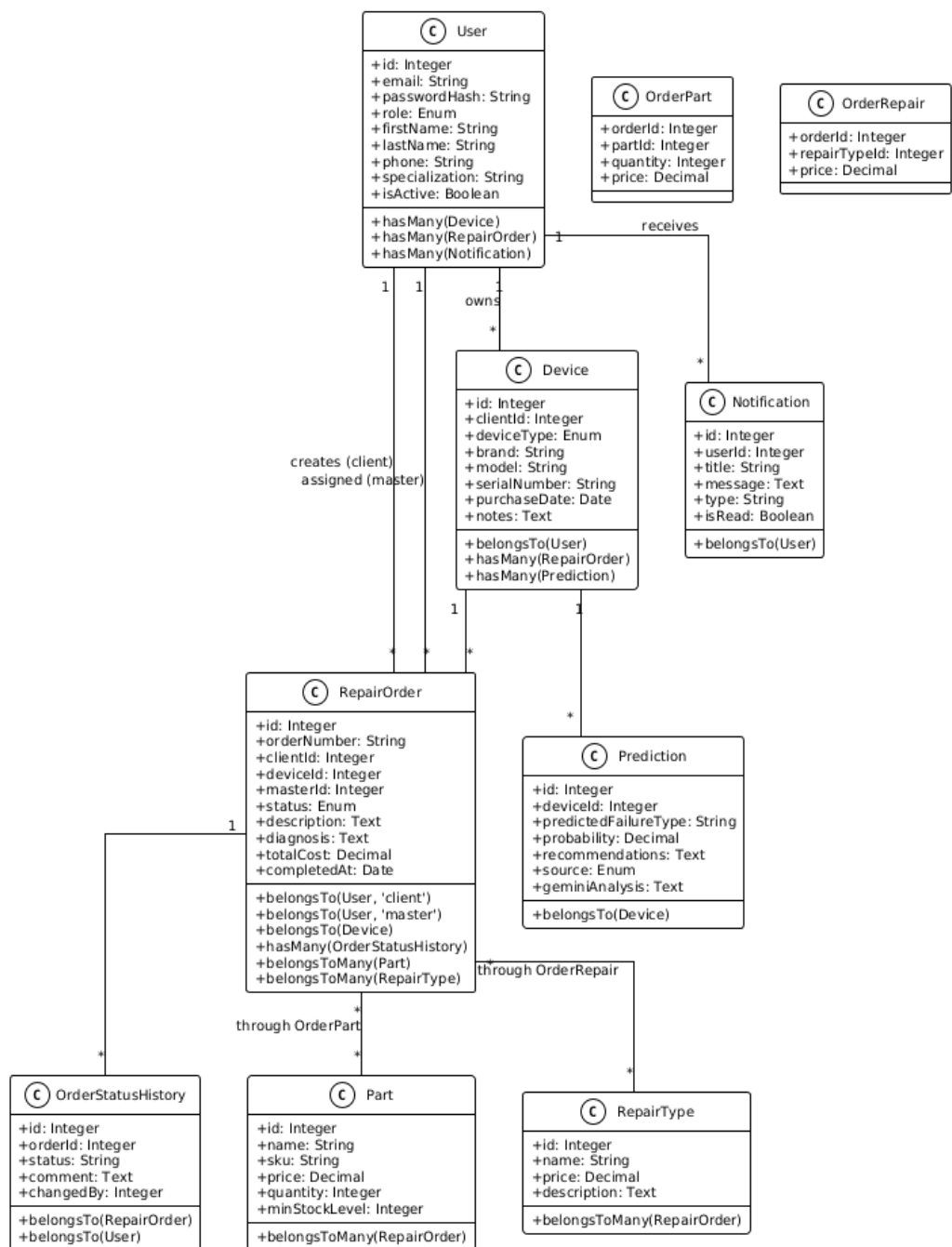


Рис. 2.4 – Діаграма класів моделей даних Sequelize ORM

Спроектowana реляційна схема забезпечує цілісність даних через систему зовнішніх ключів, підтримку повної хронології ремонтів через таблицю `order_status_history` та ефективне зберігання результатів AI-прогнозування. Описана структура бази даних є основою для проектування REST API, що розглядається в наступному підрозділі.

2.3 Проектування алгоритмів роботи системи та математичне забезпечення модуля прогнозування

Розроблювана система реалізує кілька взаємопов'язаних алгоритмів: управління життєвим циклом замовлення, автентифікацію та авторизацію користувачів, машинне прогнозування поломок, аналіз природною мовою засобами LLM та агрегацію результатів комбінованого прогнозу. Кожен алгоритм деталізовано нижче разом із відповідними UML-діаграмами.

Життєвий цикл замовлення на ремонт реалізовано як скінченний автомат (Finite State Machine), що передбачає послідовне проходження замовлення через визначені стани. У системі використовуються такі статуси: *new* - нова заявка, *diagnostics* - діагностика, *in_progress* - в роботі, *waiting_parts* – очікування запчастин, *ready* - готово та *issued* – видано. Дозволені переходи між станами наведено на рисунку 2.5. Такий підхід дає змогу обмежити довільну зміну статусів і забезпечити проходження замовлення відповідно до встановленого маршруту обробки.

Використання скінченного автомата є доцільним для моделювання процесу ремонту, оскільки кожне замовлення в будь-який момент часу може перебувати лише в одному визначеному стані. Перехід до наступного стану відбувається тільки після виконання певної дії користувача, наприклад завершення діагностики, початку ремонтних робіт, очікування необхідних запчастин або підтвердження готовності пристрою до видачі. Завдяки цьому система запобігає некоректним сценаріям, зокрема переведенню нової заявки одразу у статус «видає» без проходження проміжних етапів.

Кожна зміна статусу додатково фіксується в історії замовлення із зазначенням нового стану, часу зміни, автора дії та коментаря. Це забезпечує прозорість виконання ремонтного процесу, дозволяє адміністратору контролювати роботу майстрів і надає клієнту можливість отримувати актуальну інформацію про стан свого замовлення. Таким чином, алгоритм управління життєвим циклом замовлення поєднує контроль допустимих переходів зі збереженням повної хронології обробки заявки.

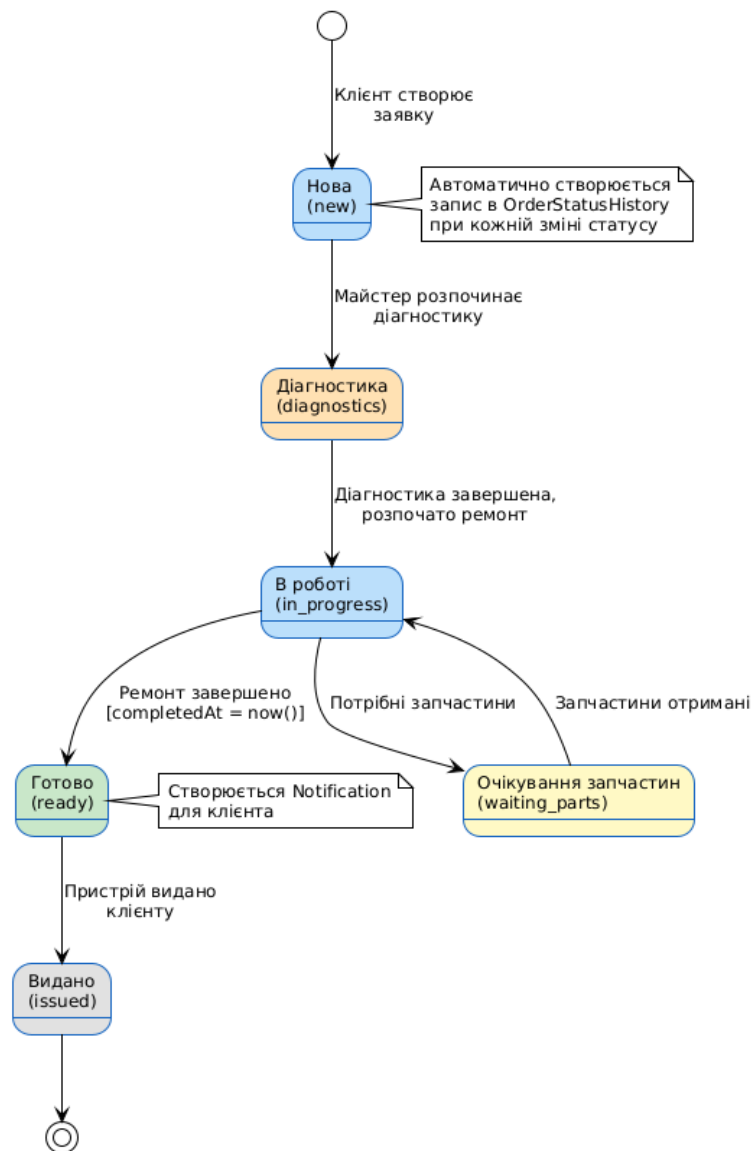


Рис. 2.5 – Діаграма станів замовлення на ремонт

Алгоритм зміни статусу замовлення передбачає отримання нового статусу, коментаря та ідентифікатора користувача, після чого система перевіряє права доступу. Адміністратор може змінювати статус будь-якого замовлення, а майстер — лише призначеного йому. Після успішної перевірки оновлюється поле status у таблиці repair_orders, а в order_status_history створюється запис із новим статусом, коментарем, автором зміни та часом виконання. Якщо статус змінено на ready або issued, додатково фіксується час завершення в полі completed_at, після чого для клієнта створюється повідомлення в таблиці notifications.

Автентифікація в системі реалізована на основі JWT із використанням access token для захищених API-запитів і refresh token для оновлення сесії. У

реалізованому прототипі обидва токени зберігаються в localStorage клієнтського застосунку: access token передається в заголовок Authorization, а refresh token використовується під час звернення до /api/auth/refresh. Такий підхід забезпечує збереження сесії між перезавантаженнями сторінки, але потребує захисту клієнтської частини від XSS-атак [24]. Діаграму послідовності процесу входу в систему наведено на рисунку 2.6.

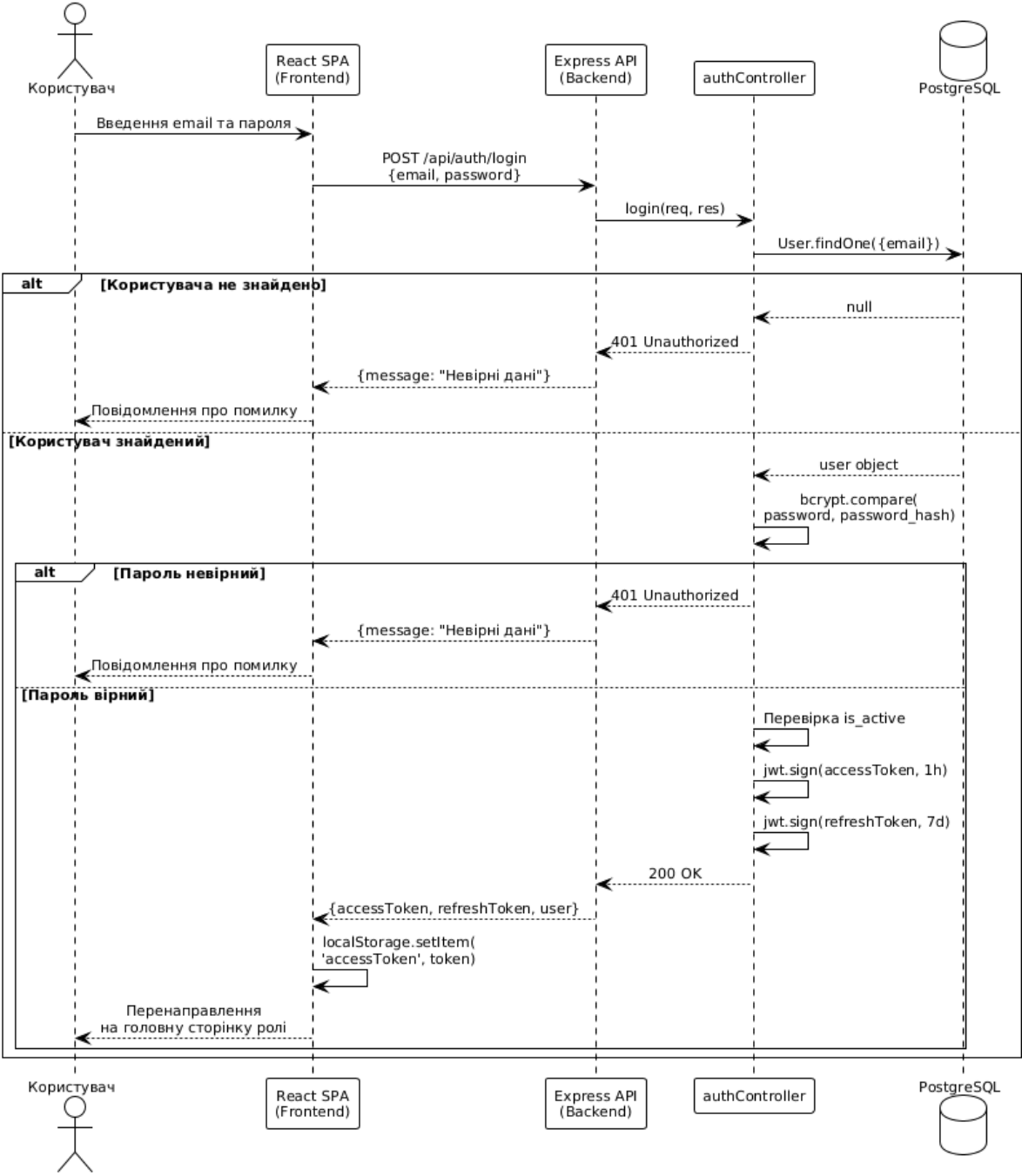


Рис. 2.6 – Діаграма послідовності процесу автентифікації

Алгоритм входу користувача в систему починається з отримання POST-запиту, що містить email та пароль. Далі система виконує пошук облікового запису в базі даних за вказаною адресою електронної пошти та порівнює хеш введеного пароля зі збереженим хешем за допомогою функції `bcrypt.compare()`. Додатково перевіряється значення поля `is_active`, оскільки заблокованим обліковим записам доступ до системи заборонено. У разі успішного проходження всіх перевірок система генерує `access token` із `payload`, що містить ідентифікатор користувача та його роль, а також `refresh token` із `payload`, що містить ідентифікатор користувача. `Access token` має термін дії одну годину, а `refresh token` – сім днів. Після цього користувачу повертаються токени та об'єкт користувача без поля хешу пароля.

Перевірка автентичності запиту виконується за допомогою `middleware authenticate`. Під час обробки запиту система витягує токен із заголовка `Authorization` у форматі `Bearer token`, після чого виконує перевірку його підпису та терміну дії за допомогою `jwt.verify()` і секретного ключа. Якщо токен є коректним, із нього отримується ідентифікатор користувача, за яким виконується пошук відповідного запису в базі даних. Після успішної перевірки об'єкт користувача додається до `req.user`, що дозволяє використовувати його дані в наступних `middleware` та контролерах.

Авторизація користувачів реалізована за допомогою `middleware authorize`, який перевіряє відповідність ролі поточного користувача вимогам конкретного маршруту. Для цього `middleware` отримує масив дозволених ролей і порівнює його зі значенням `req.user.role`. Якщо роль користувача відповідає хоча б одній із дозволених ролей, запит передається до відповідного контролера. У разі невідповідності система повертає відповідь `HTTP 403 Forbidden` без подальшої обробки запиту.

Модуль прогнозування поломок техніки базується на ансамблевому методі машинного навчання `Random Forest`. Задача прогнозування формалізується як

поєднання трьох підзадач: бінарної класифікації наявності ризику поломки, багатокласової класифікації типу можливої несправності та регресійного оцінювання ймовірності відмови пристрою.

Бінарна класифікація визначає, чи існує підвищений ризик виходу пристрою з ладу в найближчій перспективі. Формалізацію цієї задачі наведено у формулі (2.1):

$$f_{class}: X \rightarrow \{0,1\} \quad (2.1)$$

де X — вектор ознак пристрою; 0 — поломка малоїмовірна; 1 — поломка ймовірна.

Друга підзадача полягає у визначенні найбільш імовірного типу майбутньої поломки. Вона розглядається як задача багатокласової класифікації та описується формулою (2.2):

$$f_{type}: X \rightarrow \{c_1, c_2, \dots, c_{12}\} \quad (2.2)$$

де c_i — один із можливих типів поломок, зокрема пошкодження екрана, деградація батареї, проблема з роз'ємом зарядки, несправність материнської плати тощо.

Третя підзадача полягає у регресійному оцінюванні числової ймовірності відмови пристрою. Відповідна функція прогнозування подана у формулі (2.3):

$$f_{prob}: X \rightarrow [0,1] \quad (2.3)$$

де значення функції належить проміжку від 0 до 1 та інтерпретується як ймовірність виникнення поломки.

Для кожного пристрою формується вектор ознак $X=(x_1, x_2, \dots, x_8)$, склад якого наведено у таблиці 2.10.

Таблиця 2.10

Склад вектора ознак для моделі прогнозування

Ознака	Тип	Опис
x_1 — device_type	Категоріальна	Тип пристрою (phone/laptop/tablet)
x_2 — brand	Категоріальна	Бренд виробника
x_3 — last_repair_type	Категоріальна	Тип останнього ремонту

x_4 — age_months	Числова	Вік пристрою (місяці)
x_5 — total_repairs	Числова	Загальна кількість ремонтів
x_6 — months_since_last_repair	Числова	Місяці від останнього ремонту
x_7 — total_cost	Числова	Загальна вартість попередніх ремонтів
x_8 — season	Числова	Поточний сезон (1-4)

Категоріальні ознаки кодуються методом Label Encoding, а числові ознаки стандартизуються за формулою (2.4):

$$x_{scaled} = \frac{x - \mu}{\sigma} \quad (2.4)$$

де x — початкове значення ознаки; μ — середнє значення ознаки; σ — стандартне відхилення.

Алгоритм Random Forest будує ансамбль із B дерев рішень, кожне з яких навчається на випадковій підвибірці даних та випадковій підмножині ознак. Для задачі класифікації фінальне рішення визначається шляхом голосування більшості дерев, як показано у формулі (2.5):

$$\hat{y} = \text{mode}\{h_b(x)\}_{b=1}^B \quad (2.5)$$

де $h_b(x)$ — прогноз b -го дерева рішень; B — загальна кількість дерев в ансамблі; $\hat{y}$ — підсумковий прогноз моделі.

Для задачі регресії підсумкова ймовірність визначається шляхом усереднення прогнозів усіх дерев ансамблю за формулою (2.6):

$$\hat{y} = \frac{1}{B} \sum_{b=1}^B h_b(x) \quad (2.6)$$

де $h_b(x)$ — прогноз b -го дерева рішень, а B — кількість дерев в ансамблі.

У системі використовуються три моделі Random Forest: класифікатор наявності ризику поломки зі 100 деревами та $\text{max_depth} = 10$, класифікатор типу поломки зі 100 деревами та $\text{max_depth} = 12$, а також регресор ймовірності відмови

зі 100 деревами та $\max_depth = 10$. Реалізацію моделей виконано засобами бібліотеки `scikit-learn`, яка надає готові інструменти для побудови `RandomForestClassifier` та `RandomForestRegressor`, налаштування кількості дерев, глибини дерев і параметрів навчання ансамблю.

Для навчання моделей згенеровано синтетичний набір даних обсягом 800 записів із реалістичними розподілами. Ймовірності поломок залежать від типу пристрою, його віку, кількості ремонтів та бренду. Для моделювання кількості ремонтів використано розподіл Пуассона, наведений у формулі (2.7):

$$N_{repairs} \sim Pois(\lambda), \text{ де } \lambda = age_months/18 \quad (2.7)$$

де $N_{repairs}$ – кількість попередніх ремонтів пристрою; λ – параметр розподілу Пуассона; age_months – вік пристрою в місяцях.

Результати навчання моделей наведено у таблиці 2.11.

Таблиця 2.11

Метрики якості навчених моделей

Модель	Метрика	Значення
Класифікатор поломок	Accuracy	0.8875
Класифікатор поломок	F1-score	0.8902
Класифікатор поломок	Cross-validation (5-fold)	0.8850 ± 0.0161
Регресор ймовірності	MAE	0.0557
Регресор ймовірності	RMSE	0.0660

Наведені метрики дають змогу оцінити якість роботи моделей із різних сторін. Accuracy та F1-score використовуються для аналізу класифікаційної моделі, оскільки показують загальну частку правильних прогнозів і збалансованість результатів між класами. Показник cross-validation підтверджує стабільність моделі на різних підвибірках навчального набору. Для регресійної моделі використано MAE та RMSE, які характеризують середню величину помилки під час прогнозування числової ймовірності відмови пристрою.

Аналіз важливості ознак показав, що найбільший внесок у прогнозне рішення мають такі характеристики:

1. Вік пристрою (age_months) – 34.45%.

2. Кількість попередніх ремонтів (total_repairs) – 20.69%.
3. Загальна вартість ремонтів (total_cost) – 17.76%.
4. Час від останнього ремонту (months_since_last_repair) – 11.04%.

Отримані результати свідчать про достатню якість побудованих моделей для використання в межах розроблюваної системи. Класифікатор типу поломки демонструє збалансовані значення Accuracy та F1-score, що вказує на здатність моделі коректно визначати найбільш імовірну категорію несправності. Регресійна модель, яка використовується для оцінювання числової ймовірності відмови пристрою, показує низьке значення середньої абсолютної помилки, тому її результати можуть бути використані як орієнтовна кількісна оцінка ризику.

Важливо зазначити, що отримані метрики характеризують якість моделей у межах сформованої синтетичної навчальної вибірки. Тому результати прогнозування слід розглядати як демонстрацію працездатності обраного підходу та програмної реалізації модуля машинного навчання. Для використання системи в умовах реального сервісного центру доцільним є подальше донавчання моделей на фактичних історичних даних про ремонтні звернення, що дозволить підвищити точність прогнозів і краще врахувати специфіку конкретного підприємства.

У межах роботи ML-модуль розглядається не лише як окрема модель машинного навчання, а як частина загального процесу обробки запиту на прогнозування. Після отримання запиту система формує набір вхідних ознак пристрою, виконує їх попередню обробку, передає підготовлені дані до навчених моделей Random Forest та отримує результати класифікації і регресійного оцінювання. На основі отриманих значень формується відповідь, що містить імовірність відмови, прогнозований тип поломки, рівень ризику та дані, необхідні для подальшого відображення результату в інтерфейсі користувача.

Діаграму діяльності, що ілюструє повний процес ML-прогнозування від отримання запиту до формування відповіді, наведено на рисунку 2.7.

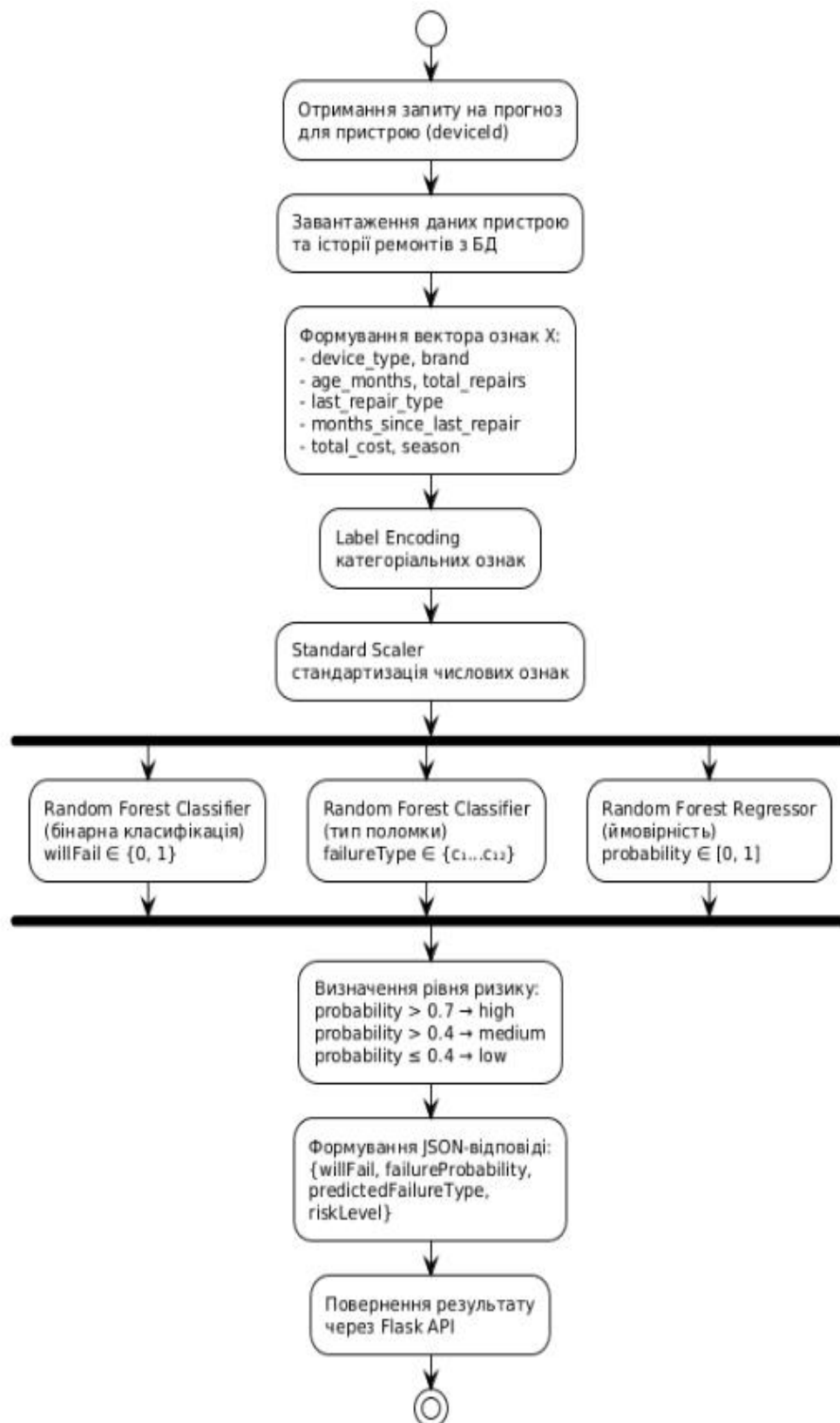


Рис. 2.7 – Діаграма діяльності алгоритму ML-прогнозування

Для поглибленого аналізу ремонтної історії пристроїв у системі використо-
вується Google Gemini 2.5 Flash – велика мовна модель, яка обробляє структуро-

Змн.	Арк.	№ докум.	Підпис	Дата

ваний текстовий промпт з описом пристрою та його попередніх ремонтів і повертає відповідь у форматі JSON. Такий підхід дає змогу доповнити результати машинного навчання природномовним аналізом ремонтної історії та сформулювати рекомендації, зрозумілі для користувача.

Алгоритм Gemini-аналізу починається зі збору інформації про пристрій, зокрема його тип, бренд, модель і вік, а також повної хронології ремонтів із датами, описами несправностей, діагнозами та вартістю виконаних робіт. На основі цих даних система формує промпт українською мовою, у якому задається вимога повернути результат у структурованому JSON-форматі. Очікувана відповідь містить загальний текстовий аналіз у полі `analysis`, масив прогнозованих поломок у полі `predictedFailures`, масив рекомендацій у полі `recommendations` та рівень ризику в полі `riskLevel` зі значенням `low`, `medium` або `high`.

Після формування промпту система надсилає запит до Google Gemini API через пакет `@google/generative-ai`. Отримана відповідь проходить етап парсингу та перевірки відповідності очікуваній структурі JSON. Після успішної валідації результати аналізу зберігаються в таблиці `predictions` із зазначенням джерела прогнозу в полі `source` зі значенням `gemini`.

Варто зазначити, що Gemini-аналіз у межах розробленої системи не замінює машинну модель прогнозування, а виконує допоміжну пояснювальну функцію. Його основне призначення полягає у формуванні зрозумілого текстового обґрунтування результатів, узагальненні ремонтної історії пристрою та підготовці практичних рекомендацій для користувача. Завдяки цьому числові результати ML-моделі доповнюються природномовним поясненням, що підвищує зручність сприйняття прогнозу адміністратором, майстром або клієнтом.

Під час запуску комбінованого прогнозу система паралельно виконує ML-прогноз та Gemini-аналіз, після чого агрегує отримані результати в єдиний прогноз. Діаграму послідовності, що ілюструє паралельне виконання двох модулів прогнозування, наведено на рисунку 2.8.

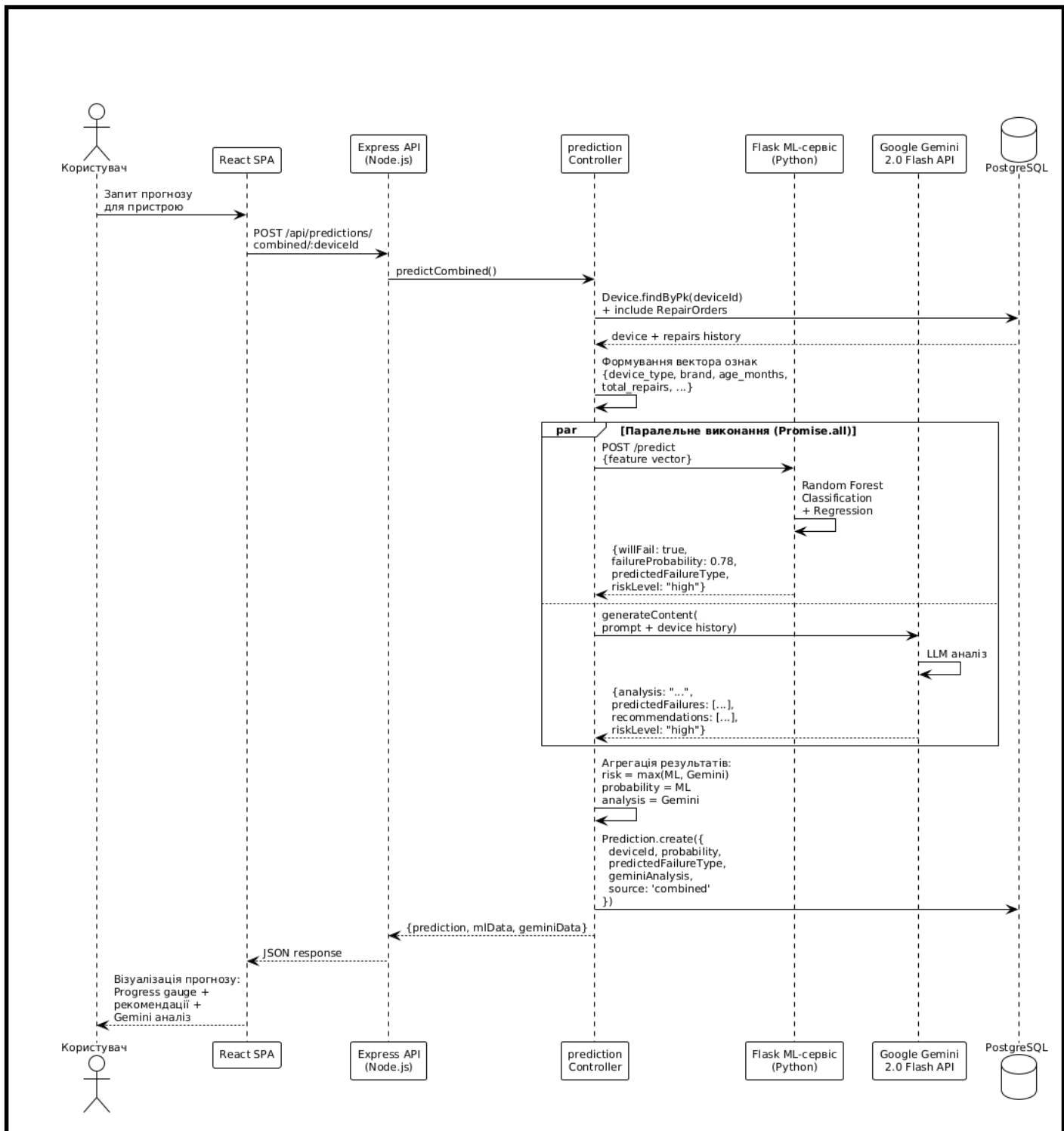


Рис. 2.8 – Діаграма послідовності комбінованого AI-прогнозування

Агрегація результатів починається з паралельного запуску ML-прогнозу та Gemini-аналізу засобами Promise.all, що дозволяє мінімізувати загальний час формування відповіді. Після отримання результатів система визначає підсумковий рівень ризику шляхом вибору вищого значення між оцінкою ML-моделі та оцінкою Gemini-аналізу. Числова ймовірність поломки береться з ML-моделі, оскільки саме вона забезпечує кількісну оцінку ризику на основі структурованих даних. Тип прогнозованої несправності також визначається за результатами ML-

класифікатора, тоді як текстовий аналіз і рекомендації для користувача формуються на основі відповіді Gemini.

Після узгодження результатів система зберігає агрегований прогноз у таблиці predictions із зазначенням джерела прогнозу в полі source зі значенням combined. Реалізований підхід забезпечує повний технологічний ланцюг від прийняття заявки на ремонт до формування AI-рекомендацій для клієнта. Поєднання ML-моделі та LLM-аналізу дозволяє об'єднати кількісну оцінку ризику зі зрозумілим природномовним поясненням, що є однією з ключових переваг розроблюваної системи порівняно з розглянутими аналогами.

2.4 Реалізація серверної частини та REST API вебсистеми

Серверна частина системи реалізована у вигляді RESTful API із використанням Express.js. Усі ендпоінти групуються за ресурсами та доступні за префіксом /api/. Реалізація побудована на принципах єдиної відповідальності та розділення рівнів, описаних у підрозділі 1.3.

Повний перелік ендпоінтів API із зазначенням HTTP-методів, шляхів та вимог до авторизації наведено у таблицях 2.12–2.16.

Таблиця 2.12

Ендпоінти модуля автентифікації (/api/auth)

Метод	Шлях	Опис	Доступ
POST	/register	Реєстрація клієнта	Публічний
POST	/login	Автентифікація	Публічний
POST	/refresh	Оновлення токенів	Публічний
GET	/me	Профіль поточного користувача	Авторизований

Модуль автентифікації забезпечує початкову взаємодію користувача із системою та відповідає за створення захищеної сесії роботи з API. Публічні маршрути реєстрації, входу та оновлення токенів доступні без попередньої авторизації, оскільки саме вони використовуються для отримання або поновлення облікових даних доступу. Маршрут отримання профілю поточного користувача належить до захищених, тому виконується лише після перевірки JWT access token.

Таблиця 2.13

Ендпоінти модуля замовлень (/api/orders)

Метод	Шлях	Опис	Доступ
GET	/	Список замовлень (з пагінацією)	Admin, Master, Client
POST	/	Створення замовлення	Admin, Client
GET	/:id	Деталі замовлення	Admin, Master, Client
PUT	/:id	Оновлення замовлення	Admin, Master
PUT	/:id/status	Зміна статусу	Admin, Master
POST	/:id/parts	Додавання запчастини	Admin, Master
POST	/:id/repair-types	Додавання типу ремонту	Admin, Master
GET	/:id/timeline	Хронологія статусів	Admin, Master, Client
GET	/track/:orderNumber	Відстеження (гість)	Публічний

Модуль замовлень є центральною частиною серверного API, оскільки забезпечує обробку основного бізнес-процесу сервісного центру. Через ці ендпоінти виконуються створення ремонтного замовлення, перегляд списку заявок із пагінацією, отримання детальної інформації про конкретне замовлення, зміна статусу, додавання використаних запчастин і типів ремонтних робіт, а також перегляд хронології виконання замовлення. Доступ до маршрутів цього модуля обмежується відповідно до ролі користувача: адміністратор має повний доступ, майстер працює переважно з призначеними замовленнями, а клієнт може переглядати лише власні заявки.

Таблиця 2.14

Ендпоінти модуля пристроїв (/api/devices)

Метод	Шлях	Опис	Доступ
GET	/	Список пристроїв	Admin, Master, Client
POST	/	Реєстрація пристрою	Admin, Client
GET	/:id	Деталі пристрою	Admin, Master, Client
PUT	/:id	Оновлення пристрою	Admin, Client
POST	/:id/photo	Завантаження фото пристрою	Admin, Client
DELETE	/:id	Видалення пристрою	Admin
GET	/:id/history	Історія ремонтів пристрою	Admin, Master, Client

Модулі користувачів, пристроїв і запчастин забезпечують підтримку довідкових та облікових даних, необхідних для роботи основного процесу ремонту. Ендпоінти користувачів використовуються адміністратором для керування обліковими записами, їхніми ролями та станом активності. Модуль пристроїв дає змогу зберігати інформацію про техніку клієнтів і пов'язувати її з ремонтними замовленнями, а модуль запчастин відповідає за ведення складського обліку, контроль залишків і оновлення кількості комплектуючих після використання в ремонті.

Таблиця 2.15

Ендпоінти модуля прогнозування (/api/predictions)

Метод	Шлях	Опис	Доступ
POST	/ml/:deviceId	ML-прогноз	Admin, Master
POST	/gemini/:deviceId	Gemini-аналіз	Admin, Master
POST	/combined/:deviceId	Комбінований прогноз	Admin, Master
GET	/device/:deviceId	Прогнози пристрою	Admin, Master, Client

Аналітичні ендпоінти призначені для формування агрегованих показників діяльності сервісного центру. Вони виконують запити до бази даних, узагальнюють інформацію про кількість замовлень, дохід, статуси заявок, типи техніки, бренди, види ремонтів і продуктивність майстрів. Отримані дані використовуються клієнтською частиною для побудови статистичних карток, графіків і таблиць на адміністративному дашборді.

Таблиця 2.16

Ендпоінти модуля аналітики (/api/analytics)

Метод	Шлях	Опис	Доступ
GET	/dashboard	Загальна статистика	Admin
GET	/orders-by-period	Динаміка замовлень за період	Admin
GET	/failure-stats	Статистика поломок	Admin
GET	/masters-performance	Продуктивність майстрів	Admin

Модуль прогнозування та сповіщень відповідає за інтелектуальну складову системи й інформування користувачів про важливі події. Ендпоінти прогнозування забезпечують запуск ML-прогнозу, Gemini-аналізу або комбінованого AI-прогнозу для конкретного пристрою, після чого результати зберігаються в базі даних і можуть бути повторно переглянуті користувачем. Ендпоінти сповіщень використовуються для отримання повідомлень про зміну статусу замовлення, готовність пристрою до видачі або інші події, пов'язані з ремонтним процесом.

Для захисту серверної частини від типових вебатак реалізовано комплекс middleware-компонентів, що підключаються до Express-застосунку на глобальному рівні до оголошення маршрутів.

Пакет Helmet автоматично встановлює набір HTTP-заголовків безпеки: Content-Security-Policy для захисту від XSS-атак, X-Frame-Options для запобігання clickjacking, Strict-Transport-Security для примусового використання HTTPS, X-Content-Type-Options для запобігання MIME-sniffing [24]. Конфігурацію Helmet встановлено з параметром Cross-Origin-Resource-Policy у режимі cross-origin для забезпечення коректної взаємодії з клієнтом на окремому порті [20].

Для захисту від атак типу brute force реалізовано два рівні обмеження кількості запитів (rate limiting) з використанням пакету express-rate-limit. Загальний лімітер обмежує кількість запитів до API до 1000 за 15 хвилин з однієї IP-адреси та застосовується до всіх маршрутів /api/. Спеціалізований лімітер автентифікації обмежує кількість запитів до маршрутів входу, реєстрації та оновлення токена до 200 за 15 хвилин [23]. У разі перевищення ліміту сервер повертає статус HTTP 429 Too Many Requests.

Серверна валідація вхідних даних реалізована через бібліотеку express-validator. Визначено централізований модуль validators.js, що містить набори правил перевірки для кожного типу запиту: registerValidator, loginValidator, createDeviceValidator, createOrderValidator, updateStatusValidator, addPartValidator та інші.

Кожен валідатор є масивом middleware-функцій, що послідовно перевіряють поля запиту та завершуються функцією validate, яка у разі невалідних даних повертає HTTP 400 із масивом описів помилок. Валідатори підключаються до маршрутів між middleware авторизації та контролером, що гарантує коректність даних ще до виконання бізнес-логіки.

Middleware authenticate перехоплює кожен запит до захищеного маршруту та виконує перевірку у такій послідовності: витягування Bearer-токена із заголовка Authorization, верифікація підпису та терміну дії токена через jwt.verify() із секретним ключем із змінних оточення, пошук користувача в базі даних за декодованим ідентифікатором, перевірка активності облікового запису (is_active). У разі успіху об'єкт користувача додається до req.user та передається до наступного обробника. За будь-якої помилки перевірки повертається HTTP 401 Unauthorized.

Middleware authorize реалізовано як функцію вищого порядку, що приймає список дозволених ролей як аргумент і повертає middleware-функцію. Перевірка зводиться до встановлення наявності ролі req.user.role у переданому масиві дозволених значень. У разі невідповідності повертається HTTP 403 Forbidden без виклику контролера. Така реалізація дозволяє декларативно описувати вимоги до доступу безпосередньо в оголошенні маршруту.

Контролер замовлень містить найбільш складну бізнес-логіку системи. Діаграму послідовності процесу зміни статусу замовлення наведено на рисунку 2.9.

Таким чином, структура REST API охоплює всі основні функціональні модулі системи: автентифікацію, управління замовленнями, користувачами, пристроями, запчастинами, аналітикою, прогнозуванням і сповіщеннями. Групування ендпоінтів за ресурсами спрощує супровід серверної частини, забезпечує логічну організацію маршрутів і дозволяє надалі розширювати функціональність системи без порушення загальної структури API.

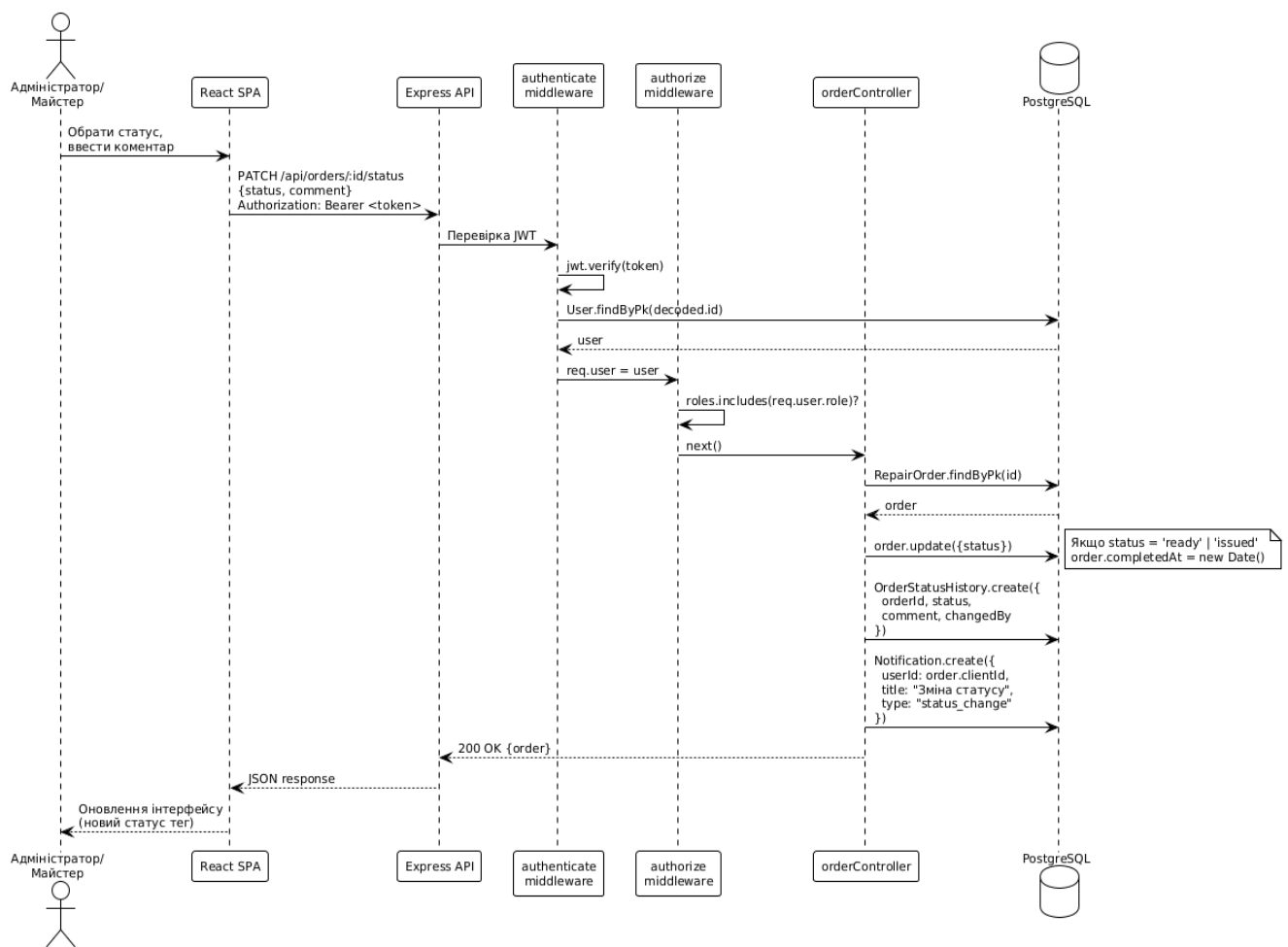


Рис. 2.9 – Діаграма послідовності зміни статусу замовлення

Метод зміни статусу замовлення виконується у такій послідовності: отримання нового статусу та коментаря з тіла запиту, оновлення поля status у таблиці repair_orders, запис у таблицю order_status_history із зазначенням автора та часу зміни, фіксація поля completed_at при встановленні статусу ready або issued, створення запису в таблиці notifications для клієнта-власника замовлення. Повний вихідний код контролера наведено у додатку А.

Метод додавання запчастини до замовлення реалізовано із використанням транзакції бази даних для забезпечення атомарності операції декременту складського залишку. Механізм LOCK.UPDATE засобами Sequelize запобігає виникненню стану гонки (race condition) при одночасному доступі кількох користувачів до одного запису запчастини: якщо кількість на складі є меншою за запитувану — транзакція відкочується та повертається HTTP 400 із відповідним повідомленням [26].

Для завантаження фотографій пристроїв застосовано бібліотеку Multer — middleware для обробки multipart/form-data запитів. Конфігурація Multer визначає: цільову директорію збереження файлів (uploads/devices/), унікальне ім'я файлу на основі 16-байтного криптографічно випадкового рядка зі збереженням оригінального розширення, фільтр MIME-типів (дозволено лише image/jpeg, image/png, image/webp) та ліміт розміру файлу 5 МБ. Завантажені файли стають доступними через статичний маршрут /uploads, а відносний шлях зберігається у полі photoUrl моделі Device.

Аналітичний контролер формує відповіді на основі серії агрегатних SQL-запитів, що виконуються через метод sequelize.query() з параметром replacements для запобігання SQL-ін'єкціям. Запит статистики полумок за типом техніки групує замовлення за полем device_type таблиці devices із підрахунком кількості через COUNT(). Запит продуктивності майстрів обчислює для кожного майстра кількість виконаних та активних замовлень, середній час ремонту через різницю між completed_at та created_at, а також загальну суму доходу через SUM(total_cost). Усі аналітичні запити підтримують фільтрацію за часовим проміжком через параметри dateFrom та dateTo, що передаються з клієнта.

Реалізована серверна частина забезпечує повний цикл обробки запитів від перевірки безпеки та валідації до виконання бізнес-логіки та формування структурованих JSON-відповідей. Описані middleware-компоненти, контролери та аналітичні запити є основою для реалізації клієнтської частини, що розглядається в наступному підрозділі.

2.5 Реалізація клієнтської частини вебсистеми

Клієнтська частина системи реалізована як SPA (Single Page Application) на базі React 18 з використанням бібліотеки компонентів Ant Design 5 та інструменту збірки Vite. Весь клієнтський код розміщено в директорії client/src/ та організовано за модульним принципом: api/ – HTTP-клієнт та функції запитів; context/ – контекст автентифікації; components/ – спільні компоненти макету;

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

pages/ – компоненти сторінок, згруповані за роллю (admin/, master/, client/, guest/, auth/); utils/ – допоміжні функції та константи.

Маршрутизацію реалізовано засобами React Router 6 з вкладеними маршрутами та компонентом ProtectedRoute. Компонент ProtectedRoute перевіряє два критерії перед відображенням сторінки: наявність автентифікованого користувача та відповідність його ролі вимогам маршруту. При незадоволенні першого критерію виконується перенаправлення на сторінку входу, другого – на головну сторінку відповідної ролі. Таким чином, несанкціонований прямий доступ до сторінок адміністратора або майстра унеможливлений на рівні клієнтського маршрутизатора.

Стан автентифікації управляється через React Context API у компоненті AuthProvider. Провайдер зберігає об'єкт поточного користувача та надає функції login, register та logout усім дочірнім компонентам через хук useAuth. При завантаженні застосунку виконується автоматична перевірка наявності збереженого access token та завантаження профілю користувача через GET /api/auth/me. Це забезпечує збереження сесії між перезавантаженнями сторінки без повторного введення облікових даних.

Для взаємодії з REST API використовується бібліотека Axios з двома налаштованими інтерцепторами. Request interceptor автоматично витягує access token із localStorage та додає його до заголовка Authorization: Bearer усіх запитів, що усуває необхідність ручного налаштування заголовків у кожній API-функції.

Response interceptor обробляє відповідь HTTP 401, яка свідчить про недійсний або прострочений access token. У разі отримання такої відповіді інтерцептор автоматично зчитує refresh token із localStorage, виконує запит POST /api/auth/refresh, отримує новий access token, зберігає оновлені токени та повторює оригінальний запит із новим заголовком Authorization. Якщо оновлення токена завершується невдачею, виконується очищення збережених токенів, вихід із системи та перенаправлення користувача на сторінку входу. Такий механізм забезпечує прозоре для користувача відновлення сесії без примусового повторного входу.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

Одним із найбільш візуально насичених компонентів системи є дашборд адміністратора, реалізований із використанням бібліотеки Recharts. Він призначений для узагальненого відображення основних показників діяльності сервісного центру та містить набір інтерактивних графіків, діаграм і статистичних елементів. У верхній частині дашборду розміщено блок статистичних карток, які відображають ключові показники системи, зокрема загальну кількість заявок, кількість активних і завершених замовлень, сумарний дохід, кількість клієнтів, майстрів та зареєстрованих пристроїв.

Для аналізу динаміки роботи сервісного центру використовується інтерактивний графік AreaChart «Динаміка заявок», який підтримує перемикання часових діапазонів за день, тиждень або місяць і має градієнтне заповнення області графіка. Розподіл замовлень за типами техніки та статусами подано за допомогою кільцевих PieChart-діаграм із відсотковими підписами та кольоровим кодуванням категорій. Для відображення змін основних показників у часі застосовано LineChart «Тренди по місяцях» із двома осями Y, що дає змогу одночасно аналізувати кількість замовлень і дохід. Додатково в інтерфейсі реалізовано кілька BarChart-діаграм, які відображають найпоширеніші типи ремонтів, розподіл замовлень за брендами та середню вартість ремонту. Окремим елементом дашборду є таблиця продуктивності майстрів, що дозволяє порівнювати працівників за кількістю виконаних замовлень і сумарним доходом.

Компонент візуалізації AI-прогнозу розміщується на сторінці пристрою клієнта та призначений для наочного подання результатів прогнозування. Числова ймовірність поломки відображається за допомогою кругового індикатора Progress із кольоровим кодуванням рівня ризику: зелений колір використовується для низької ймовірності, жовтий – для середнього рівня ризику, а червоний – для високого. Прогнозований тип поломки виводиться через Alert-компонент із відповідним рівнем серйозності. Блок Gemini-аналізу містить природномовне пояснення результатів прогнозування, а також персоналізовані рекомендації щодо профілактичного обслуговування пристрою.

Реалізовано експорт табличних даних у двох форматах засобами бібліотек, що виконуються повністю на стороні клієнта без звернення до сервера.

Для формату Excel використовується бібліотека SheetJS (xlsx). Функція exportToExcel формує робочий аркуш із заголовками та переданим масивом даних, автоматично налаштовує ширину стовпців за максимальним вмістом та ініціює завантаження файлу через бібліотеку file-saver.

Для формату PDF використовується бібліотека jsPDF з плагіном jspdf-autotable. Функція exportToPDF генерує документ в альбомній орієнтації з кольоровим заголовком та чергуванням фонових кольорів рядків таблиці. Для коректного відображення кирилиці реалізовано механізм динамічного підключення шрифту: при першому виклику функція асинхронно завантажує файл TTF-шрифту Roboto, конвертує його в Base64 та реєструє у jsPDF через методи addFileToVFS та addFont. Усі наступні виклики використовують вже кешований шрифт без повторного завантаження.

Реалізована клієнтська частина забезпечує повноцінний інтерфейс для всіх чотирьох ролей системи – від мінімалістичної сторінки відстеження замовлення для гостя до багатофункціонального дашборду адміністратора з інтерактивними графіками та AI-прогнозуванням.

Таким чином, клієнтська частина забезпечує практичну реалізацію основних функціональних можливостей, спроектованих у межах серверної частини та бази даних. Взаємодія з REST API, рольове розмежування доступу, візуалізація аналітичних показників, відображення результатів прогнозування та експорт звітності формують єдиний користувацький простір для роботи із системою. Реалізований інтерфейс поєднує облікові, аналітичні та прогностичні функції, що дозволяє використовувати вебсистему як цілісний інструмент автоматизації діяльності сервісного центру. Це забезпечує логічну завершеність програмної реалізації та демонструє взаємозв'язок між спроектованими вимогами, серверною логікою, базою даних і користувацьким інтерфейсом.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до розділу 2

У другому розділі здійснено повне проєктування та програмну реалізацію вебсистеми прогнозування поломок техніки. Визначено чотири ролі користувачів, сформульовано 19 функціональних та 11 нефункціональних вимог, специфіковано шість ключових варіантів використання із детальним описом основних та альтернативних потоків. Спроектовано реляційну базу даних із 10 таблиць, для кожної з яких наведено структуру полів із типами даних та обмеженнями; описано зв'язки між таблицями та обґрунтовано індексну стратегію для оптимізації аналітичних запитів.

Розроблено математичне забезпечення модуля прогнозування на базі алгоритму Random Forest. Задача декомпозована на три підзадачі: бінарну класифікацію поломок, багатокласову класифікацію типу несправності та регресію ймовірності. Навчено три моделі на синтетичному наборі даних обсягом 800 записів; бінарний класифікатор досяг Accuracy 88.75%, F1-score 0.89 та крос-валідаційного показника 0.885 ± 0.016 . Аналіз важливості ознак показав, що найбільший внесок у прогностичне рішення мають вік пристрою (34.5%) та кількість попередніх ремонтів (20.7%).

Реалізовано REST API з більш ніж 40 ендпоінтами, що охоплюють 8 основних ресурсів системи, із JWT-автентифікацією та RBAC-авторизацією. Впроваджено комплекс засобів безпеки: HTTP-заголовки захисту через Helmet, дворівневе обмеження кількості запитів — загальний ліміт 1000 запитів за 15 хвилин та окремий ліміт 200 запитів за 15 хвилин для маршрутів автентифікації, серверну валідацію вхідних даних через express-validator, а також транзакції бази даних із блокуванням рядків для запобігання стану гонки під час списання запчастин зі складу. Реалізовано клієнтську частину як SPA на React 18 з Ant Design 5, що включає інтерактивний аналітичний дашборд із сімома графічними візуалізаціями, вісім статистичних карток, таблицю продуктивності майстрів, компонент візуалізації AI-прогнозу з кольоровим кодуванням рівня ризику та модуль експорту звітів у форматах Excel і PDF з підтримкою кирилиці.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ ПРОГНОЗУВАННЯ ПОЛОМОК ТЕХНІКИ

3.1 Порядок встановлення та налаштування параметрів системи

Розгортання вебсистеми прогнозування поломок техніки передбачає послідовне виконання чотирьох етапів: підготовку програмного середовища, налаштування бази даних і змінних оточення, встановлення залежностей і навчання ML-моделей, а також запуск та верифікацію всіх компонентів. Діаграму діяльності поетапного процесу розгортання наведено на рисунку 3.1.

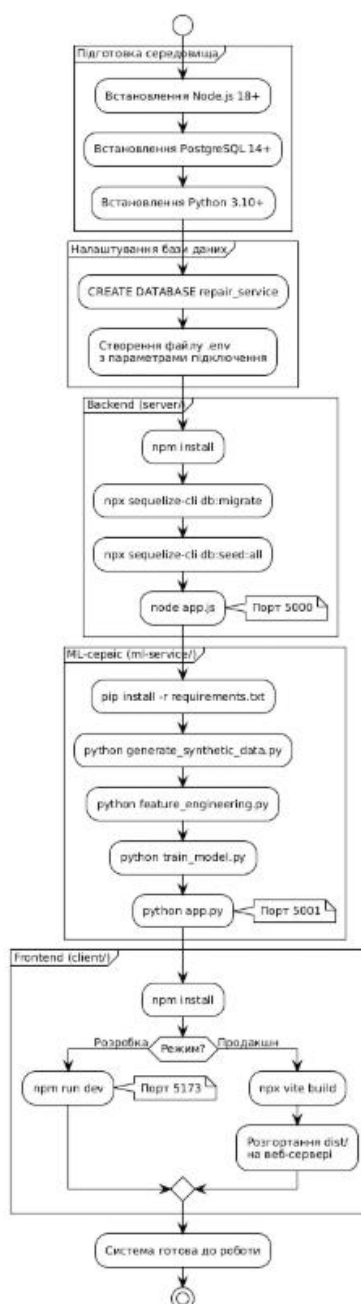


Рис. 3.1 – Діаграма діяльності розгортання системи

Для розгортання системи необхідне попереднє встановлення програмного забезпечення, перелік якого наведено в таблиці 3.1.

Таблиця 3.1

Програмне забезпечення для розгортання системи

№	Компонент	Мінімальна версія	Призначення
1	Node.js	18.0	Середовище виконання серверної та клієнтської частин
2	PostgreSQL	15.0	Система управління реляційною базою даних
3	Python	3.10+	Середовище виконання ML-сервісу прогнозування
4	npm	9.0	Менеджер пакетів Node.js
5	pip	23.0	Менеджер пакетів Python
6	Git	2.30	Система контролю версій для отримання вихідного коду

Після встановлення зазначеного програмного забезпечення необхідно отримати вихідний код репозиторію через Git та перейти до кореневої директорії проєкту. Структуру каталогів верхнього рівня розглянуто в підрозділі 2.1.

На першому етапі в СУБД PostgreSQL створюється порожня база даних з ім'ям `repair_service`, від імені облікового запису з правами адміністратора. Далі в кореневій директорії серверної частини (`server/`) створюється файл змінних оточення `.env`, перелік параметрів якого наведено в таблиці 3.2.

Таблиця 3.2

Параметри файлу конфігурації серверної частини (`.env`)

Параметр	Приклад значення	Призначення
DB_HOST	localhost	Адреса сервера PostgreSQL
DB_PORT	5432	Порт сервера PostgreSQL
DB_NAME	repair_service	Ім'я бази даних
DB_USER	postgres	Обліковий запис PostgreSQL
DB_PASSWORD	-	Пароль облікового запису
JWT_SECRET	-	Секретний ключ підпису access token

JWT_REFRESH_SECRET	-	Секретний ключ підпису refresh token
ML_SERVICE_URL	http://localhost:5001	URL Flask ML-сервісу
GEMINI_API_KEY	-	Ключ API Google Gemini

Значення JWT_SECRET та JWT_REFRESH_SECRET повинні являти собою криптографічно випадкові рядки завдовжки не менше 32 символів.

Ключ GEMINI_API_KEY отримується в Google AI Studio після реєстрації облікового запису розробника. Усі конфіденційні параметри не вносяться до системи контролю версій – файл .env включено до .gitignore.

Встановлення серверних залежностей виконується за допомогою `npm` у директорії `server/`. Після встановлення пакетів запускається інструмент `Sequelize CLI`, який послідовно застосовує міграції (`db:migrate`) та ініціалізує базу початковими даними (`db:seed:all`). Seed-скрипт створює 19 тестових облікових записів (3 адміністратори, 5 майстрів, 11 клієнтів), 27 пристроїв, 62 замовлення з різними статусами, 14 типів ремонту, 18 запчастин та повну хронологію статусів для кожного замовлення.

Встановлення залежностей ML-сервісу виконується за допомогою `pip` у директорії `ml-service/`. Файл `requirements.txt` містить такі пакети: `flask`, `flask-cors`, `scikit-learn`, `pandas`, `numpy`, `joblib`, `python-dotenv` та `pytest`. Перед першим запуском необхідно навчити прогностичні моделі, послідовно виконавши три скрипти в директорії `ml-service/training/`: `generate_synthetic_data.py` (генерація 800 навчальних записів), `feature_engineering.py` (кодування категоріальних ознак та стандартизація числових) та `train_model.py` (навчання трьох моделей `Random Forest` та збереження їх у директорії `model/` у форматі `.pkl`). Встановлення залежностей клієнтської частини аналогічно виконується за допомогою `npm` у директорії `client/`.

Система складається з трьох незалежних процесів, які запускаються паралельно. Серверна частина (`Express.js API`) запускається в директорії `server/` та прослуховує порт 5000. ML-сервіс (`Flask`) запускається в директорії `ml-service/` та прослуховує порт 5001. Клієнтська частина запускається в директорії `client/` у

режимі розробки та стає доступною на порті 5173. Фізичне розміщення компонентів та протоколи взаємодії між ними ілюструє діаграма розгортання, наведена на рисунку 3.2.

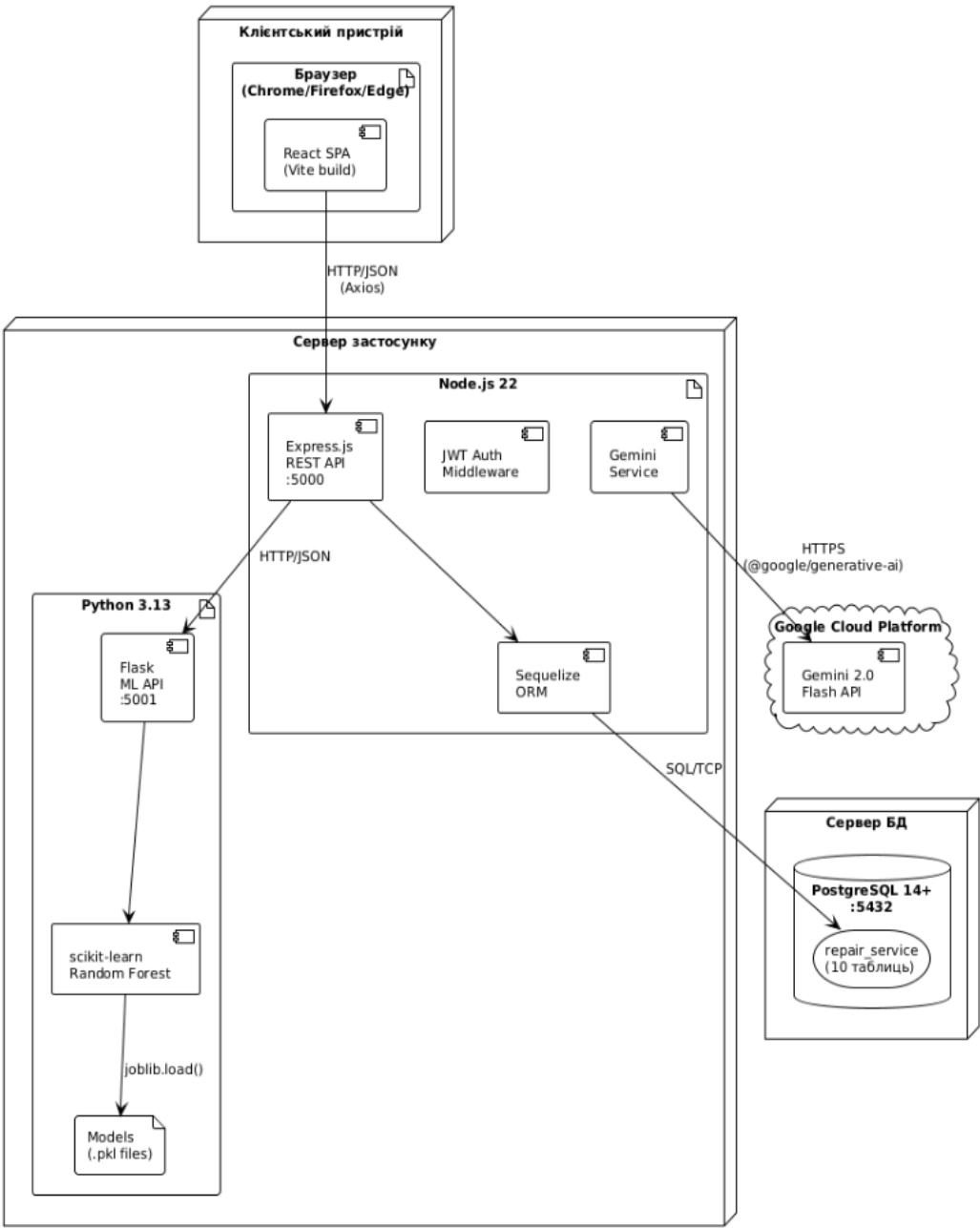


Рис. 3.2 – Діаграма розгортання системи

Верифікацію коректності запуску виконують відправкою трьох тестових HTTP-запитів: GET-запит до /api/auth/me на порт 5000 підтверджує доступність серверного API; GET-запит до /health на порт 5001 підтверджує готовність ML-сервісу до обробки прогнозів; відкриття адреси <http://localhost:5173> у браузері

підтверджує коректне завантаження клієнтського застосунку. Для входу в систему з правами адміністратора використовуються тестові облікові дані, визначені в seed-скриптах та наведені в документації репозиторію.

3.2 Структура інтерфейсу вебсистеми прогнозування поломок техніки

Інтерфейс вебсистеми організовано за принципом єдиного адаптивного макету з бічною навігаційною панеллю та верхнім заголовком. Склад навігаційного меню формується динамічно залежно від ролі автентифікованого користувача, що унеможлиблює відображення пунктів, недоступних за рольовою моделлю.

Основний макет системи (компонент MainLayout) складається з трьох зон: бічна навігаційна панель із динамічним меню, верхній заголовок та робоча область зі змінним вмістом. Верхній заголовок відображає ім'я поточного користувача, значок повідомлень із лічильником непрочитаних і кнопку виходу з системи.

Кольорова схема базується на стандартній палітрі Ant Design 5 із синім відтінком (#1677ff) як основним акцентним кольором. Для позначення статусів замовлень застосовується узгоджена система кольорового кодування: синій — для нових заявок та заявок у роботі, помаранчевий — для діагностики, жовтий — для очікування запчастин, зелений — для готових та сірий — для виданих. Інтерфейс повністю локалізований українською мовою із використанням uk_UA-локалі Ant Design, що охоплює пагінацію, фільтри, вибір дат та повідомлення валідації форм.

Система повідомлень реалізована у вигляді випадаючого списку, що відкривається натисканням на значок у заголовку. Кожне повідомлення містить заголовок, текст та час створення; непрочитані записи виділяються кольором фону. Повідомлення генеруються автоматично при зміні статусу замовлення клієнта та при призначенні майстра.

Сторінка входу побудована за мінімалістичним принципом і центрується по екрану. Вона містить логотип та назву системи «СервісПро», форму з двома

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		71

полями — email та пароль — та посилання на сторінку реєстрації для нових клієнтів. Форма реалізована через компонент Ant Design Form із вбудованою клієнтською валідацією: обов'язковість полів та перевірка формату email. При успішній автентифікації виконується перенаправлення залежно від ролі: адміністратор переходить до /admin/dashboard, майстер — до /master/orders, клієнт — до /client/orders. Зовнішній вигляд сторінки автентифікації наведено на рисунку 3.3.

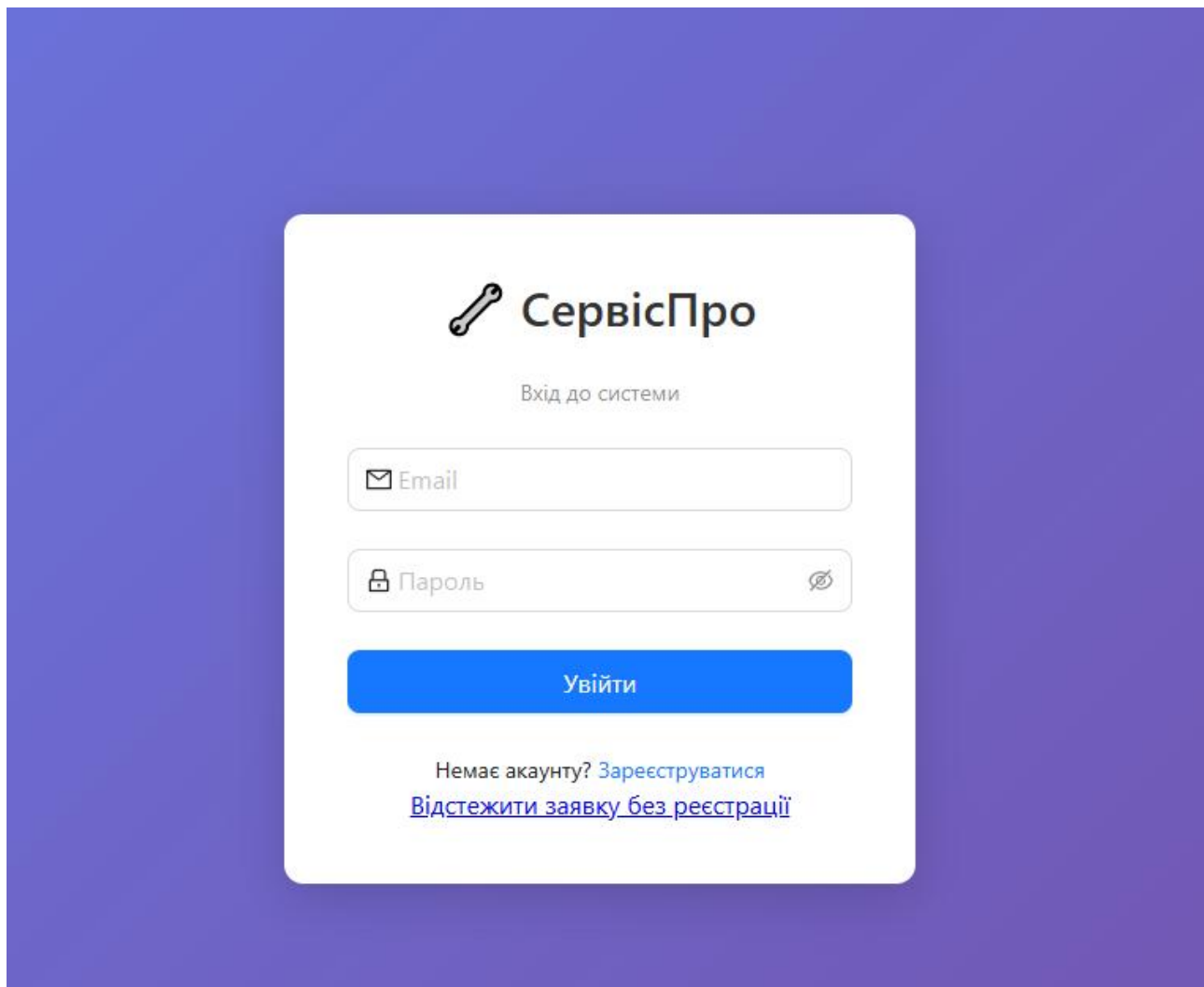


Рис. 3.3 – Сторінка автентифікації системи «СервісПро»

Сторінка реєстрації містить поля імені, прізвища, email, телефону, пароля та підтвердження пароля. Усі нові облікові записи автоматично отримують роль клієнта.

Головна сторінка адміністратора реалізована у вигляді аналітичного дашборду, призначеного для швидкого перегляду основних показників роботи

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

сервісного центру. У верхній частині сторінки розміщено вісім статистичних карток, які відображають загальну кількість заявок, кількість активних і завершених замовлень, сумарний дохід, кількість клієнтів, майстрів, пристроїв та середній час ремонту. Нижче розташовано блок аналітичних візуалізацій, реалізованих за допомогою бібліотеки Recharts. До нього входить графік динаміки заявок із градієнтним заповненням та можливістю групування даних за днями, тижнями або місяцями, кільцева діаграма розподілу замовлень за типами техніки, лінійний графік місячних трендів із двома осями Y для одночасного відображення кількості замовлень і доходу, а також кільцева діаграма розподілу заявок за статусами з кольоровим кодуванням. Додатково дашборд містить горизонтальну стовпчикову діаграму найпоширеніших типів ремонтних робіт, вертикальну діаграму розподілу ремонтів за брендами пристроїв, стовпчикову діаграму середньої вартості ремонту за категоріями техніки та таблицю продуктивності майстрів із сортуванням за кількістю виконаних замовлень і сумарним доходом. Зовнішній вигляд аналітичного дашборду наведено на рисунку 3.4.

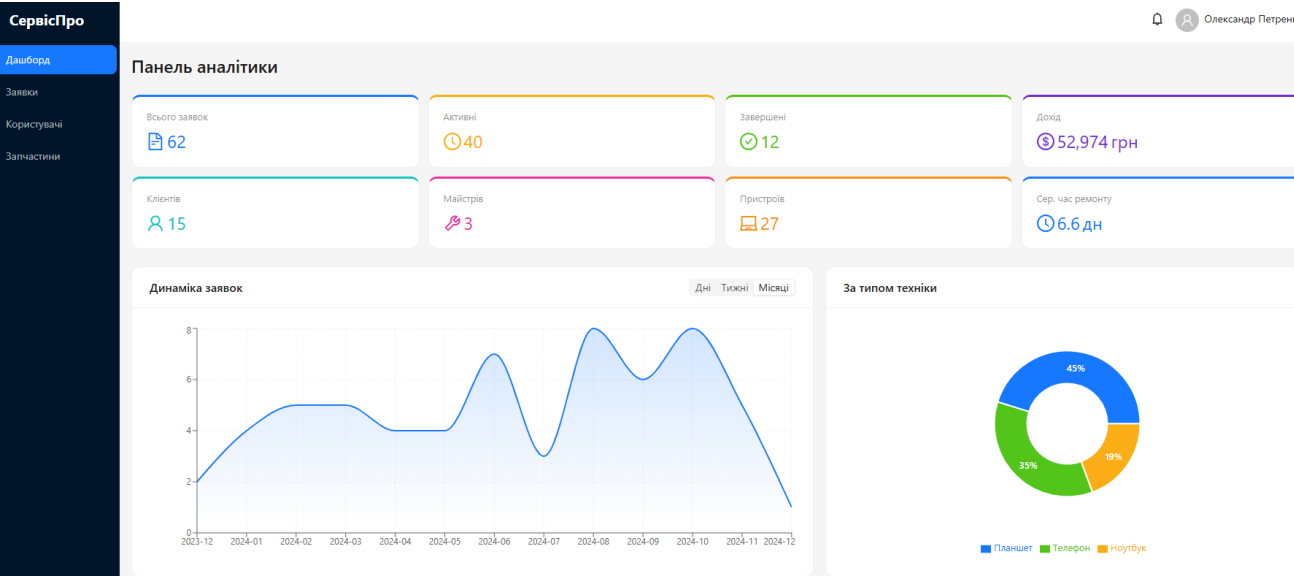


Рис. 3.4 – Аналітичний дашборд адміністратора

Аналітичний дашборд виконує не лише інформаційну, а й управлінську функцію, оскільки дозволяє адміністратору швидко оцінити поточний стан роботи сервісного центру. Наявність графіків і діаграм спрощує виявлення найбільш завантажених періодів, популярних категорій пристроїв, поширених типів

ремонтів і загальної динаміки доходу. Завдяки цьому адміністратор може приймати обґрунтовані рішення щодо розподілу навантаження між майстрами, планування закупівлі запчастин і контролю ефективності роботи сервісного центру.

Сторінка управління замовленнями містить панель фільтрації з випадючими списками для вибору статусу та пошуковим полем. Також на сторінці передбачено кнопки експорту поточної вибірки у форматах Excel і PDF та таблицю із серверною пагінацією. У таблиці відображаються номер замовлення, клієнт, пристрій, призначений майстер, статус у вигляді кольорового тегу, дата створення, вартість і доступні дії. Таке компонування дашборду забезпечує швидкий доступ до найбільш важливих показників без необхідності переходу між окремими сторінками системи. Використання різних типів візуалізацій дозволяє подати статистичні дані у зручній для аналізу формі та спростити порівняння показників за різними напрямками роботи сервісного центру. Зовнішній вигляд сторінки управління замовленнями наведено на рисунку 3.5.

Заявки на ремонт

ExcelPDF

Пошук за номером або описом

Статус

№ Заявки	Пристрій	Клієнт	Майстер	Статус	Вартість	Дата	Дії
SR2412-0015	Планшет Lenovo Tab P11	Юлія Олійник	Не призначено	Нова	0 грн	06.12.2024	
SR2411-0057	Телефон Apple iPhone 12	Катерина Марченко	Іван Коваленко	В роботі	1442 грн	16.11.2024	
SR2411-0016	Планшет Lenovo Tab P11	Юлія Олійник	Дмитро Шевченко	Готово	5053 грн	07.11.2024	
SR2411-0034	Телефон Samsung Galaxy S21	Павло Руденко	Іван Коваленко	Готово	3055 грн	05.11.2024	
SR2411-0044	Планшет Samsung Galaxy Tab A8	Максим Козлов	Іван Коваленко	Очікування запчастин	0 грн	05.11.2024	
SR2411-0006	Планшет Apple iPad Air 5	Тетяна Мельник	Андрій Бондаренко	Готово	3118 грн	02.11.2024	
SR2410-0058	Ноутбук Lenovo ThinkPad X1 Carbon	Катерина Марченко	Іван Коваленко	Діагностика	0 грн	28.10.2024	
SR2410-0023	Телефон Samsung Galaxy S21	Сергій Мороз	Не призначено	Нова	0 грн	24.10.2024	
SR2410-0060	Телефон Samsung Galaxy A54	Ірина Левченко	Іван Коваленко	В роботі	587 грн	22.10.2024	

Рис. 3.5 – Сторінка управління замовленнями (адміністратор)

Детальна сторінка замовлення призначена для перегляду повної інформації про конкретну заявку та виконання основних дій з її обробки. Вона містить блок загальної інформації, елементи зміни статусу із випадającym списком і полем коментаря, блок призначення майстра, таблицю використаних запчастин із можливістю додавання нових позицій, а також таблицю типів виконаних ремонтних робіт. Окремо на сторінці реалізовано хронологію зміни статусів у вигляді компонента Timeline, де відображаються дати та автори кожної зміни. Зовнішній вигляд детальної сторінки замовлення наведено на рисунку 3.6.

← Назад

Заявка SR2411-0057

Статус	В роботі	Клієнт	Катерина Марченко (+380672444443)	Пристрій	Телефон Apple iPhone 12
Майстер	Іван Коваленко	Вартість	1442 грн	Створено	16.11.2024
Опис	Не визначає SIM-карту				
Діагноз	Проведено діагностику. Потрібна заміна компонента				

Управління

Призначити майстра: Іван Коваленко (Телефони Ар... ▾)

Змінити статус: Новий статус ▾ Коментар (необов'язково) ✎ Оновити

Історія статусів

В роботі

20.11.2024, 00:00:00 — Іван Коваленко

Діагностика

17.11.2024, 00:00:00 — Іван Коваленко

Нова

16.11.2024, 00:00:00 — Катерина Марченко

Рис. 3.6 – Детальна сторінка замовлення

Сторінка управління складом запчастин забезпечує перегляд і редагування складських позицій. Дані подано у вигляді таблиці з колонками назви, артикулу SKU, ціни, кількості на складі, мінімального залишку та сумісних пристроїв. Якщо фактична кількість запчастини є меншою за встановлений мінімальний залишок, відповідний рядок виділяється кольором для привернення уваги адміністратора. Створення та редагування запчастин реалізовано через модальне вікно. Зовнішній вигляд сторінки управління запчастинами наведено на рисунку 3.7.

Запчастини							Excel	+ Додати
Пошук запчастин								
Назва	Категорія	Тип техніки	Бренд	Ціна	На складі	Дії		
SSD 256GB NVMe	Накопичувачі	Ноутбук	Vci	1500 грн	8	✎ ✖		
SSD 512GB NVMe	Накопичувачі	Ноутбук	Vci	2200 грн	6	✎ ✖		
Батарея iPhone 13	Батареї	Телефон	Apple	1000 грн	18	✎ ✖		
Батарея iPhone 14	Батареї	Телефон	Apple	1200 грн	15	✎ ✖		
Батарея MacBook Air M2	Батареї	Ноутбук	Apple	3500 грн	3	✎ ✖		
Батарея Samsung Galaxy ...	Батареї	Телефон	Samsung	1100 грн	10	✎ ✖		
Динамік розмовний Sams...	Динаміки	Телефон	Samsung	400 грн	15	✎ ✖		
Екран iPad Air 5	Екрани	Планшет	Apple	5500 грн	2	✎ ✖		
Екран iPhone 13	Екрани	Телефон	Apple	3800 грн	10	✎ ✖		

Рис. 3.7 – Сторінка управління складом запчастин

Сторінка управління користувачами надає адміністратору можливість переглядати список облікових записів, фільтрувати користувачів за роллю, створювати нові облікові записи з визначенням відповідної ролі, редагувати дані користувачів, а також активувати або деактивувати облікові записи. Зовнішній вигляд сторінки управління користувачами наведено на рисунку 3.8.

Олександр

Користувачі

Пошук

Роль

Додати

Ім'я	Email	Телефон	Роль	Спеціалізація	Статус	Дії
Катерина Марченко	client14@test.com	+380672444443	Клієнт	-	Активний	✎ ✖
Артем Ткачук	client10@test.com	+380671999999	Клієнт	-	Активний	✎ ✖
Ірина Левченко	client15@test.com	+380672555554	Клієнт	-	Активний	✎ ✖
Марія Ткаченко	client1@test.com	+380671000000	Клієнт	-	Активний	✎ ✖
Тетяна Мельник	client3@test.com	+380671222222	Клієнт	-	Активний	✎ ✖
Віталій Бойко	client12@test.com	+380672222221	Клієнт	-	Активний	✎ ✖
Наталія Лисенко	client4@test.com	+380671333333	Клієнт	-	Активний	✎ ✖
Анна Поліщук	client13@test.com	+380672333332	Клієнт	-	Активний	✎ ✖
Павло Руденко	client9@test.com	+380671888888	Клієнт	-	Активний	✎ ✖

Рис. 3.8 – Сторінка управління користувачами

Сторінка замовлень майстра відображає виключно замовлення, призначені поточному майстру. Компонент Tabs забезпечує швидке перемикання між групами замовлень за статусами: «Всі», «Нові», «Діагностика», «В роботі», «Очікування запчастин», «Готово». Кожне замовлення відображається у вигляді картки з номером, даними клієнта, пристроєм, статусом та датою. Детальна сторінка замовлення для майстра аналогічна адміністраторській, але без блоку призначення майстра: майстер може змінювати статус, вводити діагноз, додавати використані запчастини та переглядати хронологію. Зовнішній вигляд панелі майстра наведено на рисунку 3.9.

Мої заявки

Фільтр за статусом ▾

№ Заявки	Пристрій	Клієнт	Опис	Статус	Дата	Дії
SR2411-0057	Apple iPhone 12	Катерина Марченко	Не визначає SIM-карту	В роботі	16.11.2024	⋮
SR2411-0034	Samsung Galaxy S21	Павло Руденко	Не працює Wi-Fi модуль	Готово	05.11.2024	⋮
SR2411-0044	Samsung Galaxy Tab A8	Максим Козлов	Повільно працює, зависає	Очікування запчастин	05.11.2024	⋮
SR2410-0058	Lenovo ThinkPad X1 Carbon	Катерина Марченко	Перегрівается при роботі	Діагностика	28.10.2024	⋮
SR2410-0060	Samsung Galaxy A54	Ірина Левченко	Тріщина на екрані після удару	В роботі	22.10.2024	⋮
SR2409-0041	Apple iPhone 14	Максим Козлов	Не вмикається після падіння	Готово	21.09.2024	⋮
SR2409-0018	Apple iPad 10	Сергій Мороз	Не працює камера	В роботі	20.09.2024	⋮
SR2409-0062	Samsung Galaxy A54	Ірина Левченко	Мікрофон не працює під ча...	В роботі	05.09.2024	⋮

Рис. 3.9 – Сторінка списку замовлень майстра

Сторінка замовлень клієнта відображає всі замовлення поточного клієнта з фільтрацією за статусом. Клієнт може переглядати деталі замовлення та хронологію змін статусів, однак не має доступу до функцій зміни статусу чи редагування даних. Зовнішній вигляд сторінки замовлень клієнта наведено на рисунку 3.10.

Мої заявки

№ Заявки	Пристрій	Опис	Статус	Вартість	Дата
SR2407-0001	Lenovo Legion 5	Повільно працює, зависає	Видано	2304 грн	11.07.2024

< 1 >

Рис. 3.10 – Сторінка замовлень клієнта

На сторінці «Мої пристрої» клієнт переглядає список зареєстрованих пристроїв у вигляді карток та може додати новий пристрій через форму реєстрації. Детальна сторінка пристрою є одним із ключових інтерфейсних компонентів системи і складається з трьох блоків. Перший — інформаційний блок із назвою, брендом, моделлю, серійним номером та датою придбання пристрою. Другий — блок AI-прогнозу з кольоровою рамкою (зелена, жовта або червона залежно від рівня ризику), круговим індикатором Progress, що відображає числову ймовірність поломки з динамічним кольором (зелений — менше 40%, жовтий — 40–70%, червоний — понад 70%), тегом рівня ризику, Alert-блоком із назвою найбільш ймовірного типу поломки, текстовим аналізом від Google Gemini та нумерованим списком рекомендацій щодо обслуговування пристрою. Третій — таблиця повної історії ремонтів із номером, статусом, описом, діагнозом, вартістю та датою кожного замовлення. Зовнішній вигляд детальної сторінки пристрою з блоком AI-прогнозу наведено на рисунку 3.11.

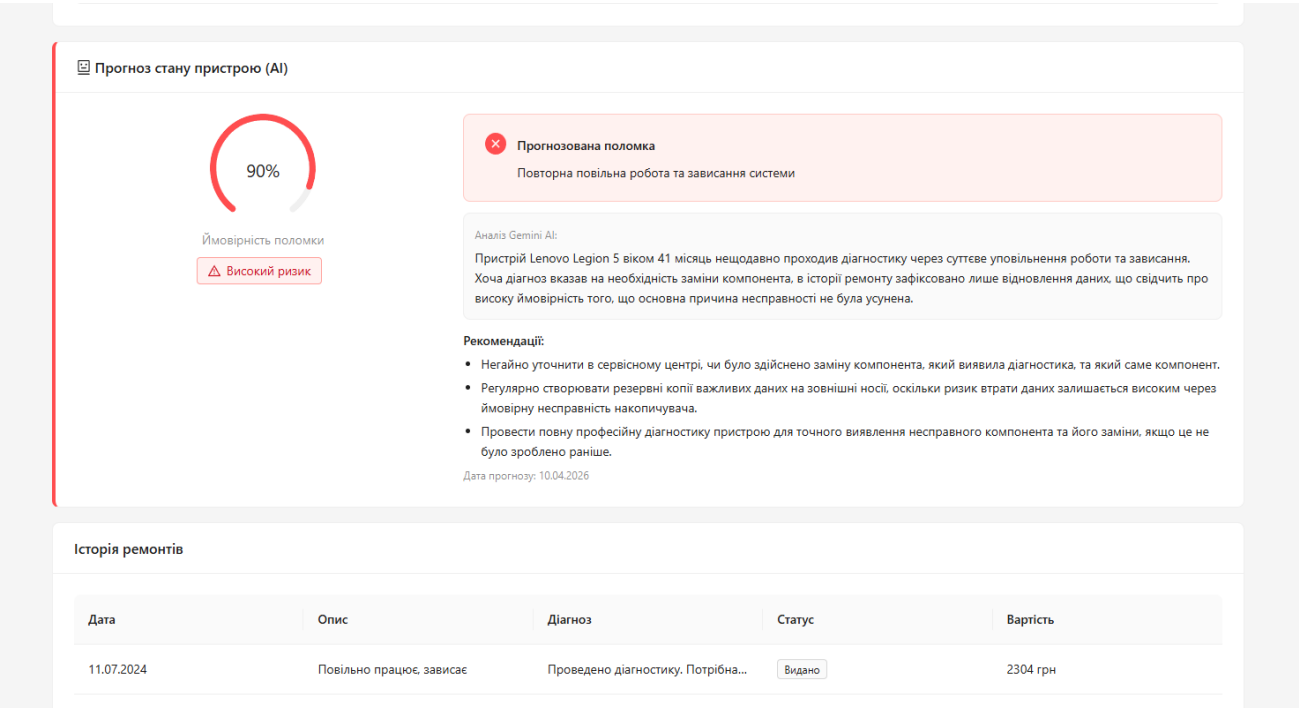


Рис. 3.11 – Детальна сторінка пристрою з візуалізацією AI-прогнозу

Сторінка відстеження замовлення доступна без автентифікації та призначена для клієнтів, що звернулися до сервісу без реєстрації в системі. Вона містить пошукове поле для введення номера замовлення та кнопку «Відстежити». При успішному знаходженні відображаються блок із ключовими даними замовлення

(номер, поточний статус у вигляді кольорового тегу, пристрій, дата створення) та хронологія змін статусів із зазначенням дат і коментарів. Відсутність результату супроводжується інформативним повідомленням із пропозицією перевірити правильність введеного номера. Зовнішній вигляд публічної сторінки відстеження наведено на рисунку 3.12.

 СервісПро

Перевірте статус вашої заявки на ремонт

SR2412-0015

Пошук

Номер заявки	SR2412-0015
Статус	Нова
Пристрій	Lenovo Tab P11
Опис проблеми	Не працює Wi-Fi модуль
Дата створення	06.12.2024

Хід виконання

Нова

06.12.2024, 00:00:00

Заявку створено

Увійти до системи

Рис. 3.12 – Публічна сторінка відстеження замовлення

Реалізований інтерфейс охоплює шість функціональних зон, що відповідають чотирьом ролям системи. Кожна панель надає лише ті елементи керування, що відповідають повноваженням відповідної ролі, що забезпечує узгодженість клієнтського рольового обмеження з серверною RBAC-авторизацією, описаною

в підрозділі 2.4. Єдина кольорова схема, узгоджена система статусів та повна локалізація інтерфейсу українською мовою забезпечують цілісність візуального досвіду для всіх категорій користувачів. Коректність функціонування реалізованого інтерфейсу у взаємодії із серверною частиною та ML-сервісом перевіряється в підрозділі 3.3.

3.3 Тестування роботи програмного комплексу

Тестування системи проводилося на чотирьох рівнях: автоматизоване тестування REST API серверної частини, автоматизоване тестування Flask ML-сервісу, функціональне тестування клієнтського інтерфейсу для всіх ролей та тестування безпеки й продуктивності системи.

Для автоматизованого тестування REST API використано фреймворк Jest у поєднанні з бібліотекою Supertest, що дозволяє виконувати HTTP-запити до Express-застосунку без реального запуску мережевого сервера [28]. Тести організовано у трьох файлах: `auth.test.js`, `orders.test.js` та `devices-predictions.test.js`. Перед запуском тестів виконується ініціалізація окремої тестової бази даних `repair_service_test` із синхронізацією схеми Sequelize та створенням тестового адміністратора; після завершення всіх тестів з'єднання з базою даних закривається.

Тести модуля автентифікації (14 тестів) охоплюють повний цикл роботи: реєстрацію нового користувача з перевіркою наявності токенів та ролі `client` у відповіді, спробу повторної реєстрації з тим самим email (очікується HTTP 409), перевірку валідації формату email та мінімальної довжини пароля (HTTP 400), вхід з коректними та некоректними обліковими даними, оновлення `access token` через `refresh token`, отримання профілю авторизованого користувача та відхилення запитів без токена або з невалідним токеном (HTTP 401).

Тести модуля замовлень (12 тестів) перевіряють: створення замовлення клієнтом із валідними даними (HTTP 201), відхилення запиту без автентифікації (HTTP 401), спробу створення замовлення з неіснуючим ідентифікатором при-

строю (HTTP 404), валідацію обов'язковості поля description (HTTP 400), отримання списку замовлень для клієнта та адміністратора, отримання деталей замовлення за ідентифікатором, зміну статусу адміністратором, відхилення невалідного значення статусу (HTTP 400), заборону зміни статусу клієнтом (HTTP 403), отримання хронології замовлення та публічне відстеження за номером.

Тести модуля пристроїв та прогнозування (11 тестів) охоплюють: створення пристрою клієнтом із валідними даними, валідацію неприпустимого типу пристрою, перевірку обов'язковості поля brand, отримання списку пристроїв та деталей за ідентифікатором, оновлення пристрою, перевірку заборони запуску прогнозу для ролі клієнта (HTTP 403), спробу прогнозу для неіснуючого пристрою (HTTP 404), отримання збережених прогнозів та перевірку ендпоінту /health ML-сервісу. Зведені результати автоматизованого тестування API наведено в таблиці 3.3.

Таблиця 3.3

Зведені результати автоматизованого тестування API

Модуль тестів	Кількість тестів	Пройдено	Статус
Автентифікація (auth.test.js)	14	14	Пройдено
Замовлення (orders.test.js)	12	12	Пройдено
Пристрої та прогнози (devices-predictions.test.js)	11	11	Пройдено
Разом	37	37	100%

Для автоматизованого тестування Flask ML-сервісу використано фреймворк pytest. Тести організовано у файлі tests/test_app.py та розподілено між чотирма класами [29].

Клас TestHealth (2 тести) перевіряє доступність ендпоінту /health та коректність структури відповіді — наявність полів status і models_loaded у JSON-об'єкті.

Клас TestPredict (4 тести) тестує ендпоінт /predict: відхилення запиту без JSON-тіла (HTTP 400), обробку неповного набору ознак із відсутніми обов'язковими полями, перевірку наявності конкретного обов'язкового поля device_type, а

також повну валідацію відповіді при валідному вхідному векторі ознак — перевіряється наявність полів willFailSoon, failureProbability, riskLevel, topFailures та recommendations, а також відповідність failureProbability діапазону та допустимим значенням riskLevel (low, medium, high).

Клас TestModelInfo (1 тест) перевіряє ендпоінт /model-info, що повертає конфігурацію завантажених моделей. Клас TestPredictEdgeCases (3 тести) тестує граничні випадки: пристрій без жодного ремонту (total_repairs = 0, last_repair_type = ""), дуже старий пристрій із великою кількістю ремонтів (age_months = 120, total_repairs = 15) та пристрій із від'ємним значенням віку для перевірки стійкості моделі до некоректних вхідних даних. Зведені результати тестування ML-сервісу наведено в таблиці 3.4.

Таблиця 3.4

Зведені результати автоматизованого тестування ML-сервісу

Клас тестів	Кількість тестів	Статус
TestHealth	2	Пройдено
TestPredict	4	Пройдено
TestModelInfo	1	Пройдено
TestPredictEdgeCases	3	Пройдено
Разом	10	100%

Тестування клієнтської частини виконувалося в режимі ручного тестування для всіх ролей користувачів: адміністратора, майстра, клієнта та гостя. Результати функціонального тестування інтерфейсу наведено в таблиці 3.5.

Таблиця 3.5

Результати функціонального тестування інтерфейсу

Сценарій	Очікуваний результат	Статус
Вхід як адміністратор	Перенаправлення на /admin/dashboard	Пройдено
Вхід як майстер	Перенаправлення на /master/orders	Пройдено
Вхід як клієнт	Перенаправлення на /client/orders	Пройдено
Невірний пароль	Повідомлення про помилку автентифікації	Пройдено
Перегляд аналітичного дашборду	8 карток, 7 графіків та таблиця майстрів	Пройдено
Фільтрація замовлень за статусом	Таблиця оновлюється відповідно до фільтра	Пройдено

Зміна статусу замовлення	Статус оновлено, запис у хронології	Пройдено
Експорт у форматі Excel	Завантаження файлу .xlsx	Пройдено
Експорт у форматі PDF	Завантаження файлу .pdf із кирилицею	Пройдено
Завантаження фото пристрою	Фото відображається на сторінці пристрою	Пройдено
AI-прогноз (клієнт)	Відображення ймовірності, аналізу Gemini та рекомендацій	Пройдено
Відстеження замовлення (гість)	Статус та хронологія відображаються	Пройдено
Перехід на неіснуючий URL	Відображення сторінки 404	Пройдено
Автоматичне оновлення access token	Прозоре оновлення через refresh token	Пройдено
Спроба доступу до чужого розділу	Перенаправлення на власну сторінку ролі	Пройдено
Валідація форм із невірними даними	Відображення повідомлень про помилки	Пройдено

Для оцінювання якості ML-моделей використано метрики, отримані в процесі навчання на тестовій вибірці та крос-валідації. Результати оцінювання наведено в таблиці 3.6.

Таблиця 3.6

Метрики якості навчених ML-моделей

Модель	Метрика	Значення
Класифікатор поломок	Accuracy	0.8875
Класифікатор поломок	F1-score	0.8902
Класифікатор поломок	Precision	0.8750
Класифікатор поломок	Recall	0.9059
Класифікатор поломок	Cross-validation (5-fold)	0.885 ± 0.016
Регресор ймовірності	MAE	0.0557
Регресор ймовірності	RMSE	0.0660
Регресор ймовірності	R ²	0.934

Класифікатор поломок демонструє збалансовані показники Precision та Recall, що свідчить про відсутність значного зміщення в бік хибнопозитивних

або хибнонегативних результатів. Низький розкид крос-валідації (± 0.016) підтверджує стабільність моделі на різних підвибірках даних. Регресор ймовірності має середню абсолютну помилку 5.57 відсоткових пункти, що є прийнятним рівнем точності для задачі прогнозування ймовірності поломки.

Тестування захищеності системи проводилося відповідно до переліку загроз, актуальних для вебзастосунків. Результати тестування безпеки наведено в таблиці 3.7.

Таблиця 3.7

Результати тестування безпеки системи

Тест безпеки	Механізм захисту	Результат
SQL-ін'єкція через параметри запиту	Sequelize ORM (параметризовані запити)	Захищено
XSS через поля форм	React (автоматичне екранування) + Helmet CSP	Захищено
Clickjacking	Helmet (X-Frame-Options: DENY)	Захищено
MIME-sniffing	Helmet (X-Content-Type-Options: nosniff)	Захищено
Brute force автентифікації	Rate limiting (200 запитів / 15 хв) + bcrypt	Захищено
DDoS на API	Rate limiting (1000 запитів / 15 хв на IP)	Захищено
Невалідні вхідні дані	express-validator на всіх POST та PUT маршрутах	Захищено
Запит без токена автентифікації	Middleware authenticate → HTTP 401	Захищено
Прострочений токен автентифікації	jwt.verify → HTTP 401	Захищено
Некоректний тип файлу при завантаженні	Multer fileFilter (лише JPEG, PNG, WebP)	Захищено
Перевищення розміру файлу	Multer limits (5 МБ)	Захищено
Race condition при роботі зі складом	Sequelize transaction + LOCK.UPDATE	Захищено
Витік хешу пароля через API	Поле пароля виключено з усіх API-відповідей	Захищено
Несанкціоновані міждоменні запити	CORS налаштовано для дозволених джерел	Захищено

Збірку клієнтської частини для продуктивного середовища виконано засобами Vite. За результатами збірки загальний розмір бандлу становить 2 502 КБ до стиснення, час збірки — близько 8 секунд, кількість оброблених модулів — 1 496; помилки та попередження компілятора відсутні. Відсутність помилок підтверджує коректність усіх імпортів, відсутність невикористаних залежностей та сумісність версій пакетів.

Висновки до розділу 3

У третьому розділі описано порядок розгортання системи, структуру її інтерфейсу та результати комплексного тестування.

Визначено вимоги до програмного середовища та описано послідовність розгортання трьох компонентів системи: серверної частини (Node.js, Express, PostgreSQL), ML-сервісу (Python, Flask, scikit-learn) та клієнтської частини (React, Vite, Ant Design). Наведено перелік параметрів конфігурації та порядок ініціалізації бази даних, навчання ML-моделей і верифікації коректності запуску. Описано структуру інтерфейсу для чотирьох ролей: адміністратор має доступ до аналітичного дашборду з сімома графіками та панелей управління замовленнями, запчастинами й користувачами; майстер — до призначених замовлень з можливістю зміни статусу та введення діагнозу; клієнт — до власних замовлень та AI-прогнозу поломок із аналізом Google Gemini; гість — до публічної сторінки відстеження замовлення.

Проведено комплексне тестування системи. Автоматизоване тестування REST API охопило 37 тестів (Jest + Supertest) у трьох модулях — усі пройдено успішно; автоматизоване тестування ML-сервісу охопило 10 тестів (pytest) із перевіркою граничних випадків — усі пройдено. Ручне функціональне тестування охопило 16 сценаріїв для всіх ролей — усі пройдено. У межах тестових сценаріїв перевірено основні механізми захисту від SQL-ін'єкцій, XSS, brute force, race condition та несанкціонованого доступу. Класифікатор поломок досяг Ассурасу 88.75% та F1-score 0.89 за стабільного показника крос-валідації 0.885 ± 0.016 ; регресор ймовірності показав MAE 5.57% та R^2 0.934.

ВИСНОВКИ

У випускній кваліфікаційній роботі вирішено актуальне практичне завдання – розроблено вебсистему прогнозування поломок техніки для автоматизації роботи сервісного центру з інтегрованим модулем машинного навчання. Розроблене рішення поєднує функції обліку ремонтних замовлень, управління клієнтами, пристроями та запчастинами, аналітичної звітності й оцінювання ризику майбутніх несправностей.

У процесі виконання роботи було проаналізовано предметну область та розглянуто наявні програмні рішення для автоматизації діяльності сервісних центрів. Аналіз показав, що більшість аналогів орієнтована переважно на облік замовлень, клієнтів і складських залишків, тоді як можливості прогнозування поломок техніки на основі ремонтної історії в них відсутні або суттєво обмежені. Це підтвердило доцільність розробки власної системи, орієнтованої на потреби малих і середніх сервісних центрів.

На основі аналізу потреб користувачів було сформульовано функціональні та нефункціональні вимоги до системи, визначено основні ролі користувачів і обґрунтовано вибір технологічного стеку. Для реалізації програмного комплексу використано Node.js та Express.js для серверної частини, PostgreSQL як систему керування базами даних, Python і Flask для сервісу машинного навчання, а також React і Ant Design для клієнтського застосунку.

У межах роботи розроблено модульну клієнт-серверну архітектуру з виділеним сервісом машинного навчання. Спроектовано реляційну базу даних, реалізовано REST API, механізми JWT-автентифікації, оновлення токенів та рольового розмежування доступу. Серверна частина забезпечує управління користувачами, пристроями, ремонтними замовленнями, запчастинами, прогнозами та сповіщеннями, а також взаємодію з ML-сервісом і Google Gemini API.

Окрему увагу приділено розробці модуля машинного навчання на основі алгоритму Random Forest. У системі реалізовано моделі для визначення типу ймовірної поломки, оцінювання ймовірності відмови пристрою та формування рекомендацій. Для навчання моделей було згенеровано синтетичний набір даних

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

із урахуванням реалістичних характеристик ремонтних звернень. Отримані результати підтверджують працездатність програмної реалізації ML-модуля та можливість його подальшого застосування після навчання на реальних історичних даних.

Клієнтський інтерфейс реалізовано з урахуванням потреб різних категорій користувачів. Адміністратор отримує доступ до аналітичного дашборду, сторінок управління замовленнями, запчастинами та користувачами, а також до експорту звітності у форматах Excel і PDF. Майстер може працювати з призначеними замовленнями, змінювати їхні статуси та переглядати необхідну інформацію для виконання ремонту. Клієнт має можливість переглядати власні пристрої, замовлення та результати AI-прогнозування, а для незареєстрованих користувачів реалізовано публічну сторінку відстеження замовлення за його номером.

Для перевірки працездатності системи проведено комплексне тестування REST API, ML-сервісу, інтерфейсу користувача та механізмів безпеки. У межах тестових сценаріїв перевірено працездатність основних механізмів захисту від SQL-ін'єкцій, XSS, brute force-атак і несанкціонованого доступу.

Практична цінність розробленого рішення полягає в поєднанні традиційних функцій обліку ремонтних замовлень із можливостями предиктивної аналітики, що підвищує інформативність системи та робить її більш корисною для роботи сервісного центру.

Отже, поставлену мету випускної кваліфікаційної роботи досягнуто, а визначені завдання виконано в повному обсязі. Розроблена вебсистема може бути використана як основа для автоматизації діяльності сервісного центру, централізованого обліку ремонтних замовлень і підтримки прийняття рішень щодо прогнозування потенційних поломок техніки. Подальший розвиток системи може передбачати навчання моделей на реальних історичних даних, розширення аналітичного модуля, інтеграцію з платіжними сервісами та впровадження додаткових каналів сповіщення клієнтів.

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

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Автоматизація роботи сервісного центру: ключові інструменти та підходи [Електронний ресурс]. – Режим доступу: <https://usap.online/blog/avtomatyzatsiia-servisnoho-tsentru-z-prohramoiu-usap-online-zruchnist-ta-efektyvnist/> (дата звернення: 15.04.2025).
2. Програмне забезпечення для сервісних центрів [Електронний ресурс]. – Режим доступу: <https://lbs.systems/service-center> (дата звернення: 15.04.2025).
3. Ключові функції програмного забезпечення для ремонтних майстерень [Електронний ресурс]. – Режим доступу: <https://roapp.io/blog/top-features-for-repair-shop-software> (дата звернення: 16.04.2025).
4. RepairDesk Pricing [Електронний ресурс] // RepairDesk. – Режим доступу: <https://www.repairdesk.co/pricing/> (дата звернення: 16.04.2025).
5. Все що потрібно знати про Node.js [Електронний ресурс]. – Режим доступу: <https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs> (дата звернення: 18.03.2025).
6. Створення та запуск проєкту на Express.js [Електронний ресурс]. – Режим доступу: <https://support.servbay.com/uk/nodejs/backend-frameworks/create-and-run-expressjs-project> (дата звернення: 18.04.2025).
7. Password Storage Cheat Sheet [Електронний ресурс] // OWASP Cheat Sheet Series. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 19.04.2025).
8. Gemini 2.5 Flash | Gemini API [Електронний ресурс] // Google AI for Developers. – Режим доступу: <https://ai.google.dev/gemini-api/docs/models/gemini-2.5-flash> (дата звернення: 01.06.2025).
9. React Documentation [Електронний ресурс]. – Режим доступу: <https://react.dev/learn> (дата звернення: 20.04.2026).
10. Recharts Documentation [Електронний ресурс]. – Режим доступу: <https://recharts.github.io/?p=/en-US> (дата звернення: 21.04.2025).

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

11. Python 3.13 Documentation [Електронний ресурс] // Python Software Foundation. – Режим доступу: <https://docs.python.org/3.13/> (дата звернення: 21.04.2025).
12. Flask Documentation [Електронний ресурс] // Pallets Projects. – Режим доступу: <https://flask.palletsprojects.com/en/stable/> (дата звернення: 22.04.2025).
13. Вимоги до апаратного забезпечення для глибокого навчання: комплексний огляд [Електронний ресурс]. – Режим доступу: <https://www.sinsmarts.com/uk/blog/deep-learning-hardware-requirements-a-comprehensive-guide-for-2025> (дата звернення: 25.04.2025).
14. Sandhu R., Ferraiolo D., Kuhn R. The NIST Model for Role-Based Access Control: Towards a Unified Standard [Електронний ресурс] // Proceedings of the 5th ACM Workshop on Role-Based Access Control. Berlin, Germany, 2000. P. 47–63. – Режим доступу: <https://csrc.nist.gov/CSRC/media/Publications/conference-paper/2000/07/26/the-nist-model-for-role-based-access-control-towards-a-unified-documents/sandhu-ferraiolo-kuhn-00.pdf> (дата звернення: 26.04.2025).
15. UML-діаграми: призначення, типи та практика застосування [Електронний ресурс]. – Режим доступу: <https://evergreens.com.ua/ua/articles/uml-diagrams.html> (дата звернення: 27.04.2025).
16. UML-діаграми: що це таке та як їх використовувати [Електронний ресурс]. – Режим доступу: <https://foxminded.ua/uml-diagramy> (дата звернення: 27.04.2025).
17. Sequelize: що це таке та як використовувати ORM для Node.js [Електронний ресурс]. – Режим доступу: <https://foxminded.ua/sequelize-shcho-tse> (дата звернення: 28.04.2025).
18. REST API Tutorial: What is REST? [Електронний ресурс]. – Режим доступу: <https://restfulapi.net/> (дата звернення: 28.03.2025).
19. Breiman L. Random Forests. Machine Learning. 2001. Vol. 45. P. 5–32. DOI: 10.1023/A:1010933404324. URL: <https://link.springer.com/article/10.1023/A:1010933404324> (дата звернення: 05.06.2026).

20. De Morais Pinto R. 5 поширених помилок при реалізації безпеки API [Електронний ресурс] / R. De Morais Pinto. – Режим доступу: <https://www.linkedin.com/pulse/5-common-mistakes-implementing-api-security-rafael-de-morais-pinto-qy1vf> (дата звернення: 03.04.2025).

21. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс] / R. T. Fielding. – Режим доступу: <https://roy.gbiv.com/pubs/dissertation/top.htm> (дата звернення: 04.06.2026).

22. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT) [Електронний ресурс] / M. Jones, J. Bradley, N. Sakimura. – Режим доступу: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 04.06.2026).

23. OWASP API Security Top 10 – 2023 [Електронний ресурс]. – Режим доступу: <https://owasp.org/API-Security/editions/2023/en/0x00-header/> (дата звернення: 04.06.2026).

24. OWASP Cross Site Scripting Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата звернення: 04.06.2026).

25. PostgreSQL Documentation: JSON Types [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/current/datatype-json.html> (дата звернення: 04.06.2026).

26. Sequelize Documentation: Transactions [Електронний ресурс]. – Режим доступу: <https://sequelize.org/docs/v6/other-topics/transactions/> (дата звернення: 04.06.2026).

27. Docker Documentation: Docker Compose [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/compose/> (дата звернення: 04.06.2026).

28. Jest Documentation: Getting Started [Електронний ресурс]. – Режим доступу: <https://jestjs.io/docs/getting-started> (дата звернення: 04.06.2026).

29. pytest Documentation: Get Started [Електронний ресурс]. – Режим доступу: <https://docs.pytest.org/en/stable/getting-started.html> (дата звернення: 04.06.2026).

30. Scikit-learn. Ensembles: Gradient boosting, random forests, bagging, voting, stacking [Електронний ресурс] // scikit-learn documentation. – Режим доступу: <https://scikit-learn.org/stable/modules/ensemble.html> (дата звернення: 15.06.2026).

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

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

require('dotenv').config();
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const path = require('path');
const { sequelize } = require('./models');
const errorHandler = require('./middleware/errorHandler');

const app = express();

app.use(helmet({ crossOriginResourcePolicy: { policy: 'cross-origin' } }));

const apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 1000,
  standardHeaders: true,
  legacyHeaders: false,
  message: { message: 'Забагато запитів, спробуйте пізніше' },
});

const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 200,
  standardHeaders: true,
  legacyHeaders: false,
  message: { message: 'Забагато спроб авторизації, спробуйте пізніше' },
});

app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:5173',
  credentials: true,
}));
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
app.use('/api/', apiLimiter);

app.use('/api/auth', authLimiter, require('./routes/auth'));
app.use('/api/users', require('./routes/users'));
app.use('/api/devices', require('./routes/devices'));
app.use('/api/orders', require('./routes/orders'));
app.use('/api/parts', require('./routes/parts'));
app.use('/api/analytics', require('./routes/analytics'));
app.use('/api/predictions', require('./routes/predictions'));
app.use('/api/notifications', require('./routes/notifications'));

app.get('/api/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

app.use(errorHandler);

const PORT = process.env.PORT || 5000;

const start = async () => {
  try {
    await sequelize.authenticate();
    console.log('Database connected');
    if (process.env.NODE_ENV === 'development') {
      await sequelize.sync({ alter: false });
    }
    app.listen(PORT, () => {
      console.log(`Server running on port ${PORT}`);
    });
  } catch (error) {
    console.error('Failed to start server:', error.message);
    process.exit(1);
  }
};

if (require.main === module) {
  start();
}

module.exports = app;

const axios = require('axios');
const { Device, RepairOrder, Prediction, User, OrderRepair, RepairType }
  = require('./models');
const { analyzeDeviceHistory } = require('./services/geminiService');

const getDeviceRepairHistory = async (deviceId) => {
  const orders = await RepairOrder.findAll({
    where: { deviceId },
    include: [{
      model: OrderRepair, as: 'orderRepairs',
      include: [{ model: RepairType, as: 'repairType' }],
    }],
    order: [['createdAt', 'ASC']],
  });

  return orders.map((o) => ({
    date: o.createdAt.toISOString().split('T')[0],
    description: o.description,
    diagnosis: o.diagnosis,
    repairTypes: o.orderRepairs?.map((r) => r.repairType?.name).join(', ') || '',
    cost: parseFloat(o.totalCost || 0),
    status: o.status,
  }));
};

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		94

```

exports.predictML = async (req, res, next) => {
  try {
    const device = await Device.findByPk(req.params.deviceId, {
      include: [{ model: User, as: 'client' }],
    });
    if (!device)
      return res.status(404).json({ message: 'Пристрій не знайдено' });

    const repairHistory = await getDeviceRepairHistory(device.id);
    const ageMonths = device.purchaseDate
      ? Math.floor((Date.now() - new Date(device.purchaseDate).getTime())
        / (1000 * 60 * 60 * 24 * 30))
      : 24;

    const currentMonth = new Date().getMonth() + 1;
    const season = currentMonth <= 2 || currentMonth === 12 ? 'winter'
      : currentMonth <= 5 ? 'spring'
      : currentMonth <= 8 ? 'summer' : 'autumn';

    const features = {
      device_type: device.deviceType,
      brand: device.brand,
      model: device.model,
      age_months: ageMonths,
      total_repairs: repairHistory.length,
      months_since_last_repair: repairHistory.length > 0
        ? Math.floor((Date.now()
          - new Date(repairHistory[repairHistory.length - 1].date).getTime())
          / (1000 * 60 * 60 * 24 * 30))
        : ageMonths,
      total_cost: repairHistory.reduce((sum, r) => sum + r.cost, 0),
      last_repair_type: repairHistory.length > 0
        ? repairHistory[repairHistory.length - 1].repairTypes : '',
      season,
    };

    let mlResult = null;
    try {
      const mlResponse = await axios.post(
        `${process.env.ML_SERVICE_URL}/predict`, features, { timeout: 10000 }
      );
      mlResult = mlResponse.data;
    } catch (err) {
      console.log('ML service unavailable, using Gemini only');
    }

    const prediction = await Prediction.create({
      deviceId: device.id,
      predictedFailureType: mlResult?.predicted_failure || null,
      probability: mlResult?.probability || null,
      predictedDate: mlResult?.predicted_date || null,
      modelVersion: mlResult?.model_version || null,
    });
  }
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    source: 'ml',
  });

  res.json({ prediction, mlResult });
} catch (error) {
  next(error);
}
};

exports.predictCombined = async (req, res, next) => {
  try {
    const device = await Device.findByPk(req.params.deviceId);
    if (!device)
      return res.status(404).json({ message: 'Пристрій не знайдено' });

    const repairHistory = await getDeviceRepairHistory(device.id);
    const ageMonths = device.purchaseDate
      ? Math.floor((Date.now() - new Date(device.purchaseDate).getTime())
        / (1000 * 60 * 60 * 24 * 30))
      : 24;

    let mlResult = null;
    try {
      const currentMonth = new Date().getMonth() + 1;
      const season = currentMonth <= 2 || currentMonth === 12 ? 'winter'
        : currentMonth <= 5 ? 'spring'
        : currentMonth <= 8 ? 'summer' : 'autumn';

      const features = {
        device_type: device.deviceType,
        brand: device.brand,
        model: device.model,
        age_months: ageMonths,
        total_repairs: repairHistory.length,
        months_since_last_repair: repairHistory.length > 0
          ? Math.floor((Date.now()
            - new Date(repairHistory[repairHistory.length - 1].date)
              .getTime()) / (1000 * 60 * 60 * 24 * 30))
          : ageMonths,
        total_cost: repairHistory.reduce((sum, r) => sum + r.cost, 0),
        last_repair_type: repairHistory.length > 0
          ? repairHistory[repairHistory.length - 1].repairTypes : '',
        season,
      };
    }
    const mlResponse = await axios.post(
      `${process.env.ML_SERVICE_URL}/predict`, features, { timeout: 10000 }
    );
    mlResult = mlResponse.data;
  } catch (err) {
    console.log('ML service unavailable');
  }
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		96

```

const geminiResult = await analyzeDeviceHistory(
  { ...device.toJSON(), ageMonths }, repairHistory
);
const topPrediction = geminiResult.predictedFailures?.[0];

const prediction = await Prediction.create({
  deviceId: device.id,
  predictedFailureType: mlResult?.predicted_failure
    || topPrediction?.type || null,
  probability: mlResult?.probability
    || topPrediction?.probability || null,
  predictedDate: mlResult?.predicted_date || null,
  modelVersion: mlResult?.model_version || null,
  source: 'combined',
  geminiAnalysis: geminiResult.analysis,
  recommendations: JSON.stringify({
    recommendations: geminiResult.recommendations,
    predictedFailures: geminiResult.predictedFailures,
    riskLevel: geminiResult.riskLevel,
    mlPrediction: mlResult,
  }),
});

res.json({ prediction, mlResult, geminiAnalysis: geminiResult });
} catch (error) {
  next(error);
}
};
const { GoogleGenerativeAI } = require('@google/generative-ai');

let genAI = null;

const getGenAI = () => {
  if (!genAI && process.env.GEMINI_API_KEY
    && process.env.GEMINI_API_KEY !== 'your-gemini-api-key-here') {
    genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
  }
  return genAI;
};

const analyzeDeviceHistory = async (deviceInfo, repairHistory) => {
  const ai = getGenAI();
  if (!ai) {
    return {
      analysis: 'Gemini API не налаштовано.',
      recommendations: [],
      predictedFailures: [],
    };
  }

  const model = ai.getGenerativeModel({ model: 'gemini-2.5-flash' });

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						97
Змн.	Арк.	№ докум.	Підпис	Дата		

```
const prompt = `Ти — експерт з ремонту електроніки. Проаналізуй історію ремонтів пристрою та дай прогноз можливих поломок.
```

ПРИСТРІЙ:

- Тип: \${deviceInfo.deviceType}
- Бренд: \${deviceInfo.brand}
- Модель: \${deviceInfo.model}
- Дата покупки: \${deviceInfo.purchaseDate || 'невідомо'}
- Вік пристрою: \${deviceInfo.ageMonths} місяців

ІСТОРИЯ РЕМОНТІВ (\${repairHistory.length} записів):

```
${repairHistory.map((r, i) =>` ${i + 1}. Дата: ${r.date}, Проблема: ${r.description},  
Діагноз: ${r.diagnosis || 'N/A'},  
Тип ремонту: ${r.repairTypes || 'N/A'},  
Вартість: ${r.cost} грн`).join("\n`)}
```

Дай відповідь СТРОГО у форматі JSON:

```
{  
  "analysis": "Загальний аналіз стану пристрою (2-3 речення)",  
  "predictedFailures": [{  
    "type": "Тип можливої поломки",  
    "probability": 0.75,  
    "timeframe": "протягом 3 місяців",  
    "reason": "Чому це ймовірно"  
  }],  
  "recommendations": ["Рекомендація 1", "Рекомендація 2"],  
  "riskLevel": "low|medium|high"  
};
```

```
try {  
  const result = await model.generateContent(prompt);  
  const responseText = result.response.text();  
  let jsonStr = responseText;  
  const jsonMatch = responseText.match(/^``(?:json)?\s*([^\s\S]*?)``/);  
  if (jsonMatch) jsonStr = jsonMatch[1].trim();  
  return JSON.parse(jsonStr);  
} catch (error) {  
  console.error('Gemini API error:', error.message);  
  return {  
    analysis: 'Не вдалося отримати аналіз від Gemini AI.',  
    recommendations: ['Зверніться до майстра для діагностики'],  
    predictedFailures: [],  
    riskLevel: 'medium',  
  };  
}  
};
```

```
module.exports = { analyzeDeviceHistory };
```

```
const { body, param, validationResult } = require('express-validator');
```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		98

```

const validate = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({
      message: 'Помилка валідації',
      errors: errors.array().map(e => e.msg),
    });
  }
  next();
};

const registerValidator = [
  body('email').isEmail().withMessage('Невірний формат email')
    .normalizeEmail(),
  body('password').isLength({ min: 6 })
    .withMessage('Пароль має бути мінімум 6 символів'),
  body('firstName').trim().notEmpty()
    .withMessage('Ім'я обов'язкове').isLength({ max: 100 }),
  body('lastName').trim().notEmpty()
    .withMessage('Прізвище обов'язкове').isLength({ max: 100 }),
  body('phone').optional().trim().isLength({ max: 20 }),
  validate,
];

const loginValidator = [
  body('email').isEmail().withMessage('Невірний формат email')
    .normalizeEmail(),
  body('password').notEmpty().withMessage('Пароль обов'язковий'),
  validate,
];

const createDeviceValidator = [
  body('deviceType').isIn(['phone', 'laptop', 'tablet', 'other'])
    .withMessage('Невірний тип пристрою'),
  body('brand').trim().notEmpty()
    .withMessage('Бренд обов'язковий').isLength({ max: 100 }),
  body('model').trim().notEmpty()
    .withMessage('Модель обов'язкова').isLength({ max: 100 }),
  body('serialNumber').optional().trim().isLength({ max: 200 }),
  body('purchaseDate').optional().isISO8601()
    .withMessage('Невірний формат дати'),
  validate,
];

const createOrderValidator = [
  body('deviceId').isInt({ min: 1 })
    .withMessage('deviceId обов'язковий'),
  body('description').trim().notEmpty()
    .withMessage('Опис обов'язковий').isLength({ max: 2000 }),
  body('deadline').optional().isISO8601()
    .withMessage('Невірний формат дати'),
  validate,
];

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

];

const updateStatusValidator = [
  param('id').isInt({ min: 1 }),
  body('status').isIn([
    'new', 'diagnostics', 'in_progress',
    'waiting_parts', 'ready', 'issued',
  ]).withMessage('Невірний статус'),
  body('comment').optional().trim().isLength({ max: 1000 }),
  validate,
];

const addPartValidator = [
  param('id').isInt({ min: 1 }),
  body('partId').isInt({ min: 1 }).withMessage('partId обов\'язковий'),
  body('quantity').isInt({ min: 1 })
    .withMessage('Кількість має бути >= 1'),
  validate,
];

const predictValidator = [
  param('deviceId').isInt({ min: 1 })
    .withMessage('Невірний deviceId'),
  validate,
];

const idParamValidator = [
  param('id').isInt({ min: 1 }).withMessage('Невірний ID'),
  validate,
];

module.exports = {
  registerValidator, loginValidator, createDeviceValidator,
  createOrderValidator, updateStatusValidator, addPartValidator,
  predictValidator, idParamValidator,
};

const { Sequelize } = require('sequelize');
const config = require('../config/database');

const env = process.env.NODE_ENV || 'development';
const dbConfig = config[env];

const sequelize = new Sequelize(
  dbConfig.database, dbConfig.username, dbConfig.password, {
    host: dbConfig.host,
    port: dbConfig.port,
    dialect: dbConfig.dialect,
    logging: dbConfig.logging,
    pool: { max: 10, min: 0, acquire: 30000, idle: 10000 },
  }
);

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const db = {};
db.sequelize = sequelize;
db.Sequelize = Sequelize;

db.User = require('./User')(sequelize, Sequelize);
db.Device = require('./Device')(sequelize, Sequelize);
db.RepairOrder = require('./RepairOrder')(sequelize, Sequelize);
db.OrderStatusHistory = require('./OrderStatusHistory')(sequelize, Sequelize);
db.Part = require('./Part')(sequelize, Sequelize);
db.OrderPart = require('./OrderPart')(sequelize, Sequelize);
db.RepairType = require('./RepairType')(sequelize, Sequelize);
db.OrderRepair = require('./OrderRepair')(sequelize, Sequelize);
db.Prediction = require('./Prediction')(sequelize, Sequelize);
db.Notification = require('./Notification')(sequelize, Sequelize);

db.User.hasMany(db.Device, { foreignKey: 'clientId', as: 'devices' });
db.Device.belongsTo(db.User, { foreignKey: 'clientId', as: 'client' });

db.Device.hasMany(db.RepairOrder, { foreignKey: 'deviceId', as: 'orders' });
db.RepairOrder.belongsTo(db.Device, { foreignKey: 'deviceId', as: 'device' });

db.User.hasMany(db.RepairOrder, { foreignKey: 'clientId', as: 'clientOrders' });
db.RepairOrder.belongsTo(db.User, { foreignKey: 'clientId', as: 'client' });

db.User.hasMany(db.RepairOrder, { foreignKey: 'masterId', as: 'masterOrders' });
db.RepairOrder.belongsTo(db.User, { foreignKey: 'masterId', as: 'master' });

db.RepairOrder.hasMany(db.OrderStatusHistory, {
  foreignKey: 'orderId', as: 'statusHistory',
});
db.OrderStatusHistory.belongsTo(db.User, {
  foreignKey: 'changedBy', as: 'changedByUser',
});

db.RepairOrder.belongsToMany(db.Part, {
  through: db.OrderPart, foreignKey: 'orderId', as: 'parts',
});
db.Part.belongsToMany(db.RepairOrder, {
  through: db.OrderPart, foreignKey: 'partId', as: 'orders',
});

db.RepairOrder.belongsToMany(db.RepairType, {
  through: db.OrderRepair, foreignKey: 'orderId', as: 'repairTypes',
});
db.RepairType.belongsToMany(db.RepairOrder, {
  through: db.OrderRepair, foreignKey: 'repairTypeId', as: 'orders',
});

module.exports = db;
'use strict';

module.exports = {

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

async up(queryInterface, Sequelize) {
  await queryInterface.createTable('users', {
    id: { type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true },
    email: { type: Sequelize.STRING, allowNull: false, unique: true },
    password_hash: { type: Sequelize.STRING, allowNull: false },
    role: { type: Sequelize.ENUM('admin', 'master', 'client'),
      allowNull: false, defaultValue: 'client' },
    first_name: { type: Sequelize.STRING, allowNull: false },
    last_name: { type: Sequelize.STRING, allowNull: false },
    phone: { type: Sequelize.STRING, allowNull: true },
    specialization: { type: Sequelize.STRING, allowNull: true },
    is_active: { type: Sequelize.BOOLEAN, defaultValue: true },
    created_at: { type: Sequelize.DATE, allowNull: false,
      defaultValue: Sequelize.literal('NOW()') },
    updated_at: { type: Sequelize.DATE, allowNull: false,
      defaultValue: Sequelize.literal('NOW()') },
  });

```

```

await queryInterface.createTable('devices', {
  id: { type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true },
  client_id: { type: Sequelize.INTEGER, allowNull: false,
    references: { model: 'users', key: 'id' }, onDelete: 'CASCADE' },
  device_type: { type: Sequelize.ENUM('phone', 'laptop', 'tablet',
    'other'), allowNull: false },
  brand: { type: Sequelize.STRING, allowNull: false },
  model: { type: Sequelize.STRING, allowNull: false },
  serial_number: { type: Sequelize.STRING, allowNull: true },
  purchase_date: { type: Sequelize.DATEONLY, allowNull: true },
  created_at: { type: Sequelize.DATE, allowNull: false,
    defaultValue: Sequelize.literal('NOW()') },
  updated_at: { type: Sequelize.DATE, allowNull: false,
    defaultValue: Sequelize.literal('NOW()') },
});

```

// Таблиці repair_types, parts, repair_orders, order_status_history,
 // order_parts, order_repairs, predictions, notifications —
 // аналогічна структура відповідно до схеми бази даних

```

await queryInterface.addIndex('devices', ['client_id']);
await queryInterface.addIndex('repair_orders', ['client_id']);
await queryInterface.addIndex('repair_orders', ['master_id']);
await queryInterface.addIndex('repair_orders', ['status']);
await queryInterface.addIndex('repair_orders', ['order_number']);
await queryInterface.addIndex('order_status_history', ['order_id']);
await queryInterface.addIndex('predictions', ['device_id']);
await queryInterface.addIndex('notifications', ['user_id', 'is_read']);
},

```

```

async down(queryInterface) {
  await queryInterface.dropTable('notifications');
  await queryInterface.dropTable('predictions');
  await queryInterface.dropTable('order_repairs');
}

```

```

await queryInterface.dropTable('order_parts');
await queryInterface.dropTable('order_status_history');
await queryInterface.dropTable('repair_orders');
await queryInterface.dropTable('parts');
await queryInterface.dropTable('repair_types');
await queryInterface.dropTable('devices');
await queryInterface.dropTable('users');
},
};

```

Додаток Б. Лістинг програмного коду ML-сервісу

```

import os
import joblib
import numpy as np
from flask import Flask, request, jsonify
from flask_cors import CORS
from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)
CORS(app)

MODEL_DIR = os.path.join(os.path.dirname(__file__), 'model')

failure_clf = None
failure_type_clf = None
probability_reg = None
encoders = None
scaler = None
failure_encoder = None

FAILURE_LABELS_UA = {
    'screen_damage': 'Пошкодження екрану',
    'battery_degradation': 'Деградація батареї',
    'charging_port': 'Проблеми з роз'ємом зарядки',
    'motherboard_failure': 'Поломка материнської плати',
    'camera_failure': 'Поломка камери',
    'keyboard_failure': 'Поломка клавіатури',
    'overheating': 'Перегрів',
    'speaker_failure': 'Несправність динаміка',
    'data_recovery': 'Втрата даних',
    'water_damage': 'Пошкодження рідиною',
    'software_issue': 'Програмна проблема',
    'touchscreen_failure': 'Поломка тачскріну',
}

def load_models():

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

global failure_clf, failure_type_clf, probability_reg
global encoders, scaler, failure_encoder
try:
    failure_clf = joblib.load(
        os.path.join(MODEL_DIR, 'failure_classifier.pkl'))
    failure_type_clf = joblib.load(
        os.path.join(MODEL_DIR, 'failure_type_classifier.pkl'))
    probability_reg = joblib.load(
        os.path.join(MODEL_DIR, 'probability_regressor.pkl'))
    encoders = joblib.load(os.path.join(MODEL_DIR, 'encoders.pkl'))
    scaler = joblib.load(os.path.join(MODEL_DIR, 'scaler.pkl'))
    failure_encoder = joblib.load(
        os.path.join(MODEL_DIR, 'failure_encoder.pkl'))
    print('Models loaded successfully')
except Exception as e:
    print(f'Warning: Could not load models: {e}')

```

```

@app.route('/health', methods=['GET'])
def health():
    models_loaded = all([failure_clf, failure_type_clf, probability_reg])
    return jsonify({'status': 'ok', 'models_loaded': models_loaded})

```

```

@app.route('/predict', methods=['POST'])
def predict():
    if not all([failure_clf, failure_type_clf, probability_reg]):
        return jsonify({'error': 'Models not loaded.'}), 503

```

```

data = request.get_json()
if not data:
    return jsonify({'error': 'No data provided'}), 400

```

```

required = ['device_type', 'brand', 'age_months', 'total_repairs',
            'months_since_last_repair', 'total_cost',
            'last_repair_type', 'season']
missing = [f for f in required if f not in data]
if missing:
    return jsonify({
        'error': f'Missing fields: {", ".join(missing)}'}, 400

```

```

try:
    X = prepare_input(data)

    will_fail = int(failure_clf.predict(X)[0])
    fail_proba = float(failure_clf.predict_proba(X)[0][1])

    ft_pred = failure_type_clf.predict(X)[0]
    ft_proba = failure_type_clf.predict_proba(X)[0]
    failure_type = failure_encoder.inverse_transform([ft_pred])[0]

    top_indices = np.argsort(ft_proba)[::-1][:3]

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

top_failures = []
for idx in top_indices:
    ft_name = failure_encoder.inverse_transform([idx])[0]
    top_failures.append({
        'type': ft_name,
        'label': FAILURE_LABELS_UA.get(ft_name, ft_name),
        'probability': round(float(ft_proba[idx]), 3),
    })

prob = float(probability_reg.predict(X)[0])
prob = max(0.0, min(1.0, prob))

if prob > 0.7:
    risk_level = 'high'
elif prob > 0.4:
    risk_level = 'medium'
else:
    risk_level = 'low'

recommendations = generate_recommendations(
    data, failure_type, prob, risk_level)

return jsonify({
    'prediction': {
        'willFailSoon': bool(will_fail),
        'failureProbability': round(prob, 3),
        'riskLevel': risk_level,
        'predictedFailureType': failure_type,
        'predictedFailureLabel': FAILURE_LABELS_UA.get(
            failure_type, failure_type),
        'topFailures': top_failures,
        'recommendations': recommendations,
    }
})
except Exception as e:
    return jsonify({'error': str(e)}), 500

def generate_recommendations(data, failure_type, prob, risk_level):
    recs = []
    age = data.get('age_months', 0)
    total_repairs = data.get('total_repairs', 0)

    if risk_level == 'high':
        recs.append('Рекомендується профілактичний огляд якнайшвидше')
    if failure_type == 'battery_degradation':
        recs.append('Рекомендується перевірка стану батареї')
    elif failure_type == 'screen_damage':
        recs.append('Рекомендується використувати захисне скло')
    elif failure_type == 'overheating':
        recs.append('Рекомендується очищення системи охолодження')
    elif failure_type == 'charging_port':

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    recs.append('Рекомендується перевірка роз'єму зарядки')
if age > 36:
    recs.append(f'Пристрою {age} міс. — розгляньте оновлення')
if total_repairs >= 3:
    recs.append(f'Ремонтувався {total_repairs} разів — підвищений ризик')
if risk_level == 'low':
    recs.append('Стан пристрою задовільний')
return recs

```

```

@app.route('/model-info', methods=['GET'])
def model_info():
    if not failure_clf:
        return jsonify({'error': 'Models not loaded'}), 503
    return jsonify({
        'models': {
            'failureClassifier': {
                'type': 'RandomForestClassifier',
                'n_estimators': failure_clf.n_estimators,
                'max_depth': failure_clf.max_depth,
            },
            'failureTypeClassifier': {
                'type': 'RandomForestClassifier',
                'n_estimators': failure_type_clf.n_estimators,
            },
            'probabilityRegressor': {
                'type': 'RandomForestRegressor',
                'n_estimators': probability_reg.n_estimators,
            },
        },
        'failureTypes': list(failure_encoder.classes_),
    })

```

```

if __name__ == '__main__':
    load_models()
    port = int(os.environ.get('ML_SERVICE_PORT', 5001))
    print(f'ML Service running on port {port}')
    app.run(host='0.0.0.0', port=port, debug=True)

```

```

import os
import sys
import joblib
import numpy as np
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    classification_report, mean_absolute_error, mean_squared_error
)
from generate_synthetic_data import generate_synthetic_data
from feature_engineering import prepare_training_data, save_artifacts

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						106
Змн.	Арк.	№ докум.	Підпис	Дата		

```

def train_models(csv_path=None, model_dir='../model'):
    if csv_path is None:
        print('Generating synthetic data...')
        df = generate_synthetic_data(800)
        csv_path = 'synthetic_repair_data.csv'
        df.to_csv(csv_path, index=False)
        print(f'Saved {len(df)} records to {csv_path}')

    print('Preparing features...')
    X, y_class, y_failure_type, y_prob, encoders, scaler, \
        failure_encoder = prepare_training_data(csv_path)

    X_train, X_test, y_class_train, y_class_test = \
        train_test_split(X, y_class, test_size=0.2, random_state=42)
    _, _, y_ft_train, y_ft_test = \
        train_test_split(X, y_failure_type, test_size=0.2, random_state=42)
    _, _, y_prob_train, y_prob_test = \
        train_test_split(X, y_prob, test_size=0.2, random_state=42)

    print("\n--- Training Failure Classifier ---")
    clf = RandomForestClassifier(
        n_estimators=100, max_depth=10, random_state=42)
    clf.fit(X_train, y_class_train)
    y_pred = clf.predict(X_test)
    print(f'Accuracy: {accuracy_score(y_class_test, y_pred):.4f}')
    print(f'Precision: {precision_score(y_class_test, y_pred):.4f}')
    print(f'Recall: {recall_score(y_class_test, y_pred):.4f}')
    print(f'F1: {f1_score(y_class_test, y_pred):.4f}')
    cv = cross_val_score(clf, X, y_class, cv=5)
    print(f'CV Accuracy: {cv.mean():.4f} (+/- {cv.std():.4f})')

    print("\n--- Training Failure Type Classifier ---")
    ft_clf = RandomForestClassifier(
        n_estimators=100, max_depth=12, random_state=42)
    ft_clf.fit(X_train, y_ft_train)
    y_pred_ft = ft_clf.predict(X_test)
    print(f'Accuracy: {accuracy_score(y_ft_test, y_pred_ft):.4f}')
    print(classification_report(
        y_ft_test, y_pred_ft,
        target_names=failure_encoder.classes_, zero_division=0))

    print("\n--- Training Probability Regressor ---")
    reg = RandomForestRegressor(
        n_estimators=100, max_depth=10, random_state=42)
    reg.fit(X_train, y_prob_train)
    y_pred_prob = reg.predict(X_test)
    print(f'MAE: {mean_absolute_error(y_prob_test, y_pred_prob):.4f}')
    print(f'RMSE: {np.sqrt(mean_squared_error(y_prob_test, y_pred_prob)):.4f}')

    os.makedirs(model_dir, exist_ok=True)

```

```

joblib.dump(clf, os.path.join(model_dir, 'failure_classifier.pkl'))
joblib.dump(ft_clf, os.path.join(model_dir, 'failure_type_classifier.pkl'))
joblib.dump(reg, os.path.join(model_dir, 'probability_regressor.pkl'))
save_artifacts(encoders, scaler, failure_encoder, model_dir)
print(f'\nModels saved to {model_dir}/')

if __name__ == '__main__':
    csv = sys.argv[1] if len(sys.argv) > 1 else None
    train_models(csv)

import numpy as np
import pandas as pd
import random

random.seed(42)
np.random.seed(42)

DEVICE_TYPES = ['phone', 'laptop', 'tablet']
BRANDS = {
    'phone': ['Apple', 'Samsung', 'Xiaomi'],
    'laptop': ['Apple', 'Lenovo', 'HP', 'Dell', 'ASUS'],
    'tablet': ['Apple', 'Samsung', 'Lenovo'],
}

FAILURE_TYPES = [
    'screen_damage', 'battery_degradation', 'charging_port',
    'motherboard_failure', 'camera_failure', 'keyboard_failure',
    'overheating', 'speaker_failure', 'data_recovery',
    'water_damage', 'software_issue', 'touchscreen_failure',
]

FAILURE_WEIGHTS = {
    'phone': {
        'screen_damage': 0.25, 'battery_degradation': 0.20,
        'charging_port': 0.15, 'water_damage': 0.10,
        'camera_failure': 0.08, 'speaker_failure': 0.07,
        'software_issue': 0.05, 'touchscreen_failure': 0.05,
        'motherboard_failure': 0.03, 'overheating': 0.02,
    },
    'laptop': {
        'battery_degradation': 0.18, 'overheating': 0.16,
        'keyboard_failure': 0.14, 'screen_damage': 0.12,
        'motherboard_failure': 0.10, 'data_recovery': 0.08,
        'charging_port': 0.07, 'software_issue': 0.06,
        'speaker_failure': 0.05, 'water_damage': 0.04,
    },
    'tablet': {
        'screen_damage': 0.22, 'battery_degradation': 0.20,
        'charging_port': 0.15, 'touchscreen_failure': 0.12,
        'software_issue': 0.10, 'speaker_failure': 0.08,
        'camera_failure': 0.05, 'water_damage': 0.05,
    }
}

```

```

        'overheating': 0.03,
    },
}

def generate_synthetic_data(n_records=800):
    records = []
    for _ in range(n_records):
        device_type = random.choice(DEVICE_TYPES)
        brand = random.choice(BRANDS[device_type])
        age_months = random.randint(1, 72)
        total_repairs = max(0, int(np.random.poisson(age_months / 18)))
        months_since_last = random.randint(
            0, min(age_months, 36)) if total_repairs > 0 else age_months
        total_cost = round(
            random.uniform(0, total_repairs * 2500), 2
        ) if total_repairs > 0 else 0

        weights = FAILURE_WEIGHTS.get(device_type, FAILURE_WEIGHTS['phone'])
        failure_types = list(weights.keys())
        probs = [p / sum(weights.values()) for p in weights.values()]

        last_repair_type = np.random.choice(
            failure_types, p=probs) if total_repairs > 0 else 'none'

        base_prob = 0.1
        age_factor = min(age_months / 60, 1.0) * 0.3
        repair_factor = min(total_repairs / 5, 1.0) * 0.25
        recency_factor = max(
            0, 1 - months_since_last / 24
        ) * 0.2 if total_repairs > 0 else 0
        brand_factor = (0.05 if brand in ['Xiaomi']
                        else -0.05 if brand in ['Apple'] else 0)

        prob = min(max(
            base_prob + age_factor + repair_factor + recency_factor
            + brand_factor + random.uniform(-0.1, 0.1), 0.02), 0.98)

        next_failure = np.random.choice(failure_types, p=probs)
        season = random.choice([1, 2, 3, 4])

        records.append({
            'device_type': device_type, 'brand': brand,
            'age_months': age_months, 'total_repairs': total_repairs,
            'months_since_last_repair': months_since_last,
            'total_cost': total_cost,
            'last_repair_type': last_repair_type,
            'season': season,
            'next_failure_type': next_failure,
            'failure_probability': round(prob, 3),
            'will_fail_soon': 1 if prob > 0.45 else 0,
        })

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						109
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return pd.DataFrame(records)
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
import joblib
import os

CATEGORICAL_COLS = ['device_type', 'brand', 'last_repair_type']
NUMERIC_COLS = ['age_months', 'total_repairs',
                'months_since_last_repair', 'total_cost', 'season']

def create_encoders(df):
    encoders = {}
    for col in CATEGORICAL_COLS:
        le = LabelEncoder()
        le.fit(df[col].astype(str))
        encoders[col] = le
    return encoders

def prepare_features(df, encoders, scaler=None, fit_scaler=False):
    df_encoded = df.copy()
    for col in CATEGORICAL_COLS:
        le = encoders[col]
        df_encoded[col + '_encoded'] = le.transform(
            df_encoded[col].astype(str))

    feature_cols = [c + '_encoded' for c in CATEGORICAL_COLS] + NUMERIC_COLS
    X = df_encoded[feature_cols].values

    if fit_scaler:
        scaler = StandardScaler()
        X = scaler.fit_transform(X)
    elif scaler is not None:
        X = scaler.transform(X)
    return X, scaler

def prepare_training_data(csv_path):
    df = pd.read_csv(csv_path)
    encoders = create_encoders(df)
    X, scaler = prepare_features(df, encoders, fit_scaler=True)

    y_class = df['will_fail_soon'].values
    failure_encoder = LabelEncoder()
    y_failure_type = failure_encoder.fit_transform(df['next_failure_type'])
    y_prob = df['failure_probability'].values

    return X, y_class, y_failure_type, y_prob, encoders, scaler, \
        failure_encoder

def prepare_single_device(device_data, encoders, scaler):

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						110
Змн.	Арк.	№ докум.	Підпис	Дата		

```

df = pd.DataFrame([device_data])
for col in CATEGORICAL_COLS:
    le = encoders[col]
    val = str(df[col].iloc[0])
    if val not in le.classes_:
        df[col] = le.classes_[0]
X, _ = prepare_features(df, encoders, scaler=scaler)
return X

```

Додаток В. Лістинг програмного коду клієнтської частини

```

import { Routes, Route, Navigate } from 'react-router-dom';
import { Spin } from 'antd';
import { useAuth } from './context/AuthContext';
import MainLayout from './components/MainLayout';
import Login from './pages/auth/Login';
import Register from './pages/auth/Register';
import AdminDashboard from './pages/admin/Dashboard';
import AdminOrders from './pages/admin/Orders';
import AdminUsers from './pages/admin/Users';
import AdminParts from './pages/admin/Parts';
import MasterOrders from './pages/master/Orders';
import MasterOrderDetail from './pages/master/OrderDetail';
import ClientOrders from './pages/client/Orders';
import ClientDevices from './pages/client/Devices';
import ClientNewOrder from './pages/client/NewOrder';
import DeviceDetail from './pages/client/DeviceDetail';
import GuestTrack from './pages/guest/TrackOrder';
import OrderDetail from './pages/admin/OrderDetail';
import NotFound from './pages/NotFound';

const ProtectedRoute = ({ children, roles }) => {
  const { user, loading } = useAuth();
  if (loading) return <Spin size="large" />;
  if (!user) return <Navigate to="/login" />;
  if (roles && !roles.includes(user.role)) return <Navigate to="/" />;
  return children;
};

const App = () => {
  const { user, loading } = useAuth();
  if (loading) return <Spin size="large" />;

  const getDefaultRoute = () => {
    if (!user) return '/login';
    switch (user.role) {
      case 'admin': return '/admin/dashboard';
      case 'master': return '/master/orders';
      case 'client': return '/client/orders';
      default: return '/login';
    }
  };
};

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						111
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return (
  <Routes>
    <Route path="/login" element={<Login />} />
    <Route path="/register" element={<Register />} />
    <Route path="/track" element={<GuestTrack />} />
    <Route path="/" element={
      <ProtectedRoute><MainLayout /></ProtectedRoute>>
      <Route path="admin/dashboard" element={
        <ProtectedRoute roles={['admin']}>
          <AdminDashboard />
        </ProtectedRoute>} />
      <Route path="admin/orders" element={
        <ProtectedRoute roles={['admin']}>
          <AdminOrders />
        </ProtectedRoute>} />
      {/* маршрути для admin/orders/:id, admin/users, admin/parts,
        master/orders, master/orders/:id,
        client/orders, client/devices, client/devices/:id,
        client/new-order — аналогічна структура */}
    </Route>
    <Route path="*" element={<NotFound />} />
  </Routes>
);
};

export default App;
import { createContext, useContext, useState, useEffect, useCallback }
  from 'react';
import { authAPI } from '../api';
import { message } from 'antd';

const AuthContext = createContext(null);
export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  const loadUser = useCallback(async () => {
    const token = localStorage.getItem('accessToken');
    if (!token) { setLoading(false); return; }
    try {
      const { data } = await authAPI.getMe();
      setUser(data.user);
    } catch {
      localStorage.removeItem('accessToken');
      localStorage.removeItem('refreshToken');
    } finally {
      setLoading(false);
    }
  }, []);
};

```

```

useEffect(() => { loadUser(); }, [loadUser]);

const login = async (email, password) => {
  const { data } = await authAPI.login({ email, password });
  localStorage.setItem('accessToken', data.accessToken);
  localStorage.setItem('refreshToken', data.refreshToken);
  setUser(data.user);
  message.success('Успішний вхід!');
  return data.user;
};

const register = async (formData) => {
  const { data } = await authAPI.register(formData);
  localStorage.setItem('accessToken', data.accessToken);
  localStorage.setItem('refreshToken', data.refreshToken);
  setUser(data.user);
  return data.user;
};

const logout = () => {
  localStorage.removeItem('accessToken');
  localStorage.removeItem('refreshToken');
  setUser(null);
};

return (
  <AuthContext.Provider value={{
    user, loading, login, register, logout, loadUser
  }}>
    {children}
  </AuthContext.Provider>
);
};
import axios from 'axios';

const api = axios.create({
  baseURL: '/api',
  headers: { 'Content-Type': 'application/json' },
});

api.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;
    if (error.response?.status === 401 && !originalRequest._retry) {

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		113

```

originalRequest._retry = true;
const refreshToken = localStorage.getItem('refreshToken');
if (refreshToken) {
  try {
    const { data } = await axios.post(
      '/api/auth/refresh', { refreshToken });
    localStorage.setItem('accessToken', data.accessToken);
    localStorage.setItem('refreshToken', data.refreshToken);
    originalRequest.headers.Authorization =
      `Bearer ${data.accessToken}`;
    return api(originalRequest);
  } catch {
    localStorage.removeItem('accessToken');
    localStorage.removeItem('refreshToken');
    window.location.href = '/login';
  }
}
return Promise.reject(error);
}
);

```

```
export default api;
```

```

export const authAPI = {
  login: (data) => api.post('/auth/login', data),
  register: (data) => api.post('/auth/register', data),
  getMe: () => api.get('/auth/me'),
};

```

```

export const ordersAPI = {
  getAll: (params) => api.get('/orders', { params }),
  getById: (id) => api.get(`/orders/${id}`),
  create: (data) => api.post('/orders', data),
  updateStatus: (id, data) => api.put(`/orders/${id}/status`, data),
  addPart: (id, data) => api.post(`/orders/${id}/parts`, data),
  track: (num) => api.get(`/orders/track/${num}`),
};

```

```

export const devicesAPI = {
  getAll: (params) => api.get('/devices', { params }),
  getById: (id) => api.get(`/devices/${id}`),
  create: (data) => api.post('/devices', data),
  getHistory: (id) => api.get(`/devices/${id}/history`),
};

```

```

export const predictionsAPI = {
  predictML: (id) => api.post(`/predictions/ml/${id}`),
  predictGemini: (id) => api.post(`/predictions/gemini/${id}`),
  predictCombined: (id) => api.post(`/predictions/combined/${id}`),
  getDevicePredictions: (id) => api.get(`/predictions/device/${id}`),
};

```