Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веборієнтованої системи для пункту прокату туристичного спорядження

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Склярова Н.Р.
(підпис) (прізвище та ініціали)

Керівник _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Склярова Надія Русланівна
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веборієнтованої системи для пункту прокату туристичного спорядження

Керівник роботи Боднарчук Ігор Орестович, кандидат технічних наук, зав. кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *травня* 20*26* року № *4/9-239*

2. Термін подання студентом завершеної роботи 22 червня 2026 р.

3. Вихідні дані до роботи Дані про спорядження в пункті прокату

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1 Теоретичні засади та аналіз предметної області. 1.1 Огляд літературних Джерел та пов'язаних досліджень. 1.2 Аналіз проблеми та постановка завдання. Розділ 2 Проєктування Інформаційної системи. 2.1 Методологічні підходи до розроблення програмного забезпечення. 2.2 Моделювання даних: ER діаграми та структура бази даних. 2.3 Планування, ризики та життєвий цикл проєкту. 2.3.1 Планування технічних та людських ресурсів. 2.3.2 Управління ризиками та стратегії їх мінімізації. 2.3.3 Модель життєвого циклу Проєкту та графік виконання (діаграма Ганта). Розділ 3 Практична реалізація та аналіз Результатів. 3.1 Загальна логіка роботи системи та реалізація серверної частини (backend). 3.2 Реалізація клієнтської частини (frontend): інтерфейси та функції. 3.3 Тестування, Перевірка доступності та обробка помилок. Розділ 4 Безпека життєдіяльності, основи Охорони праці. 4.1 Ергономічні проблеми безпеки життєдіяльності. 4.2 Етико-правове Регулювання та стандарти доступності в інформаційних технологіях. Висновки. Перелік джерел. Додаток А. Додаток Б. Додаток В. Додаток Г. Додаток Д.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Вступ. Актуальність дослідження. Мета та завдання дослідження. Об'єкт та предмет дослідження. Управління проєктом. Архітектура MVC. Моделювання бази даних. Реалізація Серверної частини. Інтерфейс користувача. Панель адміністратора. Висновки. Кінець.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	*Виконано*
3.	Виконання дослідження щодо існуючих систем автоматизації інвентаризації	17.02.2026-10.05.2026	*Виконано*
	Розроблення веб-орієнтованої системи прокату обладнання.		
4.	Оформлення розділу «Теоретичні засади та аналіз	11.05.2026-17.05.2026	*Виконано*
	Предметної області»		
5.	Оформлення розділу «Проєктування інформаційної	18.05.2026-24.05.2026	*Виконано*
	системи»		
6.	Оформлення розділу «Практична реалізація та аналіз	25.05.2026-31.05.2026	*Виконано*
	результатів»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат	17.06.2026	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2026	*Виконано*
13.	Захист кваліфікаційної роботи	27.06.2026	

Студент _____ Склярова Н.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веборієнтованої системи для пункту прокату туристичного спорядження // Кваліфікаційна робота освітнього ступеня «Бакалавр» // Склярова Надія Русланівна // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. 90, рис. – 31, табл. – 2, кресл. – 12, додат. – 5, бібліогр. – 44.

Ключові слова: веб-орієнтована система, управління запасами, прокат обладнання, автоматизація процесів, клієнт-серверна архітектура, реляційна база даних, rest api, користувацький інтерфейс

Кваліфікаційна робота присвячена дослідженню процесів автоматизації управління запасами та розробленню вебсистеми прокату туристичного обладнання.

В першому розділі кваліфікаційної роботи описано теоретичні засади та аналіз предметної області. Висвітлено вимоги стейкхолдерів та планування проєкту. Розглянуто архітектуру системи. Проаналізовано літературу та змодельовано базу даних на основі ER-діаграм.

В другому розділі кваліфікаційної роботи обґрунтовано вибір технологій. Досліджено підходи до розроблення за шаблоном MVC. Подано опис реалізації серверної частини (backend) на Node.js та інтеграції з PostgreSQL.

В третьому розділі кваліфікаційної роботи описано створення клієнтської частини (frontend) на React. Проаналізовано інтерфейси, взаємодію через REST API та стандарти інклюзивності. Проведено тестування системи та розглянуто питання охорони праці, ергономіки й безпеки даних (GDPR).

Об'єкт дослідження: процес автоматизації обліку та прокату спорядження.

Предмет дослідження: програмні засоби та методи створення вебсистеми управління запасами.

ANNOTATION

Development of a Web-Oriented System for a Tourist Equipment Rental Point // Qualification work of the educational level «Bachelor» // Skliarova Nadiia // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-42 // Ternopil, 2026 // P. 90, fig. – 31, tabl. – 2, chair. – 12, annexes. – 5, references – 44.

Keywords: web-based system, inventory management, equipment loan, process automation, client-server architecture, relational database, rest api, user interface.

The qualification work is dedicated to automating inventory management and developing a web-based equipment loan system.

The goal of the work is to design and implement a web application that digitalizes manual equipment tracking procedures.

The first section of the qualification paper considered the theoretical foundations and problem analysis. It highlighted stakeholder requirements, analyzed literature, and modeled the database.

In the second section of the qualification work, it is considered the software development approaches and technologies. It detailed the backend development using Node.js, Express, and PostgreSQL.

The third section of the qualification work describes the frontend development using React. It analyzed user interfaces, REST API integration, and conducted system testing alongside GDPR and ergonomic reviews.

Object of the study: the automation of equipment loan processes.

Subject of the study: methods and software tools for creating a web-based inventory system.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

3NF - третя нормальна форма.

ACID - атомарність, узгодженість, ізоляція, довговічність.

API - програмний інтерфейс застосунків.

Back-end - Серверна частина програмного застосунку, яка відповідає за логіку, обробку даних та взаємодію з базою даних.

EDI – рівність, різноманіття та інклюзія.

ERD – діаграма «сутність–зв’язок».

Fetch API - Сучасний інтерфейс JavaScript для виконання HTTP-запитів до сервера.

Front-end - Клієнтська частина застосунку, з якою безпосередньо взаємодіє користувач (UI).

GDPR - Загальний регламент захисту даних (ЄС).

HTTP - протокол передавання гіпертексту.

ICT - інформаційно-комунікаційні технології.

IT – інформаційні технології.

JSON - формат обміну даними JavaScript Object Notation.

LSEPI - правові, соціальні, етичні та професійні аспекти.

MVC - архітектурний шаблон «модель-подання-контролер».

Node.js - Середовище виконання JavaScript для створення серверних застосунків.

PostgreSQL - Реляційна система керування базами даних з відкритим кодом, що використовується для зберігання та керування інформацією.

PROMPT - презентація, релевантність, об’єктивність, метод, походження, актуальність.

RBAC - рольове керування доступом.

React - JavaScript-бібліотека для створення користувацьких інтерфейсів, зокрема односторінкових застосунків.

Redurancy (Надмірність) – Зайве дублювання даних, якого уникають шляхом нормалізації структури бази даних.

REST - архітектурний стиль взаємодії «представницька передача стану».

SQL - мова структурованих запитів.

UI – користувацький інтерфейс.

WCAG - рекомендації з доступності веб-контенту.

Атомарність – Властивість транзакцій бази даних, за якої всі операції виконуються повністю або не виконуються зовсім, що гарантує цілісність даних.

Запит – Операція звернення до бази даних для отримання, зміни чи видалення даних (зазвичай мовою SQL).

Зовнішній ключ - Поле таблиці, яке встановлює зв’язок із записом в іншій таблиці та забезпечує цілісність даних.

Ітеративна розробка - Підхід до створення ПЗ у вигляді послідовних циклів (ітерацій) з регулярним отриманням зворотного зв’язку.

Кінцева точка API (Endpoint) – URL-адреса, на яку надсилаються запити до API для отримання, створення чи оновлення даних.

Нормалізація – Процес оптимізації структури бази даних для усунення надмірності та забезпечення узгодженості.

Оркестрація – Автоматизоване координування роботи кількох компонентів системи для забезпечення їх узгодженої взаємодії.

Реляційна база даних - База даних, що організована у вигляді пов’язаних таблиць з рядками та стовпцями.

Схема бази даних (Database Scheme) - Структура, що визначає організацію, формат зберігання та взаємозв’язки даних у базі даних.

Хмарне розгортання (Cloud Deployment) - Процес розміщення та запуску програмного забезпечення на віддалених хмарних серверах замість локального обладнання.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах функціонування організацій, особливо в бюджетній сфері, ефективність управління матеріальними ресурсами є критично важливою. У внутрішньому магазині туристичного спорядження муніципалітету функціонує внутрішній магазин прокату туристичного спорядження. На початку проєкту процес прокату був повністю паперовим та ручним, що призводило до значної неефективності. Кожна операція, від підтвердження наявності обладнання до перевірки дати повернення, займала значний час, що спричиняло підвищення операційних витрат, зменшення можливостей надання послуг та створювало незручності як для користувачів, так і для працівників. Крім того, неточності під час відстеження залишків призводили до вимірюваних втрат, зокрема за даними обліку магазину за 2023–2024 роки розбіжність між записаним та фактичним обладнанням становила приблизно 12%. Це підкреслює нагальну потребу у впровадженні сучасної інформаційної системи для автоматизації процесів прокату, що й обумовлює актуальність даної роботи.

Мета і завдання дослідження. Метою роботи є розробка веб-застосунку для автоматизації процесу управління та відстеження прокату обладнання у внутрішньому магазині туристичного спорядження.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області та існуючих рішень для автоматизації прокату обладнання;
- визначити вимоги до програмного забезпечення, функціональність системи та розробити її архітектуру;
- обрати технології реалізації серверної та клієнтської частин, бази даних;
- реалізувати веб-застосунок з інтерфейсом користувача, що забезпечує перевірку наявності, бронювання та повернення обладнання;

- розробити базу даних для зберігання інформації про користувачів, обладнання та транзакції;
- забезпечити розмежування прав доступу на основі ролей (RBAC);
- провести тестування розробленої системи та оцінити ефективність її впровадження.

Об’єкт дослідження – процес управління прокатом обладнання в організаціях.

Предмет дослідження – методи та засоби автоматизації процесів бронювання, відстеження наявності та управління поверненням обладнання з використанням веб-технологій.

Наукова новизна одержаних результатів. Наукова новизна роботи полягає у вдосконаленні підходу до автоматизації прокатних операцій шляхом поєднання сервіс-орієнтованої архітектури з рольовою моделлю доступу для забезпечення відстеження ресурсів у реальному часі в умовах обмежених бюджетних установ.

Практичне значення одержаних результатів. Практичне значення роботи визначається створенням повнофункціонального веб-застосунку, який дозволяє автоматизувати ключові бізнес-процеси магазину прокату. Впровадження системи забезпечує зменшення часу на оформлення операцій, підвищення точності обліку обладнання, зменшення кількості помилок, пов’язаних із людським фактором, та покращення контролю за ресурсами. Розроблена система є гнучкою та може бути адаптована для використання в інших підрозділах ради або невеликих компаніях, що надають обладнання в оренду.

Структура й обсяг роботи. Представлена кваліфікаційна робота складається зі вступу, чотирьох розділів та основних висновків. Записка містить 72 сторінок пояснювального матеріалу, 31 рисунок та 2 таблиці. Перелік літератури містить 44 найменування, які розміщені на 5 сторінках, а також 5 додатків на 18 сторінках. Загальний обсяг роботи становить 90 сторінок.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел та пов’язаних досліджень

Надійна теоретична та практична основа була необхідною для проєктування веб-орієнтованої системи програмного забезпечення для автоматизації процесу прокату обладнання у внутрішньому магазині туристичного спорядження, що функціонує при муніципалітеті. Для цього потрібно було розуміння систем управління запасами, розроблення веб-застосунків [2, 5, 24] та інтеграції баз даних [1, 16, 21, 30]. Для забезпечення теоретичного підґрунтя процесу проєктування та розроблення веб-застосунку для прокату обладнання було проведено огляд літератури шляхом систематичного пошуку в електронних наукових базах та Google Scholar.

Огляд літератури був зосереджений на джерелах, опублікованих за останні 4-6 років, щоб забезпечити актуальність до сучасних практик програмування [11], методології розроблення програмного забезпечення [19] та чинних правових норм, таких як Data Protection Act 2018 [8] та Online Safety Act 2023 [10]. Основну увагу приділено рецензованим академічним книгам, практичним посібникам та журнальним статтям.

Основні пошукові терміни включали:

- “*Web-based inventory management system*”;
- “*Equipment loan system*”;
- “*Web application database*”;
- “*Database inventory tracking system*”.

Використовуючи ці ключові пошукові слова, було знайдено джерела, що охоплювали такі теми, як технічна архітектура [9, 12], інтеграція баз даних [44], безпека та відповідність нормативам [4], а також дизайн користувацького інтерфейсу (UI) [26, 31]. Критеріями відбору були роботи, які прямо описували

технічні аспекти реалізації або містили опубліковані кейс-стаді систем, що використовуються в умовах державного сектору. Прийняття рішень щодо включення джерел здійснювалося на основі критеріїв PROMPT, зі зосередженням на достовірності, релевантності до цілей проєкту та практичній корисності для вирішення технічних проблем.

Охоплена література включала як практичні аспекти реалізації, так і концептуальні питання проєктування. Деякі джерела аналізували реальні впровадження веб-орієнтованих систем [28, 29], інші досліджували хмарні технології, бази даних або дизайн інтерфейсів користувача [3]. Сукупно ці джерела забезпечили практичні рекомендації щодо створення системи, яка є ефективною, масштабованою, безпечною та відповідає законодавчим вимогам.

Katircioglu, Brown та Asghar [14], у своїй рецензованій статті, повідомили про дослідження, присвячене автоматизації систем управління запасами з використанням SQL-баз даних. Автори, що представляють навчальні та дослідницькі установи зі спеціалізацією у комп’ютерних науках та дослідженні операцій, внесли вагомий вклад у розуміння оптимізації систем і технологій баз даних. На основі тестування продуктивності та моделювання вони дійшли висновку, що системи на основі SQL забезпечують миттєве відстеження запасів, зменшують потенціал людської помилки та покращують планування ресурсів. Методологія дослідження включала створення прототипу SQL-орієнтованої моделі управління запасами та проведення порівняльного аналізу витрат і точності операцій у порівнянні з традиційними системами управління запасами.

Схожість із цим проєктом полягає у переході від ручних і паперових процедур до цифрової бази даних. Проте автори не розглянули інтеграцію SQL-баз з веб-інтерфейсами, що обмежує застосовність джерела до повної реалізації веб-застосунку.

Lorenzo Ochoa та ін. [18] у своїй публікації з оркестрації систем у розподілених середовищах запропонували застосування механізму orchestration для координації взаємодії між неоднорідними компонентами системи. Автори,

які представляють відомі європейські дослідницькі інститути у галузі комп'ютерних наук, продемонстрували, як оркестрація забезпечує узгоджену передачу даних між модулями інвентаризації, автентифікації та бронювання, тобто саме тими операціями, які є ключовими для цілей цього проєкту. Їхня робота ґрунтувалася на кейс–дослідженні middleware-технологій. Хоча джерело надало значущі концептуальні знання щодо інтеграції багатокомпонентних систем на архітектурному рівні, воно не містило детальних практичних рекомендацій щодо впровадження, що обмежує можливість його прямого застосування.

У своїй книзі Manelli та Zambon [20] виклали практичне керівництво щодо створення динамічних веб-застосунків за допомогою технологій MySQL та Apache Tomcat. Автори, досвідчені інженери-розробники з глибокою експертизою у Java-технологіях, стверджують, що «*web applications [...] can be considered the cornerstone of the modern technology both in private and government organizations*» [20, с.1]. Книга висвітлює питання дизайну UI, модульності, масштабованості та архітектурного проєктування аспекти, що залишаються актуальними незалежно від обраного технологічного стеку. Як підкреслюють автори, «*every time you type something into a web form, an application 'out there' interprets your request and prepares a web page to respond*» [20, с.1], що підкреслює інтерактивну сутність сучасних веб-рішень. Хоча наведена література зосереджена на Java-стеку, її логічні та архітектурні принципи можуть бути порівняльним орієнтиром, навіть якщо проєкт використовує інші фреймворки, такі як React або Node.js.

Автори Li та Gu [17] у своїй роботі дослідили інтеграцію гібридних баз даних у хмарних середовищах. Їхні результати засвідчили переваги таких систем щодо продуктивності та масштабованості, що прямо відповідає меті даного проєкту, а саме розгортання інвентаризаційної системи у хмарі та забезпечення гнучкості моделі зберігання даних. Однак робота значно більше зосереджена на архітектурі та продуктивності, не приділяючи уваги проблемам інтеграції гібридних баз даних із веб-інтерфейсами чи проміжним програмним

забезпеченням. Тому її практична застосовність для фронтенд-рішень була обмеженою.

Автора [15] розглянув питання оптимізації продуктивності, усунення несправностей та керування ресурсами у хмарних екземплярах SQL Server. Для цілей цього проєкту, а саме забезпечення масштабованості та надійності системи, розгорнутої у хмарі ця робота є значущою, оскільки містить практичні інструкції щодо оптимізації запитів, налаштування індексів та управління навантаженням. Водночас, оскільки рекомендації були орієнтовані виключно на SQL Server, їхня універсальність була обмежена для систем, що використовують інші СКБД, такі як PostgreSQL.

Робота Pasaribu [21] представляє кейс-дослідження створення веб-системи для інвентаризації з використанням PHP та MySQL. Автор детально описує процес розроблення, включно зі створенням схеми бази даних, реалізацією рівнів доступу та передаванням даних між модулями. Як зазначає Pasaribu [21, с.3], «inventory problems that arise can be in the form of too much inventory or too little inventory to meet customer demand». Це джерело було особливо корисним, оскільки пропонувало приклад практичного переходу від паперових систем до електронних рішень, орієнтованих на ефективність та зручність використання. Автор також підкреслює [21, с.4], що «*the use of a computerized system will save more time, do not consume a lot of energy, and result in the accuracy of data presentation*», що підтверджує обґрунтованість переходу до веб-орієнтованої системи. Незважаючи на відмінності у використаних технологіях, логіка дизайну та принципи реалізації були напряму релевантними для цього проєкту.

Загалом, розглянута література охопила концептуальні, технічні, архітектурні та правові аспекти створення веб-системи для управління прокатом обладнання, починаючи від структури бази даних і логіки бронювання, і завершуючи вибором технологій, хмарним розгортанням та вимогами до безпеки даних.

У підсумку, проаналізовані джерела забезпечили міцну теоретичну та практичну основу для розроблення веб-орієнтованої системи управління прокатом обладнання для муніципалітету. Література з SQL-баз даних, веб-розробки та хмарного розгортання безпосередньо вплинула на рішення щодо зберігання даних, архітектури системи та масштабованості. Навіть якщо деякі джерела зосереджувалися на інших технологічних стеках, їхні рекомендації стосовно підходів до проєктування залишалися актуальними [43]. Питання законодавчої відповідності та безпеки даних також суттєво вплинули на стратегію відповідності в процесі проєктування системи. Отже, обрані джерела були авторитетними, сучасними та контекстуально релевантними, надаючи цінні рекомендації як у технічній, так і в організаційній площині проєкту.

1.2 Аналіз проблеми та постановка завдання

Магазин туристичного спорядження є внутрішнім підрозділом, що керується муніципалітетом та пропонує внутрішнім працівникам й партнерським організаціям туристичне обладнання, таке як намети, пальники, дощовий одяг та комплектні набори для експедицій. На початку всі операції прокату повністю ґрунтувалися на паперових формах. Ці форми заповнювалися, подавалися та підписувалися вручну працівниками, що створювало трудомісткий процес, схильний до помилок і нерозуміння. Не існувало жодного централізованого цифрового запису, з якого можна було б легко визначити рівень запасів у режимі реального часу або історію прокату. Це виявилося особливо обмежувальним у періоди підвищеного попиту, коли тривале опрацювання форм спричиняло конфлікти в розкладі та викликало невдоволення користувачів.

Ці операційні проблеми вказували на чітку потребу в спільній, відкритій та доступній у режимі реального часу системі для управління запасами й прокатом обладнання [27]. Рішення мало забезпечувати коректне відстеження

залишків, скорочення затримок і створення структурованого та прозорого процесу як для керівництва, так і для працівників.

Залучення зацікавлених сторін було ключовим компонентом у виявленні проблеми та визначенні вимог до системи. Проєкт розпочався з неформальних інтерв'ю та спостереження за робочими процесами в магазині адміністративним персоналом, персоналом магазину під час їхньої роботи. Цей якісний процес узгоджувався з найкращими практиками, рекомендованими літературою з проєктування, орієнтованого на користувача, яка підкреслює важливість ранньої та постійної участі кінцевих користувачів на етапах збору вимог [26]. Ці зустрічі надали уявлення про недоліки поточного процесу, а також виявили конкретні побажання щодо функціональності.

Рисунок 1.1 демонструє діаграму вимог зацікавлених сторін, а саме узагальнення основних побажань та потреб, які були визначені під час дослідження.

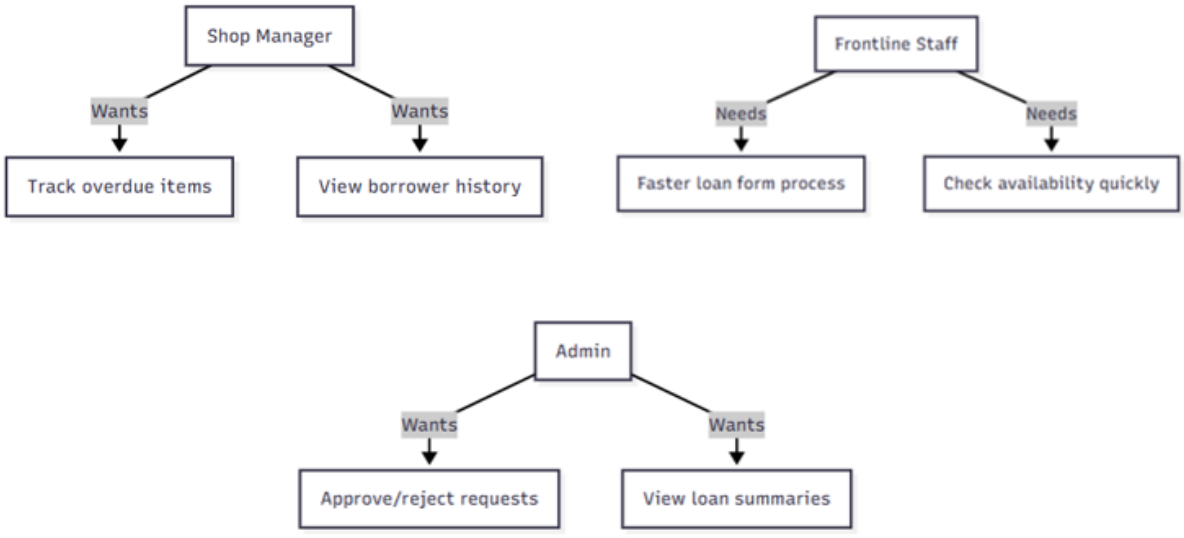

Рисунок 1.1 – Діаграма вимог зацікавлених сторін

Ці вимоги не розглядалися як окремі «запити на функції», а були переведені у функціональні специфікації та специфікації моделі даних. Наприклад, запит менеджера щодо історії позичальників функціонально

визначав необхідність додати сутність історії транзакцій, пов’язану як із записами позичальників, так і з записами обладнання в ER-діаграмі. Потреба швидко перевіряти, чи є певний предмет у наявності, також обґрунтовувала необхідність додати в API системи [24] запит доступності в режимі реального часу, щоб відповіді могли надаватися миттєво без ручної перевірки залишків.

Процес моделювання був заснований на формальних процедурах моделювання баз даних та сутностей [1, 21]. Як зазначають Bagui та Earp [3], системи з декількома функціями та ролями користувачів повинні чітко визначати сутності, атрибути та зв’язки. Розмежування об’єктів, що належать до певних ролей таких як Manager, Staff та Admin спрощує визначення рольового керування доступом (RBAC) і точних операцій, доступних для кожної ролі [2]. Це запобігає надмірному узагальненню користувачів системи та дозволяє у майбутньому розширювати можливості ролей без масштабного перепроєктування.

Спираючись на вимоги, підтверджені зацікавленими сторонами, і на перевірені моделювальні підходи, проєкт гарантував, що фінальна система буде не лише технічно здійсненною, але й точно відповідатиме реальним операційним потребам. Така узгодженість зменшила ймовірність реалізації непотрібного або дубльованого функціоналу та створила основу для масштабованого розвитку в майбутніх ітераціях.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Методологічні підходи до розроблення програмного забезпечення

На основі аналізу, який враховував технічні інструменти й технології, що використовувалися [5, 7], а також методів управління проєктом здійснено структурований огляд найважливіших ресурсів які необхідні для виконання та розробки проєкту. При цьому враховувались особливості комунікації з зацікавленими сторонами, що дозволило здійснити оперативне реагування на підтримувати зворотний зв’язок із замовником.

Оскільки проєкт мав реальний робочий контекст і виконувався одним розробником, ретельне планування ресурсів, управління ризиками та вибір відповідної методології життєвого циклу були критично важливими.

Для побудови структури та навігації вебзастосунку була створена діаграма використання вебсайту (див. Рисунок 2.1).

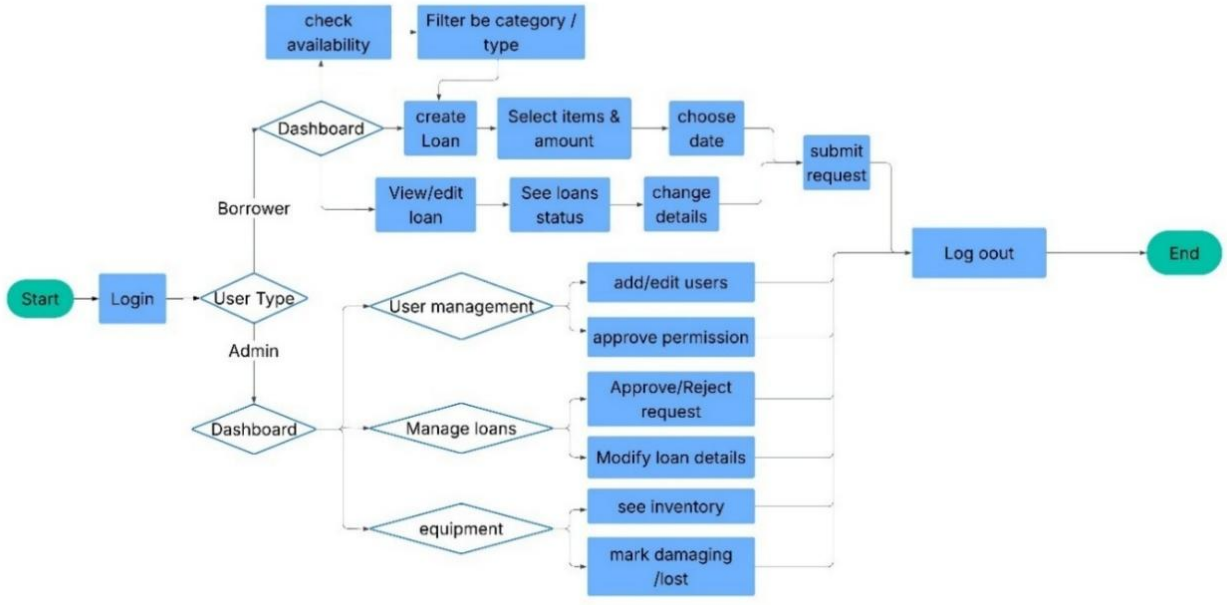

Рисунок 2.1 – Діаграма використання вебсайту

Дана діаграма ілюструє очікувану взаємодію для кожного типу користувачів: позичальників і адміністраторів. Вона уточнює основні завдання,

такі як пошук обладнання, подання запитів на прокат та перегляд історії. Потік для адміністраторів включає затвердження, редагування повернень та контроль запасів.

Її було використано для прямого впливу на дизайн інтерфейсу. Процес для позичальника було мінімізовано до лінійної послідовності рішень, а саме перегляд, запит і перегляд бронювань тоді як адмін-панель містила додаткові рівні доступу для керування позиками та запасами. Такий підхід до проєктування відповідав принципам проєктування, орієнтованого на користувача, та задачоцентричної навігації, описаним у літературі з дизайном взаємодії (interaction design) [26], а також відображав практичні робочі процеси, визначені під час обговорень із зацікавленими сторонами.

Після завершення цього етапу проєктування система була реалізована як повностековий вебзастосунок. Обрана архітектура була тришаровою клієнто-серверною моделлю, у якій кожен шар мав свою чітку функцію: клієнт (браузер) відповідав за відображення та введення, сервер (бекенд) - за логіку та потоки даних [2, 11], а база даних - за постійне зберігання та цілісність інформації [16, 22].

Ця структура відповідала шаблону Model–View–Controller (MVC) [20], у якому React виступав як View, Express як Controller, а PostgreSQL як Model. Така структурованість покращувала підтримуваність, сприяла масштабуванню та давала змогу тестувати рівні логіки, подання та зберігання даних окремо (див. Рисунок 2.2).

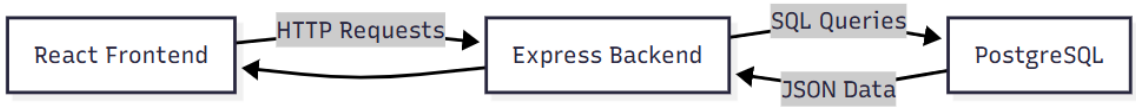

Рисунок 2.2 – Системна архітектура (MVC)

Коли позичальник надсилав форму запиту на прокат, фронтенд кодував дані форми у форматі JSON і відправляв їх у вигляді POST-запиту на бекенд.

Маршрути Express здійснювали перевірку введених даних та передавали їх як інструкції SQL, що виконувалися у PostgreSQL-схемі [19]. Це забезпечувало надійну інтеграцію між користувацьким введенням і реляційними даними. Наприклад, endpoint/loanform додавав новий запис форми позики з даними позичальника, обраними категоріями, очікуваною датою повернення та автоматично згенерованою датою й часом подання (див. лістинг 2.1).

Лістинг 2.1 – Маршрут Express для створення нової форми замовлення

```
router.post('/loanform', async (req, res) => {
  const {
    borrower_id,
    category_id,
    subcategory_id,
    uplift_date,
    return_date_estimated
  } = req.body;

  try {
    const result = await pool.query(
      `INSERT INTO "LoanForm" (
        borrower_id,
        category_id,
        subcategory_id,
        uplift_date,
        return_date_estimated,
        submitted_at,
        status
      ) VALUES ($1, $2, $3, $4, $5, NOW(), 'pending')
      RETURNING id`,
      [borrower_id,   category_id,   subcategory_id,   uplift_date,
return_date_estimated]
    );

    res.status(201).json({
      status: 'success',
      loanId: result.rows[0].id
    });
  } catch (err) {
    res.status(500).json({
      status: 'error',
      message: 'Failed to create loan',
    });
  }
});
```

Це гарантувало, що кожна заявка на позику отримувала автоматично згенерований ID і статус pending, що полегшувало адміністративні процеси, такі як затвердження або відстеження прострочень.

Endpoint/loanitems (див. лістинг 2.2) був доданий до цього функціоналу для прив'язки конкретних одиниць обладнання до форми позики. Бекенд перевіряв наявність відповідного обладнання у таблиці Equipment і коригував рівень запасів перед додаванням нового запису, тим самим запобігаючи подвійним бронюванням та оновлюючи інформацію про фактичне використання обладнання в реальному часі.

Лістинг 2.2 – Маршрут Express додавання обладнання до замовлення

```
router.post('/loanitems', async (req, res) => {
  const { loanform_id, equipment_id, quantity } = req.body;

  if (!loanform_id || !equipment_id || !quantity) {
    return res.status(400).json({ error: 'Missing required fields'
});
  }

  try {
    const availability = await pool.query(
      'SELECT available_quantity FROM "Equipment" WHERE id = $1',
      [equipment_id]
    );

    if (availability.rows.length === 0) {
      return res.status(404).json({ error: 'Equipment not found'
});
    }

    if (availability.rows[0].available_quantity < quantity) {
      return res.status(400).json({ error: 'Not enough available
quantity' });
    }

    await pool.query(
      'INSERT INTO "LoanItem" (loanform_id, equipment_id,
quantity) VALUES ($1, $2, $3)',
      [loanform_id, equipment_id, quantity]
    );

    await pool.query(
      'UPDATE "Equipment" SET available_quantity =
available_quantity - $1 WHERE id = $2',
```

```
      [quantity, equipment_id]
    );

    res.status(201).json({ status: 'success' });
  } catch (err) {
    res.status(500).json({ error: 'Failed to add item' });
  }
});
```

Ці бекенд-процеси демонстрували, як дії користувача перетворюються на правильні та узгоджені транзакції в базі даних. Завдяки наявності обмежень як на рівні бази даних (FOREIGN KEY), так і на рівні застосунку (валідація), система забезпечувала цілісність даних і оптимальний користувацький досвід [25]. Це відповідає усталеній практиці розроблення вебсистем, у яких транзакційна узгодженість між рівнями є критично важливою для стабільної роботи [14, 27].

Об’єднавши користувацькі потоки, бізнес-логіку backend та реляційну схему, процес розроблення забезпечив безпечну, передбачувану роботу застосунку та створив основу для подальшої масштабованості й адміністративного керування системою [23, 28].

2.2 Моделювання даних: ER-діаграми та структура бази даних

Підготовча робота першої фази проєктного графіка розпочалася зі збирання та узагальнення списку обладнання, що використовується в магазині. На основі відгуків від персоналу магазину були зібрані дані щодо типів предметів, їхнього стану, категорій та особливостей використання. Це дозволило сформувати структурований інвентарний список обладнання (див. Додаток А), наприклад: водонепроникна куртка – розмірний ряд S–XXL, категорія: Jackets (waterproof), рюкзаки. Цей список став основою для визначення важливих елементів даних та зв’язків між ними. Як зазначалося раніше, зворотний зв’язок від зацікавлених сторін також вплинув на визначення таких функцій, як відстеження прострочень та класифікація обладнання.

Для того щоб структурувати зібрані дані в ефективну систему, була створена діаграма «сутність–зв’язок» (ER-діаграма), яка стала фундаментом для проєктування бази даних [1] (див. Рисунок 2.3).

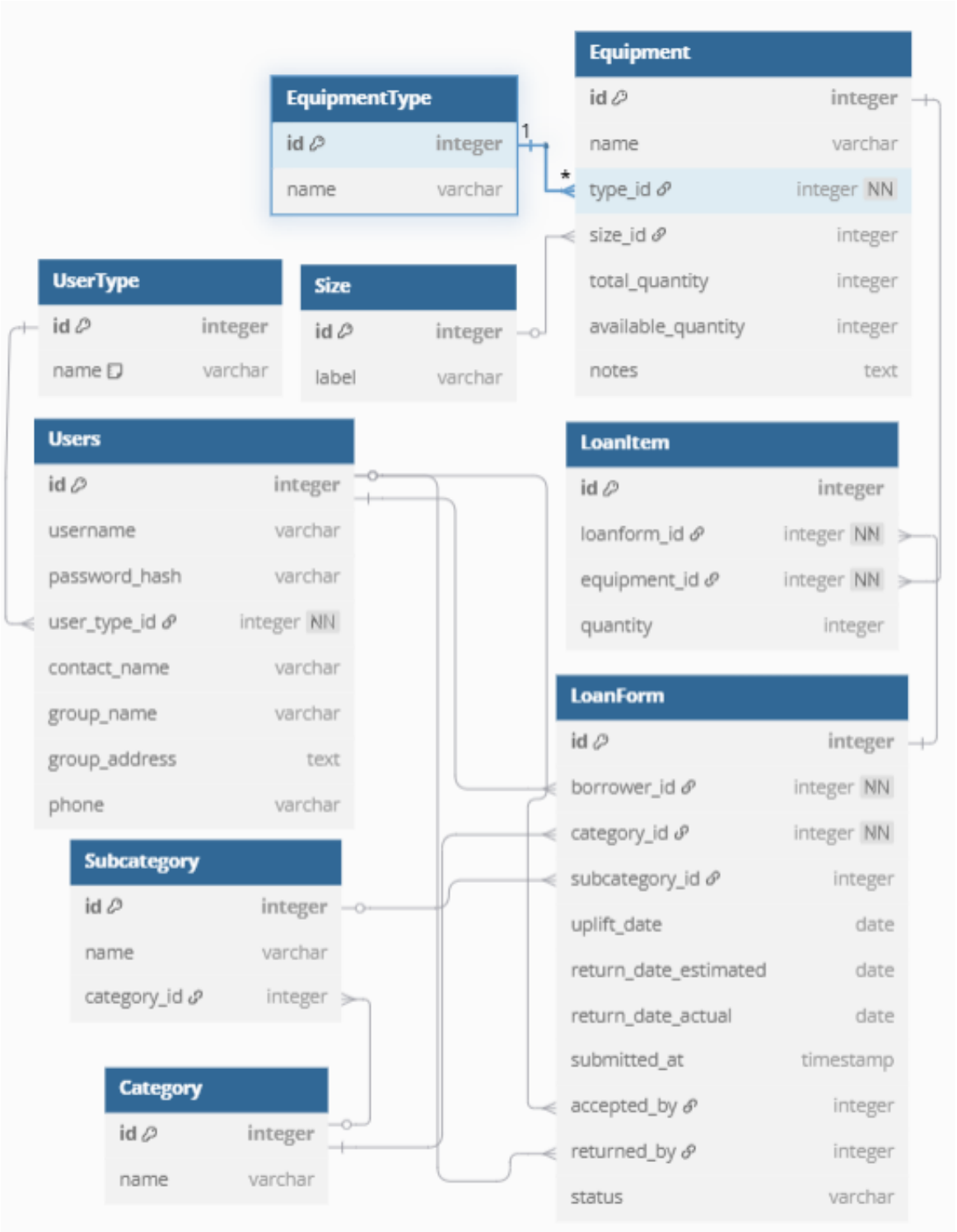

Рисунок 2.3 – ER діаграма бази даних

Відповідно до загальноприйнятої практики ERD, було визначено сутності, такі як User, Equipment, LoanForm, а також зв'язки, типу зв'язок one-to-many між LoanForm та LoanItem, з метою усунення надмірності та уточнення кардинальності [3]. Документування моделі перед реалізацією забезпечило спільне розуміння з боку зацікавлених сторін і зменшило кількість переробок під час створення схеми.

Основною метою цього процесу моделювання було створення реляційної схеми, що відповідає вимогам 3NF однієї з найбільш поширених нормалізованих форм реляційного проєктування, яка мінімізує надмірність і забезпечує цілісність даних [21]. У 3NF неключовий атрибут залежить лише від первинного ключа і не залежить від інших неключовий атрибутів. Це усуває можливість дублювання даних і забезпечує більшу надійність під час оновлень або видалення записів.

Ці зв'язки візуалізовано на рисунку 2.4, який демонструє, як таблиці користувачів, позик та обладнання нормалізуються та пов'язуються між собою.

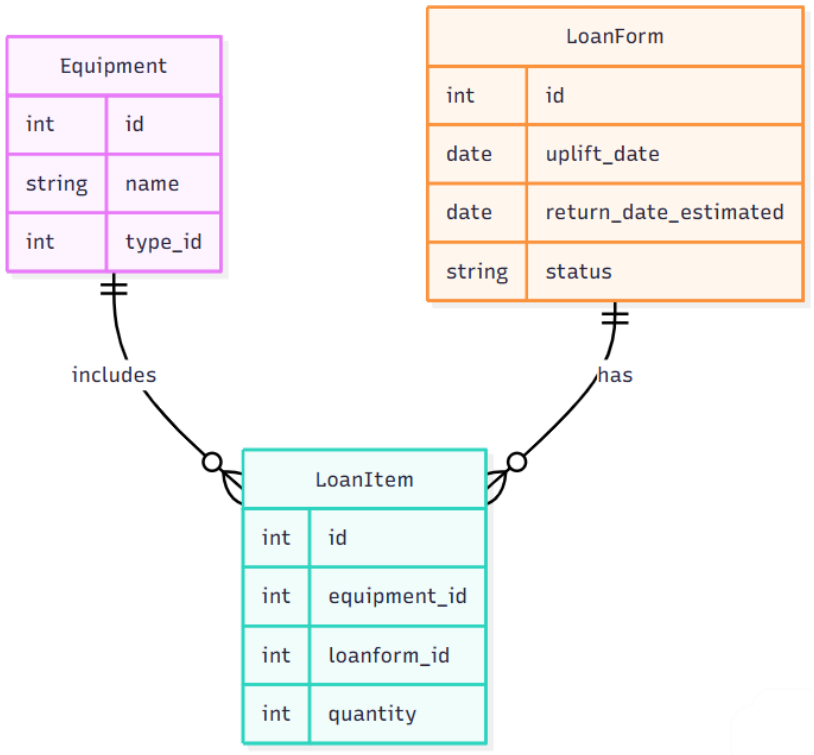

Рисунок 2.4 –ER діаграма таблиць LoanItem, LoanForm та Equipment

Цей дизайн уникає надмірності та дозволяє системі виконувати такі операції, як збільшення кількості нагадувань про прострочені повернення, підрахунок частоти позичання окремих предметів та фільтрування за типами обладнання [14].

ER-діаграма була згодом реалізована як схема PostgreSQL (Додаток В) за методичним процесом перетворення концептуальних ER-моделей у реляційні схеми, описаним Bagui та Earp [3]. Їхній підхід підкреслював важливість визначення сутностей, перетворення зв’язків у зовнішні ключі та забезпечення того, щоб результуюча схема була послідовною і мінімізувала надмірність. Відповідно до цих настанов, абстрактну схему було послідовно перетворено на операційну, яка підтримувала бізнес-вимоги магазину, зокрема відстеження позик, визначення прострочених повернень і класифікацію запасів [30].

PostgreSQL було обрано як платформу бази даних, оскільки вона забезпечує потужну підтримку реляційної цілісності, оптимізації запитів і масштабованості під час роботи з одночасними транзакціями [16]. Порівняно з альтернативами, такими як MySQL, PostgreSQL надійніше обробляє складні з’єднання та обмеження, що робить її особливо придатною для середовищ, у яких цілісність даних має вирішальне значення. Це також підтверджують Li та Gu [17], які зазначають, що переваги PostgreSQL у масштабованості та транзакційній узгодженості роблять її ефективною для систем із суворими реляційними вимогами.

Спрощений SQL-скрипт наведено нижче у Лістингу 2.3. Схема була дуже подібною до ER-моделі й забезпечувала реляційну цілісність за допомогою явних обмежень FOREIGN KEY. Наприклад, кожен запис у LoanItem має посилатися на наявний запис обладнання та відповідну форму позики, що запобігає появі записів, що втратили зв’язок із батьківською таблицею і забезпечує простежуваність даних у всій базі.

Лістинг 2.3 – Фрагмент SQL для створення таблиці

```
CREATE TABLE "LoanForm" (
  "id" integer PRIMARY KEY,
  "borrower_id" integer NOT NULL,
  "category_id" integer NOT NULL,
  "subcategory_id" integer,
  "uplift_date" date,
  "return_date_estimated" date,
  "return_date_actual" date,
  "submitted_at" timestamp,
  "accepted_by" integer,
  "returned_by" integer,
  "status" varchar
);
CREATE TABLE "LoanItem" (
  "id" integer PRIMARY KEY,
  "loanform_id" integer NOT NULL,
  "equipment_id" integer NOT NULL,
  "quantity" integer
);
```

Таблиця LoanForm зберігає інформацію про кожну операцію прокату, включно з позичальником, категорією та датами повернення. Таблиця LoanItem зберігає окремі елементи обладнання у вигляді записів для кожної форми разом із кількістю. Обмеження зовнішніх ключів запобігають збереженню будь-якого запису без відповідної форми позики, а також гарантують, що обладнання не може бути видано, якщо його немає в інвентарі. Структура підтримує базовий функціонал, зокрема створення нагадувань про прострочення, підрахунок частоти позичання та формування коректних адміністративних звітів [44].

База даних була заповнена тестовими інвентарними даними для перевірки її функціональності. Виконувалися запити для підтвердження того, що зв’язки між таблицями повертають коректні результати, а також для ознайомлення з тим, як схема підтримує реальні робочі процеси. Наприклад, SQL-запит у Лістингу 2.4 повертав список поточно позиченого обладнання та дати його повернення, що надавало персоналу можливість швидко перевіряти прострочені повернення:

Лістинг 2.4 – SQL-запит для отримання наразі позиченого обладнання

```
SELECT e.name AS equipment_name,
       li.quantity AS quantity_borrowed,
       lf.uplift_date,
       lf.return_date_estimated
FROM "Equipment" e
JOIN "LoanItem" li ON e.id = li.equipment_id
JOIN "LoanForm" lf ON lf.id = li.loanform_id
WHERE lf.status = 'pending';
```

Результати цього запиту (див. рисунок 2.5) відповідали вимогам зацікавлених сторін, оскільки надавали чіткий список позиченого обладнання, інформацію про позичальника та заплановану дату повернення. Це підтримувало такі операційні функції, як планування експедицій та моніторинг прострочених повернень, які були визначені під час інтерв'ю з персоналом як суттєві проблеми паперової системи.

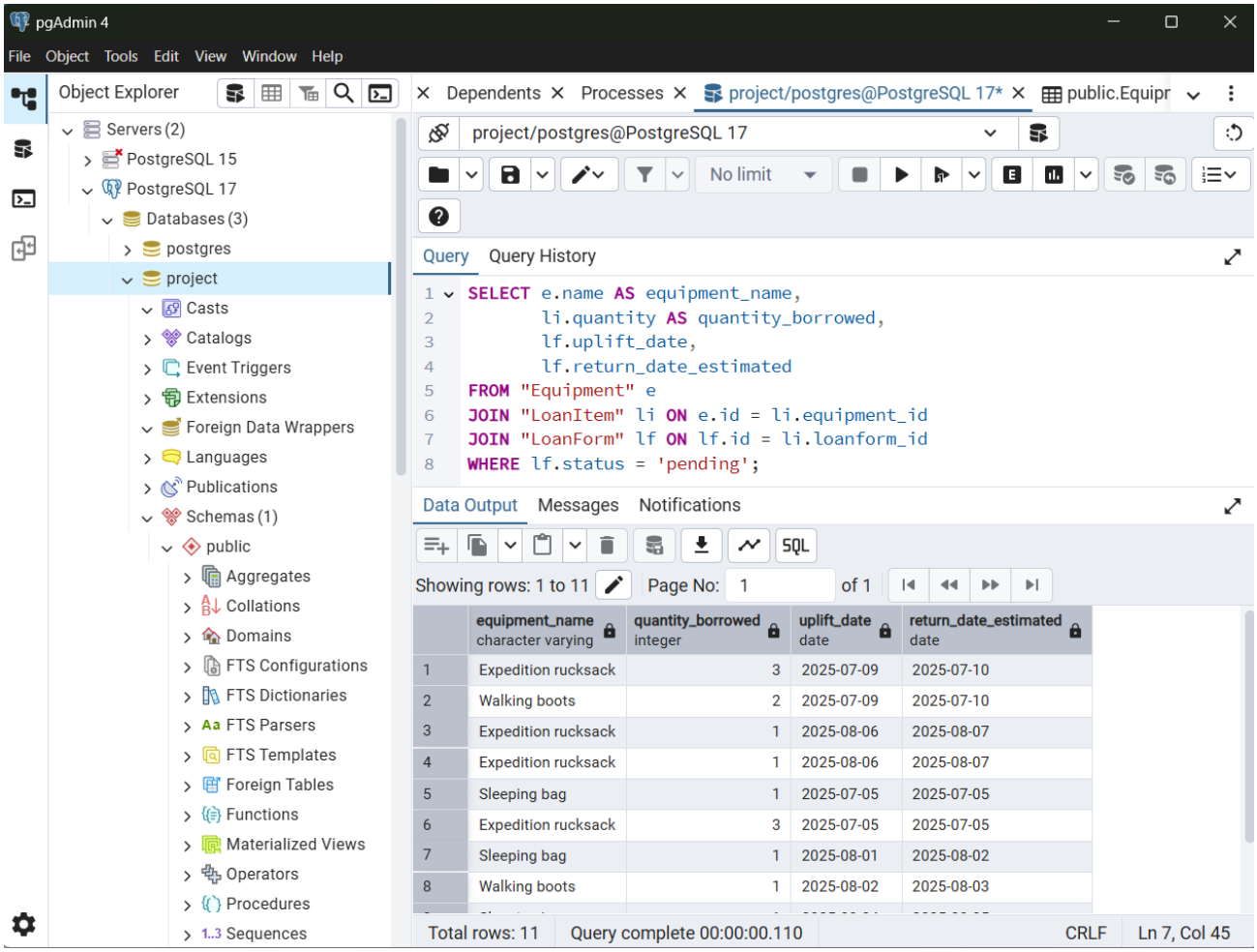

Рисунок 2.5 – Знімок екрана результатів SQL-запиту в pgAdmin4

Для подальшої ілюстрації перетворення ER-моделі на виконувану схему у рисунку 2.6 наведено структуру стовпців таблиці LoanForm. Визначення включало такі поля, як uplift_date, return_date_estimated та status, необхідні для функцій перевірки прострочених повернень, оповіщення про запізнення та відстеження адміністративного затвердження. Завдяки явному визначенню типів даних (наприклад, date, timestamp, varchar) та обмежень схема забезпечувала узгодженість інформації та запобігала суперечностям між записами.

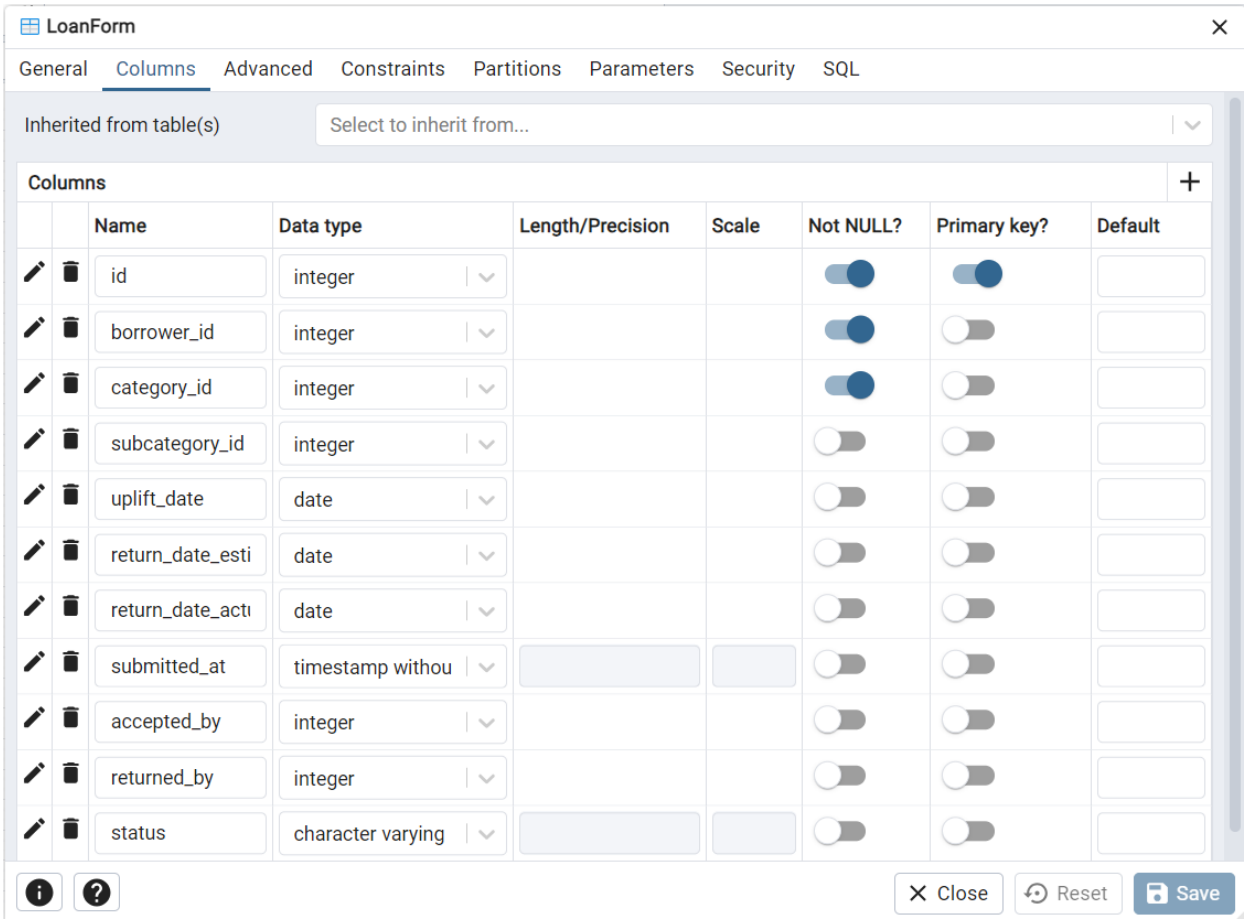

Рисунок 2.6 – Структура стовпців таблиці LoanForm у PostgreSQL

Ця ілюстрація демонструє, як логічно добре спроєктована ER-модель може бути перетворена на реляційну схему, що зберігає не лише теоретичну цілісність, а й ефективно підтримує щоденну роботу. Структура відповідала загальним рекомендаціям з проєктування баз даних [3], уникаючи надмірності,

забезпечуючи цілісність між таблицями та дозволяючи гнучкість для майбутнього розширення системи [15, 22]. Знімки використаної схеми з тестовими даними наведені у Додатку Г, що підтверджує успішне перетворення концептуальної моделі на функціонуючу базу даних. ER-модель та реляційна схема разом формують «скелет» системи, на основі якого було побудовано фронтенд-застосунок та функціонал, доступний користувачу.

2.3 Планування, ризики та життєвий цикл проєкту

2.3.1 Планування технічних та людських ресурсів

Успішність проєкту залежала від збалансування обраних технологій та доступного персоналу з реалістичними цілями, а також від урахування навчальних потреб, що випливали з виконання проєкту. Технологічна платформа була побудована з використанням перевірених інструментів, які дають змогу реалізувати модульну розробку, безпечне оброблення даних та поетапне нарощування функціональності [28]. Фронтенд було створено за допомогою React [31], а бекенд – за допомогою Node.js і Express [24], обраних через їх підтримку асинхронного програмування, механізмів рендерингу стану та можливості реалізації RESTful API [11, 19]. До таких тем належали шаблони проєктування на основі компонентів, клієнтсько-серверна взаємодія через HTTP-запити, а також розділення відповідальностей за допомогою багаторівневої архітектури [2]. У ході попередніх етапів розроблення ці знання були реалізовані на практиці під час створення, отримання та динамічного відображення даних про обладнання.

PostgreSQL було обрано як реляційну систему керування базами даних через її стабільність, підтримку реляційної цілісності та рольового керування доступом (RBAC), що було критично важливо для безпечного управління записами про обладнання [16]. Хоча схема бази даних змінювалася із зростанням складності, а зокрема в процесі ітеративної розробки, щоб це не

створювало проблем завдяки передбаченим міграціям і регулярним резервним копіям [17].

Людські ресурси також відігравали центральну роль у створенні та вдосконаленні системи. Співпраця з працівниками магазину забезпечила глибоке розуміння предметної області, зокрема щодо робочих процесів, пов'язаних з прокатом обладнання, описом стану спорядження та фіксацією повернень. Їхні коментарі під час зустрічей безпосередньо вплинули на схему бази даних та ранні рішення щодо дизайну інтерфейсу [26]. Рекомендації від IT-відділу забезпечили безпеку розгортання системи, хоча й супроводжувалися технічними вимогами щодо відповідності, які були враховані на ранньому етапі планування [13]. Постійний зворотний зв'язок із науковим керівником також сприяв узгодженню з академічними вимогами, оскільки чернетки розділів проходили перевірку на якість та цілісність розроблення.

У цілому проєкт був добре збалансований між технічною реалізацією та академічними принципами. Кожен компонент від логіки інтерфейсу до моделі даних, що відповідав як операційним потребам, так і структурованим практикам. Завдяки ефективному використанню наявних ресурсів і керуванню на основі попередніх навчальних результатів, проєкт залишався у межах графіка, залишався життєздатним і міг адаптуватися до перешкод у межах його часових обмежень.

2.3.2 Управління ризиками та стратегії їх мінімізації

Планування ризиків було необхідним для прогнозування та контролю ймовірних проблем, що могли виникати під час розроблення. Оскільки в проєкті були задіяні реальні користувачі (персонал магазину муніципалітету), сторонні інструменти (React, PostgreSQL) та встановлені часові обмеження, ризики виникали з технічних, особистісних і процесуальних аспектів.

У таблиці 2.1 нижче наведено основні ризики, розглянуті на етапі планування. Вони згруповані за категоріями: технічні ресурси, людські ресурси

та особиста спроможність, навички й знання, а також методи. Кожен ризик було оцінено за ймовірністю виникнення, потенційним впливом та загальним рівнем ризику і доповнено запланованими діями зі зменшення або нейтралізації наслідків [23].

Таблиця 2.1 – Оцінка ризиків

Категорія	**Подія**	**Ймовірність**	**Вплив**	**Рівень ризику**	**Заходи з пом'якшення ризику**
1	**2**	**3**	**4**	**5**	**6**
Технічні ресурси	Несподівані помилки у розгорнутій системі	Середня	Високий	Високий	Заплановано додатковий час для тестування та виправлення помилок під час етапу розгортання. Проведено ретельне тестування перед фінальним розгортанням.
	Проблеми інтеграції між front-end і back-end	Середня	Середній	Середній	Проводилось поступове тестування API. Використовувалися інструменти на кшталт Postman для перевірки бекенд-ендпоінтів.
	Збій сервера/бази даних локально під час тестування	Низька	Високий	Високий	Створено локальну резервну копію бази даних (наприклад, SQL-дампи); сервіси запускались через скрипти з можливістю відновлення.
	Невідповідність середовищ розробки, версії Node/PostgreSQL	Середня	Середній	Середній	Детально задокументовано налаштування середовища (наприклад, у README). Використано менеджери версій або Docker (коли дозволяв час).

Продовження таблиці 2.1

1	2	3	4	5	6
Технічні ресурси	Проблеми сумісності між різними браузерами/пристроями	Середня	Середній	Середній	Виконувалось тестування протягом усього циклу розробки; застосовано принципи адаптивного дизайну.
	Невідповідність середовищ розробки, версії Node/PostgreSQL	Середня	Середній	Середній	Детально задокументовано налаштування середовища (наприклад, у README). Використано менеджери версій або Docker (коли дозволяв час).
	Проблеми сумісності між різними браузерами/пристроями	Середня	Середній	Середній	Виконувалось тестування протягом усього циклу розробки; застосовано принципи адаптивного дизайну.
Людські ресурси та особисті обставини	Неочікувані особисті обставини (наприклад, хвороба, сімейні події)	Низька	Високий	Середній	Збережено гнучкий графік, передбачено резервний час у плані. Підготовлено запасний план для виконання роботи у випадку тимчасової недоступності.
	Неможливість вкластися у фінальний термін розгортання	Низька	Високий	Високий	Регулярно відслідковувався прогрес. Завдання були розбиті на дрібніші частини з постійним моніторингом виконання.
	Труднощі під час написання фінальної документації	Середня	Середній	Середній	Документація розпочата заздалегідь і оновлювалася поступово. Пріоритезовано ключові розділи, виділено достатньо часу на редагування.

Продовження таблиці 2.1

1	2	3	4	5	6
	Затримки у зворотному зв’язку від зацікавлених сторін	Середня	Середній	Середній	Регулярно проводились зустрічі зі стейкхолдерами. Встановлено чіткі дедлайни для отримання зворотного зв’язку та забезпечено своєчасну комунікацію.
Навички і вміння	Обмежений досвід у проведенні користувацького тестування або перевірці доступності	Середня	Середній	Середній	Підготовлено структуровані тестові сценарії. За можливості проводилось спостереження за реальними користувачами, занотовувалися всі зауваження.
	Обмежений досвід у проведенні тестування користувачів та інтеграції їхнього зворотного зв’язку	Середня	Середній	Середній	Використовувалися інструменти логування та відстеження помилок. Читалася технічна документація та аналізувалися приклади застосунків.
Методи	Пізні зміни або доповнення, запитані зацікавленими сторонами	Низька	Середній	Низький	Встановлено чіткі межі для змін. Список функцій було зафіксовано перед фінальним етапом розробки (feature freeze).
	Надмірна залежність від одного інструмента або платформи	Низька	Високий	Середній	Наперед визначались альтернативи. Здійснювалося збереження даних у стандартних форматах (наприклад, CSV, SQL).

Виявлені у таблиці 2.1 ризики були не лише задокументовані, але й безпосередньо опрацьовані під час розроблення шляхом застосування відповідних заходів зі зменшення їхнього впливу. Наприклад, проблеми інтеграції між фронтендом і бекендом вирішувалися шляхом поступового тестування API та використання інструмента Postman. Несподівані помилки були мінімізовані завдяки виділенню додаткового часу на ітеративне тестування, а відмови бази даних – шляхом регулярного створення резервних копій і використання скриптів для відновлення роботи сервісів [15].

Проблеми, пов’язані з особистою доступністю часу та написанням документації, були частково усунені шляхом підтримання гнучкого графіка та послідовного написання фрагментів фінального звіту. Ризики, що виникали через обмежений досвід у проведенні тестування користувачів або інтеграції зворотного зв’язку від стейкхолдерів, також були зменшені завдяки ефективному плануванню тестових сценаріїв та проведенню регулярних комунікаційних зустрічей із чітко визначеними дедлайнами [4].

Завдяки таким діям найсерйозніші ризики були передбачені й вирішені за допомогою детальних та конкретних планів. Загалом, процес планування забезпечив систематизоване управління технічними, особистісними та процесуальними ризиками протягом усього життєвого циклу проєкту.

2.3.3 Модель життєвого циклу проєкту та графік виконання (діаграма Ганта)

Вибір моделі розроблення програмного забезпечення безпосередньо вплинув на планування, контроль обсягу робіт і комунікацію із зацікавленими сторонами. Оскільки цей проєкт передбачав участь реальних користувачів (працівників магазину), постійні зміни у вимогах до дизайну та особисте навчання, найкраще підходив саме ітеративний підхід [9, 12].

У наведеній нижче таблиці 2.2 порівнюються три моделі життєвого циклу розроблення програмного забезпечення, а саме Waterfall, Iterative і Scrum, разом

з найважливішими факторами, що мали значення для цього проєкту. Метою було пояснити, чому саме підхід Iterative був обраний.

Таблиця 2.2 – Порівняння моделей життєвого циклу програм

Аспект	**Waterfall**	**Iterative**	**Scrum (Agile)**
1	**2**	**3**	**4**
Гнучкість змін	Не підходить, оскільки еволюційні вимоги або пізній зворотний зв'язок складно врахувати на пізніх етапах.	Краще підходить для поступового вдосконалення; корисний для врахування зворотного зв'язку під час розроблення.	Найбільш гнучкий; заохочує постійне вдосконалення на основі зворотного зв'язку користувачів, але може бути надмірним для одного розробника з обмеженим доступом до користувачів.
Залучення користувачів	Не підходить; проєкт значно вигравав від реального зворотного зв'язку персоналу, який користувався системою.	Дозволяє структуровану участь користувачів після кожного етапу; добре узгоджується з раннім внутрішнім тестуванням і циклами зворотного зв'язку.	Ідеально в теорії, але складно реалізувати в цьому випадку через відсутність повної команди розроблення й обмежений доступ до кінцевих користувачів.
Складність управління	Легко планувати, але існує ризик не задовольнити реальні потреби через відсутність ітеративного зворотного зв'язку.	Керовано для одного розробника; структуроване планування з можливістю гнучкості без необхідності щоденної співпраці зі стейкхолдерами.	Потребує тісної співпраці, частих зустрічей, оновлення беклогу та планування спринтів; може перевантажити масштаб і ресурси проєкту.

Продовження таблиці 2.2

1	2	3	4
Складність реалізації	Передбачуваний шлях, але існує ризик розробити неправильне рішення, якщо початкові припущення неточні.	Дизайн і розроблення можуть еволюціонувати крок за кроком, а уроки з одного етапу покращують наступний.	Потребує модульної реалізації з ранніх етапів і постійної інтеграції; ефективно для команд, але складніше при роботі одного розробника.
Технічна реалізація	Архітектуру планують уперед, але існує ризик невідповідності реальним сценаріям використання.	Структура може еволюціонувати природно, у процесі дослідження логіки бронювання й функцій відстеження під час тестування.	Працює добре з сучасними інструментами та хмарним розгортанням, але потребує безперервної інтеграції та вбудованих процесів тестування.
Придатність для цього проєкту	Бракує гнучкості й зворотного зв'язку користувачів; високий ризик невідповідності потребам через статичне планування, неможливість стейкхолдерів змінювати сайт.	Забезпечує баланс між структурою та адаптивністю; підтримує вдосконалення на основі зворотного зв'язку, що відповідає реальним умовам роботи системи прокату обладнання. Добре підходить стейкхолдерам, оскільки вони можуть вносити зміни у вебсайт у будь-який час.	Добре узгоджується в теорії, але може бути надто вимогливим з точки зору процесів і співпраці для одного розробника й невеликого проєкту. Добре підходить стейкхолдерам, оскільки вони можуть вносити зміни у вебсайт у будь-який час.

Ітеративна модель забезпечила баланс між структурованістю та гнучкістю. Вона дала змогу враховувати зворотний зв’язок користувачів, виконувати поетапний розвиток системи та вносити зміни в середині проєкту, усе це було необхідністю для цього реального, невеликого за масштабом проєкту. Waterfall був надто негнучким, а Scrum вимагав командних процесів, що були непридатними для умов розроблення одним розробником [43].

Графік виконання проєкту був поділений на чотири загальні фази: збір вимог, розроблення прототипу, інтеграція бекенда та тестування, а також фінальні вдосконалення. На рисунку 2.7 було подано узагальнений вигляд повного графіка, а саме контекстуалізовану діаграму Ганта, де кожен тиждень був позначений коротким описом виконаної роботи.

Рисунок 2.7 був чітким графічним представленням розкладу, де кожен рядок позначав окрему фазу проєкту, а кожен блок відображав конкретні завдання, такі як проєктування ERD [1], програмування макету UI [31], інтеграція API [24] або зустрічі із зацікавленими сторонами. Добре позначеними були етапи створення першого прототипу, перший цикл тестування користувачами та фінальне розгортання. Такий підхід дозволив простежити перекривання завдань, залежності між ними та критичні моменти, коли отриманий зворотний зв’язок впливав на подальшу розробку.

Більш детальний розподіл подано у додатку Б, де кожна активність вказана відповідно до тижня її виконання. До них входили: дослідження, моделювання ERD, реалізація фронтенду, виконання тестових процедур, тестування міграції бази даних, удосконалення інтерфейсу користувача та офіційні зустрічі із зацікавленими сторонами. Розподіл також включав залежності, такі як необхідність завершити проєктування схеми бази даних перед інтеграцією бекенда, а також активності, які могли виконуватися паралельно, наприклад, удосконалення UI одночасно з підготовкою до користувацького тестування.

		Sc...	Task Name	Duration	Start	End
1			Project Tm470	155 days	10/02/2025	12/09/2025
2			⊟ Phase 1 - Requirements Gathering	20 days	10/02/2025	09/03/2025
3			⊞ Research & Discovery	5 days	10/02/2025	16/02/2025
9			⊞ Requirements Definition	5 days	17/02/2025	21/02/2025
15			⊞ Market & Technical Review	5 days	24/02/2025	28/02/2025
21			⊞ System Design & Submission TMA01	2 days	03/03/2025	04/03/2025
26			⊟ Phase 2 - Prototype Development	45 days	04/03/2025	05/05/2025
27			⊞ Data Collection & Initial Planning	8 days	05/03/2025	14/03/2025
32			⊞ User & System Analysis	10 days	17/03/2025	28/03/2025
38			⊞ Database Design & Refinement	10 days	31/03/2025	11/04/2025
44			⊞ Documentation & Submission TMA02	5 days	28/04/2025	02/05/2025
50			⊟ Phase 3 - Backend Integration and Testing	50 days	05/05/2025	11/07/2025
51			⊞ UI/UX Design & Setup	10 days	05/05/2025	16/05/2025
57			⊞ Core Functionality Development	10 days	19/05/2025	30/05/2025
63			⊞ User Roles & Testing	10 days	02/06/2025	13/06/2025
69			⊞ Refinement & Submission TMA03	10 days	30/06/2025	11/07/2025
74			⊟ Phase 4 - Final Improvements	45 days	14/07/2025	12/09/2025
75			⊞ Feedback Implementation & Final Testing	10 days	14/07/2025	25/07/2025
81			⊞ Deployment Preparation	10 days	04/08/2025	15/08/2025
88			⊞ Final Evaluation & Bug Fixes	10 days	18/08/2025	29/08/2025
94			⊞ Documentation & Submission	10 days	01/09/2025	12/09/2025

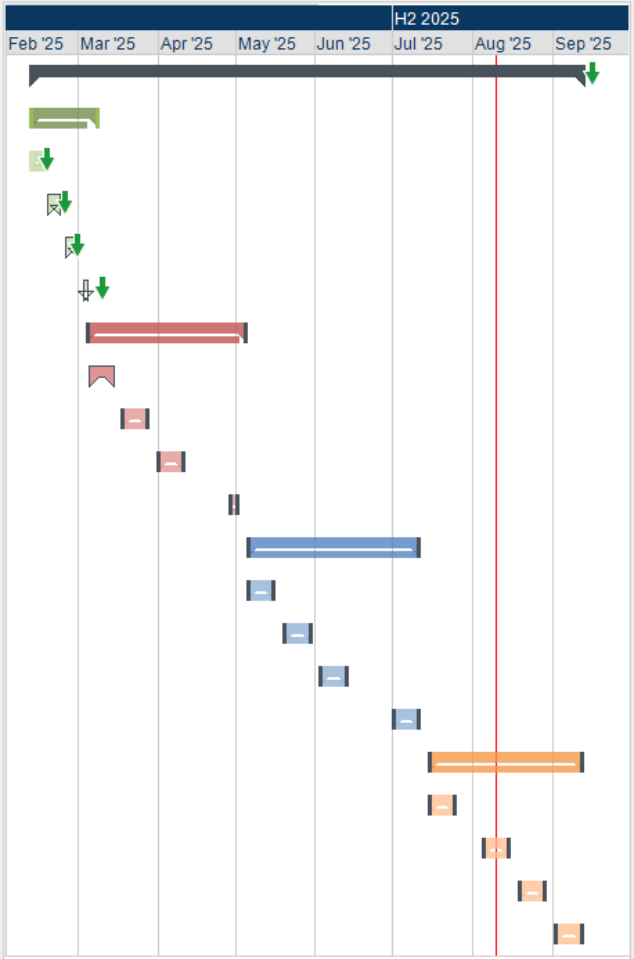

Рисунок 2.7– Діаграма Ганта

Графік був сформований на основі попереднього аналізу ризиків (див. таблиця 2.1) і рішень щодо розподілу ресурсів. Наприклад, додатковий час для тестування інтеграції було враховано для зменшення ризику проблем у комунікації між фронтендом і бекендом, а буферні тижні були додані після завершення фаз 2 і 3, щоб компенсувати можливі затримки у зворотному зв'язку від зацікавлених сторін. Такі кроки зменшували вплив передбачуваних ризиків, таких як непередбачені помилки, невідповідність середовищ або обмеження особистого часу.

Додавання описових позначок до кожного тижня спростило порівняння між запланованим і фактичним прогресом. Коли ранній етап інтеграції бекенда затримався через неочікувані проблеми з міграцією PostgreSQL, діаграму Ганта було оновлено, і частину роботи з удосконалення UI було перенесено на раніші етапи, щоб зберегти загальний темп розроблення. Це забезпечило, що жодна окрема затримка не зрушить весь графік розгортання. Збір вимог був завершений вчасно, чому сприяла постійна взаємодія з працівниками магазину, а розроблення прототипу випереджало графік у деяких частинах, що дозволило приділити додатковий час поліруванню інтерфейсу. Буферні періоди виявилися корисними, оскільки вони поглинули затримки, спричинені інтеграційними проблемами, не вплинувши на кінцевий термін [29].

Загалом діаграма Ганта використовувалася як інструмент для планування та для відстеження фактичного прогресу. Вона дала змогу визначати вузькі місця, підтримувала своєчасне прийняття рішень і допомагала у структурованому ітеративному процесі, доступністю ресурсів та стратегіями пом'якшення ризиків протягом усього життєвого циклу проєкту.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Загальна логіка роботи системи та реалізація серверної частини (backend)

Логіка застосунку була поділена на два окремі репозиторії: frontend (ema_app) та backend (ema_service). Фронтенд був розроблений за допомогою React, де кожен компонент керував певним представленням, наприклад, переглядом обладнання, формами бронювання або адмін-панеллю [31]. Компонентно-орієнтований підхід сприяв модульності та простоті підтримки, що відповідає сучасним практикам дизайну взаємодії (interaction design) [26].

Бекенд, створений за допомогою Express, надавав ендпоінти, такі як /api/equipment, /api/loans та /api/users. Ці маршрути оголошувалися у файлі маршрутизації серверної частини та передавалися у функції контролерів, які взаємодіяли з базою даних PostgreSQL через бібліотеку pg [19]. Така архітектура зберігала чітке розділення відповідальностей та відповідала принципу, що API це угода між клієнтом і сервером.

Комунікація між фронтендом і бекендом здійснювалась через Fetch API для виконання асинхронних HTTP-запитів [24]. Наприклад, коли користувач переглядав список обладнання, фронтенд відправляв GET-запит на бекенд і отримував відповідь у форматі JSON, яка містила записи обладнання з бази даних (див. Рисунок 3.1).

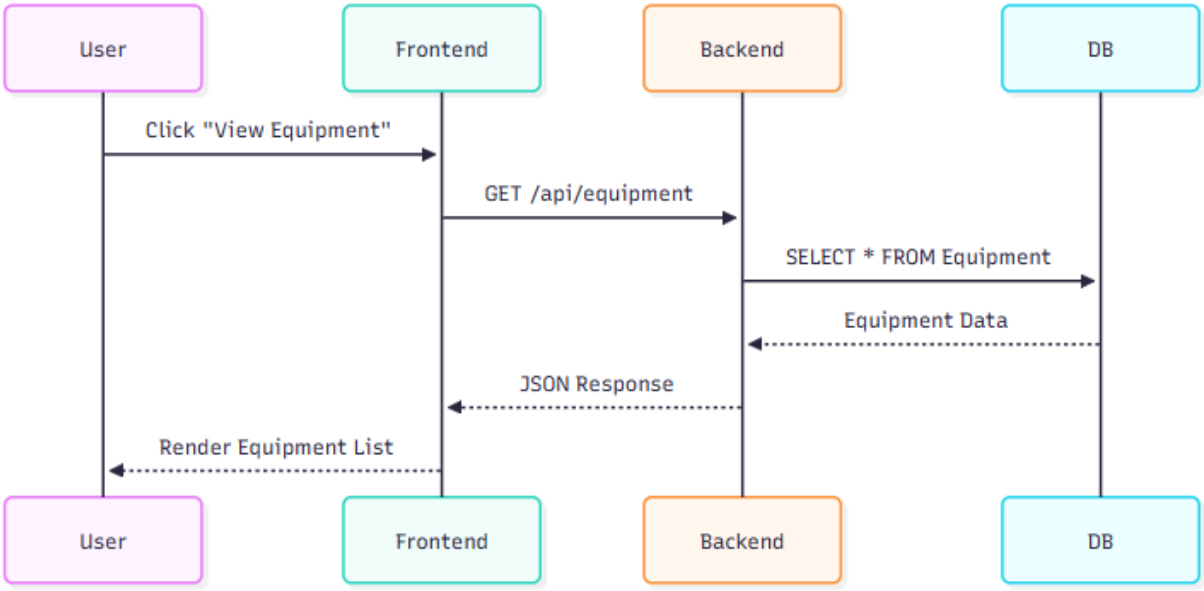

Рисунок 3.1 - Потік даних між фронтендом та бекендом за допомогою REST API

Скорочений фрагмент такої логіки наведений у Лістингу 3.1 і демонструє, як React отримував усі дані щодо наявного обладнання з бекенда.

Лістинг 3.1 – Отримання списку всього обладнання

```
const fetchAllEquipment = async () => {
  try {
    const response = await fetch('http://localhost:3000/api/equipment');
    if (!response.ok) throw new Error('Failed to fetch equipment');
    const data = await response.json();
    setAllEquipment(data);
  } catch (err) {
    alert(`Error loading equipment: ${err instanceof Error ? err.message : 'Unknown error'}`);
  }
};
```

Відповідний маршрут бекенда, який обробляє цей запит, показаний у Лістингу 3.2. Цей Express-маршрут виконував запит до бази даних PostgreSQL, отримував усі записи обладнання та повертав їх клієнту у форматі JSON. Це

наочно демонструвало, як API-endpoints забезпечують комунікацію між клієнтом та сервером.

Лістинг 3.2 – маршрут Express для обладнання

```
router.get('/equipment', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM "Equipment" 
ORDER BY name');
    res.json(result.rows);
  } catch (err) {
    console.error('Error fetching equipment:', err);
    res.status(500).json({ error: 'Failed to fetch equipment' });
  }
});
```

База даних запитувалась через пул з’єднань, керований модулем pg, а параметри підключення (назва бази даних, хост, порт і облікові дані) були налаштовані у файлі середовища, який використовувався всіма процесами. Така конфігурація підвищувала масштабованість і спрощувала налагодження відповідно до найкращих практик інтеграції вебсервісів [18].

Під час проєктування RESTful API дотримувалися рекомендацій, за якими кожен endpoint представляє окремий ресурс (наприклад, /loans або /equipment), що робило логіку бекенда розширюваною та модульною без порушення наявної функціональності. Це узгоджується з літературою з проєктування вебзастосунків, яка підкреслює переваги ресурсно-орієнтованих API для повторного використання та роз’єднання компонентів [20]. Рисунок 3.2 показує абстракцію процесу створення позики, що базувався на атомарних транзакціях для забезпечення узгодженості.

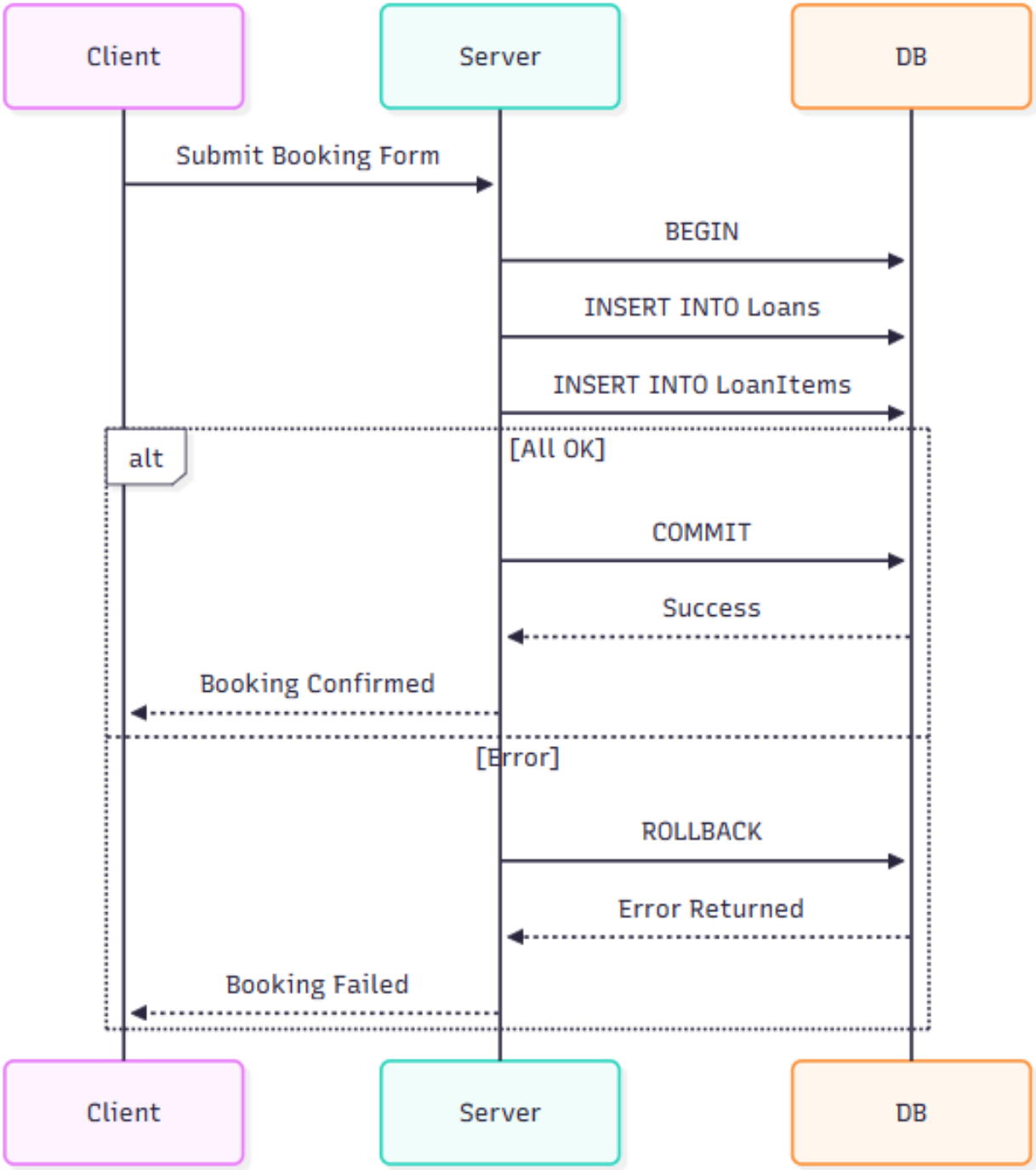

Рисунок 3.2 - Потік транзакцій для створення замовлення (ACID)

Підсумовуючи, фронтенд (React) надсилає HTTP-запити на бекенд (Express), який їх опрацьовує та взаємодіє з базою даних (PostgreSQL). Архітектура дотримується багаторівневого, модульного дизайну та відповідає сучасним практикам розроблення, що робить її масштабованою та зручною для подальшого розширення, наприклад, для додавання аналітики або розширених ролей доступу.

3.2 Реалізація клієнтської частини (frontend): інтерфейси та функції

Фронтенд застосунку був розроблений таким чином, щоб забезпечити чистий, інтуїтивно зрозумілий інтерфейс, зберігаючи при цьому всі основні функції системи прокату обладнання. Кожна сторінка виконувала певну потребу користувача, щоб надати безперервний, ефективний досвід як адміністраторам, так і звичайним користувачам [31].

Застосунок розпочинався зі сторінки входу (див. Рисунок 3.3), на якій користувачі вводили своє ім’я користувача та пароль, щоб увійти до системи. Повний програмний код реалізації компонента сторінки входу (фронтенд-логіка та розмітка) наведено у Додатку Д. Сторінка містила поля введення облікових даних і кнопку входу, яка запускала процес автентифікації на сервері.

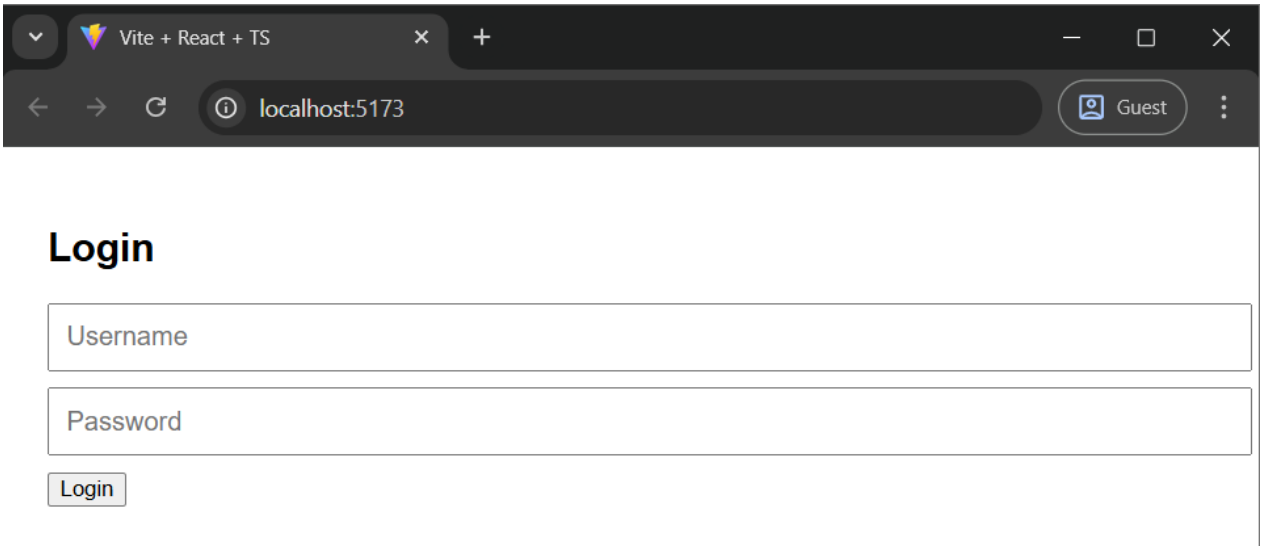

Рисунок 3.3 – Сторінка входу в систему

Після входу звичайний користувач переходив на домашню сторінку користувача (див. Рисунок 3.4). На сторінці були представлені навігаційні посилання, які дозволяли користувачу подати нову заявку на позику, перевірити наявність обладнання та переглянути історію бронювань. Інтерфейс був спроєктований для швидкого доступу до основних функцій, пов’язаних із

прокатом обладнання. Дизайн підкреслював мінімальну кількість навігаційних кроків, що відповідало принципам простоти та ефективності [26].

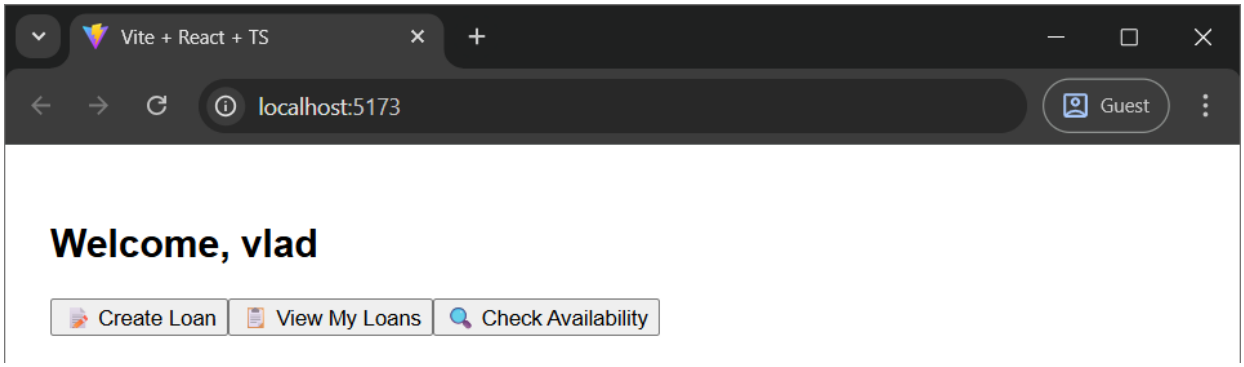

Рисунок 3.4 - Головна сторінка користувача для користувачів (позичальників)

Коли користувач обирав створити нову позику, він переходив на сторінку створення заявки (Рисунок 3.5). Вона містила поля форми для введення необхідної інформації, такої як мета позики, бажаний термін та категорія обладнання.

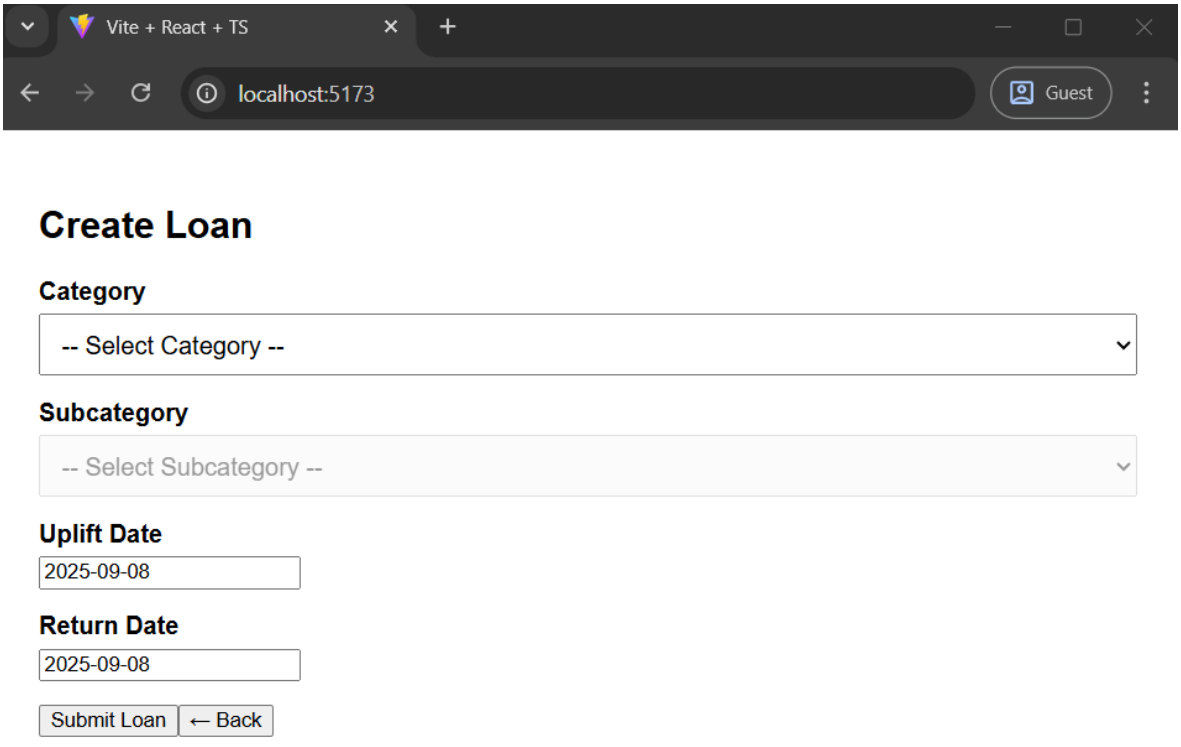

Рисунок 3.5 – Сторінка для створення замовлення

Важливою особливістю цієї сторінки було динамічне фільтрування підкатегорій відповідно до обраної категорії. Ця функція гарантувала, що користувачі бачитимуть лише релевантні підкатегорії, що зменшувало плутанину та запобігало некоректним введенням. Наприклад, якщо обиралася категорія “Course”, випадаючий список підкатегорій автоматично пропонував лише відповідні значення, такі як “Navigation” (див. Рисунок 3.6).

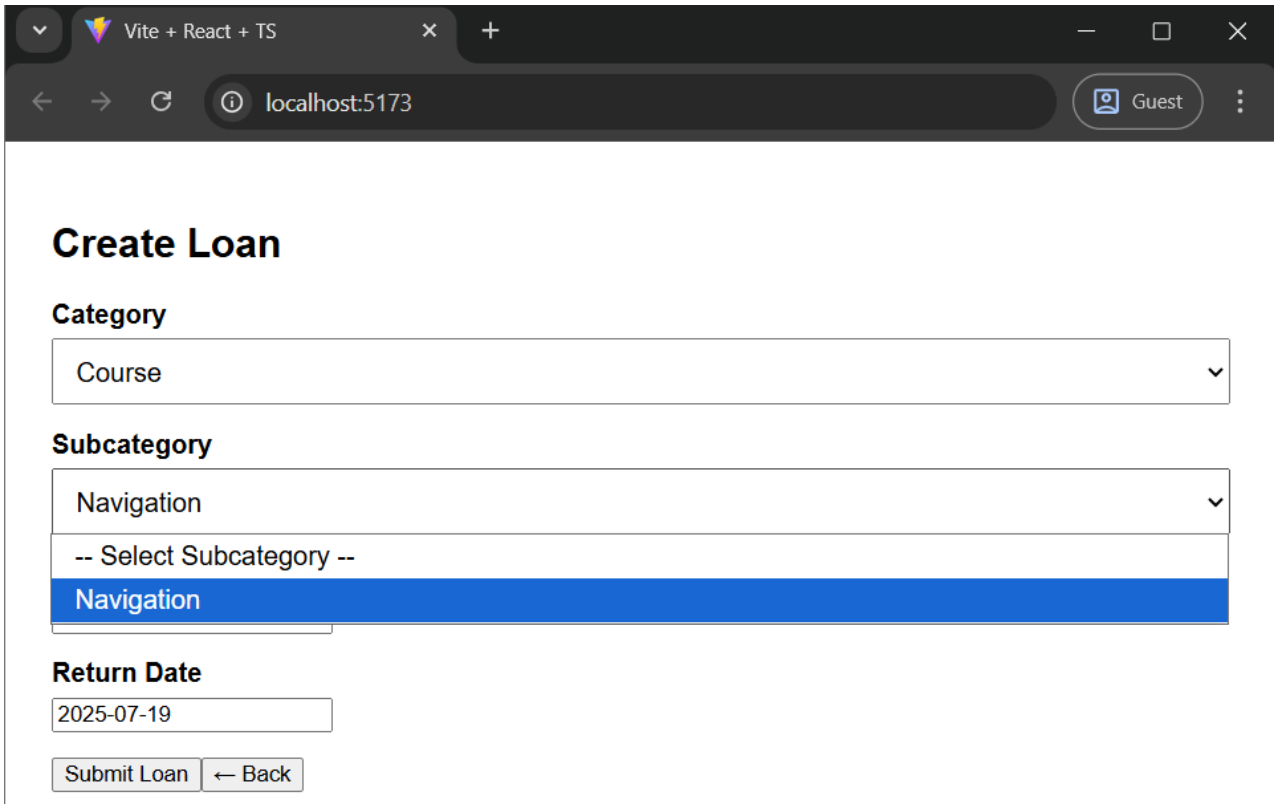

Рисунок 3.6 – Фільтрування підкатегорій для категорії «Course» на сторінці створення позики

Аналогічно, коли обиралася категорія “Expedition”, список підкатегорій оновлювався динамічно, щоб містити релевантні опції, наприклад, “Camping” (див. Рисунок 3.7).

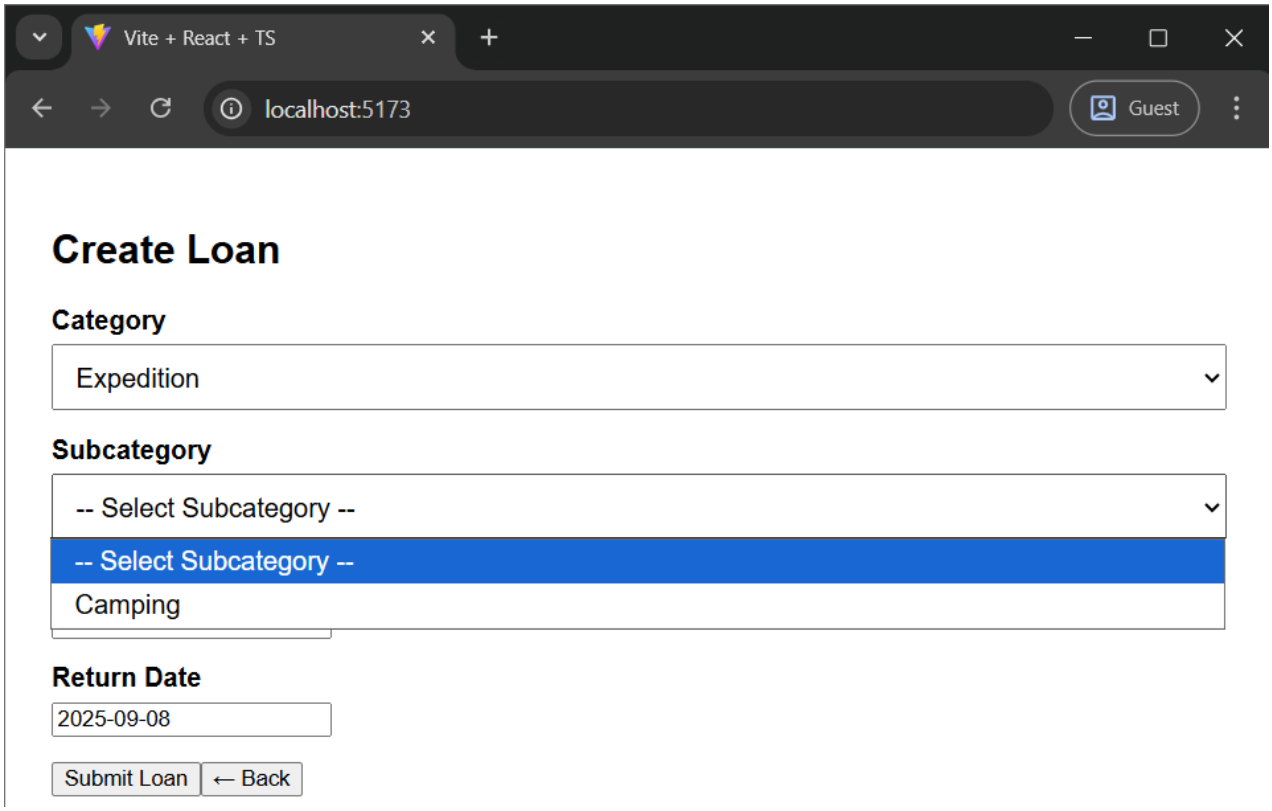

Рисунок 3.7 – Фільтрування підкатегорій для категорії «Expedition» на сторінці створення позики

Ця можливість відображала реальні робочі процеси, виявлені під час інтерв'ю зі стейкхолдерами, коли доступність та вид обладнання залежали від контексту. Бекенд-запит, який підтримував цю поведінку, отримував підкатегорії з бази даних (див. Лістинг 3.3):

Лістинг 3.3 - SQL-запит для отримання підкатегорій

```
'SELECT *
FROM "Subcategory"
WHERE category_id = $1'
```

Щоб допомогти користувачам у виборі відповідного обладнання, сторінка додавання предметів (див. Рисунок 3.8) містила список усього доступного обладнання з чекбоксами. Користувачі могли вибрати декілька предметів, які потрібно додати до заявки, що значно спрощувало процес бронювання.

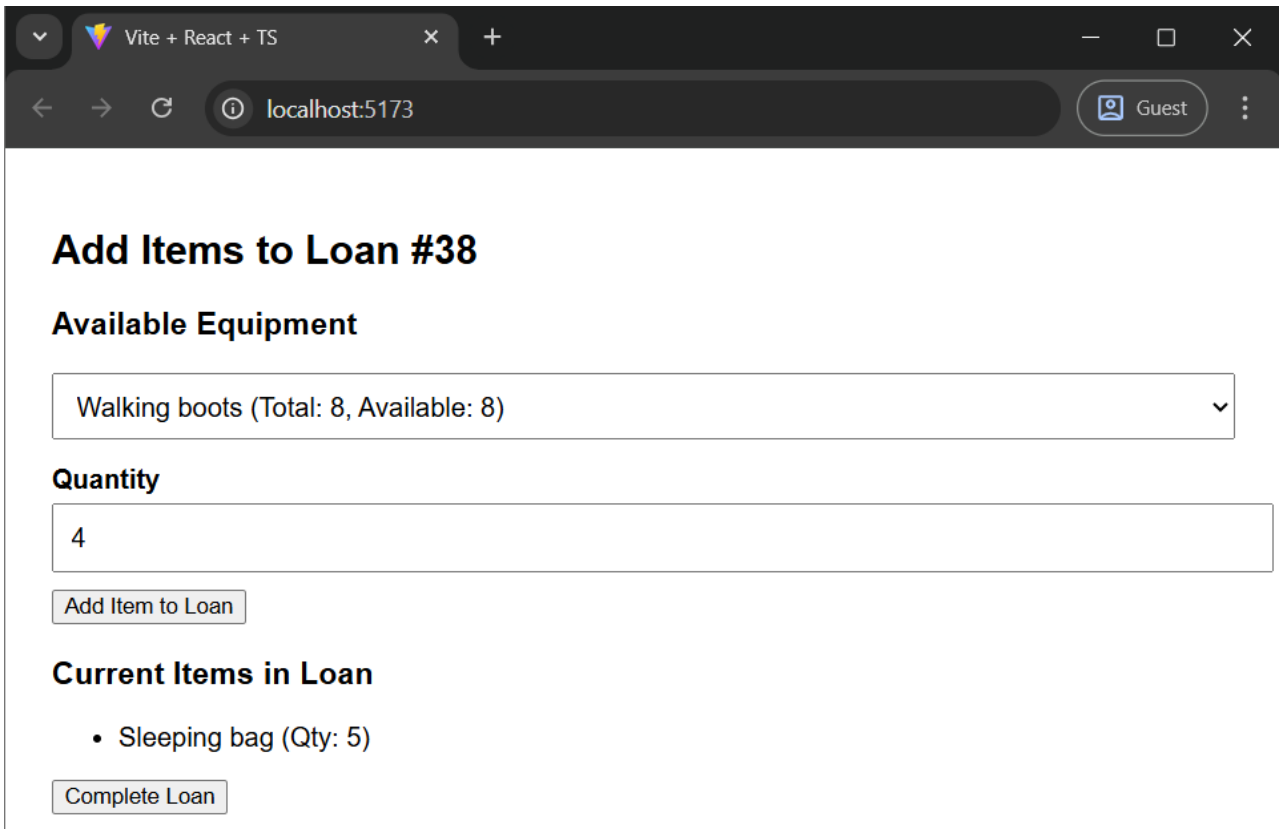

Рисунок 3.8 - Сторінка додавання елементів

Сторінка перевірки наявності (Рисунок 3.9) дозволяла користувачам обирати бажані дати позики за допомогою інтерактивного вибору дат. Вона динамічно відображала доступність обладнання на основі введених дат, що допомагало уникати конфліктів та організовувати позики ефективно.

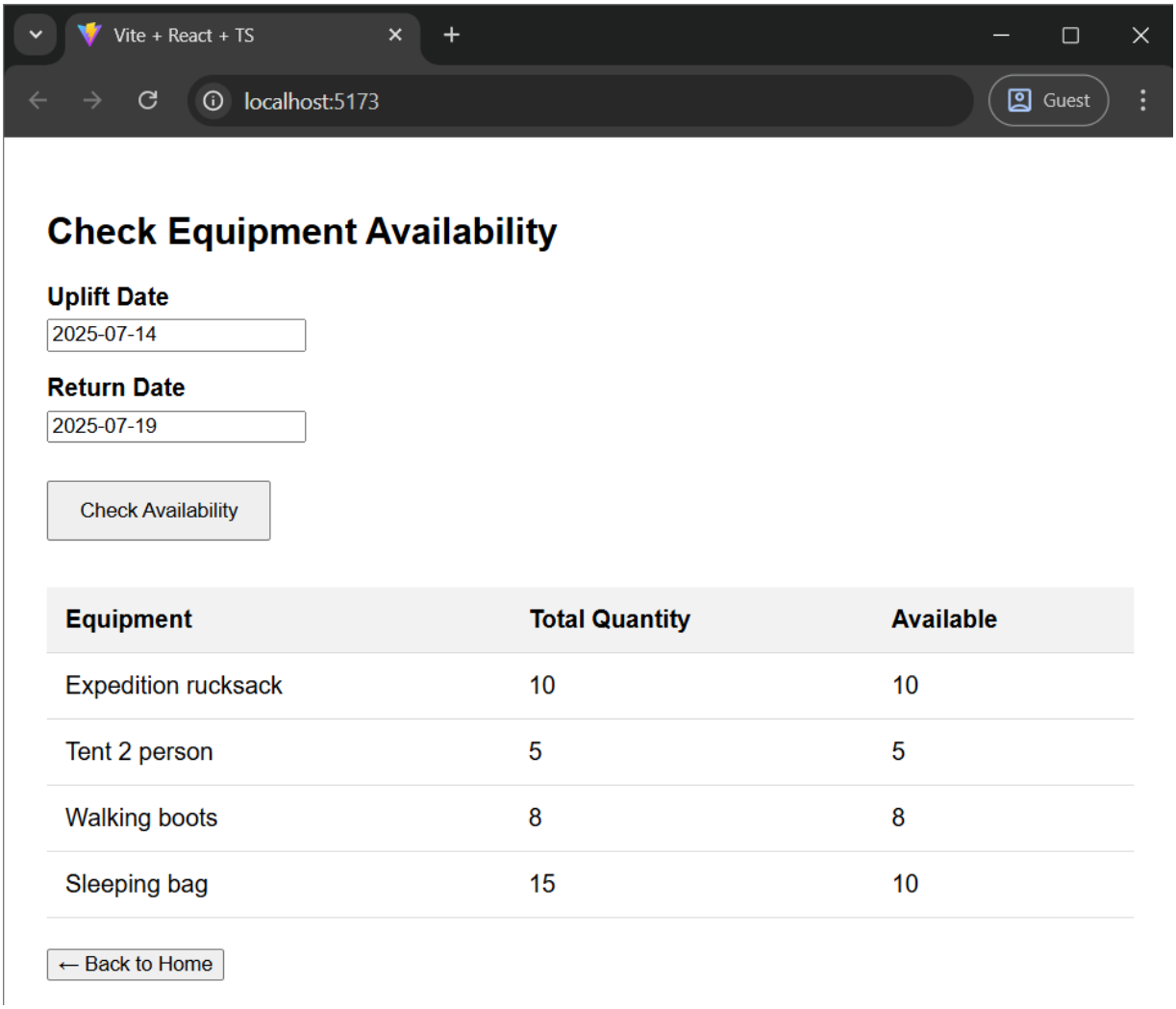

Рисунок 3.9 - Сторінка перевірки наявності

Сторінка отримувала дані про наявність обладнання шляхом запиту до PostgreSQL з урахуванням можливих перетинів бронювань [16]. Запит для цієї функції наведено у Лістингу 3.4.

Лістинг 3.4 – SQL-запит для отримання доступного обладнання

```
SELECT
    e.id,
    e.name,
    e.total_quantity,
    e.available_quantity,
    (e.total_quantity - COALESCE((
     SELECT SUM(li.quantity)
      FROM "LoanItem" li
      JOIN "LoanForm" lf ON li.loanform_id = lf.id
      WHERE li.equipment_id = e.id
       AND lf.return_date_actual IS NULL
```

```
       AND (
/* Check if the equipment is booked on the day before uplift */
(lf.uplift_date  <=  $1::date  AND  lf.return_date_estimated  >=
$1::date) OR
/* Check if the equipment is booked on the day after return */
(lf.uplift_date  <=  $2::date  AND  lf.return_date_estimated  >=
$2::date) OR
/* Check if the equipment is booked during the entire period */
(lf.uplift_date  >=  $1::date  AND  lf.return_date_estimated  <=
$2::date)
)
), 0)) AS available_during_period
FROM "Equipment" e
 WHERE e.available_quantity > 0
 AND (e.total_quantity - COALESCE((
 SELECT SUM(li.quantity)
 FROM "LoanItem" li
 JOIN "LoanForm" lf ON li.loanform_id = lf.id
 WHERE li.equipment_id = e.id
 AND lf.return_date_actual IS NULL
 AND (
(lf.uplift_date  <=  $1::date  AND  lf.return_date_estimated  >=
$1::date) OR
(lf.uplift_date  <=  $2::date  AND  lf.return_date_estimated  >=
$2::date) OR
(lf.uplift_date  >=  $1::date  AND  lf.return_date_estimated  <=
$2::date)
)), 0)) > 0`
```

Це забезпечувало автоматичне виявлення конфліктів, щоб користувачі могли планувати позики без дублювання.

Користувачі також могли переглядати всі свої заявки на сторінці перегляду позик (див. Рисунок 3.10). Сторінка показувала як минулі, так і активні заявки з чіткими статусами, наприклад approved, pending, rejected, що дозволяло легко відстежувати їхній стан.

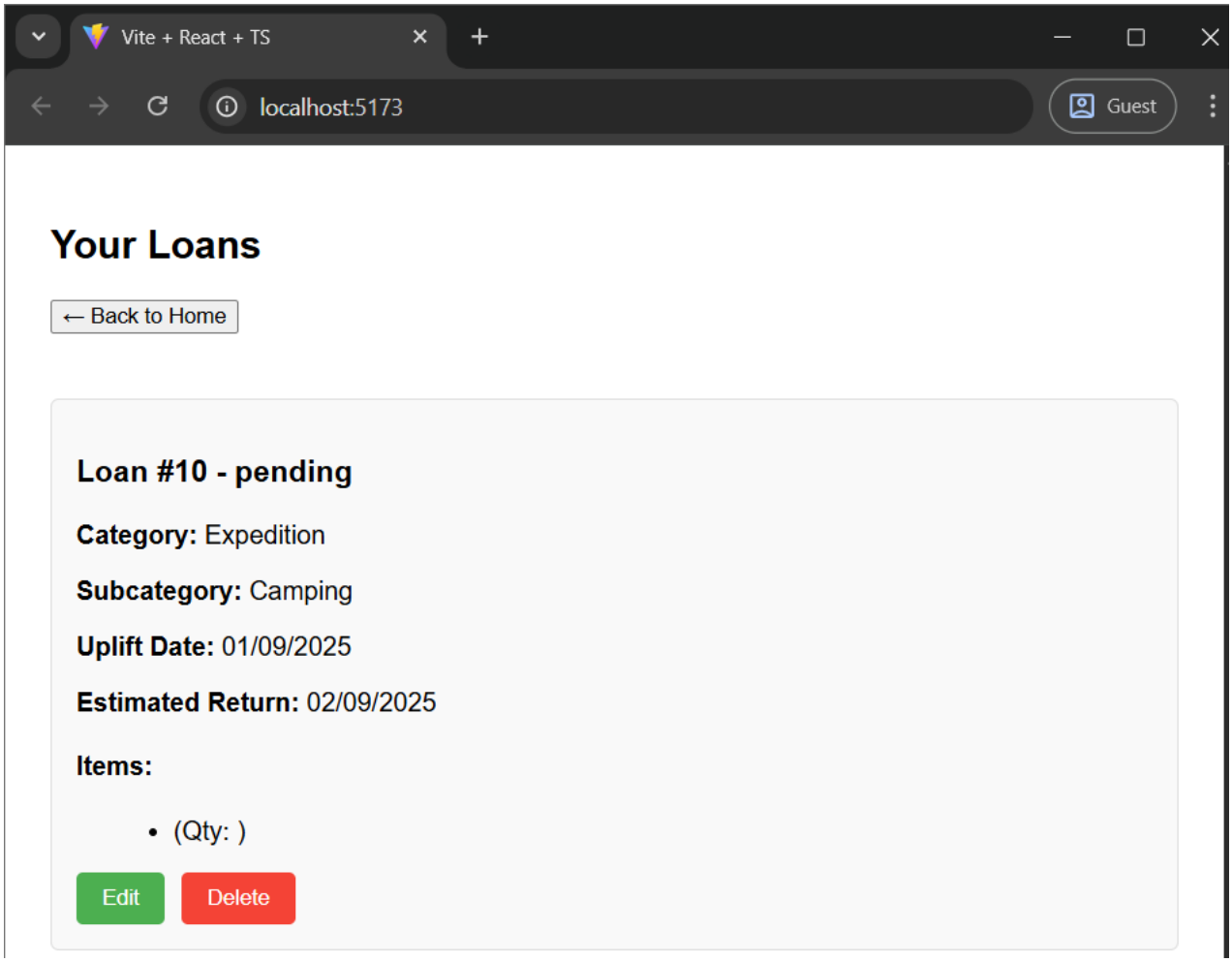

Рисунок 3.10 – Сторінка перегляду позик для поточного користувача

Окрім перегляду, користувачі могли керувати заявками. Коли користувач обирав видалити заявку, система негайно показувала сповіщення-підтвердження (див. Рисунок 3.11). Такий миттєвий зворотний зв’язок допомагав уникнути невизначеності та посилював передбачуваність роботи системи.

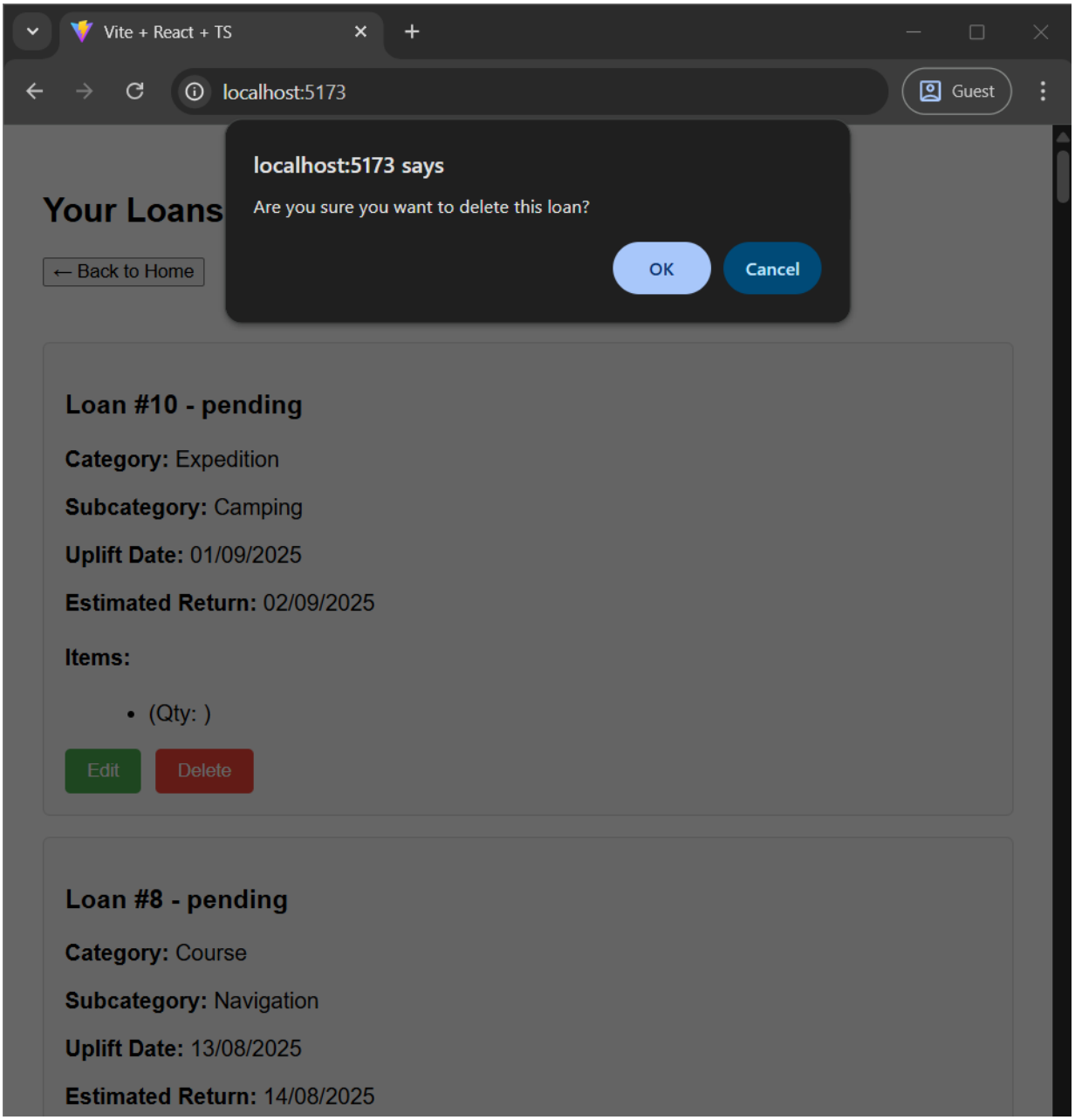

Рисунок 3.11 – Повідомлення про підтвердження видалення позики

Після видалення сторінка автоматично оновлювалася, показуючи оновлений список позик без видаленого запису (Рисунок 3.12). Це підтверджувало успішність дії та забезпечувало точність історії позик користувача.

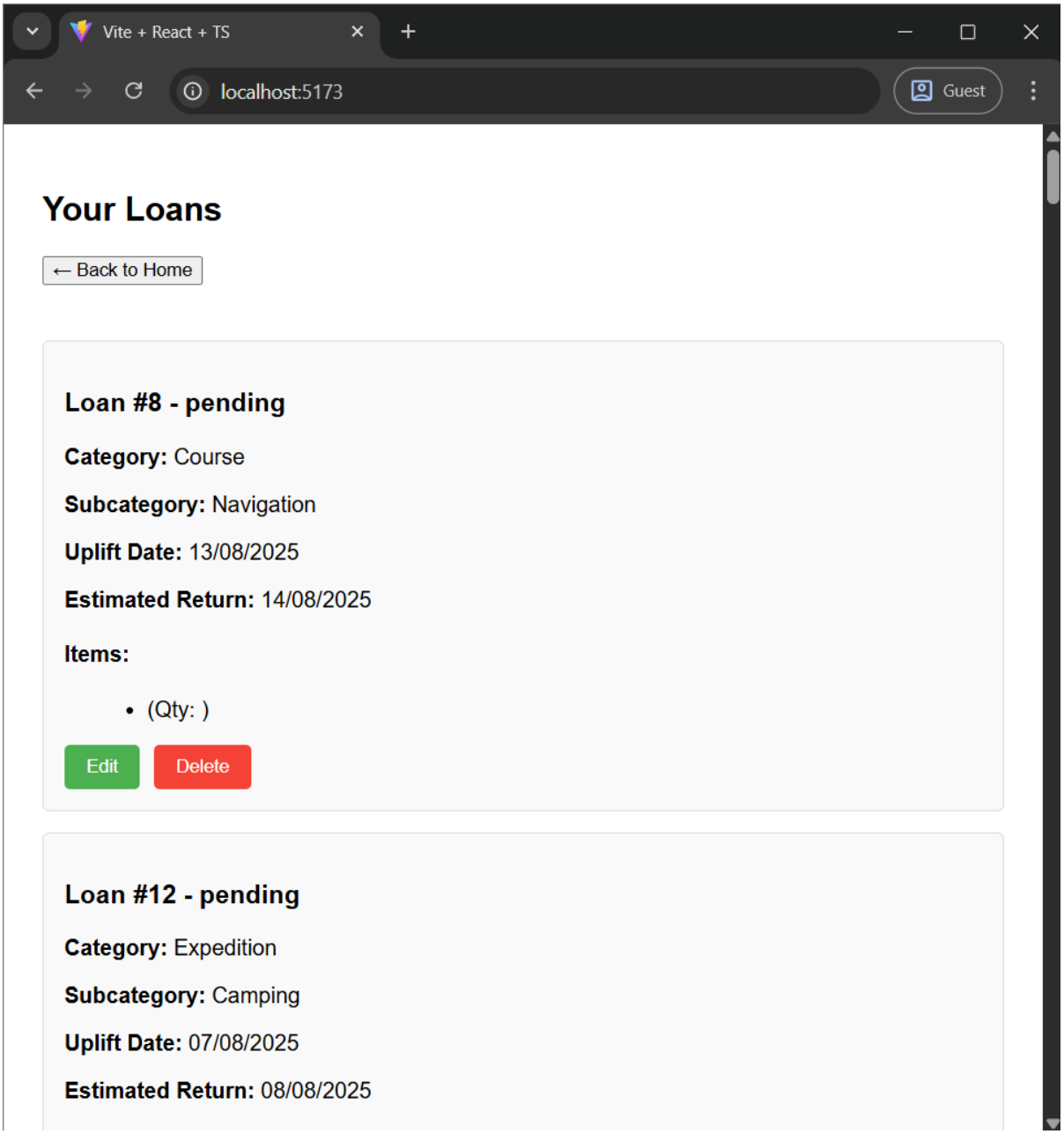

Рисунок 3.12 – Оновлений список позик після видалення

Адміністратори мали унікальну адмін-панель (див. Рисунок 3.13), яка містила оглядові інформаційні картки щодо стану заявок та запасів обладнання. Ця панель дозволяла адміністраторам швидко оцінювати поточне навантаження системи та доступні ресурси [27].

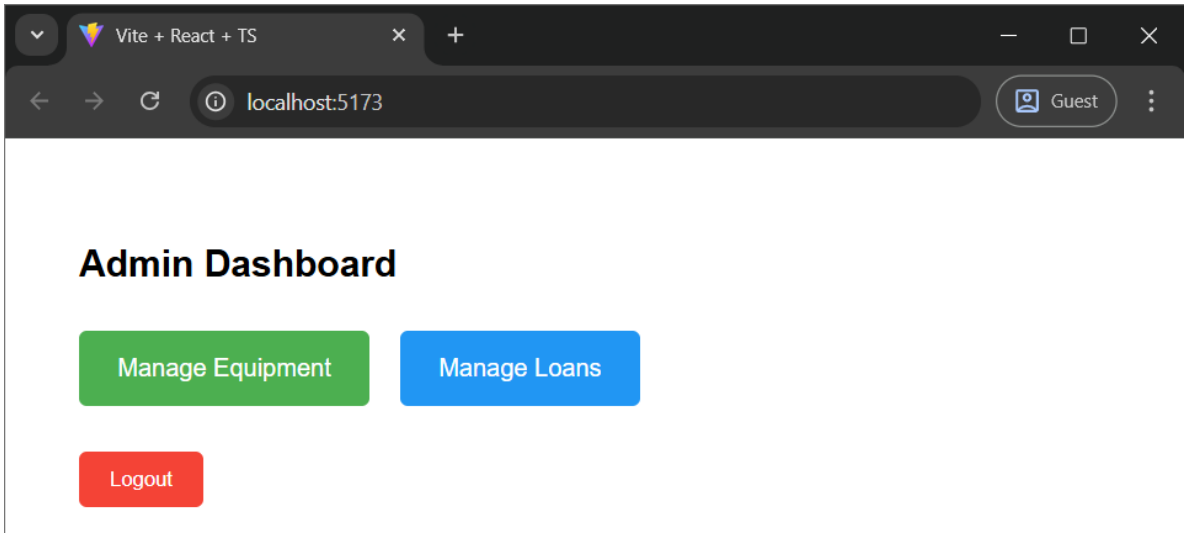

Рисунок 3.13 - Головна сторінка адміністратора

Сторінка керування обладнанням (Рисунок 3.14) забезпечувала адміністратора повним списком наявного обладнання. За її допомогою адміністратори могли додавати нове обладнання, змінювати існуючі записи або видаляти застарілі позиції, забезпечуючи актуальність інвентаря.

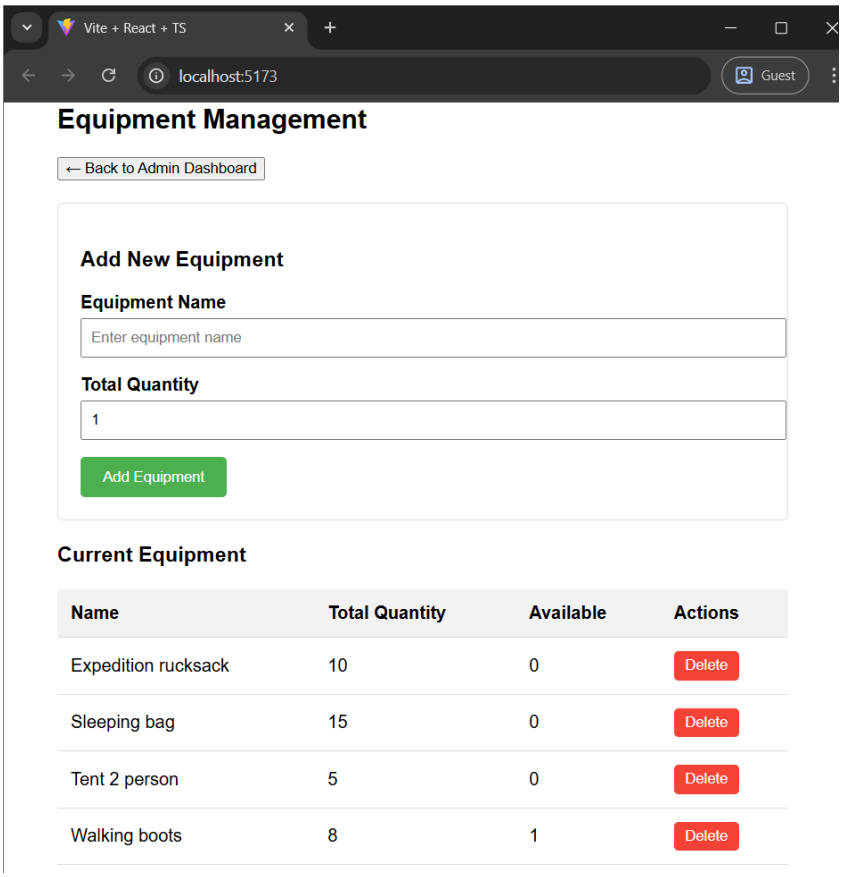

Рисунок 3.14 - Сторінка керування обладнанням в панелі адміністратора

Під час додавання нового предмета адміністратор вводив його назву у відповідне поле (Рисунок 3.15). Цей спрощений процес дозволяв швидко оновлювати інвентар у разі надходження нового обладнання.

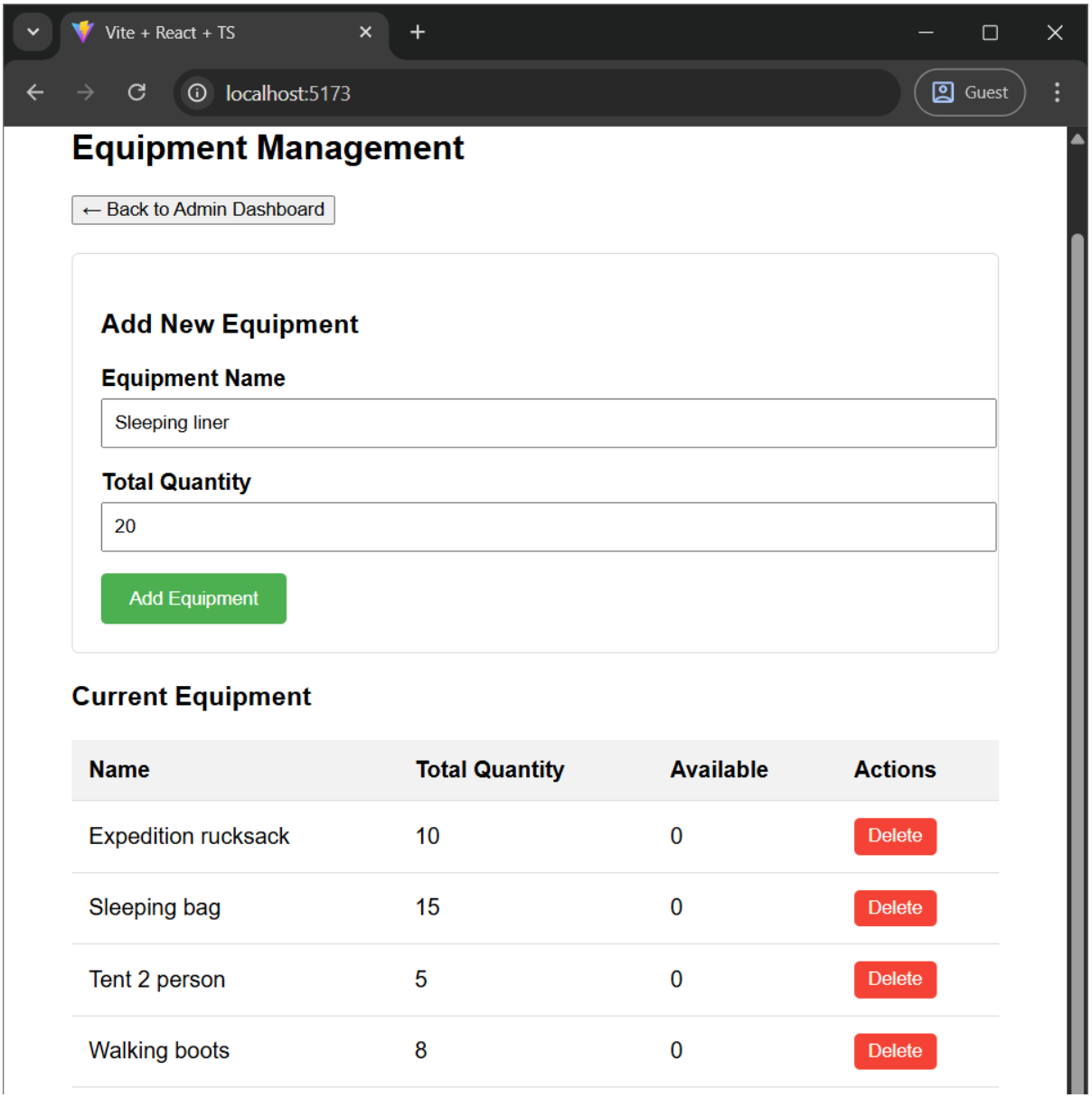

Рисунок 3.15 – Додавання нової назви обладнання

Після відправлення форми сторінка автоматично оновлювалася, показуючи актуальний список обладнання (Рисунок 3.16). Це негайно підтверджувало успішне збереження нового запису у базі даних та підтримувало ефективну організацію запасів.

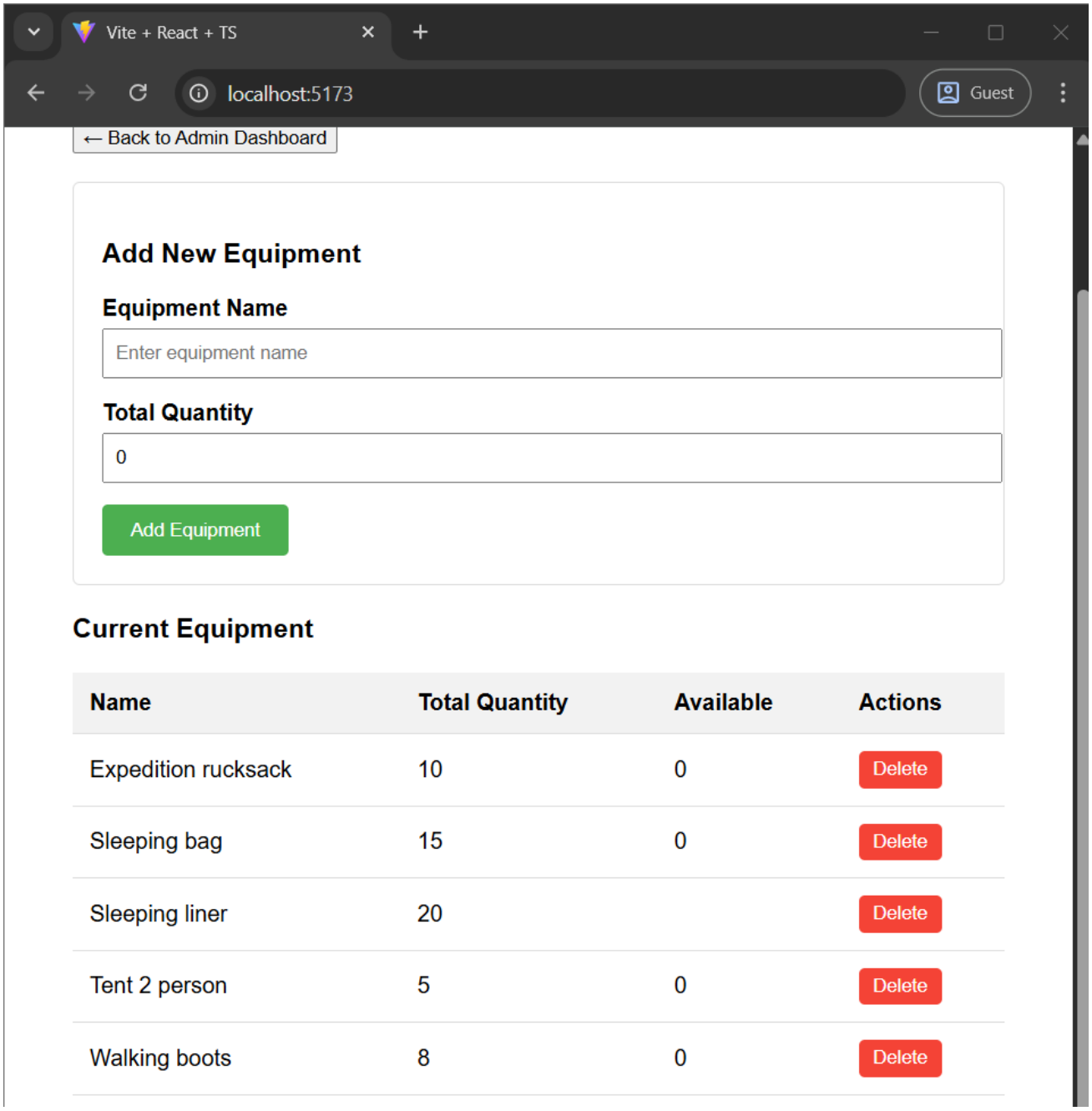

Рисунок 3.16– Оновлений список обладнання після додавання нової позиції

Нарешті, сторінка керування позиками адміністратора (Рисунок 3.17) дозволяла адміністраторам переглядати всі подані заявки на позику. Тут адміністратор міг затвердити, відхилити або відзначити предмети як повернені. Хоча сторінка ще перебувала в розробці, її макет був спроєктований так, щоб забезпечувати всебічне відображення всіх позик у системі [12].

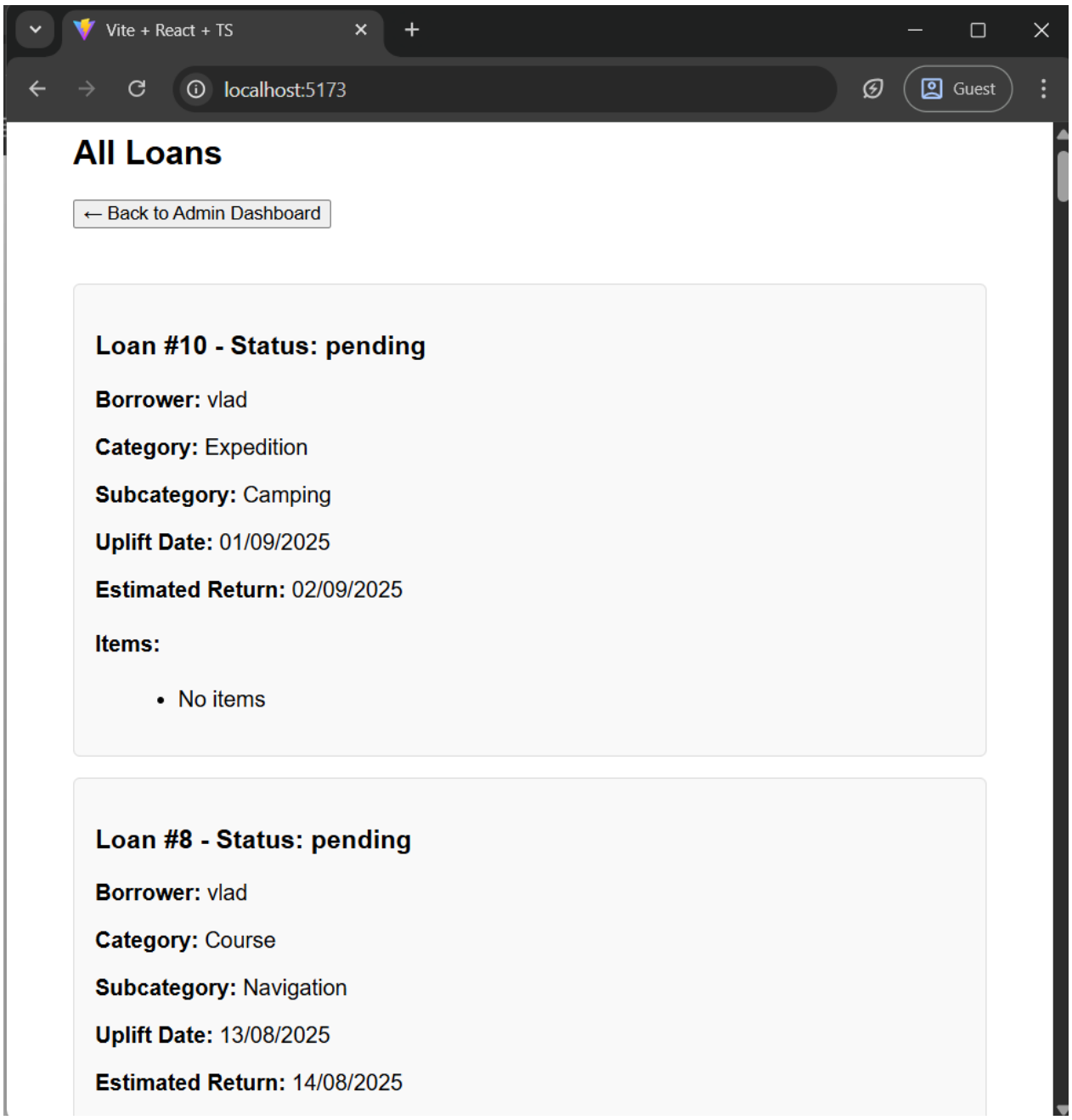

Рисунок 3.17 - Сторінка управління адміністративними позиками

Загалом фронтенд-сторінки були розроблені з урахуванням зручності використання, простоти та ефективності. Динамічні функції, включно з фільтруванням за категорією-підкатегорією та перевіркою доступності, наочно показували, як бекенд-логіка покращує користувацький досвід. Послідовні структури та адаптивний дизайн забезпечували безшовну взаємодію користувачів із системою, що було важливим для підтримання залученості та задоволеності.

3.3 Тестування, перевірка доступності та обробка помилок

Користувацький інтерфейс (UI) був розроблений відповідно до Web Content Accessibility Guidelines (WCAG) 2.1, щоб забезпечити доступність та інклюзивність [32]. Наприклад, текст і кнопки були створені з контрастністю, що перевищувала мінімальну вимогу WCAG AA – 4.5:1, щоб забезпечити кращу видимість для користувачів із порушеннями зору. Кроки навігації були свідомо зведені до мінімуму та логічно розділені між позичальниками й адміністраторами, щоб мінімізувати когнітивне навантаження та забезпечити ефективне виконання завдань [26, 31] (див. Рисунки 3.4 та 3.13).

Рисунок 3.18 ілюструє приклад перевірки введення в реальному часі під час входу, коли неправильні облікові дані, введені користувачем, миттєво відображались у вигляді чіткого повідомлення про помилку. Це відповідає принципам юзабіліті щодо попередження помилок і надання зворотного зв’язку [26].

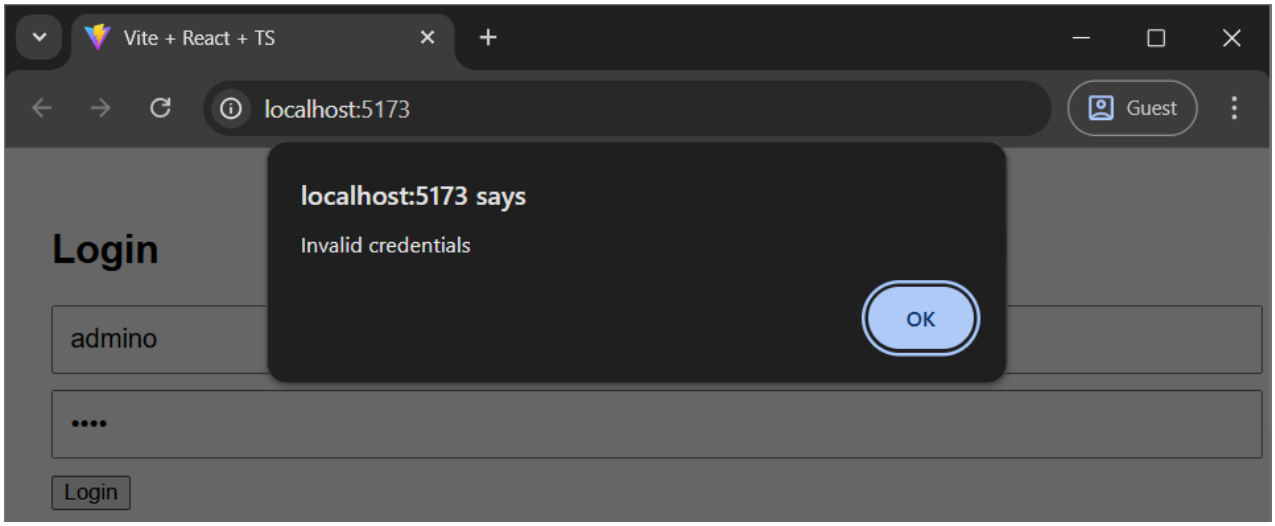

Рисунок 3.18 – Повідомлення про недійсні облікові дані для входу

Під час подання запиту на позику система вимагала від користувача обрати категорію у випадку, якщо жодну з них не було вибрано. Це гарантувало, що вся необхідна інформація була введена перед тим, як запит можна було обробити [31] (див. Рисунок 3.19).

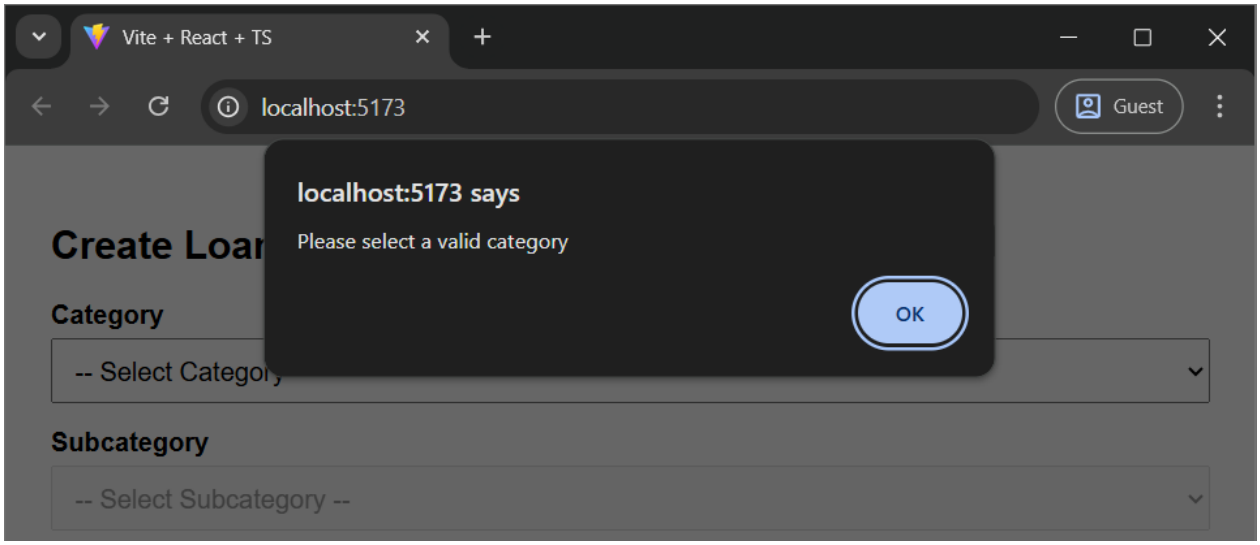

Рисунок 3.19 – Повідомлення про те, що користувач не обрав категорію замовлення

Аналогічна поведінка спостерігалась тоді, коли користувачі намагалися оформити позику без вибору обладнання або кількості. Інтерфейс показував повідомлення про помилку. Така валідація зменшувала кількість неповних заявок і запобігала потенційним адміністративним помилкам під час обробки позик [11, 25] (див. Рисунок 3.20).

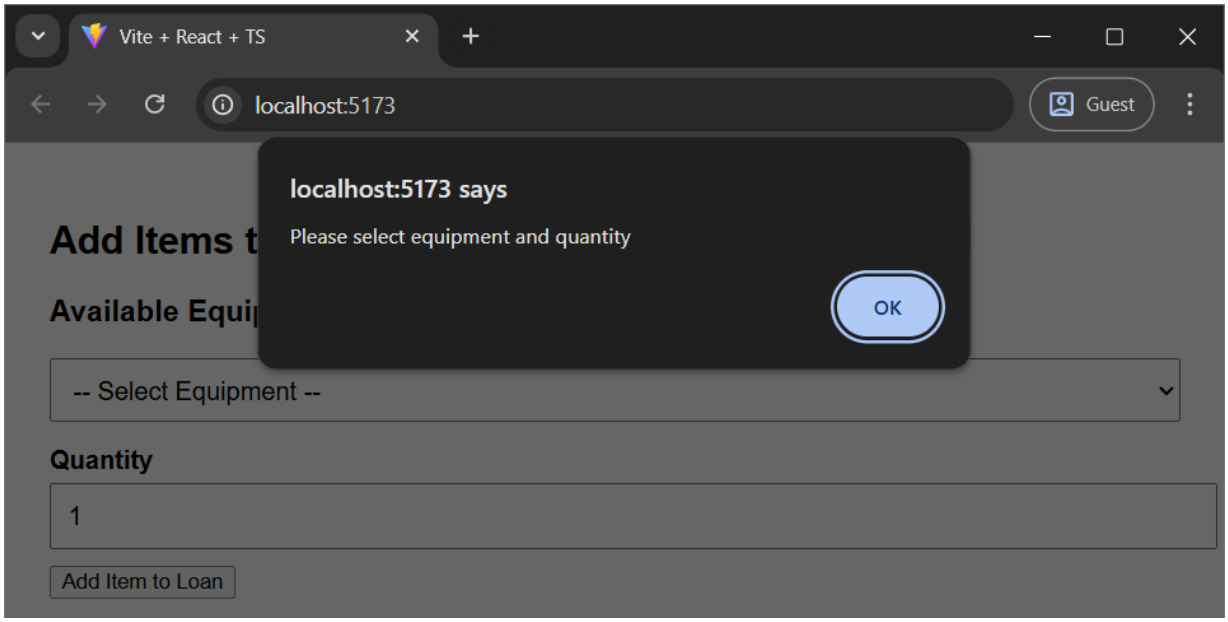

Рисунок 3.20 – Повідомлення про те, що користувач не обрав обладнання або кількість для додавання

Після успішного подання система повідомляла про завершення дії спеціальним повідомленням, що гарантувало користувачеві, що заявку на позику було надіслано та оброблено (див. Рисунок 3.21).

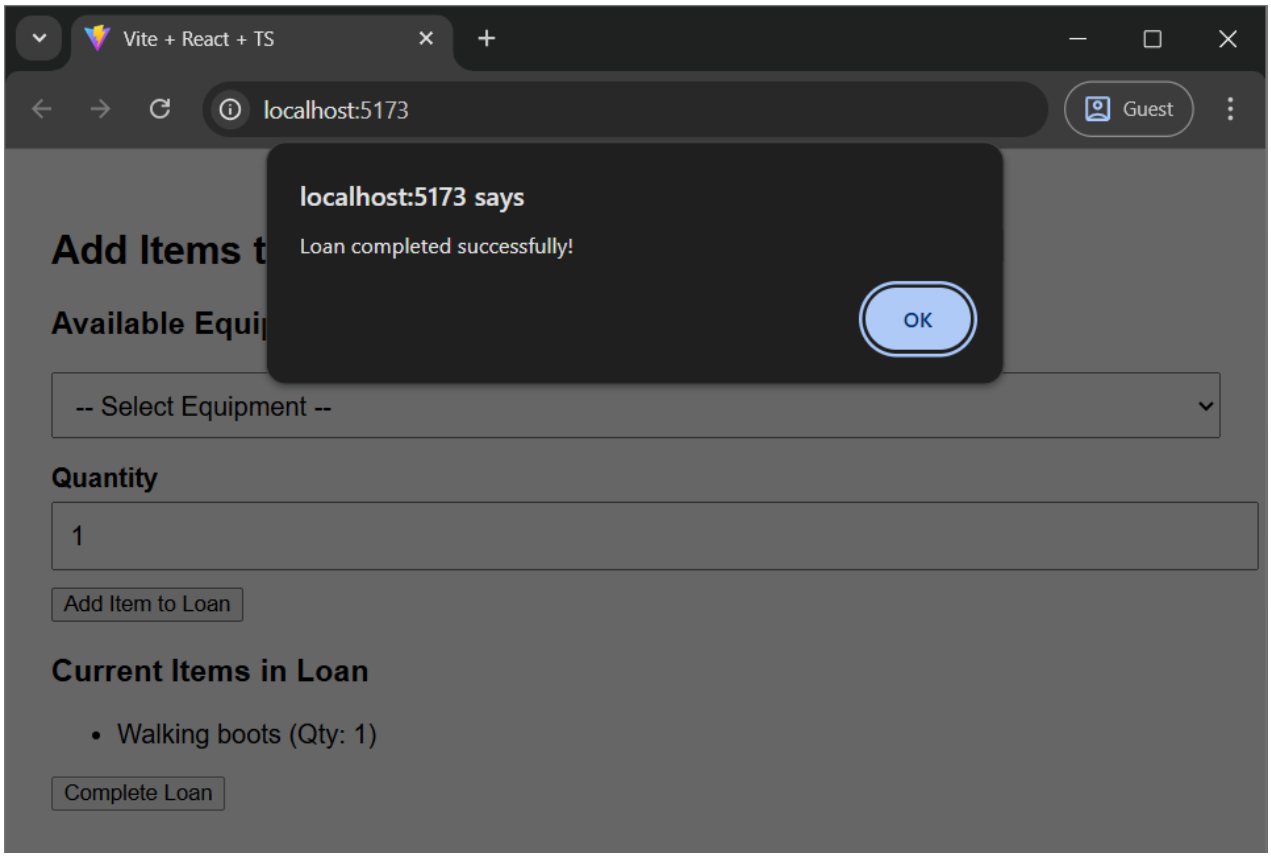

Рисунок 3.21 – Повідомлення про те, що замовлення оформлено

Так само, коли адміністратор видаляв запис про обладнання, система показувала сповіщення про те, що відповідний елемент було видалено (Рисунок 3.22). Такий захист забезпечував помітність критичних змін у запасах і зменшував ризик непомічених або випадкових видалень [22].

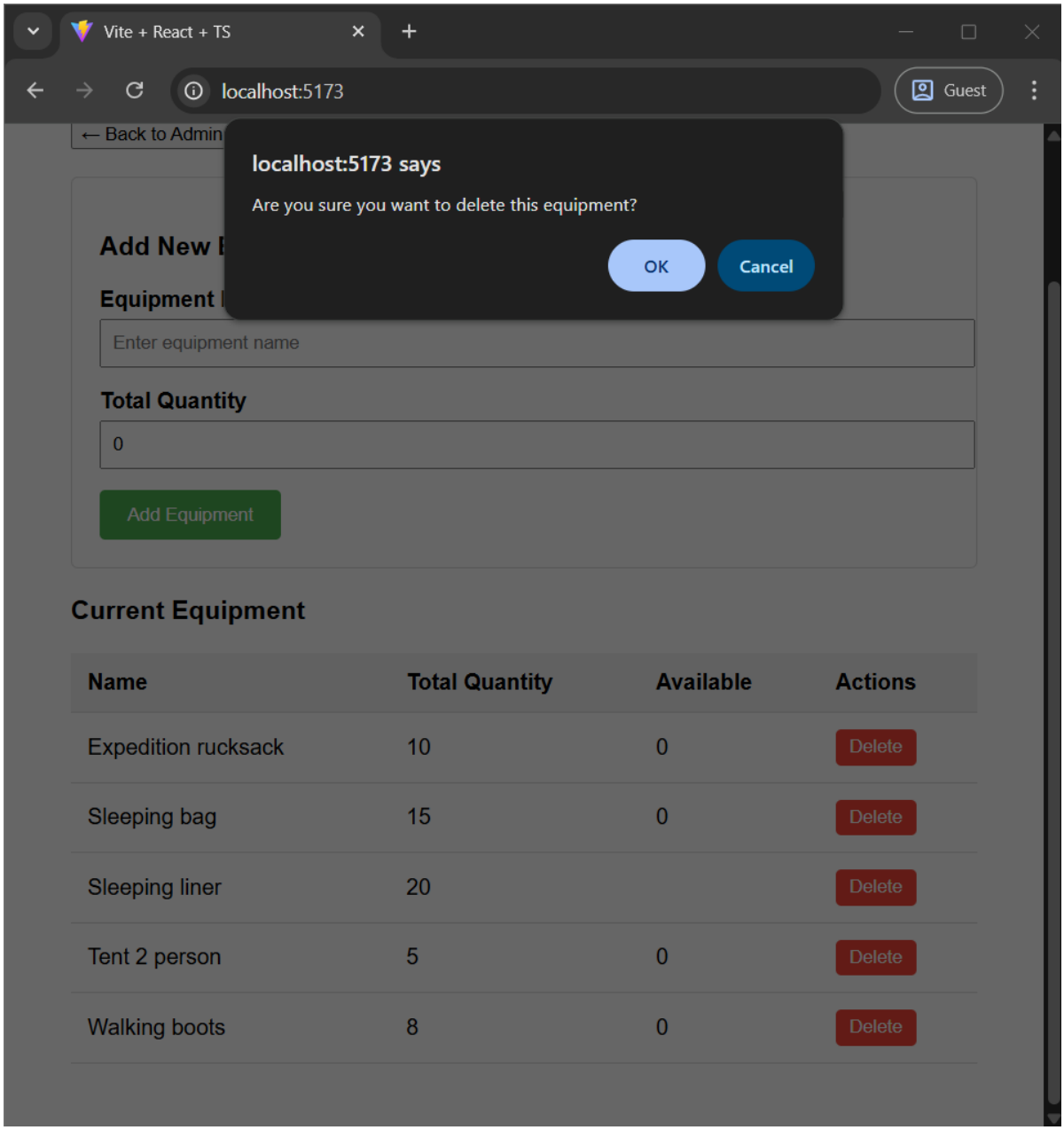

Рисунок 3.22 – Системне повідомлення, що підтверджує видалення обладнання

Список обладнання після цього автоматично оновлювався, відображаючи оновлений набір даних із видаленим елементом (див. Рисунок 3.23). Це негайне оновлення підсилювало точність даних і давало адміністраторам упевненість, що зміни були коректно застосовані до бази даних [7].

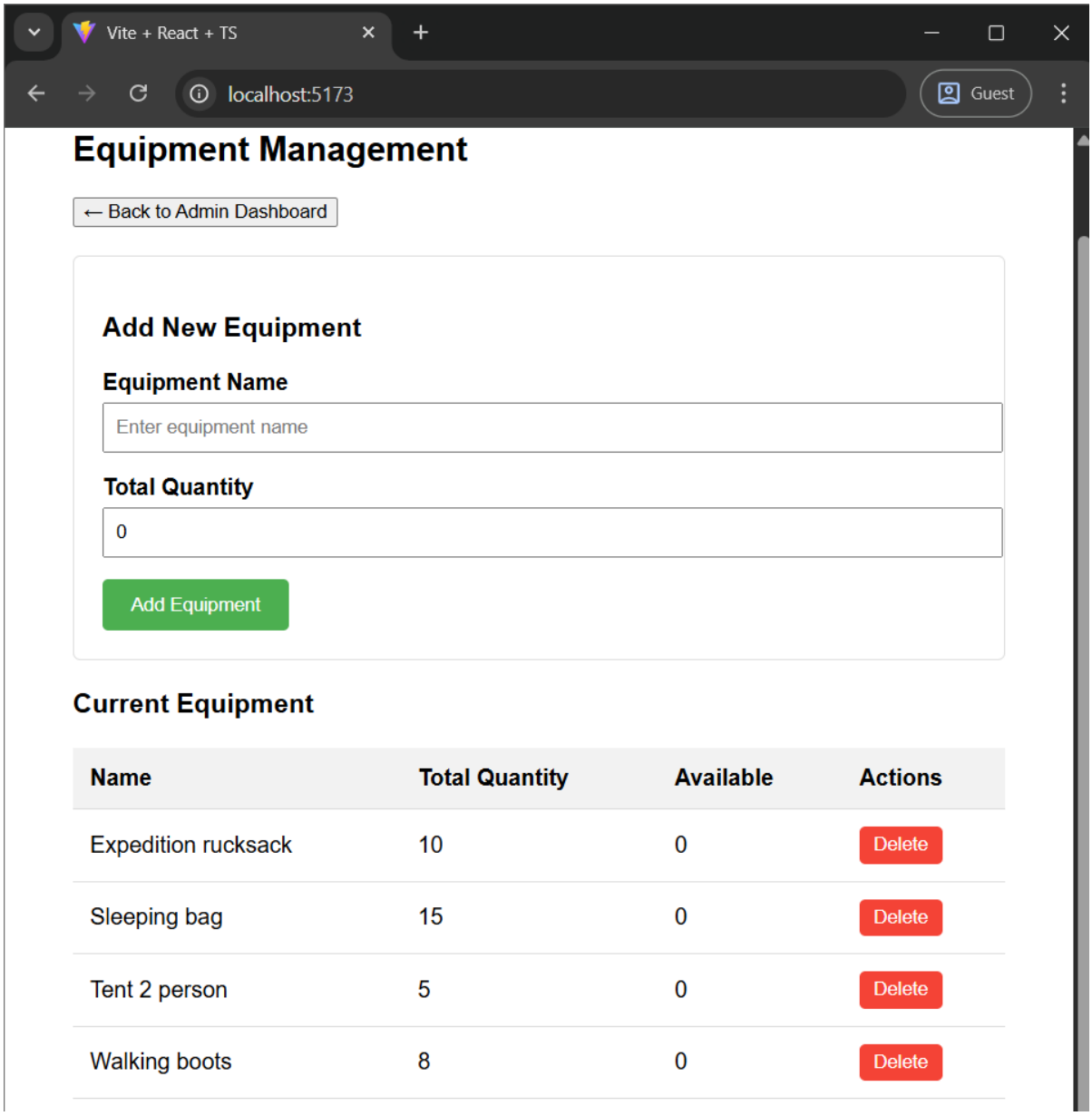

Рисунок 3.23 – Оновлений список обладнання після видалення

Ці функції доступності та валідації були включені на основі академічних рекомендацій, а також з урахуванням зворотного зв’язку від зацікавлених сторін. Члени команди розроблення з магазину наголошували на простоті та мінімізації помилок, що узгоджувалося з принципами, прийнятими академічною спільнотою [4]. Завдяки включенню вимог WCAG, теорії дизайну взаємодії та реального зворотного зв’язку від користувачів, система досягла балансу між теоретичною найкращою практикою та реальною зручністю використання [26, 32].

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Ергономічні проблеми безпеки життєдіяльності

Розроблення веб-орієнтованої системи прокату обладнання передбачає роботу оператора з комп'ютерною технікою протягом тривалого часу, що впливає на його працездатність та стан здоров'я. Відповідно до вимог безпеки життєдіяльності, необхідно забезпечити оптимальні умови праці для персоналу, який обслуговує систему, з урахуванням психофізіологічних факторів, ергономічних вимог до організації робочого місця та дотримання режимів праці й відпочинку [36, 39, 41].

При роботі з комп'ютерною технікою основними шкідливими факторами є: зорове навантаження, статичне навантаження на м'язи спини та шиї, електромагнітне випромінювання, підвищений рівень шуму, а також психоемоційне напруження, пов'язане з відповідальністю за коректність обробки даних. Для мінімізації впливу цих факторів необхідно дотримуватись гігієнічних вимог до організації робочого місця, зокрема: забезпечити раціональне розміщення монітора (на відстані 50–70 см від очей, з верхнім краєм на рівні очей), використовувати регульоване крісло з підтримкою поперекового відділу хребта, забезпечити достатнє природне та штучне освітлення робочої зони без утворення відблисків на екрані [35, 36, 37, 40, 42].

Важливим аспектом безпеки життєдіяльності є дотримання режиму праці та відпочинку. Згідно з гігієнічними нормативами, тривалість безперервної роботи з комп'ютерною технікою не повинна перевищувати 2 годин, після чого необхідно робити перерву тривалістю 10–15 хвилин для відпочинку очей та виконання фізичних вправ для зняття статичного навантаження. Загальна тривалість роботи з комп'ютером протягом робочого дня не повинна перевищувати 6 годин [42].

У разі виникнення надзвичайних ситуацій (пожежа, ураження електричним струмом, раптове погіршення самопочуття) персонал повинен

бути ознайомлений з правилами надання долікарської допомоги та порядком евакуації. При ураженні електричним струмом необхідно негайно вимкнути джерело живлення, оцінити стан постраждалого, при відсутності свідомості та дихання розпочати серцево-легеневу реанімацію (непрямий масаж серця та штучне дихання) до прибуття медичних працівників. У разі виникнення пожежі слід негайно повідомити пожежну службу (тел. 101) та розпочати евакуацію з приміщення згідно з планом евакуації [39].

Таким чином, дотримання вимог безпеки життєдіяльності при розробленні та експлуатації веб-орієнтованої системи прокату обладнання дозволяє мінімізувати ризики для здоров'я персоналу, забезпечити комфортні умови праці та підвищити ефективність роботи системи [41].

4.2 Етико-правове регулювання та стандарти доступності в інформаційних технологіях

Проєктування веборієнтованої системи прокату обладнання порушило певні правові, соціальні, етичні та професійні питання (LSEPI), а також питання рівності, різноманіття та інклюзії (EDI). Ці аспекти були розглянуті, щоб гарантувати, що система буде не лише функціональною й безпечною, але також справедливою, інклюзивною та відповідально розробленою [10, 33].

З правового погляду головним питанням було дотримання UK General Data Protection Regulation (UK GDPR) [6] та Data Protection Act 2018 [8]. Оскільки система обробляла персональні дані, а саме контактну інформацію, ім'я та історію позик – найважливішим завданням було забезпечити, щоб дані оброблялися чесно, законно та прозоро. Для виконання цих вимог система шифрувала дані як під час їхнього зберігання, так і під час передачі, застосовувала контроль доступу на основі ролей (RBAC) для обмеження доступу до чутливих даних і анонімізувала персональні дані, де це було можливо. Крім того, було впроваджено чітку політику зберігання даних, щоб персональні дані не зберігалися довше, ніж потрібно, і видалялися безпечно,

коли вони більше не були потрібні. Усі ці кроки відповідали рекомендаціям Information Commissioner’s Office [13] та демонстрували дотримання принципу «захисту даних за проєктом і за замовчуванням».

Соціальні та етичні чинники також були важливими, оскільки проєкт мав уникати виключення або дискримінації окремих груп користувачів через спосіб використання системи. Система була розроблена таким чином, щоб врахувати користувачів із різним рівнем технічних навичок і не створити різкого переходу з ручної процедури до цифрової. Крім того, система мала запобігати зловживанням, включаючи несанкціоноване використання або шахрайські бронювання, та забезпечувати прозорість того, як обробляються дані користувачів [34].

Високі стандарти професійності були забезпечені відповідно до BCS Code of Conduct [4], особливо тих його положень, що стосуються суспільних інтересів, професійної компетентності та чесності, а також відповідальності перед відповідними органами. Це означало застосування безпечних практик кодування, проведення всебічного тестування для виявлення та усунення вразливостей, документування процесу розроблення та створення системи, яка є надійною й придатною для підтримки. Проєкт передбачав відкриту комунікацію зі стейкхолдерами, регулярні оновлення про прогрес та отримання зворотного зв’язку, щоб гарантувати, що система відповідатиме їхнім потребам.

Система також сприяла просуванню принципів рівності, різноманіття та інклюзії (EDI). Це було досягнуто шляхом забезпечення доступності для всіх користувачів незалежно від їхніх фізичних або когнітивних можливостей. Система відповідала Web Content Accessibility Guidelines (WCAG) 2.1 [32], які є глобально визнаними стандартами доступності цифрового контенту. Вона включала функції доступності, такі як підтримка скринрідера, керування з клавіатури та наявність alt-текстів для користувачів із порушеннями зору. Тестування проводилось за участю різноманітної групи стейкхолдерів, щоб впевнитися, що система підходить широкому колу користувачів.

Насамкінець, елементи EDI та LSEPI були враховані як на етапі планування, так і на етапі розроблення. Система була спрямована на відповідність законодавству, дотримання принципів інклюзивного дизайну, таких як гнучкість використання та сприйнятність інформації (наприклад, використання високого контрасту в UI та масштабованих шрифтів), а також відповідність професійним стандартам інженерії програмного забезпечення. Ці рекомендації були адаптовані з універсальних практик дизайну, зосереджених на зручності для максимально широкого кола користувачів [26, 38]. У поєднанні з функціями, що відповідають WCAG 2.1, та постійним зворотним зв’язком від стейкхолдерів система стала етичною, безпечною та соціально відповідальною [4, 32].

ВИСНОВКИ

У першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр» подано огляд літературних джерел за темою автоматизації систем управління запасами та прокату обладнання, що дозволило визначити сучасні підходи до розроблення веб-орієнтованих систем. Розглянуто технічну архітектуру, інтеграцію баз даних, питання безпеки та відповідності нормативним вимогам, зокрема Data Protection Act 2018 та Online Safety Act 2023. Висвітлено ключові підходи до проєктування, орієнтованого на користувача, та рольове керування доступом (RBAC). Проаналізовано існуючу проблему в магазині туристичного спорядження, виявлено недоліки паперової системи прокату, сформульовано функціональні вимоги до системи та завдання розроблення.

У другому розділі кваліфікаційної роботи досліджено методологічні підходи до розроблення програмного забезпечення, зокрема застосування шаблону Model–View–Controller (MVC) та тришарової клієнт-серверної архітектури. Обґрунтовано вибір технологій реалізації: React для клієнтської частини, Express для серверної частини, PostgreSQL як реляційної системи керування базами даних. Сформовано ER-діаграму бази даних, що відповідає вимогам третьої нормальної форми (3NF), та реалізовано схему PostgreSQL із забезпеченням реляційної цілісності через обмеження FOREIGN KEY. Спроєктовано REST API для взаємодії між компонентами системи. Виконано планування проєкту з використанням ітеративного життєвого циклу, проведено оцінку ризиків та розроблено стратегії їх мінімізації.

У третьому розділі кваліфікаційної роботи розроблено веб-застосунок для автоматизації прокату обладнання, що включає автентифікацію та авторизацію користувачів, перегляд наявного обладнання з можливістю фільтрації, створення заявок на прокат з автоматичним присвоєнням статусу «pending». Запропоновано механізми перевірки доступності обладнання в реальному часі з автоматичним оновленням залишків та адміністративні функції управління обладнанням і затвердження позик. Спроєктовано архітектуру взаємодії

компонентів системи, реалізовано REST API ендпоінти для виконання основних операцій. Протестовано систему, включаючи перевірку API засобами Postman, кросбраузерне та мобільне тестування, а також перевірку відповідності принципам доступності WCAG 2.1 (контрастність, керування з клавіатури, повідомлення про помилки).

У розділі «Безпека життєдіяльності, основи охорони праці» висвітлено правові, соціальні, етичні та професійні аспекти (LSEPI), а також питання рівності, різноманіття та інклюзії (EDI), що виникли при проєктуванні веб-орієнтованої системи прокату обладнання. Систему спроєктовано з урахуванням принципів доступності для різних категорій користувачів та з дотриманням вимог законодавства щодо захисту даних.

Практичне значення одержаних результатів полягає в автоматизації ключових бізнес-процесів магазину прокату, що забезпечує зменшення часу на оформлення операцій, підвищення точності обліку обладнання, зниження ризику помилок, пов’язаних з людським фактором, покращення контролю за ресурсами через автоматичні сповіщення про прострочені повернення, а також можливість адаптації системи для використання в інших підрозділах ради або невеликих компаніях, що надають обладнання в оренду. Достовірність результатів підтверджується успішним тестуванням усіх основних функцій системи, відповідністю реалізованої бази даних принципам реляційної цілісності та позитивним зворотним зв’язком від зацікавлених сторін.

Виконання кваліфікаційної роботи дозволило суттєво розвинути навички full stack розроблення (React, Express, PostgreSQL), управління проєктами (ітеративне планування, управління ризиками, ведення журналу проєкту) та комунікації із зацікавленими сторонами. Отриманий досвід самостійної організації роботи, рефлексивної практики та інтеграції технічних рішень з академічним звітуванням є цінним для подальшої професійної діяльності в галузі розроблення програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ

1 Alkin Tezuysal and Ahmed, I. (2024). *Database Design and Modeling with PostgreSQL and MySQL*. Packt Publishing Ltd.

2 Anh, V. (2021). *Real-time backend architecture using Node.js, Express and Google Cloud Platform*. www.theseus.fi. Доступно за: https://urn.fi/URN:NBN:fi:amk-202102232616.

3 Bagui, S. and Earp, R. (2003) *Database Design Using Entity-Relationship Diagrams*. Доступно за: https://lira.epac.to/DOCS-TECH/DataBase/Design/Database%20Design%20Using%20Entity-Relationship%20Diagram.pdf.

4 BCS (2022) *BCS Code of Conduct | BCS*, *www.bcs.org*. Доступно за: https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/.

5 Brown, E. (2019). *Web Development with Node and Express*. O'Reilly Media.

6 Council regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons about the processing of personal data and the free movement of such data (United Kingdom General Data Protection Regulation)(Text with EEA relevance). Доступно за: https://www.legislation.gov.uk/eur/2016/679.

7 Creative Practice (2023). *Що таке JavaScript*. Cases.Media. Доступно за: https://cases.media/article/sho-take-javascript?srsltid=AfmBOopUKeWDSECM566H2UUO7DWEyKqj3_ucMzJgb3Tw8fZDqzYV7JYv.

8 Data Protection Act 2018, c. 12. Доступно за: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacte.

9 Duda, O., Kramar, T., Melnyk, A., Zakharia, O. and Skaletskyi, P. (2025). System Architecture of Data Organization for Smartcities Based on the Data Mesh Concept. *Journal of Lviv Polytechnic National University 'Information Systems*

and Networks', 17, p.pp. 411 - 424. Доступно за: https://science.lpnu.ua/sisn/all-volumes-and-issues/volume-17-2025/system-architecture-data-organization-smartcities-based.

10 GOV UK (2023) *Online Safety Act 2023*, *Legislation.gov.uk*. Доступно за: https://www.legislation.gov.uk/ukpga/2023/50/enacted.

11 Griggs, B. (2020). *Node Cookbook*. Packt Publishing Ltd.

12 Hlazunova, O., Andriushchenko, V., Korolchuk, V. And Voloshyna, T. (2024). Web-Oriented System Of The Electronic Dean's Office: Implementation Of The Precedent Of Forming The Student's Individual Plan. *Information Technology and Society*, (2 (13)), pp.26–33. doi:https://doi.org/10.32689/maup.it.2024.2.4.

13 ICO (2023) *Security, including cyber security*, *ico.org.uk*. Доступно за: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/security/.

14 Katircioglu, K., Brown, T.M. and Asghar, M. (2007) 'An SQL-based cost-effective inventory optimisation solution', *IBM journal of research and development*, 51(3–4), pp. 433–445. Доступно за: https://doi.org/10.1147/rd.513.0433.

15 Korotkevitch, D. (2022) 'SQL Server in the Cloud', in *SQL Server Advanced Troubleshooting and Performance Tuning*. United States: O'Reilly Media, Incorporated.

16 Le, Q.H. and Martelo, D. (2021). *Developing modern database applications with PostgreSQL : use the highly available object-relational database to build scalable and reliable apps*. Birmingham: Packt Publishing.

17 Li, C. and Gu, J. (2019) 'An integration approach of hybrid databases based on SQL in cloud computing environment', *Software, practice & experience*, 49(3), pp. 401–422. Доступно за: https://doi.org/10.1002/spe.2666.

18 Lorenzo Ochoa, O. *et al.* (2017) 'Integration through orchestration', *Journal of Enterprise Information Management*, 30(4), pp. 555–582. Доступно за: https://doi.org/10.1108/jeim-02-2016-0060.

19 Mammino, L. and Casciaro, M. (2025). *Node.js Design Patterns*. Packt Publishing Ltd.

20 Manelli, L. and Zambon, G. (2020) *Beginning Jakarta EE Web Development: Using JSP, JSF, MySQL, and Apache Tomcat for Building Java Web Applications*. 3rd ed. Berkeley, CA: Apress L. P. Доступно за: https://doi.org/10.1007/978-1-4842-5866-8.

21 Mannino, M. (2022). *Database Design*. SAGE Publications.

22 Obe, R.O. and Hsu, L.S. (2017). *PostgreSQL: Up and Running*. 'O'Reilly Media, Inc.'

23 Orobchuk, B., Babiuk, S., Buniak, O., Sysak, I. and Kostyk, L. (2023). Development of an educational laboratory stand at the base fast-acting automatic reserve input. *Scientific journal of the Ternopil national technical university*, 112(4), pp.12–25. doi:https://doi.org/10.33108/visnyk_tntu2023.04.012.

24 Panchal, Y. and Ravi Kumar Gupta (2024). *Building Scalable Web Apps with Node.js and Express*. Orange Education Pvt Ltd.

25 Piscatello, M. (2023). Reducing Structured Query Language Injection Vulnerabilities Through Functional Programming Principles. 18, pp.425–432. doi:https://doi.org/10.1109/southeastcon51012.2023.10114959.

26 Preece, J., Rogers, Y. and Sharp, H. (2015). *Interaction design: beyond human-computer interaction*. Chichester, West Sussex: John Wiley & Sons Ltd. Доступно за: https://books.google.com.ua/books?id=n0h9CAAAQBAJ&printsec=frontcover&hl=uk&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

27 S Pasaribu, J. (2021) 'Development of a Web-Based Inventory Information System', *International Journal of Engineering, Science and Information Technology*, 1(2), pp. 24–31. Доступно за: https://doi.org/10.52088/ijesty.v1i2.51.

28 Strutynska, I., Kozbur, H., Dmytrotsa, L., Bodnarchuk, I. and Hlado, O. (2019). Small and Medium Business Structures Clustering Method Based on Their Digital Maturity. *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*. doi:https://doi.org/10.1109/picst47496.2019.9061464.

29 Tkachenko, O., Tkachenko, K., Tkachenko, O. and Vozniuk, V. (2024). J-FINDER – A WEB-ORIENTED JOB SEARCH DECISION SUPPORT SYSTEM. *Cybersecurity: Education, Science, Technique*, 1(25), pp.355–378. doi:https://doi.org/10.28925/2663-4023.2024.25.355378.

30 Vadlamani, V. (2024). Introduction to PostgreSQL Database Management. In: PostgreSQL Skills Development on Cloud. Apress, Berkeley, CA. https://doi.org/10.1007/979-8-8688-0817-3_1

31 Vyas, R. (2022). Comparative Analysis on Front-End Frameworks for Web Applications. *International Journal for Research in Applied Science and Engineering Technology*, 10(7), pp.298–307. doi:https://doi.org/10.22214/ijraset.2022.45260.

32 Web Content Accessibility Guidelines (WCAG). Доступно за: https://www.w3.org/TR/WCAG/.

33 Верховна Рада України, 1996. *Конституція України (із змінами і доповненнями)*. Київ: Відомості Верховної Ради України.

34 Верховна Рада УРСР, 1971. *Кодекс законів про працю України (із змінами і доповненнями)*. Київ: Відомості Верховної Ради.

35 Держспоживстандарт України, 2004. *ДСТУ ISO 9241-5:2004. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози*. Київ: Держспоживстандарт України.

36 Держспоживстандарт України, 2011. *ДСТУ 7234:2011. Дизайн і ергономіка. Обладнання виробниче. Загальні вимоги дизайну та ергономіки*. Київ: Держспоживстандарт України.

37 ДП «УкрНДНЦ», 2015. *ДСТУ 8604:2015. Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги*. Київ: ДП «УкрНДНЦ».

38 ДП «УкрНДНЦ», 2021. *ДСТУ ISO/TS 16976-8:2021. Засоби індивідуального захисту органів дихання. Людські чинники. Частина 8. Ергономічні чинники*. Київ: ДП «УкрНДНЦ».

39 Жидецький, В.Ц., 2020. *Охорона праці користувачів комп'ютерів: підручник*. Львів: Афіша.

40 Мінекономрозвитку України, 2013. *ДСТУ 7299:2013. Дизайн і ергономіка. Робоче місце оператора*. Київ: Мінекономрозвитку України.

41 Мінсоцполітики України, 2018. *НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями*. Затверджено наказом від 14.02.2018 №207. Київ: Міністерство соціальної політики України.

42 МОЗ України, 1998. *ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин*. Київ: Міністерство охорони здоров'я України.

43 Мосій, Л., Козбур, Г., Струтинська, І., Мосій, О. and Яцишин, В., (2024). *Information technology to support the digital transformation of small and medium-sized businesses.*

44 Нікітіна, Т.С. and Морозова, О.І. (2019). Порівняльний аналіз продуктивності баз даних SQL та NOSQL. *Системи управління, навігації та зв'язку*, [online] 1(1), pp.125–128. Доступно за: http://www.irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?I21DBN=LINK&P21DBN=UJRN&Z21ID=&S21REF=10&S21CNR=20&S21STN=1&S21FMT=ASP_meta&C21COM=S&2_S21P03=FILA=&2_S21STR=suntz_2019_1_26.

ДОДАТКИ

ДОДАТОК А

Список всього обладнання

1. Rucksacks
   - Expedition rucksack
   - Day rucksack
   - Team backpack cover
   - Rucksack liner
2. Sleeping bag/mat
   - Liner bag
   - Sleeping bag
   - Sleeping bag liner
   - Sleeping mat
3. Insect protection
   - Midge net
4. Walking equipment
   - Hi-Viz vest
   - Walking poles
   - Head torches
5. Warmers
   - Hat
   - Gloves
   - Fleece
6. Jacket
   - Waterproof jacket S
   - Waterproof jacket M
   - Waterproof jacket L
   - Waterproof jacket XL
   - Waterproof jacket XXL
7. Trousers

Waterproof trousers S

Waterproof trousers M

Waterproof trousers L

Waterproof trousers XL

Waterproof trousers XXL

8. Boots

Walking boots size 1/32

Walking boots size 2/34

Walking boots size 3/36

Walking boots size 4/37

Walking boots size 5/38

Walking boots size 6/39

Walking boots size 6 ½ /40

Walking boots size 7/41

Walking boots size 8/42

Walking boots size 9/43

Walking boots size 9½ /44

Walking boots size 10/45

Walking boots size 11/46

Walking boots size 12/47

Walking boots size 13/48

Gaiters

9. Tent/shelter

Tent 2-person

Tent 3-person

Tent for the gold expedition

Tent 1 person

Group shelter

10. Stove

Stove S

Stove L

11. Fuel

    Fuel 0.5 L

    Fuel 1,0 L

12. First Aid

    First aid kit

    Survival bag

13. Orienteering

    Compass

    Map case

ДОДАТОК Б

Детальна діаграма Ганта

	Sc...	Task Name	Duration	Start	End
3		⊟ **Week 1**	**5 days**	**10/02/2025**	**16/02/2025**
4		Arrange and conduct informal interviews with store staff	1 day	10/02/2025	10/02/2025
5		Observe how they currently handle booking/lending	1 day	10/02/2025	10/02/2025
6		Ask what problems they face	1 day	10/02/2025	10/02/2025
7		Make basic notes	1 day	10/02/2025	10/02/2025
8		Summarise findings	4 days	11/02/2025	14/02/2025
9		⊟ **Week 2**	**5 days**	**17/02/2025**	**21/02/2025**
10		Write a first draft of the project objectives	5 days	17/02/2025	21/02/2025
11		Define user types	1 day	17/02/2025	17/02/2025
12		Clarify system boundaries	1 day	18/02/2025	18/02/2025
13		Create a list of core features	2 days	19/02/2025	20/02/2025
14		Finalise the requirements document	1 day	21/02/2025	21/02/2025
15		⊟ **Week 3**	**5 days**	**24/02/2025**	**28/02/2025**
16		Search for similar small-scale inventory or booking systems	1 day	24/02/2025	24/02/2025
17		Take screenshots or notes of interesting features/layouts.	1 day	25/02/2025	25/02/2025
18		Write a short literature/market review summary	2 days	24/02/2025	25/02/2025
19		Note any legal/data issues	1 day	26/02/2025	26/02/2025
20		Collect or bookmark useful tools and references for development.	2 days	27/02/2025	28/02/2025
21		⊟ **Week 4**	**2 days**	**03/03/2025**	**04/03/2025**
22		Make a basic diagram of how users interact with the system	1 day	03/03/2025	03/03/2025
23		List main components of the system	1 day	03/03/2025	03/03/2025
24		Decide tech stack	1 day	04/03/2025	04/03/2025
25		Submit TMA01	1 day	04/03/2025	04/03/2025
26		⊟ **Phase 2**	**45 days**	**04/03/2025**	**05/05/2025**

Рисунок Б.1– Діаграма Ганта, фаза 1

	Task Name	Duration	Start	End
27	⊟ **Week 5-6**	**8 days**	**05/03/2025**	**14/03/2025**
28	Collected and reviewed all information from the equipment store.	2 days	05/03/2025	06/03/2025
29	Grouped equipment and categories	2 days	07/03/2025	10/03/2025
30	Discussed with staff how the tracking will work	4 days	11/03/2025	14/03/2025
31	Created a rough draft of the database	2 days	13/03/2025	14/03/2025
32	⊟ **Week 7-8**	**10 days**	**17/03/2025**	**28/03/2025**
33	Identified what types of users will interact with the system	2 days	18/03/2025	19/03/2025
34	Reflected on how booking works in other equipment websites and systems.	3 days	19/03/2025	21/03/2025
35	Took notes on useful UX patterns from other websites	2 days	24/03/2025	25/03/2025
36	Drafted key user actions, Listed initial feature requirements	3 days	26/03/2025	28/03/2025
37	Meet with the tutor	1 day	28/03/2025	28/03/2025
38	⊟ **Week 9-10**	**10 days**	**31/03/2025**	**11/04/2025**
39	Finalise entity-relationship structure	5 days	31/03/2025	04/04/2025
40	Define attributes and validation rules for each entity	4 days	03/04/2025	08/04/2025
41	Design basic data flow	3 days	09/04/2025	11/04/2025
42	Define scenarios for how data will be queried and updated	2 days	09/04/2025	10/04/2025
43	Update database plan based on new insights from design reflection..	1 day	11/04/2025	11/04/2025
44	⊟ **Week 11**	**5 days**	**28/04/2025**	**02/05/2025**
45	Write a detailed description of the database structure	3 days	28/04/2025	30/04/2025
46	Include design analysis and user logic based on research of similar systems.	2 days	28/04/2025	29/04/2025
47	Explain how database supports user goals and aligns with project requireme…	2 days	29/04/2025	30/04/2025
48	Summarise what have been done so far	1 day	01/05/2025	01/05/2025
49	Submit TMA02	1 day	02/05/2025	02/05/2025

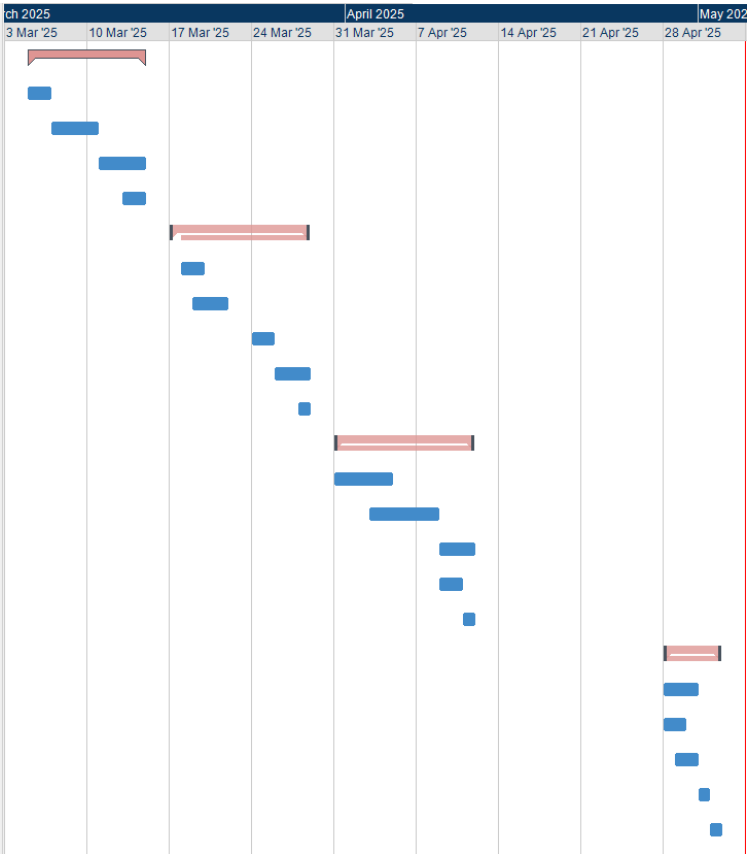

Рисунок Б.2 – Діаграма Ганта, фаза 2

	Task Name	Duration	Start	End
51	⊟ **Week 12-13**	**10 days**	**05/05/2025**	**16/05/2025**
52	Create wireframes or mockups for key screens	2 days	05/05/2025	06/05/2025
53	Decide layout structure	1 day	07/05/2025	07/05/2025
54	Choose tools and libraries	1 day	08/05/2025	08/05/2025
55	Set up project repository and initial file structure.	1 day	09/05/2025	09/05/2025
56	Start implementing the landing page and basic navigation components.	5 days	12/05/2025	16/05/2025
57	⊟ **Week 14-15**	**10 days**	**19/05/2025**	**30/05/2025**
58	Build the equipment list page	2 days	19/05/2025	20/05/2025
59	Connect frontend to database	3 days	21/05/2025	23/05/2025
60	Implement logic to display availability based on current bookings.	2 days	26/05/2025	27/05/2025
61	Add forms for booking request	1 day	28/05/2025	28/05/2025
62	Meeting with tutor to show first working prototype	2 days	29/05/2025	30/05/2025
63	⊟ **Week 16-17**	**10 days**	**02/06/2025**	**13/06/2025**
64	Implement login and basic user role logic	2 days	02/06/2025	03/06/2025
65	Add ability to approve or reject bookings	3 days	04/06/2025	06/06/2025
66	Add return-tracking functionality	1 day	09/06/2025	09/06/2025
67	Do internal testing	3 days	10/06/2025	12/06/2025
68	Write short feedback report	1 day	13/06/2025	13/06/2025
69	⊟ **Week 18-19**	**10 days**	**30/06/2025**	**11/07/2025**
70	Refactor code	4 days	30/06/2025	03/07/2025
71	Write summary for TMA03	3 days	03/07/2025	07/07/2025
72	Final testing to ensure booking system works	3 days	08/07/2025	10/07/2025
73	Submit TMA03	1 day	11/07/2025	11/07/2025
74	⊟ **Phase 4**	**45 days**	**14/07/2025**	**12/09/2025**

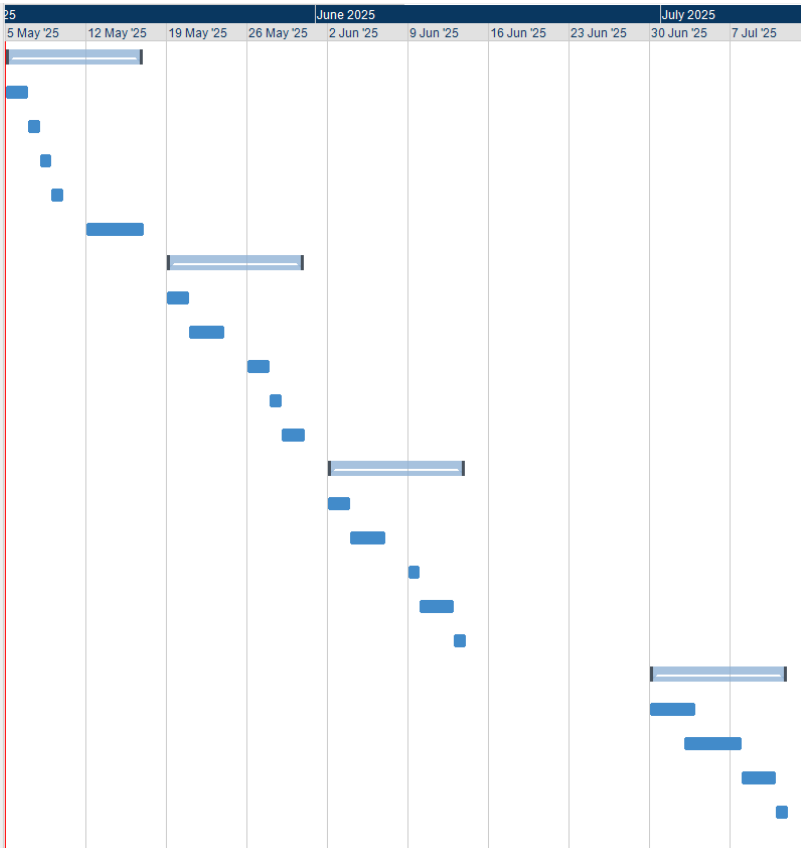

Рисунок Б.3 – Діаграма Ганта, Фаза 3

	Task Name	Duration	Start	End
75	**Week 20-21**	**10 days**	**14/07/2025**	**25/07/2025**
76	Review feedback from testing and users.	2 days	14/07/2025	15/07/2025
77	Check code based on feedback	3 days	16/07/2025	18/07/2025
78	Implement additional features or modifications	2 days	21/07/2025	22/07/2025
79	Conduct final usability testing with real users	2 days	23/07/2025	24/07/2025
80	Polish final touches	1 day	25/07/2025	25/07/2025
81	**Week 22-23**	**10 days**	**04/08/2025**	**15/08/2025**
82	Choose a cloud platform for deployment	1 day	04/08/2025	04/08/2025
83	Set up deployment pipelines and deploy the web app	2 days	05/08/2025	06/08/2025
84	Test deployed version for bugs and functionality	4 days	07/08/2025	12/08/2025
85	Ensure scalability, security and performance.	2 days	12/08/2025	13/08/2025
86	Write a deployment report, explaining the steps and issues encountered.	1 day	14/08/2025	14/08/2025
87	Meeting with tutor to discuss deployment status.	5 days	11/08/2025	15/08/2025
88	**Week 24-25**	**10 days**	**18/08/2025**	**29/08/2025**
89	Collect feedback from all stakeholders	2 days	18/08/2025	19/08/2025
90	Analyze the feedback	1 day	20/08/2025	20/08/2025
91	Perform final bug fixes	3 days	21/08/2025	25/08/2025
92	Conduct performance testing	4 days	25/08/2025	28/08/2025
93	Write the Final Evaluation Report	4 days	26/08/2025	29/08/2025
94	**Week 26-27**	**10 days**	**01/09/2025**	**12/09/2025**
95	Complete and finalize the project documentation	5 days	01/09/2025	05/09/2025
96	Prepare final report for each section	4 days	08/09/2025	11/09/2025
97	Submit EMA	1 day	12/09/2025	12/09/2025

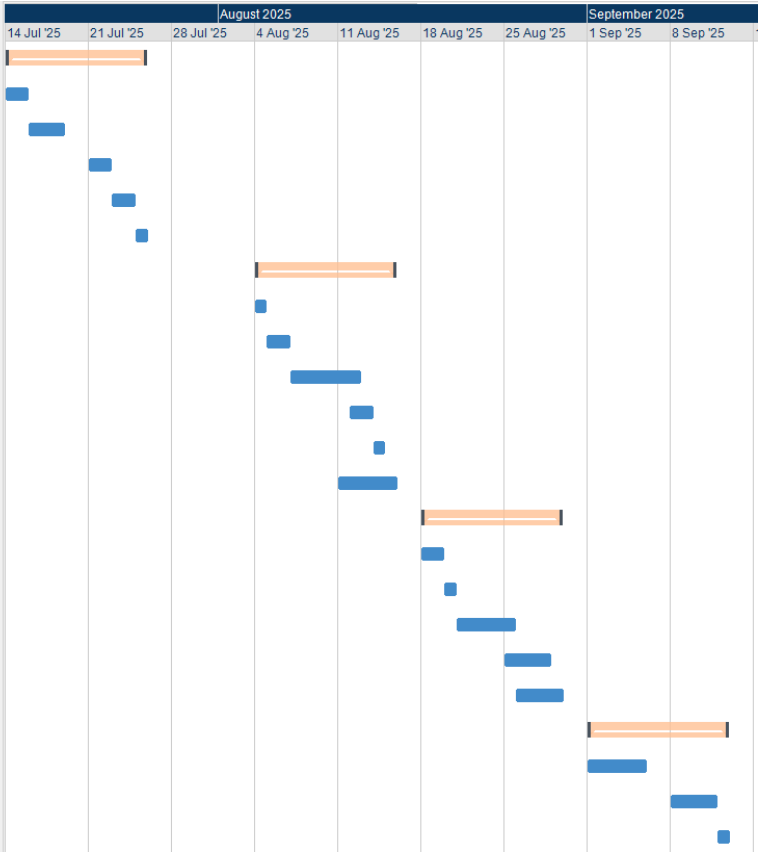

Рисунок Б.4– Діаграма Ганта, фаза 4

ДОДАТОК В

SQL Запит для імпорту бази даних з dbdatabase.io до PostgreSQL

```
CREATE TABLE "Users" (
  "id" integer PRIMARY KEY,
  "username" varchar,
  "password_hash" varchar,
  "user_type_id" integer NOT NULL,
  "contact_name" varchar,
  "group_name" varchar,
  "group_address" text,
  "phone" varchar
);

CREATE TABLE "UserType" (
  "id" integer PRIMARY KEY,
  "name" varchar
);

CREATE TABLE "Equipment" (
  "id" integer PRIMARY KEY,
  "name" varchar,
  "type_id" integer NOT NULL,
  "size_id" integer,
  "total_quantity" integer,
  "available_quantity" integer,
  "notes" text
);

CREATE TABLE "EquipmentType" (
  "id" integer PRIMARY KEY,
  "name" varchar
);

CREATE TABLE "Size" (
  "id" integer PRIMARY KEY,
  "label" varchar
);

CREATE TABLE "LoanForm" (
  "id" integer PRIMARY KEY,
  "borrower_id" integer NOT NULL,
  "category_id" integer NOT NULL,
  "subcategory_id" integer,
  "uplift_date" date,
  "return_date_estimated" date,
  "return_date_actual" date,
  "submitted_at" timestamp,
  "accepted_by" integer,
  "returned_by" integer,
  "status" varchar
```

```
);

CREATE TABLE "LoanItem" (
  "id" integer PRIMARY KEY,
  "loanform_id" integer NOT NULL,
  "equipment_id" integer NOT NULL,
  "quantity" integer
);

CREATE TABLE "Category" (
  "id" integer PRIMARY KEY,
  "name" varchar
);

CREATE TABLE "Subcategory" (
  "id" integer PRIMARY KEY,
  "name" varchar,
  "category_id" integer
);

COMMENT ON COLUMN "UserType"."name" IS 'e.g. Manager,
Borrower, Admin';
ALTER TABLE "Users" ADD FOREIGN KEY ("user_type_id")
REFERENCES "UserType" ("id");
ALTER TABLE "Equipment" ADD FOREIGN KEY ("type_id")
REFERENCES "EquipmentType" ("id");
ALTER TABLE "Equipment" ADD FOREIGN KEY ("size_id")
REFERENCES "Size" ("id");
ALTER TABLE "LoanItem" ADD FOREIGN KEY ("loanform_id")
REFERENCES "LoanForm" ("id");
ALTER TABLE "LoanItem" ADD FOREIGN KEY ("equipment_id")
REFERENCES "Equipment" ("id");
ALTER TABLE "LoanForm" ADD FOREIGN KEY ("borrower_id")
REFERENCES "Users" ("id");
ALTER TABLE "LoanForm" ADD FOREIGN KEY ("category_id")
REFERENCES "Category" ("id");
ALTER TABLE "LoanForm" ADD FOREIGN KEY ("subcategory_id")
REFERENCES "Subcategory" ("id");
ALTER TABLE "LoanForm" ADD FOREIGN KEY ("accepted_by")
REFERENCES "Users" ("id");
ALTER TABLE "LoanForm" ADD FOREIGN KEY ("returned_by")
REFERENCES "Users" ("id");
```

ДОДАТОК Г

Database data in all tables

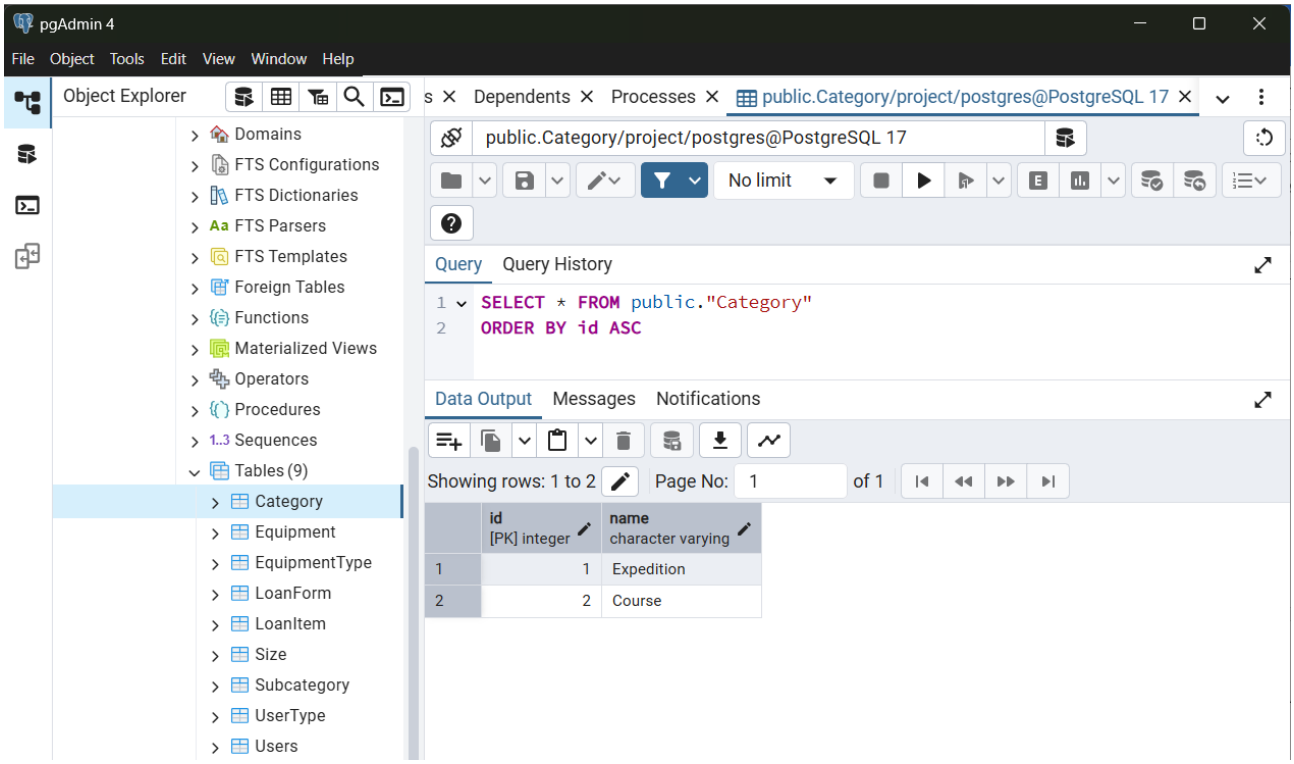

Рисунок Г.1 – Дані в таблиці «Category»

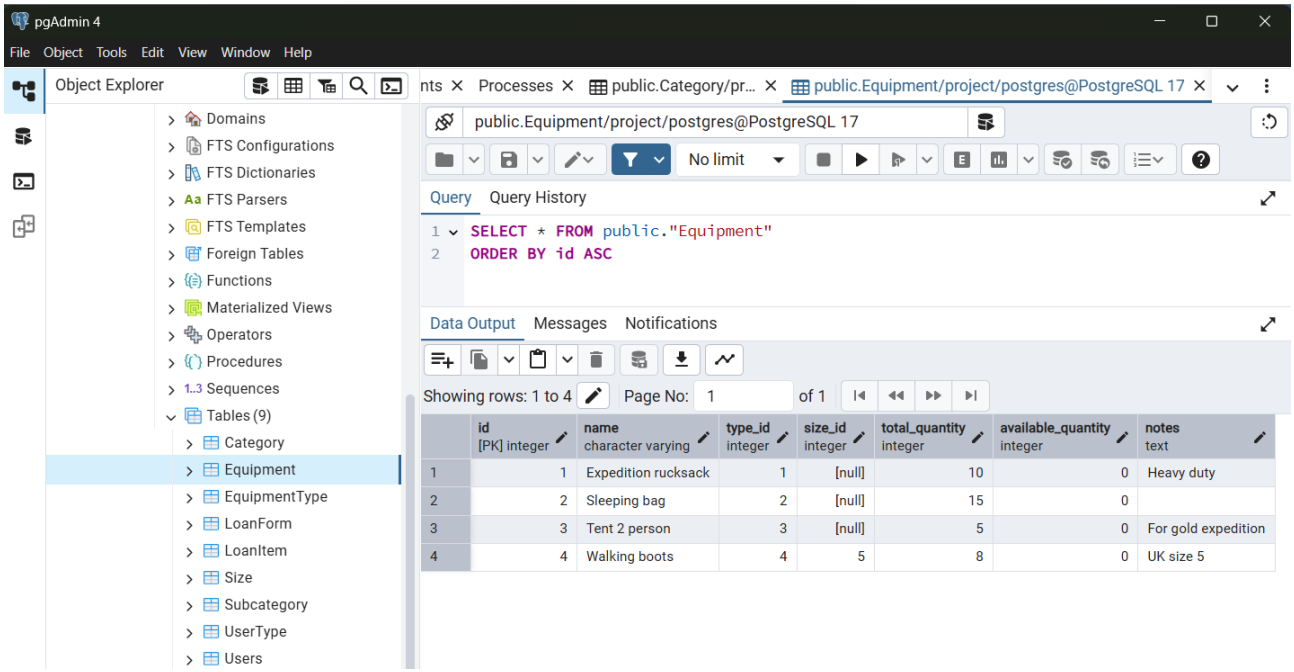

Рисунок Г.2 – Дані в таблиці «Equipment»

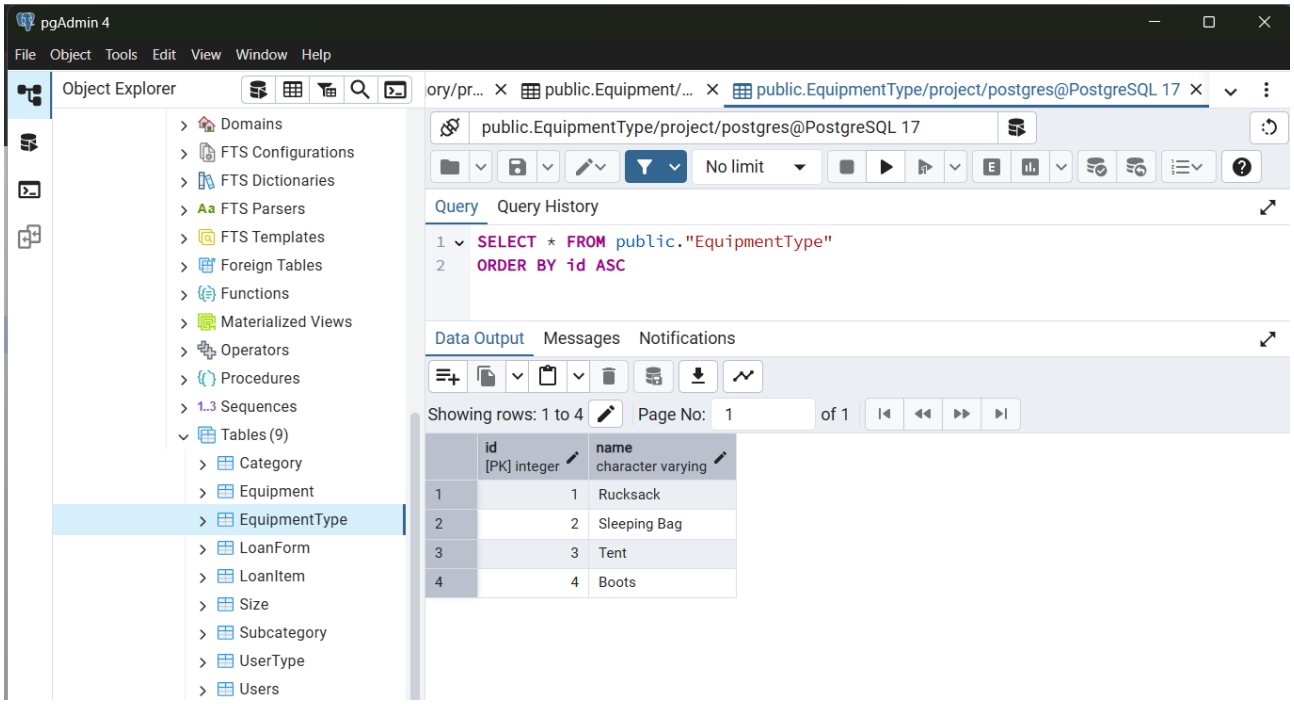

Рисунок Г.3 – Дані в таблиці «EquipmentType»

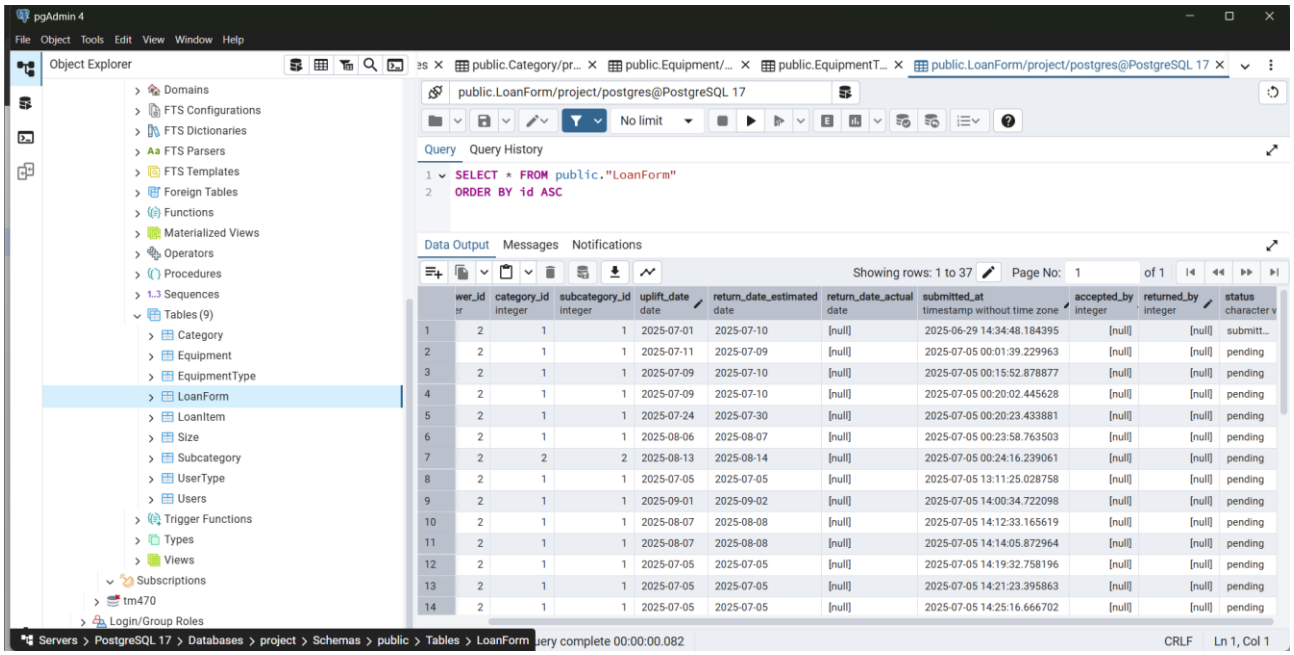

Рисунок Г.4 – Дані в таблиці «LoanForm»

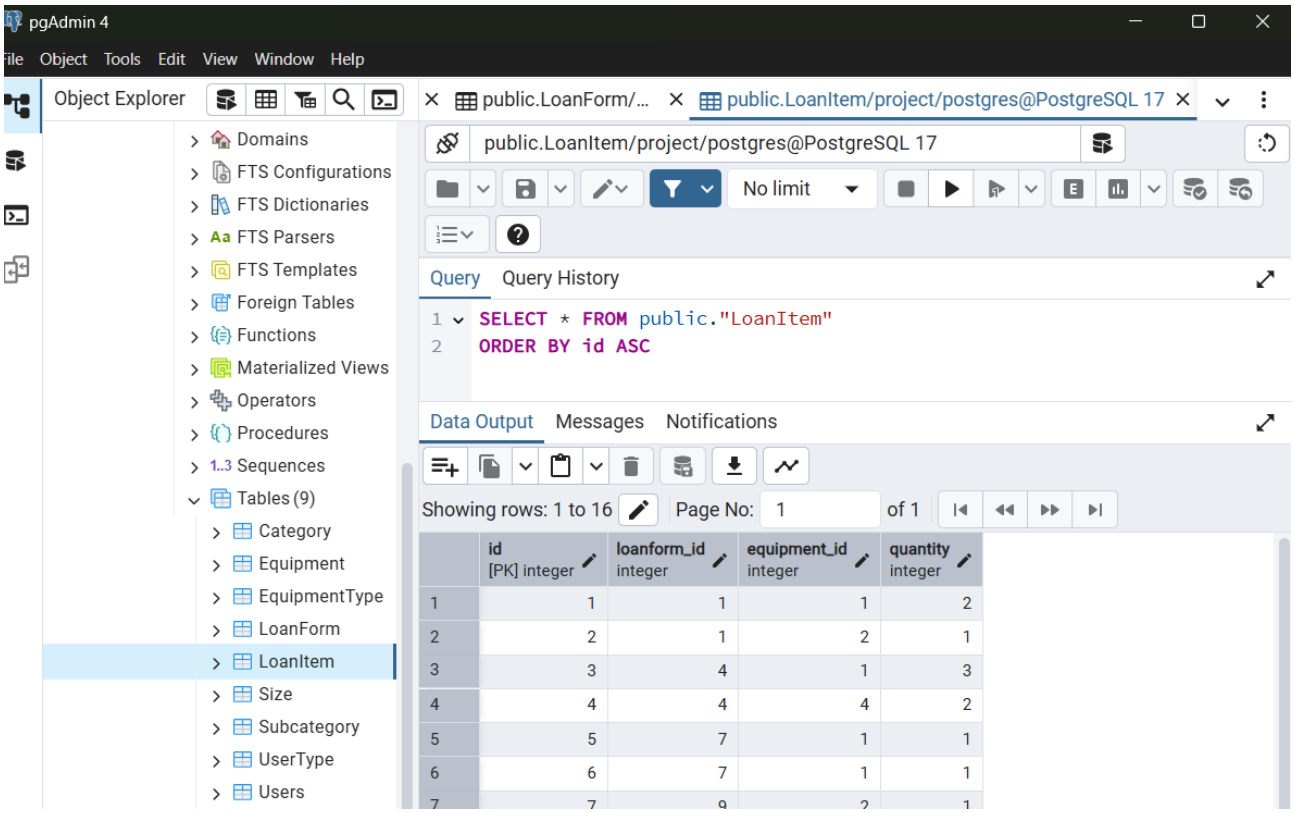

Рисунок Г.5 – Дані в таблиці «LoanItem»

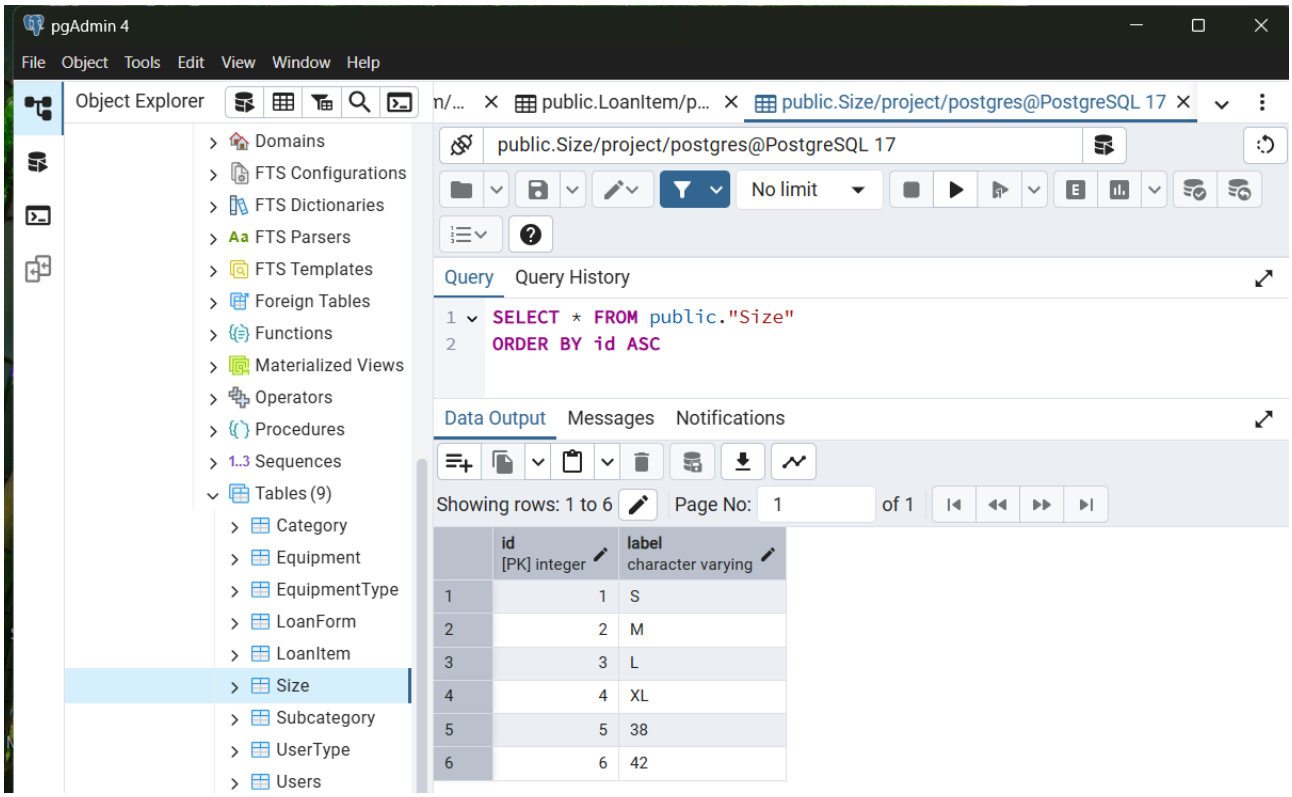

Рисунок Г.6 – Дані в таблиці «Size»

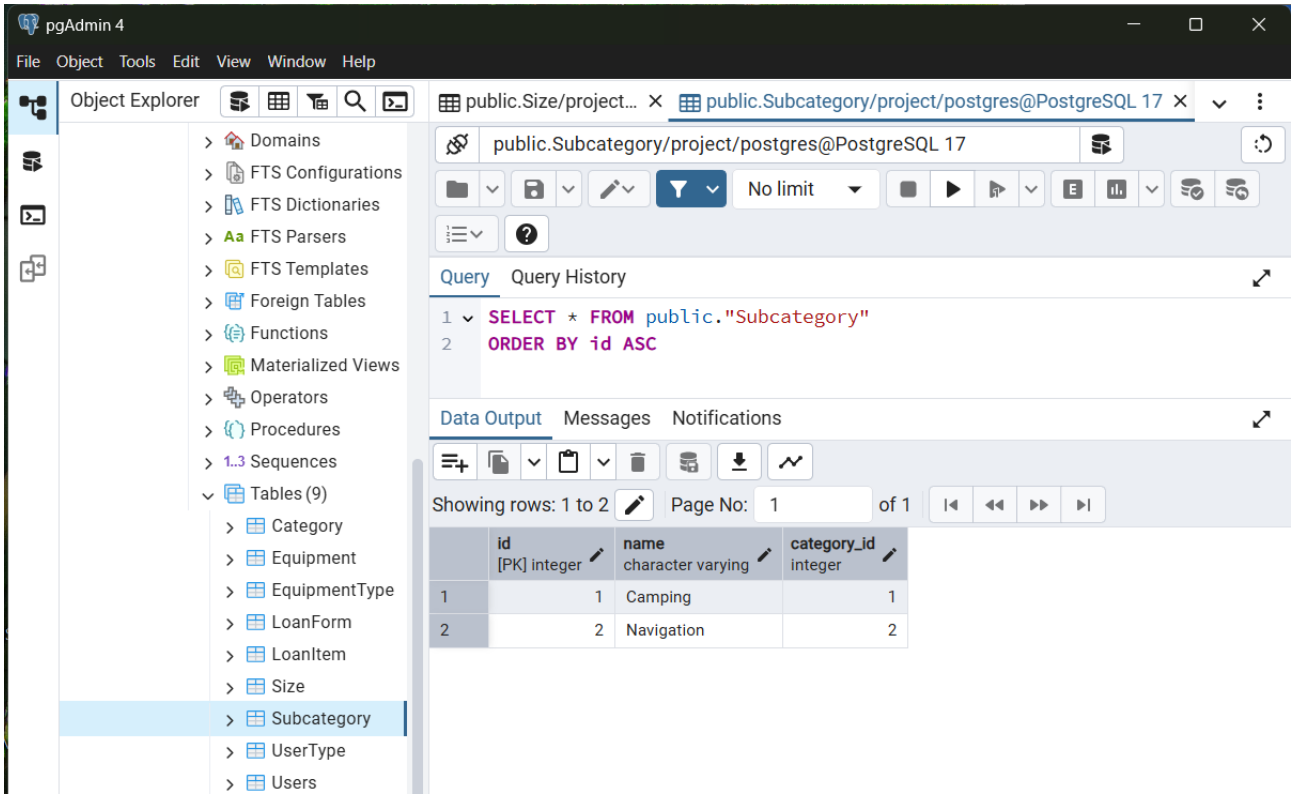

Рисунок Г.7 – Дані в таблиці «Subcategory»

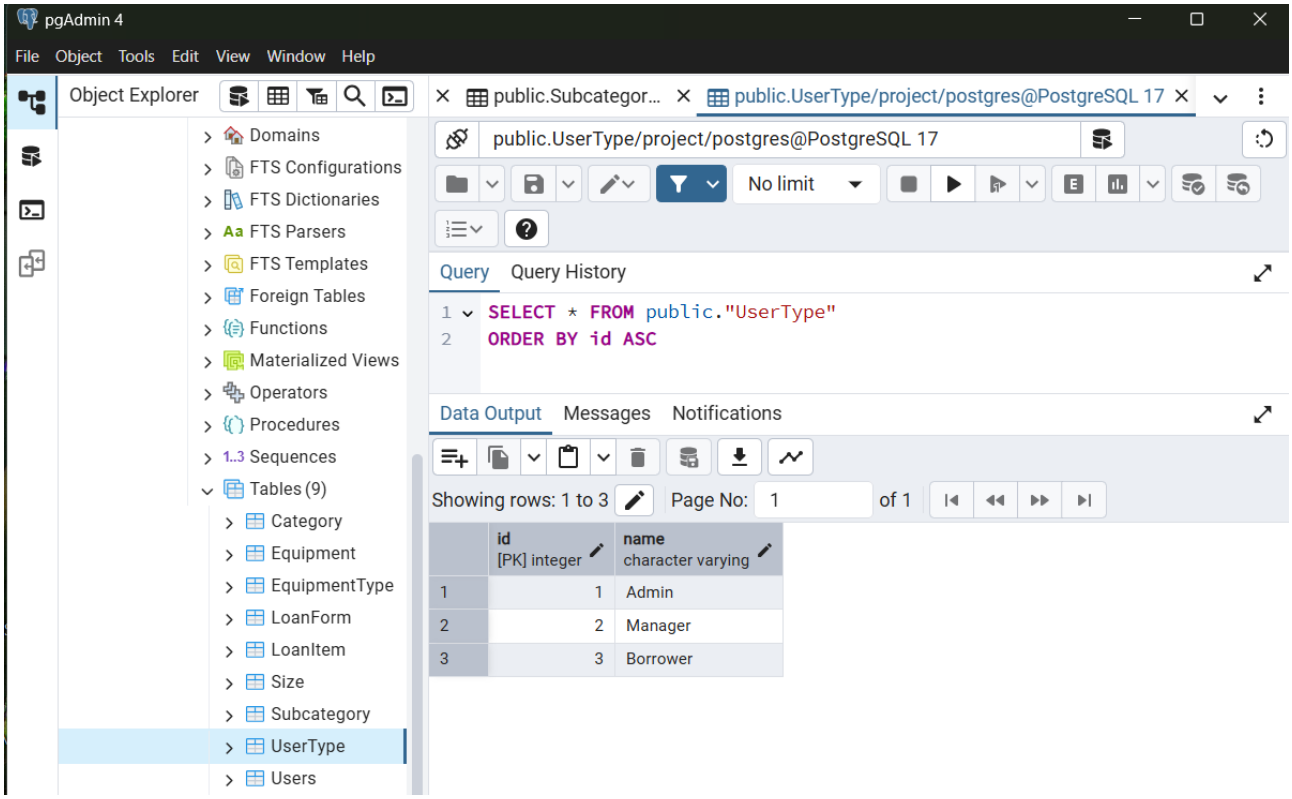

Рисунок Г.8 – Дані в таблиці «UserType»

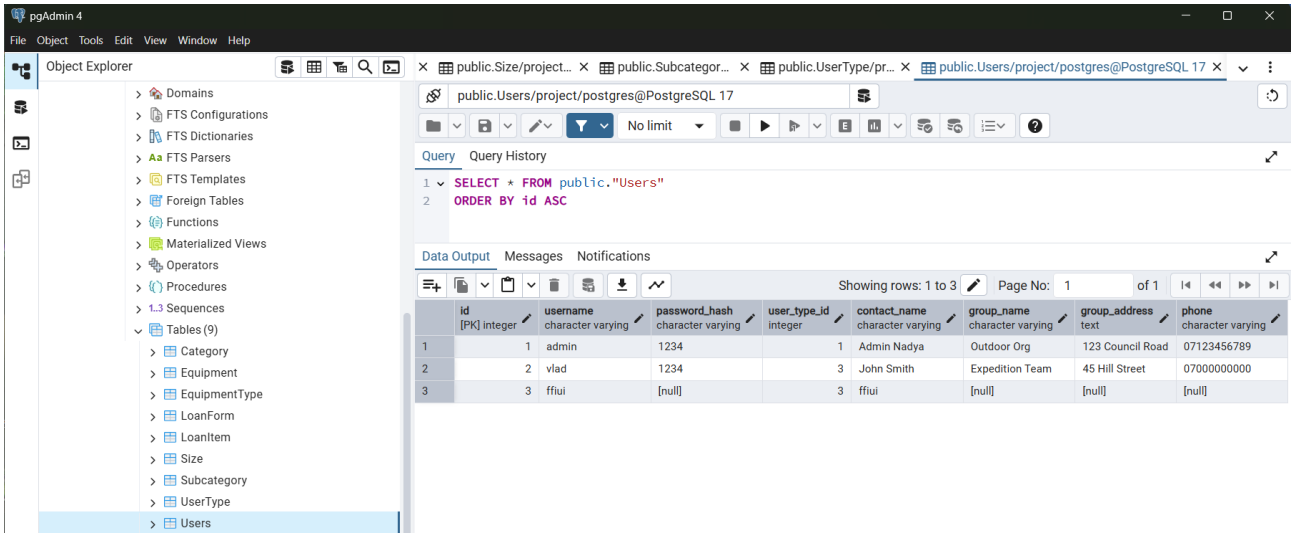

Рисунок Г.9 – Дані в таблиці «Users»

ДОДАТОК Д

Код для сторінки «Login»

Лістинг Д.1 – сторінка входу (frontend)

```
  // Login function
  const login = async () => {
    try {
      const response = await fetch('http://localhost:3000/login',
{
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password }),
      });

      const data = await response.json();
      if (!response.ok) throw new Error(data.error || 'Login
failed');

      if (data.status === 'success') {
        setUser(data.user);
        setStage(data.user.user_type_id === 1 ? 'adminHome' :
'userHome');
      } else {
        throw new Error(data.error || 'Login failed');
      }
    } catch (err) {
      const errorMessage = err instanceof Error ? err.message :
'Login failed';
      alert(errorMessage);
    }
  };
// return part with stage Login
return (
    <div style={styles.container}>
      {/* Login */}
      {stage === 'login' && (
        <div>
          <h2>Login</h2>
          <input
            placeholder="Username"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
            style={styles.input}
          />
          <input
            placeholder="Password"
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            style={styles.input}
```

```
          />
          <button onClick={login}>Login</button>
        </div>
      )}
```

Лістинг Д.2 – Сторінка входу (backend)

```
  app.post('/login', async (req: Request, res: Response) => {
    const { username, password } = req.body;

    if (!username || !password) {
      return res.status(400).json({ error: 'Username and password
required' });
    }

    try {
      // Simple authentication (replace with proper password
hashing)
      const result = await pool.query(
        `SELECT id, username, user_type_id
         FROM "Users"
         WHERE username = $1 AND password_hash = $2`,
        [username, password]
      );

      if (result.rows.length > 0) {
        const user = result.rows[0];
        return res.json({
          status: 'success',
          user: {
            id: user.id,
            username: user.username,
            user_type_id: user.user_type_id
          }
        });
      } else {
        return res.status(401).json({ error: 'Invalid credentials'
});
      }
    } catch (err) {
      console.error('Login error:', err);
      return res.status(500).json({ error: 'Database error' });
    }
  });
```