МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Проєктування та розробка frontend-сервісу для відстеження тари та її вмісту на базі React»**

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Мостовий Віталій Олександрович

Керівник: *старший викладач кафедри інформаційних технологій та аналітики даних*
Клебан Юрій Вікторович
Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних
_____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від 20 травня 2026 р.

Острог, 2026

АНОТАЦІЯ

кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка frontend-сервісу для відстеження тари та її вмісту на базі React.

Автор: ***Мостовий Віталій Олександрович***

Науковий керівник: старший викладач кафедри економіко-математичного моделювання та інформаційних технологій ***Клебан Юрій Вікторович***

Захищена «……»…… 2026 року.

Пояснювальна записка до кваліфікаційної роботи: 96 с., 12 рис., 3 табл., 6 додатків, 30 джерел.

Ключові слова: FRONTEND, REACT, СКЛАДСЬКІ ОПЕРАЦІЇ, ОБЛІК ТАРИ, ПАКУВАННЯ, ЛОГІСТИКА, ВЕБЗАСТОСУНОК, АВТОМАТИЗАЦІЯ, UX-СЦЕНАРІЇ, КОНТРОЛЬ ЗАЛИШКІВ

Короткий зміст праці:

У кваліфікаційній роботі розглянуто процес проєктування та розробки frontend сервісу TrackTara, призначеного для відстеження тари та її вмісту в межах складських операцій. Актуальність теми зумовлена необхідністю підвищення ефективності роботи складів, зменшення кількості помилок під час обробки замовлень та забезпечення прозорості логістичних процесів. У роботі проаналізовано сучасні підходи до організації обліку товарів і тари, визначено ключові проблеми, які виникають під час пакування, формування замовлень і відвантаження, зокрема складність контролю залишків, ризик помилок оператора та відсутність зручних інструментів для швидкої обробки інформації. Окрему увагу приділено вимогам до користувацького інтерфейсу, який має забезпечувати інтуїтивність, швидкість виконання дій і мінімізацію навантаження на оператора.

У теоретичній частині обґрунтовано вибір підходів і засобів для реалізації frontend частини застосунку. Розглянуто особливості використання бібліотеки React для створення компонентної структури інтерфейсу, застосування Vite як інструменту збірки для забезпечення швидкої розробки, використання Redux Toolkit для централізованого керування станом застосунку, а також React-Bootstrap для побудови адаптивного та зручного інтерфейсу. Визначено основні принципи побудови

архітектури frontend сервісу, що включають модульність, розділення відповідальності між компонентами та можливість масштабування функціоналу. Передбачено використання сервісного шару з підтримкою mock-режиму, що дозволяє імітувати роботу системи без необхідності підключення серверної частини та забезпечує можливість демонстрації і тестування функціоналу.

Практична частина роботи присвячена безпосередній реалізації вебзастосунку TrackTara. Описано структуру застосунку, організацію компонентів, маршрутизацію та взаємодію між окремими модулями. Основну увагу приділено реалізації ключових сценаріїв роботи користувача, зокрема процесу пакування товарів. У цьому сценарії оператор послідовно виконує низку дій: вводить номер візка, вводить або сканує код товару чи тари, після чого система автоматично відображає інформацію про клієнта, замовлення, позицію та доступний залишок. Реалізовано можливість створення нових коробок для конкретного клієнта або додавання товарів до вже існуючих коробок, що дозволяє гнучко організовувати процес пакування. Передбачено функціонал розподілу товарів з одного візка між кількома коробками, а також можливість перекладання товарів між коробками у разі необхідності. Завершальним етапом є формування та друк пакувального листа, який містить інформацію про вміст коробок і використовується під час відвантаження.

У застосунку реалізовано низку механізмів контролю, спрямованих на зменшення кількості помилок під час виконання операцій. Зокрема, впроваджено перевірку залишків, яка не дозволяє запакувати більше товару, ніж фактично знаходиться у візку. Також передбачено систему попереджень і підтверджень дій у випадках, коли оператор намагається працювати з візком, що належить до іншої логістичної траси. Це дозволяє уникнути змішування замовлень і підвищує точність виконання операцій.

Окрему увагу приділено логістичним аспектам роботи системи. Для кожної товарної позиції використовується довідкове значення ваги одиниці, на основі якого здійснюється автоматичний розрахунок ваги рядка, вмісту тари та коробок. Такий підхід дає змогу оцінювати навантаження на упаковку та планувати подальші етапи

транспортування. Хоча розрахунок має оціночний характер, він є достатнім для підтримки прийняття рішень у процесі пакування.

З метою демонстрації функціональних можливостей розробленого рішення реалізовано mock-режим, що дозволяє працювати із застосунком без підключення до серверної частини. Це спрощує тестування, презентацію та подальший розвиток системи. Демо-версію вебзастосунку розгорнуто на платформі Vercel, що забезпечує зручний доступ до нього через браузер.

У результаті виконання роботи створено frontend сервіс, який забезпечує підтримку основних складських процесів, пов’язаних з обліком тари та її вмісту. Запропоноване рішення дозволяє оптимізувати роботу операторів, скоротити час виконання типових операцій, зменшити кількість помилок і підвищити рівень контролю за процесами пакування та відвантаження. Практична цінність роботи полягає у можливості використання розробленого застосунку як основи для впровадження в реальних умовах або подальшого розширення функціоналу.

SUMMARY

Keywords: FRONTEND, REACT, WAREHOUSE OPERATIONS, CONTAINER TRACKING, PACKING, LOGISTICS, WEB APPLICATION, AUTOMATION, UX SCENARIOS, STOCK CONTROL

Abstract:

This qualification work examines the design and development process of the TrackTara frontend service intended for tracking containers and their contents within warehouse operations. The relevance of the topic is driven by the need to improve warehouse efficiency, reduce operational errors, and ensure transparency in logistics processes. The study analyzes modern approaches to inventory and container management, identifies key challenges in packing, order processing, and shipping, including stock control complexity, operator errors, and lack of efficient tools for data handling. Special attention is given to user interface requirements aimed at ensuring usability, speed, and ease of operation.

The theoretical part substantiates the selection of tools and approaches for frontend development. It describes the use of React for building a component-based interface, Vite as a fast build tool, Redux Toolkit for centralized state management, and React-Bootstrap for creating a responsive and user-friendly interface. The architecture principles of the application are defined, including modularity, separation of concerns, and scalability. A service layer with a mock mode is implemented to simulate system behavior without a backend, enabling demonstration and testing.

The practical part focuses on the implementation of the TrackTara web application. The structure of the application, component organization, routing, and module interaction are described. Particular attention is given to the packing workflow, where the operator enters a cart number, scans or inputs item or container codes, and receives detailed information about the client, order, item, and available quantity. The system allows creating new boxes or adding items to existing ones, distributing items across multiple boxes, and transferring items between them. The process concludes with generating and printing packing slips.

Error prevention mechanisms are implemented, including stock control to prevent overpacking and warning systems when working with carts assigned to different routes. Logistics functionality includes weight estimation based on reference unit weights, allowing calculation of item lines, container contents, and box loads. Although approximate, this feature supports decision-making during packing.

A mock mode is implemented for demonstration without backend integration, and the application is deployed on Vercel for easy access. The developed frontend service supports key warehouse operations related to container and inventory tracking. The practical value lies in reducing operation time, minimizing errors, and improving control over packing and shipping processes, with potential for real-world implementation and further development.

ЗМІСТ

ВСТУП

Складські операції, пов’язані з обліком тари та рухом товарів, залишаються ділянкою з підвищеним ризиком помилок через інтенсивність ручних дій і високу швидкість обробки замовлень. Навіть незначні неточності під час пакування або відбору можуть призводити до пересорту, нестач, затримок у відвантаженні та додаткових витрат на виправлення. У практиці складів часто поєднуються кілька паралельних процесів: видача товару у візок, комплектація замовлень, пакування у коробки, контроль маршруту (траси) та формування супровідних документів. За відсутності зручних інструментів оператор змушений утримувати в пам’яті значну кількість контексту, що підвищує навантаження і створює передумови для помилок.

Окрему роль відіграє інтерфейс користувача. Сканер-орієнтований підхід, за якого основні дії виконуються через введення або сканування кодів, дозволяє пришвидшити роботу та зменшити кількість ручних операцій. Водночас такий підхід потребує чіткої логіки перевірок і миттєвого зворотного зв’язку: система має одразу повідомляти про невідповідності, показувати релевантні дані про замовлення, позиції та залишки, а також не дозволяти виконати дію, що порушує обмеження. Саме поєднання швидкості, зрозумілості та контрольованості визначає якість інструмента для складу.

У цьому контексті актуальною є розробка спеціалізованого frontend-сервісу, який забезпечує оператору чіткі сценарії роботи та мінімізує ризик помилок. Робота присвячена проєктуванню та реалізації такого сервісу — TrackTara — для відстеження тари (контейнерів) та її вмісту. Предметна область охоплює повний цикл операцій: формування замовлень, видачу товару у візок, пакування у коробки, можливість перекладання між коробками, друк пакувального листа, ведення реєстру нестач і контроль маршруту (траси). Ключовий UX-сценарій побудований навколо послідовності «візок → код товару/тари → перевірка → дія», де система відображає клієнта, замовлення, позицію та доступний залишок і не дозволяє запакувати більше, ніж фактично є у візку.

Метою роботи є створення frontend-сервісу TrackTara для відстеження тари та її вмісту на базі React, який забезпечує швидку й безпечну роботу оператора під час пакування та відвантаження, а також підтримує контроль залишків і маршруту.

Для досягнення поставленої мети визначено такі задачі дослідження:

• проаналізувати предметну область складських операцій із фокусом на обліку тари, пакуванні та відвантаженні;

• визначити ключові сценарії взаємодії оператора із системою та сформулювати вимоги до сканер-орієнтованого інтерфейсу;

• спроєктувати структуру frontend-застосунку з урахуванням модульності та можливості розширення;

• обґрунтувати вибір технологічного стеку (React, Vite, Redux Toolkit, React-Bootstrap) для реалізації інтерфейсу;

• розробити сервісний шар із mock-режимом для демонстрації роботи без повного бекенду;

• реалізувати сценарій пакування з підтримкою створення коробок, додавання товарів і розподілу між кількома коробками;

• забезпечити можливість перекладання товарів між коробками з урахуванням коректності залишків;

• впровадити механізми перевірок: контроль перевищення доступного залишку та попередження при роботі з візком іншої траси;

• реалізувати відображення даних про клієнта, замовлення, позиції та поточні залишки у відповідь на введення/сканування кодів;

• додати формування та друк пакувального листа для підсумкового оформлення відправлення;

• реалізувати облік довідкової ваги одиниці товару та підрахунок ваги рядків, тари та коробок;

• виконати тестування ключових сценаріїв та підготувати демонстраційне розгортання.

Об’єктом дослідження є процеси складського обліку тари та операції пакування і відвантаження товарів. Предметом дослідження є методи та засоби

проєктування і реалізації frontend-інтерфейсу, орієнтованого на сканерні сценарії взаємодії, а також логіка контролю коректності дій оператора під час роботи з тарою і товарами.

Методологічну основу роботи становлять аналіз предметної області та наявних підходів до організації складських процесів, проєктування структури застосунку і алгоритмів обробки дій користувача, реалізація інтерфейсу користувача з урахуванням принципів зручності та швидкодії, а також тестування сценаріїв роботи в умовах, наближених до реальних. Особлива увага приділяється проєктуванню стану застосунку та взаємодії компонентів, що дозволяє забезпечити передбачуваність поведінки системи і спростити її підтримку.

Розроблений застосунок TrackTara реалізовано як frontend-рішення з використанням React у поєднанні з Vite для швидкої збірки, Redux Toolkit для керування станом і React-Bootstrap для побудови інтерфейсу. Архітектура передбачає наявність сервісного шару з mock-режимом, який імітує відповіді системи та дозволяє демонструвати функціональність без підключення серверної частини. Це дає можливість відпрацювати користувацькі сценарії, перевірити логіку обмежень і підготувати інтерфейс до інтеграції.

Ключовий сценарій пакування побудований так, щоб мінімізувати зайві дії: оператор вводить номер візка, далі вводить або сканує код товару чи тари, після чого система одразу відображає інформацію про клієнта, замовлення, позицію та доступний залишок. На основі цих даних оператор створює коробку для конкретного клієнта або додає позиції до вже існуючої коробки. Передбачено можливість розподілу товарів з одного візка між кількома коробками, що відповідає реальним сценаріям, коли замовлення формуються частинами. У разі потреби оператор може перекладати товари між коробками, при цьому система контролює коректність залишків і не допускає перевищення доступної кількості.

Окремим елементом є система перевірок і попереджень. Під час спроби виконати дію, що суперечить правилам, користувач отримує зрозуміле повідомлення і підтвердження, якщо це допустимо. Зокрема, реалізовано попередження при роботі з візком іншої траси, що допомагає уникнути змішування потоків замовлень.

Контроль залишків запобігає ситуаціям, коли в коробку намагаються додати більше товару, ніж фактично є у візку. Такий підхід забезпечує баланс між швидкістю роботи та надійністю результату.

У межах роботи також враховано логістичні аспекти, пов’язані з оцінкою ваги. Для кожної позиції використовується довідкове значення ваги одиниці товару (weightKg), на основі якого обчислюється вага рядка, а також сумарна вага вмісту тари та коробок. Це дозволяє орієнтовно оцінювати навантаження та приймати рішення під час пакування. Хоча розрахунок є оціночним, він корисний для практичних задач і не ускладнює інтерфейс.

Для перевірки працездатності та демонстрації можливостей застосунку виконано розгортання демо-версії на платформі Vercel за адресою https://front-t5no.vercel.app/login. Для входу передбачено тестові облікові записи operator@test.com, admin@test.com, sales@test.com із паролем password123. Наявність кількох ролей дозволяє відтворити різні сценарії взаємодії з системою та перевірити її поведінку.

Практична значущість роботи полягає у створенні інструмента, який зменшує час виконання типових складських операцій і водночас знижує ймовірність помилок. Чіткі сценарії, швидка реакція інтерфейсу та вбудовані перевірки допомагають оператору зосередитися на виконанні задач без додаткового когнітивного навантаження. Запропонований підхід може бути використаний як основа для подальшого розвитку систем обліку тари та інтеграції з іншими складськими сервісами.

Скорочення / термін	Розшифрування	Пояснення
API	Application Programming Interface	Набір правил і методів, за допомогою яких окремі програмні компоненти можуть взаємодіяти між собою.
CRUD	Create, Read, Update, Delete	Чотири базові операції створення, читання, оновлення та видалення даних.
CSS	Cascading Style Sheets	Мова стилів, що використовується для опису оформлення вебсторінок.
DOM	Document Object Model	Модель, що представляє HTML-документ як дерево об'єктів, з якими можна взаємодіяти за допомогою скриптів.
HTML	HyperText Markup Language	Мова розмітки, призначена для створення структури вебсторінок.
HTTP/HTTPS	HyperText Transfer Protocol / HyperText Transfer Protocol Secure	Протокол передавання даних у вебі та його захищена версія з використанням шифрування.
ID	Identifier	Унікальне цифрове або символьне позначення, що дозволяє ідентифікувати елемент у системі.
JS	JavaScript	Мова програмування, яка широко застосовується у веброзробці для реалізації логіки клієнтської частини.
JSON	JavaScript Object Notation	Текстовий формат обміну структурованими даними між компонентами системи.
KPI	Key Performance Indicator	Ключовий показник ефективності, який використовується для оцінювання результативності процесів.
REST	Representational State Transfer	Підхід до побудови вебсервісів, заснований на використанні HTTP-методів і ресурсів (за потреби в описі взаємодії).

Скорочення / термін	Розшифрування	Пояснення
SPA	Single Page Application	Односторінковий вебзастосунок, у якому перехід між розділами виконується без повного перезавантаження сторінки.
UI	User Interface	Інтерфейс користувача, сукупність елементів, через які користувач взаємодіє із системою.
UX	User Experience	Користувацький досвід, що характеризує зручність, зрозумілість і швидкість роботи з інтерфейсом.
Vite	Vite	Інструмент для швидкого запуску й збирання фронтенд-проєктів, який використовується як середовище розробки та збірки застосунку.
WMS	Warehouse Management System	Клас інформаційних систем, призначених для управління складськими операціями та обліком ресурсів складу.
Web-застосунок	Web application	Прикладне програмне забезпечення, що працює у веббраузері та забезпечує доступ до функцій системи через мережу.
БД	База даних	Структурований набір даних, організований для зберігання та обробки інформації.
СУБД	Система управління базами даних	Програмне забезпечення, призначене для створення, зберігання, модифікації та доступу до баз даних.
Візок	—	Одиниця внутрішньоскладської комплектації, у яку тимчасово відбирають товарні позиції замовлення перед пакуванням.
Замовлення	—	Набір товарних позицій, які необхідно відібрати, перевірити, запакувати та підготувати до відвантаження.

Скорочення / термін	Розшифрування	Пояснення
Комплектація (відбір)	—	Процес підбору товарів відповідно до замовлення та формування їх у візок.
Коробка	—	Тара, що використовується для фінального пакування замовлення перед відвантаженням.
Пакування	—	Процес розміщення відібраних товарів у коробку(и) із контролем відповідності позицій і кількості.
Пакувальний лист	—	Супровідний документ, що містить перелік позицій відправлення та використовується для контролю і друку.
Пересорт	—	Помилка складської обробки, за якої відвантажується товар, що не відповідає замовленню (не той артикул/модифікація/варіант).
Реєстр нестач	—	Облік (журнал) випадків нестач і невідповідностей, які виявляються під час відбору або пакування.
Тара (контейнер)	—	Упаковка або ємність, що використовується у складських операціях та підлягає обліку.
Траса (маршрут)	—	Умовне позначення маршрутизації/правил переміщення чи контролю в межах складського процесу, що використовується для попереджень і перевірок.
weightKg	Weight in kilograms	Значення ваги в кілограмах, що застосовується в обліку/контролі під час операцій із тарою або відправленням.
Оператор	—	Користувач системи, який виконує основні складські операції (комплектацію, пакування, друк документів) згідно з правами доступу.

Скорочення / термін	Розшифрування	Пояснення
Адміністратор	–	Користувач із розширеними правами керування доступом і налаштуваннями системи.
Sales	Sales	Роль користувача (представник відділу продажу), яка працює з функціями системи відповідно до наданих прав доступу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРИЙНЯТТЯ АРХІТЕКТУРНИХ РІШЕНЬ

1.1. Опис предметного середовища та процесу діяльності

Склад як операційне середовище характеризується високою інтенсивністю виконання дій, великою кількістю однотипних операцій та необхідністю дотримання точності на кожному етапі обробки замовлень. У межах одного робочого циклу оператор взаємодіє з десятками товарних позицій, контейнерів і супровідних даних, що формує складну систему взаємозв’язків між елементами процесу. Основне завдання складу полягає не лише у фізичному переміщенні товарів, а й у забезпеченні їх коректного обліку, ідентифікації та підготовки до відвантаження відповідно до замовлень клієнтів.

У типовому сценарії обробки замовлення процес починається з його формування та передачі на склад. Далі відбувається комплектація — етап, на якому товари відбираються зі складу та розміщуються у візок. Візок у цьому контексті виступає проміжною одиницею обліку, яка акумулює всі позиції, що входять до замовлення або групи замовлень. На цьому етапі оператор повинен правильно ідентифікувати кожну позицію, врахувати її кількість і забезпечити відповідність замовленню. Уже на цьому кроці можливі помилки, пов’язані з неправильним вибором товару або кількості.

Наступним етапом є пакування, яке передбачає розподіл товарів із візка по коробках. Саме тут виникає найбільша кількість варіацій дій: один візок може

обслуговувати кілька коробок, одна коробка може містити товари з різних частин замовлення, а також можливе перекладання товарів між коробками. Така гнучкість є необхідною з практичної точки зору, але водночас значно ускладнює контроль процесу. Оператор повинен постійно відслідковувати, які товари вже запаковані, які ще залишилися у візку, і до якої коробки належить кожна позиція.

Завершальним етапом є формування пакувального листа — документа, що відображає вміст коробок і використовується при відвантаженні. Його коректність критично важлива, оскільки саме на основі цього документа здійснюється передача товару клієнту або наступному логістичному етапу. Помилки на цьому етапі можуть призвести до повернень, додаткових витрат і втрати довіри.

Рисунок 1.1. – Загальна схема складського процесу “комплектація → пакування → друк документів”

Джерело: розроблено автором із використанням штучного інтелекту

Окремим аспектом є контроль маршруту або “траси”. У складських умовах замовлення можуть бути розподілені за різними логістичними напрямками, і змішування цих потоків є небажаним. Без автоматизованої підтримки оператор змушений самостійно контролювати цей параметр, що значно підвищує ризик помилок, особливо при роботі з кількома замовленнями одночасно.

У реальних умовах складської діяльності часто використовується підхід “as-is”, який передбачає часткову або повну відсутність спеціалізованих інструментів. У такому випадку оператор покладається на паперові документи, усні

інструкції або базові електронні засоби. Це означає, що значна частина інформації обробляється вручну, без автоматичних перевірок. Наприклад, під час пакування оператор може орієнтуватися на список товарів, але не має інструменту, який би автоматично перевіряв, чи не перевищено кількість, або чи всі позиції враховані.

Ручний підхід створює кілька ключових проблем. По-перше, це відсутність актуальної інформації в реальному часі. Дані про стан замовлення можуть бути застарілими або неповними, що ускладнює прийняття рішень. По-друге, це висока залежність від уважності оператора. Людський фактор відіграє вирішальну роль, і навіть досвідчений працівник може допустити помилку при високому навантаженні. По-третє, це складність контролю і перевірки виконаних дій, оскільки відсутній централізований механізм фіксації операцій.

Типовими помилками є пересорт, коли товар потрапляє до іншого замовлення або коробки, ніж потрібно; нестача, коли кількість товару не відповідає заявленій; а також затримки, пов’язані з необхідністю повторної перевірки або виправлення помилок. Кожна з цих проблем має негативний вплив на ефективність роботи складу і може призводити до додаткових витрат.

У контексті “to-be” підходу пропонується використання спеціалізованого вебзастосунку TrackTara, який змінює логіку виконання операцій. Замість ручного контролю система бере на себе функцію перевірки і підказки, залишаючи оператору лише виконання підтверджених дій. Це дозволяє значно знизити навантаження і підвищити точність.

Ключовою особливістю є орієнтація інтерфейсу на швидку роботу, зокрема в умовах використання сканера. Оператор вводить або сканує код товару чи тари, після чого система автоматично відображає пов’язану інформацію: клієнта, замовлення, позицію та доступний залишок. Це дозволяє миттєво оцінити ситуацію і прийняти рішення без додаткових дій.

Система також забезпечує контроль залишків, що виключає можливість пакування більшої кількості товару, ніж фактично є у візку. Це одна з ключових функцій, яка безпосередньо впливає на зменшення кількості помилок. Додатково

реалізовано механізм попереджень при роботі з іншою трасою, що дозволяє уникнути змішування логістичних потоків.

Облік тари реалізовано таким чином, щоб оператор міг легко відслідковувати стан кожної коробки. Інтерфейс відображає, які товари вже додані, і дозволяє виконувати операції перекладання у разі необхідності. Це забезпечує гнучкість і адаптивність процесу.

Ще одним важливим аспектом є облік ваги. Для кожної товарної позиції використовується довідкове значення weightKg, на основі якого система розраховує вагу рядків і коробок. Це дозволяє отримати приблизну оцінку навантаження без необхідності виконання окремих вимірювань.

Важливо зазначити, що система підтримує різні ролі користувачів, зокрема operator, admin і sales. Це дозволяє розмежувати доступ до функцій і забезпечити відповідність правам користувача. Наприклад, оператор зосереджується на виконанні операцій, тоді як інші ролі можуть мати доступ до додаткових функцій.

Таким чином, перехід до використання системи TrackTara змінює характер роботи складу. Основний акцент переноситься з ручного виконання дій на взаємодію з інтерфейсом, який підтримує і контролює процес. Це дозволяє зменшити кількість помилок, пришвидшити виконання операцій і підвищити загальну ефективність.

Підсумовуючи, предметне середовище складських операцій є складним і динамічним, з високими вимогами до точності та швидкості. Основні проблеми пов’язані з людським фактором і відсутністю автоматизованого контролю. Запропонований підхід із використанням вебзастосунку дозволяє вирішити ці проблеми шляхом впровадження зрозумілого інтерфейсу, перевірок і підтримки ключових процесів. Це створює основу для подальшого вдосконалення складських операцій і підвищення їх ефективності.

1.2. Огляд наявних підходів і рішень

Організація складських процесів у сучасних логістичних системах базується на різних підходах до управління інформацією, обліку товарів та підтримки

операційної діяльності персоналу. Вибір конкретного підходу визначається розміром підприємства, інтенсивністю операцій, рівнем цифровізації та вимогами до швидкості обробки замовлень. У межах даного розділу розглядаються основні підходи, що застосовуються на практиці, з аналізом їх переваг та недоліків у контексті складських процесів, орієнтованих на пакування, комплектацію та контроль руху товарів.

Першим і найбільш базовим підходом є використання паперових носіїв інформації або простих табличних редакторів. У такій моделі вся операційна діяльність фіксується вручну: оператор працює зі списками замовлень, відмічає виконані дії, контролює кількість товарів та стан упаковок. Найчастіше використовується табличний формат, у якому кожен рядок відповідає товарній позиції, а стовпці містять кількість, статус виконання та додаткові примітки.

Основною перевагою цього підходу є його простота та відсутність необхідності у складному програмному забезпеченні. Він може бути впроваджений практично без витрат на інфраструктуру та не потребує спеціального навчання персоналу. Однак у реальних умовах складської роботи цей підхід швидко демонструє свої обмеження. Найбільш критичним недоліком є повна відсутність автоматичної перевірки даних. Усі операції залежать виключно від уважності працівника.

Додатковою проблемою є відсутність синхронізації між різними користувачами. Якщо кілька операторів працюють одночасно, виникає ризик дублювання або розсинхронізації даних. Також відсутня можливість отримання актуального стану складу в режимі реального часу. Це особливо критично у випадках, коли замовлення змінюються динамічно.

Другим підходом є використання універсальних складських систем (WMS-рішень), які забезпечують комплексну автоматизацію процесів обліку та управління складом. Такі системи зазвичай включають модулі управління запасами, обробки замовлень, контролю переміщення товарів, аналітики та формування звітності. Вони дозволяють централізовано зберігати інформацію та забезпечують контроль над усіма етапами логістичного процесу.

Головною перевагою таких систем є високий рівень структурованості даних та можливість автоматизації значної частини операцій. Вони дозволяють зменшити кількість ручних помилок, забезпечують контроль залишків і надають інструменти для аналізу ефективності роботи складу. Крім того, такі системи часто підтримують інтеграцію з іншими бізнес-процесами підприємства.

Водночас універсальні складські системи мають суттєві недоліки з точки зору операційного використання. Основною проблемою є складність інтерфейсів та надлишковість функціоналу. Оператори, які виконують фізичні дії на складі, часто змушені витрачати значний час на навігацію між модулями та пошук необхідної інформації. Це знижує швидкість виконання базових операцій.

Крім того, такі системи потребують тривалого навчання персоналу та адаптації до специфіки конкретного підприємства. У ситуаціях високої інтенсивності операцій навіть невелике ускладнення інтерфейсу може призводити до зниження продуктивності.

Третім підходом є використання індивідуально розроблених (кастомних) систем, які створюються під конкретні бізнес-процеси підприємства. Такі рішення дозволяють максимально точно врахувати специфіку роботи складу, включаючи структуру замовлень, логіку пакування та внутрішні правила обліку.

Перевагою кастомних систем є їх гнучкість та адаптивність. Вони можуть бути оптимізовані під конкретні сценарії роботи, що дозволяє підвищити ефективність операцій. Інтерфейс таких систем зазвичай розробляється з урахуванням реальних потреб користувачів, що позитивно впливає на швидкість виконання задач.

Однак недоліком є висока залежність від розробників та складність подальшого супроводу. Будь-які зміни у бізнес-процесах можуть вимагати доопрацювання системи, що створює додаткові витрати часу та ресурсів. Також існує ризик обмеженої масштабованості у разі зростання навантаження.

Окрему категорію становлять сканер-орієнтовані системи, які базуються на взаємодії користувача із системою через сканування або введення кодів товарів, тари або замовлень. У таких системах основний акцент робиться на мінімізації кількості ручних дій та швидкому отриманні інформації.

Після сканування коду система автоматично визначає відповідний об’єкт і відображає пов’язані з ним дані: замовлення, кількість товару, статус обробки та інші параметри. Такий підхід значно зменшує час пошуку інформації та знижує ймовірність помилок, пов’язаних із ручним введенням даних.

Перевагою цього підходу є висока швидкість операцій та зручність використання в умовах інтенсивної роботи. Однак ефективність таких систем напряму залежить від якості реалізації логіки перевірок і стабільності інтерфейсу. У разі недостатньої оптимізації можливі затримки або помилки у відображенні даних.

У сучасних умовах значного поширення набули вебзастосунки, побудовані за архітектурою Single Page Application (SPA). Такі системи забезпечують швидку взаємодію користувача з інтерфейсом без необхідності повного перезавантаження сторінок. Це особливо важливо для складських систем, де оператор виконує повторювані дії у безперервному режимі.

SPA-рішення дозволяють забезпечити високу швидкість відгуку інтерфейсу, динамічне оновлення даних та зручну роботу зі станом застосунку. Крім того, вони добре підходять для реалізації сканер-орієнтованих сценаріїв, оскільки дозволяють миттєво реагувати на введення користувача.

У контексті даної роботи найбільш доцільним є поєднання сканер-орієнтованого підходу із SPA-архітектурою, що дозволяє отримати баланс між простотою використання та функціональністю. Така комбінація забезпечує швидкий доступ до даних, мінімізацію кліків та зменшення кількості помилок під час виконання операцій.

Окремо слід зазначити, що жоден із розглянутих підходів у чистому вигляді не забезпечує повного задоволення вимог сучасних складських процесів. Ручні методи є надто повільними та ненадійними, універсальні системи — складними у використанні, а кастомні рішення — дорогими у підтримці. Сканер-орієнтовані та SPA-рішення є найбільш збалансованими з точки зору швидкості та зручності.

Таким чином, аналіз існуючих підходів дозволяє зробити висновок про доцільність розробки легких, швидких та орієнтованих на конкретні операційні сценарії вебзастосунків, які поєднують у собі автоматизацію перевірок, простоту

інтерфейсу та високу швидкість взаємодії. Саме на цих принципах базується підхід, реалізований у системі TrackTara.

1.3. Постановка задачі та вимоги до системи

Сучасні складські процеси характеризуються високою інтенсивністю операцій, великою кількістю однотипних дій та необхідністю точного контролю кожного етапу руху товарів. У межах даної роботи розглядається програмна система TrackTara, яка призначена для автоматизації ключових операцій складу, пов’язаних із комплектацією замовлень, пакуванням у тару, контролем переміщення товарів та формуванням супровідної документації. Висока кількість ручних операцій у традиційних підходах призводить до появи помилок, втрати часу та складності контролю актуального стану замовлень, що і формує потребу у впровадженні спеціалізованого програмного рішення.

Основною задачею системи є зменшення кількості помилок, що виникають у процесі ручного обліку, а також підвищення швидкості виконання операцій за рахунок спрощення інтерфейсу та впровадження сканер-орієнтованого сценарію взаємодії. Окрему увагу приділено мінімізації кількості кроків, які виконує оператор, оскільки складські процеси відбуваються в умовах високого навантаження і потребують максимально швидких реакцій. Система повинна забезпечувати швидкий доступ до інформації про замовлення, товари, візки та коробки, а також надавати оператору лише ті дії, які необхідні в конкретному контексті.

Архітектурно система реалізована як вебзастосунок (SPA), орієнтований на роботу в браузері без складної інсталяції. У демонстраційній версії використовується мок-логіка без повноцінної серверної інтеграції, що дозволяє відтворити основні сценарії роботи користувача та протестувати ключові бізнес-процеси на рівні фронтенду.

Функціональна структура системи включає набір взаємопов’язаних модулів, які забезпечують повний цикл складських операцій. До них належать:

- модуль авторизації користувачів;

- головний навігаційний хаб (операційна панель складу);
- модуль роботи з замовленнями та комплектації у візки;
- модуль пакувальної станції (розподіл товарів у коробки);
- модуль друку пакувального листа;
- модуль перевірок і попереджень по маршрутах (трасах);
- реєстр нестач та невідповідностей;
- модуль обліку тари та контейнерів із підтримкою вагових характеристик;
- адміністративний модуль керування користувачами та ролями.

Кожен із зазначених модулів виконує окрему частину загального процесу, але всі вони об’єднані спільною логікою обробки даних та єдиним інтерфейсним підходом, що забезпечує цілісність роботи системи.

Система підтримує рольову модель доступу, яка передбачає поділ користувачів на три основні категорії: operator, admin та sales. Такий підхід дозволяє розмежувати відповідальність та обмежити доступ до функціоналу відповідно до посадових обов’язків. Оператор виконує основні складські операції, адміністратор відповідає за налаштування системи та управління користувачами, а користувач ролі sales має доступ до перегляду інформації щодо замовлень.

Процес авторизації реалізовано через введення електронної пошти та пароля. У демонстраційній реалізації використовується мок-автентифікація без підключення до серверної частини. Після успішного входу користувач потрапляє до головного навігаційного хабу, який є центральною точкою доступу до всіх функціональних модулів системи. Така структура дозволяє швидко переходити між операційними екранами без зайвих навігаційних кроків.

Ключовий сценарій роботи системи пов’язаний із процесом комплектації та пакування замовлень. Він починається з додавання товарів у візок, який виступає проміжною логістичною одиницею. Візок акумулює товари до моменту їх подальшого розподілу у коробки. Взаємодія користувача з системою здійснюється через введення або сканування коду товару чи тари, після чого система автоматично визначає відповідний об’єкт та відображає пов’язану інформацію, включаючи замовлення, клієнта та залишок доступних одиниць.

На етапі пакування відбувається розподіл товарів із візка у коробки. Система підтримує роботу з кількома коробками одночасно, що дозволяє адаптуватися до реальних умов складської діяльності, де одне замовлення може бути розподілене на кілька одиниць тари. Також передбачена можливість переміщення товарів між коробками, що підвищує гнучкість операцій та зменшує ймовірність помилок при комплектації.

Важливою складовою є контроль залишків. Система перевіряє кількість доступного товару у візку та не дозволяє виконати операцію, якщо вона перевищує фактичний залишок. Це дозволяє уникнути ситуацій перепакування або втрати контролю над обліком. Додатково реалізовано перевірки, пов’язані з маршрутами (трасами), які запобігають змішуванню різних логістичних напрямків в одному процесі.

Після завершення пакування система забезпечує формування та друк пакувального листа. Цей документ містить структуровану інформацію про вміст коробок і використовується як контрольний елемент під час подальших етапів обробки замовлення та відвантаження.

Окремим функціональним блоком є реєстр нестач, у якому фіксуються випадки відсутності або невідповідності товарів. Такий механізм дозволяє здійснювати контроль якості складських операцій та забезпечує можливість подальшого аналізу проблемних ситуацій у процесі обробки замовлень.

Модуль обліку тари забезпечує роботу з коробками та контейнерами як з окремими логістичними одиницями. Для кожної одиниці тари може враховуватись її вміст та орієнтовна вага, що базується на довідковому параметрі weightKg. Це дозволяє оцінювати загальне навантаження та логістичні характеристики сформованих відправлень.

Функціональні вимоги до системи визначають перелік можливостей, які вона повинна забезпечувати:

* реалізація авторизації та рольового доступу;

* підтримка процесу комплектації замовлень у візки;

* забезпечення пакування товарів у коробки з можливістю розподілу;

* формування та друк пакувального листа;

* реалізація перевірок маршрутів із системою попереджень;

* ведення реєстру нестач та невідповідностей;

* облік тари та контроль вагових характеристик;

* забезпечення швидкого сканер-орієнтованого введення даних.

Нефункціональні вимоги системи визначають якісні характеристики її роботи. Інтерфейс повинен бути максимально простим, інтуїтивно зрозумілим та орієнтованим на швидке виконання операцій. Важливою вимогою є мінімізація кількості взаємодій користувача з системою для досягнення результату. Також система повинна забезпечувати високу швидкість відгуку, оскільки затримки у складських процесах безпосередньо впливають на ефективність роботи.

Додатково до нефункціональних вимог належать вимоги до стабільності роботи інтерфейсу, зрозумілості повідомлень про помилки та передбачуваності поведінки системи у різних сценаріях використання. Кожна дія користувача повинна отримувати чіткий візуальний або текстовий відгук.

Обмеження демонстраційної версії системи полягають у використанні мок-сервісів замість повноцінної серверної частини. Це означає, що всі дані моделюються на рівні фронтенду, без реального підключення до бази даних або зовнішніх API. Такий підхід дозволяє зосередитися на реалізації інтерфейсу та логіки взаємодії користувача з системою.

Таким чином, постановка задачі полягає у створенні вебзастосунку TrackTara, який забезпечує швидку, контрольовану та зручну роботу складського персоналу з основними операціями обліку, пакування та відвантаження товарів.

1.4. Теоретичні відомості

У межах даного підрозділу розглядаються базові теоретичні положення, що лежать в основі реалізації програмної системи TrackTara. Основна увага приділяється концепції побудови односторінкових вебзастосунків, особливостям використання бібліотеки React для формування інтерфейсу користувача, а також

загальним принципам організації складських процесів і вимогам до інтерфейсів, орієнтованих на інтенсивну операційну роботу.

Складські процеси в загальному вигляді включають набір операцій, пов’язаних із прийманням, зберіганням, комплектацією та відвантаженням товарів. Однією з ключових операцій є комплектація замовлень, яка передбачає відбір необхідних товарів зі складу відповідно до сформованого переліку та їх тимчасове групування для подальшого пакування. Наступним етапом є пакування, під час якого товари розподіляються по відповідній тарі з урахуванням логістичних вимог і характеристик замовлення. Ці процеси потребують високої точності та швидкості виконання, оскільки безпосередньо впливають на якість обслуговування замовників.

У сучасних інформаційних системах для підтримки подібних процесів широко застосовуються вебтехнології, зокрема односторінкові застосунки (SPA — Single Page Application). SPA передбачає роботу інтерфейсу без повного перезавантаження сторінки, що дозволяє забезпечити більш швидку та плавну взаємодію користувача із системою. Замість переходу між окремими HTML-сторінками відбувається динамічне оновлення окремих частин інтерфейсу на основі змін стану застосунку.

Бібліотека React є одним із найпоширеніших інструментів для побудови SPA-застосунків. Вона дозволяє створювати інтерфейси у вигляді компонентної структури, де кожен компонент відповідає за окрему частину інтерфейсу та може перевикористовуватися в різних частинах системи. Такий підхід спрощує розробку, підтримку та масштабування застосунку, а також забезпечує більш передбачувану поведінку інтерфейсу за рахунок роботи зі станом компонентів.

Важливою особливістю React є декларативний підхід до побудови інтерфейсу. Це означає, що розробник описує бажаний стан інтерфейсу, а бібліотека самостійно забезпечує його відображення та оновлення при зміні даних. Такий підхід є особливо зручним для систем, де інтерфейс повинен швидко реагувати на дії користувача, як це характерно для складських застосунків.

Для створення та запуску сучасних React-застосунків часто використовується інструмент збірки Vite. Він забезпечує швидке середовище розробки, оптимізує процес компіляції коду та дозволяє швидко оновлювати зміни під час розробки.

Основною перевагою Vite є висока швидкість старту проєкту та швидке оновлення модулів у процесі розробки, що особливо важливо при роботі з великими інтерфейсними системами.

У контексті складських інформаційних систем важливу роль відіграє UX-підхід, орієнтований на швидку операційну взаємодію. Інтерфейси таких систем повинні бути максимально простими, зрозумілими та мінімізувати кількість дій користувача. Це пов’язано з тим, що оператори працюють в умовах високої інтенсивності операцій і не можуть витрачати час на складну навігацію або зайві підтвердження.

Окремим аспектом є концепція сканер-орієнтованого інтерфейсу. Вона передбачає, що основним способом взаємодії користувача із системою є введення або сканування кодів товарів, тари або замовлень. Після цього система повинна швидко обробити введені дані та надати користувачу релевантну інформацію або доступні дії. Такий підхід дозволяє значно зменшити кількість ручних операцій та підвищити швидкість виконання складських задач.

UX у подібних системах базується на принципах мінімізації кліків, скорочення кількості переходів між екранами та забезпечення чіткого візуального зворотного зв’язку. Користувач повинен одразу розуміти результат своєї дії, а система — оперативно реагувати на введення інформації. Це дозволяє знизити ймовірність помилок і підвищити загальну ефективність роботи.

Таким чином, теоретична база системи TrackTara формується на перетині сучасних вебтехнологій та вимог до організації складських процесів. Використання React як основи SPA у поєднанні з інструментами збірки дозволяє реалізувати швидкий та зручний інтерфейс, а застосування UX-принципів, орієнтованих на сканерну взаємодію, забезпечує відповідність системи реальним умовам роботи складського персоналу.

Висновки до розділу 1

У межах першого розділу було розглянуто загальні положення, пов’язані з предметною областю складських процесів та розробкою вебзастосунку TrackTara. Проаналізовано особливості організації роботи складу, зокрема процеси комплектації замовлень, пакування товарів, формування супровідної документації та контроль переміщення продукції між логістичними одиницями. Встановлено, що основною проблемою традиційних підходів до управління складом є висока залежність від ручних операцій, що призводить до виникнення помилок, затримок та ускладнення контролю актуального стану даних.

Під час аналізу існуючих підходів до автоматизації складських процесів було визначено, що використання паперових носіїв та табличних редакторів є недостатньо ефективним через відсутність автоматичної перевірки даних та складність синхронізації. Універсальні складські системи забезпечують широкий функціонал, однак часто є надмірно складними для операторів, які виконують рутинні операції. Натомість спеціалізовані та сканер-орієнтовані рішення дозволяють підвищити швидкість роботи та зменшити кількість помилок.

У ході розгляду предметної області було встановлено, що ключовими вимогами до сучасних складських систем є простота інтерфейсу, швидкість виконання операцій та мінімізація кількості взаємодій користувача із системою. Особливу роль відіграє можливість швидкого отримання інформації через сканування кодів, що відповідає реальним умовам роботи складського персоналу.

Розроблювана система TrackTara позиціонується як вебзастосунок, орієнтований на підтримку основних операцій складу, включаючи роботу з візками, пакування у коробки, друк пакувальних листів та ведення реєстру нестач. Додатково враховується рольова модель доступу та необхідність контролю логістичних маршрутів.

Теоретичні основи, розглянуті у розділі, підтверджують доцільність використання підходу Single Page Application та бібліотеки React для реалізації інтерфейсу системи. Це дозволяє забезпечити швидку реакцію інтерфейсу, динамічне оновлення даних та зручну компонентну структуру застосунку.

Використання інструменту Vite сприяє оптимізації процесу розробки та прискоренню збірки проєкту.

Окрему увагу приділено принципам UX-дизайну, орієнтованим на інтенсивну роботу користувача. Встановлено, що для складських систем критично важливими є мінімізація кількості дій, чіткість візуального зворотного зв’язку та адаптація інтерфейсу до сканер-орієнтованого сценарію взаємодії.

Таким чином, результати розгляду першого розділу формують теоретичну та практичну основу для подальшого проєктування та реалізації системи TrackTara, яка спрямована на підвищення ефективності складських процесів та зменшення кількості операційних помилок.

РОЗДІЛ 2. ІНФОРМАЦІЙНА МОДЕЛЬ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ TRACKTARA

2.1. Аналіз предметної області

Предметною областю системи TrackTara є внутрішні складські процеси, пов’язані з обробкою замовлень, їх комплектацією у тимчасові логістичні одиниці (візки), подальшим пакуванням у коробки, а також контролем коректності виконання операцій. Додатково система охоплює функції обліку тари, контролю залишків товарів, ведення реєстру нестач і перевірок логістичних маршрутів (трас). Усі ці процеси мають спільну рису — високу інтенсивність операцій та значну залежність від точності введення даних оператором.

У межах системи інформаційна модель побудована навколо декількох ключових сутностей, які відображають реальні об’єкти складської діяльності. Основною сутністю є замовлення (Order), яке виступає центром усієї логіки обробки. Замовлення містить набір позицій (items[]), список візків (carts[]), а також службову інформацію, таку як клієнт, траса, сектор і статус обробки. Таким чином, Order можна розглядати як агрегуючу сутність, яка об’єднує всі етапи життєвого циклу складської операції.

Кожна позиція замовлення представлена як окремий елемент масиву items[]. Вона містить інформацію про товар, який необхідно обробити: ідентифікатор продукту, код продукту (productCode), заплановану кількість (quantity), вже відібрану кількість (pickedQuantity), одиницю виміру, вагу одиниці (weightKg), а також статус виконання. Такий підхід дозволяє відокремити планову частину замовлення від фактичного стану виконання.

Візок (cart) у межах замовлення є проміжною логістичною сутністю. Він представлений у вигляді елементів масиву carts[] всередині Order. Кожен візок містить cartNumber, набір позицій items[], а також додаткові службові поля, які пов’язують його з маршрутом або лінією обробки. Візок виконує роль тимчасового контейнера, у який здійснюється відбір товарів перед пакуванням.

Окремо в системі існує поняття рядка у візку (cart item), який відображає фактичне розміщення товару у конкретному візку. Такий елемент містить ідентифікатор позиції, код контейнера (containerCode) та кількість товару, що була поміщена у візок. Це дозволяє відстежувати фактичний рух товарів між етапами комплектації та пакування.

Сутність контейнера (Container) реалізована через MockContainerService і використовується для обліку фізичної тари на складі. Кожен контейнер має унікальний код (uniqueCode), залишок кількості (currentQuantity) та додаткові атрибути, що можуть включати розташування або сектор. Контейнери виступають як фізичні одиниці зберігання, які можуть бути пов’язані з товарами або замовленнями.

Довідник товарів реалізований через mockProducts і містить базову інформацію про продукти: ідентифікатор, назву, код продукту (productCode), опис, вагу одиниці (weightKg) та інші довідкові поля. Цей довідник є джерелом базових даних, які використовуються при формуванні замовлень та розрахунках.

Окремо в системі використовується доменна модель salesDomainStore, яка містить додаткові довідникові структури, зокрема клієнтів, маршрути (RouteMaster) та інформацію про пакувальні коробки (mockBoxes). Ці дані використовуються для зв’язку замовлень із логістичними правилами та організацією пакування.

Реєстр нестач (MockBrakiMagService) є окремою сутністю, яка містить записи про виявлені невідповідності або відсутність товарів. Кожен запис фіксує інформацію про проблему, що дозволяє здійснювати подальший контроль якості складських операцій.

Реєстр візків (MockCartRegistry) використовується для валідації номерів візків. Він містить перелік допустимих ідентифікаторів візків, які можуть бути використані у процесі комплектації.

З точки зору класифікації даних, у системі можна виділити три основні типи інформації:

Таблиця 2.1

Класифікація даних системи TrackTara

Тип даних	Приклади сутностей	Призначення
Довідникові	mockProducts, клієнти, RouteMaster	Статична інформація для операцій
Транзакційні	Order, items, carts	Операційні дані процесів
Контрольні	BrakiMag, валідація візків	Фіксація помилок та перевірок

Вхідними даними системи є дії користувача, які реалізуються через введення або сканування кодів. Основними вхідними ідентифікаторами виступають cartNumber (номер візка), productCode (код товару) та containerCode (код тари). У деяких сценаріях також використовується код коробки (boxCode), який застосовується на етапі пакування.

Таблиця 2.2

Основні вхідні дані

Елемент вводу	Опис
cartNumber	Ідентифікатор візка
productCode	Код товару
containerCode	Код тари
boxCode	Код пакувальної коробки

Вихідними даними системи є сформовані структури, що відображають результат обробки операцій. До них належать оновлені замовлення, заповнені коробки, пакувальні листи, а також записи в реєстрі нестач.

Таблиця 2.3

Основні вихідні дані

Елемент результату	Опис
оновлене замовлення	зміна pickedQuantity та статусів

коробка (PackingBox)	сформований набір товарів
пакувальний лист	друкований результат пакування
запис нестачі	фіксація відхилень

Важливою особливістю предметної області є наявність багаторівневої структури даних: замовлення → візок → коробка → товар. Така ієрархія визначає логіку переміщення товарів у системі та впливає на всі алгоритми перевірки.

Окремо слід відзначити обмеження предметної області, пов’язані з демонстраційною реалізацією системи. Зокрема, використовується мок-логіка замість реального серверного середовища, що означає відсутність синхронізації з зовнішніми ERP або WMS системами. Усі дані існують у межах фронтенд-стану, що дозволяє моделювати поведінку системи без складної інфраструктури.

Таким чином, предметна область TrackTara формує чітко структуровану модель складських процесів, у якій ключові сутності взаємопов’язані через логіку обробки замовлень та переміщення товарів. Це дозволяє перейти до етапу проєктування інформаційної моделі та формалізації зв’язків між об’єктами системи.

2.2. Проєктування системи

Проєктування системи TrackTara базується на формалізації предметної області у вигляді взаємопов’язаних сутностей та визначенні логічних зв’язків між ними. Основною метою даного етапу є створення інформаційної моделі, яка відображає реальні складські процеси та забезпечує коректну підтримку операцій комплектації, пакування та контролю.

У центрі інформаційної моделі знаходиться сутність замовлення (Order), яка об’єднує всі операції та виступає основною точкою агрегації даних. Кожне замовлення містить набір позицій (items[]) та набір візків (carts[]), що відображає розподіл процесу комплектації на окремі логістичні одиниці.

Сутність Order має зв’язок типу один-до-багатьох із сутністю Cart. Це означає, що одне замовлення може містити декілька візків, тоді як кожен конкретний візок

існує виключно в межах одного замовлення. Такий підхід відповідає реалізації в мок-сервісах, де carts[] є вкладеною структурою.

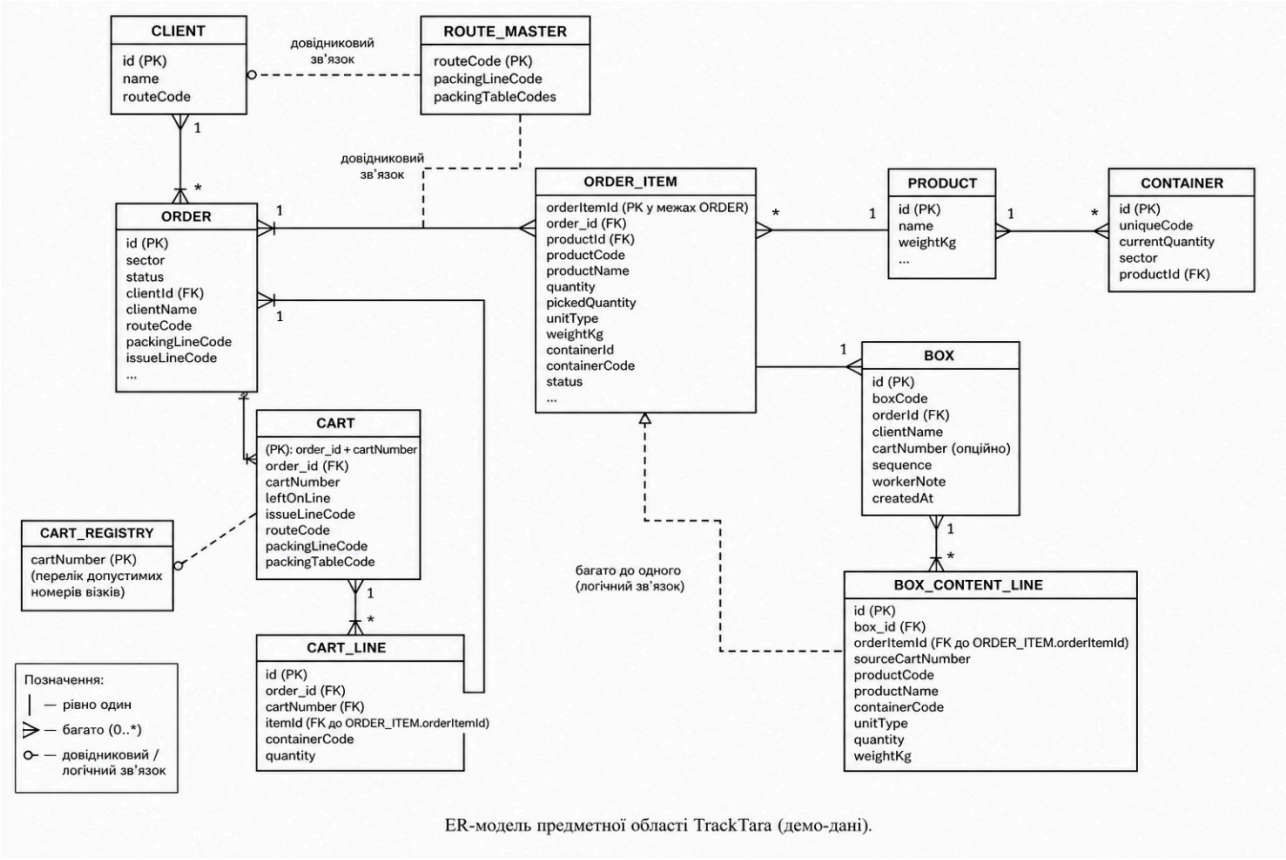
Кожен візок (Cart) у свою чергу містить набір елементів (cart.items[]), які відображають фактичні операції відбору товарів. Ці елементи доцільно розглядати як окрему сутність CartLine. Зв’язок між Cart і CartLine також є один-до-багатьох. При цьому у демонстраційній реалізації відсутні жорсткі обмеження унікальності для CartLine, що означає можливість дублювання записів для однієї позиції замовлення при повторному відборі.

Позиції замовлення (OrderItem) є логічно окремою сутністю, хоча фізично представлені як масив items[] у структурі Order. Вони містять інформацію про товар, заплановану кількість та стан виконання. Зв’язок між Order та OrderItem є один-до-багатьох.

Окремо в системі реалізована сутність пакувальної коробки (PackingBox). Кожна коробка має зв’язок із замовленням через orderId, що визначає належність коробки до конкретної операції. Одне замовлення може містити декілька коробок, що дозволяє розподіляти товари відповідно до логістичних вимог.

Вміст коробки представлений у вигляді набору елементів contents[], які доцільно виділити в окрему сутність BoxContentLine. Кожен такий елемент містить посилання на позицію замовлення (orderItemId), а також ідентифікатор візка (sourceCartNumber), з якого було взято товар. Це створює логічний зв’язок між коробкою, візком та позицією замовлення.

Таким чином, сутність BoxContentLine виступає як точка перетину декількох зв’язків: вона належить до конкретної коробки, посилається на позицію замовлення та опосередковано пов’язана з візком через його номер. Це дозволяє відстежувати походження кожної одиниці товару.



ER-модель предметної області TrackTara (демо-дані).

Рисунок 2.1. – ER-модель предметної області TrackTara.

Джерело: Розробка автора

На рисунку 2.1 подано ER-діаграму, що ілюструє логічну структуру даних системи TrackTara, включно з ключовими сутностями (Order, OrderItem, Cart, CartLine, PackingBox, BoxContentLine, Product, Container, Client, RouteMaster) та їх зв’язками з відповідними кратностями. Діаграма відображає вкладеність візків у межах замовлення, зв’язок рядків коробок із позиціями замовлення через orderItemId, а також логічний зв’язок із візком через sourceCartNumber у межах того ж замовлення.

Окремо у моделі присутні довідникові сутності, які не входять безпосередньо до транзакційних зв’язків, але використовуються для збагачення даних. До них належать Product (довідник товарів), Container (довідник тари) та RouteMaster (довідник маршрутів). Зв’язок між Product та OrderItem реалізується через productId, хоча у демонстраційній реалізації це не є жорстким зовнішнім ключем. Аналогічно, зв’язок із контейнерами реалізується через containerCode.

Маршрути (траси) у системі представлені як довідникові значення, що зберігаються у RouteMaster, тоді як у замовленні та візку використовуються лише їх коди (routeCode). Це дозволяє уникнути складної структури зв’язків, зберігаючи при цьому необхідну інформацію для перевірок.

Крім ER-моделі, для опису поведінки системи доцільно використовувати UML-діаграми. Однією з ключових є діаграма варіантів використання (Use Case), яка відображає взаємодію користувачів із системою.

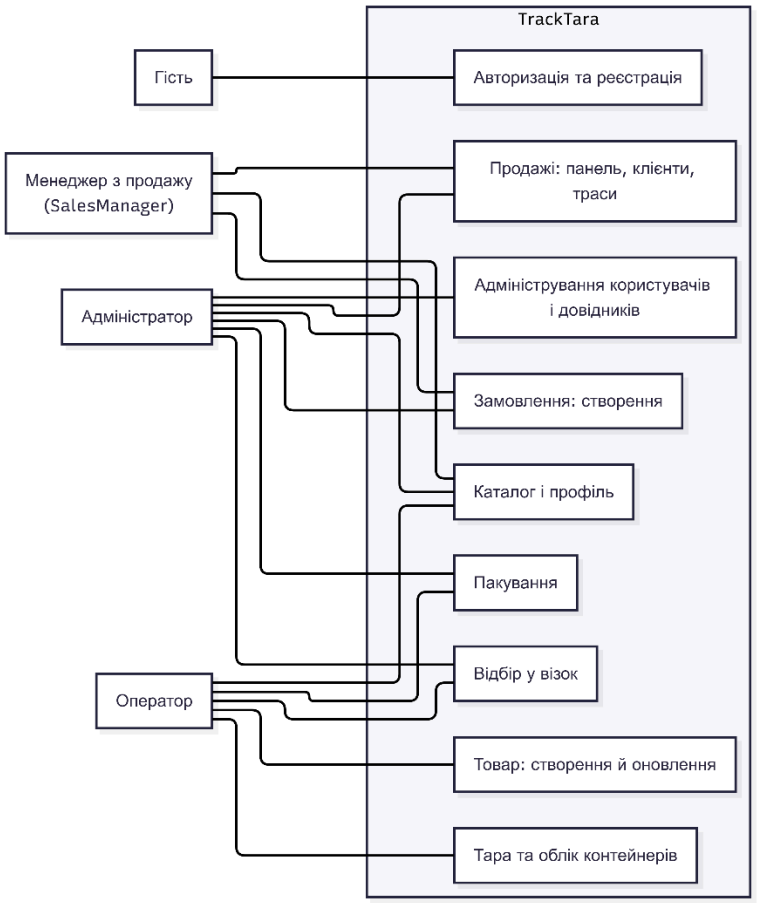


Рисунок 2.2. – UML діаграма варіантів використання для ролей operator, admin, sales

Джерело: Розробка автора

На діаграмі варіантів використання доцільно виділити три основні ролі: operator, admin та sales. Оператор виконує основні складські операції, включаючи видачу товарів у візки, пакування у коробки та фіксацію нестач. Адміністратор відповідає за управління користувачами та базові налаштування системи. Роль sales пов’язана з переглядом інформації про замовлення та клієнтів.

Кожна роль має доступ лише до визначеного набору функцій, що забезпечує розмежування прав доступу. Це особливо важливо для запобігання помилкам та несанкціонованим діям у системі.

Іншою важливою UML-діаграмою є діаграма послідовності, яка відображає виконання ключового сценарію пакування.

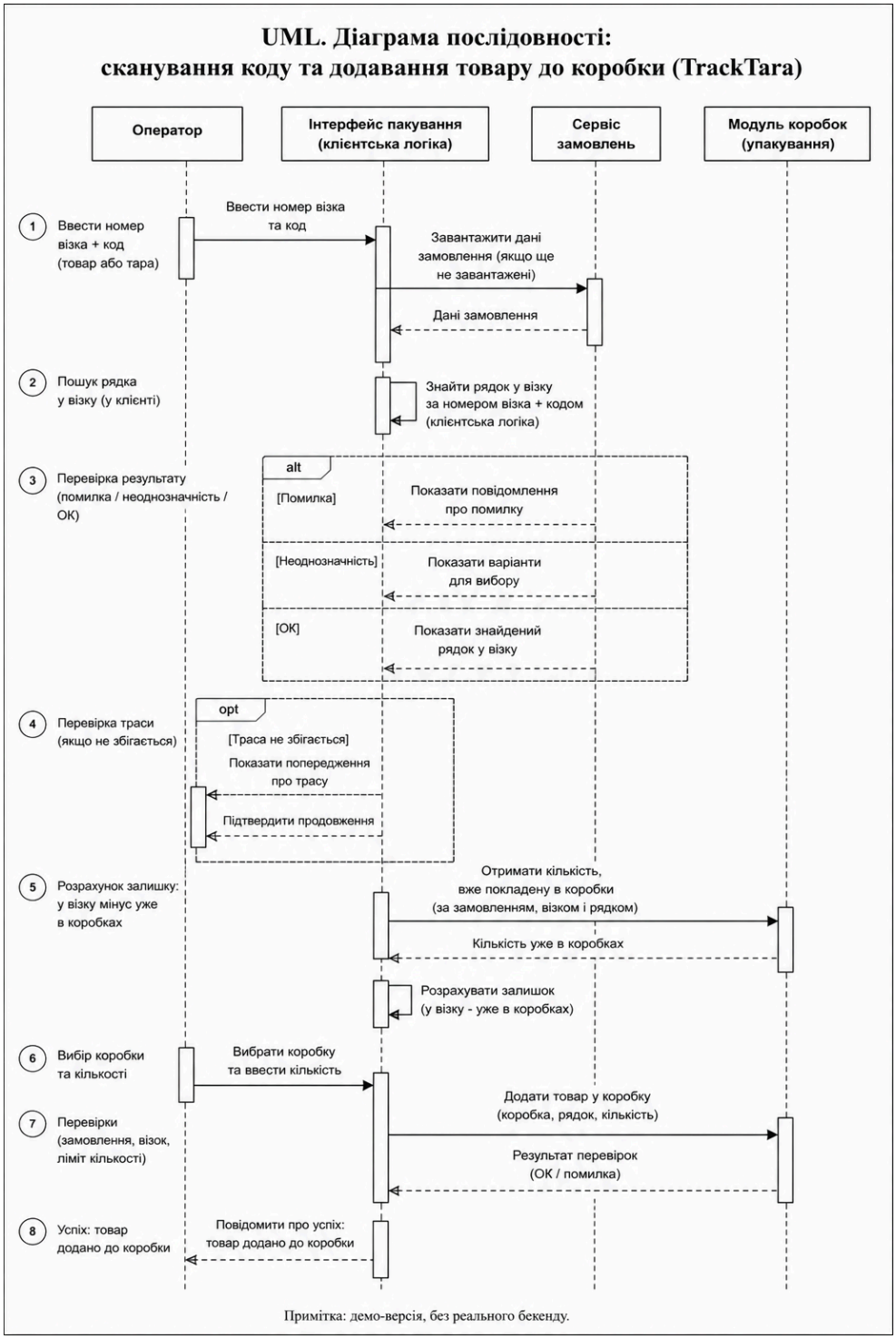


Рисунок 2.3. – UML діаграма послідовності для сценарію «сканування коду та додавання товару до коробки»

Джерело: Розробка автора

На цій діаграмі показано взаємодію між користувачем, інтерфейсом та внутрішньою логікою системи. Сценарій починається з введення номера візка та коду товару або контейнера. Система виконує пошук відповідного рядка у візку, перевіряє доступну кількість для пакування та, у разі успішної перевірки, додає товар до вибраної коробки. У випадку невідповідності умов система генерує повідомлення про помилку або запитує підтвердження.

Важливою частиною проєктування є також визначення навігаційної структури системи. TrackTara реалізована як набір взаємопов’язаних модулів, доступ до яких здійснюється через головний навігаційний екран. Основними модулями є робота з замовленнями, пакувальна станція, реєстр нестач та адміністративна частина. Така структура забезпечує логічне розділення функціоналу та спрощує роботу користувача.

Таким чином, проєктування системи TrackTara базується на чіткій інформаційній моделі, яка відображає структуру складських процесів, а також на використанні UML-діаграм для опису поведінки системи. Це створює основу для подальшої формалізації алгоритмів та реалізації функціоналу.

2.3. Математичне та алгоритмічне забезпечення

У даному підрозділі формалізовано ключові правила функціонування системи TrackTara, які забезпечують коректність операцій пакування, узгодженість даних та запобігання помилкам користувача. На відміну від класичних математичних моделей, у цьому випадку основна увага приділяється алгоритмам перевірки, інваріантам та обчисленням, що безпосередньо використовуються у процесі роботи інтерфейсу.

2.3.1. Перевірка «візок + код товару/тари»

Ключовим етапом процесу пакування є ідентифікація товарної позиції за номером візка та відсканованим кодом товару або тари. Саме від коректності цієї перевірки залежить можливість подальшого виконання операцій пакування.

У системі TrackTara пошук виконується на клієнтській стороні з використанням даних замовлень, які вже завантажені до пам'яті застосунку. Перед початком перевірки введене користувачем значення нормалізується: із нього видаляються зайві пробіли, а символи приводяться до єдиного регістру. Це дозволяє уникнути помилок, пов'язаних із різними способами введення даних.

Пошук виконується послідовно. Спочатку система знаходить усі замовлення, що містять візок із зазначеним номером. Після цього серед позицій відповідного візка здійснюється пошук рядка, код товару або код тари якого збігається з введеним значенням. Для кожного знайденого збігу додатково перевіряється відповідність позиції замовлення.

Після завершення пошуку можливі три варіанти розвитку подій:

- якщо відповідних записів не знайдено, система повідомляє користувача про відсутність потрібної позиції або про те, що візок із таким номером не зареєстрований у поточних видачах;
- якщо знайдено декілька однакових відповідностей, операція блокується, а користувач отримує повідомлення про неоднозначність результатів пошуку;
- якщо знайдено єдиний відповідний запис, система використовує його для подальшого виконання операції пакування.

Таким чином, коректний сценарій роботи передбачає наявність лише одного запису, який відповідає заданій комбінації номера візка та коду товару або тари. Такий підхід забезпечує однозначну ідентифікацію об'єкта операції, мінімізує ризик помилок під час пакування та підвищує надійність роботи системи.

2.3.2. Розрахунок залишку для пакування

Однією з ключових перевірок у системі TrackTara є визначення кількості товару, яку ще можна додати до пакувальної коробки. Така перевірка необхідна для забезпечення коректності складських операцій і запобігання ситуаціям, коли оператор помилково пакує більшу кількість товару, ніж була фактично відібрана у візок.

Для кожної позиції система зберігає інформацію про кількість товару, яка була видана у конкретний візок. Під час пакування додатково враховується кількість цього самого товару, яка вже була розподілена між коробками в межах відповідного замовлення. На основі цих даних визначається доступний залишок для подальшого пакування.

Перед виконанням операції додавання товару до коробки система здійснює послідовність перевірок. Спочатку визначається загальна кількість товару, яка вже була упакована для відповідної позиції замовлення. Далі ця кількість порівнюється з кількістю товару, що міститься у візку. Якщо весь товар уже розподілено по коробках, подальше пакування блокується.

У випадку, коли оператор намагається додати до коробки більшу кількість товару, ніж доступна для пакування, система забороняє виконання операції та відображає повідомлення про перевищення допустимого залишку. Лише якщо введена кількість не перевищує доступний залишок, товар може бути успішно доданий до вибраної коробки.

Таким чином, логіка контролю залишків ґрунтується на постійному зіставленні фактично відібраної кількості товару з кількістю, яка вже була упакована. Це дозволяє підтримувати узгодженість даних, запобігати помилкам оператора та виключати можливість повторного пакування одних і тих самих товарних позицій.

2.3.3. Перекладання між коробками

У процесі роботи із замовленнями може виникати необхідність змінити розподіл товарів між уже сформованими пакувальними коробками. Для цього в системі TrackTara реалізовано функцію перекладання товарів між коробками, яка дозволяє коригувати результати пакування без повторного виконання всього процесу.

Перед виконанням операції система перевіряє коректність вхідних даних. Насамперед контролюється, щоб обидві коробки належали до одного замовлення, оскільки перенесення товарів між різними замовленнями є недопустимим. Далі перевіряється введена кількість товару: вона повинна бути більшою за нуль і не може перевищувати кількість товару, яка фактично міститься у вихідній коробці.

Після успішного проходження перевірок система зменшує кількість товару у вихідній коробці та переносить зазначену кількість до цільової коробки. Якщо відповідна позиція товару вже існує у цільовій коробці, її кількість оновлюється. Якщо ж така позиція відсутня, система автоматично створює новий запис у складі вмісту коробки.

У випадку, коли користувач намагається перенести більшу кількість товару, ніж доступна у вихідній коробці, або обирає коробки з різних замовлень, виконання операції блокується, а користувач отримує відповідне повідомлення про помилку.

Важливою особливістю даної операції є те, що вона змінює лише розподіл товарів між коробками, не впливаючи на загальну кількість упакованих товарів у межах замовлення. Завдяки цьому користувач може виправляти помилки пакування та оптимізувати розміщення товарів без ризику втрати даних або порушення цілісності обліку.

Таким чином, механізм перекладання між коробками забезпечує гнучкість процесу пакування та дозволяє оперативно коригувати результати роботи оператора із збереженням узгодженості даних системи.

2.3.4. Розрахунок ваги

У системі TrackTara для кожного товару зберігається довідкове значення ваги одиниці продукції. Ця інформація використовується для орієнтовної оцінки маси сформованих коробок і подальшого формування пакувальних документів.

Під час пакування система автоматично визначає вагу кожної товарної позиції залежно від кількості одиниць, доданих до коробки. На основі отриманих значень обчислюється загальна вага коробки як сума ваг усіх товарів, що входять до її складу.

Результати таких розрахунків використовуються насамперед для формування пакувального листа та контролю процесу відправлення. Це дозволяє оператору отримати орієнтовну інформацію про масу сформованого відправлення без виконання додаткових ручних підрахунків.

Крім того, система може відображати приблизну вагу окремих позицій у візку, а також вагу товару, що залишилася доступною для пакування. Такі значення мають інформаційний характер і допомагають користувачеві швидше оцінювати результати виконуваних операцій.

Слід зазначити, що розрахунок ваги базується на довідкових значеннях, зазначених у характеристиках товарів. Тому отримані результати є оціночними та можуть незначно відрізнятися від фактичної ваги під час фізичного зважування.

Таким чином, механізм оцінки ваги в системі TrackTara виконує допоміжну функцію, підвищуючи зручність роботи користувачів і забезпечуючи додатковий контроль під час підготовки товарів до відправлення.

2.3.5. Обробка помилок та неоднозначностей

У системі реалізовано набір перевірок, що забезпечують стабільність роботи та коректну реакцію на некоректні дії користувача.

Основні класи помилок:

1. Помилки введення:

– відсутній номер візка або код;

– некоректна кількість (≤ 0).

2. Помилки пошуку:

– відсутній візок у замовленнях;

– відсутній рядок з відповідним кодом;

– неоднозначність (декілька збігів).

3. Помилки бізнес-логіки:

– перевищення доступної кількості;

– невідповідність траси (з підтвердженням);

– відсутність вибраної коробки;

– невідповідність замовлення при операціях з коробками.

4. Помилки операцій над коробками:

– спроба перекладання більше, ніж доступно;

– різні замовлення у коробок;

– некоректні параметри операції.

Реакція системи включає:

– блокування операції;

– відображення повідомлення користувачу;

– у деяких випадках — запит підтвердження.

Таким чином, система використовує комбінацію жорстких перевірок та контрольованих попереджень для забезпечення коректності даних та зручності роботи користувача.

2.4. Забезпечення якості (перевірки, тестування, типові збої)

Якість роботи системи TrackTara забезпечується поєднанням перевірок на рівні інтерфейсу, контролю бізнес-логіки у сервісному шарі, обробки помилок і тестування основних сценаріїв роботи.

Значна частина перевірок виконується ще на рівні інтерфейсу, до запуску бізнес-операцій. У модулі пакування багато елементів керування мають динамічні стани доступності. Наприклад, кнопки створення коробки, швидкого додавання та друку стають активними лише після успішного сканування. Поле введення кількості та кнопка додавання блокуються у випадку, коли залишок для пакування дорівнює нулю.

Додатково перевіряється наявність вибраної коробки, коректність введеної кількості та відповідність доступного залишку. Частина перевірок дублюється у сервісному шарі, де при порушенні умов генеруються виключення з поясненням причини помилки. Це дозволяє зберігати узгодженість логіки навіть у випадках паралельної роботи в кількох вкладках браузера.

Окремі перевірки реалізовано у модулі створення замовлень. Клієнт є обов’язковим полем, кількість не може перевищувати залишок контейнера, а кнопка створення замовлення блокується на час виконання операції, що запобігає випадковому повторному створенню замовлення.

У реєстрі нестач система контролює, щоб кількість повернення не перевищувала зафіксовану нестачу, а контейнер для повернення був або порожнім, або вже містив той самий продукт.

Формування пакувального листа також супроводжується перевірками. Перед друком усіх коробок система перевіряє їх наявність; якщо коробок немає, користувач отримує інформаційне повідомлення. Додатково враховується можливість блокування спливаючих вікон браузером під час друку.

Обробка помилок у застосунку реалізована комбінованим способом. Частина помилок виявляється безпосередньо в інтерфейсі через умовні перевірки, після чого користувач отримує повідомлення через react-toastify. Для асинхронних операцій використовується конструкція try/catch, яка дозволяє перехоплювати помилки сервісів і показувати зрозумілий текст помилки.

Для перевірки коректності роботи системи використовується автоматизоване тестування на базі Vitest, @testing-library/react і jsdom. У package.json передбачено скрипти test, test:watch і test:coverage.

Тести охоплюють кілька рівнів системи:

– перевірку helper-функцій і доменних правил;

– тестування сервісного шару;

– перевірку Redux-конфігурації;

– тестування рольової маршрутизації та React-компонентів.

Окремо перевіряється робота гібридного режиму mock/real API, логіка ролей, маршрутизація, інваріанти пакування та поведінка мок-сховища.

Крім автоматизованих тестів, використовується статичний аналіз коду через ESLint і ручне сценарне тестування основних процесів: видача у візок, пакування, перекладання між коробками, повернення товару з реєстру нестач та друк документів.

Під час експлуатації можливі типові помилки та збої:

– введення неіснуючого коду товару або тари;

– відсутність видачі для вибраного візка;

– неоднозначність під час пошуку позиції;

– перевищення доступного залишку;

– спроба додавання без вибору коробки;

– відсутність коробок для друку;

– блокування браузером спливаючих вікон;

– пошкоджений JWT-токен у localStorage.

Для більшості таких ситуацій система або блокує виконання дії, або показує зрозуміле повідомлення користувачу.

Таким чином, забезпечення якості у TrackTara базується на багаторівневій системі перевірок, тестуванні ключових модулів і контролі критичних бізнес-обмежень. Це дозволяє підтримувати стабільність роботи застосунку як у mock-режимі, так і при роботі з реальним API.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ ВЕБ-ЗАСТОСУНКУ TRACKTARA

3.1. Засоби розробки та обґрунтування вибору технологій

Розробка frontend-сервісу TrackTara здійснювалася із використанням сучасного стеку веб-технологій, орієнтованого на створення швидких, масштабованих та зручних у підтримці односторінкових застосунків (SPA). Вибір інструментів визначався вимогами до швидкодії інтерфейсу, простоти розробки, гнучкості архітектури та можливості швидкого розгортання демонстраційної версії без повноцінної серверної інфраструктури.

Базовою технологією реалізації інтерфейсу є бібліотека React версії 18. Використання React обумовлено його компонентною моделлю, яка дозволяє розділяти інтерфейс на незалежні, перевикористовувані частини. Такий підхід є особливо ефективним для систем із великою кількістю однотипних операційних екранів, як у випадку складських процесів. React забезпечує декларативний стиль програмування, що спрощує керування станом інтерфейсу та зменшує кількість помилок при зміні даних.

Для організації навігації в застосунку використано бібліотеку React Router (версія 7), яка забезпечує маршрутизацію між сторінками без перезавантаження браузера. Це дозволяє реалізувати логічну структуру застосунку у вигляді окремих модулів (пакування, видача у візок, тара, реєстр нестач тощо), зберігаючи при цьому швидкість взаємодії користувача з системою. Додатково маршрутизація використовується для реалізації механізму обмеження доступу до окремих сторінок залежно від ролі користувача через компонент захищених маршрутів.

Для побудови інтерфейсу користувача використано бібліотеку react-bootstrap у поєднанні з Bootstrap 5. Такий вибір дозволяє швидко створювати уніфікований і зрозумілий інтерфейс із мінімальними витратами часу на стилізацію. Компоненти Bootstrap забезпечують адаптивність інтерфейсу, що є важливим при роботі на

різних пристроях та роздільних здатностях екрана. Крім того, використання готових UI-компонентів сприяє дотриманню єдиного стилю у всіх модулях системи.

Інструментом для збірки та запуску проєкту обрано Vite (версія 6). Основною перевагою Vite є висока швидкість старту середовища розробки та миттєве оновлення змін у коді завдяки механізму гарячої заміни модулів. Це значно підвищує продуктивність розробника, особливо при роботі з великими інтерфейсними застосунками. Крім того, Vite забезпечує оптимізовану збірку проєкту для подальшого розгортання.

Управління станом застосунку реалізовано за допомогою Redux Toolkit у поєднанні з бібліотекою react-redux. Такий підхід дозволяє централізовано зберігати дані застосунку, що є критично важливим для системи, де одні й ті самі дані використовуються в різних модулях. Redux забезпечує передбачуваність змін стану та спрощує синхронізацію між компонентами.

У проєкті застосовано redux-thunk для реалізації асинхронних операцій. Це дозволяє виконувати завантаження даних через сервісний шар, не порушуючи загальної архітектури керування станом. Асинхронні дії використовуються для отримання даних про товари, замовлення та інші сутності.

Додатково використано redux-persist, який забезпечує збереження стану застосунку в локальному сховищі браузера. Це дозволяє зберігати ключові дані між перезавантаженнями сторінки та підвищує зручність роботи користувача.

Для організації взаємодії з даними використано сервісний шар, який абстрагує джерело даних від інтерфейсу. У демонстраційній версії застосунку використовується режим мок-сервісів, у якому всі дані моделюються на стороні клієнта. При цьому в проєкті передбачено можливість переходу до роботи з реальним сервером через використання HTTP-клієнта axios.

Таким чином, обраний стек технологій забезпечує баланс між простотою реалізації, швидкістю розробки та можливістю подальшого масштабування. Використання React, Redux Toolkit та Vite дозволяє створити сучасний інтерфейс із високою швидкістю роботи, тоді як застосування мок-сервісів забезпечує автономність демонстраційної версії та спрощує процес тестування.

3.2. Вимоги до технічного та програмного забезпечення

Для коректного функціонування веб-застосунку TrackTara не потребується спеціалізованого апаратного або програмного середовища. Оскільки система реалізована у вигляді односторінкового застосунку (SPA), усі основні операції виконуються у браузері користувача. Це дозволяє використовувати стандартне робоче місце без додаткового встановлення спеціалізованого програмного забезпечення.

Мінімальні вимоги до технічного забезпечення включають наявність сучасного персонального комп’ютера або ноутбука, здатного забезпечити стабільну роботу сучасного веб-браузера. Практичною орієнтирною конфігурацією є наявність не менше 4 ГБ оперативної пам’яті та процесора, достатнього для виконання офісних завдань. Для більш комфортної роботи рекомендується 8 ГБ оперативної пам’яті, що дозволяє уникнути затримок при роботі з великими обсягами даних у браузері.

Застосунок не має жорстких вимог до операційної системи та може використовуватися на будь-якій сучасній ОС, що підтримує актуальні версії веб-браузерів. У практичному використанні доцільно застосовувати операційні системи сімейства Windows (версії 10 або 11), однак це не є обмеженням, оскільки застосунок є кросплатформеним.

Ключовим програмним компонентом є веб-браузер. Рекомендується використання браузерів на базі рушія Chromium (Google Chrome, Microsoft Edge) останніх стабільних версій, оскільки вони забезпечують повну підтримку сучасних стандартів JavaScript та оптимальну продуктивність. Інтерфейс застосунку адаптований для роботи у стандартних розмірах вікна браузера, без необхідності використання специфічних налаштувань.

Для повноцінної роботи системи необхідне стабільне підключення до мережі Інтернет. Це пов’язано з тим, що демонстраційна версія застосунку розгорнута на хостингу статики, і первинне завантаження відбувається через мережу. Доступ до

застосунку здійснюється через захищене з’єднання HTTPS відповідно до стандартних налаштувань хостингу. Після завантаження основні операції в режимі демонстрації виконуються на клієнтській стороні без постійного звернення до серверу, оскільки дані моделюються за допомогою мок-сервісів.

Особливістю робочого середовища є орієнтація інтерфейсу на швидке введення даних, у тому числі із використанням сканера штрихкодів. У типовій конфігурації сканер працює в режимі HID (емуляція клавіатури) і передає зчитані значення безпосередньо у поле вводу браузера. Це не потребує додаткових драйверів або інтеграцій у застосунку. Водночас система підтримує і ручне введення даних із клавіатури, що забезпечує універсальність використання.

З точки зору програмного забезпечення користувача, не вимагається встановлення додаткових клієнтських програм. Уся функціональність доступна через браузер, включаючи роботу з замовленнями, пакування, облік тари та перегляд реєстрів. Друк документів (пакувальних листів) здійснюється за допомогою стандартних засобів браузера, що використовують системні налаштування друку операційної системи.

У контексті розробки застосунку використовуються інструменти Node.js та пакетний менеджер npm для керування залежностями та запуску середовища розробки. Для локальної розробки застосовується інструмент Vite, який забезпечує швидкий запуск проєкту та ефективну збірку. Однак ці інструменти не є необхідними для кінцевого користувача і використовуються лише на етапі розробки.

У перспективі промислового використання системи доцільно передбачити розгортання серверної частини, що включає backend-сервіс, базу даних та API для обміну даними. Це дозволить забезпечити централізоване зберігання інформації, контроль доступу та інтеграцію з іншими системами. У межах даної роботи така інфраструктура не реалізована, оскільки проєкт зосереджений на розробці клієнтського інтерфейсу та демонстрації основних бізнес-процесів.

Таким чином, вимоги до технічного та програмного забезпечення для використання TrackTara є мінімальними, що робить систему доступною для впровадження без значних витрат на інфраструктуру. Одночасно закладена

можливість розширення архітектури у напрямку повноцінного серверного рішення у майбутньому.

3.3. Опис програмної реалізації

3.3.1. Архітектура frontend та структура проєкту

Frontend-частина TrackTara реалізована як SPA-застосунок із розділенням логіки на окремі рівні: інтерфейс, маршрутизацію, керування станом і сервісний шар. Такий підхід спрощує підтримку проєкту, дозволяє повторно використовувати компоненти та зменшує залежність між окремими модулями.

Структура проєкту організована за функціональним принципом.

Основні екрани системи розміщені в папці pages. Тут знаходяться сторінки авторизації, Hub, видачі у візок, пакування, роботи з тарою, продажів, замовлень, користувачів, реєстру нестач та інших модулів.

У папці components зібрані перевикористовувані елементи інтерфейсу: Layout, Header, Footer, AppSidebar, MobileDock, NavDrawer, а також службові компоненти для авторизації, синхронізації вкладок і рольових перенаправлень.

Маршрути системи винесені в окремий модуль routes. Компонент BasicRoute містить основну структуру маршрутів, а ProtectedRoute відповідає за перевірку доступу до захищених сторінок.

Redux-конфігурація розташована в store. Тут знаходяться слайси стану, асинхронні дії, селектори та базовий store.js із підключеним redux-persist.

Сервісний шар винесено в utils/services. У цьому модулі знаходяться ServiceFactory, mock- і real-сервіси, а також спільна логіка роботи з API.

Додаткові утиліти згруповані в utils/config, utils/http, utils/helpers, hooks та інших допоміжних папках. Тут реалізовано HTTP-клієнт, конфігурацію API, допоміжні функції для ролей, ваги, статистики, а також кастомні React hooks.

Контроль доступу до сторінок виконується через ProtectedRoute. Під час переходу на маршрут система перевіряє наявність токена, декодує JWT та порівнює ролі користувача зі списком дозволених ролей.

Якщо користувач не авторизований, його перенаправляє на сторінку входу. Якщо обліковий запис ще не має робочої ролі, відкривається сторінка очікування призначення ролі.

Стан застосунку централізовано зберігається у Redux. У store винесено дані користувачів, продуктів, контейнерів, фільтрів, ролей, історії змін та налаштувань застосунку.

Частина даних із коротким життєвим циклом не зберігається в Redux окремо. Наприклад, результати сканування або тимчасові дані пакування обробляються через локальний стан сторінок і сервісний шар.

Для асинхронних операцій використовується redux-thunk. Це дозволяє виконувати завантаження та оновлення даних через сервіси без винесення HTTP-логіки у компоненти.

Для збереження стану між перезавантаженнями сторінки використовується redux-persist із localStorage. Завдяки цьому після оновлення сторінки користувач не втрачає поточну сесію, фільтри або частину робочого стану.

Окремо використовується mock-сховище _mockDb, яке відповідає за збереження демонстраційних даних і синхронізацію між вкладками браузера.

Інтерфейс системи адаптований для різних розмірів екранів. На комп’ютерах використовується бічне меню AppSidebar, а на планшетах і мобільних пристроях — нижній MobileDock та висувне меню NavDrawer.

Усі навігаційні елементи формуються з одного джерела — warehouseNav. Завдяки цьому меню автоматично змінюється залежно від ролі користувача.

Для адаптивності таблиць, форм і модальних вікон використовуються компоненти Bootstrap, класи table-responsive та додаткові CSS-правила для невеликих екранів. Це дозволяє зберігати коректне відображення інтерфейсу як на десктопних, так і на мобільних пристроях.

3.3.2. Сервісний шар (mock/real), організація даних

Сервісний шар надає уніфікований інтерфейс доступу до даних та інкапсулює бізнес-операції. Ключовим елементом є ServiceFactory, який на основі конфігурації API_CONFIG.USE_MOCK_API (формується зі змінної середовища VITE_USE_MOCK_API) повертає mock- або real-реалізації сервісів. Це дозволяє переключати джерело даних в одному місці, не змінюючи коду компонентів, екшенів і Redux-логіки.

У демонстраційному режимі (USE_MOCK_API=true) застосунок використовує mock-шар. Дані користувачів, продуктів, контейнерів, замовлень, коробок, клієнтів, трас, реєстру нестач та інших сутностей зберігаються у локальному mock-сховищі.

Основою такого підходу є модуль _mockDb.js, який реалізує спрощену модель роботи з даними у браузері. Кожна «таблиця» створюється через функцію defineTable, що формує Proxy-об’єкт і автоматично синхронізує зміни з localStorage. Усі дані зберігаються під єдиним ключем tt:mockdb:v1, завдяки чому стан системи зберігається навіть після перезавантаження сторінки.

Додатково реалізовано синхронізацію між вкладками браузера через подію window.storage. Якщо в одній вкладці створити замовлення, коробку або змінити дані контейнера, інші відкриті вкладки автоматично отримають оновлений стан без перезавантаження сторінки.

Для безпечної роботи зі станом використовуються допоміжні функції clone, replaceArray та subscribeMockDb. Вони дозволяють уникати небажаних мутацій Redux-стану, атомарно оновлювати колекції та підписуватись на зміни mock-сховища.

Окрему роль у структурі застосунку виконує доменне сховище salesDomainStore. Воно містить логіку, пов’язану з пакуванням і продажним контекстом: довідники клієнтів і трас, коробки, створення та оновлення пакувальних коробок, перенесення товарів між коробками, а також підрахунок уже упакованої кількості товару.

Саме через salesDomainStore реалізується контроль залишку для пакування за формулою:

remaining = cartLine.quantity − packedQuantity.

Тобто система враховує не лише кількість товару у візку, а й те, скільки вже було додано в коробки для відповідного orderId, sourceCartNumber та orderItemId.

У проєкті також присутні real-сервіси для роботи з REST API через axios. Для більшості сутностей ServiceFactory обирає між Mock*- та real-реалізацією залежно від значення USE_MOCK_API.

Для окремих операційних модулів застосовано гібридний підхід через operationalApiShared.js. У такому режимі сервіс спочатку виконує HTTP-запит до backend, а якщо відповідний ендпоінт відсутній або ще не реалізований (наприклад 404, 405 або 501), автоматично переходить на локальний mock-режим.

Такий підхід дозволяє розробляти frontend незалежно від готовності серверної частини та поступово інтегрувати реальний API без переписування інтерфейсу.

Організація даних у mock-режимі імітує структуру реальної бази даних. Замовлення містять вкладені carts та items, коробки — contents, а операції над даними виконуються через сервіси з перевірками бізнес-правил: контроль залишків, перевірка ролей, обмеження кількості, перевірка належності до одного замовлення та інші інваріанти.

Завдяки цьому демонстраційна версія TrackTara може працювати без серверної частини, зберігаючи при цьому єдину структуру сервісів і логіку, сумісну з майбутнім підключенням реального backend.

3.3.3. Ролі користувачів і доступи

У системі реалізовано рольову модель доступу з чотирма основними ролями: Operator, Administrator, SalesManager та Guest. Перші три ролі є робочими та відкривають доступ до функціональних модулів системи. Роль Guest використовується для нових або ще не налаштованих облікових записів. Такий

користувач може увійти в систему, але не має доступу до складських, продажних чи адміністративних розділів до моменту призначення робочої ролі адміністратором.

Контроль доступу реалізовано на рівні маршрутизації за допомогою компонента ProtectedRoute. Під час відкриття захищеного маршруту система перевіряє наявність access token, декодує JWT, визначає ролі користувача та порівнює їх із переліком дозволених ролей для конкретної сторінки.

Якщо токен відсутній або некоректний, користувач автоматично перенаправляється на сторінку входу (/login). Якщо користувач має лише роль Guest, система відкриває сторінку очікування призначення ролі (/pending-role). У випадку, коли роль не відповідає маршруту, замість вмісту сторінки відображається повідомлення про відсутність доступу.

Роль Operator призначена для виконання складських операцій. Користувач із цією роллю має доступ до видачі товарів у візок (/warehouse/pick), пакування (/packing), перегляду каталогу товарів, а також до роботи з тарою та контейнерами. Для оператора доступні сторінки створення, редагування та перегляду контейнерів (/tare, /tare/create, /tare/update/:id, /tare/detail/:containerId). Також оператор може створювати й оновлювати товари, оскільки ці операції безпосередньо пов’язані з обліком складу.

Роль SalesManager орієнтована на продажний контекст системи. Користувач має доступ до панелі продажів (/sales), довідника клієнтів і трас (/sales/clients), створення замовлень (/orders/create) та перегляду каталогу товарів. Після успішного входу користувач із роллю SalesManager автоматично перенаправляється до розділу продажів.

Administrator має розширені права доступу та виконує функції адміністрування системи. Для цієї ролі доступні сторінки керування користувачами (/users), довідники типів продуктів і контейнерів (/productType, /container/containerTypes), реєстр номерів візків (/carts/registry), а також реєстр нестач (/brakimag).

У логіці перевірки доступу адміністратор має повний доступ до захищених маршрутів системи, тому може працювати як із адміністративними, так і зі складськими чи продажними модулями.

У попередніх версіях застосунку існувала роль User, однак вона не мала окремого функціонального призначення і могла дублювати логіку Guest. В актуальній реалізації роль User вилучена з основного переліку ролей. Для зворотної сумісності старі значення user під час нормалізації ролей автоматично перетворюються на Guest.

Декодування JWT використовується не лише для перевірки доступу, а й для формування профілю користувача та службових полів аудиту в окремих операціях. Такий підхід забезпечує узгодженість доступів, правильне формування навігації та контроль дій користувачів у системі.

3.3.4. Модулі системи

У системі TrackTara реалізовано кілька основних функціональних модулів, кожен із яких відповідає за окремий етап складського процесу.

Модуль обліку тари забезпечує повний цикл роботи з контейнерами: перегляд списку (/tare), створення, редагування та перегляд деталей контейнера. Контейнер пов’язується з товаром через productId, а у формах відображаються основні параметри: код, тип, сектор, ряд, кількість, одиниця виміру та додаткові примітки. На сторінці деталей користувач може додавати або вилучати товар із контейнера, переглядати історію операцій та видаляти контейнер після підтвердження дії. Для адміністратора окремо доступне керування типами контейнерів і секторами складу.

Модуль товарів реалізований через сторінку /products, сторінку деталей товару та форми створення і редагування. Для кожного товару зберігається історія змін і пов’язаних операцій. Окремо адміністратор може керувати довідником типів товарів.

Модуль видачі у візок (/warehouse/pick) використовується для комплектації замовлень. Під час видачі система формує або оновлює запис візка та додає позиції у cart.items[]. У цьому ж процесі оператор може зафіксувати нестачу товару через addToBrakiMag. Додатково виконується перевірка номера візка та автоматичне визначення траси й лінії пакування.

Модуль пакування (/packing) є одним із ключових у системі. Основний сценарій включає введення номера візка та коду товару або тари, пошук відповідної позиції у візку та подальше додавання товару в коробку. Для швидкого пакування використовується окремий блок із вибором коробки та кількості, а альтернативно підтримується модальне додавання через таблицю коробок. Система дозволяє перекладати товари між коробками одного замовлення, контролює залишки та перевіряє відповідність траси. Для пришвидшення роботи використовуються disabled-стани кнопок і автоматичний вибір першої доступної коробки після сканування. Окремо реалізовано друк пакувального листа засобами браузера.

Модуль створення замовлень (/orders/create) дозволяє менеджеру або адміністратору формувати нові замовлення. Позиції додаються з контейнерів різних секторів, після чого система автоматично групує їх та створює окремі замовлення для кожного сектора. Для пошуку використовується фільтрація за назвою товару, кодом контейнера, сектором і кодом продукту. Також передбачено захист від повторного створення замовлення.

Модуль продажів включає панель продажів (/sales) та довідник клієнтів і трас (/sales/clients). Дані клієнтів, трас і коробок зберігаються у доменному сховищі salesDomainStore, яке також використовується в модулях пакування та створення замовлень.

Модуль реєстру нестач (/brakimag) призначений для перегляду та обробки випадків нестач. Адміністратор може виконувати пошук записів, переглядати інформацію про нестачі та повертати знайдений товар назад на склад із можливістю повного або часткового повернення.

Модуль реєстру візків (/carts/registry) дозволяє адміністратору вести список допустимих номерів візків. Інтерфейс підтримує додавання, перегляд і видалення записів.

Модуль адміністрування включає сторінку користувачів (/users), адміністративну панель (/admin/dashboard), сторінку профілю (/profile) та сторінку очікування ролі (/pending-role) для користувачів без призначеної робочої ролі.

3.3.5. Друк пакувального листа

Друк реалізовано засобами браузера: для коробки або набору коробок формується HTML-представлення, яке відкривається у новому вікні і передається на друк. Перед друком перевіряється наявність коробок; за їх відсутності відображається інформаційне повідомлення і вікно друку не відкривається. Для окремої коробки контекст завжди наявний; у випадку порожнього вмісту у документі відображається відповідний текст. Обробляється також ситуація блокування pop-up вікон.

3.3.6. Реалізація ваги та підсумків

У системі використовується довідкова вага одиниці товару (weightKg). Вага рядка визначається як добуток кількості на вагу одиниці, а вага коробки — як сума ваг усіх рядків. Обчислення виконуються на клієнті та застосовуються для відображення оцінок і формування пакувального листа. Окремого агрегованого показника «вага візка» не зберігається; у прев'ю можуть відображатися оцінки по рядках.

У підсумку програмна реалізація TrackTara забезпечує повний цикл складських операцій у межах клієнтського застосунку, поєднуючи модульну архітектуру, сервісний шар із можливістю перемикання джерел даних та централізоване керування станом.

3.4. Керівництво користувача

У підрозділі наведено покрокові інструкції роботи з веб-застосунком TrackTara для основних ролей користувачів. Опис орієнтований на типові сценарії використання системи у складських процесах.

3.4.1. Вхід у систему

1. Відкрити веб-застосунок у браузері.
2. На сторінці входу ввести email та пароль.
3. Натиснути кнопку входу.
4. Після успішної авторизації виконується перехід на індексну сторінку:

– якщо користувач має лише роль SalesManager — відбувається автоматичний перехід у модуль продажів (/sales);

– для інших ролей відкривається головна сторінка складу (WarehouseHub).

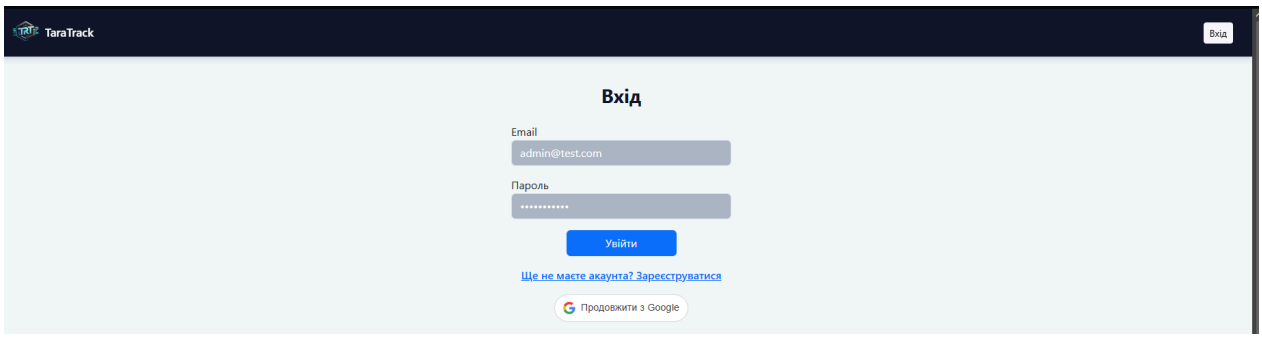


Рисунок 3.1. — Форма входу в систему

Джерело: Розробка автора

3.4.2. Навігація по системі (Hub)

Після входу користувач потрапляє до центрального навігаційного екрану системи — Hub. Для ролей Operator та Administrator відображається WarehouseHub, який виконує функцію швидкого доступу до основних модулів: видача у візок, пакування, тара, замовлення та інші розділи.

Hub не є аналітичною панеллю, а слугує точкою входу до операційних екранів. Інтерфейс побудований таким чином, щоб користувач міг швидко перейти до необхідної дії без зайвих переходів. Для ролі SalesManager, яка не виконує складських операцій, система автоматично перенаправляє користувача до модуля продажів.

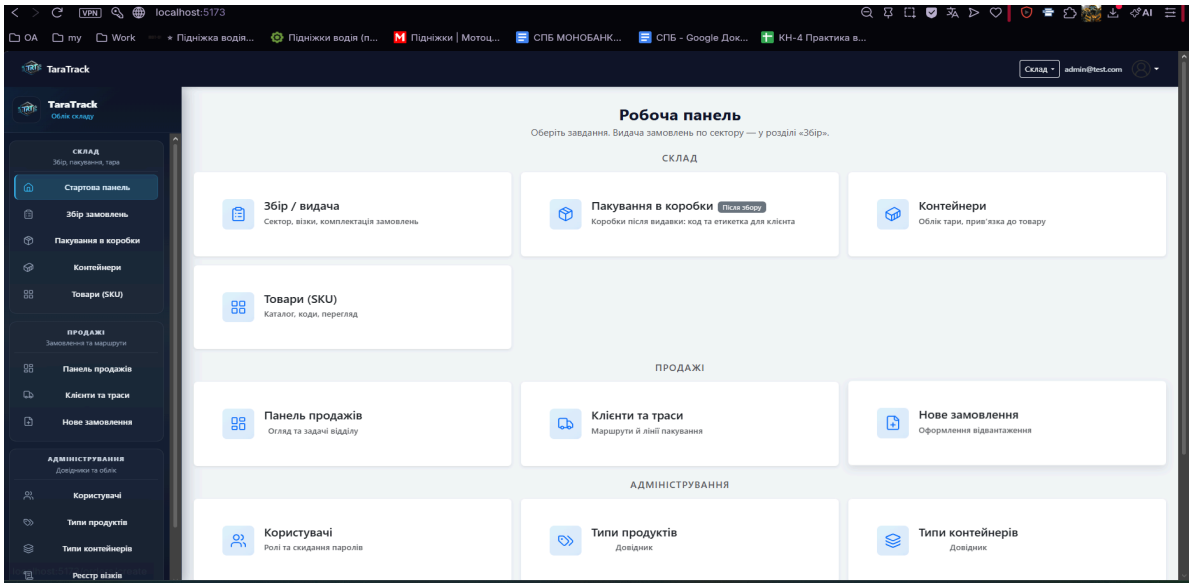


Рисунок 3.2. — Головна сторінка (Hub)

Джерело: Розробка автора

3.4.3. Робота з тарою (контейнерами)

Система підтримує повний цикл обліку контейнерів.

Перегляд списку контейнерів:

1. Перейти у розділ «Тара» (/tare).
2. Ознайомитися зі списком контейнерів у таблиці.
3. Використати дії у таблиці для переходу до редагування або перегляду.

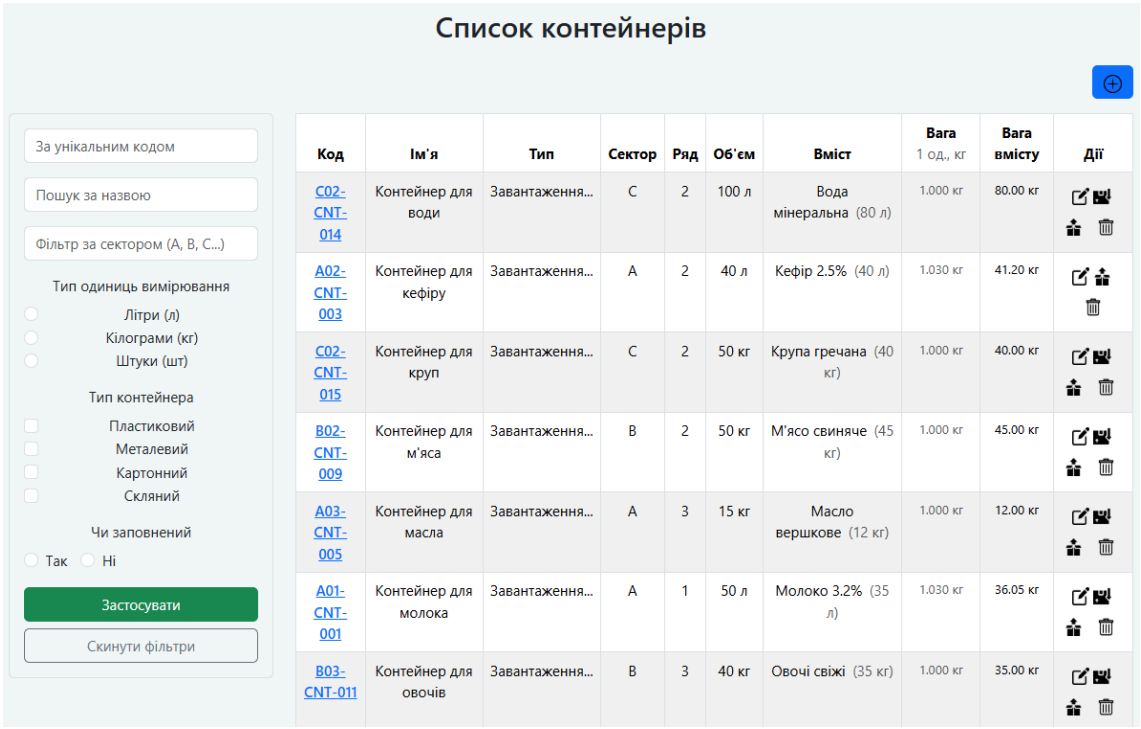


Рисунок 3.3. — Список контейнерів

Джерело: Розробка автора

Створення контейнера:

1. Натиснути «Створити контейнер».
2. Заповнити форму з основними параметрами.
3. Зберегти зміни.

Редагування контейнера:

1. Обрати контейнер у списку або перейти на сторінку деталей.
2. Внести зміни у поля.
3. Підтвердити збереження.

Робота зі сторінкою деталей:

1. Відкрити сторінку контейнера (/tare/detail/:containerId).
2. За потреби змінити метадані.
3. Виконати додавання або зняття товару з тари.
4. За потреби видалити контейнер (після підтвердження дії).

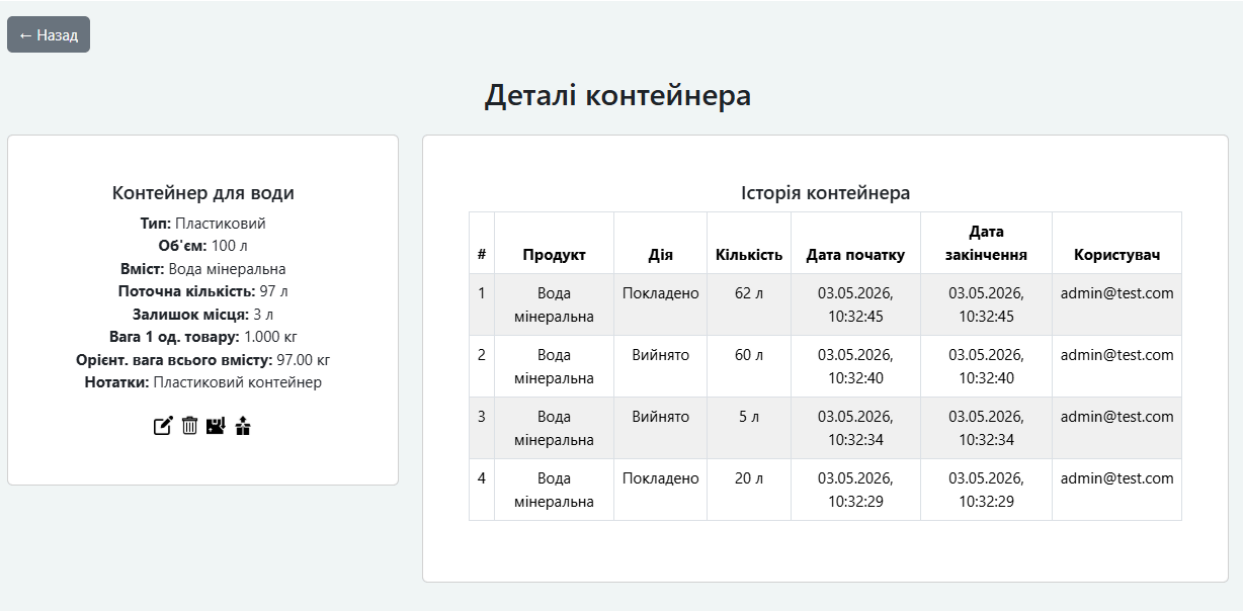


Рисунок 3.4. — Сторінка деталей контейнера

Джерело: Розробка автора

3.4.4. Створення замовлення

1. Перейти до розділу створення замовлення (/orders/create).
2. Обрати клієнта зі списку. Поле є обов’язковим, оскільки саме клієнт визначає трасу та лінію пакування.
3. За потреби скоригувати лінію пакування або видачі. За замовчуванням ці значення підставляються автоматично з траси клієнта, однак адміністратор може змінити їх вручну.
4. Знайти необхідні позиції у списку «Доступні для замовлення». Для пошуку можна використовувати назву товару, код контейнера, сектор або код товару. Пошук виконується без урахування регістру. Позиції з різних секторів відображаються у спільному списку та сортуються за сектором і номером ряду.
5. Для додавання позиції натиснути кнопку «Додати». У модальному вікні вказати необхідну кількість та підтвердити дію. За замовчуванням система підставляє доступний залишок контейнера. Після підтвердження позиція

додається до таблиці «Продукти в замовленні», де відображаються сектор, рядок, кількість та орієнтовна вага.

6. За потреби кількість можна змінити або видалити позицію зі списку до моменту створення замовлення.
7. Після формування списку натиснути кнопку «Створити замовлення». Якщо вибрані товари належать до різних секторів, система автоматично згрупує їх та створить окреме замовлення для кожного сектора. Після завершення операції відображається повідомлення з переліком створених замовлень.
8. Під час виконання запиту кнопка створення блокується. Це запобігає випадковому повторному створенню замовлення подвійним натисканням кнопки або клавіші Enter.

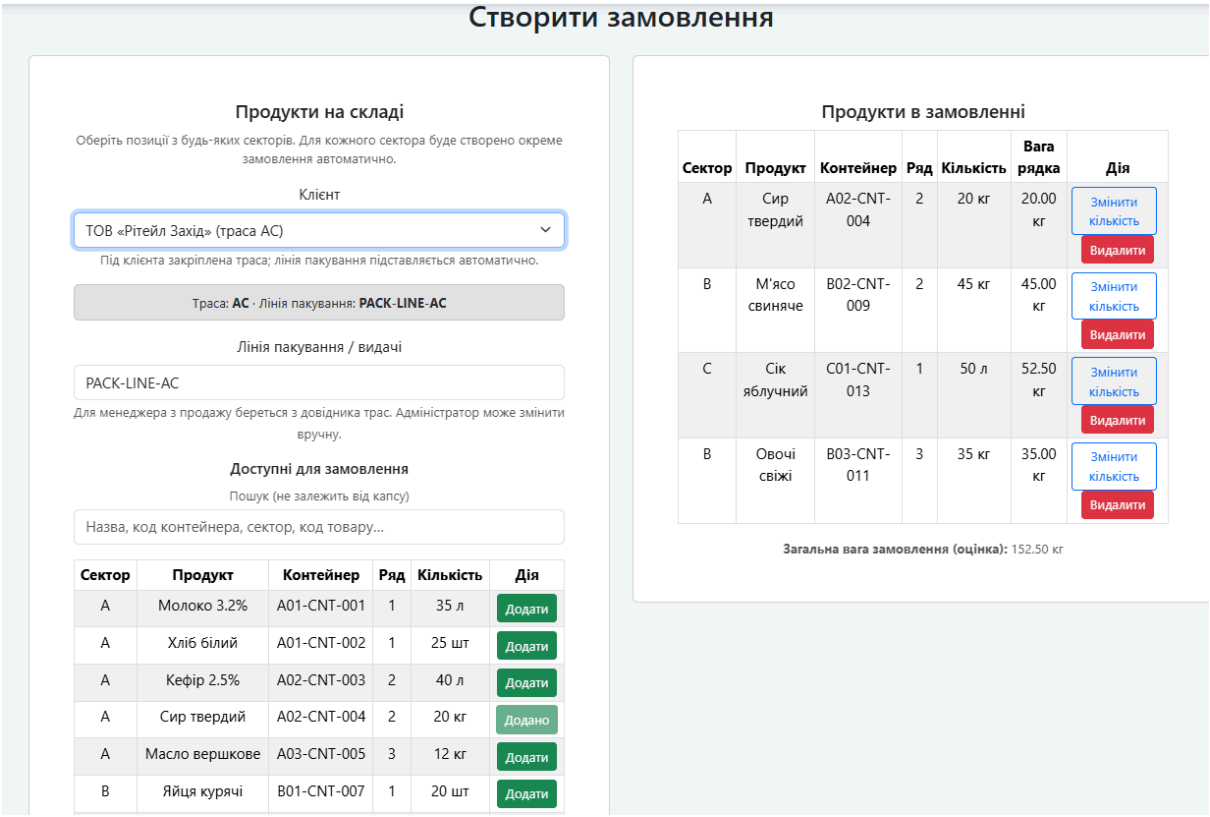


Рисунок 3.5. — Форма створення замовлення

Джерело: Розробка автора

3.4.5. Видача у візок

Операція виконується через модальне вікно, що має покрокову структуру.

1. Відкрити модальне вікно видачі.
2. Ввести номер візка.
3. Ввести або відсканувати код товару.
4. Ввести код контейнера.
5. Вказати кількість.
6. У випадку повного відбору:
   – ввести код лінії видачі;
   – натиснути «Завершити».
7. У випадку часткового відбору:
   – натиснути «Продовжити», після чого операція виконується без додаткового підтвердження.

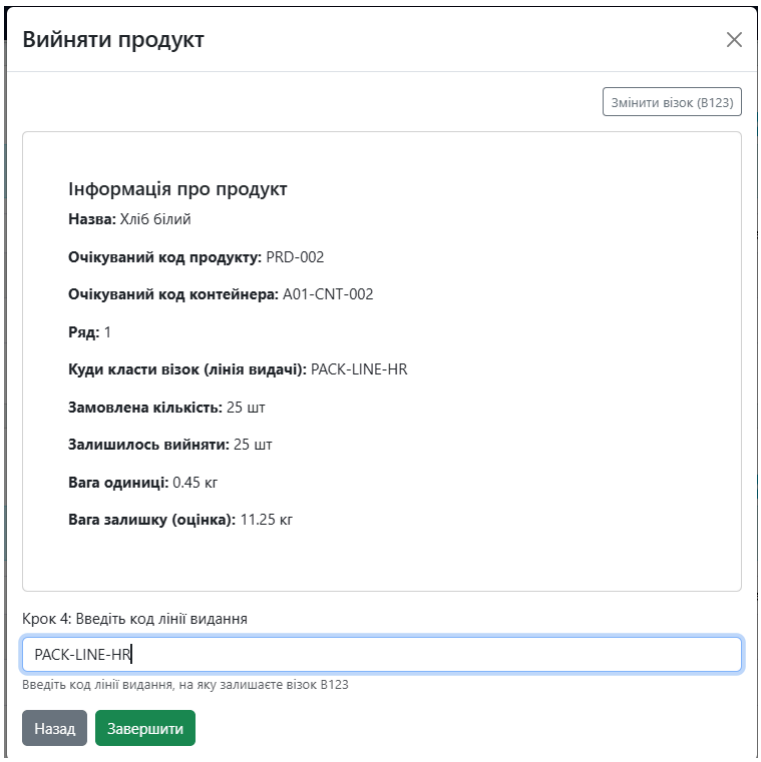


Рисунок 3.6. — Видача у візок

Джерело: Розробка автора

Додавання до реєстру нестач:

1. У процесі видачі натиснути «Додати в реєстр нестач».

2. Вказати кількість нестачі.

3. Обрати або ввести причину.

4. Підтвердити додавання.

3.4.6. Пакування: коробки, перекладання, друк

Основний сценарій пакування:

1. Ввести номер візка.

2. Ввести або відсканувати код товару або тари.

3. Натиснути «Перевірити».

4. Ознайомитися з результатом перевірки та залишком.

Швидке додавання у коробку:

– використовується після успішного сканування;

– у блоці «Коробка» автоматично підставляється перша доступна коробка (за наявності);

– кількість за замовчуванням дорівнює залишку;

– натиснути «Додати в коробку» для виконання операції.

Модальне додавання:

– використовується через таблицю коробок;

– відкривається модальне вікно додавання;

– користувач вручну обирає параметри;

– підтвердження викликає ту саму операцію додавання.

Перекладання між коробками:

1. Відкрити вміст коробки.

2. Натиснути «Перекласти».

3. Обрати цільову коробку.

4. Вказати кількість.

5. Підтвердити операцію.

Друк пакувального листа:

1. Обрати коробку або замовлення.
2. Натиснути «Друк».
3. Підтвердити дію у стандартному діалозі браузера.

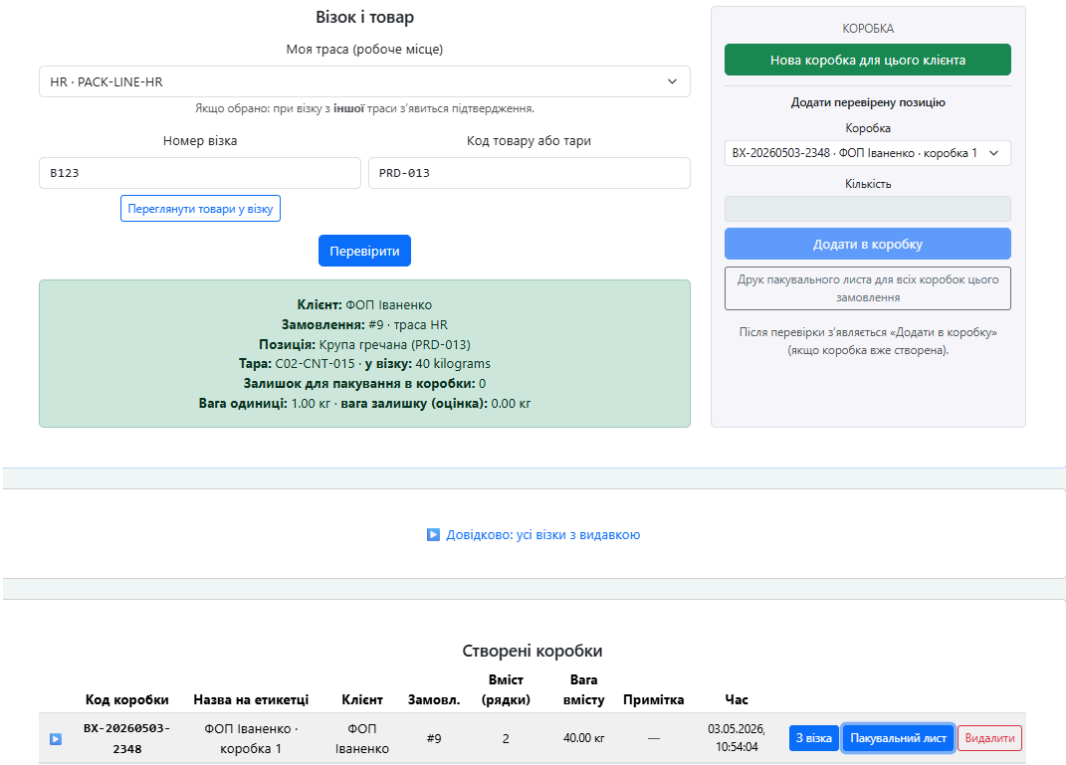


Рисунок 3.7. — Екран пакування

Джерело: Розробка автора

3.4.7. Реєстр нестач

Реєстр нестач призначений для фіксації випадків, коли під час видачі товару у візок оператор виявляє відсутність необхідної кількості товару або іншу невідповідність. Запис створюється безпосередньо в процесі видачі та містить назву і код продукту, код контейнера, кількість нестачі, одиницю виміру, орієнтовну вагу, причину, номер замовлення, користувача, який додав запис, а також дату створення.

Перегляд реєстру доступний адміністратору через окремий розділ системи (/brakimag). На сторінці відображається таблиця всіх зафіксованих випадків нестач. Для пошуку потрібного запису передбачено поле фільтрації без урахування регістру.

Пошук може виконуватися за назвою товару, кодом продукту, кодом контейнера, причиною, користувачем або номером замовлення.

Якщо товар, внесений до реєстру нестач, пізніше знайдено, адміністратор може повернути його на склад без ручного редагування контейнерів. Для цього використовується кнопка «На склад» у відповідному рядку таблиці. Після натискання відкривається модальне вікно, у якому потрібно вказати кількість товару та контейнер, до якого він буде повернений.

Система дозволяє повернути як повну кількість нестачі, так і лише її частину. Для вибору доступні порожні контейнери або контейнери, у яких уже знаходиться той самий продукт. Після підтвердження операції система оновлює залишок у контейнері та змінює дані реєстру нестач.

Якщо повернуто всю кількість товару, запис автоматично видаляється з реєстру. У випадку часткового повернення в реєстрі залишається оновлений залишок нестачі. Також адміністратор може вручну видалити запис через кнопку «Видалити», однак така дія впливає лише на сам запис у реєстрі та не змінює фактичний стан складських залишків.

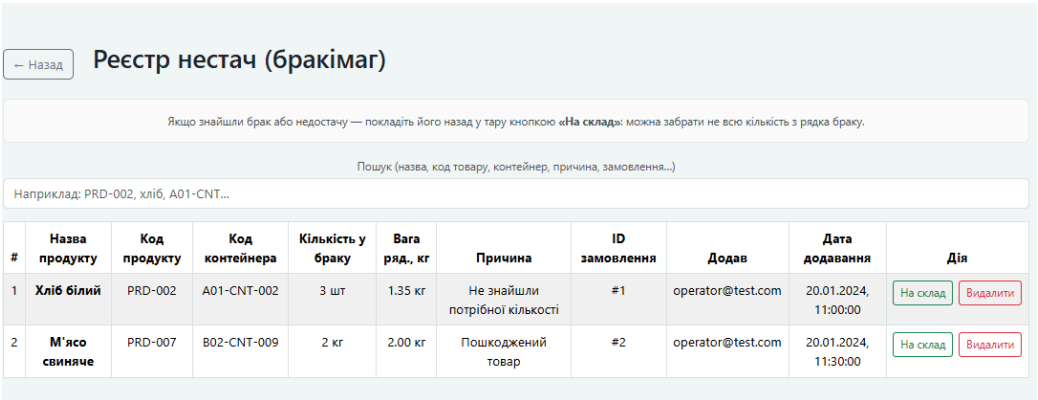


Рисунок 3.8. — Реєстр нестач

Джерело: Розробка автора

3.4.8. Типові помилки та повідомлення системи

Система використовує повідомлення для інформування користувача про помилки та обмеження:

– «Вкажіть номер візка та відскануйте або введіть код товару чи тари.»

– «Знайдено кілька збігів для цих даних (неоднозначність). Зверніться до адміністратора.»

– «У цьому візку немає позиції з таким кодом товару чи тари.»

– «У системі немає видачі товарів у візок з таким номером.»

– «Візок з таким номером не знайдено серед видач на візок.»

– «Оберіть коробку або спочатку створіть нову.»

– «Кількість не може перевищувати залишок (N).»

– «Перевірку скасовано: невідповідність траси.»

Повідомлення відображаються у вигляді спливаючих повідомлень або діалогів підтвердження та блокують некоректні операції.

Таким чином, керівництво користувача охоплює всі основні сценарії роботи із системою та дозволяє ефективно використовувати функціональні можливості TrackTara.

3.5. Розгортання та супровід

У цьому підрозділі описано процес локального запуску, збірки та демонстраційного розгортання застосунку TrackTara.

Для роботи з проєктом необхідно встановити Node.js версії 18 або новішої та пакетний менеджер npm. Перед першим запуском потрібно виконати команду npm install для встановлення всіх залежностей.

У проєкті використовуються такі основні команди:

– npm run dev — запуск локального сервера Vite у режимі розробки;

– npm run build — створення продакшен-збірки;

– npm run preview — локальний перегляд готової збірки;

– npm run lint — перевірка коду через ESLint;

– npm test — запуск тестів через Vitest;

– npm run test:watch та npm run test:coverage — запуск тестів у режимі спостереження та формування звіту про покриття.

Після виконання команди npm run build система створює каталог dist/, у якому містяться готові статичні файли застосунку.

Конфігурація середовища здійснюється через файл .env. Основним параметром є VITE_USE_MOCK_API, який визначає режим роботи застосунку:

– true — робота у mock-режимі без реального сервера;

– false — робота з REST API.

Для інтеграції з backend використовуються змінні середовища з префіксом VITE_API_*, які задають адреси сервісів.

Демонстраційне розгортання застосунку виконується на платформі Vercel. Після збірки застосунок розміщується як набір статичних файлів і працює через HTTPS-з’єднання.

Для коректної роботи маршрутизації використовується файл vercel.json із правилом rewrites, яке перенаправляє всі запити на головний документ застосунку. Це дозволяє відкривати вкладені маршрути без помилки 404.

У демонстраційному режимі частина стану зберігається у localStorage через redux-persist. Додатково mock-сервіси використовують окреме локальне сховище, що дозволяє зберігати дані між перезавантаженнями сторінки та синхронізувати їх між вкладками браузера.

У випадку некоректної роботи системи рекомендується повторно увійти в систему, очистити localStorage браузера або виконати жорстке перезавантаження сторінки.

До типових проблем під час запуску належать:

– відсутність встановлених залежностей;

– помилки у файлі .env;

– неправильні адреси API;

– застарілі дані у localStorage;

– блокування спливаючих вікон браузером під час друку.

У перспективі система може бути доповнена повноцінним backend-сервісом, централізованою базою даних і серверною реалізацією всіх API-модулів.

Таким чином, розгортання TrackTara не потребує складної інфраструктури та дозволяє швидко запускати демонстраційну версію застосунку для тестування й перевірки функціоналу.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи було розглянуто програмне та технічне забезпечення веб-застосунку TrackTara, а також особливості його реалізації та використання у контексті складських операцій.

Обґрунтовано вибір сучасного стеку технологій, зокрема React як основи побудови інтерфейсу користувача, Vite як інструмента швидкої збірки та розробки, а також використання Redux Toolkit для централізованого управління станом. Застосування redux-persist дозволило забезпечити збереження даних між сесіями, а redux-thunk — реалізувати асинхронну взаємодію з сервісним шаром.

Визначено вимоги до технічного та програмного забезпечення, які є мінімальними для роботи системи, що забезпечує її доступність для впровадження без значних витрат на інфраструктуру. Описано особливості роботи у браузері, використання сканера як пристрою введення та необхідність доступу до мережі Інтернет для завантаження застосунку.

Розглянуто архітектуру frontend-застосунку, яка базується на модульному підході з чітким розділенням рівнів: інтерфейс, маршрутизація, стан та сервісний шар. Окрему увагу приділено використанню ServiceFactory для перемикання між mock-режимом та реальним API, що забезпечує гнучкість розробки та можливість подальшої інтеграції.

Детально описано реалізацію основних функціональних модулів системи, включаючи облік тари, видачу у візок, пакування, роботу з коробками та ведення реєстру нестач. Показано, що ключові бізнес-процеси реалізовані на клієнтській стороні із дотриманням необхідних перевірок та обмежень.

Особливу увагу приділено модулю пакування як центральному елементу системи. Реалізовано сценарії швидкого та модального додавання товарів у коробки, контроль залишків, попередження щодо невідповідності траси та можливість перекладання між коробками. Це дозволяє мінімізувати помилки оператора та забезпечити контрольованість процесу.

Описано механізм формування та друку пакувального листа із використанням стандартних засобів браузера, а також реалізацію обчислення ваги на основі довідкових значень.

Розроблено керівництво користувача, яке охоплює всі основні сценарії роботи із системою та дозволяє швидко освоїти функціонал без додаткових інструкцій.

Окремо розглянуто процес розгортання застосунку, включаючи локальний запуск, збірку та демонстраційне розміщення на хостингу. Визначено типові проблеми та способи їх усунення, що спрощує супровід системи.

У цілому, розроблений frontend-застосунок TrackTara забезпечує повноцінну підтримку ключових складських процесів у межах демонстраційної моделі та створює основу для подальшого розвитку з інтеграцією серверної частини та розширенням функціональності.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було поставлено мету розробки веб-застосунку для підтримки складських операцій, зокрема обліку тари, комплектування замовлень, пакування товарів у коробки та контролю супровідних процесів. Для досягнення цієї мети сформульовано та послідовно вирішено ряд задач, що охоплюють аналіз предметної області, проєктування інформаційної моделі, визначення вимог до системи, вибір технологічного стеку, реалізацію клієнтської частини та перевірку працездатності основних сценаріїв. У практичній частині реалізовано frontend-застосунок TrackTara, який демонструє роботу ключових процесів складської діяльності у вигляді інтерактивної SPA-системи.

У результаті виконаної роботи створено клієнтський веб-застосунок, що підтримує основні ролі користувачів — оператора складу, адміністратора та менеджера продажів — із розмежуванням доступу до функціоналу на рівні маршрутизації. Реалізовано основні модулі системи: видача товарів у візки, пакування у коробки, створення замовлень, ведення реєстру нестач, облік тари та управління довідниками. Особливу увагу приділено сценаріям пакування, які є центральними для системи: реалізовано механізм перевірки «візок + код товару або тари», розрахунок залишку для пакування, швидке та модальне додавання товарів у коробки, а також можливість перекладання між коробками в межах одного замовлення. Інтерфейс системи орієнтовано на інтенсивну роботу оператора, що проявляється у мінімізації кількості дій, використанні великих елементів керування та швидкому зворотному зв’язку через повідомлення.

Архітектура застосунку побудована за принципами односторінкових веб-додатків із використанням бібліотеки React як основи для побудови інтерфейсу. Такий підхід дозволяє забезпечити динамічне оновлення стану без перезавантаження сторінки та покращує взаємодію користувача із системою. Для організації централізованого стану застосовано Redux Toolkit, що дозволяє забезпечити узгодженість даних між різними модулями. Використання redux-persist дало змогу зберігати стан між перезавантаженнями сторінки, що є важливим для безперервності

роботи користувача. Асинхронні операції реалізовано за допомогою redux-thunk, що дозволяє інкапсулювати виклики сервісного шару.

Інструмент збірки Vite обрано з огляду на швидкість запуску та ефективність розробки, що особливо важливо для сучасних frontend-проєктів. Маршрутизація реалізована за допомогою React Router із використанням захищених маршрутів, які забезпечують контроль доступу на основі ролей користувача. Інтерфейс побудовано з використанням Bootstrap як основного UI-набору, що забезпечує адаптивність та уніфікований вигляд, а також доповнено власними стилями та окремими компонентами.

У роботі реалізовано сервісний шар із можливістю перемикання між демонстраційним режимом (mock API) та потенційним режимом роботи з реальним сервером. Це досягається за допомогою конфігурації середовища, що дозволяє використовувати одні й ті самі інтерфейси сервісів незалежно від джерела даних. У демонстраційній версії всі дані обробляються на клієнтській стороні, що дозволяє відтворити логіку роботи системи без розгортання backend-інфраструктури.

З точки зору надійності, у системі реалізовано базові механізми перевірки даних та обробки помилок. Передбачено валідацію введених значень, контроль допустимих операцій, а також інформування користувача через повідомлення. Використання збереження стану дозволяє уникнути втрати даних під час перезавантаження сторінки. Водночас існують обмеження: відсутність серверної частини, відсутність синхронізації між користувачами, обмежена безпека в демонстраційному режимі.

Практична цінність розробленого застосунку полягає у можливості використання як прототипу для автоматизації складських процесів. Система може бути корисною для операторів складу, адміністраторів та менеджерів продажів. Запропонований підхід дозволяє поетапно впроваджувати автоматизацію без значних витрат.

Подальший розвиток системи може бути спрямований на: розробку backend та бази даних; впровадження OpenAPI; автоматизоване тестування; покращення

безпеки; реалізацію офлайн-режиму; додавання моніторингу; розширення аналітики; локалізацію інтерфейсу.

Таким чином, у роботі досягнуто поставленої мети — розроблено frontend-застосунок TrackTara, який демонструє можливості автоматизації складських процесів та може слугувати основою для подальшого розвитку інформаційної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. React – A JavaScript library for building user interfaces. URL: https://react.dev/ (дата звернення: 15.03.2026).
2. Vite Documentation. Next Generation Frontend Tooling. URL: https://vitejs.dev/ (дата звернення: 12.02.2026).
3. Redux Toolkit Documentation. URL: https://redux-toolkit.js.org/ (дата звернення: 20.01.2026).
4. React Router Documentation. URL: https://reactrouter.com/ (дата звернення: 10.02.2026).
5. Axios Documentation. Promise based HTTP client for the browser. URL: https://axios-http.com/ (дата звернення: 05.01.2026).
6. Bootstrap Documentation. URL: https://getbootstrap.com/ (дата звернення: 18.12.2025).
7. OWASP Foundation. Web Application Security Testing Guide. URL: https://owasp.org/ (дата звернення: 22.03.2026).
8. IETF. The OAuth 2.0 Authorization Framework (RFC 6749). URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 11.02.2026).
9. IETF. JSON Web Token (JWT) (RFC 7519). URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 09.01.2026).
10. Mozilla Developer Network. Web Docs. URL: https://developer.mozilla.org/ (дата звернення: 03.03.2026).
11. Google Developers. OAuth 2.0 for Web Server Applications. URL: https://developers.google.com/identity/protocols/oauth2 (дата звернення: 14.02.2026).
12. Vercel Documentation. Deployment and Hosting Platform. URL: https://vercel.com/docs (дата звернення: 21.03.2026).
13. ECMAScript Specification. JavaScript Language Specification. URL: https://tc39.es/ecma262/ (дата звернення: 08.01.2026).
14. Node.js Documentation. URL: https://nodejs.org/ (дата звернення: 17.02.2026).
15. Nielsen J. Usability Engineering. – Academic Press, 1994.
16. Pressman R. Software Engineering: A Practitioner’s Approach. – McGraw-Hill, 2014.
17. Sommerville I. Software Engineering. – Pearson, 2016.
18. Gamma E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1994.
19. W3C. Web Accessibility Initiative (WAI). URL: https://www.w3.org/WAI/ (дата звернення: 02.04.2026).
20. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 06.03.2026).
21. Microsoft Docs. REST API design best practices. URL: https://learn.microsoft.com/ (дата звернення: 25.02.2026).
22. Google Web Fundamentals. Progressive Web Apps. URL: https://web.dev/ (дата звернення: 01.03.2026).

23. Stack Overflow. Developer community discussions. URL: https://stackoverflow.com/ (дата звернення: 10.03.2026).
24. Redux Persist Documentation. URL: https://github.com/rt2zz/redux-persist (дата звернення: 19.01.2026).
25. JavaScript Info. Modern JavaScript Tutorial. URL: https://javascript.info/ (дата звернення: 27.02.2026).
26. DigitalOcean. How To Build a React Application. URL: https://www.digitalocean.com/community/tutorials (дата звернення: 16.02.2026).
27. Amazon Web Services. Introduction to Warehouse Management Systems. URL: https://aws.amazon.com/ (дата звернення: 05.04.2026).
28. Oracle. Database Concepts. URL: https://docs.oracle.com/ (дата звернення: 09.03.2026).
29. Atlassian. Software Development Best Practices. URL: https://www.atlassian.com/ (дата звернення: 18.03.2026).
30. IEEE Xplore. Research on warehouse management systems. URL: https://ieeexplore.ieee.org/ (дата звернення: 12.04.2026).

ДОДАТКИ

Додаток А. Скріншоти інтерфейсу

Додаток містить екранні знімки основних екранів клієнтської частини системи TrackTara для ілюстрації користувацького інтерфейсу та ключових сценаріїв.

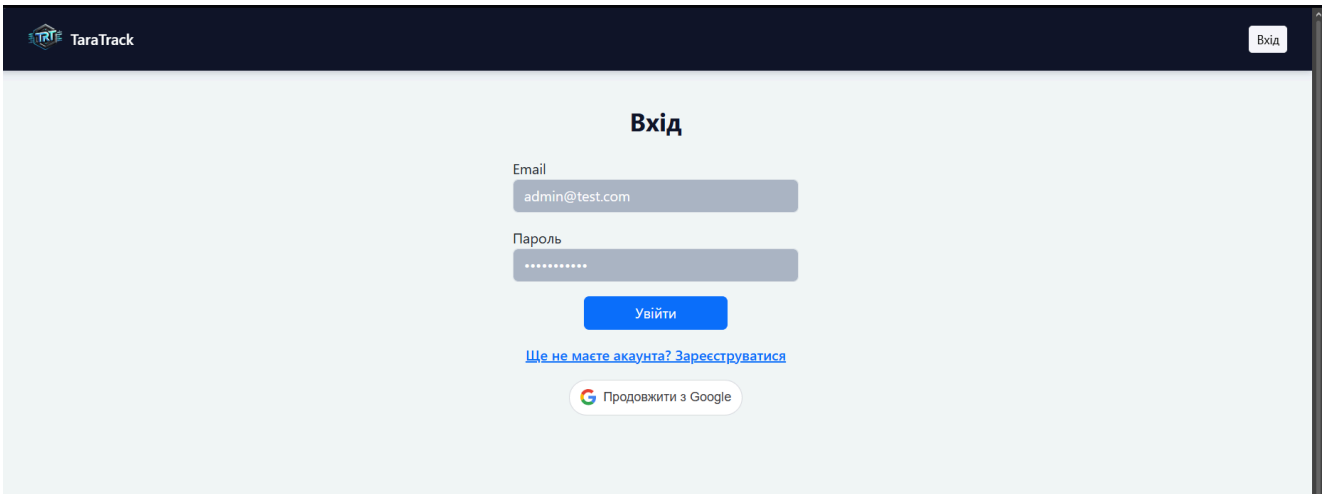


Рис. Дод. А.1 — Екран входу

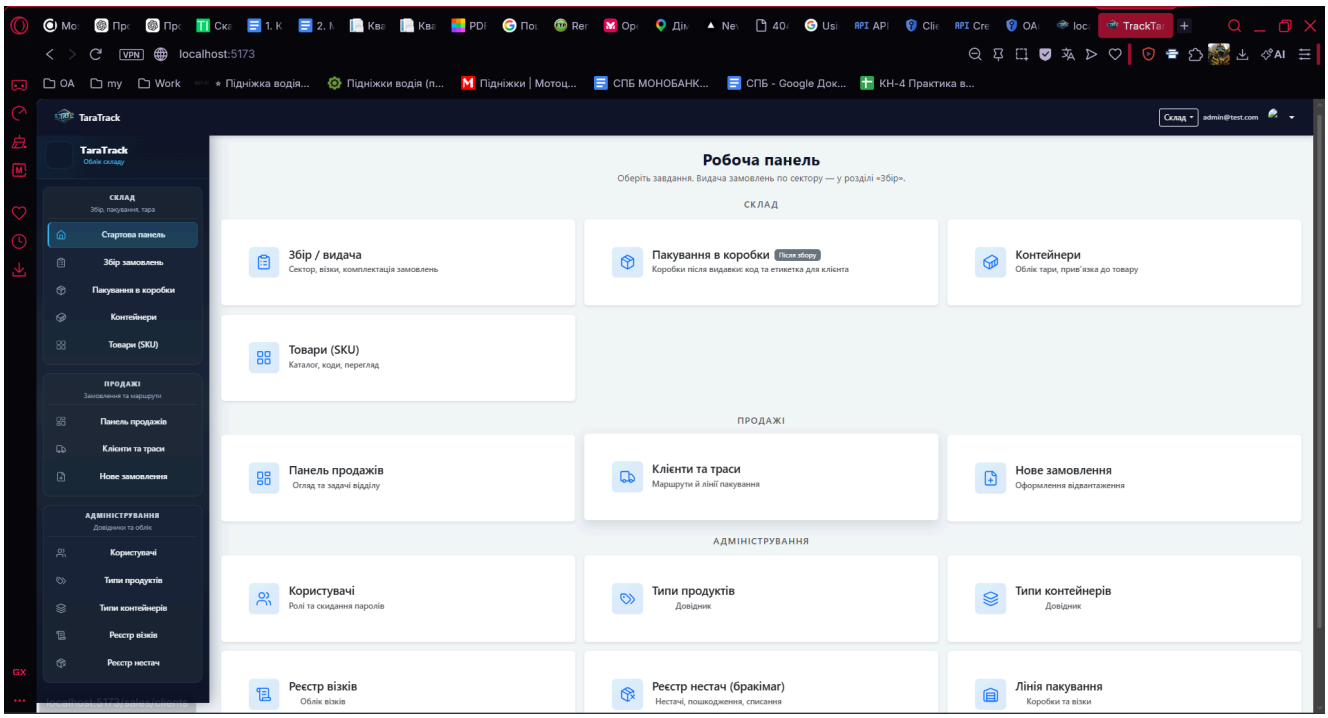


Рис. Дод. А.2 — Стартова панель (Warehouse Hub)

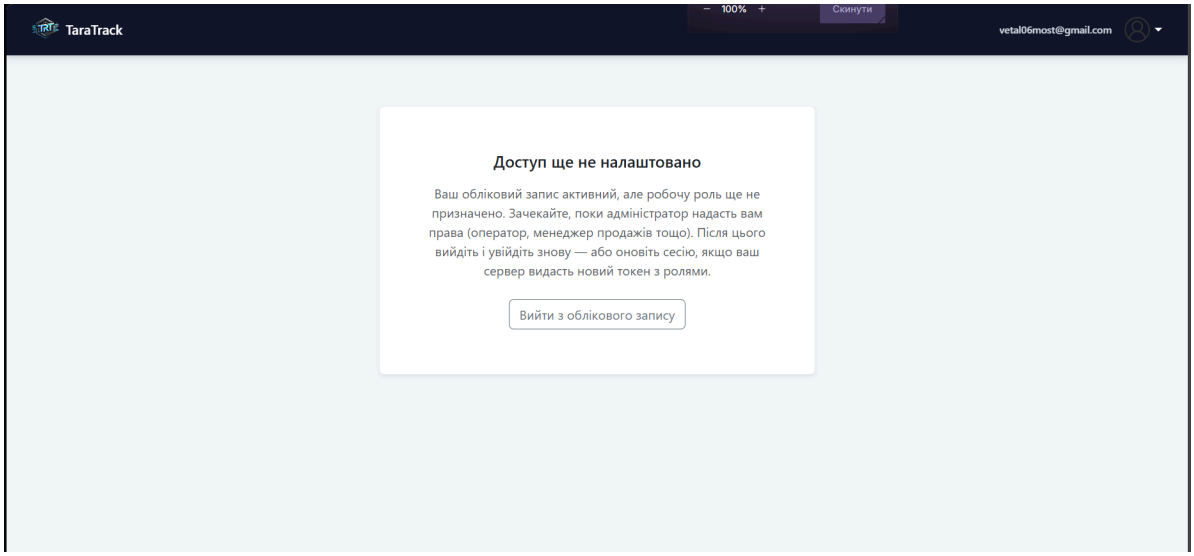


Рис. Дод. А.3 — Екран для користувача без ролі

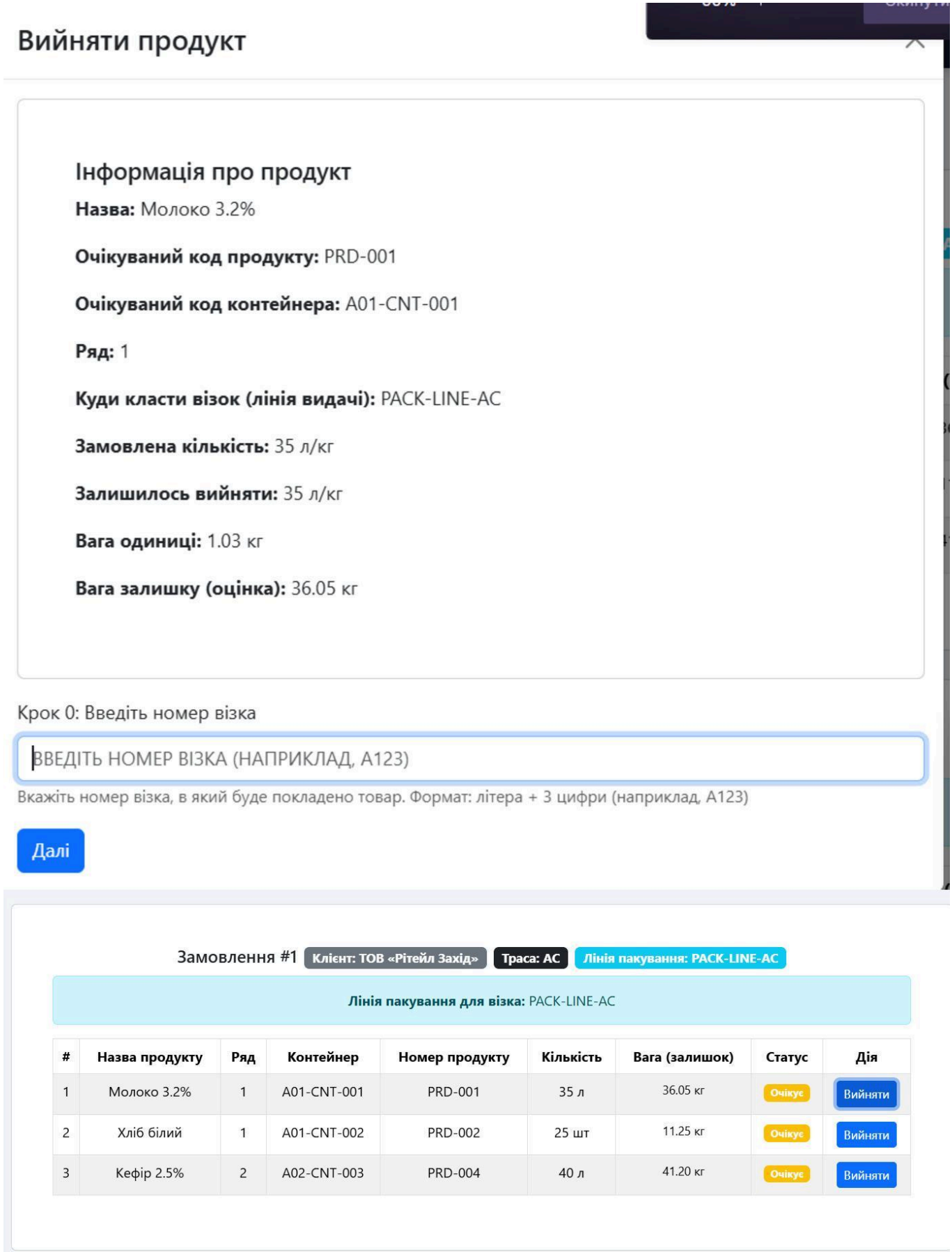


Рис. Дод. А.4 — Видача у візок

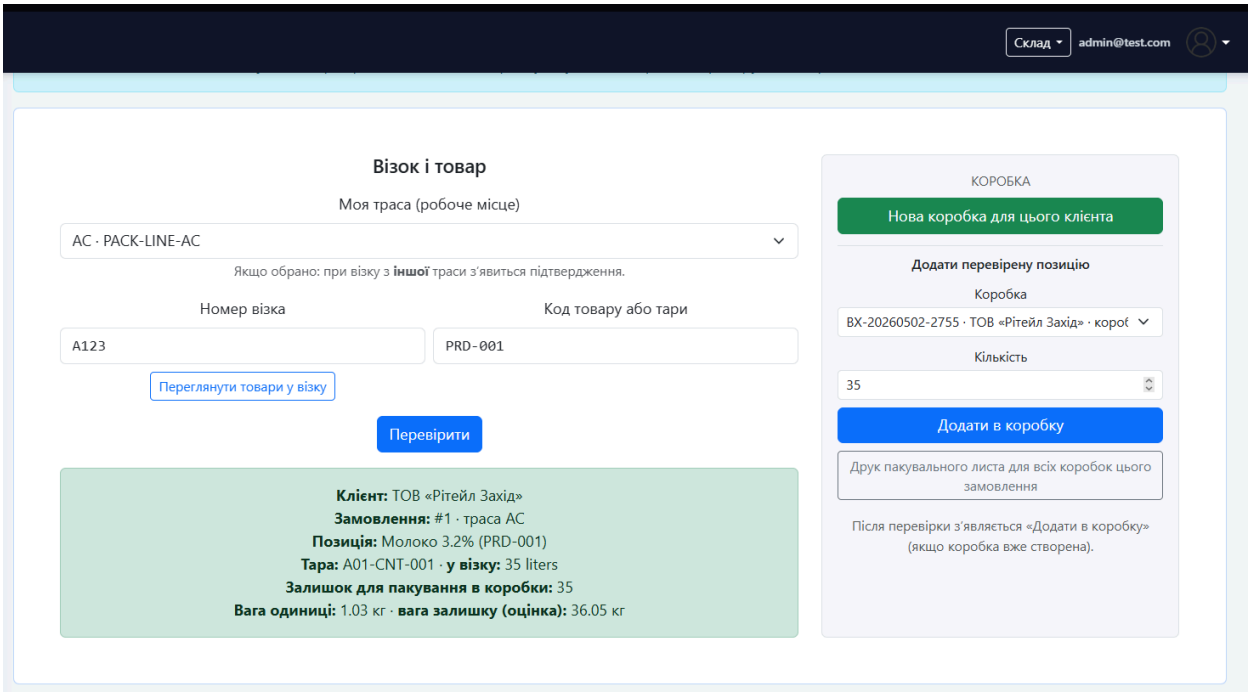


Рис. Дод. А.5 — Пакування в коробки

Рис. Дод. А.6 — Вміст візка

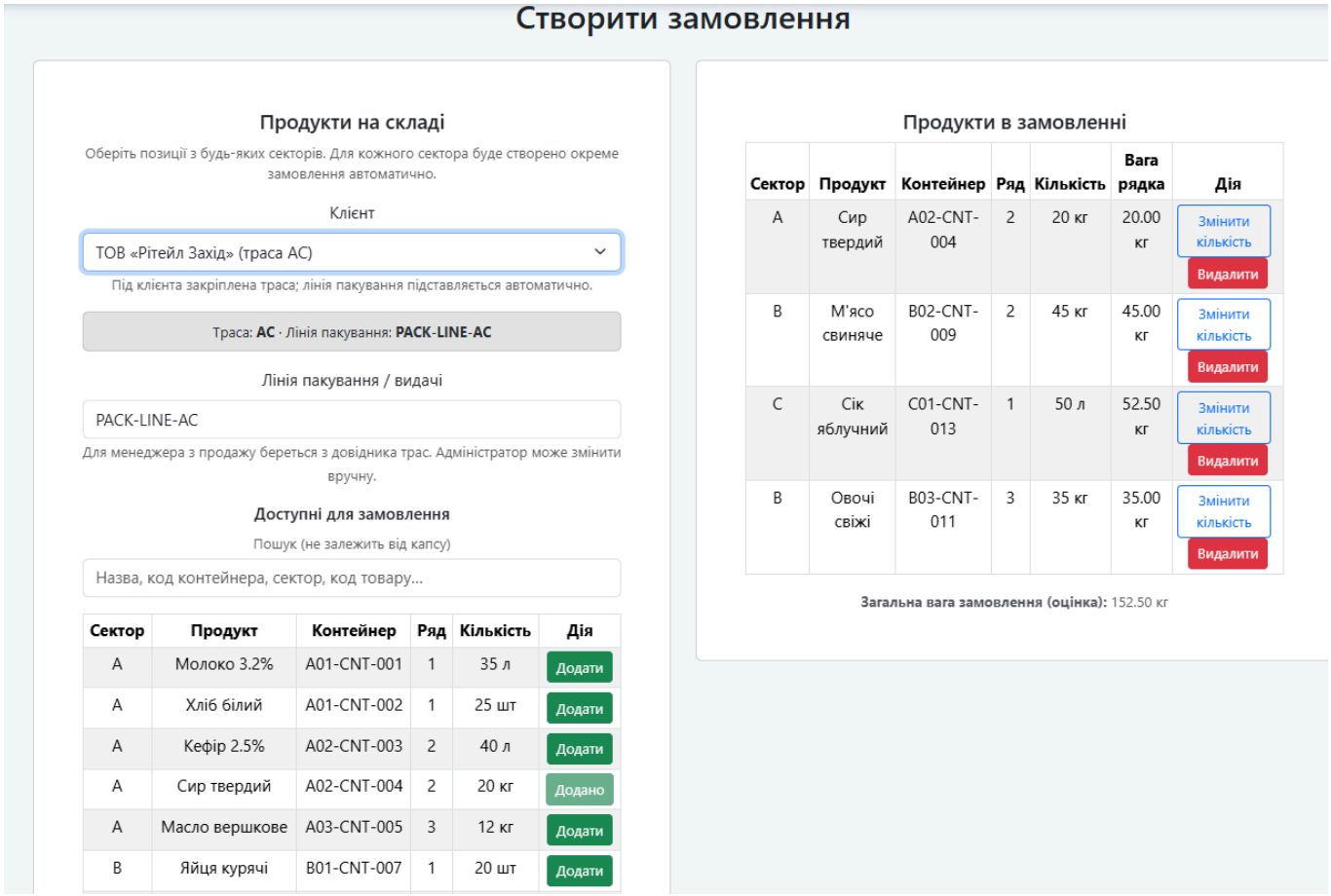


Рис. Дод. А.7 — Створення замовлення

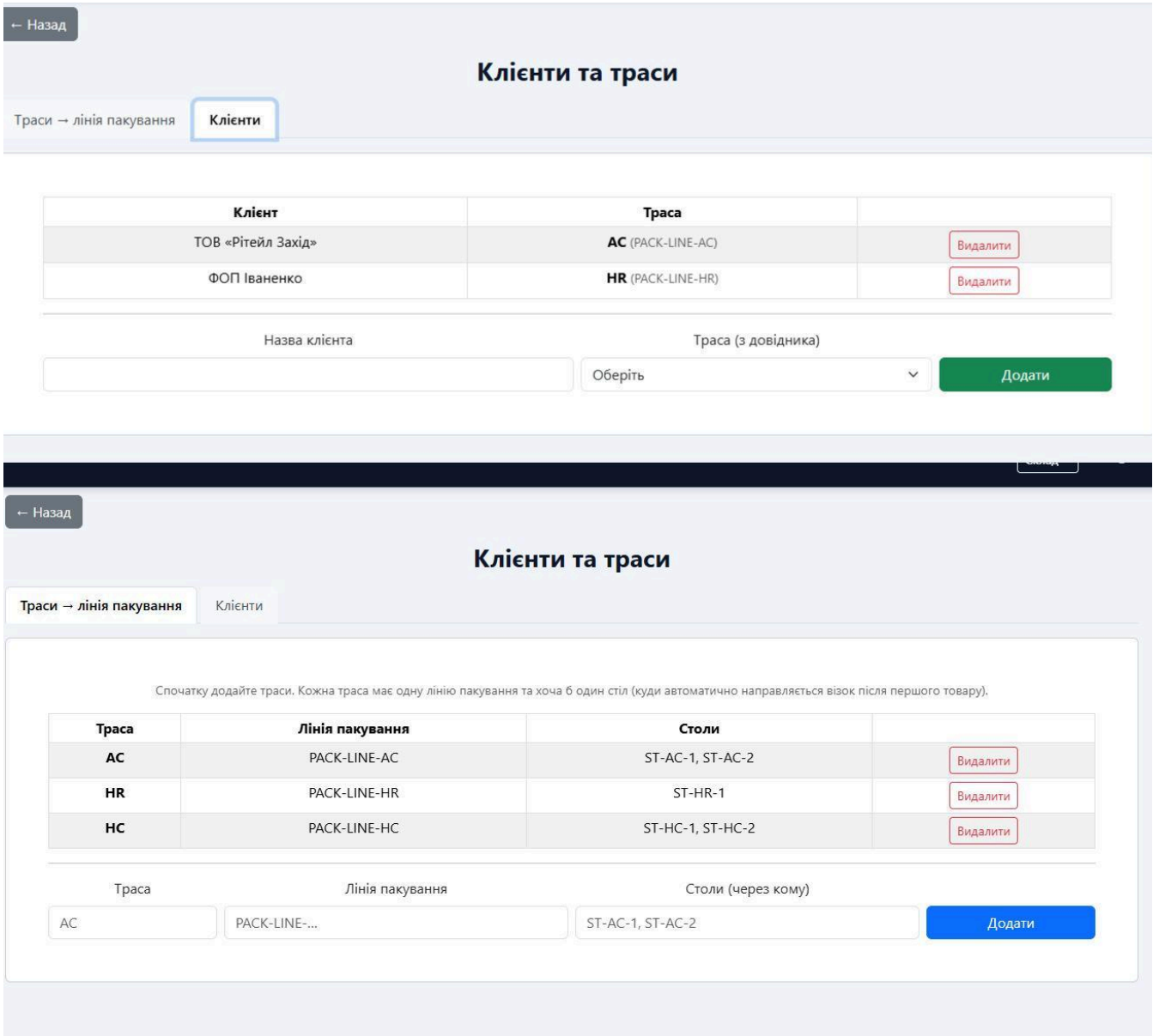


Рис. Дод. А.8 — Панель продажів

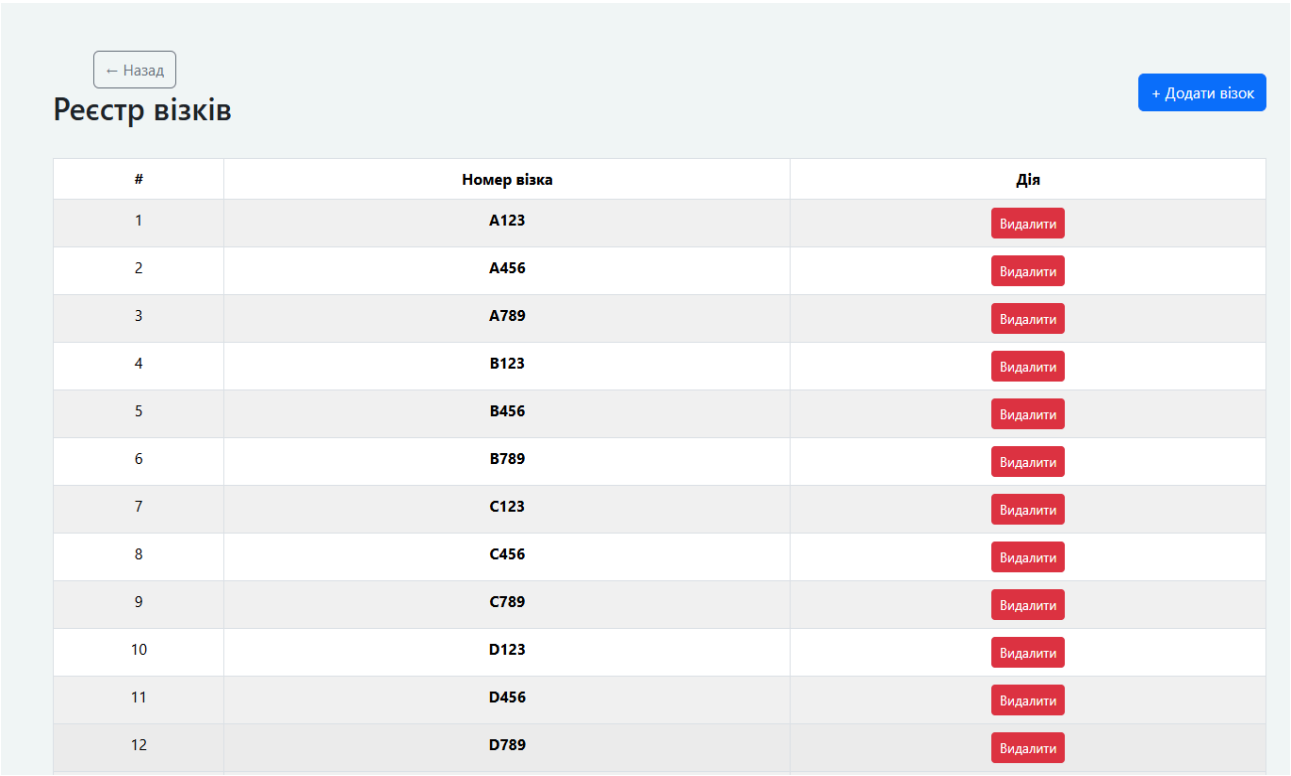


Рис. Дод. А.9 — Реєстр тари

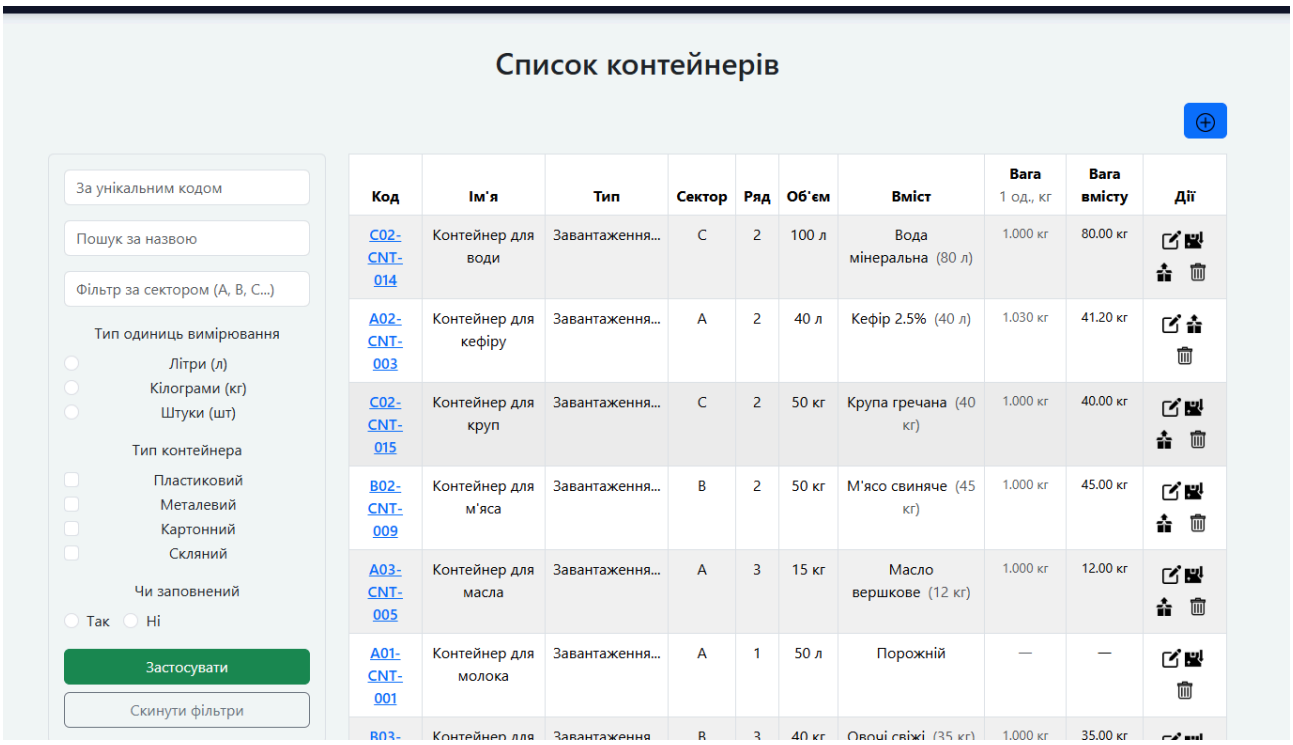


Рис. Дод. А.10 — Сторінка контейнерів

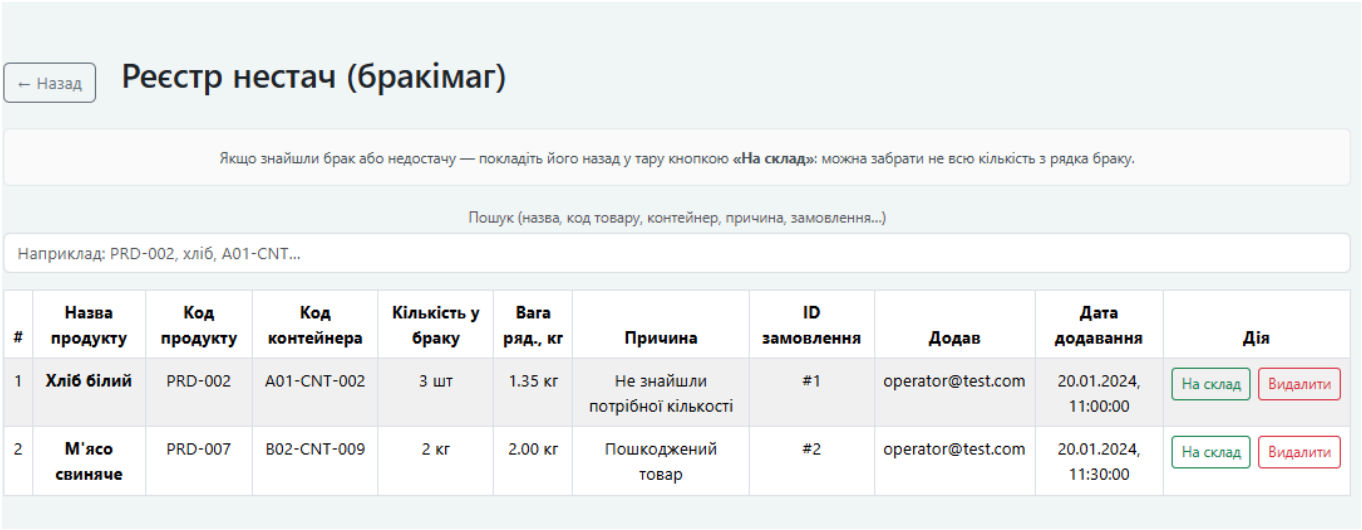


Рис. Дод. А.11 — Реєстр нестач

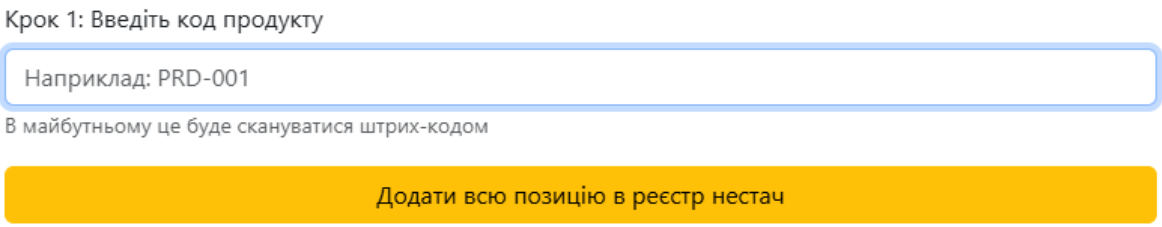


Рис. Дод. А.12 — Бракування товару

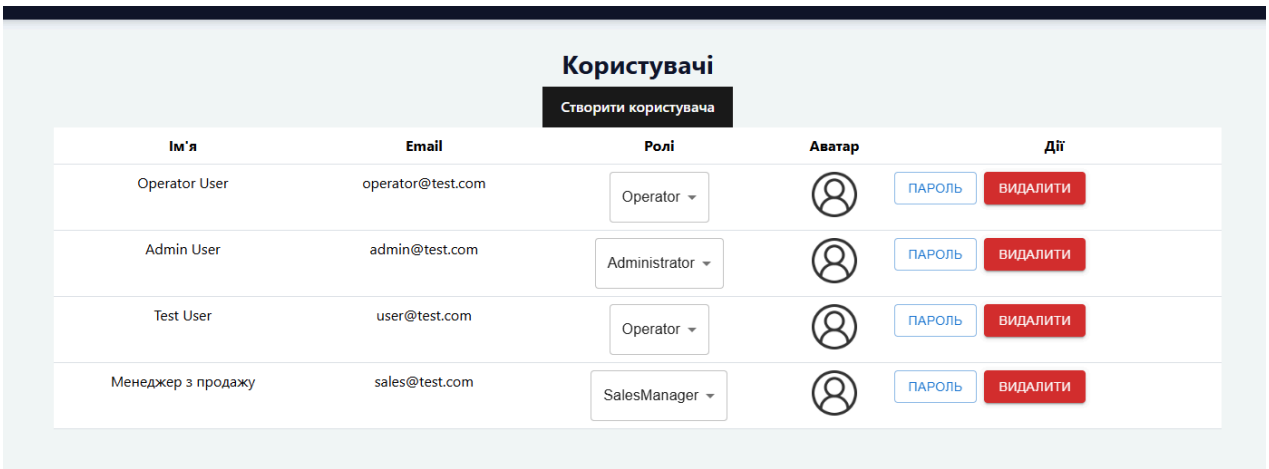


Рис. Дод. А.13 — Користувачі та ролі

Додаток Б. UML та ER-діаграми

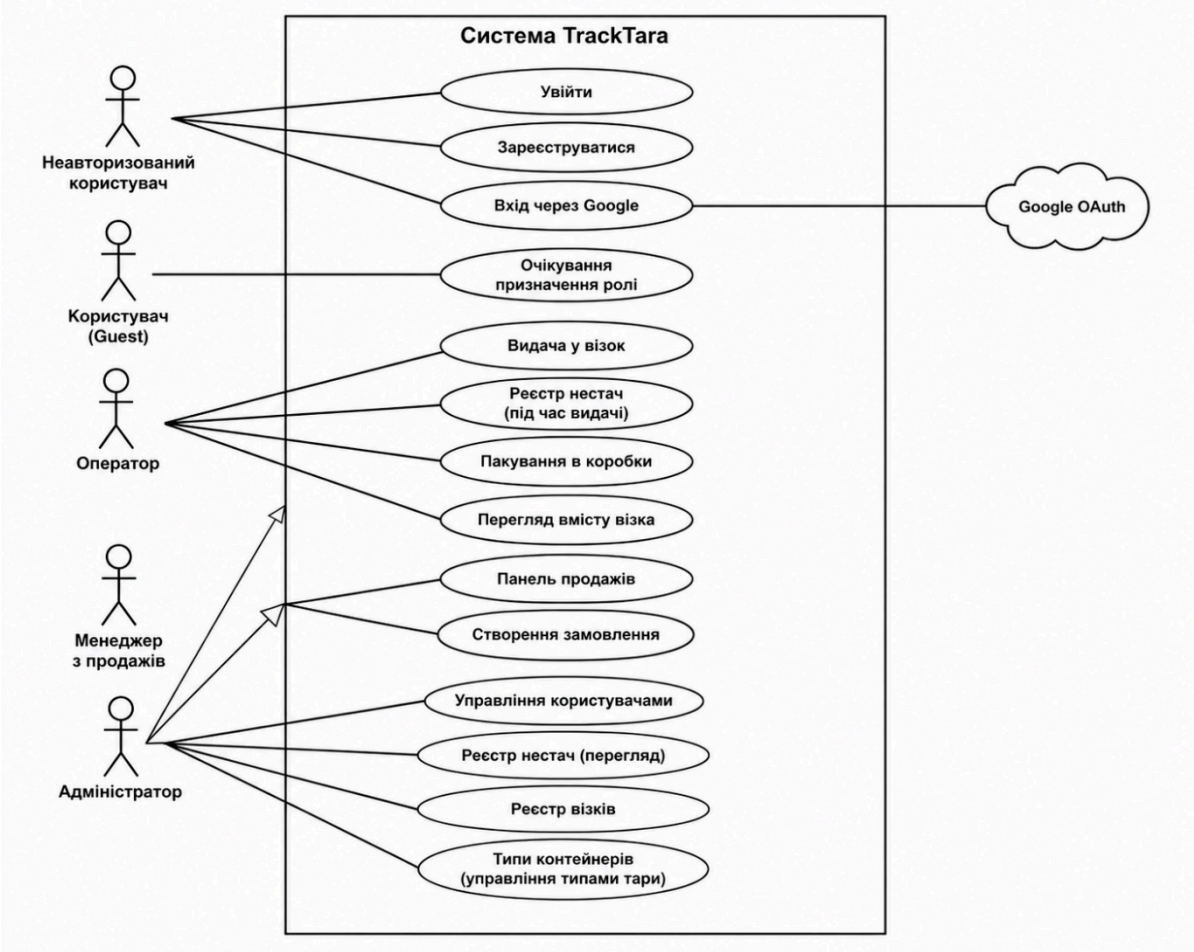


Рис. Дод. Б.1 — Діаграма варіантів використання

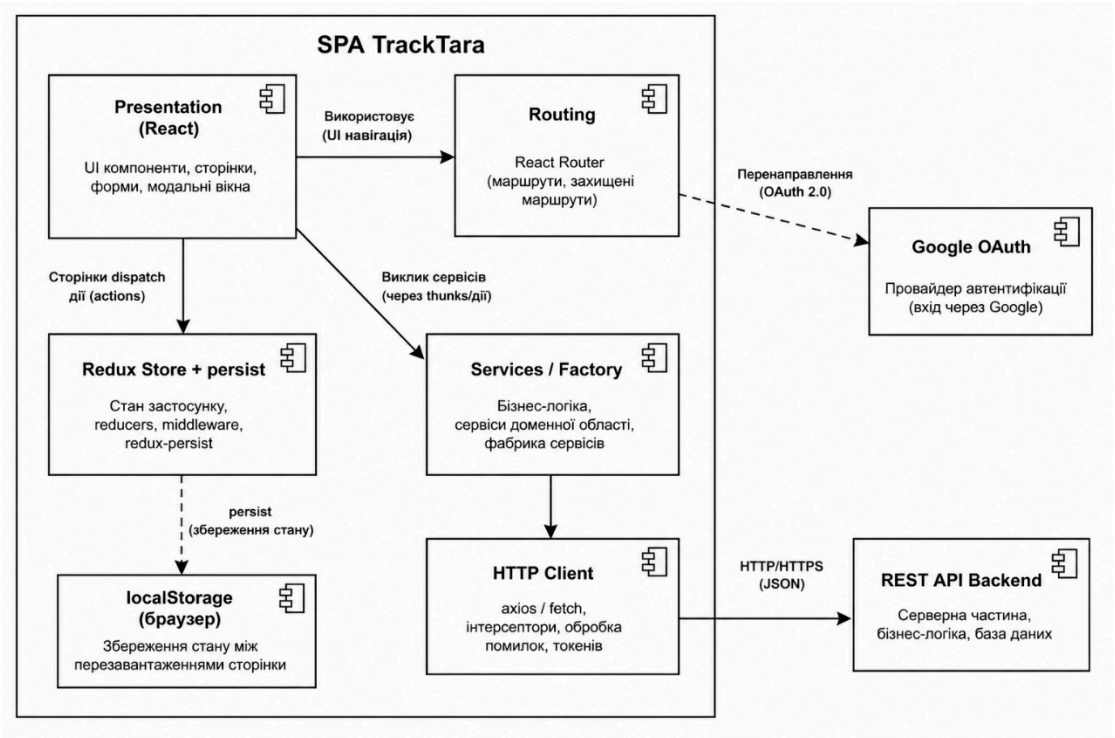


Рис. Дод. Б.2 — Діаграма компонентів

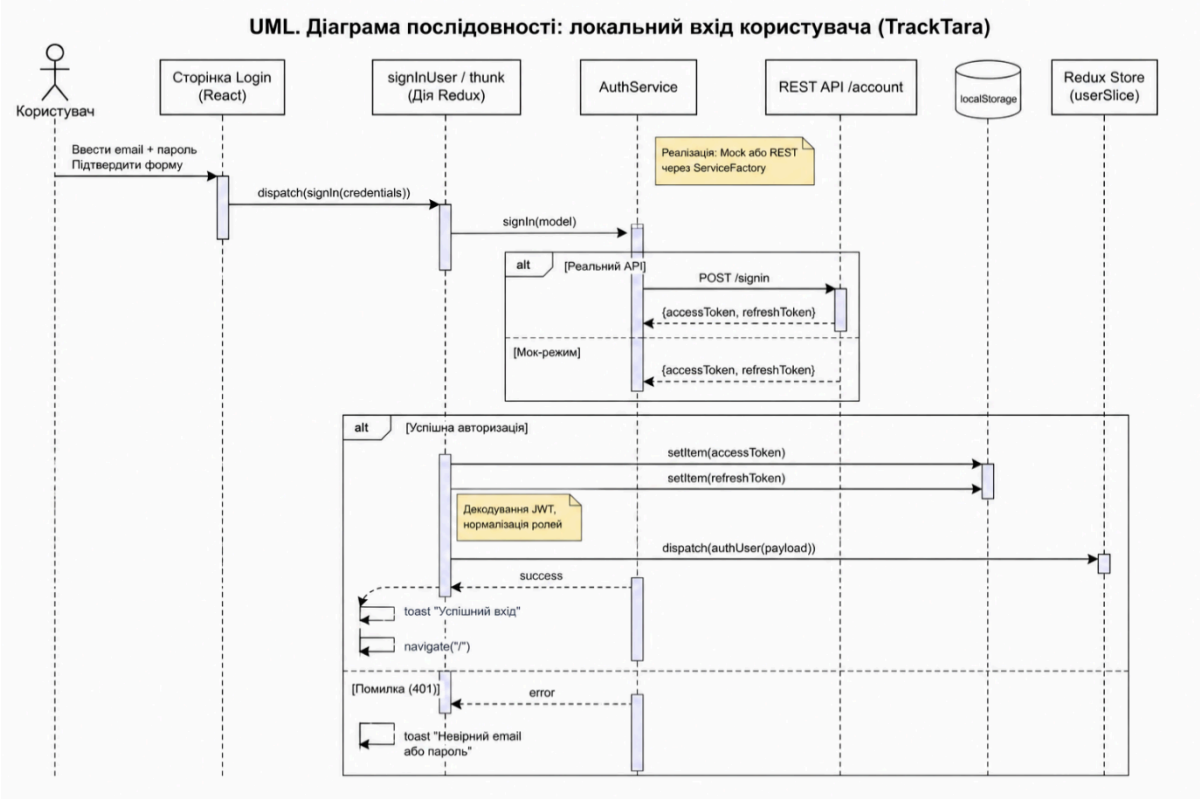


Рис. Дод. Б.3 — Діаграма послідовностей (логін)

Рис. Дод. Б.4 — Оновлення токена

Рис. Дод. Б.6 — Діаграма розгортання

Додаток В. Функціональні вимоги

Таблиця В.1

Специфікація функціональних вимог

ID	Назва	Опис	Актори	Пріоритет
B.1	Авторизація	Вхід користувача	Усі	Високий
B.6	Видача	Видача у візок	Operator	Високий
B.8	Пакування	Додавання у коробки	Operator	Високий
B.10	Замовлення	Створення	Admin/Sales	Високий

Додаток Г. Тест-кейси

Таблиця Г.1

Основні тест-кейси

№	Модуль	Кроки	Результат
1	Логін	Ввести дані	Успішний вхід
2	Пакування	Додати товар	Додано
3	Видача	Ввести код	Оброблено

ДОДАТОК Д. Матриця та ключовий код

Д.1 Матриця «вимога → тест»

Таблиця Д.1

Відповідність вимог та тест-кейсів

ID вимоги	‘Тест-кейси (№ з табл. Г.1)	Коментар
B.1	1, 2, 22	Авторизація та негативний сценарій
B.2	4	OAuth та бекенд
B.3	6, 7	Дозволені маршрути
B.4	5	Екран очікування ролі
B.5	6	Hub
B.6	8, 9	Видача
B.7	10	Нестачі
B.8	11, 12	Пакування
B.9	13	Перекладання коробок
B.10	14	Замовлення
B.11	15, 16	Тара
B.12	(окремий план)	Продукти — доповнити тест-кейси
B.13	23	Продажі
B.14	17	Користувачі
B.15	25	Реєстр нестач
B.16	(за наявності)	Реєстр візків
B.17	(за наявності)	Профіль
B.18	19	Сесія / перезавантаження
B.19	20, 21	Mock / real

ID вимоги	‘Тест-кейси (№ з табл. Г.1)	Коментар
B.20	2, 9	Повідомлення
B.21	24	Адаптивність
B.22	22	Відсутність токена
B.23	18	Вихід із системи
B.24	19	Робота у вкладках

Д.2 Лістинги програмного коду (основні фрагменти TrackTara)

Примітка: у текстовому редакторі (Word) фрагменти коду необхідно оформити шрифтом **Courier New, 9–10 pt**, з одинарним інтервалом.

ЛІСТИНГ Д.2.1 — файл: `src/utils/config/apiConfig.js`

```
// Конфігурація для перемикання між реальним API та моками.
// Дефолт: моки УВІМКНЕНО (зокрема для Vercel-демо без
бекенду).
// Реальний API — лише при явному VITE_USE_MOCK_API=false.
const useMockDefault = import.meta.env.VITE_USE_MOCK_API
!== 'false';

// Спільний хост для legacy-ендпойнтів (категорії,
виробники, кошик тощо).
const legacyOrigin =
  import.meta.env.VITE_API_LEGACY_ORIGIN ||
'http://13.60.245.135:4312';

export const API_CONFIG = {
  USE_MOCK_API: useMockDefault,

  BASE_URLS: {
    ACCOUNT: import.meta.env.VITE_API_ACCOUNT_URL ||
'http://localhost:5081/account',
    USERS: import.meta.env.VITE_API_USERS_URL ||
'http://localhost:5081/users',
    PRODUCTS: import.meta.env.VITE_API_PRODUCTS_URL ||
'http://localhost:5081/products',
    CONTAINERS: import.meta.env.VITE_API_CONTAINERS_URL ||
'http://localhost:5081/containers',
```

```
    CONTAINER_TYPES:
      import.meta.env.VITE_API_CONTAINER_TYPES_URL ||
'http://localhost:5081/containers-type',
    PRODUCT_TYPES:
      import.meta.env.VITE_API_PRODUCT_TYPES_URL ||
'http://localhost:5081/product-types',
    ROLES: import.meta.env.VITE_API_ROLES_URL ||
'http://localhost:5081/roles',
    SECTORS: import.meta.env.VITE_API_SECTORS_URL ||
'http://localhost:5081/sectors',
    CATEGORIES: import.meta.env.VITE_API_CATEGORIES_URL ||
`${legacyOrigin}/categories`,
    MANUFACTURERS:
import.meta.env.VITE_API_MANUFACTURERS_URL ||
`${legacyOrigin}/manufacturers`,
    CART_ITEMS: import.meta.env.VITE_API_CART_ITEMS_URL ||
`${legacyOrigin}/cart-items`,
    PRODUCT_IMAGES_ORIGIN:
      import.meta.env.VITE_API_PRODUCT_IMAGES_ORIGIN ||
legacyOrigin,
    // Операційні модулі з гібридним fallback на моки, якщо
ендпоінта ще немає.
    BRAKIMAG: import.meta.env.VITE_API_BRAKIMAG_URL ||
'http://localhost:5081/brakimag',
    CART_REGISTRY:
      import.meta.env.VITE_API_CART_REGISTRY_URL ||
'http://localhost:5081/cart-registry',
    PACKING_BOXES:
      import.meta.env.VITE_API_PACKING_BOXES_URL ||
'http://localhost:5081/packing-boxes',
  },

  MOCK_DELAY: 500,
};
```

ЛІСТИНГ Д.2.2 — файл: `src/utils/services/ServiceFactory.js` (фрагмент)

```
import { API_CONFIG } from '../config/apiConfig';

// Реальні сервіси
import { AuthService } from './AuthService';
import { UserService } from './UserService';
import { ProductService } from './ProductService';
import * as ContainerService from './ContainerService';
```

```
import * as ContainerTypesService from
'./ContainerTypesService';
import * as ProductTypesService from
'./ProductTypesService';
import { RoleService } from './RoleService';
import { SectorService as RealSectorService } from
'./SectorService';
import * as ContainerHistoryService from
'./ContainerHistoryService';
import * as ProductHistoryService from
'./ProductHistoryService';
import { OrderService } from './OrderService';
import { ClientRouteService } from './ClientRouteService';

// Мок-сервіси
import { MockAuthService } from './mock/MockAuthService';
import { MockUserService } from './mock/MockUserService';
import { MockProductService } from
'./mock/MockProductService';
import * as MockContainerService from
'./mock/MockContainerService';
import * as MockContainerTypesService from
'./mock/MockContainerTypesService';
import * as MockProductTypesService from
'./mock/MockProductTypesService';
import { MockRoleService } from './mock/MockRoleService';
import { MockSectorService } from
'./mock/MockSectorService';
import * as MockContainerHistoryService from
'./mock/MockContainerHistoryService';
import * as MockProductHistoryService from
'./mock/MockProductHistoryService';
import { OrderService as MockOrderService } from
'./mock/MockOrderService';
import { MockClientRouteService } from
'./mock/MockClientRouteService';

const useMock = API_CONFIG.USE_MOCK_API;

export const AuthServiceInstance = useMock ?
MockAuthService : AuthService;
export const UserServiceInstance = useMock ?
MockUserService : UserService;
export const ProductServiceInstance = useMock ?
MockProductService : ProductService;
```

```
export const ContainerServiceInstance =
  useMock ? MockContainerService : ContainerService;
export const ContainerTypesServiceInstance =
  useMock ? MockContainerTypesService :
ContainerTypesService;
export const ProductTypesServiceInstance =
  useMock ? MockProductTypesService : ProductTypesService;
export const RoleServiceInstance = useMock ?
MockRoleService : RoleService;
export const ContainerHistoryServiceInstance =
  useMock ? MockContainerHistoryService :
ContainerHistoryService;
export const ProductHistoryServiceInstance =
  useMock ? MockProductHistoryService :
ProductHistoryService;
export const OrderServiceInstance = useMock ?
MockOrderService : OrderService;
export const ClientRouteServiceInstance =
  useMock ? MockClientRouteService : ClientRouteService;
export const SectorService = useMock ? MockSectorService :
RealSectorService;

// Зворотна сумісність зі старими імпортами по імені
класу/об'єкта.
export { AuthServiceInstance as AuthService };
export { UserServiceInstance as UserService };
export { ProductServiceInstance as ProductService };
export { RoleServiceInstance as RoleService };
export { OrderServiceInstance as OrderService };
export { ClientRouteServiceInstance as ClientRouteService
};

// Операційні модулі: при VITE_USE_MOCK_API=false виконують
HTTP-виклик
// до бекенду, а у разі відсутності ендпоінта (404/405/501)
автоматично
// відкочуються на персистентні моки через
operationalApiShared.
export {
  getAllPackingBoxes, createPackingBox, deletePackingBox,
  addToPackingBox, transferPackingBoxContent,
} from './PackingBoxService';

export {
  getAllBrakiMagItems, addToBrakiMag, removeFromBrakiMag,
```

```
  updateBrakiMagItem, transferBrakiMagToContainer,
} from './BrakiMagService';

export {
  validateCartNumber, getAllCarts,
  addCartToRegistry, deleteCartFromRegistry,
} from './CartRegistryService';
```

ЛІСТИНГ Д.2.3 — файл: `src/routes/ProtectedRoute.jsx`

```
const ProtectedRoute = ({ children, allowedRoles }) => {
  const token = localStorage.getItem("accessToken");
  let user = null;

  try {
    user = token ? jwtDecode(token) : null;
  } catch {
    user = null;
  }

  if (!token || !user) {
    return <Navigate to="/login" replace />;
  }

  const userRoles =
getNormalizedRolesFromAccessToken(token);

  if (
    isAwaitingRoleAssignment(userRoles) &&
    !allowedRoles.includes(APP_ROLES.Guest)
  ) {
    return <Navigate to="/pending-role" replace />;
  }

  const isAuthorized = isRoleAllowed(userRoles,
allowedRoles);

  return isAuthorized
    ? children
    : <ErrorMessage error="Немає доступу до цього розділу.
Зверніться до адміністратора." />;
};
```

ЛІСТИНГ Д.2.4 — файл: `src/utils/helpers/userRoles.js`

(фрагмент)

```
export const APP_ROLES = {
  Administrator: "Administrator",
  Operator: "Operator",
  SalesManager: "SalesManager",
  // Обліковий запис без робочих прав — очікує призначення
ролі адміністратором.
  Guest: "Guest",
};

const CANONICAL_ORDER = [
  "Administrator",
  "Operator",
  "SalesManager",
  "Guest",
];

// Мапить довільний рядок з бекенду / старих токенів на
канонічну роль.
// Легасі-роль "User" зі старої версії продукту прав не
дає, тому
// явним аліасом перетворюється на Guest.
export function normalizeRoleName(raw) {
  if (raw == null || raw === "") return null;
  const s = String(raw).trim();
  if (!s) return null;

  const byCanon = CANONICAL_ORDER.find(
    (c) => c.toLowerCase() === s.toLowerCase()
  );
  if (byCanon) return byCanon;

  const compact = s.toLowerCase().replace(/[\s_\-]+/g, "");
  const aliases = {
    admin: APP_ROLES.Administrator,
    administrator: APP_ROLES.Administrator,
    operator: APP_ROLES.Operator,
    salesmanager: APP_ROLES.SalesManager,
    user: APP_ROLES.Guest,
    guest: APP_ROLES.Guest,
  };
  return aliases[compact] || null;
}
```

```
// Користувач у статусі «гість»: має лише роль Guest, без
жодної робочої ролі.
export function isAwaitingRoleAssignment(userRoles) {
  if (!Array.isArray(userRoles) || userRoles.length === 0)
return false;
  const hasWorkRole = userRoles.some((r) =>
    [APP_ROLES.Operator, APP_ROLES.Administrator,
APP_ROLES.SalesManager].includes(r)
  );
  return userRoles.includes(APP_ROLES.Guest) &&
!hasWorkRole;
}

// Чи дозволено вхід на маршрут зі списком allowedRoles
(адмін — повний доступ).
export function isRoleAllowed(userRoles, allowedRoles) {
  if (!Array.isArray(userRoles) || !userRoles.length)
return false;
  if (userRoles.includes(APP_ROLES.Administrator)) return
true;
  return allowedRoles.some((r) => userRoles.includes(r));
}
```

ЛІСТИНГ Д.2.5 — файл: `src/components/RoleHomeRedirect.jsx`

```
const RoleHomeRedirect = () => {
  const roles = useAppRoles();

  const onlySales =
    roles.includes("SalesManager") &&
    !roles.includes("Operator") &&
    !roles.includes("Administrator");

  if (onlySales) return <Navigate to="/sales" replace />;

  return <WarehouseHub />;
};
```

ЛІСТИНГ Д.2.6 — файл: `src/utils/http/HttpClient.js`

(інтерцептор)

```
this.axiosInstance.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;

    if (error.response?.status === 401 &&
!originalRequest._retry) {
      originalRequest._retry = true;

      try {
        const token = await refreshToken(
          originalRequest,
          this.setAuthorizationToken.bind(this)
        );

        originalRequest.headers["Authorization"] = `Bearer
${token}`;
        return this.axiosInstance(originalRequest);
      } catch (refreshError) {
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
);
```

ЛІСТИНГ Д.2.7 — файл: `src/utils/http/tokenService.js`

```
export const refreshToken = async (originalRequest,
setAuthorizationToken) => {
  if (isRefreshing) {
    return new Promise((resolve, reject) => {
      failedQueue.push({ resolve, reject });
    }).then((token) => {
      originalRequest.headers["Authorization"] = `Bearer
${token}`;
      setAuthorizationToken(token);
      return token;
    });
  }

  isRefreshing = true;

  try {
```

```
    const refreshToken =
localStorage.getItem("refreshToken");
    const accessToken =
localStorage.getItem("accessToken");

    const { AuthService } = await import('../services');
    const tokens = await AuthService.refreshToken({
refreshToken, accessToken });

    localStorage.setItem("accessToken",
tokens.accessToken);
    localStorage.setItem("refreshToken",
tokens.refreshToken);

    setAuthorizationToken(tokens.accessToken);

    const decoded = jwtDecode(tokens.accessToken);
    const user = normalizeUserPayloadForAuth(decoded);

    store.dispatch(authUser(user));

    processQueue(null, tokens.accessToken);

    return tokens.accessToken;
  } catch (error) {
    processQueue(error, null);
    await logoutUser()(store.dispatch);
    return Promise.reject(error);
  } finally {
    isRefreshing = false;
  }
};
```

ЛІСТИНГ Д.2.8 — файл: `src/store/store.js` (redux-persist)

```
const persistConfig = {
  key: 'tracktara',
  version: 2,
  storage,
  // Усі зведені ред'юсери — кошик, фільтри, кеш списків
тощо
  // переживають F5 (localStorage, ~5 МБ ліміт).
  whitelist: [
    'user',
    'containerHistory',
```

```
    'productHistory',
    'role',
    'category',
    'manufacturer',
    'product',
    'appSettings',
    'users',
    'filters',
    'containers',
    'cartItem',
    'productTypes',
    'containerTypes',
  ],
  // Міграція очищає застарілі поля, які лишились у
localStorage
  // після попередніх версій (напр. favoriteProducts на
user).
  migrate: (persistedState) => {
    if (!persistedState) return Promise.resolve(undefined);
    const next = { ...persistedState };
    if (
      next.user &&
      Object.prototype.hasOwnProperty.call(next.user,
'favoriteProducts')
    ) {
      delete next.user.favoriteProducts;
    }
    return Promise.resolve(next);
  },
};

const persistedReducer = persistReducer(persistConfig,
rootReducer);

export const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST,
PURGE, REGISTER],
      },
    }).concat(thunk),
});
```

```
export const persistor = persistStore(store);
```

ДОДАТОК Е

React Router	Маршрутизація в React
Vite	Інструмент збірки frontend
Mock API	Імітація серверних відповідей
Контейнер (тара)	Одиниця зберігання товару
Замовлення	Набір позицій для відбору
Видача у візок	Переміщення товару у візок
Пакування	Формування коробок
Реєстр нестач	Облік нестач
ER-модель	Модель сутностей і зв’язків
UML	Мова моделювання
Use Case	Сценарій взаємодії
CORS	Політика доступу між доменами
Environment variables	Змінні оточення
Rehydrate	Відновлення стану з пам’яті