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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Web-додаток для адміністрування та управління замовленнями з інтеграцією Web3-оплати та системою геолокаційного підбору сервісів»

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

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

_____ Павло СЯГРОВСЬКИЙ

Виконав: здобувач вищої освіти
групи ТЦР-42

Павло СЯГРОВСЬКИЙ

Керівник: Вадим ЧИТУЛЯН
старший викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Технології цифрового розвитку»

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сягровському Павлу Дмитровичу

1. Тема кваліфікаційної роботи: «Web-додаток для адміністрування та управління замовленнями з інтеграцією Web3-оплати та системою геолокаційного підбору сервісів»

керівник кваліфікаційної роботи старший викладач Вадим ЧИТУЛЯН, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вхідні дані до кваліфікаційної роботи: теоретичні відомості про сучасні підходи до розробки веб-систем для адміністрування та управління сервісними замовленнями, матеріали щодо використання клієнтських і серверних технологій для побудови інтерактивних веб-застосунків, технічна документація з React, TypeScript, Vite, Node.js, Express, Prisma, PostgreSQL, MinIO та мережі Polygon Amoy, а також інформаційні матеріали щодо інтеграції Web3-оплат, геолокаційного підбору сервісних центрів і автоматизованої калькуляції вартості послуг.

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

1. Аналіз предметної області авто-послуг, особливостей обробки клієнтських заявок, платежів і взаємодії між клієнтом, сервісним центром та адміністративною частиною системи.
2. Аналіз та порівняння існуючих аналогів програмних рішень для керування сервісними замовленнями, геолокаційного підбору послуг і підтримки цифрових платіжних механізмів.
3. Дослідження програмних засобів і технологій для розробки веб-додатку, вибір клієнтського та серверного стеку, засобів зберігання даних, роботи з файлами та інтеграції Web3-компонентів.
4. Проєктування та розробка функціоналу веб-додатку AutoAssist+ для адміністрування та управління замовленнями, автоматичної калькуляції вартості, геолокаційного підбору сервісних центрів, Web3-оплат і цифрового підтвердження виконання робіт.
5. Тестування розробленого програмного забезпечення, перевірка коректності роботи основних модулів, оцінка результатів функціонування системи та формулювання висновків щодо можливостей її практичного застосування.

5. Перелік графічного матеріалу: презентація

1. Актуальність теми, мета, об’єкт і предмет дослідження.
2. Аналіз існуючих аналогів і порівняння їх функціональних можливостей.
3. Вимоги до програмного забезпечення та основні сценарії використання системи.
4. Архітектура веб-додатку та загальна схема взаємодії його модулів.
5. Діаграми UML, що відображають структуру та логіку функціонування системи.
6. Блок-схеми основних процесів обробки замовлень, платежів і підтвердження виконання робіт.
7. Засоби реалізації програмного забезпечення та обґрунтування вибору технологічного стеку.
8. Інтерфейс веб-додатку та основні екранні форми користувача.
9. Результати тестування, апробація роботи системи та підсумкові висновки.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	20.02.2026 – 28.02.2026	Виконано
2	Аналіз та дослідження існуючих аналогів	01.03.2026 – 10.03.2026	Виконано
3	Аналіз технологій для реалізації web-додатку, Web3-оплати та геолокаційного підбору сервісів	11.03.2026 – 20.03.2026	Виконано
4	Проєктування архітектури платформи AutoAssist+, моделі даних та API-контрактів	21.03.2026 – 31.03.2026	Виконано
5	Програмна реалізація платформи	01.04.2026 – 20.04.2026	Виконано
6	Тестування платформи	21.04.2026 – 30.04.2026	Виконано
7	Оформлення роботи: вступ, висновки, реферат	01.05.2026 – 08.05.2026	Виконано
8	Розробка демонстраційних матеріалів	07.05.2026 – 08.05.2026	Виконано
9	Попередній захист роботи	08.05.2026 – 11.05.2026	Виконано

Здобувач вищої освіти _____ Павло СЯГРОВСЬКИЙ

Керівник
кваліфікаційної роботи _____ Вадим ЧИТУЛЯН

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 85 стор., 33 табл., 22 рис., 34 джерел.

Мета роботи – покращення процесу адміністрування та управління сервісними замовленнями шляхом розробки веб-додатку AutoAssist+ з автоматизованою калькуляцією вартості, геолокаційним підбором сервісних центрів, підтримкою Web3-оплат і формуванням цифрового підтвердження виконання робіт.

Об'єкт дослідження – процеси обробки клієнтських заявок, калькуляції вартості послуг, оплати та підтвердження виконання робіт у сфері авто-послуг.

Предмет дослідження – програмне забезпечення веб-додатка AutoAssist+ та методи інтеграції Web3-оплат, геолокаційного підбору сервісів і CRM-процесів у єдину інформаційну систему.

Короткий зміст роботи: У роботі проаналізовано існуючі підходи до автоматизації сервісних замовлень у CRM-системах, методи геолокаційного підбору сервісних центрів і засоби інтеграції Web3-платежів у веб-застосунки. Розглянуто інструментальні засоби реалізації системи, зокрема React, TypeScript, Vite, Leaflet, ethers.js, Node.js, Express, Prisma, PostgreSQL та MinIO. Сформовано технічне завдання, спроєктовано архітектуру веб-додатка та реалізовано основні функціональні модулі, серед яких керування замовленнями, автоматичне оцінювання вартості, платіжний модуль, нотифікації та механізм формування цифрового доказу виконання робіт. Реалізовано геолокаційний підбір сервісних центрів із використанням формули Гаверсина, верифікацію Web3-транзакцій за параметрами txHash, chainId, кількості підтверджень і тайм-ауту, а також генерацію PDF-чеків із

QR-посиланням на блокчейн-експлорер. Проведено модульне, інтеграційне та smoke-E2E тестування розробленого застосунку.

Галузь використання – CRM-системи для сервісних компаній у сфері авто-послуг, логістики та страхування, що потребують автоматизованої калькуляції вартості, геолокаційного підбору виконавців, прозорих Web3-оплат і цифрового підтвердження виконання робіт.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, WEB3-ОПЛАТА, CRM, ГЕОЛОКАЦІЯ, PROOF-HASH, REACT, NODE.JS.

ЗМІСТ

УМОВНІ ПОЗНАЧЕННЯ

ПЗ – програмне забезпечення

IDE – інтегроване середовище розробки

API – прикладний програмний інтерфейс

ORM – об’єктно-реляційне відображення

DB – база даних

CRUD – Create, Read, Update, Delete

UI – інтерфейс користувача

UX – досвід користувача

SPA – Single Page Application

JS – JavaScript

TS – TypeScript

SDK – Software Development Kit

CI/CD – безперервна інтеграція та доставка

Docker – система контейнеризації

Web3 – децентралізована веб-інфраструктура

txHash – хеш транзакції у блокчейні

QR – Quick Response код

PDF – Portable Document Format

MinIO – об’єктне сховище даних, аналог Amazon S3

CRM – Customer Relationship Management

UML – уніфікована мова моделювання

WSS – WebSocket Secure

JWT – JSON Web Token

GIS – геоінформаційна система

Haversine – формула для обчислення відстані між координатами

proofHash – криптографічний хеш цифрового доказу виконання робіт

Polygon Amoy – тестова мережа блокчейну для Web3-транзакцій

REST – архітектурний стиль побудови веб-API

MetaMask – клієнтський гаманець для Web3-аутентифікації

HTTP/HTTPS – протокол передачі гіпертексту та його захищена версія

JSON – JavaScript Object Notation

ER (ERD) – модель «сутність–зв’язок», ER-діаграма

FSM – скінченний автомат станів

RPS – кількість запитів за секунду

SLA – угода про рівень сервісу

SLO – цільовий рівень обслуговування

RPO – допустима втрата даних

RTO – час відновлення після збою

E2E – end-to-end тестування

RBAC – керування доступом на основі ролей

SIWE – Sign-In with Ethereum

TTL – час життя токена, nonce або тимчасового посилання

WAF – вебзастосунковий міжмережевий екран

RPC – віддалений виклик процедур

DLQ – черга помилкових повідомлень

OpenAPI – специфікація опису та документування API

MVP – мінімально життєздатний продукт

SRS – специфікація вимог до програмного забезпечення

FR – функціональна вимога

NFR – нефункціональна вимога

DSR – запит суб’єкта персональних даних

K8s – скорочена назва Kubernetes

DR – аварійне відновлення системи після збою

Vault – система керування секретами

Helm – менеджер пакетів для Kubernetes

BullMQ – бібліотека для організації черг завдань на базі Redis

Socket.IO – бібліотека для обміну подіями в реальному часі

k6 – інструмент навантажувального тестування

Prometheus – система збору та зберігання метрик

Grafana – система візуалізації метрик і дашбордів

Alertmanager – сервіс обробки та доставки сповіщень

OpenTelemetry – фреймворк для телеметрії, трасування та збору метрик

Proof-of-Completion – умовне позначення цифрового доказу виконання робіт, який формується після завершення сервісного замовлення та підтверджує факт виконання послуги

ВСТУП

Актуальність теми. Сучасний розвиток цифрових технологій зумовив активне впровадження веб-платформ у сферу сервісних послуг, де особливого значення набувають швидкість обробки звернень, автоматизація бізнес-процесів, прозорість фінансових операцій і можливість оперативного підбору виконавця. У сфері авто-послуг, логістики та страхування виникає потреба в інтегрованих інформаційних системах, здатних поєднати керування клієнтськими заявками, автоматизовану калькуляцію вартості, геолокаційний пошук сервісних центрів і механізми підтвердження оплати.

Поширені CRM-системи забезпечують облік клієнтів, супровід замовлень і зміну статусів, однак у більшості випадків не надають можливості відкритої верифікації транзакцій та цифрового підтвердження факту виконання робіт. Водночас Web3-технології дозволяють реалізувати перевірну модель фінансової взаємодії, за якої кожна транзакція може бути підтверджена через блокчейн-мережу, а результат виконання робіт може бути зафіксований у вигляді цифрового доказу. Поєднання Web3-оплат, геолокаційного підбору сервісів і класичного CRM-підходу створює передумови для побудови сучасної сервісної платформи з підвищеним рівнем довіри, автоматизації та прозорості. Саме це зумовлює актуальність теми кваліфікаційної роботи.

Мета роботи – покращення процесу адміністрування та управління сервісними замовленнями шляхом розробки веб-додатку AutoAssist+ з автоматизованою калькуляцією вартості, геолокаційним підбором сервісних центрів, підтримкою Web3-оплат і формуванням цифрового підтвердження виконання робіт.

Для досягнення мети у кваліфікаційній роботі необхідно розв’язати такі завдання:

– провести теоретичне обґрунтування використання Web3-платежів, геолокаційних алгоритмів і CRM-підходів у сервісних веб-платформах;
– виконати аналіз існуючих аналогів і визначити їхні переваги та недоліки;
– спроєктувати архітектуру веб-додатку, включно з клієнтською та серверною частинами;
– реалізувати функціональні модулі для роботи з користувачами, замовленнями, оплатами, нотифікаціями та геолокацією;
– інтегрувати механізм Web3-оплати через гаманець MetaMask і блокчейн Polygon Amoy;
– розробити систему формування PDF-чеків із QR-кодом і цифровим доказом виконання робіт;
– провести тестування програмного забезпечення та оцінити результати його роботи.

Об’єкт дослідження – веб-системи керування заявками, замовленнями та платежами у сфері сервісних послуг.

Предмет дослідження – методи інтеграції Web3-оплат і геолокаційних алгоритмів у CRM-процеси сервісних платформ.

Методи дослідження. Під час виконання кваліфікаційної роботи використано методи аналізу предметної області та існуючих рішень, порівняльного аналізу сучасних клієнтських і серверних технологій, проєктування інформаційних систем за допомогою UML-діаграм, ER-моделей і блок-схем, а також експериментальної розробки прототипу на базі стеку React, Node.js, PostgreSQL і Prisma ORM. Для перевірки працездатності запропонованого рішення застосовано методи функціонального, інтеграційного та сценарного тестування.

Наукова новизна одержаних результатів. У роботі запропоновано підхід до побудови веб-платформи, у якій класична CRM-архітектура поєднується з Web3-верифікацією транзакцій і цифровим підтвердженням завершення робіт. Реалізовано модель обробки замовлення, що охоплює створення заявки, автоматичну калькуляцію вартості, фіксацію суми перед оплатою, перевірку транзакції за параметрами txHash, chainId і кількості підтверджень, а також формування цифрового доказу виконання робіт. Додатково запропоновано механізм геолокаційного підбору сервісних центрів на основі формули Гаверсина з урахуванням відстані до клієнта.

Практична значущість одержаних результатів. Розроблений веб-додаток AutoAssist+ забезпечує комплексне керування сервісними заявками в режимі реального часу, підтримує інтеграцію з Web3-гаманцями, дозволяє виконувати оплату в блокчейн-мережі Polygon Amoy, формує PDF-чеки з QR-кодом і забезпечує цифрове підтвердження завершення кожної заявки. Практична цінність системи полягає в автоматизації процесів прийому та обробки замовлень, зниженні ризику людських помилок, підвищенні довіри клієнтів і можливості використання запропонованого рішення в автосервісі, логістиці та страхових сценаріях.

Апробація результатів та публікації. Основні положення та результати кваліфікаційної роботи можуть бути використані під час подальшої розробки, тестування й удосконалення веб-платформ для автоматизації сервісних замовлень із підтримкою Web3-оплат, геолокаційного підбору виконавців і цифрового підтвердження виконання робіт.

1. Сягровський П. Д. «Застосування Web3 та геолокаційних сервісів у платформі AutoAssist+ для цифровізації сервісних процесів» // VII Всеукраїнська науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. К.: ДУІКТ, 2026. С. 7–9.

2. Сягровський П. Д. «Веб-платформа AutoAssist+ для автоматизації сервісних замовлень із геолокаційним підбором та Web3-верифікацією платежів» // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026, С. 23-26.

3. Сягровський П. Д. «Інтеграція блокчейн-технологій у веб-платформі AutoAssist+ для верифікації сервісних платежів» // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026, (прийнято до друку).

4. Сягровський П. Д. «Інтелектуалізація обробки сервісних замовлень у веб-платформі AutoAssist+» // VI Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. К.: ДУІКТ, 2026, (прийнято до друку).

1 АНАЛІЗ ТА ХАРАКТЕРИСТИКА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Предметна область та проблематика сервісних замовлень

Розробка сервісної платформи для авто-послуг є актуальною через потребу швидко приймати заявки, підбирати виконавця за місцем події та підтверджувати оплату без ручних перевірок. Для клієнта важливі прозорість ціни, зрозумілий статус замовлення і доступ до чека після оплати. Для компанії важливі централізоване керування замовленнями, контроль навантаження між сервісними центрами та збереження доказів виконання робіт.

Наявні CRM-системи зазвичай дають облік звернень і статусів, але не охоплюють геолокаційний підбір сервісу та перевірку Web3-платежу в межах одного сценарію. Сервісні агрегатори добре працюють з картою і логістикою, однак не дають повного контролю над платіжним та документним етапом. Тому доцільно побудувати веб-систему, у якій замовлення, оцінка вартості, платіж, чек і доказ виконання пов’язані між собою.

Таблиця 1.1

Проблеми предметної області та способи їх вирішення в AutoAssist+

Проблема	Наслідок	Рішення в AutoAssist+
Невдалий вибір виконавця	зростають логістичні витрати	пошук центрів за координа радіусом
Оплата не прив’язана до оцінки	виникають спори щодо вартості	lock-механізм перед оплатою
Розрізнене зберігання файлів	складно підтвердити факт виконання	спільне зберігання файлів, receipt і proof-hash

1.2 Бізнес-процеси платформи

Платформа орієнтована на компанії, що працюють із замовленнями, прив’язаними до місця виконання робіт: автосервіси, страхові сервіси, евакуація, виїзна діагностика. Для таких сценаріїв важливі швидкий прийом заявки, автоматичний підбір виконавця, контроль вартості та прозора фіксація оплати.

У системі беруть участь три основні ролі: клієнт, співробітник сервісної компанії та адміністратор. Клієнт створює заявку, додає опис проблеми, координати і файли. Співробітник працює з оцінкою вартості, контролює статуси та завершення робіт. Адміністратор керує сервісними центрами, профілями калькуляції та загальними налаштуваннями середовища.

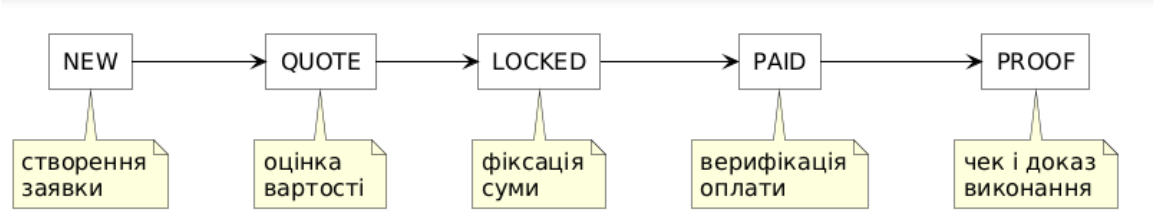


Рис. 1.1 Узагальнений життєвий цикл сервісного замовлення

Таблиця 1.2

Ролі користувачів та основні сценарії взаємодії

Роль	Дії в системі	Результат
Клієнт	створення заявки, перегляд карти, підтвердження оцінки, оплата	оформлене замовлення, чек, proof
Співробітник	перегляд заявок, корекція оцінки, зміна статусу, завершення робіт	актуальний стан заявки і документи
Адміністратор	керування центрами, профілями калькуляції, правами доступу	налаштоване середовище роботи платформи

Логіка обробки заявки побудована послідовно і допускає лише визначені переходи між станами. Це спрощує аудит подій, зменшує кількість неузгоджених дій і дозволяє пов’язати платіжний етап із конкретним замовленням.

1.3 Огляд існуючих рішень і підходів

На ринку існують три групи рішень, які частково закривають потрібні задачі: CRM-системи загального призначення, сервісні агрегатори та Web3-платіжні інструменти. Усі вони корисні, але кожна група має свої обмеження, якщо оцінювати повний цикл обробки сервісної заявки.

Таблиця 1.3

Порівняння наявних підходів і запропонованої системи

Критерій	CRM	Агрегатори	Web3 інструменти	AutoAssist+
Облік замовлень	повний	частковий	відсутній	повний
Геолокаційний підбір	обмежений	повний	відсутній	повний
Прозора перевірка платежу	ні	ні	так	так
Чек і proof у єдиному процесі	частково	частково	ні	так
Налаштування ролей і бізнес-логіки	так	обмежено	ні	так
Готовність до кастомізації	висока, але складна	низька	середня	повністю керована власником системи

1.4 Узагальнені вимоги до системи

Вимоги до платформи формуються з урахуванням повного життєвого циклу сервісного замовлення: створення заявки, підбору виконавця, розрахунку вартості, оплати, формування документів і підтвердження виконання робіт. На рівні першого розділу доцільно подати лише узагальнені групи вимог, а деталізацію функціональних і нефункціональних вимог розкрити в технічному завданні.

Таблиця 1.4

Узагальнені вимоги до платформи

Група вимог	Зміст
Замовлення	створення заявки, перегляд картки, контроль статусів, зберігання історії подій
Геолокація	підбір сервісного центру за координатами, радіусом і відстанню до клієнта
Оплата	фіксація суми перед оплатою, підтримка класичного платежу та Web3-верифікації
Документи	формування PDF-чека, зберігання вкладень і створення цифрового proof
Безпека	рольове розмежування доступу, перевірка прав на ресурс, захист файлів
Надійність	ідемпотентність критичних операцій, контроль станів, журналювання подій

Таке групування дозволяє відокремити загальні вимоги предметної області від детального технічного завдання, яке використовується на етапі проєктування та реалізації системи.

1.5 Архітектурні підходи до побудови платформи

Платформа побудована за клієнт-серверною схемою. Клієнтський застосунок відповідає за інтерфейс і взаємодію з користувачем, серверна частина – за бізнес-логіку, перевірку даних і роботу із зовнішніми сервісами. Такий поділ спрощує підтримку системи та дозволяє окремо масштабувати інтерфейс, API і сховище файлів.

Для критичних операцій використовується ідемпотентна логіка. Це стосується блокування суми, підтвердження платежу та генерації документів. Повторний запит не повинен створювати другий платіж або другий proof для одного й того самого замовлення.

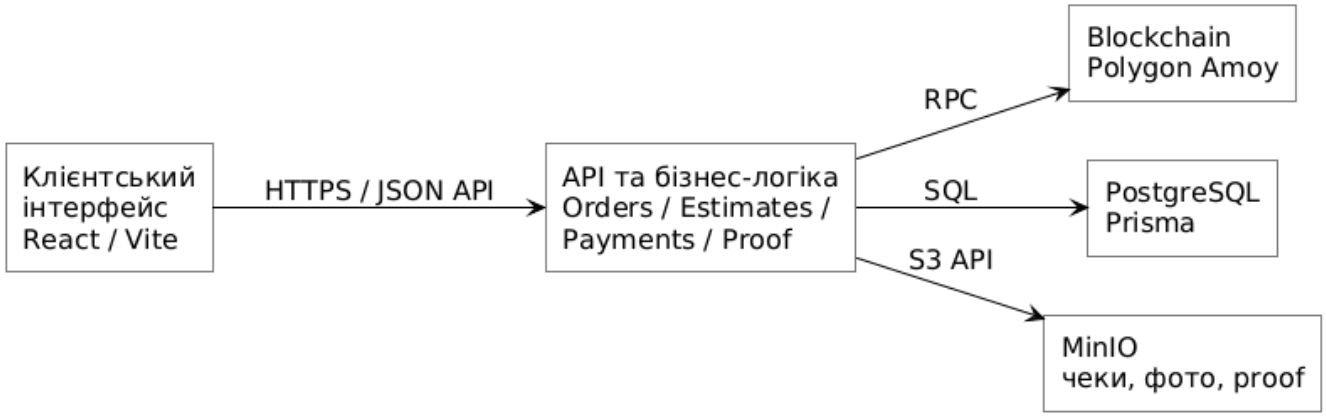


Рис. 1.2 Високорівнева архітектура платформи AutoAssist+

На схемі виділено основні компоненти системи: клієнтський інтерфейс, API та бізнес-логіку, базу даних PostgreSQL, файлове сховище MinIO і блокчейн-мережу Polygon Amoy. Такий поділ дозволяє відокремити роботу інтерфейсу від серверної логіки, зберігання даних, файлів і перевірки Web3-транзакцій.

1.6 Технологічний стек

Технологічний стек добирали за трьома критеріями: сумісність між клієнтською та серверною частинами, простота локального розгортання і наявність бібліотек для Web3, PDF та геолокації.

Таблиця 1.5

Технологічний стек платформи

Рівень	Інструмент	Роль	Причина вибору
Frontend	React + TypeScript + Vite	інтерфейс і клієнтська логіка	компонентна модель і типізація
Карти	Leaflet	робота з картою і координатами	просте підключення і достатній набір можливостей
Backend	Node.js + Express	REST API і бізнес-логіка	єдина мова розробки та широка екосистема
ORM і БД	Prisma + PostgreSQL	модель даних і транзакції	типобезпечний доступ до БД
Черги	BullMQ + Redis	фонові задачі та повторні спроби	стабільна асинхронна обробка
Файли	MinIO	чеки, фото, вкладення	S3-сумісне сховище
Web3	ethers.js + Polygon Amoy	підпис повідомлень і верифікація txHash	підходить для тестування Web3-сценарію
Інфрастр уктура	Docker Compose	локальне розгортання	швидкий запуск усіх сервісів

Обраний стек забезпечує повний цикл роботи платформи: від створення заявки в інтерфейсі до зберігання даних, обробки файлів, перевірки платежів і локального розгортання системи.

1.7 Узагальнена модель даних

Модель даних побудована навколо сутності замовлення. Саме вона пов’язує користувача, сервісний центр, оцінку вартості, платіж, чек, вкладення і доказ виконання. Така структура дозволяє відстежувати стан заявки від моменту створення до завершення робіт.

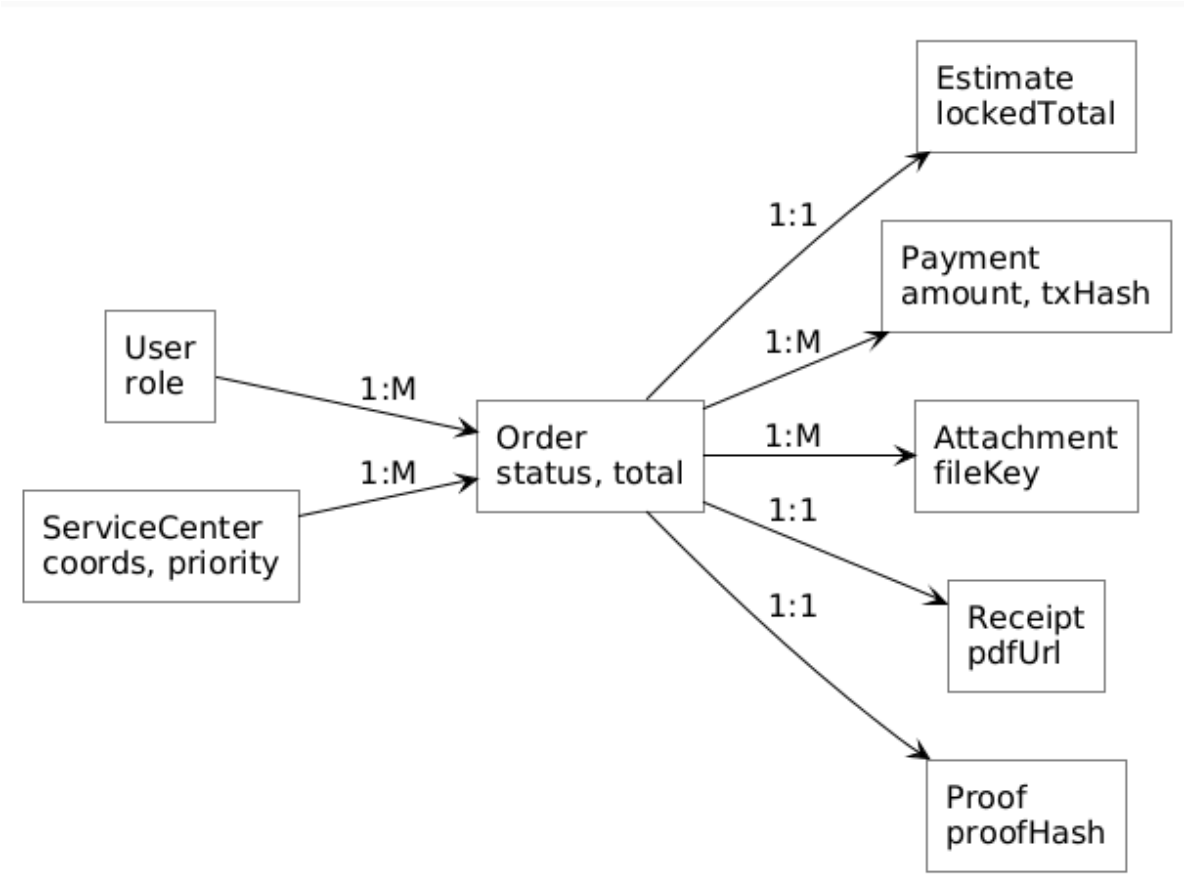


Рис. 1.3 Узагальнена модель основних сутностей платформи

На схемі показано, що користувач і сервісний центр можуть бути пов’язані з багатьма замовленнями, а кожне замовлення має власну оцінку вартості, платежі, вкладення, чек і proof. Оцінка вартості використовується перед оплатою, платіж зберігає суму та txHash, чек підтверджує фінансову операцію, а proof містить цифровий доказ завершення робіт.

Важливими інваріантами є незмінність заблокованої суми перед оплатою, унікальність txHash для Web3-транзакцій і можливість повторно обчислити proof-hash без зміни результату.

1.8 Принципи безпеки та захисту даних

У платформі потрібно захистити кілька груп даних: персональні відомості клієнта, вкладення, платіжну інформацію та події життєвого циклу. Тому безпека розглядається не як окремий модуль, а як набір правил, що діє для всіх підсистем.

Окрему увагу приділено журналюванню подій. Кожна зміна стану замовлення, успішна чи неуспішна перевірка оплати, формування чека та доступ до вкладення мають фіксуватися у логах або журналі подій замовлення.

Оскільки платформа працює із замовленнями, файлами, платіжними операціями та Web3-перевіркою, ризики безпеки потрібно розглядати на рівні всього життєвого циклу заявки. Основними напрямами захисту є контроль доступу, перевірка параметрів платежу, обмеження доступу до файлів, ідемпотентність критичних операцій і валідація вхідних даних. Такий підхід дозволяє зменшити ймовірність помилкових дій, дублювання операцій і несанкціонованого доступу до даних.

Таблиця 1.6

Основні ризики та засоби захисту

Ризик	Можливий наслідок	Засіб захисту
Несанкціонований доступ до замовлень	витік даних клієнта	рольова модель доступу і перевірка власника ресурсу
Підміна параметрів Web3-платежу	помилкове підтвердження оплати	перевірка chainId, суми, адреси та підтверджень
Доступ до чужих файлів	витік фото або чеків	тимчасові підписані URL і журнал доступу
Повторне виконання критичної дії	дублювання платежу або документа	ідемпотентність запитів
Некоректні вхідні дані	помилка бізнес-логіки	валідація запитів і контроль станів

Визначено проблеми предметної області, описано бізнес-процеси платформи та показано, чому для сервісного сценарію недостатньо окремо CRM, окремо карти або окремого платіжного модуля. Для AutoAssist+ потрібне поєднання цих компонентів у межах одного процесу.

Також сформовано узагальнені вимоги до системи. Отримані результати створюють основу для подальшого опису реалізації, тестування та оцінювання працездатності розробленого прототипу.

2 ВИМОГИ ДО РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Роль технічного завдання у життєвому циклі проєкту

Технічне завдання для проєкту AutoAssist+ використовується як основний документ, у якому фіксуються межі системи, функціональні та нефункціональні вимоги, правила інтеграції, обмеження і критерії приймання. Для даного проєкту воно потрібне не лише для опису MVP, а й для узгодження логіки замовлень, оплат, геолокаційного підбору, формування чеків і цифрового підтвердження виконання робіт.

Таблиця 2.1

Основні функції технічного завдання у проєкті

Функція	Що фіксується	Практичне значення для проєкту
Узгодження меж	Склад MVP, дозволені сценарії, зовнішні інтеграції	Не допускає розростання функціоналу без контролю
Формалізація вимог	Функціональні та нефункціональні вимоги	створює основу для реалізації та тестування
Трасувальність	Зв’язок вимог з API, БД, тестами та діаграмами	Спрощує перевірку повноти реалізації
Приймання результату	Acceptance-критерії, очікувані статуси та артефакти	Полегшує фінальну перевірку системи

Отже, технічне завдання виступає основою для подальшого проєктування, реалізації та перевірки системи.

2.2 Діаграми проєктування системи

Для уточнення вимог і логіки роботи використано набір графічних моделей: діаграму варіантів використання, діаграму послідовностей, діаграму станів замовлення, ER-модель і діаграму розгортання. Ці артефакти відображають ролі користувачів, основний сценарій обробки заявки, допустимі переходи між станами, структуру даних і склад інфраструктури системи.

2.2.1 Діаграма варіантів використання

Діаграма також показує розмежування відповідальності між учасниками системи. Неавторизований користувач має доступ лише до публічної інформації та можливості створення звернення, клієнт працює зі своїми замовленнями, оплатами й документами, персонал сервісу відповідає за операційну обробку заявок, а адміністратор керує довідниками, сервісними центрами та аудитом. Таке розділення потрібне для подальшого формування рольової моделі доступу і перевірки прав користувачів на рівні серверної частини.

Окремо на діаграмі виділено сценарії, пов’язані з оплатою, отриманням чека та підтвердженням виконання робіт. Це дозволяє показати, що платформа не обмежується лише створенням заявки, а охоплює повний цикл її супроводу. Надалі ці варіанти використання застосовуються як основа для функціональних вимог, API-контрактів і тестових сценаріїв.

Діаграма варіантів використання демонструє основні дії, які виконують користувачі системи залежно від своєї ролі. Вона дозволяє визначити межі відповідальності між клієнтом, персоналом сервісу та адміністратором. Такий підхід спрощує подальше проєктування функціональних модулів і перевірку відповідності системи поставленим вимогам.

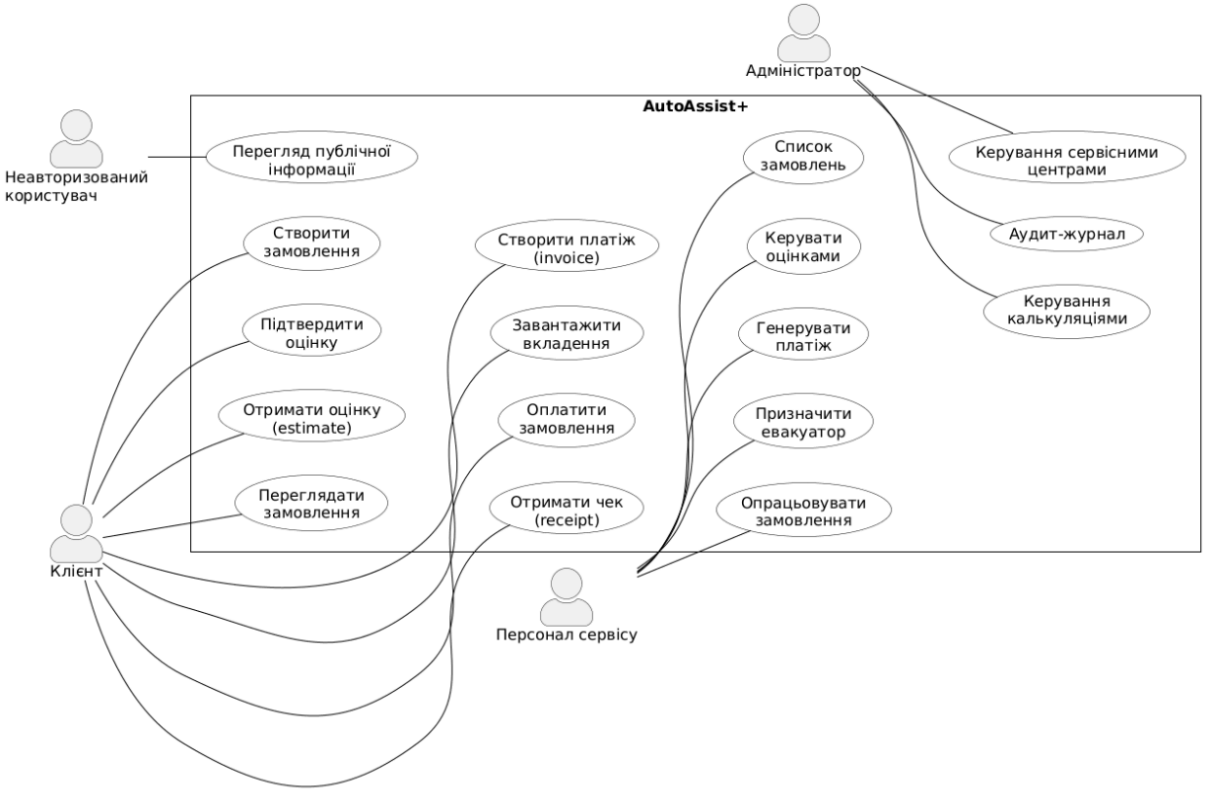


Рис. 2.1 Діаграма варіантів використання AutoAssist+

2.2.2 Діаграма послідовностей

На діаграмі послідовностей показано основний сценарій: створення заявки, отримання оцінки, фіксацію суми, верифікацію Web3-транзакції, генерацію чеку та формування доказу виконання. Така схема потрібна для узгодження API-викликів, точок синхронізації та зовнішніх звернень до блокчейн-вузла.

Діаграма також показує порядок взаємодії між клієнтським інтерфейсом, API-сервером і внутрішніми сервісами системи. Вона дозволяє простежити, у який момент створюється замовлення, коли виконується оцінка вартості, як підтверджується платіж і коли формується цифрове підтвердження. Це важливо для узгодження логіки фронтенду, серверної частини та зовнішнього Web3-компонента.

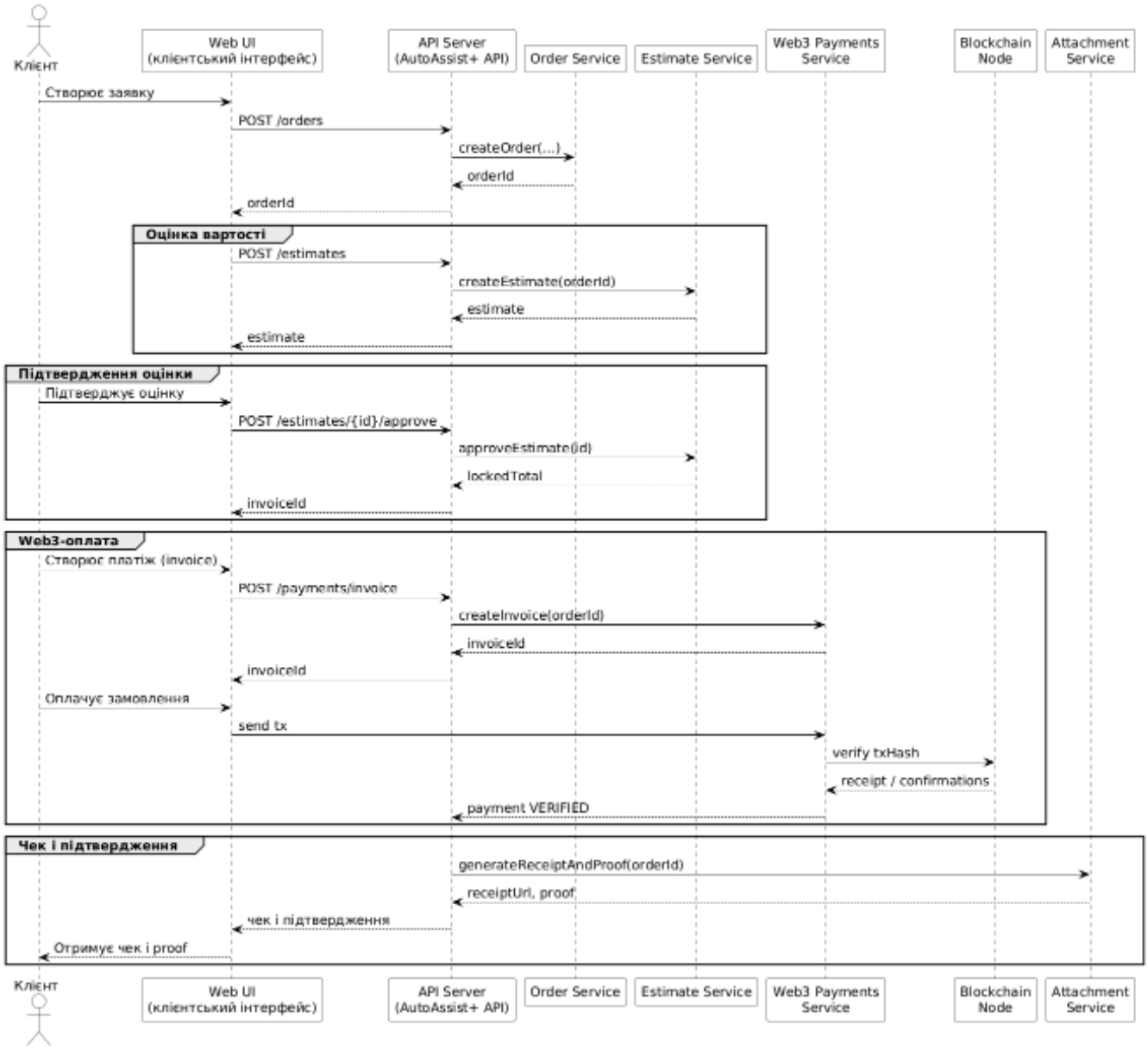


Рис. 2.2 Діаграма послідовностей основного процесу

Діаграма станів показує, як замовлення переходить між етапами: створення, оцінка, фіксація суми, оплата та завершення. Кожен перехід виконується лише після відповідної дії або підтвердження системи.

2.2.3 Діаграма станів замовлення

Діаграма станів фіксує допустимі переходи між етапами життєвого циклу замовлення. Вона використовується для серверної перевірки переходів і не допускає завершення заявки без підтвердження оплати або початку робіт.

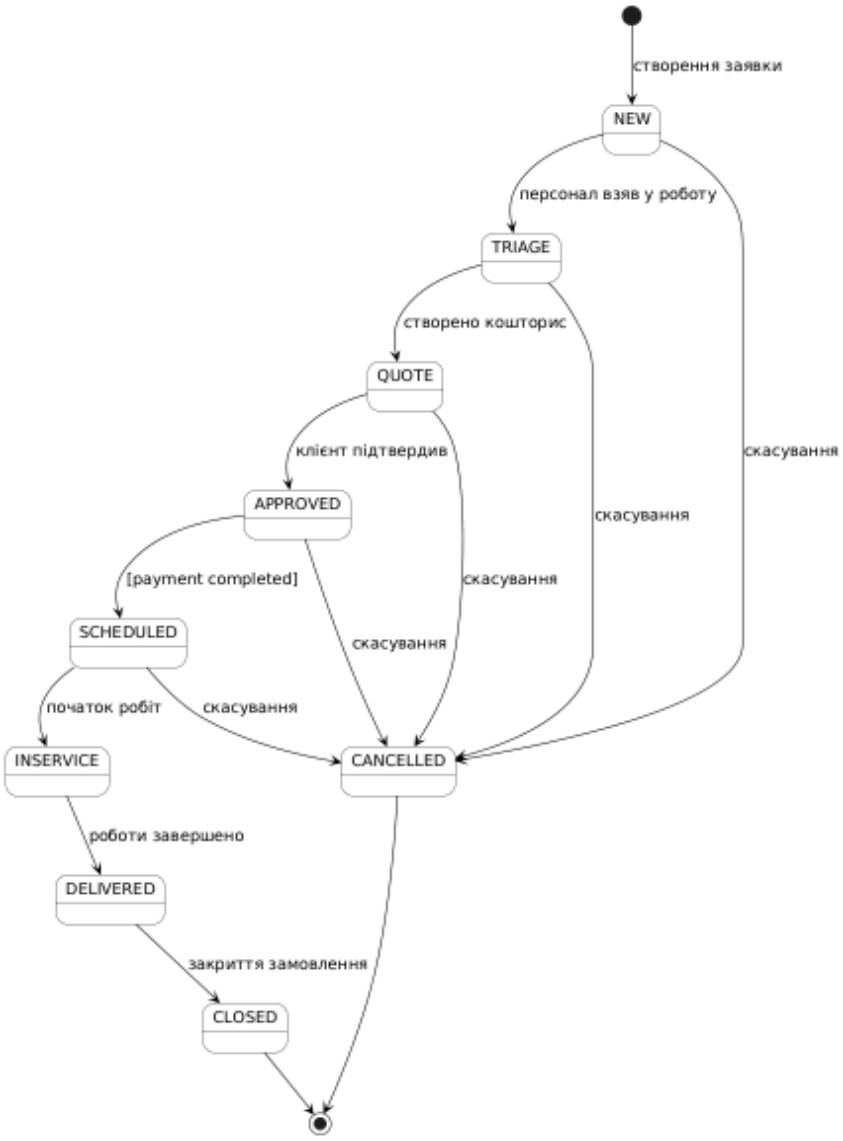


Рис. 2.3 Діаграма станів замовлення

Таблиця 2.2

Ключові стани замовлення та умови переходу

Стан	Коли встановлюється
NEW	Після створення заявки
TRIAGE	Після взяття в роботу
QUOTE	Після формування оцінки
APPROVED	Після підтвердження estimate
SCHEDULED	Після успішної оплати
INSERVICE	Після початку робіт
READY	Після завершення робіт
DELIVERED / CLOSED	Після передачі результату клієнту
CANCELLED	Після допустимого скасування

2.2.4 Узагальнена ER-модель

ER-модель описує логічну схему даних. Центральною сутністю виступає Order, з якою пов’язані оцінка, платіж, вкладення, сервісний центр, таймлайн подій та інші об’єкти, що формують повний цикл опрацювання заявки.

Центральною сутністю ER-моделі є Order, яка об’єднує основні процеси системи: створення заявки, вибір сервісного центру, розрахунок вартості, оплату та збереження підтверджувальних файлів. Пов’язані сутності ServiceCenter, CalcProfile, Payment та Attachment забезпечують зберігання даних про виконавця, правила оцінювання, платіж і вкладення.

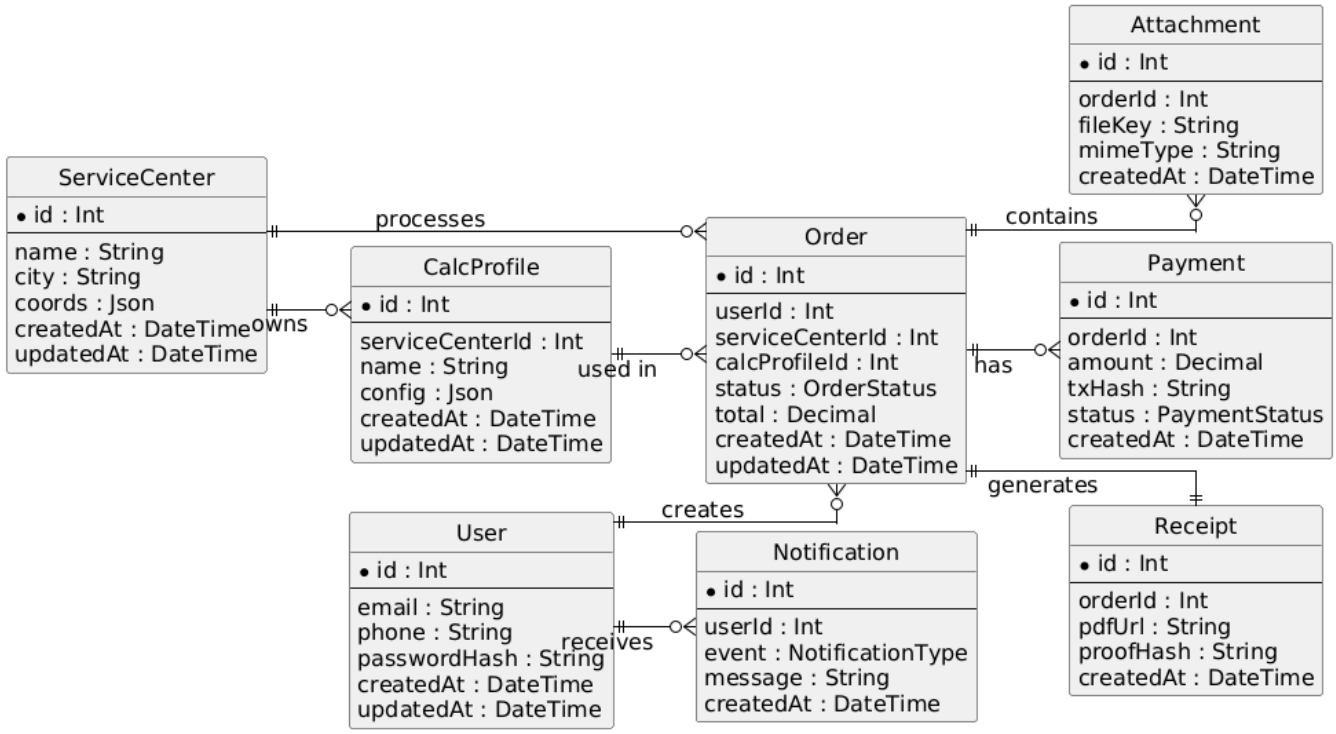


Рис. 2.4 Узагальнена ER-модель AutoAssist+

2.2.5 Діаграма розгортання та компонентів

Діаграма розгортання показує склад інфраструктури: браузер, вебчастину, API, базу даних, Redis, MinIO, засоби моніторингу та зовнішній blockchain node. Усередині API-сервера виділено ключові компоненти, що відповідають за замовлення, оцінки, платежі, документи та повідомлення.

Така схема показує, як окремі частини системи взаємодіють між собою під час обробки запитів.

Вебчастина передає дані до API, сервер працює з базою даних, кешем, файловим сховищем і зовнішнім blockchain node.

Окреме виділення сервісів усередині API спрощує підтримку, масштабування та контроль основних бізнес-процесів.

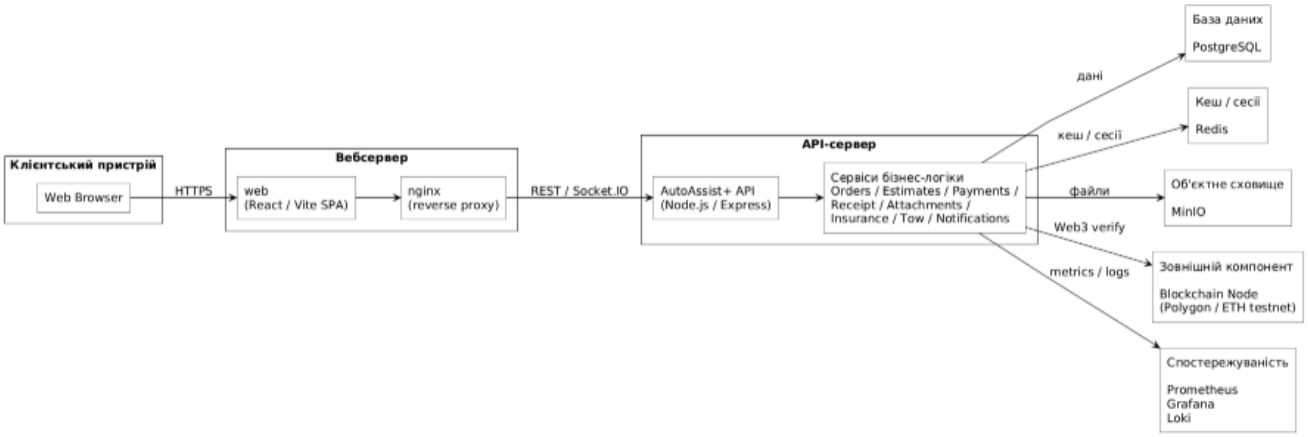


Рис. 2.5 Діаграма розгортання AutoAssist+

2.2.6 Узагальнення

Сукупність наведених діаграм дозволяє розглядати як зв’язану систему з визначеними ролями, сценаріями, станами, структурою даних та інфраструктурними залежностями. Надалі ці артефакти використовуються як основа для формування вимог, проєктування API та реалізації основних модулів платформи.

2.3 Формування технічного завдання для проєкту AutoAssist+

Наведено у форматі, близькому до SRS. Основні пункти згруповано так, щоб від текстового опису швидко переходити до перевірюваних таблиць: мета, межі MVP, ролі, вимоги, обмеження, API, модель даних, критерії приймання та ризики.

2.3.1 Мета проєкту

Проєкт спрямований на покращення процесу адміністрування та управління сервісними замовленнями за рахунок поєднання в одній платформі створення заявки, автоматичного оцінювання вартості, геолокаційного підбору виконавця, перевірки Web3-оплати та формування цифрового підтвердження виконання робіт.

Таблиця 2.3

Цілі проєкту та очікувані результати

Ціль	Показник або очікуваний результат
Скоротити час оформлення заявки	Клієнт може створити заявку без звернення до оператора
Уніфікувати розрахунок вартості	Оцінка виконується за CalcProfile і повертає деталізовану структуру
Підтвердити оплату без ручної перевірки	Платіж верифікується за txHash, мережею, сумою й адресою
Надати клієнту підтверджуючі документи	Після оплати формується PDF-чек і proof-of-completion
Покращити вибір сервісного центру	Геомодуль повертає найближчі доступні центри за координатами

2.3.2 Обсяг робіт та межі MVP

У складі MVP залишено тільки ті функції, які потрібні для наскрізного сценарію замовлення. Частину можливостей свідомо винесено за межі першої версії, щоб не ускладнювати реалізацію та тестування.

Такий поділ дозволяє зосередити першу версію системи на ключовому бізнес-процесі: створенні заявки, розрахунку вартості, оплаті та формуванні підтвердження.

Таблиця 2.4

Склад MVP та функції, винесені за межі першої версії

У межах MVP	Поза межами MVP
Створення, перегляд і фільтрація замовлень	Інтеграції з комерційними платіжними провайдерами
Автокалькуляція та lock суми перед оплатою	Фіскальні інтеграції та облік ПРРО
Web3-верифікація платежу в тестовій мережі	Повний on-chain proof у виробничих мережах
Геопошук сервісних центрів за радіусом	Маршрутизація з урахуванням трафіку та ETA
Генерація чеку, proof, вкладень і нотифікацій	Розширена аналітика та прогнозне ранжування центрів

2.3.3 Рольова модель та контроль доступу

Контроль доступу будується на RBAC і перевірці належності ресурсу на сервері. Клієнт працює лише зі своїми замовленнями, персонал – з операційною частиною, адміністратор – з довідниками, конфігураціями та аудитом.

Таблиця 2.5

RBAC-матриця основних дій

Дія / роль	Неавторизований	Клієнт	Персонал сервісу	Адміністратор
Реєстрація та вхід	Так	Так	Так	Так
Створення замовлення	Ні	Так	Ні	Ні
Перегляд власних замовлень	Ні	Так	Ні	Ні
Опрацювання замовлень	Ні	Ні	Так	Так
Керування оцінками	Ні	Ні	Так	Так
Ініціювання платежу	Ні	Так	Ні	Ні
Генерація / перевипуск чеків	Ні	Перегляд	Так	Так
Керування сервісними центрами	Ні	Ні	Ні	Так

2.3.4 Функціональні вимоги

Функціональні вимоги подано в компактному табличному вигляді. Для кожної вимоги визначено короткий опис і перевірюваний результат, який надалі може бути прив’язаний до тест-кейсу.

Таблиця 2.6

Перелік функціональних вимог

ID	Вимога	Критерій приймання
FR-01	Створення замовлення	Система приймає валідовані дані, координати й вкладення; у таймлайні фіксується подія CREATED.
FR-02	Автокалькуляція	Повертається структура estimate з total, items і calcProfile.
FR-03	Lock суми	Зафіксована сума не змінюється під час оплати; повторний виклик не дублює ефект.
FR-04	Web3-верифікація	Перевіряються txHash, chainId, сума, адреса одержувача та кількість підтверджень.
FR-05	Генерація PDF-чеку	Після успішної оплати формується чек із QR-посиланням на ресурс перевірки.
FR-06	Proof-of-completion	Створюється канонічний звіт із детермінованим SHA-256 proofHash.
FR-07	Геопошук центрів	Система повертає центри в заданому радіусі з distanceKm і сортуванням.
FR-08	Сповіщення	Користувачі отримують ключові оновлення статусів у реальному часі.
FR-09	Адміністрування	Адміністратор керує довідниками, профілями калькуляції та аудитом.

Для Web3-платежів окремо визначено типові помилки верифікації, щоб система могла коректно реагувати на різні ситуації. Це дозволяє не переводити замовлення в оплачений статус без підтвердженої транзакції. Також такі коди

спрощують повторну перевірку платежу та запобігають дублюванню вже оброблених операцій.

Таблиця 2.7

Коди помилок для Web3-верифікації платежів

Код	Умова	Реакція системи
WRONG_CHAIN	chainId не відповідає конфігурації	Повертається помилка і платіж не підтверджується
TX_NOT_FOUND	Провайдер не знаходить txHash	Запит можна повторити пізніше
TX_TIMEOUT	Не набрано потрібну кількість підтверджень	Платіж залишається у проміжному стані
AMOUNT_MISMATCH	Сума транзакції не збігається з lock	Система забороняє переведення у VERIFIED
ALREADY_VERIFIED	txHash уже оброблено	Повертається існуючий результат без дублювання

2.3.5 Нефункціональні вимоги

Нефункціональні вимоги задають межі продуктивності, безпеки, доступності й спостережуваності. Вони потрібні для того, щоб система коректно працювала не лише функціонально, а й у реальному середовищі.

Таблиця 2.8

Нефункціональні вимоги системи

ID	Категорія	Вимога
NFR-01	Продуктивність	Критичні API-ендпойнти мають вкладатися у встановлені порогові значення затримки.
NFR-02	Надійність	Критичні операції виконуються транзакційно та підтримують ідемпотентність.
NFR-03	Масштабованість	API та фонова обробка допускають горизонтальне масштабування.
NFR-04	Безпека	Використовуються JWT/SIWE, TLS, валідація входу та контроль доступу до файлів.
NFR-05	Спостережуваність	У логах присутній traceId; збираються метрики та налаштовано алертинг.

2.3.6 Обмеження та припущення

Для першої версії системи зафіксовано низку практичних обмежень. Їх наведено окремо, оскільки вони прямо впливають на оцінку складності реалізації та на підготовку тестового середовища.

Таблиця 2.9

Основні обмеження та конфігураційні припущення

Параметр	Поточне припущення / обмеження
Blockchain-мережа	Використовується тестова мережа Polygon Amoy з фіксованим chainId.
Геопошук	Розрахунок ведеться за відстанню між координатами без урахування трафіку.
Вкладення	Застосовуються обмеження за типом, розміром і строком дії presigned URL.
Таймаути	Перевірка txHash і зовнішні звернення обмежуються конфігурованим таймаутом.

У другому розділі сформовано вимоги до розробки програмного забезпечення. Описано роль технічного завдання, наведено діаграми проєктування системи та визначено основні вимоги до функціональності, доступу, API-контрактів, моделі даних, обмежень і критеріїв приймання.

Використання UML-діаграм, ER-моделі та табличного подання вимог дозволило структурувати матеріал і показати зв’язок між ролями користувачів, сценаріями роботи, станами замовлення та технічними компонентами системи. Отримані результати створюють основу для подальшої реалізації, тестування та оцінювання працездатності розробленого прототипу.

3 ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ ДОДАТКУ AUTOASSIST+

3.1 Вибір технологічного стеку та обґрунтування

Платформа побудована на єдиному стеку TypeScript для клієнтської та серверної частин. Це спрощує повторне використання типів, зменшує кількість помилок на межі API і робить кодову базу більш цілісною. Для клієнта обрано React і Vite, для серверної логіки – Node.js, для зберігання даних – PostgreSQL і Prisma, для черг – Redis та BullMQ, для файлів – MinIO, а для Web3-модуля – ethers.js і мережу Polygon Amoy.

Таблиця 3.1

Порівняння обраних засобів з альтернативами

Компонент	Обраний варіант	Типова альтернатива	Короткий висновок
SPA	React	Vue / Angular	React зручний для великої кількості компонентів
API	Express	NestJS / Fastify	Express достатній для MVP і простіший у налаштуванні
ORM	Prisma	TypeORM / Sequelize	Prisma дає чистішу схему і міграції
Сховище	MinIO	Локальний диск / AWS S3	MinIO підходить для локального та тестового середовища
Web3	ethers.js	web3.js	ethers.js має зрозуміліший API для верифікації
Контейнери	Docker Compose	ручний запуск сервісів	Compose відтворює середовище однією командою

Вибір технологій здійснювався з урахуванням сумісності між компонентами, простоти локального розгортання, підтримки Web3-верифікації, роботи з файлами та

можливості подальшого масштабування системи. Порівняння обраних засобів з типовими альтернативами наведено в таблиці 3.1.

3.2 Середовище розробки та швидкий запуск

Локальне середовище організовано так, щоб новий розробник міг швидко підняти всі сервіси без ручного встановлення окремих залежностей. Основний сценарій – запуск контейнерів із БД, Redis, сховищем файлів та допоміжними сервісами, після чого окремо піднімаються клієнтська і серверна частини.

Швидкий запуск середовища передбачає встановлення залежностей, запуск контейнерів інфраструктури та окремий старт клієнтської і серверної частин. Команди запуску винесено до додатків, оскільки вони мають довідковий характер і не впливають на опис основних програмних засобів реалізації.

Таблиця 3.2

Склад локального середовища розробки

Сервіс	Призначення	Тип запуску	Примітка
web	Клієнтська частина	pnpm dev	Інтерфейс користувача
api	Серверний застосунок	pnpm dev	REST API та WebSocket
PostgreSQL	База даних	Docker	Основні бізнес-дані
Redis	Черги і кеш	Docker	BullMQ jobs
MinIO	Файли	Docker	Вкладення, чеки, proof
Prometheus /Grafana	Моніторинг	Docker	Метрики під час тестування

3.3 Архітектура системи та структура репозиторію

Архітектура має багатошарову побудову: браузер, клієнтська SPA, серверний API, фонові сервіси, база даних, файлове сховище та спостережуваність. Така схема дозволяє ізолювати бізнес-логіку на сервері, а ресурсоємні дії винести у черги.

Таблиця 3.3

Структура монорепозиторію

Каталог	Вміст	Практичне призначення
apps/web	React SPA	Форми, карта, кабінет клієнта, сторінки оплат
apps/api	Node.js API	Маршрути, сервіси, валідація, інтеграції
packages/shared	Спільні типи	DTO, константи, коди помилок
infra/compose	Docker Compose	Підняття локального середовища
docs	Схеми і специфікації	OpenAPI, UML, RUNBOOK
tests	Тести і профілі навантаження	Інтеграційні, e2e та k6 сценарії

Таблиця 3.4

Основні серверні сервіси та відповідальність

Сервіс	Основні дії	Результат
Orders / Estimates	створення й оновлення замовлень, автокалькуляція, фіксація суми	замовлення, події таймлайну, оцінка вартості та підсумкова сума
Payments / Web3	створення платежу, перевірка txHash, контроль статусу оплати	платіж зі статусом VERIFIED або FAILED
Documents / Notifications	формування PDF-чека, proof-hash, сповіщення та WebSocket-події	підтвердні документи у MinIO та оновлення статусів для користувача

3.4 API, OpenAPI та контрактна дисципліна

Усі дії клієнта виконуються через REST API. Формальний опис маршрутів зберігається в OpenAPI, тому структура запитів, відповіді та коди помилок залишаються узгодженими між фронтендом, тестами і документацією.

Таблиця 3.5

Основні групи API платформи

Група маршрутів	Призначення	Приклади
/orders	Робота із замовленнями	create, list, detail, update status
/estimates	Калькуляція вартості	auto-estimate, approve, lock
/payments	Оплати	invoice, classic pay, web3 verify
/receipts	Чеки	generate, list, get link
/attachments	Вкладення	upload, list, confirm
/notifications	Події і сповіщення	count, list, mark read
/wallet	Web3-автентифікація	nonce, verify, link wallet
/admin/*	Адміністрування	calc profiles, service centers

Для всіх маршрутів застосовується єдиний підхід до формування відповідей: успішні операції повертають результат виконаної дії, помилки валідації супроводжуються поясненням причини, а конфліктні стани блокуються на рівні серверної бізнес-логіки. Такий підхід дозволяє фронтенду, тестам і OpenAPI-документації працювати з однаковими контрактами.

3.5 Транзакційність та ідемпотентність критичних операцій

Фінансові та статусні операції в системі виконуються атомарно. Це важливо для сценаріїв, де одна дія впливає одразу на кілька сутностей: замовлення, платіж,

чек і таймлайн. Окремо реалізовано ідемпотентність, щоб повторний запит не створював дублікати.

Для цього в системі використовуються транзакційні операції, перевірка поточного стану, унікальність txHash та ідемпотентність повторних запитів. Допустимі переходи станів уже визначено в діаграмі станів розділу 2, а в цьому підрозділі розглянуто саме реалізаційні інваріанти критичних операцій.

Таблиця 3.6

Критичні транзакційні операції та інваріанти

Операція	Пов’язані сутності	Що перевіряється	Інваріант
Lock estimate	Order + Estimate	Поточний статус і сума	Після lock сума не змінюється
Verify payment	Order + Payment	chainId, amount, txHash	Один txHash = один підтверджений платіж
Generate receipt	Payment + Receipt	Статус оплати	Чек не генерується без оплати
Create proof	Order + Proof + Attachment	Статус робіт і дані звіту	Один proof для одного order
Status transition	Order + Timeline	Поточний і цільовий стан	Недопустимі переходи блокуються

3.6 Дані та зберігання: PostgreSQL, Prisma, файли, MinIO, геопошук

Підсистема зберігання поєднує реляційну базу даних для основних бізнес-сутностей, окреме файлове сховище для вкладень і документів, а також геодані для пошуку сервісних центрів. Такий поділ знижує навантаження на базу даних, спрощує аудит і дозволяє окремо контролювати доступ до файлів.

Основні бізнес-дані зберігаються у PostgreSQL, а Prisma використовується для опису моделей, міграцій та типізованого доступу до бази.

Файли, PDF-документи та proof-матеріали винесено в MinIO, щоб не перевантажувати реляційну базу великими вкладеннями.

Таблиця 3.7

Розподіл даних між сховищами

Тип даних	Місце зберігання	Приклади	Причина вибору
Структуровані	PostgreSQL	orders, payments, receipts	Транзакційність і зв’язки
Схема доступу	Prisma	моделі та міграції	Типобезпечний доступ
Файли	MinIO	фото, PDF, proof-json	Окреме S3-сумісне сховище
Тимчасові посилання	MinIO presigned URL	відкриття документа	Контрольований доступ
Геодані	PostgreSQL / coords	serviceCenter.lat/lng	Пошук найближчих центрів
Службові метадані	PostgreSQL	mime, file key, size	Аудит і прив’язка до order

Таблиця 3.8

Політика зберігання файлів і доступу

Категорія	Приклад	Спосіб доступу	Примітка
Вкладення замовлення	фото поломки	presigned GET	тільки після перевірки прав
Фінансові документи	PDF-чек	server generated link	короткий TTL
Proof-матеріали	JSON, фото, QR	керований доступ	використовується в аудиті
Службові прев’ю	thumbnail	внутрішній доступ	допоміжні артефакти UI

3.7 Web3-інтеграція, платежі та верифікація транзакцій

Web3-модуль не підписує транзакції замість користувача. Його задача – коректно отримати txHash, перевірити мережу, адресу одержувача, суму і кількість підтверджень, а потім синхронізувати результат з внутрішнім станом замовлення.

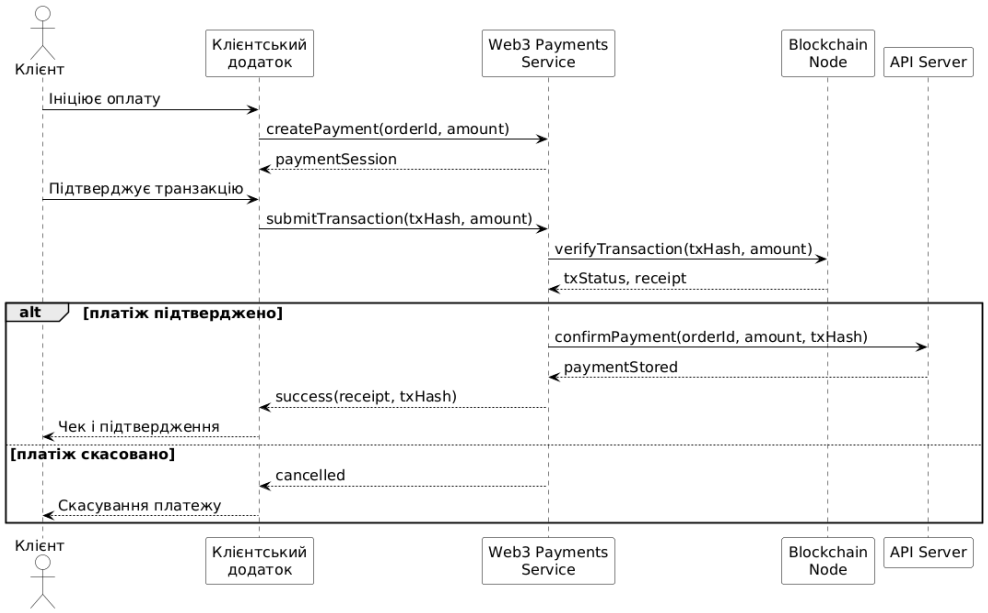


Рис. 3.1 Послідовність верифікації Web3-платежу

Таблиця 3.9

Умови успішної верифікації транзакції

Параметр	Що перевіряється	Очікуване значення
chainId	Мережа, у якій відправлено tx	80002 (Polygon Amoy)
to	Адреса одержувача	Сервісна адреса платформи
amount	Фактичний переказ	дорівнює lock-сумі
confirmations	Кількість підтверджень	не менше заданого порога
tx uniqueness	унікальність і receipt	txHash не використано раніше, receipt отримано

Якщо хоча б одна з умов не виконується, система не переводить платіж у підтверджений стан. Це дозволяє уникнути помилкового зарахування, повторного використання txHash і розбіжностей між оплатою та сумою, зафіксованою в замовленні.

3.8 Документи та доказ виконання: PDF-чеки і Proof-of-Completion

Після успішної оплати система формує підтвердні артефакти: PDF-чек для користувача та proof-of-completion для технічного підтвердження виконання робіт. PDF-чек використовується як фінансовий документ, а proof-of-completion фіксує завершення замовлення, вкладення, координати, виконавця та контрольний хеш цілісності.

Таблиця 3.10

Підтвердні документи та їх призначення

Артефакт	Основний зміст	Призначення
PDF-чек	омер замовлення, сума, метод оплати, txHash, дані клієнта, QR-посилання	підтвердження факту оплати
Proof-of-Completion	orderId, completedAt, coords, attachments, performedBy, proofHash	доказ завершення робіт і контроль цілісності
Presigned URL	тимчасове посилання на файл	безпечний доступ до документа без відкритого зберігання
QR-посилання	шлях до сторінки перевірки	швидка зовнішня перевірка документа

Обидва документи пов’язуються із замовленням і зберігаються у файловому сховищі. Доступ до них надається контрольовано, через тимчасові посилання або сторінку перевірки.

3.9 Якість, безпека та експлуатаційна надійність

Для забезпечення стабільної роботи передбачено кілька рівнів перевірки: автоматизоване тестування, контроль якості коду, перевірку API-контрактів, моніторинг основних метрик і базові експлуатаційні процедури. Такий підхід дозволяє виявляти помилки ще до розгортання системи та швидко реагувати на збої під час роботи.

Таблиця 3.11

Засоби забезпечення якості та експлуатаційної надійності

Напрям	Що перевіряється або контролюється	Засіб або очікуваний результат
Модульне тестування	Окремі функції та утиліти	перевірка базової логіки
Інтеграційне тестування	взаємодія API, БД і сервісів	контроль коректності бізнес-процесів
E2E / smoke-тестування	критичний сценарій від заявки до чека	підтвердження працездатності основного потоку
CI/CD-перевірки	lint, typecheck, build, автотести	готовність коду до розгортання
Моніторинг	latency API, error rate, queue lag, DB health	виявлення деградації та збоїв
Безпека	залежності, конфігурації, доступ до файлів	зниження ризику несанкціонованого доступу
Експлуатація	restart сервісів, перевірка логів, backup/restore	відновлення роботи після типових збоїв

Окрему увагу приділено критичним сценаріям: створенню замовлення, автокалькуляції, фіксації суми, Web3-верифікації платежу, генерації чека та формуванню proof-of-completion. Саме ці частини системи мають найбільший вплив

на коректність бізнес-процесу, тому вони повинні покриватися тестами й контролюватися через журнали подій та метрики.

У результаті реалізація супроводжується не лише описом програмних модулів, а й набором практичних засобів перевірки, моніторингу та підтримки. Це підвищує надійність прототипу й створює основу для подальшого розгортання системи в реальному середовищі.

У третьому розділі описано програмні засоби реалізації AutoAssist+: обраний технологічний стек, локальне середовище розробки, архітектуру системи, структуру репозиторію, основні API-групи, принципи транзакційності, зберігання даних, Web3-верифікацію платежів, формування підтвердних документів і засоби забезпечення якості.

Реалізація побудована навколо повного життєвого циклу сервісного замовлення: створення заявки, автокалькуляції вартості, фіксації суми, підтвердження платежу, генерації PDF-чека та формування proof-of-completion. Для підтримки стабільності системи передбачено перевірку API-контрактів, тестування критичних сценаріїв, контроль станів замовлення, моніторинг основних метрик і базові експлуатаційні процедури.

4 РОЗРОБКА ТА ТЕСТУВАННЯ ДОДАТКУ AUTOASSIST+

4.1 Розробка інтерфейсу

4.1.1 Концепція UX/UI для AutoAssist+

Метою інтерфейсу є забезпечення швидкої та інтуїтивної взаємодії клієнта й сервіс-центру під час створення замовлення, отримання оцінки, фіксації суми (lock), виконання оплати, формування чека та доказу виконання (proof). Логіка сценарію організована як послідовність кроків.

Картографічний модуль реалізує геопошук сервіс-центрів поруч із місцем події з використанням Leaflet, відображаючи найближчі центри за відстанню та пріоритетом. Адміністративна панель (OrdersBoard) забезпечує контроль замовлень із фільтрацією за статусами та ключовими атрибутами. Для доступності передбачено валідацію полів (телефон/e-mail/VIN), застосування ARIA-атрибутів, підтримку клавіатурної навігації та узгоджені повідомлення про помилки.

У межах інтерфейсу важливо було не лише розмістити окремі екрани, а й забезпечити зрозумілий перехід між ними. Користувач повинен послідовно пройти шлях від створення заявки до отримання підтвердних документів без зайвих ручних дій. Тому основний UX-флоу побудовано навколо станів замовлення та дій, які доступні тільки на відповідному етапі.

Такий підхід зменшує кількість помилкових сценаріїв: оплату не можна виконати до фіксації суми, чек не формується до підтвердження платежу, а proof створюється лише після завершення необхідних дій. Узагальнену логіку основного користувацького сценарію наведено на рисунку 4.1.

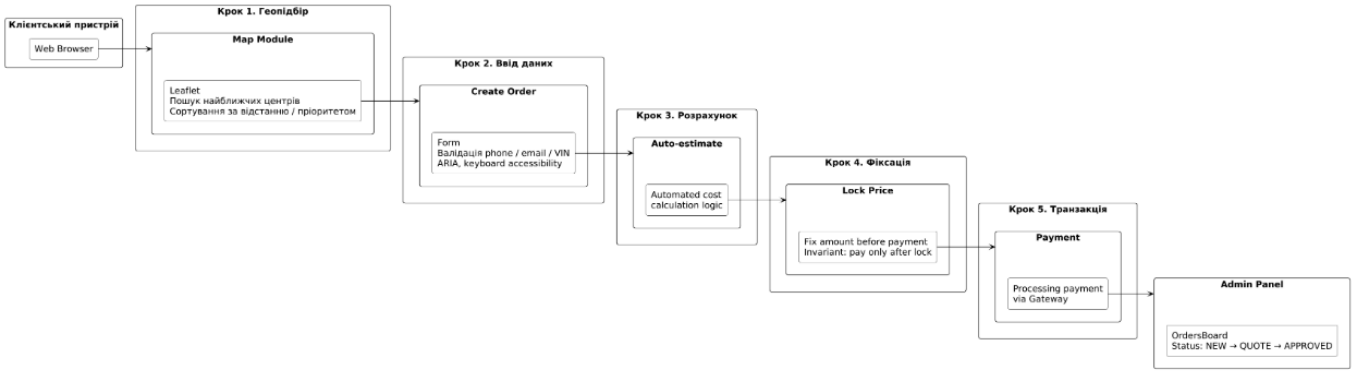


Рис. 4.1 Логіка основного UX-флоу (Create → Auto-estimate → Lock → Pay → Receipt/Proof)

Пояснення. Представлено послідовність ключових кроків, які формують цілісний користувацький сценарій. Така структура мінімізує когнітивне навантаження, забезпечує керовані переходи між станами та створює основу для контролю інваріантів, зокрема фіксації суми перед оплатою.

4.1.2 Реалізація у фронтенді

Фронтенд реалізовано на базі React, TypeScript і Vite. Основними частинами інтерфейсу є карта сервісних центрів, форма створення замовлення, адміністративна панель замовлень, сторінки керування оцінками та модуль сповіщень у реальному часі.

Інтеграція з API виконується через API_BASE з import.meta.env. JWT зберігається у cookie або у сховищі клієнта як aa_token (залежно від режиму авторизації). Вхід гаманцем реалізовано через SIWE: отримання nonce → підпис повідомлення → verify та отримання токенів (ендпойнти /api/wallet/nonce, /api/wallet/verify).

Завантаження вкладень реалізовано через presigned URL: клієнт отримує підписане посилання від бекенду та виконує прямий upload у MinIO; після завершення завантаження створюється запис Attachment, а воркер формує прев'ю (за потреби).

4.1.3 Сценарії використання інтерфейсу

Сценарій створення замовлення включає вибір точки на мапі, заповнення мінімальних полів, додавання фото/файлів та отримання первинної оцінки (estimate). Після виконання lock доступна ініціація Web3-оплати. Адміністративний сценарій включає фільтрацію списку замовлень, зміну статусів, призначення виконавця та взаємодію зі сповіщеннями у реальному часі.

Для узагальнення екранних сценаріїв було виділено ключові дії користувача, послідовність переходів між екранами та очікуваний результат після виконання кожної дії. Такий формат дає змогу швидко перевірити, чи відповідає інтерфейс основному бізнес-процесу платформи та чи всі важливі стани замовлення відображаються для користувача зрозуміло.

Таблиця 4.1

Екранні сценарії (UI) та очікувані результати

Сценарій	Кроки	Очікуваний результат
Створення замовлення	Map → OrderForm → Submit	Order створено, статус NEW
Автокалькуляція	Auto-estimate	Estimate DRAFT, показано total
Lock	Lock price	Estimate/Order переходить у LOCKED
Web3 Pay	Підпис/tx у MetaMask	Отримання txHash, стан Pending
Verify	POST verify	Payment VERIFIED, доступний receipt/proof

4.2 Створення бази даних

4.2.1 Вибір СКБД, ORM і підходи

Для зберігання даних використовується PostgreSQL, а доступ до даних реалізовано через Prisma. Така комбінація забезпечує транзакційність, контроль міграцій, типобезпечний клієнт та можливість зберігати частину структури у JSON-полях (наприклад, payload таймлайна або proof). Геозапити в межах MVP виконуються за формулою Гаверсина з можливістю подальшого переходу на PostGIS.

4.2.2 Основні сутності моделі даних

Ключовими сутностями є User, Order, Estimate, EstimateItem, Payment, Receipt, Proof, ServiceCenter, Attachment, Notification, OrderTimeline.

4.2.3 Фрагмент схеми Prisma

```
model Order {
 id            Int          @id @default(autoincrement())
  clientId      Int
  status        OrderStatus  @default(NEW)
  coords        Json?
  totalAmount   Decimal?     @db.Decimal(18, 2)
  currency      String?      @db.VarChar(8)
  createdAt     DateTime     @default(now())
  updatedAt     DateTime     @updatedAt


  estimates     Estimate[]
  payments      Payment[]
  receipt       Receipt?
  proof         Proof?
  timeline      OrderTimeline[]
}


model Payment {
  id        Int           @id @default(autoincrement())
  orderId   Int
  type      PaymentType
  amount    Decimal       @db.Decimal(18, 2)
  currency  String        @db.VarChar(8)
  txHash    String?       @unique
  chainId   Int?
  status    PaymentStatus @default(PENDING)
  createdAt DateTime      @default(now())


  order     Order         @relation(fields: [orderId], references: [id])
}


model Proof {
  id            Int      @id @default(autoincrement())
  orderId       Int      @unique
  proofHash     String   @unique
  reportJsonKey String
  createdAt     DateTime @default(now())


  order         Order    @relation(fields: [orderId], references: [id])
}
```

Рис. 4.2 Фрагмент Prisma-схеми для сутностей Order, Payment і Proof

Пояснення. Моделі відображають зв’язки замовлення з оплатами, чеком і доказом виконання. Унікальність txHash та proofHash забезпечує захист від повторного зарахування та дублювання доказів.

4.2.4 Процес створення міграцій

У розробці міграції створюються та застосовуються локально командою prisma migrate dev. У промисловому середовищі застосовуються лише затверджені міграції командою prisma migrate deploy. Для ризикових змін даних використовуються окремі job-скрипти з idempotent-логікою та перевірками до/після виконання.

4.2.5 Резервне копіювання та відновлення даних

```
$ts = (Get-Date).ToString('yyyyMMddHHmmss')
pg_dump -h <host> -U postgres -Fc -f "autoassist-backup-$ts.dump" autoassist


mc cp "autoassist-backup-$ts.dump" s3/backup/


pg_restore -h <host> -U postgres -d autoassist -v "autoassist-backup-20251102090000.dump"
```

Рис. 4.3 Приклад скрипту резервного копіювання та відновлення

Пояснення. Використовується формат custom (-Fc), що дозволяє відновлювати дані вибірково та забезпечує сумісність із pg_restore. Оффсайт-копіювання підвищує стійкість до втрати локальної інфраструктури.

4.3 Серверна логіка, черги та web3

Серверна частина відповідає за обробку замовлень, розрахунок оцінки вартості, фіксацію платежів, генерацію документів, роботу з файлами та синхронізацію Web3-операцій із внутрішнім станом системи. Логіка реалізована пошарово: маршрути приймають запити, сервіси виконують бізнес-операції, репозиторний рівень працює з базою даних, а фонові задачі використовуються для асинхронної обробки подій.

4.3.1 Модулі

Серверна частина організована за шарами:

```
- API layer: routes/*.ts;
- services: orders.service.ts, estimates.service.ts, payments.service.ts, receipts.service.ts, proof.service.ts, walletLogin.service.ts;
- queues: BullMQ workers (attachments previews, email notifications, receipts generation);
- storage: MinIO (presigned URLs);
- web3: ethers.js, SIWE (nonce → sign → verify).
```

Рис. 4.4 Шарова організація серверної частини платформи AutoAssist+

4.3.2 Ідемпотентність (патерн)

Критичні POST-операції (lock, verify-payment, генерація receipt) виконуються ідемпотентно через ключ idempotencyKey. У БД зберігається факт виконання операції для запобігання дублюванню ефектів при повторних запитах.

Фрагмент: перевірка lock (приклад сервісу)

```
if (estimate.status === "LOCKED") {
  throw makeError("ALREADY_LOCKED", "Estimate already locked")
}
```

Рис. 4.5 Перевірка стану замовлення перед фіксацією (LOCKED)

4.3.3 Потік Web3-оплати (логіка)

Потік Web3-оплати реалізовано так:

– клієнт ініціює транзакцію у гаманці та отримує txHash;

– виконується POST /api/payments/web3/verify { orderId, txHash, chainId };

– бекенд викликає getTransactionReceipt(txHash), перевіряє chainId, адресу призначення та суму;

– очікуються REQUIRED_CONFIRMATIONS з таймаутом VERIFY_TIMEOUT_SEC;

– payment.status змінюється на VERIFIED, після чого генерується Receipt (PDF) і Proof.

4.3.4 Формування proof-доказу

```
{
  "orderId": 123,
  "completedAt": "2025-11-02T10:12:34Z",
  "evidence": {
    "photos": ["https://minio/.../1.jpg"],
    "coords": { "lat": 50.45, "lng": 30.523 }
  },
  "performedBy": { "id": 12, "role": "mechanic" }
}
Hash: sha256Hex(canonicalJson(proof)).
```

Рис. 4.6 Приклад canonical proof JSON

Пояснення. Канонізація забезпечує детермінований порядок полів і однаковий результат хешування для ідентичних даних. SHA-256 використовується як компактний ідентифікатор доказу для контролю незмінності.

4.4 Контроль версій і CI/CD

Для супроводу розробки використовується контроль версій на основі Git. Зміни вносяться через окремі гілки, після чого перевіряються перед злиттям в основну гілку проєкту. Такий підхід дозволяє відстежувати історію змін, швидко знаходити причину помилок і зменшувати ризик потрапляння нестабільного коду до робочої версії системи.

4.4.1 Git та PR процес

Розробка виконується через гілки feature/* та fix/* з подальшим pull request. Перед злиттям обов’язково виконуються CI-перевірки. Для трасувальності у PR вказуються ідентифікатори вимог FR-xx/NFR-xx.

4.4.2 CI-перевірки

```
- install залежностей;
- статичний аналіз (eslint, prettier --check, tsc);
- unit-тести;
- build;
- integration smoke;
- OpenAPI validate;
- опціональна публікація артефактів.
- name: Validate OpenAPI
  run: npx @apidevtools/swagger-cli validate apps/api/src/openapi.json
```

Рис. 4.7 Фрагмент CI-кроку валідації OpenAPI

Пояснення. Контрактна дисципліна забезпечує відповідність реалізації задекларованим схемам і зменшує ризик несумісностей між фронтендом та бекендом.

4.4.3 CD & deployment

Для розгортання застосовуються multi-stage Dockerfiles та Helm-чарти. Міграції виконуються prisma migrate deploy. Відкат здійснюється helm rollback або поверненням попереднього тегу образів; за потреби виконується відновлення БД із резервної копії.

У результаті контроль версій і CI/CD-процес забезпечують керовану розробку, повторюваність перевірок і зниження ризику помилок під час розгортання. Для AutoAssist+ це особливо важливо, оскільки система містить взаємопов’язані модулі замовлень, оплат, документів, Web3-перевірки та зберігання файлів.

4.5 Тестування програмного забезпечення

4.5.1 Стратегія тестування

Застосовується піраміда тестування: unit → integration → E2E; performance-тести виконуються окремо. Для критичних модулів контролюється покриття та стабільність виконання тестів у CI.

4.5.2 Unit-тести (Jest)

Перевіряються функції autoCalculateEstimate, canonicalJson, sha256Hex та схеми валідації (Zod), зокрема граничні значення, округлення та обробка помилок.

4.5.3 Інтеграційні тести (Jest + Supertest)

Набори тестів покривають основні флоу: create → estimate → lock; verify txHash → receipt; детермінізм proof hash; геопошук; SIWE nonce/verify. У назві тесту вказується пов’язаний FR-ID для трасувальності.

4.5.4 E2E (Playwright/Cypress)

E2E-сценарій відтворює повний шлях користувача: створення замовлення, завантаження вкладень, отримання estimate, lock, симуляція платежу, verify та перегляд receipt/proof в інтерфейсі.

4.5.5 Performance (k6) – приклад

```
import http from "k6/http";
import { check } from "k6";


export let options = { vus: 50, duration: "2m" };


export default function () {
  const payload = JSON.stringify({ /* minimal order */ });
  const res = http.post("http://localhost:8080/api/estimates", payload, {
    headers: { "Content-Type": "application/json" },
  });
  check(res, { "status is 201": (r) => r.status === 201 });
}
```

Рис. 4.8 Приклад k6-скрипту для навантажувального тестування

Пояснення. Скрипт моделює типову операцію автокалькуляції та дозволяє отримати p50/p95 затримки, error rate і throughput. Результати порівнюються з цільовими NFR.

4.5.6 Таблиці тест-кейсів (розширені витяги)

Таблиця 4.3

Авторизація (SIWE + JWT)

Крок	Дані	Очікуваний результат
GET nonce	–	200, nonce, nonce TTL активний
POST verify	siweMessage, signature	200, видача токенів
Негативний	nonce повторно	400, помилка REPLAY_NONCE
Негативний	chainId не 80002	400, помилка WRONG_CHAIN

Таблиця 4.4

Створення замовлення та оцінка

Крок	Дані	Очікуваний результат
Create order	мін. поля + coords	201, orderId, status=NEW
Auto-estimate	orderId, profile	201, items[], total, status=DRAFT
Lock	orderId	200, status=LOCKED
Негативний lock	повторний lock	409, ALREADY_LOCKED

Таблиця 4.5

Web3 verify

Крок	Дані	Очікуваний результат
Verify	orderId, txHash, chainId	200, VERIFIED після N confirmations
Негативний	txHash не знайдено	404, TX_NOT_FOUND
Негативний	сума ≠ lock	409, AMOUNT_MISMATCH
Негативний	таймаут	408/409, TX_TIMEOUT

4.6 Інструкція користувача

4.6.1 Запуск локального середовища

Вимоги: Node.js 18+, Docker Desktop, pnpm. Локальний запуск виконується через docker compose та запуск модулів у dev-режимі.

```
cd c:\IT\projects\diplom\autoassist
pnpm install
cd apps/api
npx prisma generate
docker compose -f infra/compose/docker-compose.yml up --build
pnpm --filter @autoassist/api test
Health: GET http://localhost:8080/api/healthz → { ok: true }.
```

Рис. 4.9 Послідовність команд для запуску локального середовища

4.6.2 Сценарії користувача

Основні сценарії роботи з платформою поділяються за ролями. Клієнт створює замовлення, отримує оцінку вартості, фіксує суму та виконує оплату. Персонал сервісу працює із замовленнями, змінює статуси та додає підтвердження виконання. Адміністратор контролює довідники, профілі калькуляції та загальний стан системи.

4.6.3 Типові помилки під час роботи

Під час локального запуску або використання системи можуть виникати типові помилки: недоступність контейнерів, неправильна адреса API, відсутність підключення до MinIO або помилка Web3-верифікації. Для таких випадків передбачено перевірку health-check, журналів сервера, статусу контейнерів і повідомлень інтерфейсу.

4.7 Моніторинг, логування, безпека, приватність

Використовується Prometheus для метрик (затримки, помилки, розміри черг), Grafana для візуалізації та Alertmanager для сповіщень. Логування виконується у форматі JSON із correlationId/traceId. Трасування реалізується через OpenTelemetry для HTTP та worker-задач. Безпека забезпечується TLS, JWT із коротким TTL та refresh, SIWE з nonce TTL та rate-limit, а секрети зберігаються у Vault/K8s sealed-secrets.

Для експлуатаційного контролю системи додатково визначено набір контрольних метрик, подій аудиту та операційних алертів, що використовуються під час супроводу, локалізації інцидентів і перевірки стабільності після релізу.

Таблиця 4.6

Ключові метрики моніторингу

Метрика	Джерело	Цільове значення	Призначення
p95 latency /api/orders	Prometheus HTTP metrics	≤ 300 мс	Контроль швидкості створення та перегляду замовлень
p95 latency /api/payments/web3/ verify	Prometheus HTTP metrics	≤ 800 мс без очікування N conf.	Оцінка швидкодії перевірки транзакції
Queue lag previews	BullMQ / Redis	≤ 30 с	Контроль своєчасного формування прев'ю вкладень
Queue lag receipts	BullMQ / Redis	≤ 15 с	Контроль генерації PDF-чеків після підтвердження оплати
Error rate 5xx	Prometheus + logs	< 1 %	Виявлення деградації API або зовнішніх залежностей
DB health	PostgreSQL	active	перевірка доступності основного сховища даних

Окрему увагу приділено журналюванню подій, які впливають на фінансовий стан замовлення або підтвердження виконання робіт. Такі події використовуються для аудиту, відновлення послідовності дій і перевірки правильності переходів між станами.

Таке журналювання дозволяє відстежити повний шлях замовлення та швидко знайти етап, на якому виникла помилка.

Для критичних подій зберігаються дані про замовлення, користувача, час виконання та traceId.

Таблиця 4.7

Події аудиту та журналювання

Подія	Коли фіксується	Основні атрибути	Практичне значення
ORDER_CREATED	Після успішного POST /orders	orderId, clientId, coords, traceId	Підтверджує факт створення заявки
ESTIMATE_LOCKED	Після lock оцінки	orderId, estimateId, total, actorId	Фіксує суму, з якою пов’язується платіж
PAYMENT_VERIFIED	Після валідації txHash	orderId, paymentId, txHash, chainId	Підтверджує завершення Web3-платежу
RECEIPT_ISSUED	Після генерації PDF	receiptId, orderId, storageKey	Дозволяє відстежити випуск чеку та посилання
PROOF_CREATED	Після формування proof	orderId, proofHash, reportJsonKey	Підтверджує створення цифрового доказу

Для експлуатаційного контролю системи додатково визначено набір операційних алертів. Вони використовуються для швидкого виявлення технічних проблем під час роботи платформи: затримок Web3-верифікації, помилок генерації

чеків, проблем із завантаженням вкладень або зростання кількості серверних помилок.

Кожен алерт має умову спрацювання, пріоритет і базову дію реагування. Це дозволяє швидко визначити проблемний компонент системи та виконати першочергові дії для відновлення стабільної роботи AutoAssist+.

Особливо важливо контролювати ті події, які впливають на завершення основного сценарію користувача: підтвердження платежу, формування PDF-чека, створення proof-документа та доступ до вкладень. Саме ці точки є критичними для перевірки коректності реалізації й подальшого супроводу системи.

Таблиця 4.8

Операційні алерти та базові дії реагування

Алерт	Умова спрацювання	Пріоритет	Базова дія
Web3 verify timeout	Частка TX_TIMEOUT > 5 % за 15 хв	High	Перевірити RPC-провайдер, таймаути та повторні спроби
Receipt generation lag	Черга receipts > 20 задач або lag > 60 с	Medium	Перевірити worker, Redis, MinIO та доступ до сховища
Attachment upload failures	Помилки presigned upload > 3 %	Medium	Перевірити MinIO, TTL посилань, типи і розміри файлів
API 5xx spike	Error rate 5xx > 1 % за 10 хв	High	Переглянути логи API, останній реліз і зовнішні залежності
DB unavailable	health-check БД не проходить	High	перевірити PostgreSQL, міграції та підключення

4.8 Runbook розгортання, відкату та відновлення

Runbook містить release-чекліст, порядок резервного копіювання БД і сховища, виконання міграцій, smoke-перевірки, моніторинг метрик після релізу та процедури відкату. У DR-процедурах визначено регулярність дампів, офсайт-копії та контроль виконання цільових RTO/RPO.

Runbook використовується як коротка інструкція для типових ситуацій під час супроводу системи. У ньому фіксується не лише дія адміністратора або розробника, а й очікуваний результат після виконання цієї дії. Це дає змогу швидше локалізувати проблему та не виконувати зайві дії під час інциденту.

Для AutoAssist+ до базових сценаріїв runbook належать перевірка доступності API, контроль стану PostgreSQL і Prisma-міграцій, перевірка MinIO та presigned URL, аналіз черг Redis/BullMQ, діагностика Web3-верифікації за txHash і chainId, а також перевірка генерації PDF-чека та proof-of-completion.

4.9 Практичні фрагменти та шаблони реалізації

Окрему увагу приділено тим фрагментам, які безпосередньо впливають на коректність основного бізнес-процесу. До них належать перевірка Web3-платежу, узгодження суми замовлення перед оплатою, фіксація результату транзакції, створення чека та збереження proofHash. Саме ці частини мають бути формалізовані, оскільки помилка в них може призвести до неправильного статусу замовлення або дублювання платіжної операції.

Практичні шаблони також виконують роль допоміжної документації для подальшого супроводу системи. За їх допомогою можна швидко зрозуміти, які дані передаються між модулями, які відповіді очікуються від API та які зміни вносяться

до структури бази даних. Це спрощує перевірку реалізації, написання тестів і внесення змін без порушення вже описаної логіки роботи .

У межах цього підрозділу наведено три типи матеріалів. Перший демонструє контракт API для маршруту перевірки Web3-платежу. Другий описує послідовність взаємодії між клієнтом, API, платіжним сервісом і blockchain node. Третій показує приклад міграції бази даних, необхідної для збереження хешу доказу виконання робіт.

Такий набір фрагментів дозволяє показати не лише загальну архітектуру системи, а й конкретну реалізацію критичних операцій. Особливу роль тут відіграє перевірка Web3-платежу, оскільки саме вона підтверджує факт оплати та впливає на подальший статус замовлення.

4.9.1 OpenAPI фрагмент

```
"/api/payments/web3/verify": {
  "post": {
    "summary": "Verify Web3 payment by txHash",
    "requestBody": {
      "required": true,
      "content": {
        "application/json": {
          "schema": {
            "type": "object",
            "properties": {
              "orderId": { "type": "integer" },
              "txHash":  { "type": "string"  },
              "chainId": { "type": "integer" }
            },
            "required": ["orderId", "txHash"]
          }
        }
      }
    },
    "responses": {
      "200": { "description": "Payment verified" },
      "400": { "description": "Invalid request" },
      "404": { "description": "Order or tx not found" }
    }
  }
}
```

Рис. 4.10 Фрагмент OpenAPI-специфікації маршруту перевірки Web3-платежу

4.9.2 PlantUML-опис послідовності estimate → lock → pay

Послідовність estimate → lock → pay описує основний сценарій роботи із замовленням після створення заявки. Спочатку клієнт надсилає запит на автоматичний розрахунок вартості, після чого система визначає суму через EstimateService. Далі користувач підтверджує розрахунок, і сума фіксується, щоб вона не змінювалася під час оплати.

Після фіксації суми виконується платіж. PaymentService перевіряє транзакцію через зовнішній Blockchain RPC та повертає результат перевірки в API. Якщо платіж підтверджено, замовлення переходить у наступний статус, а система може сформувати підтвердження оплати.

```
@startuml
 actor Client
 participant "API" as API
 participant "EstimateService" as ES
 participant "PaymentService" as PS
 participant "Blockchain RPC" as BC


 Client -> API: POST /api/estimates/auto
 API -> ES: autoCalculateEstimate(orderId, profile)
 ES --> API: estimate(total, DRAFT)
 API --> Client: 201 estimate


 Client -> API: POST /api/estimates/{orderId}/lock
 API -> ES: lockEstimate(orderId)
 ES --> API: estimate(LOCKED)
 API --> Client: 200 LOCKED


 Client -> Wallet: send tx
 Wallet --> Client: txHash


 Client -> API: POST /api/payments/web3/verify
 API -> PS: verifyTx(orderId, txHash, chainId)
 PS -> BC: getTransactionReceipt(txHash)
 BC --> PS: receipt(confirmations, to, amount)
 PS --> API: payment(VERIFIED)
 API --> Client: 200 VERIFIED
 @enduml
```

Рис. 4.11 UML-опис послідовності автокалькуляції, фіксації суми та перевірки Web3-платежу

4.9.3 Фрагмент SQL-міграції для додавання proofHash

```
ALTER TABLE "Receipt" ADD COLUMN "proofHash" text;
```

Рис. 4.12 Фрагмент SQL-міграції для додавання поля proofHash у таблицю Receipt

Наведені фрагменти показують, як окремі частини системи пов’язані між собою на рівні контрактів, сценаріїв і структури даних. OpenAPI забезпечує узгодженість між фронтендом і сервером, PlantUML дозволяє перевірити послідовність бізнес-процесу, а міграція бази даних фіксує зміну, необхідну для збереження доказу виконання робіт.

У результаті ці шаблони можуть використовуватися як основа для подальшого супроводу, тестування та розширення. Вони також спрощують перевірку того, що реалізація відповідає вимогам, описаним у попередніх розділа

ВИСНОВКИ

У кваліфікаційній роботі розроблено веб-платформу AutoAssist+, яка призначена для автоматизації роботи із сервісними замовленнями. Система охоплює основні етапи обробки заявки: створення замовлення, оцінювання вартості, фіксацію суми перед оплатою, проведення платежу, формування чека та створення цифрового підтвердження виконання робіт. Під час розробки основну увагу приділено тому, щоб взаємодія між клієнтом, сервісним партнером і адміністратором була зрозумілою, швидкою та надійною.

У ході виконання роботи було проаналізовано підходи до побудови сучасних веб-систем і вибрано технології, які найкраще підходять для реалізації такого проєкту. Для клієнтської частини використано React і TypeScript, для серверної – Node.js, Prisma та PostgreSQL. Для зберігання файлів обрано MinIO, а для роботи з Web3-платежами – ethers.js. Такий набір інструментів дозволив реалізувати систему з чіткою структурою, зручною логікою роботи та можливістю подальшого розширення.

Також було розглянуто наявні рішення, які використовуються для автоматизації сервісних процесів. У результаті аналізу встановлено, що більшість таких систем або зосереджені лише на CRM-функціях, або не підтримують перевірку Web3-платежів і цифрове підтвердження результату виконання робіт. Саме тому доцільним стало створення платформи, яка поєднує роботу із замовленнями, геолокаційний підбір сервісних центрів, автоматичний розрахунок вартості та верифікацію платежів у межах одного застосунку.

У роботі реалізовано клієнтську частину системи, яка дає змогу створювати замовлення, вибирати локацію, додавати вкладення, переглядати результати

автокалькуляції та відстежувати зміни станів замовлення. Інтерфейс побудовано так, щоб основні дії були послідовними й зрозумілими для користувача. Окрему увагу приділено сценаріям оплати, перегляду чека та підтвердження виконання робіт.

Серверна частина забезпечує обробку замовлень, оцінок, платежів, файлів і документів. У системі реалізовано логіку формування PDF-чеків і створення цифрового доказу виконання робіт на основі канонічного JSON та SHA-256. Описано структуру бази даних, підхід до міграцій, резервного копіювання та відновлення. Це дозволило зробити систему більш цілісною та зменшити ризик помилок під час обробки даних.

Окремо розроблено підхід до тестування. Для перевірки працездатності застосунку використано модульні, інтеграційні, E2E та навантажувальні тести. Для цього застосовано Jest, Playwright та k6. Такий підхід дав змогу перевірити як окремі функції, так і повний ланцюг роботи системи – від створення замовлення до формування чека і proof.

У роботі також опрацьовано питання безпеки. Реалізовано механізми автентифікації на основі JWT і SIWE, керування доступом за ролями, захист від повторної обробки транзакцій і базові правила зберігання даних. Для інфраструктури передбачено підхід до керування секретами та загальні заходи захисту конфіденційної інформації.

Отже, поставлену мету кваліфікаційної роботи досягнуто, а визначені у вступі завдання виконано. У результаті створено працездатний прототип веб-платформи AutoAssist+, який може бути використаний як основа для подальшого розвитку повноцінної системи у сфері авто-послуг, логістики, страхових сервісів і технічної підтримки.

Подальше вдосконалення системи може бути пов’язане з розширенням алгоритмів підбору сервісних центрів, покращенням аналітичних можливостей,

підтримкою додаткових платіжних сценаріїв, розвитком механізмів цифрового підтвердження виконаних робіт та впровадженням системи в реальне виробниче середовище.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Mozilla Developer Network. Web APIs Documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ (дата звернення 04.05.2026).
2. React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/ (дата звернення 04.05.2026).
3. TypeScript Handbook [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/ (дата звернення 04.05.2026).
4. Prisma ORM Documentation [Електронний ресурс]. – Режим доступу: https://www.prisma.io/docs/ (дата звернення 04.05.2026).
5. PostgreSQL Global Development Group. PostgreSQL: Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення 04.05.2026).
6. MinIO Documentation [Електронний ресурс]. – Режим доступу: https://min.io/docs/ (дата звернення 04.05.2026).
7. ethers.js Documentation [Електронний ресурс]. – Режим доступу: https://docs.ethers.org/ (дата звернення 04.05.2026).
8. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/ (дата звернення 04.05.2026).
9. OpenAPI Initiative. OpenAPI Specification [Електронний ресурс]. – Режим доступу: https://spec.openapis.org/ (дата звернення 04.05.2026).
10. GitHub Actions Documentation [Електронний ресурс]. – Режим доступу: https://docs.github.com/en/actions/ (дата звернення 04.05.2026).
11. HashiCorp Vault Documentation [Електронний ресурс]. – Режим доступу: https://developer.hashicorp.com/vault/docs/ (дата звернення 04.05.2026).

12. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/ (дата звернення 04.05.2026).

13. Kubernetes Documentation [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/ (дата звернення 04.05.2026).

14. Grafana k6 Documentation [Електронний ресурс]. – Режим доступу: https://grafana.com/docs/k6/latest/ (дата звернення 04.05.2026).

15. Jest Documentation [Електронний ресурс]. – Режим доступу: https://jestjs.io/docs/ (дата звернення 04.05.2026).

16. Playwright Documentation [Електронний ресурс]. – Режим доступу: https://playwright.dev/docs/ (дата звернення 04.05.2026).

17. PlantUML Documentation [Електронний ресурс]. – Режим доступу: https://plantuml.com/ (дата звернення 04.05.2026).

18. Prometheus Documentation [Електронний ресурс]. – Режим доступу: https://prometheus.io/docs/ (дата звернення 04.05.2026).

19. OpenTelemetry Documentation [Електронний ресурс]. – Режим доступу: https://opentelemetry.io/docs/ (дата звернення 04.05.2026).

20. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

21. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston: Addison-Wesley, 2005. 496 p.

22. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 533 p.

23. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken: Wiley, 2011. 256 p.

24. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. New York: IEEE, 1998.

25. ISO/IEC/IEEE 29119-1:2013. Software and systems engineering – Software testing – Part 1: Concepts and definitions.

26. OWASP Foundation. OWASP Top 10: The Ten Most Critical Web Application Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/ (дата звернення 04.05.2026).

27. SpruceID. Sign-In with Ethereum [Електронний ресурс]. – Режим доступу: https://siwe.xyz/ (дата звернення 04.05.2026).

28. Socket.IO Documentation [Електронний ресурс]. – Режим доступу: https://socket.io/docs/ (дата звернення 04.05.2026).

29. Leaflet Documentation [Електронний ресурс]. – Режим доступу: https://leafletjs.com/reference.html (дата звернення 04.05.2026).

30. BullMQ Documentation [Електронний ресурс]. – Режим доступу: https://docs.bullmq.io/ (дата звернення 04.05.2026).

31. Сягровський П. Д. «Застосування Web3 та геолокаційних сервісів у платформі AutoAssist+ для цифровізації сервісних процесів» // VII Всеукраїнська науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. К.: ДУІКТ, 2026. С. 7–9.

32. Сягровський П. Д. «Веб-платформа AutoAssist+ для автоматизації сервісних замовлень із геолокаційним підбором та Web3-верифікацією платежів» // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026, С. 23-26.

33. Сягровський П. Д. «Інтеграція блокчейн-технологій у веб-платформі AutoAssist+ для верифікації сервісних платежів» // Всеукраїнська науково-технічна

конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026, (прийнято до друку).

34. Сягровський П. Д. «Інтелектуалізація обробки сервісних замовлень у веб-платформі AutoAssist+» // VI Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. К.: ДУІКТ, 2026, (прийнято до друку).

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку

Web-додаток для адміністрування та управління замовленнями з інтеграцією Web3-оплати та системою геолокаційного підбору сервісів

Виконав студент 4 курсу
Групи ТЦР-42
Сягровський Павло
Дмитрович
Керівник роботи
старший викладач Вадим Олегович Читулян

Київ-2026

AutoAssist+
Методологічна база проєкту

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета

Підвищення ефективності обробки замовлень, прозорості фінансової взаємодії та керованості процесу шляхом розробки платформи AutoAssist+.

Об'єкт

Процеси створення, обробки, супроводу та оплати сервісних замовлень у цифровому середовищі.

Предмет

Методи, моделі та програмні засоби побудови веб-платформи для автоматизації сервісних замовлень із Web3-верифікацією.

Обґрунтування: Використання Web3 та геолокації як фундаменту для підвищення рівня довіри та швидкості обробки заявок.

AutoAssist+

Послідовність виконання роботи

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

01 Проаналізувати існуючі CRM- та сервісні платформи, визначити їхні обмеження для сценаріїв авто-послуг.

02 Сформувати функціональні й нефункціональні вимоги до платформи AutoAssist+ та визначити межі MVP.

03 Спроєктувати архітектуру, модель даних, API-контракти та життєвий цикл сервісного замовлення.

04 Реалізувати модулі створення заявки, автокалькуляції вартості, геолокаційного підбору сервісного центру.

05 Реалізувати платіжний сценарій, Web3-верифікацію txHash, формування PDF-чека та proof-hash.

06 Провести перевірку ключових сценаріїв і оцінити працездатність розробленого MVP.

AutoAssist+

Порівняння підходів із платформою

АНАЛІЗ АНАЛОГІВ

Порівняння показує, що наявні рішення закривають лише окремі частини процесу, тоді як AutoAssist+ поєднує заявку, геопошук, оплату та цифрове підтвердження в одному сценарії.

Критерій	CRM-системи	Сервісні агрегатори	Web3-інструменти	AutoAssist+
Облік замовлень	повний	частковий	відсутній	**повний**
Геолокаційний підбір	обмежений	повний	відсутній	**повний**
Перевірка платежу	ні	ні	так	**так**
Чек і proof у процесі	частково	частково	ні	**так**
Ролі та бізнес-логіка	так	обмежено	ні	**так**

Перевага AutoAssist+ — повний цикл сервісного замовлення з прозорою оплатою та перевірюваними цифровими артефактами.

AutoAssist+

Функціональні та нефункціональні вимоги

ВИМОГИ ДО ПЛАТФОРМИ

Функціональні вимоги

1. Створення, перегляд і фільтрація сервісних замовлень із підтримкою статусів життєвого циклу заявки.
2. Геолокаційний підбір сервісних центрів у межах радіуса та сортування за відстанню.
3. Автоматизоване формування оцінки вартості та фіксація суми (lock-механізм).
4. Підтримка класичного платіжного сценарію та Web3-верифікації (chainId, сума, підтвердження).
5. Генерація PDF-чеків, proof-hash та безпечний доступ до вкладень за тимчасовими посиланнями.

Нефункціональні вимоги

1. Перевірка коректності обміну даними між клієнтською та серверною частинами системи
2. Надійність за рахунок транзакційності, ідемпотентності та використання черг повідомлень.
3. Безпека через рольову модель доступу (RBAC), JWT-аутентифікацію та підписані URL.
4. Масштабування завдяки контейнеризації та розподіленій архітектурі сервісів.
5. Спостережуваність: журнали подій, health-checks та збір метрик у реальному часі.

Вимоги сформовані з урахуванням сучасних стандартів безпеки Web3 та високих вимог до відмовостійкості корпоративних систем.

AutoAssist+

Використані технології та їх роль у платформі

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Frontend & Core

- **React + Vite:** інтерфейс замовлень, профілю та оплати.
- **TypeScript:** типізація та стабільність API.

Backend & Logic

- **Node.js + Express:** серверна бізнес-логіка.
- **Socket.IO:** оновлення статусів у реальному часі.

Database & ORM

- **PostgreSQL:** основне сховище даних.
- **Prisma ORM:** типобезпечні моделі та міграції.

Storage & Tasks

- **MinIO:** PDF-чеки та медіа-вкладення.
- **Redis + BullMQ:** черги та фонові задачі.

Web3 & Blockchain

- **ethers.js:** верифікація txHash та chainId.
- **Polygon Amoy:** тестове середовище Web3.

Infra & Ops

- **Docker Compose:** локальне розгортання.
- **Prometheus + Grafana:** моніторинг метрик.

Усі компоненти обрано для підтримки повного циклу заявки: створення → оцінка → оплата → чек → proof.

Що вирішує AutoAssist+

ПОСТАНОВКА ЗАДАЧІ

1. Єдиний цифровий сценарій для сервісної заявки від створення до завершення.
2. Прозора оплата: класичний сценарій та Web3-підтвердження транзакції.
3. Цифрові артефакти після оплати: PDF-чек, QR-посилання, доказ виконання.

Ключова ідея

Система не лише зберігає дані в CRM, а й формує перевірювані цифрові результати кожного замовлення.

Наскрізний UX-flow: Create → Auto-estimate → Lock → Pay → Receipt/Proof

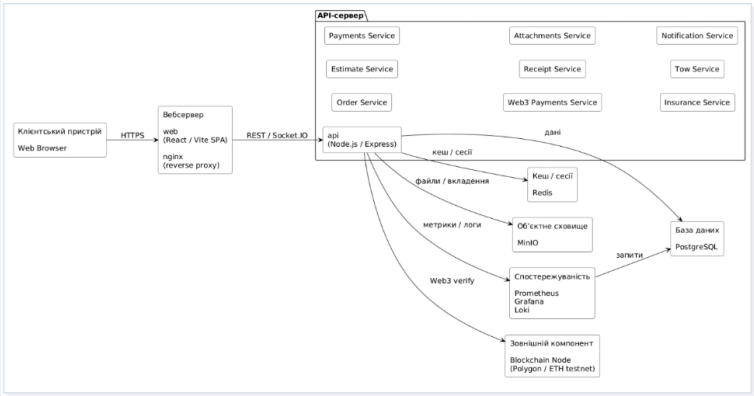
Компонентна модель рішення

АРХІТЕКТУРА

1. SPA-клієнт на React/Vite взаємодіє з API через REST та Socket.IO.
2. Серверна частина реалізує модулі Orders, Estimates, Payments, Receipts, Attachments, Tow та Insurance.
3. Дані зберігаються у PostgreSQL, кеш і черги працюють через Redis, файли — у MinIO.

Стек

React · Node.js · Prisma · PostgreSQL · Redis · MinIO · Polygon Amoy



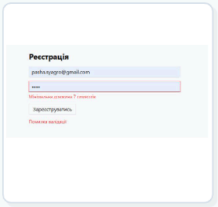
Клієнтський інтерфейс, API-модулі, база даних, кеш, сховище та блокчейн-нода працюють як єдина інфраструктура.

AutoAssist+

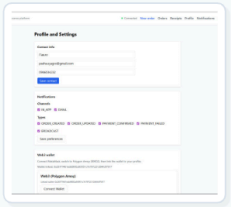
Клієнтський кабінет і створення заявки

ДЕМОНСТРАЦІЯ ІНТЕРФЕЙСУ

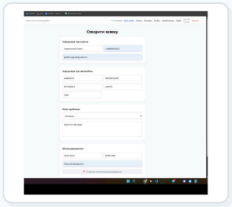
1. Авторизація підтримує класичний вхід та сценарій входу через гаманець.
2. У профілі зберігаються контактні дані, канали сповіщень та Web3-wallet.
3. Форма замовлення об’єднує дані клієнта, авто, опис проблеми та геолокацію.



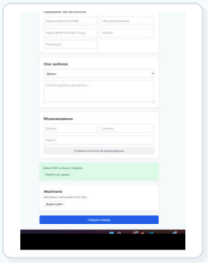
Авторизація



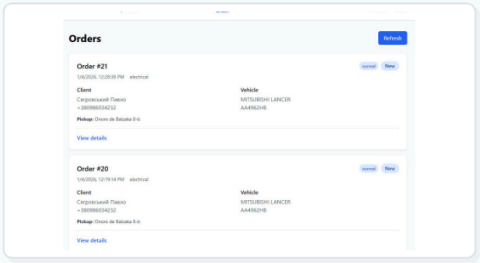
Профіль і сповіщення



Нова заявка



Успішне створення



Список замовлень користувача

AutoAssist+

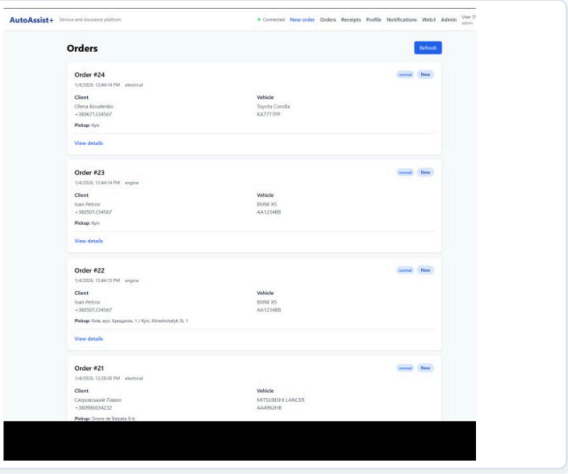
Контроль життєвого циклу замовлення

ОБРОБКА ЗАМОВЛЕННЯ

1. Замовлення проходить через статуси та залишається доступним як користувачу, так і персоналу сервісу.
2. Адміністративна панель дозволяє переглядати заявки, їх атрибути та поточний стан.
3. Усі дії готують систему до наступного кроку — оцінки та оплати.

Admin view

Сторінка адміністрування концентрує поточні заявки та дає змогу швидко переходити до деталей конкретного замовлення.



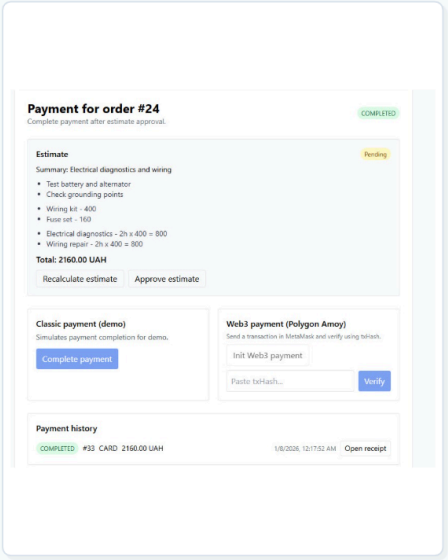
Панель замовлень для персоналу / адміністратора

AutoAssist+

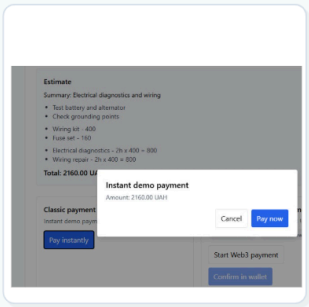
Сторінка оплати та альтернативні сценарії

ОПЛАТА

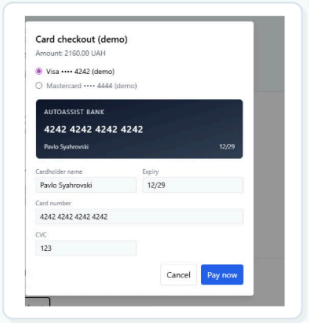
1. Перед оплатою користувач бачить estimate та підсумкову суму по замовленню.
2. Підтримуються три варіанти: demo-класична оплата, card checkout та Web3-сценарій.
3. Стани payment history показують, коли транзакція вже проведена або очікує підтвердження.



Головна сторінка оплати



Миттєва demo-оплата



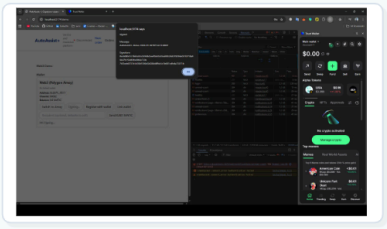
Сценарій card checkout

AutoAssist+

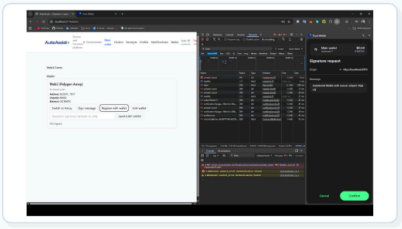
Підключення гаманця, мережа та підпис

WEB3-СЦЕНАРІЙ

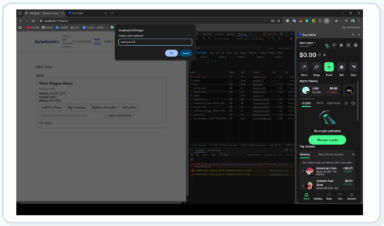
1. Користувач підключає гаманець та надає дозвіл на взаємодію із застосунком.
2. Після перемикання на потрібну мережу виконується підпис службового повідомлення.
3. Цей сценарій використовується для безпечної прив'язки wallet та подальшої Web3-оплати.



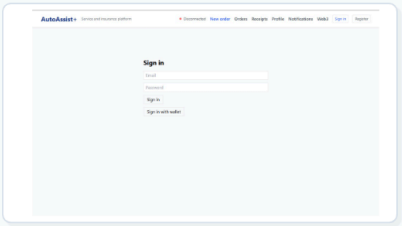
Ініціалізація гаманця



Дозвіл на підключення



Wallet connected



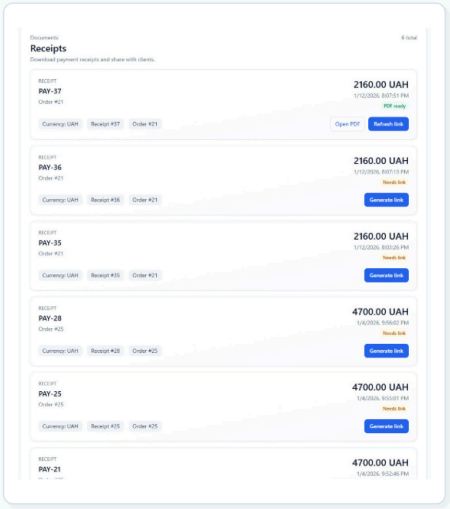
Підпис повідомлення

AutoAssist+

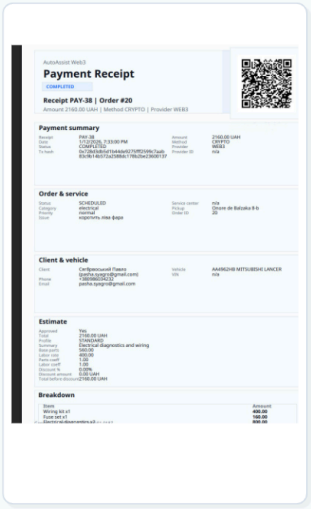
Що отримує користувач після підтвердження оплати

АРТЕФАКТИ

1. Список квитанцій містить історію платежів та кнопки генерації або оновлення посилання.
2. PDF-чек містить реквізити, статус, QR-код і параметри замовлення.
3. У випадку Web3-оплати в чеку фіксується txHash, що дозволяє перевірити транзакцію.



Список квитанцій



PDF-чек Web3

АПРОБАЦІЯ РЕЗУЛЬТАТІВ

1. Сягровський П. Д. «Застосування Web3 та геолокаційних сервісів у платформі AutoAssist+ для цифровізації сервісних процесів» // VII Всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IoT». Збірник тез. К.: ДУІКТ, 2026. С. 7–9.
2. Сягровський П. Д. «Веб-платформа AutoAssist+ для автоматизації сервісних замовлень із геолокаційним підбором та Web3-верифікацією платежів» // VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026, С. 23-26.
3. Сягровський П. Д. «Інтеграція блокчейн-технологій у веб-платформі AutoAssist+ для верифікації сервісних платежів» // Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026, (прийнято до друку).
4. Сягровський П. Д. «Інтелектуалізація обробки сервісних замовлень у веб-платформі AutoAssist+» // VI Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. К.: ДУІКТ, 2026, (прийнято до друку).

ВИСНОВКИ

01 Проведено **аналіз існуючих CRM- та сервісних платформ**, що дозволило виявити обмеження традиційних рішень у частині геолокаційного підбору виконавців та прозорої перевірки платежів.

02 Розроблено **архітектуру платформи AutoAssist+**, яка об'єднує інтерфейс, бізнес-логіку та модулі обробки замовлень, платежів і цифрового підтвердження робіт.

03 Реалізовано механізм **геолокаційного підбору** сервісних центрів, що забезпечує пошук найближчих виконавців і підвищує оперативність обробки заявок.

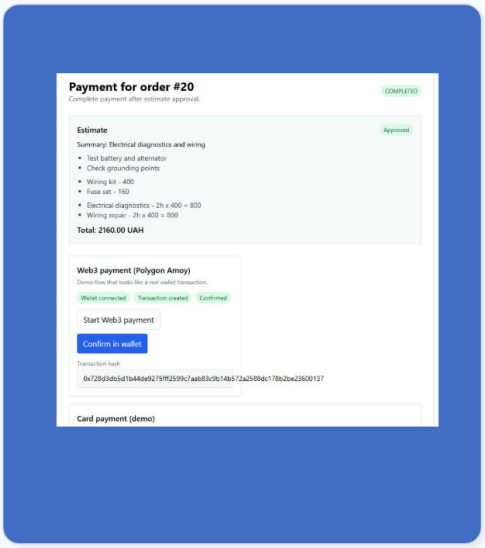
04 Впроваджено модуль **автоматизованої оцінки вартості** перед оплатою, що дозволяє впорядкувати процес і зменшити кількість ручних дій.

05 Реалізовано **Web3-верифікацію платежів** (txHash) та формування PDF-чека і proof-hash як цифрового підтвердження виконання робіт.

06 **Практична цінність:** використання платформи у сфері авто-послуг, логістики та страхування для швидкості, прозорості та контролю процесів.

Дякую за увагу

AutoAssist+
Питання та відповіді



Ключовий результат

Працюючий MVP з повним циклом заявки та підтримкою Web3-оплати