НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

«На правах рукопису»
УДК _004.62_

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Інтегровані інформаційні системи»

зі спеціальності 126 «Інформаційні системи та технології»

на тему: «Інформаційна система управління складською логістикою з підтримкою офлайн-доступу та аналітики»

Виконав:
студент 2 курсу, групи ІА-42мп
Бойко Нікіта Олександрович _____

Керівник:
доцент каф. ІСТ, к.т.н.
Амонс Олександр Анатолійович _____

Рецензент:
доцент каф. ІПІ, д.т.н.
Крамар Юлія Михайлівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти — другий (магістерський)

Спеціальність — 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інтегровані інформаційні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Бойку Нікіті Олександровичу

1. Тема дисертації «Інформаційна система управління складською логістикою з підтримкою офлайн-доступу та аналітики»,науковий керівник дисертації Амонс Олександр Анатолійович, к.т.н., доц.,затверджені наказом по університету від «06» 11 2025 р. № 4841-с

2. Термін подання студентом дисертації «15»12 2025 р.

3. Об’єкт дослідження: забезпечення безперебійної роботи складських логістичних вузлів у розподіленому середовищі з можливими втратами мережевого з’єднання.

4. Вихідні дані: система, яка буде підтримувати безперебійну роботу складської логістики із забезпеченням основного функціоналу в режимі офлайн для користувачів у межах від 100 до 10000 одночасно; проста в опануванні та користуванні.

5. Перелік завдань, які потрібно розробити: провести аналіз існуючих рішень, зпроектувати архітектуру системи, реалізувати серверну та клієнтську частини, провести тестування, підготувати текстову та графічну частини пояснювальної записки.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: діаграма варіантів використання, діаграма компонентів, ER-діаграма, діаграма класів, діаграма послідовності, діаграма розгортання.

7. Орієнтовний перелік публікацій: не заплановано.

8. Дата видачі завдання 01.09.2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Аналіз предметної області	01.09.2025 – 10.09.2025	
2.	Аналіз існуючих рішень	11.09.2025 – 20.09.2025	
3.	Формування обмежень та вимог до системи	20.09.2025 – 30.09.2025	
4.	Дослідження методів реалізації	01.10.2025 – 05.10.2025	
5.	Розробка архітектури продукту	06.10.2025 – 16.10.2025	
6.	Розробка програмного забезпечення	17.10.2025 – 18.11.2025	
7.	Аналіз розробленого рішення	19.11.2025 – 22.11.2025	
8.	Оформлення пояснювальної записки	23.11.2025 – 03.12.2025	

Студент Нікіта БОЙКО

Науковий керівник Олександр АМОНС

РЕФЕРАТ

Інформаційна система управління складською логістикою з підтримкою офлайн-доступу та аналітики: 160 с., 22 табл., 34 рис., 8 дод., 26 джерел.

ОФЛАЙН-ДОСТУП, СИНХРОНІЗАЦІЯ ДАНИХ, СКЛАДСЬКА ЛОГІСТИКА, JAVA, SPRING BOOT, REACT, POSTGRESQL, REDIS, REST API.

Актуальність теми зумовлена необхідністю забезпечення працездатності складських логістичних вузлів в умовах нестабільного або відсутнього мережевого з'єднання. Аналіз існуючих рішень, виявив обмежені можливості щодо автономної роботи та синхронізації даних у розподіленому середовищі, що обґрунтовує доцільність розробки спеціалізованої системи з офлайн-підтримкою роботи критичного функціоналу.

Метою роботи є створення ефективної інформаційної системи управління складською логістикою з підтримкою автономного режиму роботи основного функціоналу та синхронізації даних. Для досягнення мети вирішено наступні завдання: проаналізовано сучасні логістичні рішення, спроєктовано архітектуру розподіленої системи, реалізовано механізми офлайн-роботи та синхронізації даних, а також розроблено інтуїтивний інтерфейс користувача для роботи з існуючим функціоналом.

Об'єктом дослідження є процес забезпечення безперебійної роботи складських логістичних вузлів у розподіленому середовищі, а предметом — методи та програмні засоби їх автоматизації з підтримкою офлайн-доступу та синхронізації даних. У роботі застосовано методи системного аналізу, проектування розподілених інформаційних систем, об'єктно-орієнтоване моделювання та принципи програмної інженерії.

Практичне значення роботи підтверджується створенням програмного продукту, який може застосовуватись у логістичних центрах та дистрибуційних мережах.

ABSTRACT

Information system for warehouse logistics management with offline access and analytics support: 160 p., 22 tab., 34 draw., 8 app., 26 sources. OFFLINE ACCESS, DATA SYNCHRONIZATION, WAREHOUSE LOGISTICS, JAVA, SPRING BOOT, REACT, POSTGRESQL, REDIS, REST API.

The relevance of the topic is driven by the need to ensure the stable operation of warehouse logistics nodes under conditions of unstable or missing network connectivity. Analysis of existing systems revealed limited capabilities for autonomous operation and reliable data synchronization in distributed environments, which justifies the development of a new solution with offline support for critical warehouse processes.

The aim of the work is to create an effective warehouse logistics management system with support for offline execution of core functionality and deferred synchronization. To achieve this aim, the following tasks were completed: existing logistics solutions were analyzed, a distributed system architecture was developed, offline-operation and synchronization mechanisms were implemented, and an intuitive user interface for warehouse operations was created.

The object of the research is the process of ensuring uninterrupted functioning of warehouse logistics nodes in a distributed environment, and the subject is the methods and tools for their automation with offline access and data synchronization. The work employs methods of system analysis, distributed systems design, object-oriented modeling, and software engineering.

The practical significance of the work is confirmed by the creation of a software product that can be used in logistics centers and distribution networks.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

БД — база даних — організоване сховище інформації, структурованої визначеним чином

СУБД — система управління базою даних — програмне забезпечення для роботи з базами даних

API — Application Programming Interface — набір правил і контрактів для взаємодії між програмними компонентами.

CRUD — Create, Read, Update, Delete — базові операції над даними в інформаційних системах

CSV — Comma Separated Values — формат структурованого тексту зі значеннями, розділеними комами

DAO — Data Access Object — шаблон проектування, що інкапсулює роботу зі сховищем даних

DOM — Document Object Model — ієрархічне представлення структури документа у вигляді дерева

DTO — Data Transfer Object — об’єкт для передачі даних між процесами чи шарами системи

HTTP — HyperText Transfer Protocol — протокол обміну даними між клієнтом і сервером

HTTPS — HyperText Transfer Protocol Secure — захищена версія HTTP з шифруванням

IBAN — International Bank Account Number — це міжнародний формат банківського рахунку, який використовується для спрощення та стандартизації банківських переказів між країнами.

IDB — IndexedDB — вбудоване сховище браузера для локального зберігання даних

JPA — Java Persistence API — стандарт доступу до реляційних баз даних через ORM

JSON — JavaScript Object Notation — текстовий формат обміну

структурованими даними

JWT — JSON Web Token — формат токенів для аутентифікації та авторизації у вебсистемах

ORM — Object-Relational Mapping — технологія відображення об’єктів у реляційні таблиці

REST — Representational State Transfer — архітектурний стиль побудови вебсервісів

SaaS — Software as a Service — це хмарна модель доставки програмного забезпечення, яка дозволяє користувачам отримувати доступ до програм через Інтернет за підпискою, замість того, щоб встановлювати їх локально на свій комп'ютер

SHA-256 — Secure Hash Algorithm 256-bit — алгоритм криптографічного хешування даних

SQL — Structured Query Language — мова формування запитів до реляційних баз даних

SWIFT — Society for Worldwide Interbank Financial Telecommunications — це міжнародна система для безпечного обміну фінансовою інформацією та здійснення платежів між банками та фінансовими установами по всьому світу.

TTL — Time-to-Live — максимальний термін існування пакета даних або час, протягом якого запис DNS вважається актуальним.

UI — User Interface — інтерфейс користувача

UX — User Experience — досвід взаємодії користувача із системою

XML — eXtensible Markup Language — текстовий формат для зберігання та передачі структурованих даних

ВСТУП

Упродовж розвитку промисловості та торгівлі процеси зберігання, обліку та переміщення матеріальних ресурсів постійно ускладнювалися, що вимагало впровадження нових підходів до управління складською логістикою. Перші складські приміщення виконували лише функцію збереження запасів, проте зі збільшенням обсягів виробництва та розширенням товарообігу виникла потреба у формалізації та систематизації логістичних операцій. Це спричинило появу методик обліку, стандартів інвентаризації та процедур контролю руху товарів.
З часом логістичні процеси стали невід'ємною частиною усіх економічних систем.

Для підприємств з великими товарними потоками склад став ключовим логістичним вузлом, що забезпечує ефективну взаємодію між постачанням, виробництвом та дистрибуцією. Сучасний склад виконує комплекс взаємопов’язаних операцій: приймання товарів, розміщення у зонах зберігання, облік залишків, внутрішні переміщення, комплектація замовлень та відвантаження.

Кожна з цих операцій вимагає точності, швидкості та узгодженості, що унеможливлює їх ефективне виконання без належної автоматизації. У таких умовах впровадження сучасних інформаційних систем стає критичним інструментом для підвищення точності операцій, оптимізації ресурсів та забезпечення прозорості всіх етапів логістичного процесу.

Зі зростанням кількості номенклатури, складності логістичних процесів та потребою в оперативному контролі з’явилися спеціалізовані інформаційні системи управління складами. Вони дозволяють централізовано обробляти дані, контролювати рух товарів, зменшувати кількість помилок та оптимізувати навантаження на персонал. Однак у практиці багатьох підприємств зберігається проблема залежності роботи WMS-систем від стабільного мережевого з’єднання.

У реальних умовах роботи з логістикою трапляються ситуації, коли доступ до мережі є нестабільним або повністю відсутнім. Це характерно для комплексних логістичних процесів, віддалених виробничих зон, мобільних логістичних груп чи

складських комплексів зі слабкою інфраструктурою. Втрата з’єднання може призвести до зупинки критичних операцій, неможливості виконання обліку, накопичення помилок та спотворення даних після відновлення роботи. Така залежність від онлайн-режиму формує потребу у створенні систем, здатних забезпечувати автономне виконання основного функціоналу та коректну синхронізацію інформації після відновлення мережі.

Саме тому актуальною стає розробка інформаційної системи управління складською логістикою з підтримкою офлайн-доступу, яка дозволить забезпечити безперервність логістичних процесів, зменшити вплив технічних обмежень на діяльність персоналу та підвищити надійність операцій з товарами.

Метою даної магістерської дисертації є створення інформаційної системи для забезпечення автономної роботи складської логістики та підтримки основних операцій у разі нестабільного або відсутнього мережевого з’єднання. Для цього потрібно вирішити наступні завдання:

а) провести аналіз існуючих рішень у сфері автоматизації складських процесів;

б) визначити функціональні вимоги системи, ролі користувачів та сценарії їх взаємодії;

в) виконати вибір засобів розробки з урахуванням вимог до офлайн-режиму;

г) визначити та провести проектування архітектури системи, механізмів синхронізації та бази даних;

д) реалізувати програмний продукт.

Практичне значення одержаних результатів полягає у створенні програмного продукту для автоматизації складської логістики, що включає:

а) реалізацію автономної роботи та збереження операцій у режимі офлайн;

б) систему управління складськими операціями та обліком товарів;

в) механізми синхронізації даних після відновлення мережевого з’єднання.

На захист виносяться наступні положення:

а) архітектурне рішення інформаційної системи управління складською логістикою, що відрізняється підтримкою автономного виконання критичних

операцій;

б) спосіб організації роботи складських користувачів в умовах нестабільного з’єднання із забезпеченням коректної синхронізації результатів операцій.

Структура роботи. Магістерська дисертація складається з наступних розділів: вступ, п’ять основних розділів, висновки, список використаних джерел із 26 найменувань. Графічна частина містить 8 креслеників формату А3. Загальний обсяг роботи становить 160 сторінок.

1 ОПИС ТА ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Опис та дослідження предметної області є одним з головних етапів у процесі розробки інформаційної системи, оскільки саме на цьому етапі формується цілісне розуміння специфіки та особливостей середовища функціонування, а також визначаються вимоги до автоматизації взаємодії об’єктів, процесів та явищ.

Предметна область складської логістики охоплює широкий спектр операцій, пов’язаних із прийманням, зберіганням, обліком, внутрішнім переміщенням та відвантаженням матеріальних ресурсів. Ці операції є невід’ємною частиною логістичних ланцюгів постачання та суттєво впливають на ефективність діяльності підприємства.

Сучасні складські комплекси характеризуються високим рівнем динамічності та обсягами бізнес-процесів, що зумовлює необхідність точності обліку та оперативності виконання логістичних операцій. Водночас на ефективність функціонування складської системи значною мірою впливають зовнішні та внутрішні фактори: нестабільність електропостачання та зв’язку, особливості розташування складських приміщень, щільність робочого графіка, а також навантаження на логістичні вузли. Це створює потребу у використанні інформаційних систем, які забезпечують надійну роботу навіть за умов часткової або повної відсутності зв’язку.

Детальне дослідження предметної області дозволяє виявити обмеження, характерні проблеми та вимоги, що висуваються до інформаційних систем складської логістики, зокрема до їх продуктивності, масштабованості, зручності використання та здатності працювати в автономному режимі. Отримані результати є базою для розробки функціональних і нефункціональних вимог до майбутнього програмного забезпечення, а також вибору інструментів реалізації та обґрунтування архітектурних рішень.

1.1 Аналіз основних логістичних процесів у логістичних вузлах

Основні процеси, притаманні логістичним вузлам, охоплюють цикл операцій, пов’язаних із обігом товарів. Ефективність забезпечення виконання цих процесів безпосередньо впливає на швидкість обробки замовлень, точність обліку товарів та стабільність роботи всієї системи. Здебільшого, логістичні вузли виконують функцію головного елементу інфраструктури у ланцюгу постачання, забезпечуючи узгодження матеріальних потоків між постачальниками, покупцями, різними підприємствами та іншими клієнтами, що користуються послугами складського зберігання.

Організація приймання товарів переважно потребує перевірки відповідності фактичних поставок супровідним документам, оцінку стану товару та його реєстрацію у обліковій системі. Цей етап є критично важливим, оскільки помилки у реєстрації товару призводять до втрати відповідності існуючої у системі інформації з реальністю, що негативно впливає на інвентаризацію, планування роботи з залишками та коректність функціонування системи як такої.

Згодом товари розміщуються відповідно до стандартів цього логістичного вузла, на основі ваги, розмірів, типу чи ціни, тощо. Цей процес включає визначення відповідних секцій зберігання та оновлення інформації про фізичне місцезнаходження товарів у системі. Від ефективності та точності цього процесу залежить швидкість подальших операцій із комплектації замовлень та внутрішніх переміщень.

Важливим компонентом логістичних процесів є управління запасами, що передбачає постійне оновлення даних щодо кількості та стану товарів. Для підтримання точності обліку проводяться планові та позапланові інвентаризації, результати яких необхідно своєчасно відображати в інформаційній системі.

Внутрішньоскладські переміщення виконуються з метою оптимізації використання складських площ, підготовки товарів до комплектації або переміщення між зонами обробки. До таких операцій належать перенесення товарів, їх пересортування та підготовка до відвантаження. Ці дії потребують

чіткого документування для запобігання втратам та помилкам.

Комплектація замовлень є одним із найбільш важливих та непростих процесів, який передбачає підбір товарів відповідно до специфікацій замовлення, перевірку їх стану та підготовку до відправлення. Якість та швидкість комплектації безпосередньо впливає на цілісність товарів та рівень якості обслуговування клієнтів.

Фінальним етапом є відвантаження, що потребує підготовки до транспортування, оформлення відповідної документації та передачу товарів до служби доставки. Цей процес є завершальним і повинен бути налаштований надійно з метою мінімізації ризиків у логістиці.

Аналіз логістичних процесів дозволяє визначити найбільш важливий функціонал, що залежить від забезпечення коректного оновлення даних, а відповідності наявної інформації реальному становищу. Це формує підґрунтя для подальшого проектування інформаційної системи, яка повинна забезпечити автономну роботу та гарантувати коректність синхронізації даних після відновлення мережевого доступу.

1.2 Аналіз особливостей управління складськими операціями

Ефективне керування складськими процесами може бути досягнуте з залученням різних методів та інструментів, що забезпечують координоване функціонування основних логістичних процесів. Якщо діяльність складського підприємства зосереджена на виконанні дій пов’язаних із обробкою товарів, то управлінський рівень забезпечує контроль, планування, моніторинг та оптимізацію цих дій у межах логістичного вузла або цілого підприємства. Технологічні можливості для якісного керування складськими операціями є визначальним фактором у підтриманні стабільності роботи логістичної інфраструктури та досягнення необхідної продуктивності.

Однією з ключових складових ефективного керування складом є забезпечення актуальності інформації в системі. Організаційні рішення, пов’язані

з розподілом робочого навантаження між працівниками складу, визначенням пріоритетності операцій або плануванням переміщення товарів, повинні базуватися виключно на достовірних даних.

Будь-які затримки чи неточності неминуче спричиняють помилки в роботі системи. Це може призвести до проблем із розміщенням товарів, неправильного розподілу ресурсів або дублювання операцій — усе це реально уникнути завдяки гарантуванню узгодженості інформації. Також не менш важливим аспектом керування є контроль виконання складських операцій, що охоплює відстеження прогресу, внесення змін до наявних даних або створення нових записів.

Розроблена система повинна забезпечувати можливість контролю діяльності робочого персоналу, визначення проблемних місць, а також отримання інформації про критичні ситуації, такі як нестача місця для зберігання товарів, затримка у виконанні замовлень, закінчення запасів, затримки у обробці запланованих відправлень, тощо.

Окрему увагу в управлінні складом займає планування операцій. Це включає визначення оптимального порядку виконання завдань, прогнозування навантаження на певні зони складу та планування використання технічних засобів. Через динамічний характер роботи з логістикою це планування повинно бути адаптивним, із можливістю коригування дій відповідно до змін у ситуації.

Особливістю управління складськими операціями у розподіленому середовищі є залежність усього функціоналу від доступності й достовірності даних. У ситуаціях, коли логістичний вузол працює в умовах нестабільного або відсутнього мережевого з’єднання, традиційні методи керування втрачають ефективність, оскільки дані не можуть бути своєчасно оновлені та синхронізовані. Це створює ризики розбіжностей у стані запасів, втрати інформації чи виникнення невідповідностей між фактичними та системними даними. За цієї причини система повинна передбачити надійний механізм обробки даних під час офлайн-сесії.

На підприємстві роботи зі складом логістичні операції зазвичай передбачають взаємодії між різними користувачами: операторами, комірниками,

логістами, менеджерами зміни тощо. Кожна роль має ряд обов’язків і чітко визначений набір доступу до потрібної їм інформації та функціоналу. Логістична система повинна забезпечувати чітке розмежування прав доступу, індивідуальні сценарії роботи та контроль за виконанням завдань та мати в наявності зрозумілі інструменти керування, що напряму впливає на точність операцій та ефективність використання.

Аналіз особливостей управління складськими операціями свідчить про необхідність застосування інформаційних систем, здатних забезпечувати своєчасне оновлення даних, підтримувати автономну роботу та узгоджувати результати операцій при відновленні доступу до мережі. Це визначає базові вимоги до архітектури майбутньої системи та формує підґрунтя для подальшого проектування функціональних і технічних компонентів.

1.3 Аналіз вимог до забезпечення автономності функціонування систем складської логістики

Автономність функціонування інформаційних систем складської логістики вимагає врахування низки технічних і організаційних факторів, які визначають здатність системи працювати без безпосереднього доступу до мережевих ресурсів. У процесі організації роботи складу часто виникають ситуації, коли підключення до центрального сервера є нестабільним або тимчасово відсутнім. У таких умовах система повинна залишатися працездатною, забезпечуючи можливість користувачам виконувати базові операції та отримувати доступ до всіх існуючих та необхідних даних.

Однією з ключових вимог до автономного режиму є наявність локальних механізмів збереження інформації, які дозволяють фіксувати результати операцій без необхідності негайної передачі даних на сервер. Такі механізми повинні гарантувати цілісність даних, підтримувати їх доступність та забезпечувати захист від випадкових втрат. Це особливо важливо для операцій, що виконуються у зонах зі слабким сигналом або при використанні мобільних робочих станцій.

Ще одним аспектом автономності є можливість обробки накопичених дій після відновлення з’єднання. Система повинна підтримувати формування черги операцій, визначати їх пріоритети та виконувати синхронізацію у правильній послідовності. При цьому важливо реалізувати механізми уникнення дублювання та суперечності у даних, а результати роботи залишалися узгодженими. Наявність перевірок, що виявляють конфліктні зміни, дозволяє уникнути помилок, які можуть виникнути під час співставлення локальних і серверних даних.

Для забезпечення автономності у роботі критичного функціоналу під час відсутності з’єднання необхідно мінімізувати залежність користувача від централізованої інфраструктури. Інтерфейс системи має надавати інформацію про доступний функціонал та існуючі дані у разі відсутності мережі, а також дозволяти виконання операцій, які є критичними для роботи складу. Це включає можливість реєстрації руху товарів, фіксації залишків, перегляду інформації про позиції та виконання внутрішніх задач. Система повинна передбачити можливість збереження реальної інформації за час роботи з доступом до серверу у локальному сховищі на стороні клієнта, що є гарантією існування цих даних на стороні сервера. Також варто передбачити унікальність та незмінюваність ідентифікаторів записів у базі даних для забезпечення узгодженості.

Окремим важливим аспектом у реалізації є вимоги до безпеки локально збереженої інформації. Оскільки частина даних тимчасово не надходить на сервер, система повинна забезпечувати її захист на рівні пристрою користувача. До цього належать шифрування, контроль доступу, автентифікація та періодичні перевірки цілісності даних.

Аналіз вимог для реалізації автономної роботи системи свідчить, що дане програмне забезпечення зобов’язане поєднувати ресурси локального середовища, механізми синхронізації та засоби контролю достовірності даних. Лише дотримання цих принципів дозволить реалізувати надійну підтримку безперервності найбільш важливих операцій на складі, за рахунок цього також буде досягнуто зменшення залежності від стабільності мережевої інфраструктури.

Висновки до розділу 1

У рамках цього розділу було розглянуто основні елементи обраної предметної області логістичних процесів з залученням складського зберігання. Було проаналізовано головні процеси, що притаманні роботі з логістичними вузлами, також було виявлено їх залежність від надійності та швидкості обробки даних. Було досліджено особливості управління логістичними операціями на складі, зокрема вимоги до контролю існуючих процесів, координації та планування майбутніх дій робочого персоналу.

Окрему увагу приділено вимогам до реалізації задля забезпечення автономності роботи інформаційної системи, що дозволить безперервність у виконанні головних логістичних операцій у разі нестабільного або відсутнього мережевого з’єднання. Здобуті результати формують базу для подальшого проектування інформаційної системи та визначення її функціональних і технічних вимог.

2 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Розробка інформаційної системи автоматизації роботи зі складською логістикою вимагає детального дослідження відповідних програмних продуктів у даній області. Це необхідно для визначення актуальних підходів та напрямків розвитку систем управління складом та роботи з логістикою з метою використання перевірених та надійних практик, а також уникання існуючих проблем та помилок.

Окремо необхідно приділити увагу дослідженню функціональних можливостей існуючих рішень, рушіїв, оточень та стратегій, що застосовуються для оптимізації ключових процесів у таких системах. Одним з найбільш важливих аспектів вартих уваги є розгляд потенційних архітектурних рішень, оскільки вони впливають на здатність системи бути масштабованою, ефективною, адаптивною та надійною незалежно від варіантів використання.

Значний вплив мають результати аналізу функціоналу розробленого для забезпечення підтримки автономного режиму роботи у вже існуючих системах, це дозволить запозичити найбільш цінні та перевірені часом практики для інтеграції у розробленому програмному забезпеченні з метою найбільш ефективної реалізації.

У більшості існуючих систем розрахунок ведеться на функціонування в режимі постійного підключення до інтернету та зв’язку, що є прийнятним, якщо сервер та клієнт розміщені у стабільній мережевій інфраструктурі, але відсутність логіки роботи в умовах нестійкого або відсутнього зв’язку може спричиняти відчутні проблеми для підприємства та його клієнтської бази. Для запобігання цьому зазвичай використовують механізми локального збереження та синхронізації даних після відновлення підключення, що необхідно врахувати та передбачити реалізацію цього функціоналу при проектуванні нової системи.

Аналіз існуючих рішень дозволить визначити слабкі та сильні технічні аспекти, а також виокремити вимоги до інформаційної системи. Результати проведеного аналізу будуть використані у якості підґрунтя для вибору

інструментів та технологій розробки, формування архітектури застосунку і впровадження логіки забезпечення автономної роботи з критичним функціоналом, що допоможе гарантувати точність даних та стійкість до змін у зовнішньому середовищі, що можуть створити проблеми для злагодженого функціонування.

2.1 Аналіз корпоративних WMS-систем

Корпоративні WMS-системи є найбільш комплексними рішеннями, призначеними для автоматизації та оптимізації логістичних і складських процесів у великих логістичних центрах і підприємствах із розгалуженою інфраструктурою. Такі системи зазвичай характеризуються високою масштабованістю, широким набором інтеграцій зі сторонніми сервісами, гнучкістю конфігурації та можливістю підтримувати роботу у режимі великого інформаційного навантаження.

Основною метою зазвичай виступає централізоване управління складами, забезпечення високої точності обліку та мінімізація помилок у процесах зберігання й переміщення товарів. Водночас корпоративні рішення орієнтовані на роботу у стабільному мережевому середовищі, через що переважна більшість не враховує необхідність підтримки автономного режиму роботи.

SAP Extended Warehouse Management — це корпоративна система управління складськими операціями, що входить до екосистеми SAP та призначена для автоматизації складських процесів у великих логістичних комплексах із високим рівнем складності.

Функціональні можливості:

а) управління складськими процесами та завданнями;

б) 3D-візуалізація складських площ;

в) контроль продуктивності персоналу;

г) моніторинг ефективності підприємства;

д) автоматичне управління складськими потоками.

Переваги**:**

а) гнучке налаштування під специфіку складу;

б) більш проста інтеграція, ніж у SAP;

в) зручні інструменти для контролю персоналу;

г) підтримка складних зональних схем.

Недоліки:

а) залежність від стабільного підключення до серверної частини;

б) офлайн-функціонал реалізований частково;

в) висока вартість ліцензування;

г) складність підтримки у середовищах зі змінним доступом до мережі.

Oracle Warehouse Management Cloud — це хмарна система управління складом, яка забезпечує централізоване виконання та моніторинг логістичних операцій у реальному часі через інфраструктуру Oracle Cloud.

Функціональні можливості:

а) централізований контроль складських операцій у хмарі;

б) управління товарними потоками в реальному часі;

в) автоматизація документообігу та відвантажень;

г) REST API для інтеграцій;

д) аналітичні звіти та планування.

Переваги**:**

а) швидке розгортання у хмарному середовищі;

б) хороша масштабованість;

в) висока доступність сервісів у глобальній інфраструктурі Oracle;

г) зручність для розподілених мереж складських центрів.

Недоліки:

а) повна залежність від Інтернету;

б) обмежені можливості офлайн-режиму;

в) критичні процеси перериваються при втраті мережі;

г) складнощі із роботою у великогабаритних складських приміщеннях зі слабким покриттям.

Infor WMS — це модульна система управління складськими процесами, що поєднує операційний контроль, планування, управління персоналом та інструменти аналітики для оптимізації роботи складських центрів.

Функціональні можливості:

а) управління складськими процесами та завданнями;

б) 3D-візуалізація складських площ;

в) контроль продуктивності персоналу;

г) моніторинг KPI;

д) автоматичне управління складськими потоками.

Переваги:

а) гнучке налаштування під специфіку складу;

б) більш проста інтеграція, ніж у SAP;

в) зручні інструменти для контролю персоналу;

г) підтримка складних зональних схем.

Недоліки:

а) залежність від стабільного підключення до серверної частини;

б) офлайн-функціонал реалізований частково;

в) висока вартість ліцензування;

г) складність підтримки у середовищах зі змінним доступом до мережі.

Аналіз корпоративних WMS показує, що ці системи мають потужний функціонал та здатні підтримувати широкі логістичні процеси, проте їх архітектура передбачає роботу у стабільно підключеному середовищі. Для задачі створення системи з автономною підтримкою ці рішення є цінним джерелом архітектурних і функціональних ідей, але не відповідають вимогам щодо офлайн-режиму.

2.2 Аналіз комерційних та хмарних рішень для середнього бізнесу

Існуючі комерційні та хмарні WMS-рішення, орієнтовані на малий і середній бізнес. Здебільшого, вони пропонують спрощений підхід до

автоматизації складських операцій, так як їх основна мета це забезпечення швидкого та легкого старту роботи з системою, що вимагає простоти впровадження і управління запасами та замовленнями без складної технічної інфраструктури. Такі системи часто постачаються за моделлю SaaS, інтегруються з різними маркетплейсами та системами електронної комерції. Також вони орієнтовані на максимальну зручність інтерфейсу і низький поріг технічних знань необхідних для коректного користування. Водночас більшість з цих систем лімітовані за гнучкістю налаштувань, не підтримують розширення існуючих сценаріїв залежно від індивідуальних потреб та ігнорують реалізацію офлайн-функціонування.

Odoo Inventory — модуль системи Odoo для управління складськими операціями.

Функціональні можливості:

а) контроль переміщень товарів та формування маршрутів;

б) інтеграція з продажами, закупівлями, бухгалтерією та виробництвом;

в) формування складських документів та інвентаризацій;

г) розширення функціональності через додаткові модулі;

д) можливість локального або хмарного розгортання.

Переваги:

а) модульність та масштабованість;

б) широка екосистема інтеграцій;

в) гнучкість налаштувань під конкретні бізнес-процеси;

г) можливість роботи на власній інфраструктурі.

Недоліки:

а) потреба у спеціалістах для конфігурації;

б) відсутність повної автономності клієнта;

в) складні сценарії вимагають додаткових модулів;

г) критичні операції залежать від серверної частини.

Fishbowl Inventory — система обліку та управління запасами для виробничих і дистрибуційних підприємств середнього масштабу.

Функціональні можливості:

а) інвентаризація та контроль товарних переміщень;

б) підтримка кількох складів;

в) базове виробниче планування;

г) інтеграція з QuickBooks та іншими ERP;

д) управління замовленнями та формування виробничих завдань.

Переваги:

а) можливість локального розгортання;

б) мінімальні вимоги до інфраструктури;

в) практичні інструменти для малого виробництва;

г) простий для освоєння функціонал.

Недоліки:

а) обмежена підтримка складних логістичних схем;

б) обмежена масштабованість;

в) відсутність механізмів автономної роботи;

г) залежність від центральної бази даних.

Unleashed — хмарна система управління запасами для виробничих та торгових компаній

Функціональні можливості:

а) детальний контроль запасів з відстеженням партій, серійних номерів та термінів придатності;

б) управління продажами, закупівлями та внутрішніми переміщеннями;

в) підтримка базових виробничих операцій (BOM, складання/розбирання товарів);

г) інтеграція з Xero, QuickBooks, Amazon, Shopify та POS-системами;

д) формування операційних і фінансових звітів у режимі реального часу.

Переваги:

а) можливість роботи з виробничими сценаріями, на відміну від типових

SaaS WMS;

б) потужні інструменти аналітики запасів і рухів;

в) зручні механізми інтеграції з бухгалтерськими та торговельними платформами;

г) хмарна модель із мінімальними вимогами до інфраструктури компанії.

Недоліки:

а) відсутність підтримки складських зон і складних маршрутів переміщення товарів;

б) неможливість роботи при відсутності мережі, оскільки всі операції виконуються онлайн;

в) обмежена кастомізація та залежність від вбудованих бізнес-правил;

г) слабкі можливості масштабування у розподілених логістичних мережах.

Аналіз хмарних та комерційних рішень дозволив визначити, що такі системи значно спрощують роботу для кінцевого користувача та забезпечують швидку автоматизацію основних складських операцій. Однак їх архітектура орієнтована на стабільне мережеве середовище, а офлайн-функціонал реалізовано лише частково або не реалізовано.

Для розроблюваної інформаційної системи доцільно використати сильні сторони цих рішень, такі як зручність інтерфейсу, доступність та простоту інтеграцій. І також варто уникнути їх основних обмежень та недоліків, забезпечивши автономність головного функціоналу з відповідними архітектурними рішеннями, що передбачає синхронізацію та уникнення конфліктів через підтримку роботи у режимі нестабільного зв’язку.

2.3 Аналіз плагінів та розширень для платформ електронної комерції

У сегменті малого та середнього бізнесу для управління запасами та виконання складських операцій часто використовують різні плагіни та розширення, вбудовані в популярні платформи електронної комерції.

Ці рішення орієнтовані на швидку інтеграцію з інтернет-магазинами,

автоматизацію обліку запасів і просте управління замовленнями без залучення складної інфраструктури. Вони підходять для бізнесів з невеликим чи середнім обсягом товарів, коли достатньо базового функціоналу для обробки замовлень, координованого обліку та синхронізації запасів між каналами продажів.

WooCommerce Inventory Plugins — це плагін для WooCommerce, що додає до платформи функції управління запасами, контролю залишків, обліку складу та базового складського менеджменту прямо в межах магазину.

Функціональні можливості:

а) ведення запасів та автоматичне оновлення залишків при замовленнях;

б) просте управління запасами через інтерфейс адміністратора;

в) контроль наявності товарів перед підтвердженням замовлення;

г) ведення історії запасів і змін;

д) можливість інтеграції з додатковими плагінами для логістики або експорту/імпорту даних.

Переваги:

а) не потребує окремої WMS — все працює всередині WooCommerce;

б) бажане рішення для невеликих або нових магазинів з невеликими обсягами;

в) просте налаштування, не потребує технічних навичок;

г) мінімальні витрати на впровадження.

Недоліки:

а) обмежений набір функцій — підходить лише для простих сценаріїв;

б) слабка або відсутня підтримка розширеної складської логістики та зонального зберігання;

в) залежність від роботи інтернет-магазину — без підключення сайт може не працювати;

г) відсутність офлайн-режиму або механізмів автономної роботи.

Shopify Apps для управління запасами та логістикою — це набір додатків для Shopify, які розширюють функціонал магазину, додаючи можливості обліку запасів, управління замовленнями, відвантаженнями та інтеграцію з логістичними

провайдерами.

Функціональні можливості:

а) відстеження залишків і синхронізація між різними каналами продажів;

б) створення та управління замовленнями, чергами відвантажень;

в) інтеграція з кур’єрськими службами та експорт замовлень у формати доставки;

г) управління складами / запасами прямо з адмінки Shopify;

д) базова аналітика продажів та запасів.

Переваги:

а) швидка інтеграція із існуючим магазином Shopify;

б) зручний користувацький інтерфейс, дружній до користувачів без технічних навичок;

в) мінімальні початкові витрати;

г) автоматична синхронізація запасів при продажах.

Недоліки:

а) обмежені логістичні можливості — не підходить для складських комплексів з великою кількістю товарних позицій;

б) залежність від постійного підключення до інтернету та серверів Shopify;

в) відсутність гнучкої логіки для робіт зі складською логікою (зони, переміщення, виробництво);

г) обмежена масштабованість для компаній із розгалуженою логістичною мережею.

Magento Inventory Management Extensions — це розширення для Magento Adobe Commerce, що додають розширені інструменти управління запасами, підтримку кількох складів, маршрутів постачання та базової логістики в рамках e-commerce екосистеми.

Функціональні можливості:

а) підтримка кількісної складської моделі з визначенням джерел запасів;

б) автоматичний розподіл замовлень між складами залежно від наявності товару;

в) відстеження руху товарів та локальна інвентаризація;

г) інтеграція з кур’єрськими службами та службами фулфілменту;

д) формування звітів про продажі, запаси та залишки по складах.

Переваги:

а) гнучкі можливості конфігурації — підходить для магазинів із кількома складами;

б) тісна інтеграція з е-комерційною платформою Magento;

в) підтримка розподіленого виконання замовлень (multi-source fulfillment);

г) можливість розширення функціоналу через додаткові модулі та API.

Недоліки:

а) висока складність налаштування порівняно з іншими e-commerce платформами;

б) відсутність нативної підтримки повноцінного WMS-рівня (зональність, маршрути внутрішніх переміщень);

в) відсутність офлайн-режиму та залежність від центральної БД;

г) значне навантаження на сервер при великому обсязі замовлень або даних.

Плагіни та розширення для платформ електронної комерції — це зручні та швидкі рішення для малого та середнього бізнесу, які надають функціонал базового обліку запасів та прийому/обробки замовлень без потреби у складній інфраструктурі. Вони забезпечують швидкий старт, легку інтеграцію з каналами продажів та мінімальні витрати на впровадження.

Проте, ці рішення мають обмежений функціонал — вони не підходять для складських мереж із комплексним логістичним функціоналом, багаторівневими зонами зберігання або необхідністю автономної роботи.

Для розроблюваної інформаційної системи, доцільно врахувати їх простоту й інтеграційність, але не треба розраховувати виключно на це. Обов’язково потрібно забезпечити реалізацію повноцінної WMS із підтримкою офлайн-режиму, деталізованим обліком та масштабованістю.

Висновки до розділу 2

Аналіз існуючих рішень у сфері управління складською логістикою демонструє значну різноманітність підходів до автоматизації складських операцій залежно від масштабу підприємства, бізнес-потреб та технічних вимог. Корпоративні WMS-системи пропонують широкий функціонал, здатний підтримувати складні логістичні процеси великих підприємств, але їх архітектура орієнтована на роботу у стабільному мережевому середовищі та не передбачає повноцінної автономності хоча б основного функціоналу.

Комерційні та хмарні рішення для середнього бізнесу забезпечують швидке впровадження, простоту використання та наявність інтеграцій із торговими й фінансовими платформами, однак їх функціональні можливості обмежені стандартними сценаріями без можливості редагування, а залежність від постійного інтернет-з’єднання робить їх непридатними для роботи в умовах нестабільного зв’язку.

Плагіни та розширення для платформ електронної комерції є найменш функціонально складними, забезпечуючи лише базовий облік та синхронізацію замовлень, що конструктивно не дозволяє застосовувати їх у середовищах зі складною логістикою чи великими обсягами даних.

Отже, жоден із розглянутих класів систем не забезпечує необхідного балансу між гнучкістю, масштабованістю та можливістю автономної роботи. Це підтверджує актуальність розробки спеціалізованої інформаційної системи, яка імплементує переваги різних підходів — зручність і доступність, властиві комерційним рішенням, структурованість та логічну організацію корпоративних WMS, а також простоту інтеграції, характерну для рішень у сфері e-commerce.

Головною вимогою до цільової інформаційної системи є забезпечення стабільного автономного функціонування з гарантованою синхронізацією даних, що дозволить уникати збоїв і підтримувати безперервність логістичних процесів навіть у умовах нестабільного мережевого середовища.

3 АРХІТЕКТУРНЕ ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Архітектурне проектування є фундаментальним етапом циклу розробки будь-якого програмного забезпечення. Цей процес визначає структурну організацію, функціональну логіку та дозволяє зробити базу для реалізації продуктивної, масштабованої та надійної системи. У контексті розробки архітектурні рішення є визначальними, оскільки саме вони регламентують можливість автономного функціонування вузлів системи та забезпечують коректну синхронізацію даних у розподіленому середовищі на рівні з коректним функціонуванням системи як такої.

На даному етапі організовується побудова вимог до компонентів системи, визначаються ключові елементи, протоколи взаємодії, правила роботи з даними, алгоритми обробки даних, налаштування оточення та необхідні технології.

Раціонально зпроектована архітектура дозволяє реалізувати надійну інформаційну систему, що буде стійкою до зовнішнього впливу, високого навантаження, адаптовану до розширення функціоналу та гнучку у налаштуванні.

У межах цього розділу буде здійснено декомпозицію системи на концептуальному та логічному рівнях, визначено функціональні та технічні вимоги, описано інтерфейси взаємодії компонентів, обґрунтовано вибір технологій, а також буде проведено аналіз механізмів забезпечення повноцінної реалізації програмного забезпечення, що повністю буде відповідати визначеним очікування до готової реалізації.

3.1 Вимоги до системи

Вимоги до інформаційної системи управління складською логістикою першочергово визначають очікування до функціональних та технічних можливостей системи, потенційні обмеження, умови експлуатації та передбачають визначення стратегії для забезпечення працездатності цільового функціоналу за будь-яких умов.

Інформаційна система повинна забезпечити коректність, безперервність та надійність роботи як у регулярному режимі роботи, так і в автономному за наявних перешкод у мережевій, енергетичній чи програмній інфраструктурі. Дані вимоги засновані на аналізі предметної області, бізнес-процесів підприємства, а також особливостей роботи користувачів у умовах нестабільного або відсутнього інтернет-з’єднання.

Цільова інформаційна система управління складською логістикою з підтримкою офлайн-доступу та аналітики має забезпечити функціонал для роботи з основними логістичними процесами, складським зберіганням та реалізувати механізми відмовостійкості.

Функціональні вимоги:

а) реєстрації та керування користувачами, включаючи створення облікових записів, зміну профільних даних, призначення ролей, блокування та контроль активності;

б) авторизації та доступу до ресурсів на основі ролі користувача, забезпечуючи аутентифікацію, перевірку прав доступу, обмеження виконання операцій та захист даних з урахуванням політик безпеки;

в) керування складами та їх внутрішньою структурою, зокрема створенням та модифікацією складів, зон, секцій і комірок, підтримкою їх статусів та параметрів, а також організацією просторового розміщення товарів;

г) керування товарами та номенклатурою, включаючи створення і редагування товарних позицій, їх атрибутів, одиниць вимірювання та класифікаційних ознак;

д) обліку та контролю запасів, з можливістю перегляду залишків, їх переміщення між комірками та складами, фіксації змін і забезпечення цілісності складських даних;

е) виконання складських операцій, таких як приймання, відвантаження, переміщення, інвентаризація та коригування, із реєстрацією результатів та оновленням відповідних станів системи;

є) роботи з офлайн-чергою та механізмами синхронізації, що передбачає

локальне збереження операцій, їх відкладене виконання у разі відсутності мережі та подальше узгодження з сервером;

ж) перегляду аналітичних даних, включаючи агреговані показники, статистику рухів, історію операцій та формування звітів за діяльністю складу;

з) використання тегів та класифікацій, що забезпечують групування сутностей, швидкий пошук та удосконалення навігації в системі;

і) керування системними налаштуваннями, ролями та правами доступу, які визначають політики безпеки, параметри роботи та поведінку системи у цілому.

Нефункціональні вимоги:

а) вимоги до продуктивності, що передбачають швидку реакцію інтерфейсу, оптимальний час обробки запитів та здатність підтримувати одночасну роботу багатьох користувачів без суттєвих затримок;

б) вимоги до надійності, які забезпечують стабільне функціонування системи, відсутність втрати даних, збереження коректного стану операцій та відновлення роботи після збоїв або втрати з’єднання;

в) вимоги до безпеки, що включають захист даних користувачів, шифрування передавання інформації, контроль доступу за ролями, безпечне зберігання облікових даних та аудит ключових дій;

г) вимоги до масштабованості, які дозволяють розширювати систему шляхом збільшення обчислювальних ресурсів або додавання нових компонентів без змін у базовій архітектурі;

ґ) вимоги до сумісності, що передбачають коректну роботу на сучасних браузерах, відповідність стандартам вебтехнологій та можливість інтеграції з зовнішніми сервісами;

д) вимоги до доступності, які гарантують безперервну роботу системи, мінімізацію часу простою та можливість користування основними функціями у режимі обмеженого або відсутнього інтернет-з’єднання;

е) вимоги до зручності використання, що забезпечують інтуїтивний інтерфейс, логічну організацію функцій, візуальні підказки та мінімізацію кількості дій для виконання типових операцій;

є) вимоги до підтримуваності, що визначають легкість оновлення, супроводу та розширення системи, читабельність коду та доступність документації;

ж) вимоги до цілісності даних, які передбачають коректність збереження інформації, узгодженість між клієнтською та серверною частинами, а також захист від несанкціонованих змін.

Визначені вимоги становлять цілісну концептуальну основу для розробки цільової інформаційної системи. Охоплення критичних аспектів, таких як консистентність даних та доступність сервісів, дозволяє перейти до етапу системного архітектурного моделювання. Також специфікація залишається відкритою для декомпозиції та уточнення на подальших стадіях життєвого циклу розробки з урахуванням специфіки предметної області.

3.2 Сценарії використання системи

На основі визначених функціональних вимог до системи, сценарії використання наведені у додатку Б. Детально опишемо кожен прецедент, необхідні ролі, передумови дії та результати.

Вхід до системи здійснюється всіма ролями за умови наявності в користувача зареєстрованого облікового запису. Під час авторизації він вводить логін і пароль та підтверджує операцію. Після успішної перевірки облікових даних система надає доступ відповідно до призначеної ролі.

Вихід із системи виконується користувачами всіх ролей та потребує попередньої авторизації. Користувач ініціює завершення сеансу, після чого система коректно припиняє роботу поточного підключення. У результаті доступ до функціоналу блокується до наступної авторизації.

Перегляд транзакцій доступний усім ролям за умови авторизації та наявності відповідних записів у системі. Користувач отримує можливість переглядати перелік транзакцій, застосовувати фільтрування та відкривати деталізовані відомості. У підсумку йому надається актуальна інформація щодо

проведених операцій.

Створення транзакції виконується всіма ролями за умови авторизації та наявності фінансових рахунків. Користувач визначає тип транзакції, заповнює необхідні дані та підтверджує створення. У результаті формується новий запис, а у разі роботи в офлайн-режимі транзакція потрапляє до черги відкладених операцій.

Оновлення транзакції доступне всім ролям за умови авторизації та існування транзакції, що не перебуває у фінальному статусі. Користувач вносить зміни до параметрів та зберігає оновлені дані. Внаслідок цього інформація про транзакцію коригується відповідно до внесених змін.

Перегляд товарів здійснюється всіма ролями за умови авторизації та наявності товарних позицій. Користувач переглядає перелік товарів і пов’язані з ними атрибути. У результаті він отримує доступ до актуальної номенклатури.

Створення товару виконується всіма ролями за умови авторизації та існування відповідного продукту. Користувач заповнює атрибути, обирає продукт і підтверджує створення. У результаті формується нова товарна одиниця, а при офлайн-роботі вона додається до відкладеної черги.

Оновлення товару здійснюється всіма ролями за умови авторизації та наявності товарної позиції. Користувач редагує параметри товару та зберігає зміни. Внаслідок цього інформація про товар оновлюється.

Перегляд постачань доступний усім ролям за умови авторизації та існування відповідних записів. Користувач переглядає перелік постачань та їх основні параметри. У результаті надаються актуальні дані щодо надходжень.

Створення постачання виконується всіма ролями за умови авторизації та доступності товарів і складів. Користувач формує нове постачання та додає необхідні позиції. У результаті створюється відповідний запис, а при роботі офлайн — він потрапляє до черги синхронізації.

Оновлення постачання здійснюється всіма ролями за умови авторизації та існування не фіналізованого запису. Користувач редагує відповідні параметри і зберігає зміни. Внаслідок цього інформація про постачання оновлюється.

Перегляд груп товарів виконується всіма ролями за наявності авторизації та груп у системі. Користувач отримує можливість переглядати перелік груп і їхній склад. У результаті відображається структурована інформація щодо групування товарів.

Створення груп товарів доступне всім ролям за умови авторизації. Користувач формує нову групу. У результаті до системи додається новий елемент групування.

Оновлення груп товарів здійснюється всіма ролями за умови авторизації та існування груп. Користувач змінює параметри вибраної групи. Внаслідок цього оновлюються її характеристики.

Перегляд складів виконується всіма ролями після авторизації за умови існування складів. Користувач переглядає структуру доступних складів. У результаті відображається їхня актуальна конфігурація.

Створення складів можливе для ролі власника за наявності авторизації. Користувач створює нову складську одиницю. У системі з’являється відповідний запис, а при офлайн-роботі він додається до черги.

Оновлення складу виконується менеджером або власником за умови авторизації та існування записів. Користувач змінює параметри складу та зберігає результат. Внаслідок цього інформація оновлюється.

Перегляд користувачів доступний усім ролям за наявності авторизації та користувачів у системі. Користувач переглядає перелік усіх облікових записів. У результаті надається відповідна інформація.

Оновлення користувачів здійснюється менеджером або власником за умови існування відповідного запису. Користувач змінює ролі або параметри профілю. У результаті дані оновлюються.

Створення користувача виконується власником після авторизації. Він вводить необхідні дані нового облікового запису. У системі реєструється новий користувач.

Перегляд історії змін товарів доступний усім ролям за умови авторизації та наявності записів. Користувач переглядає журнал змін. У результаті

відображаються відомості про модифікації товарів.

Перегляд фінансової статистики здійснюється всіма ролями за умови авторизації та наявності фінансових даних. Користувач переглядає відповідні показники. У підсумку власник отримує аналітичну інформацію.

Перегляд статистики продажів доступний усім ролям за наявності даних про продажі. Користувач аналізує показники за товарами або періодами. У результаті система відображає відповідну статистику.

Перегляд тегів виконується всіма ролями після авторизації за умови існування тегів. Користувач переглядає їх перелік і прив’язки. Внаслідок цього він отримує необхідну інформацію.

Створення тегів здійснюється всіма ролями за наявності авторизації. Користувач формує новий тег. У системі з’являється новий запис, а у разі офлайн-роботи він додається до черги синхронізації.

Оновлення тегів виконують усі ролі за умови існування відповідного тегу. Користувач вносить зміни до його параметрів. У результаті інформація оновлюється.

Запуск обробки офлайн-черги доступний усім ролям за наявності авторизації, відкладених операцій та відновленого інтернет-з’єднання. Система здійснює синхронізацію створених офлайн сутностей та операцій. У підсумку всі валідні задачі фіксуються на сервері.

Перегляд секцій зберігання доступний усім ролям після авторизації та наявності відповідних елементів. Користувач переглядає зони, секції та комірки. Внаслідок цього відображається актуальна структура зберігання.

Створення секції зберігання здійснюється всіма ролями за наявності авторизації та існування складів. Користувач вводить параметри нової зони, секції або комірки та підтверджує створення. У результаті утворюється нова одиниця зберігання, а при офлайн-режимі вона додається до черги.

Оновлення секції зберігання виконується менеджером або власником за умови авторизації та існування відповідного запису. Користувач змінює параметри або статус секції. У підсумку дані оновлюються.

Перегляд фінансових рахунків доступний усім ролям за умови існування таких записів. Користувач переглядає параметри рахунків. У результаті система відображає актуальну інформацію.

Створення фінансового рахунку здійснюється всіма ролями після авторизації. Користувач вводить реквізити рахунку та підтверджує створення. Внаслідок цього рахунок додається до системи, а при офлайн-роботі — до черги.

Оновлення фінансового рахунку виконується всіма ролями за умови авторизації та існування відповідного запису. Користувач змінює параметри рахунку та зберігає результат. У підсумку оновлюється інформація про фінансовий рахунок.

3.3 Структурна схема системи

Структурна схема інформаційної системи відображає загальну організацію її компонентів, їх взаємодію та функціональне призначення. Система побудована за багаторівневим підходом і складається з клієнтської частини, серверної частини, підсистем зберігання даних, а також допоміжних сервісів, що забезпечують коректну роботу бізнес-процесів, синхронізацію даних та підтримку автономного режиму. Кожен із компонентів системи виконує визначену роль у забезпеченні її цілісного функціонування.

Клієнтська частина реалізує інтерфейс користувача, обробляє введені дані та відповідає за взаємодію з локальними ресурсами, включаючи механізми кешування та офлайн-чергу. Серверна частина забезпечує виконання бізнес-логіки, контроль доступу, обробку запитів і підтримку транзакційності операцій. Підсистема зберігання даних містить централізовану базу даних, у якій зберігаються всі сутності системи, а допоміжні сервіси, такі як модуль синхронізації, кешування, журналювання та аналітичні інструменти, сприяють підвищенню надійності, продуктивності та узгодженості роботи системи.

3.3.1 Перше наближення системи

Інформаційна система розробляється з акцентом на максимальну зручність та доступність для широкого кола користувачів. Це вимагає підтримки роботи на різних пристроях та операційних системах. Щоб задовольнити різноманітні потреби користувачів та забезпечити незалежність від конкретної платформи, було обрано універсальну архітектуру. Її базова структурна схема наведена на рисунку 3.1.

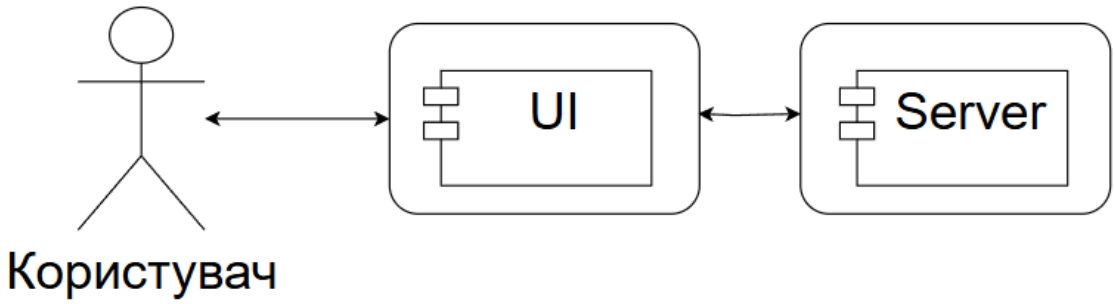


Рисунок 3.1 — Базова структура системи

Дану схему необхідно декомпозувати, застосувати принцип модульності, що передбачає чітке розмежування основної бізнес-логіки та рівня взаємодії з користувачем.

Такий підхід обумовлений необхідністю підтримки різноманітних каналів доступу до системи. Запропонована структура підлягає подальшій декомпозиції, проте на концептуальному рівні вона складається з трьох ключових компонентів: підсистеми роботи з даним, модуля бізнес-логіки та модуля, що надає інтерфейс для взаємодії з клієнтським застосунком. зображена на рисунку 3.2.

Цей поділ на окремі модулі гарантує, що система отримує чітку структурованість що полегшує внесення змін, підвищує надійність та забезпечує можливість використання різних інтерфейсів і сценаріїв доступу без зміни внутрішньої логіки, роблячи архітектуру значно гнучкішою та масштабованою

для подальшого розвитку.

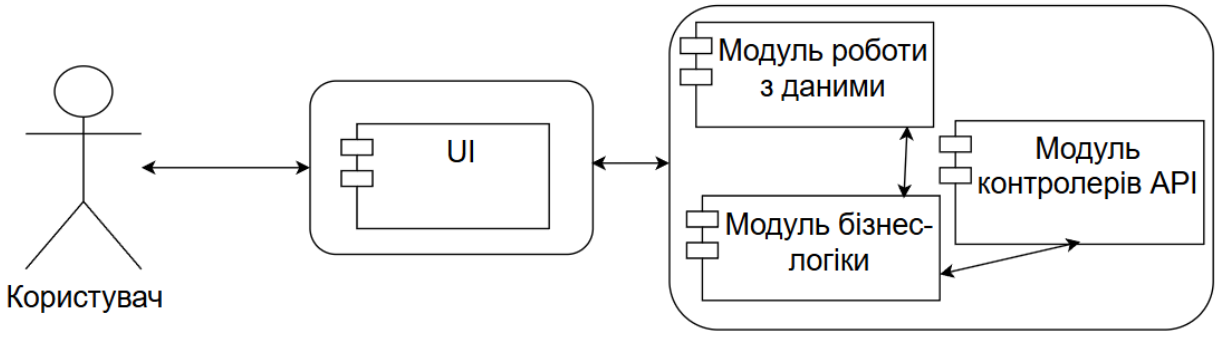


Рисунок 3.2 — Перша декомпозиція системи

Розглянемо модуль роботи з даними. Так як сховище даних може бути різним, варто звернутися до ORM [1] та DAO патернів. Це забезпечить гнучкість реалізації, так як у майбутньому під час розробки конкретної імплементації може виникнути потреба зміни сховища даних, або ж розширення існуючого логіки.

ORM дозволить працювати зі сховищем базуючись на об’єктах, які описують сутності, що дозволить не прив’язуватися до конкретної імплементації інструментів роботи з даними. Більшість ORM фреймворків мають власну реалізацію мови запитів, яка динамічно перетворюється під синтаксичні вимоги тієї чи іншої бази даних.

Також DAO дозволяє ізолювати усю логіку роботи зі сховищем та конвертацію збережених даних у відповідні моделі під потреби логіки застосунку в одному шарі, розділяючи зони відповідальності частин застосунку. Бізнес-логіка застосунку визначає набір функціоналу, що буде доступний користувачам застосунку, він використовує уже існуючу реалізацію роботи з даними для доступу до них та виконує перетворення залежно від задачі функції.

Щоб забезпечити комунікацію між сервером та клієнтською частиною треба звернутися до імплементації шару контролерів, що дозволяють зробити логіку отримання запитів ззовні та викликати виконання конкретного функціоналу. Цей модуль зроблено для роботи з серверною логікою. У випадку потреби таку систему можна легко адаптувати до інфраструктурних змін, або ж змін у логіці застосунку чи способах використання. Розбиття на модулі дозволяє виконувати

зміни виключно там де це потрібно.

Загальний принцип взаємодії буде базуватися на основі HTTP-запитів зображено на рисунку 3.3.

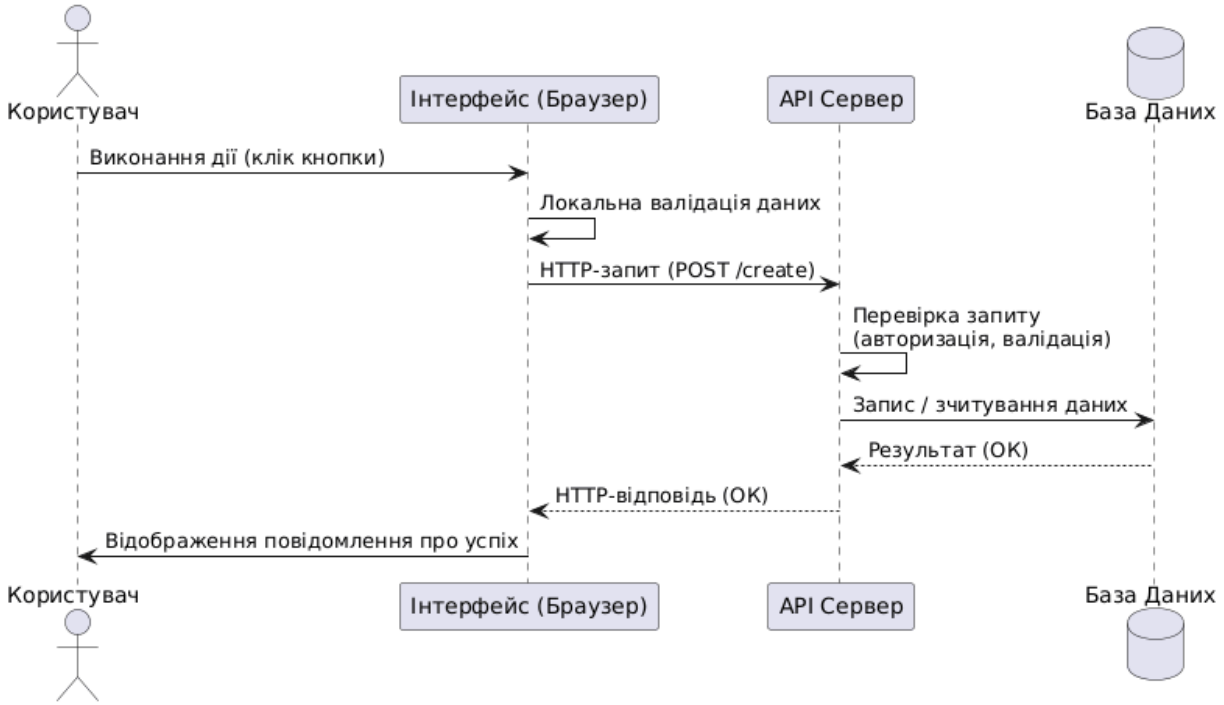


Рисунок 3.3 — Загальна комунікація

Результатом наближення є зформоване початкове бачення її структури та функціональності, окреслити основні складові та визначити їх роль у забезпеченні роботи всієї інформаційної системи. На цьому етапі встановлюється цілісне розуміння ключових процесів, взаємодії між компонентами та очікуваних результатів, що надалі слугує базисом для детального проектування, розроблення архітектури та уточнення вимог. Завдяки такому підходу забезпечується послідовність, логічність і обґрунтованість подальших етапів створення системи.

3.3.2 Архітектура клієнт-серверної взаємодії

На основі першого наближення розроблюваної інформаційної системи, було визначено її ключові функції. Це надає конструктивну базу для розгляду архітектурних рішень, що забезпечують інфраструктурні можливості до реалізації

застосунку, що задовольнить усі раніше окреслені функціональні та нефункціональні вимоги. Зокрема підтримку роботи з великою кількістю сутностей, забезпечення автономного режиму, синхронізацію даних та чіткий розподіл ролей користувачів, це визначає загальний напрям формування архітектури програмного забезпечення.

З огляду на потребу в централізованому управлінні даними та контролі доступу, клієнт-серверна модель була обрана як базовий архітектурний підхід. Така модель дозволяє забезпечити узгодженість даних між усіма користувачами системи за рахунок зосередження всієї логіки у серверному компоненті, оскільки вся критична бізнес-логіка виконується на серверній стороні, а клієнтська частина лише відповідає за інтерфейс та виконує тільки те, що дозволяє серверна. Це є особливо важливим для систем складської логістики, де помилки у відображенні залишків або транзакцій можуть мати значні операційні наслідки.

Окремим чинником, який вплинув на вибір архітектури, є необхідність підтримки офлайн-режиму. Завдяки можливості виконання частини функцій на клієнті, зокрема кешування та накопичення відкладених операцій, система забезпечує безперервність роботи навіть за умов нестабільного мережевого підключення. При цьому відкладені операції синхронізуються з сервером після відновлення зв’язку, що підвищує надійність та гнучкість системи.

Клієнт-серверна архітектура також забезпечує достатній рівень масштабованості [2] та можливість подальшого розвитку системи. Завдяки чіткому розмежуванню відповідальностей між клієнтською частиною, сервером та підсистемами зберігання даних, стає можливим як розширення функціональності, так і інтеграція нових сервісів без суттєвого втручання в існуючу логіку. Таким чином, обраний архітектурний підхід є найбільш доцільним для реалізації інформаційної системи складської логістики відповідно до поставлених вимог та очікуваних умов її експлуатації.

Отже, застосування клієнт-серверної моделі є виправданим технологічним рішенням, що забезпечує оптимальне поєднання зручності користувацького інтерфейсу з потужними обчислювальними можливостями серверного ядра та

налаштування їх взаємодії. Цей підхід не лише вирішує поточні завдання щодо підтримки автономності та надає базу для забезпечення коректної роботи з інформацією, але й закладає потенціал для майбутнього розвитку та масштабування програмного забезпечення. Сформована архітектура взаємодії компонентів забезпечує цілісне розуміння того, як клієнтська частина, сервер та зберігання даних працюють у межах єдиної системи. Вона визначає обов'язки кожного компонента та встановлює чіткі правила обміну інформацією між ними.

Такий підхід підвищує керованість та передбачуваність роботи програмного комплексу, оскільки усі операції виконуються в узгодженому технологічному середовищі. Завдяки цьому система зберігає стабільність, підтримує логічну цілісність процесів і забезпечує коректність обробки даних в умовах різного рівня навантаження.

Запропонована клієнт-серверна архітектура узгоджує роботу всіх ключових компонентів системи та забезпечує їх послідовну взаємодію на технічному й логічному рівнях. Такий підхід гарантує надійність обробки даних, прозорість виконання бізнес-операцій і можливість масштабування без порушення цілісності програмного комплексу. Це робить обраний архітектурний підхід достатнім і обґрунтованим для реалізації функціональності інформаційної системи складської логістики, що задовольнить усі вимоги кінцевого користувача інформаційної системи.

3.3.3 Застосунок

Логічним наступним кроком розробки прикладного програмного забезпечення є вибір платформи, яка гратиме визначальну роль у можливостях застосунку, доступних технологіях, умовах експлуатації та рівню зручності для користувачів. На цьому етапі важливо оцінити різні класи клієнтських платформ, зокрема вебзастосунки, комп'ютерний додаток або ж рішення на основі мобільних технологій, порівнявши їх властивості з вимогами системи та особливостями майбутнього середовища використання. Такий підхід дозволяє

сформувати обґрунтоване рішення щодо технологічної основи застосунку та визначити подальшу траєкторію його проектування.

Застосунки у вигляді комп'ютерних додатків традиційно забезпечують високий рівень продуктивності та доступ до локальних ресурсів, проте їх розробка та супровід потребують окремої підтримки операційних систем, складного розгортання й регулярного оновлення на кожному робочому місці. Для систем складської логістики це створює додаткові експлуатаційні витрати, ускладнює масштабування та знижує гнучкість впровадження. Подібні обмеження роблять цю модель менш придатною для розподілених робочих середовищ, де різні користувачі можуть працювати з різних пристроїв.

Мобільні застосунки орієнтовані на компактність, високу мобільність та інтеграцію з апаратними сенсорами, що може бути корисним у складських умовах. Як приклад, при скануванні штрих кодів або роботі з графічними матеріалами за рахунок вбудованих камер, тощо. Однак розробка повноцінного мобільного рішення потребує створення окремих версій для різних платформ, таких як Android та iOS, забезпечення сумісності й підтримки нативного функціоналу. Крім того, мобільний інтерфейс має обмеження щодо обсягу матеріалу на екрані за рахунок невеликих дисплеїв, що ускладнює виконання багатьох операцій, пов'язаних з аналітикою, керуванням довідниками та обробкою складних форм.

Вебплатформа забезпечує універсальний доступ із будь-якого пристрою, що має браузер, не потребує індивідуального інсталювання та дозволяє централізовано оновлювати застосунок без втручання кінцевих користувачів. Вона поєднує мобільність, мінімальні вимоги до середовища виконання та достатню гнучкість для реалізації складних користувацьких сценаріїв. Крім того, сучасні вебтехнології дозволяють підтримувати офлайн-режим, локальне кешування та інтерактивну взаємодію, що робить вебзастосунок оптимальним рішенням для системи складської логістики, де важливими є доступність, масштабованість і простота розгортання.

Головні переваги вебплатформи:

а) вебзастосунок працює на будь-якому пристрої з браузером;

б) оновлення застосунку виконуються централізовано;

в) вебплатформа підтримує локальне зберігання та офлайн-режим;

г) вебінтерфейси легко масштабуються під різні сценарії роботи;

д) вебтехнології гарантують швидку розробку та модифікацію функціоналу.

Але варто виділити і недоліки:

а) вебзастосунки мають обмежений доступ до системних ресурсів пристрою;

б) продуктивність вебрішень нижча порівняно з нативними програмами;

в) якість офлайн-роботи залежить від можливостей браузера;

г) стабільна робота потребує надійного мережевого з’єднання;

д) інтеграція зі спеціалізованим обладнанням є ускладненою.

Обрана платформа має обмеження, але вибір веб для реалізації цього застосунку є обґрунтованим через його переваги:

а) працює у браузері без встановлення;

б) оновлюється централізовано;

в) підтримує локальне зберігання й офлайн;

г) легко масштабується під різні пристрої;

д) швидко розширюється функціонально.

Таким чином, використання вебплатформи як основи застосунку забезпечує зручний доступ, простоту розгортання та високу гнучкість, що дозволяє реалізувати функціональність відповідно до вимог системи та підтримувати ефективну роботу користувачів у різних умовах.

Наявні обмеження вебтехнологій можуть бути компенсовані відповідними архітектурними рішеннями, оптимізацією інтерфейсу та впровадженням механізмів автономної роботи.

За потреби система може бути розширена адаптивними або нативними мобільними інтерфейсами, що дозволить охопити ширший спектр пристроїв і сценаріїв використання.

3.4 Вибір та обґрунтування залучених технологічних рішень

Проектування інформаційної системи передбачає визначення технологічних рішень, що забезпечують її функціональність, надійність і відповідність заданим вимогам. Вибір технологій визначає підхід до реалізації програмних компонентів, організації взаємодії між ними та підтримки необхідних режимів роботи. У процесі відбору важливо враховувати можливості інструментів, їхню сумісність, продуктивність і придатність до масштабування. Сформований набір технологічних рішень створює основу для ефективної розробки та стабільної роботи системи.

3.4.1 Мови програмування

Вибір мови програмування для серверної частини інформаційної системи є одним із ключових технологічних рішень, оскільки саме вона визначає підхід до реалізації бізнес-логіки, можливості масштабування, продуктивність, надійність застосунку та шляхи інтеграції з іншими сервісами. Сучасний технологічний ринок пропонує широкий спектр мов, придатних для створення веборієнтованих серверних рішень, серед яких найбільш поширеними є Java [3], C#, Python, JavaScript (Node.js), Go та PHP. Кожна з них має власну екосистему, інструментарій, переваги та сфери застосування, тому доцільно розглянути їх з огляду на специфіку завдань системи складської логістики.

Однією з популярних універсальних мов є JavaScript [4] у середовищі Node.js, який вирізняється високою гнучкістю та широким застосуванням у веброзробці. Node.js забезпечує асинхронну модель обробки запитів, що добре підходить для систем зі значною кількістю одночасних підключень. Завдяки єдиній мові на клієнті та сервері зменшується складність проєкту. Однак Node.js менш придатний для складних обчислювальних операцій, вимагає додаткової уваги до керування потоками та має нижчу стабільність у довготривалих корпоративних рішеннях.

Мова Python часто застосовується в аналітичних системах, сервісах швидкої розробки, машинному навчанні та інтеграційних застосунках. Її перевагою є простота синтаксису, велика кількість бібліотек та висока швидкість розробки. Проте інтерпретована природа Python не забезпечує достатнього рівня продуктивності для важких серверних навантажень, а одночасна обробка великої кількості запитів потребує додаткових рішень, що ускладнює масштабування.

Мова C# та платформа .NET є потужним корпоративним інструментом, який пропонує сучасний набір засобів для серверної логіки, багатопоточності та роботи з великими обсягами даних. Вона добре підходить для систем зі складною внутрішньою структурою та високими вимогами до надійності. Недоліком є прив’язка екосистеми до інфраструктури Microsoft, а також складніше розгортання у багатоплатформному середовищі, що може створювати обмеження для систем, які мають функціонувати на різних типах серверів.

Мова Go (Golang) набуває популярності у розробці високопродуктивних сервісів зі значним навантаженням завдяки ефективній роботі з потоками та низьким накладним витратам. Go забезпечує високу продуктивність і добру масштабованість у мікросервісній архітектурі. Водночас екосистема Go менш розвинена порівняно з Java чи .NET, а засоби для розробки корпоративних рішень є більш обмеженими, що створює додаткові складнощі при реалізації складних бізнес-процесів.

Мова PHP історично є одним із найпоширеніших інструментів для веброзробки, особливо в контексті створення сайтів і контент-орієнтованих систем. Хоча сучасні версії PHP значно покращили продуктивність і можливості, ця мова менш придатна для високонавантажених систем реального часу та складної корпоративної логіки. Її використання є доцільним у контексті простих вебдодатків або CRM-систем середнього рівня.

На тлі перелічених альтернатив Java залишається однією з найстабільніших і технологічно зрілих мов для серверної розробки. Java має багаторічний розвиток екосистеми, стандартизовану платформу Java Virtual Machine та широкий вибір інструментів для реалізації масштабованих, надійних і продуктивних систем.

Одним із головних її переваг є підтримка багатопоточності та можливість ефективної роботи під високим навантаженням. Завдяки використанню JVM програми на Java можуть працювати на будь-якій операційній системі без зміни коду, що забезпечує платформну незалежність і зручність розгортання.

Додатковою перевагою Java є розвинена корпоративна екосистема, яка включає такі фреймворки та технології, як Spring Framework [5], Hibernate [6], JPA, Jakarta EE, інструменти для побудови мікросервісів, а також розгалужені можливості для тестування й моніторингу систем. Це робить Java одним із найкращих виборів для проектів, де важлива стабільність, структурованість, передбачуваність і довгострокове технічне обслуговування.

З огляду на вимоги системи складської логістики — багатокористувацький доступ, обробку великої кількості операцій, підтримку складної бізнес-логіки, надійний серверний механізм та можливість подальшого масштабування — Java є оптимальним вибором. Її продуктивність, стабільність, екосистема та інструментарій забезпечують основу для створення надійної, безпечної та довготривалої інформаційної системи, орієнтованої на реальні бізнес-процеси.

Клієнтська частина інформаційної системи виконує ключову роль у забезпеченні взаємодії користувача з функціональністю застосунку. Для реалізації інтерфейсу важливо обрати мову програмування, яка забезпечує достатню продуктивність, універсальність, інтерактивність та підтримку сучасних вебтехнологій. На сьогодні існує кілька платформ і мов, що можуть бути використані для побудови клієнтських застосунків, серед яких JavaScript, TypeScript, Dart, Kotlin/JS, а також фреймворки для кросплатформної розробки на основі інших мов. Кожна з цих технологій має свої властивості, які визначають доцільність її використання у веборієнтованих системах.

Базовим інструментом для клієнтської розробки є JavaScript, який виступає стандартною мовою для виконання коду у браузері. Завдяки підтримці всіма сучасними браузерами JavaScript забезпечує можливість створення інтерактивних інтерфейсів, обробки подій, динамічного оновлення сторінок та роботи з веб-API.

Його екосистема охоплює широкий спектр інструментів, бібліотек і

фреймворків, що значно прискорює розробку складних інтерфейсних рішень. Оскільки JavaScript виконується безпосередньо на стороні клієнта, він дозволяє реалізувати високий рівень інтерактивності, зменшити навантаження на сервер та забезпечити швидку реакцію системи на дії користувача.

На відміну від альтернатив, JavaScript є єдиною мовою, яка підтримується браузерами нативно, без проміжних компіляційних кроків та без додаткового інтерпретатора. Це означає швидке завантаження, прямий доступ до DOM [7], веб-API, IndexedDB [8], Service Workers та інших механізмів, які є критичними для реалізації офлайн-режиму та локального зберігання даних. JavaScript дозволяє гнучко поєднувати локальну обробку даних, інтерактивність та роботу через мережу, що є необхідним для складських систем, які можуть працювати навіть при тимчасовій втраті з’єднання.

Важливою перевагою JavaScript є наявність фреймворків та бібліотек, таких як React [9], Vue.js, Angular, SolidJS, які дозволяють створювати масштабовані компоненти інтерфейсу, організовувати стан застосунку та забезпечувати високу продуктивність.

У контексті даного проекту використання React дає змогу реалізувати модульну структуру UI, ефективно керувати станом клієнтської частини та інтегрувати механізми офлайн-черги та синхронізації з сервером через IndexedDB. Це також спрощує підтримку та розширення функціональності в майбутньому.

З огляду на специфіку системи складської логістики — динамічну роботу з великою кількістю сутностей, підтримку офлайн-режиму, інтерактивну взаємодію з користувачем та необхідність швидкого оновлення інтерфейсу — JavaScript є оптимальною мовою для реалізації клієнтської частини. Його універсальність, широка екосистема, підтримка браузерами та наявність інструментів для організації офлайн-роботи забезпечують стабільність і гнучкість застосунку. Це робить JavaScript найбільш доцільним вибором для створення сучасного вебінтерфейсу, адаптованого до реальних умов експлуатації складської інформаційної системи.

3.4.2 Бібліотеки та фреймворки

Після визначення мов програмування для серверної та клієнтської частин Java та JavaScript відповідно, наступним кроком є вибір бібліотек і фреймворків, які забезпечують ефективну реалізацію застосунку відповідно до поставлених вимог. Для кожної з мов існує широкий спектр технологічних рішень, що відрізняються архітектурними підходами, рівнем абстракції, можливостями масштабування та зрілістю екосистеми. Тому важливо розглянути найбільш поширені фреймворки для Java та JavaScript і визначити, які з них найбільш підходять для створення інформаційної системи складської логістики. У екосистемі Java існує кілька сучасних платформ для розроблення вебзастосунків: Spring Boot, Jakarta EE, Micronaut, Quarkus та низка легших мікрофреймворків. Вони різняться підходами до конфігурації, швидкістю роботи, підтримкою інструментів та можливостями інтеграції з іншими сервісами.

Jakarta EE (колишній Java EE) є традиційною корпоративною платформою, що забезпечує стандартизований набір специфікацій для побудови складних систем. Вона пропонує потужний набір інструментів для транзакцій, безпеки та обробки даних, однак її використання пов’язане з певною громіздкістю та необхідністю розгортання на спеціалізованих сервер контейнерах, що ускладнює впровадження у контейнеризованих або хмарних середовищах.

Micronaut та Quarkus є легковаговими рішеннями, орієнтованими на мікросервісну архітектуру та швидкий старт застосунків. Вони забезпечують високу продуктивність і зменшені витрати на пам’ять, проте їхня екосистема менш розвинена порівняно зі Spring, особливо щодо роботи з транзакціями, складною бізнес-логікою та доступом до великої кількості інтеграційних модулів.

Найповнішим рішенням є Spring Boot — фреймворк, що став індустріальним стандартом для побудови Java-орієнтованих вебсервісів. Він забезпечує автоматичну конфігурацію, модульність, підтримку REST-сервісів[10], інтеграцію з базами даних, інструменти безпеки, роботу з транзакціями, механізми кешування та зручні засоби тестування. Велика кількість готових

рішень, розширень та бібліотек дозволяє швидко впроваджувати функціональність без необхідності створювати низькорівневі механізми самостійно. У контексті складської системи з великою кількістю сутностей, транзакцій, ролей користувачів та вимогою стабільної роботи — Spring Boot є оптимальним інструментом серверної розробки.

Щодо клієнтської частини, то екосистема JavaScript включає велику кількість засобів для створення динамічних інтерфейсів. Найпоширенішими підходами є використання фреймворків React, Vue.js, Angular та Svelte, а також низки бібліотек для керування станом, маршрутизації та роботи з локальними сховищами.

Angular є комплексним фреймворком із суворою структурою, вбудованими модулями та великим набором інструментів. Він добре підходить для великих корпоративних проектів, але його жорстка архітектура та складність роблять його менш гнучким у застосунках, де потрібне швидке розширення інтерфейсу та просте управління компонентами.

Vue.js забезпечує низький поріг входу та простоту створення невеликих динамічних інтерфейсів. Попри це, Vue менш структурований і менш поширений у складних корпоративних рішеннях, що впливає на доступність інструментів, бібліотек і готових інтеграцій.

Svelte генерує оптимізований код і забезпечує високу продуктивність, але його екосистема та індустріальна зрілість все ще поступаються основним фреймворкам, що може створювати додаткові обмеження під час масштабування.

Найбільш збалансованим рішенням є React — бібліотека, що дозволяє створювати компонентні інтерфейси, ефективно управляти станом, працювати з асинхронними даними та інтегрувати локальні механізми збереження інформації. React має широку екосистему, включаючи засоби маршрутизації, керування глобальним станом, оптимізації продуктивності та взаємодії з веб-API. Його компонентний підхід дозволяє легко масштабувати інтерфейс, розділяти відповідальності між модулями та впроваджувати складні UI-сценарії — зокрема, механізми офлайн-черги, відкладені операції, локальні кеші та оптимізовані

списки сутностей, що є ключовими елементами системи складської логістики.

React є також технологічно зрілим та добре підтримуваним рішенням, що має велику спільноту та значну кількість перевірених практикою інструментів. Завдяки стабільності екосистеми та регулярним оновленням бібліотеки забезпечується тривала підтримка застосунку та можливість поступового розширення його функціональності без необхідності кардинальних змін архітектури.

React легко інтегрується з сучасними інструментами збірки, тестування та оптимізації, що сприяє підвищенню продуктивності інтерфейсу та зменшенню витрат на супровід. Така гнучкість і адаптивність роблять бібліотеку придатною для систем, де важливо підтримувати стабільність, швидку реакцію на дії користувача та надійність функціонування в різних умовах та середовищах експлуатації.

Загалом, вибір Spring Boot та React є логічним і технічно обґрунтованим з огляду на вимоги системи. Spring Boot забезпечує стабільну серверну інфраструктуру, підтримку транзакційності, гнучкість у розширенні функціональності та інтеграцію з базою даних.

React, у свою чергу, забезпечує модульність, інтерактивність, підтримку офлайн-режиму та можливість створення швидкого, адаптивного інтерфейсу для складських операцій. Разом ці технології формують надійний фундамент для розроблення сучасного вебзастосунку, здатного ефективно функціонувати в умовах реальних логістичних процесів.

3.4.3 Сховище даних

Проектування підсистеми зберігання даних є критично важливим етапом, оскільки саме вона забезпечує довготривалу цілісність інформації, коректність виконання бізнес-операцій та можливість масштабування застосунку. У сучасних програмних системах використовуються два основних підходи до організації даних: реляційні сховища та нереляційні, кожен з яких пропонує власний набір

більш детальних реалізацій. Кожен із них також має власну модель даних, механізми транзакційності та рівень забезпечення консистентності, що суттєво впливає на вибір технології під конкретні вимоги системи.

Реляційні СУБД, такі як PostgreSQL [11], MySQL чи Oracle, використовують структуровану модель даних на основі таблиць та підтримують складні зв’язки між сутностями. Їхньою перевагою є повна підтримка ACID-властивостей [12] (Atomicity, Consistency, Isolation, Durability), що гарантує надійність транзакцій та можливість виконання складних запитів. Для систем з великою кількістю залежних сутностей та транзакційних операцій реляційний підхід є найбільш передбачуваним і безпечним.

Нереляційні сховища, такі як MongoDB, Cassandra, DynamoDB, Redis [13], ElasticSearch більше орієнтовані на зберігання неструктурованих даних, гнучкі схеми та високу горизонтальну масштабованість. Вони забезпечують високу швидкість запису та читання, але зазвичай жертвують строгими транзакційними гарантіями, повною консистентністю або підтримкою складних зв’язків між сутностями.

Для системи складської логістики, де існують залежності між товарами, складами, партіями, транзакціями руху товару та історичними даними, реляційна модель є більш природною. Вона дозволяє гарантувати коректність залишків, послідовність операцій та цілісність бізнес-процесів, що є ключовими вимогами.

Серед доступних реляційних систем найбільш придатним рішенням є PostgreSQL, оскільки ця СУБД поєднує високу продуктивність, гнучкість та набір функцій, необхідних для роботи з великими обсягами взаємопов’язаних даних. PostgreSQL повністю підтримує ACID-механізми, що гарантує надійність транзакцій навіть у середовищі із значною кількістю одночасних операцій. Завдяки широкому вибору рівнів ізоляції, системі легко адаптувати поведінку транзакцій під різні бізнес-сценарії, забезпечуючи баланс між продуктивністю та консистентністю.

Крім того, PostgreSQL має потужний механізм індексації, включаючи спеціалізовані структури для прискорення пошуку в складних доменах даних, що

є важливим при роботі з великою кількістю товарних позицій та історичних записів. Підтримка складних запитів, тригерів, процедур і розширень дозволяє реалізувати частину бізнес-логіки на рівні бази, зменшуючи навантаження на сервер застосунку та спрощуючи підтримку системи. Сукупність цих властивостей робить PostgreSQL оптимальним вибором для ролі основного сховища критично важливої інформації.

У системі існує група даних, які оновлюються з високою частотою, мають обмежений строк актуальності та не потребують довготривалого зберігання в основній реляційній базі. До таких належать тимчасові токени доступу, службові маркерні значення, проміжні обчислення, курси валют та інші швидкозмінні параметри, що використовуються для перевірок або короткочасних операцій.

Зберігання подібних даних у реляційній СУБД є недоцільним через зайве навантаження на транзакційний механізм та відсутність потреби в їхній довготривалій консистентності. Це створює потребу у швидкодійному сховищі, яке підтримує автоматичний час життя елементів і забезпечує миттєвий доступ до інформації, що використовується у великій кількості дрібних операцій. Саме такі вимоги роблять доцільним застосування спеціалізованого високошвидкісного кеш-сховища, здатного ефективно працювати з даними, для яких TTL є ключовою характеристикою.

Із доступних рішень для організації високошвидкісного кешування найчастіше використовуються Memcached, Hazelcast та Redis, які забезпечують роботу з даними безпосередньо в оперативній пам’яті. Memcached орієнтований переважно на просту модель ключ-значення і не підтримує складні структури чи механізми керування даними, що обмежує його застосування у системах із різноманітними типами тимчасових об’єктів. Hazelcast надає ширші можливості, однак є значно складнішим у налаштуванні та інтеграції, а його розгортання потребує додаткових ресурсів та адміністративних витрат.

Найбільш збалансованим рішенням у цьому контексті є Redis, який поєднує високу швидкодію з підтримкою різноманітних структур даних, механізмів TTL, черг, лічильників та публікацій, що дозволяє охопити широкий спектр сценаріїв

роботи з тимчасовою інформацією.

Завдяки цим властивостям Redis забезпечує оптимальну продуктивність системи та ефективно доповнює реляційну базу даних, знімаючи з неї навантаження, пов’язане із частими операціями на основі короткочасних даних.

Висновки до розділу 3

У цьому розділі було здійснено комплексне архітектурне проектування інформаційної системи, що охоплює визначення функціональних і нефункціональних вимог, формування сценаріїв використання та побудову загальної структурної схеми. Було розроблено діаграму компонентів системи, зображену у додатку В. На основі аналізу предметної області сформовано перше наближення системи та обґрунтовано вибір клієнт-серверної моделі взаємодії, що забезпечує необхідний рівень керованості, надійності та узгодженості даних.

Окрему увагу приділено застосунку та вибору технологічних рішень, включаючи програмні платформи, бібліотеки, фреймворки та інструменти зберігання інформації. У результаті визначено доцільність використання Java та JavaScript для реалізації серверної та клієнтської частин, а також оптимальність поєднання PostgreSQL як основної реляційної СУБД та Redis як високошвидкісного кеш-сховища. Сукупність прийнятих рішень формує цілісну архітектурну основу, що забезпечує масштабованість, продуктивність і стабільність функціонування системи відповідно до поставлених вимог.

4 РЕАЛІЗАЦІЯ СИСТЕМИ

Реалізація інформаційної системи охоплює практичне втілення концепцій та архітектурних рішень, визначених на етапі проектування. На цьому етапі формується структура бази даних, розробляється бізнес-логіка, створюється серверна частина та клієнтський застосунок, що забезпечують повний цикл функціонування системи. Особлива увага приділяється організації механізмів захисту даних, обробці транзакцій, підтримці офлайн-режиму та зручності взаємодії користувача з інтерфейсом.

У межах розділу описуються основні сутності системи та їхня взаємодія, механізми реалізації складських операцій, структура серверного API, а також підходи до організації інтерфейсу та налаштувань робочого середовища. Наведено особливості поведінки застосунку під час реальної експлуатації, включаючи обробку запитів, відображення даних, валідацію та синхронізацію з сервером після роботи в автономному режимі.

Окремим етапом є тестування розробленої системи, яке включає модульні й ручні перевірки для підтвердження коректності роботи компонентів, надійності бізнес-логіки та відповідності функціональних можливостей встановленим вимогам. Також подано інструкцію користувача, що описує принципи роботи із системою, вимоги до середовища та порядок виконання основних операцій.

У сукупності цей розділ демонструє практичне формування готового програмного продукту, що відповідає поставленим завданням, забезпечує стійкість роботи та підтримує основні процеси складської логістики.

4.1 Захист даних

Забезпечення безпеки даних є фундаментальною вимогою при створенні сучасних інформаційних систем, особливо тих, що працюють зі складськими операціями, персональними даними та даними доступу користувачів. Захист інформації охоплює комплекс технічних та організаційних заходів, спрямованих

на запобігання несанкціонованому доступу, підміні даних, втраті цілісності або порушенню конфіденційності. У процесі реалізації системи важливо не лише захистити канали передачі та механізми аутентифікації, але й забезпечити безпечне зберігання критичних даних, таких як паролі, токени доступу, службові параметри та внутрішні ідентифікатори. Саме тому на етапі реалізації було обрано сучасні, перевірені часом криптографічні алгоритми та методи авторизації, які відповідають актуальним стандартам галузі.

Архітектура системи базується на кількох рівнях безпеки, що працюють сумісно: шифрування та хешування даних, механізми аутентифікації та авторизації, контроль доступу на основі ролей, захист від підробки запитів, безпечне управління токенами та аудит операцій. Такий підхід дозволяє гарантувати, що окремі вектори атаки — перехоплення даних, компрометація облікових записів, доступ до бази даних чи використання підроблених токенів — не призведуть до повного порушення безпеки системи.

Для генерації унікальних службових ідентифікаторів у системі застосовується алгоритм SHA-256. Це криптографічна хеш-функція, що забезпечує одностороннє перетворення даних у 256-бітне значення, яке не дозволяє відновити початкову інформацію. Хешування використовується для формування ідентифікаторів офлайн-завдань, службових маркерів або інших елементів, яким необхідна унікальність без відображення вихідних даних.

Застосування SHA-256 забезпечує високу стійкість до колізій, передбачувану структуру вихідного значення та можливість безпечно зберігати інформацію, що використовується в механізмах синхронізації та внутрішньої логіки системи. Алгоритм добре підтримується, швидко виконується та відповідає сучасним криптографічним стандартам, що робить його оптимальним для службових операцій, де важлива незворотність перетворення.

Паролі користувачів не можуть зберігатися у відкритому вигляді, оскільки компрометація бази даних у такому випадку відразу відкриває доступ до облікових записів. Для цього в системі використано алгоритм BCrypt — адаптивну криптографічну хеш-функцію, яка спеціально розроблена для

безпечного зберігання паролів.

BCrypt [14] застосовує механізм сольових значень, що унеможливлює використання rainbow-tables, а також виконує багаторазові ітерації обчислення хешу, що значно уповільнює brute-force підбір. Його адаптивність дозволяє збільшувати складність хешування зі зростанням обчислювальних потужностей, підтримуючи актуальний рівень безпеки упродовж тривалого часу.

Використання BCrypt у системі гарантує, що навіть у разі витоку даних доступ до реальних паролів залишиться неможливим, а зловмисник зіткнеться з практично непридатною для декрипту інформацією.

Для авторизації в системі використовується JSON Web Token [15]. Загальний алгоритм авторизації користувача зображено на рисунку 4.1. Це формат компактних криптографічно захищених токенів, які містять інформацію про користувача та його роль, а також підписані секретним ключем серверної частини.

Токен передається клієнту після успішної аутентифікації та використовується в подальших запитах, що дозволяє організувати модуль авторизації без збереження стану.

Поєднання механізмів захисту паролів за допомогою BCrypt та безстатусної авторизації через JWT формує надійну архітектуру безпеки системи. BCrypt відповідає за максимально стійке зберігання облікових даних, тоді як JWT дає змогу швидко та ефективно перевіряти права доступу без додаткових навантажень на сервер.

Така комбінація забезпечує баланс між продуктивністю та високим рівнем захищеності, створюючи міцний фундамент для подальшого масштабування та розвитку системи. Додатково в системі реалізовано механізми обробки токенів, що забезпечує гнучке керування доступом облікових даних.

Це дозволяє серверу може не лише перевіряти коректність підпису та цілісність JWT, а й контролювати його стан у реальному часі через спеціальні списки блокування.

Такий підхід значно підвищує рівень безпеки та дозволяє ефективно реагувати на потенційні загрози.

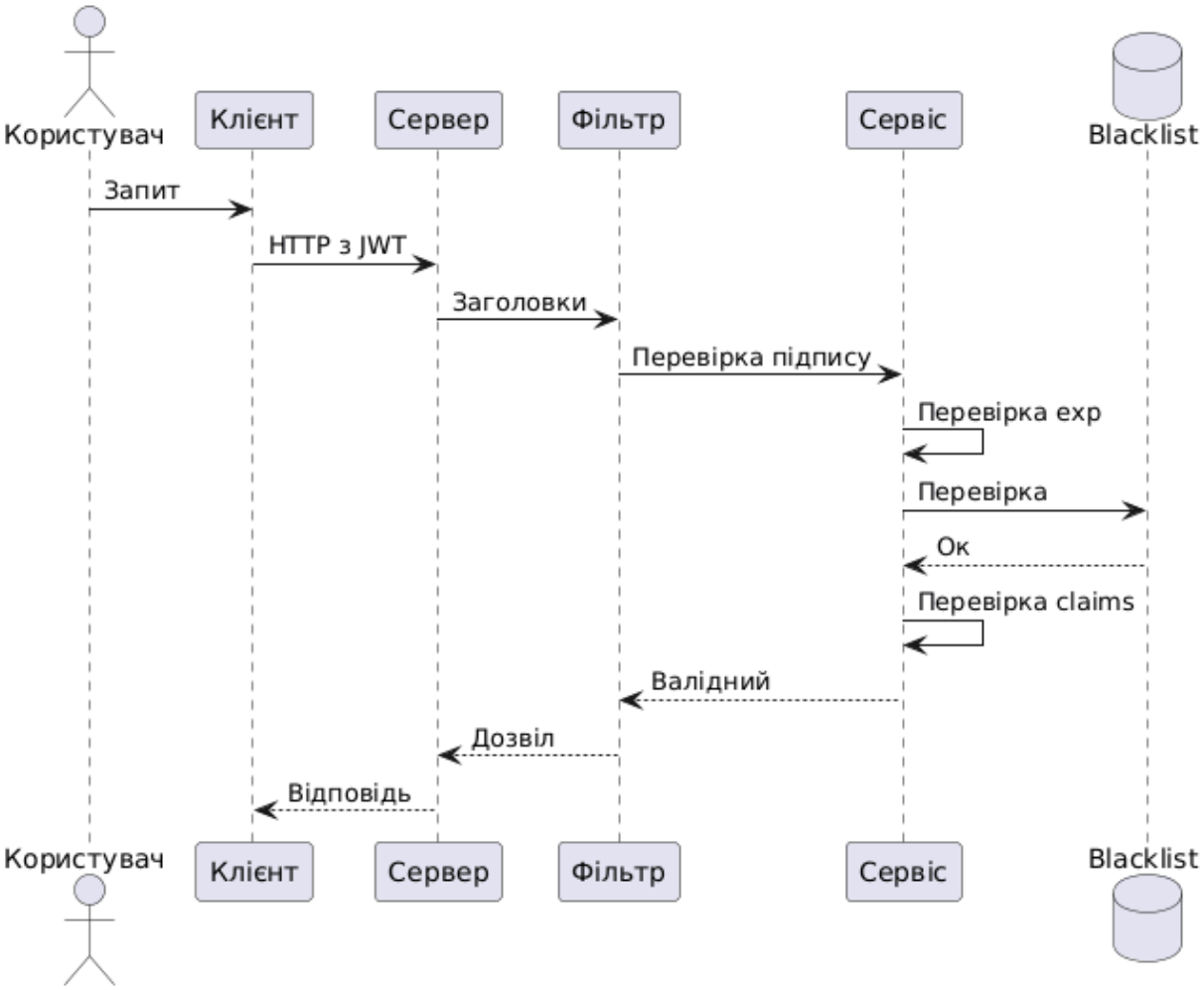


Рисунок 4.1 — Загальний принцип авторизації в системі на основі JWT-токена

Функціональні переваги:

а) токени є *stateless*, тому сервер не зберігає стан сесії та не потребує окремої таблиці активних сеансів;

б) відсутня необхідність у серверних механізмах трекінгу сесій, що покращує масштабованість;

в) токен містить закодовані дані про роль користувача, що прискорює перевірку прав доступу;

г) механізм підпису унеможливлює підробки токена без знання секретного ключа;

д) JWT легко інтегрується з фронтенд-застосунками та мікросервісами;

е) підтримує встановлення часу дії, що знижує ризики компрометації.

Функціональні недоліки:

а) у разі витоку токена зловмисник отримує тимчасовий доступ до системи до моменту його закінчення;

б) неможливо відкликати токен миттєво без додаткових механізмів (чорні списки, ротація секрету);

в) збільшений розмір токена порівняно з класичними sessionID.

Попри ці особливості, JWT є оптимальним механізмом для системи, яка працює у вебсередовищі, підтримує офлайн-режим і потребує мінімізації серверного стану. Stateless-авторизація також спрощує розгортання у контейнерному середовищі, де можуть використовуватися кілька інстансів серверної частини.

Поєднання SHA-256 для службових ідентифікаторів, BCrypt для зберігання паролів та JWT як механізму авторизації формує багаторівневу модель захисту, у якій кожен інструмент виконує свою роль.

Ці інструменти дозволяють гарантувати захист персональних даних відповідно до вимог галузі. Система залишається стійкою до типових атак, включаючи перехоплення даних, вгадування паролів, модифікацію токенів та спроби несанкціонованого доступу, що особливо важливо для інформаційної системи, яка підтримує критичні складські процеси.

Окрім серверної частини реалізації логіки безпеки та конфігурування алгоритму перевірок, ще існують багаторівневі валідації даних на предмет відповідності очікуваній довжині, відсутності заборонених символів, логічному значенню чисел, наявності коректних форматів полів, унікальності значень та узгодженості введених даних між собою.

Ця логіка реалізована як на серверній так і на клієнтській стороні застосунку, що гарантує базову коректність даних, що надходять на сторону сервера з клієнтського застосунку.

4.2 Визначення структури БД

Розробка концептуальної, логічної та фізичної моделей даних є одним з головних етапів проектування інформаційної системи складської логістики, оскільки кожен рівень моделювання описує структуру даних з різним ступенем абстракції. Такий підхід дозволяє забезпечити відповідність моделі вимогам системи, узгодити структуру даних із бізнес-процесами та підготувати ефективну основу для реалізації бізнес-логіки.

Концептуальна модель у вигляді ER-діаграми [16] є першим і найбільш абстрактним рівнем представлення даних. На цьому етапі визначаються основні сутності системи — склади, локації, товари, запаси, операції руху товару, користувачі, ролі, групи товарів, тегування, а також сутності щодо офлайн-черги. Модель описує лише сутності та зв’язки між ними, без уточнення атрибутів і технічних характеристик. Це дозволяє отримати цілісне уявлення про структуру даних і підтвердити її відповідність предметній області.

Логічна модель даних деталізує концептуальну модель та визначає атрибути кожної сутності, а також точний тип зв’язків між об’єктами. На цьому етапі уточнюються поля таблиць: коди товарів, назви складів, ідентифікатори користувачів, кількість одиниць запасу, типи операцій, статуси, параметри транзакцій і службові метадані. Логічна модель залишається незалежною від конкретної СУБД, але вже дає змогу описати повну структуру майбутньої бази даних.

Фізична модель даних адаптує логічну модель під конкретну СУБД, обрану у проекті, і це PostgreSQL. На цьому рівні визначаються типи даних для кожного атрибута, створюються первинні та зовнішні ключі, обмеження цілісності, індекси та механізми оптимізації. Також на фізичному рівні враховуються особливості PostgreSQL, такі як підтримка JSONB, ефективні індекси GIN/GiST, унікальні обмеження, тригери та каскадні операції. Результатом є готова структура БД, придатна для безпосереднього використання застосунком.

Такий покроковий підхід дозволяє побудувати базу даних, яка буде

стабільною, логічно цілісною та оптимальною для роботи з великим обсягом сутностей складської системи, включно з товарами, залишками, операціями руху, користувачами та іншими службовими сутностями. Розглянемо ключові сутності системи.

Склад характеризується:

а) унікальним кодом, що дозволяє однозначно його ідентифікувати;

б) назвою, яка використовується у внутрішніх процесах;

в) адресою;

г) інформацією про режим роботи;

д) контактними номерами;

е) менеджером, відповідальним за операційну діяльність;

ж) статусом активності.

Користувачі системи мають різні рівні доступу залежно від ролі, що визначає їхню взаємодію з логістичними процесами.

У системі присутні три основні типи користувачів:

а) оператор — виконує щоденні складські операції;

б) менеджер — керує складом, має доступ до аналітики та конфігурації;

в) власник — адмініструє всю систему та управляє всіма об’єктами.

Кожен користувач характеризується власним іменем, ідентифікаційними даними, електронною поштою, статусом та часовим поясом. Товари описують номенклатуру, яка зберігається на складах, і містять інформацію про базові властивості.

Товар характеризується:

а) унікальним кодом;

б) назвою;

в) описом;

г) ціною та валютою;

д) датою створення та оновлення;

е) фізичними параметрами (вага та габарити).

Додатково для кожного товару можуть існувати фотографії, що мають

власний зовнішній ідентифікатор і URL-посилання.

Для забезпечення можливості гнучкої класифікації товарів використовуються теги та групи запасів.

Тег характеризується:

а) назвою;

б) статусом активності.

Група запасів містить:

а) код;

б) назву;

в) статус активності.

Запаси товарів зберігаються у вигляді одиниць запасу, які представляють партії товару на конкретному складі.

Товар характеризується:

а) номером партії;

б) кодом;

в) товаром, якому вона відповідає;

г) групою запасу;

д) складом, на якому вона знаходиться;

е) датою придатності;

ж) доступною кількістю;

з) статусом;

и) активністю;

і) секцією зберігання.

Це дозволяє відстежувати фізичний стан та локацію кожної партії.

Система підтримує складську структуру, що включає зони, секції та комірки. Секція зберігання характеризується:

а) належністю до складу;

б) кодом;

в) статусом активності.

Локації формують вкладену структуру, яка забезпечує деталізацію простору

та оптимізацію розміщення товарів.

Для організації пересувань товарів використовується сутність відправлення. Вона характеризує переміщення товарів і містить:

а) код операції;

б) склад-відправник;

в) склад-одержувач або адресу доставки;

г) товар, що переміщується;

д) кількість;

е) ініціатора;

ж) статус;

з) дату створення;

и) напрямок переміщення.

Ця сутність дозволяє контролювати рух товарів між складами або до зовнішніх клієнтів.

Додатково система підтримує фінансові операції.

Транзакція характеризується:

а) унікальним ідентифікатором;

б) референсом;

в) типом фінансового потоку;

г) призначенням;

д) статусом;

е) сумою;

ж) валютою;

з) бенефіціаром;

и) зовнішніми референсами;

і) датами створення та оплати;

й) платіжним провайдером.

Бенефіціар, у свою чергу, містить IBAN, SWIFT-код, ім’я, номер картки та статус активності.

Історія одиниць запасу забезпечує відстеження всіх змін, що відбуваються з

партіями товару.

Кожен запис включає:

а) посилання на одиницю запасу;

б) назву товару та ціну на момент зміни;

в) валюту;

г) дату логування;

д) стару та нову групу запасу;

е) старий та новий склад;

ж) кількість до та після зміни;

з) стару і нову дату придатності;

и) статуси до та після;

і) зміну секції зберігання;

й) зміну активності партії.

Це забезпечує повний аудит руху товарів та змін у їх стані. Такий рівень деталізації сутностей дозволяє побудувати логічну модель даних, на основі якої формується концептуальна ER-діаграма зображена у додатку Г та логічна модель зображена у додатку Д, також для збереження цих елементів необхідно визначити фізичну модель даних, яка визначить структуру на рівні бази даних у додатку Е. Після цього визначаються типи даних, зовнішні зв’язки та структурні обмеження, що застосовуються у фізичній моделі бази даних. Результатом є повністю сформована структура зберігання, яка відповідає вимогам системи та забезпечує її стабільну і передбачувану роботу у всіх операційних сценаріях.

4.3 Розроблення бізнес логіки системи

Бізнес-логіка реалізує функціонал програмної системи, саме вона визначає правила оброблення даних, взаємодію між сутностями та послідовність виконання операцій. На цьому етапі здійснюється формалізація вимог та перетворення у чіткі алгоритми, що забезпечують коректність, цілісність та узгодженість роботи інформаційної системи складської логістики.

4.3.1 Основні сутності системи

Розглянемо основні сутності системи, що втілюють у собі головні елементи роботи зі складською системою.

Клас Warehouse представляє одну з ключових сутностей системи — склад, що описує логістичний об’єкт, його характеристики, контактні дані, графік роботи та відповідального менеджера.

Атрибути класу:

а) id — унікальний ідентифікатор складу, який генерується автоматично;

б) code — унікальний символьний код складу;

в) name — назва складу;

г) address — JSON-об’єкт типу Address, що містить структуровані дані про місцезнаходження;

д) working_hours — JSON-структура типу WorkingHours, яка визначає робочий час по днях тижня;

е) phones — список контактних телефонів у вигляді масиву varchar[];

ж) manager — пов’язаний об’єкт Employee, який відповідає за управління складом;

з) manager_id — зовнішній ключ на співробітника-менеджера, що спрощує передачу та серіалізацію;

и) is_active — ознака активності складу.

Методи класу генеруються за допомогою анотацій Lombok і включають конструктори, гетери, сетери, а також допоміжні методи порівняння й формування рядкового представлення.

Клас User представляє сутність системи, що описує обліковий запис користувача, його авторизаційні дані, часові параметри входу, ролі та статуси доступу.

Атрибути класу:

а) id — унікальний ідентифікатор користувача;

б) username — унікальне ім’я користувача для авторизації;

в) password — хешований пароль;

г) email — унікальна адреса електронної пошти;

д) login_time — час останнього успішного входу;

е) logout_time — час виходу;

ж) role — роль користувача в системі, що визначає набір дозволів;

з) status — статус акаунта (активний, заблокований тощо);

и) timezone — часовий пояс, який використовується для формування подій і журналів активності.

Клас реалізує інтерфейс UserDetails, тому містить методи перевірки стану облікового запису, що використовуються Spring Security для визначення активності, блокування, строку дії облікових даних.

Додаткові методи getLoginTime() та getLogoutTime() відповідають за коректну JSON-серіалізацію часових значень. Інші методи (гетери, сетери, конструктори) автоматично генеруються анотаціями Lombok.

Клас Product представляє сутність товару, який може зберігатися на складах, мати характеристики, фотографії та пов’язані теги.

Атрибути класу:

а) id — унікальний ідентифікатор товару;

б) code — незмінний унікальний код товару, що використовується як зовнішній ключ у пов’язаних таблицях;

в) title — назва товару;

г) description — текстовий опис;

д) price — ціна у вигляді BigInteger для уникнення похибок;

е) currency — валюта ціни;

ж) tags — список тегів, пов’язаних через таблицю product_tags за принципом many-to-many;

з) photos — список фотографій товару;

и) created_at — дата й час створення запису;

к) updated_at — дата оновлення;

л) weight, length, width, height — вагово-габаритні характеристики, що

дозволяють будувати логістичні моделі розміщення вантажів.

Усі допоміжні методи даються анотаціями Lombok, а зв'язки дозволяють гнучко отримувати інформацію про товар разом із тегами та зображеннями.

Клас StockItem представляє сутність, що описує конкретну партію товару на певному складі, її кількісні характеристики, стан та просторове розміщення.

Атрибути класу:

а) id — унікальний ідентифікатор запису;

б) batch_version — версія партії, яка використовується для контролю оптимістичної блокування або синхронізації;

в) code — унікальний код одиниці зберігання;

г) product — пов’язаний товар, для якого створено цей запис;

д) group_id / stockItemGroup — група товарів (наприклад, партія поставки);

е) storage_section — секція складу, де фізично розташовано товар;

ж) product_id — зовнішній ключ на товар;

з) warehouse_id — ідентифікатор складу, до якого належить партія;

и) expiry_date — дата придатності, що використовується для ротації товарів;

к) available_quantity — кількість доступних одиниць;

л) status — статус товарної позиції (в наявності, списано, заблоковано тощо);

м) is_active — ознака активності запису;

н) storage_section_id — зовнішній ключ секції зберігання.

Завдяки використанню зв’язків ManyToOne сутність дозволяє легко отримати всю пов’язану інформацію про товар, групу та розміщення. Lombok забезпечує генерацію стандартних методів, а сама сутність є основою для інвентаризації, синхронізації стоків та побудови складської аналітики.

Ці сутності описують головні елементи інформації у системі, окрім цього ще існує ряд допоміжних сутностей, призначених для підвищення комфорту використання, покращення логістичних процесів або ж розширення функціоналу як такого. Загальну діаграму класів зображено у додатку Ж.

4.3.2 Серверна частина системи

Реалізація серверної частини системи це монолітний застосунок, побудований за принципами багаторівневої архітектури, що забезпечує чітке розділення відповідальності та спрощує масштабування і підтримку коду. Логіка оброблення запитів структурована на три основні рівні: контролери, сервіси та репозиторії. Контролери відповідають за прийом і маршрутизацію HTTP-запитів, сервіси містять бізнес-логіку та правила оброблення даних, а репозиторії забезпечують доступ до бази даних та роботу з JPA-сутностями. Такий підхід дозволяє підтримувати чисту структуру коду, спрощує тестування, підвищує надійність і забезпечує можливість розширення функціоналу без порушення існуючих модулів.

Так як весь застосунок реалізовано на основі Spring Framework, кожен діючий елемент серверної частини є Spring Bean. Framework ініціалізує об’єкти з існуючих класів, що мають відповідні анотації @Service, @Component, @Configuration та інші і зберігає їх у контекст, після чого вони можуть бути перевикористані за принципом інверсії контролю.

Додатково в архітектурі серверної частини передбачено механізми валідації вхідних даних, оброблення винятків та централізоване логування запитів, що підвищує стабільність роботи системи та полегшує діагностику можливих помилок. Взаємодія між рівнями здійснюється через чітко визначені інтерфейси, що мінімізує залежності та забезпечує гнучкість у разі зміни логіки або заміни окремих компонентів. Особлива увага приділяється реалізації механізмів транзакційності, які гарантують цілісність даних під час виконання складських операцій та синхронізаційних процесів.

Окремим елементом реалізації є підтримка оброблення офлайн-операцій, які після відновлення мережевого з’єднання надходять на сервер для виконання та фіксації у базі даних. Для таких випадків забезпечено черговість виконання, перевірку коректності даних та механізми запобігання конфліктам, що можуть виникнути при паралельній роботі кількох клієнтів.

Приклад конфігураційного класу в контексті Spring Framework зображено на рисунку 4.2.

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(securedEnabled = true)
@RequiredArgsConstructor
public class ApplicationSecurityUsersConfiguration {
    private final AuthenticationEntryPoint authenticationFailureEntryPoint;
    private final AuthenticationLogoutSecurityHandler authenticationLogoutSecurityHandler;
    private final AuthorizationTokenRequestFilter authorizationTokenRequestFilter;
    private final PreLogoutTokenBasedFilter preLogoutTokenBasedFilter;
    private final UserDetailsSecurityService userDetailsSecurityService;
    private final AuthenticationManager authenticationManager;

    @Bean
    public SecurityFilterChain apiSecurityFilterChain(HttpSecurity httpSecurity) throws Exception {
        return httpSecurity
            .securityMatcher("/api/**", "/logout")
            .csrf(AbstractHttpConfigurer::disable)
            .formLogin(AbstractHttpConfigurer::disable)
            .cors(AbstractHttpConfigurer::disable)
            .sessionManagement(session - > session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .addFilterBefore(authorizationTokenRequestFilter, UsernamePasswordAuthenticationFilter.class)
            .addFilterBefore(preLogoutTokenBasedFilter, LogoutFilter.class)
            .authorizeHttpRequests(authentication - > authentication.requestMatchers("/api/**", "/logout").authenticated().anyRequest().denyAll())
            .logout(logout - > logout.logoutRequestMatcher(request - >
                    request.getRequestURI().equals("/logout") && request.getMethod().equals(HttpMethod.POST.name()))
                .logoutSuccessHandler(authenticationLogoutSecurityHandler)
                .invalidateHttpSession(true)
                .clearAuthentication(true)
                .deleteCookies("JSESSIONID", AbstractRememberMeServices.SPRING_SECURITY_REMEMBER_ME_COOKIE_KEY))
            .exceptionHandling(exceptionHandlingConfigurer - > exceptionHandlingConfigurer.authenticationEntryPoint(authenticationFailureEntryPoint))
            .userDetailsService(userDetailsSecurityService)
            .authenticationManager(authenticationManager)
            .build();
    }
}
```

Рисунок 4.2 — Загальна конфігурація безпеки

Шар контролерів передбачає аналогічне створення класів з відповідними компонентами, які дозволяють фреймворку ідентифікувати клас як контролер та зареєструвати його у ланцюжку роботи з диспетчером сервлетів, що відповідає за прийом вхідних запитів, використання фільтрів, переадресацію до відповідного методу контролеру та повернення відповіді.

Існує декілька видів контролерів, ті які повертають представлення у вигляді сторінки, вони ж звичайні контролери і найчастіше використовуються для систем з інтерфейсом, що генерується на стороні серверу. Це може бути Java Server Page, ресурси з HTML [17] сторінками, що обробляються з використанням шаблонізаторів.

Окремо в системі функціонують контролери, що забезпечують роботу з клієнтськими запитами та повертають дані у форматі JSON або XML, виконуючи роль повноцінних REST-контролерів. Вони відповідають за маршрутизацію запитів, формування відповідей та взаємодію із сервісним шаром.

На рівні з ними існує група додаткових, допоміжних контролерів зображено на рисунку 4.3, які можуть імітувати як стандартну REST-відповідь, так і альтернативні варіанти, залежно від контексту використання. Їх основною метою є оброблення помилкових або виняткових ситуацій: такі контролери перехоплюють помилки, формують структуровану та коректну відповідь для клієнта і не передають керування безпосередньо диспетчеру сервлетів, який за замовчуванням генерує стандартні HTML-сторінки помилок. Завдяки цьому система забезпечує єдиний формат повернення помилок, зберігає консистентність API та покращує керованість у випадку виняткових ситуацій.

```
@RestControllerAdvice
@Slf4j
public class ExceptionHandlerController {
    @ExceptionHandler(Exception.class)
    public ResponseEntity < ? > handleApplicationExceptions(Exception e) {
        log.error("Exception {} {}", e.getClass().getSimpleName(), e.getMessage(), e);

        if (e instanceof ApplicationException exception) {
            return ResponseEntity.status(exception.getStatus()).body(exception);
        } else if (e instanceof MethodArgumentNotValidException exception) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body(new ApplicationException(exception.getBindingResult().getFieldErrors().stream()
                    .map(DefaultMessageSourceResolvable::getDefaultMessage)
                    .filter(Objects::nonNull)
                    .toList()
                    .toString(), HttpStatus.BAD_REQUEST));
        } else if (e instanceof BadCredentialsException) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(new ApplicationAuthenticationException(e.getMessage()));
        } else if (e instanceof ValidationException ||
            e instanceof MissingServletRequestParameterException ||
            e instanceof MissingServletRequestPartException ||
            e instanceof HttpMessageNotReadableException || e instanceof HandlerMethodValidationException) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body(new ApplicationException(e.getMessage(), HttpStatus.BAD_REQUEST));
        } else if (e instanceof AccessDeniedException || e instanceof AuthorizationDeniedException) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN)
                .body(new ApplicationException(e.getMessage(), HttpStatus.FORBIDDEN));
        }

        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body(new ApplicationException(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR));
    }
}
```

Рисунок 4.3 — Зразок допоміжного контролера обробки

Розглянемо розроблені сервіси більш детально. Кожен з них має власний цикл існування у застосунку, який можна визначити за допомогою анотацій. За замовчуванням сервіси ініціалізуються та інстанціюються лише один раз на весь життєвий цикл контексту за допомогою Spring Framework. Вони можуть використовувати інші залежності з контексту фреймворку, якщо їм потрібна

логіка зосереджена в них. Наприклад, UserService відповідає за повний спектр операцій, пов’язаних з керуванням користувачами системи, включаючи пошук, створення, оновлення, валідацію даних та перевірку прав доступу, поєднує бізнес-логіку, безпеку та валідацію, формуючи єдиний центр оброблення всіх операцій користувацького домену і зображено на рисунку 4.4.

Інші сервіси працюють за аналогічним принципом, кожен реалізуючи власний набір операцій для відповідної доменної сутності та забезпечуючи ізольоване виконання бізнес-логіки на своєму рівні.

```
@Service
@Slf4j
@Validated
@RequiredArgsConstructor
public class UserService {
    private final UserRepository userRepository;
    private final UserMapper userMapper;
    private final EntityManager entityManager;
    private final PasswordEncoder passwordEncoder;
    private final FieldValidator fieldValidator;

    public static void assertAuthenticatedUserRoles(List < UserRole > roles) {
        getCurrentlyAuthenticatedUser().filter(user - > roles.contains(user.getRole()))
            .orElseThrow(() - > new ApplicationAuthenticationException(("User role has to be one of [%s]").formatted(roles)));
    }

    public static Optional < User > getCurrentlyAuthenticatedUser() {
        SecurityContext securityContext = SecurityContextHolder.getContext();
        if (securityContext != null) {
            Authentication authentication = securityContext.getAuthentication();

            if (authentication != null && authentication.isAuthenticated()) {
                if (authentication.getPrincipal() instanceof User) {
                    return Optional.ofNullable((User) securityContext.getAuthentication().getPrincipal());
                } else {
                    log.warn("Authentication principal is not of type RegularUser.class");
                }
            }
        }

        return Optional.empty();
    }
```

Рисунок 4.4 — Бізнес-логіка роботи з користувачами

Окрім контролерів та сервісів необхідно мати можливість працювати зі сховищем даних, для цього існують репозиторії. Аналогічно до решти їх цикл існування залежить від Spring Framework, він їх інстанціює та налаштовує. Репозиторії мають набір функціоналу за замовчуванням для збереження та отримання записів, у випадку потреби можуть бути розширені зображено на рисунку 4.5.

Кожна сутність системи має свій окремий репозиторій, що дає змогу чітко

розмежувати доступ до таблиць та підтримувати безпечну роботу з даними. Репозиторії базуються на інтерфейсах Spring Data JPA і завдяки цьому не потребують явної реалізації — стандартні методи для створення, пошуку, оновлення та видалення записів генеруються автоматично. У випадках, коли необхідна складніша логіка вибірки або специфічні умови фільтрації, у репозиторіях можна оголошувати додаткові методи, назва яких сама визначає параметри SQL-запиту, або використовувати анотацію @Query для ручного формування запитів. Така модель роботи дозволяє зменшити обсяг шаблонного коду, підвищує гнучкість при побудові запитів та забезпечує ізоляцію доступу до бази даних від інших шарів застосунку.

```
public interface UserRepository extends JpaRepository < User, Long > {
    User findUserByUsername(String username);

    User findUserByEmail(String email);

    List < User > findUsersByRole(UserRole role, Pageable pageable);

    List < User > findUsersByStatus(UserStatus status, Pageable pageable);

    @Query("UPDATE User u SET u.loginTime = CURRENT_TIMESTAMP WHERE u.id = :id")
    @Modifying
    @Transactional
    void updateLoginTime(@Param("id") Long id);

    @Query("UPDATE User u SET u.logoutTime = CURRENT_TIMESTAMP WHERE u.id = :id")
    @Modifying
    @Transactional
    void updateLogoutTime(@Param("id") Long id);

    boolean existsByUsername(String username);

    boolean existsByEmail(String email);
}
```

Рисунок 4.5 — Репозиторій роботи з користувачами

Також важливою складовою серверної частини є конфігурація застосунку, що визначається у вигляді властивостей і підтримує використання змінних оточення . Такий підхід дає змогу гнучко налаштовувати поведінку системи залежно від середовища виконання — локальної розробки, тестового стенду чи продуктивного сервера — без зміни коду. Змінні оточення дозволяють безпечно передавати вразливі параметри, зокрема налаштування підключення до бази

даних, секретні ключі, параметри безпеки та інші конфіденційні значення. Конфігураційні файли містять загальні параметри роботи застосунку та можуть бути розширені окремими профілями, що забезпечує чітку сегментацію налаштувань для різних сценаріїв запуску.

Конфігураційні файли можуть мати кілька варіантів залежно від середовища, у якому виконується система, що дозволяє розподіляти налаштування між окремими профілями. Spring підтримує механізм профілів, завдяки якому для кожного середовища створюється свій файл властивостей.

Кожен із них може містити власні параметри бази даних, логування, безпеки або інтеграцій з іншими сервісами. Завдяки цьому застосунок може автоматично завантажувати потрібні конфігурації залежно від активного профілю, що робить систему гнучкою, безпечною та зручною для розгортання у різних умовах.

Загалом серверна частина системи формує надійну основу її функціонування, забезпечуючи чітке розподілення відповідальностей між окремими компонентами, безпечну обробку даних та узгоджену роботу всіх механізмів. Завдяки багаторівневій архітектурі, використанню конфігураційних профілів та інтеграції зі Spring Framework серверна логіка залишається гнучкою, масштабованою та простою в супроводі.

Така структура дозволяє системі ефективно реагувати на зміни вимог, легко розширювати функціонал і підтримувати стабільну роботу як у звичайному, так і в навантаженому середовищі. Це створює стабільний фундамент для взаємодії з клієнтською частиною та гарантує узгодженість усіх операцій, що виконуються всередині системи.

4.3.3 Застосунок

Клієнтський застосунок реалізовано з використанням бібліотеки React, яка забезпечує компонентний підхід до побудови інтерфейсу та дає можливість ефективно керувати станом, маршрутизацією та відображенням даних. Архітектура фронтенду складається з окремих функціональних компонентів,

сервісів для роботи з API та допоміжних класів, що інкапсулюють логіку оброблення даних.

Маршрутизація організована через спеціальний роутер, який відповідає за переходи між сторінками та підвантаження відповідних компонентів інтерфейсу. У цьому підрозділі буде розглянуто структуру застосунку, приклади типових сервісів та компонентів, а також механізми взаємодії з серверною частиною системи.

Розглянемо компонент роутингу застосунку, він відповідає за переадресацію користувача залежно від URL та прав на відповідну сторінку.

Роутинг клієнтського застосунку побудований на основі бібліотеки React Router та забезпечує навігацію між сторінками інтерфейсу. Кожен маршрут відповідає окремому компоненту, а захищені сторінки обгорнуті у спеціальний компонент ProtectedRoute, що перевіряє автентифікацію користувача. Такий підхід дозволяє розмежувати доступ та формувати структуру застосунку у вигляді чітко визначених шляхів, що полегшує масштабування та підтримку інтерфейсу.

Сервіси клієнтської частини інкапсулюють логіку взаємодії з REST-API та дозволяють централізовано виконувати запити до серверної частини. У сервісах зосереджена логіка фільтрації, форматування параметрів, оброблення помилок та підтримка роботи в офлайн-режимі. Завдяки цьому компоненти інтерфейсу залишаються максимально чистими й відповідають лише за відображення даних, тоді як бізнес-логіка виконується на рівні сервісів.

Конфігураційні файли клієнтського застосунку використовуються для розділення налаштувань між різними середовищами, такими як локальна розробка, тестування чи продакшн. Через них визначаються адреси серверів, порти, параметри роботи з кешем та інші змінні, які можуть відрізнятися залежно від умов виконання. Така структура дозволяє легко перемикатися між середовищами та забезпечує гнучкість під час розгортання застосунку.

Для демонстрації роботи клієнтської частини розглянемо декілька прикладів коду, що відображають ключові елементи застосунку. Зокрема, роутера, сервісів та конфігураційних файлів, які показують підхід до організації навігації, взаємодії

з API та налаштування середовища виконання.

Роутинг виконує функцію навігації. Розглянемо зразок сервісів, кожен з яких відповідальний за роботу з тією чи іншою логічною сутністю системи. Цей сервіс відповідає за виконання запитів на сервер для авторизації.

Конфігураційні модулі клієнтського застосунку використовуються для централізованого зберігання параметрів, що визначають поведінку інтерфейсу та правила взаємодії з іншими частинами системи. У конфігураціях можуть міститися адреси серверних точок доступу, налаштування кешування, параметри відображення, формати дат, правила оброблення помилок та інші значення, що використовуються багатьма компонентами.

Винесення цих параметрів у окремі конфіг-файли дозволяє уникнути дублювання інформації, спрощує модифікацію логіки та забезпечує кращу структурованість коду. Завдяки цьому застосунок залишається легким у підтримці, а всі важливі налаштування зосереджені у чітко визначеному місці, що спрощує його розширення та адаптацію.

У підсумку клієнтський застосунок демонструє структурований підхід до побудови інтерфейсу, де компоненти, сервіси та конфігурації працюють узгоджено та доповнюють один одного. Завдяки використанню React і чіткій організації коду забезпечено зрозумілу навігацію, зручну взаємодію з сервером і гнучкість у підтримці та подальшому розширенні функціоналу системи.

4.4 Розробка користувацького інтерфейсу

Розробка користувацького інтерфейсу охоплює створення візуальної частини застосунку, що формує зовнішній вигляд і взаємодію користувача з системою. На цьому етапі визначаються структура сторінок, розташування елементів, стильове оформлення та логіка відображення інформації. Візуальна реалізація має забезпечувати зручність роботи, відповідність функціональним вимогам і послідовність у поданні даних. У цьому розділі подано підходи та рішення, що були використані для побудови інтерфейсу, а також приклади

основних екранних форм.

4.4.1 Ініціалізація конфігурації для роботи з системою

Щоб користуватися системою необхідно попередньо створити користувача з роллю власника, так як реєстрація нових користувачів доступна лише безпосередньо у системі. Це реалізовано на стороні сервера інструментами міграції баз даних, якщо в базі відсутній користувач з роллю власника, його буде створено автоматично. Для неавторизованих користувачів доступна лише сторінка входу в систему. Окрім існування користувача, необхідно також внести адресу сервера та інші конфігураційні дані. Це необхідно для надсилання основних запитів, перевірки стану серверу, валідації авторизаційної інформації та ініціації сервісних запитів до сервера.

4.4.2 Авторизація користувача у системі

Для доступу до системи кожен користувач повинен авторизуватися, це реалізовано за допомогою сервісу та сторінки авторизації. Сторінка авторизації містить мінімальний набір елементів зображена на рисунку 4.6, необхідних для входу до системи, такі як поля введення імені користувача та пароля, кнопку підтвердження та механізми відображення можливих помилок. Інтерфейс побудований таким чином, щоб забезпечити простоту та швидкість виконання операції входу, а також коректно обробляти як успішні, так і неуспішні спроби авторизації. При введенні некоректних даних користувач отримує відповідне повідомлення, що дозволяє швидко виправити помилку та повторити спробу входу.

Після успішного введення облікових даних система виконує їх перевірку на серверній стороні, де реалізовано механізм автентифікації на основі JWT-токенів. У разі підтвердження достовірності даних користувачу видається маркер доступу, що використовується для подальших запитів до захищених ресурсів системи. Це

забезпечує розмежування прав доступу та підвищує загальний рівень безпеки.

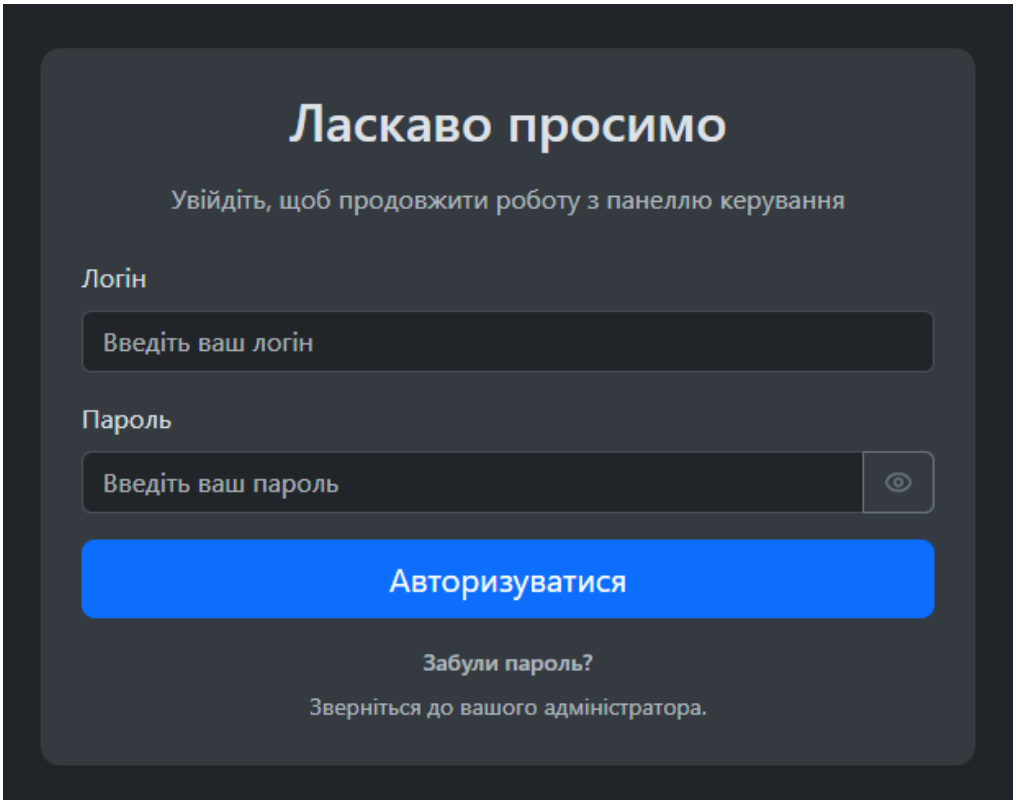


Рисунок 4.6 — Сторінка авторизації

У разі успішного входу система виконує ряд запитів для отримання необхідної для функціонування інформації.

4.4.3 Інтерфейс застосунку під час роботи

Якщо користувач успішно авторизується у системі, то автоматично відбувається переадресація на сторінку особистого кабінета зображеного на рисунку 4.7. Ця сторінка відображає інформацію про користувача, надає можливість зміни паролю та демонструє навігаційне меню для переходу між різними сторінками застосунку, використано принцип групування за логічним сенсом. Кожен пункт навігаційного меню може мати дочірні пункти, як наприклад ті, що відносяться до товарів, або складів, чи фінансового обігу, аналітики, тощо. Також надається доступ до виходу з системи. У цьому разі очищуються службові дані на стороні клієнта, виконується запит на сторону сервера для фіксування

виходу з системи та деактивації JWT-токену і повертається керування до сторінки авторизації.

Інтерфейс розроблено максимально інтуїтивним та простим, водночас інформативним, щоб задовольнити всі потреби кінцевого користувача інформаційної системи автоматизації роботи зі складською логістикою та аналітикою і підтримкою офлайн-функціональності.

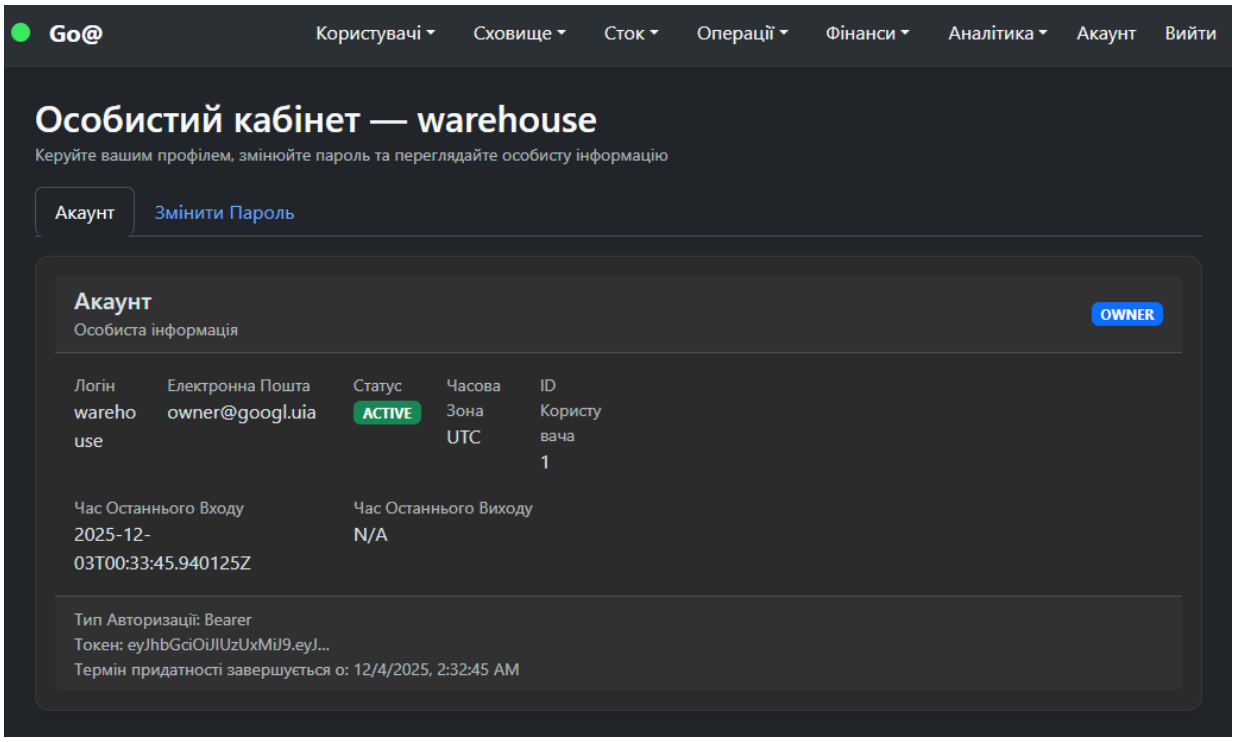


Рисунок 4.7 — Особистий кабінет користувача

Розглянемо роботу зі сторінкою керування користувачами зображеної на рисунку 4.8, доступ до перегляду користувачів мають всі, водночас створювати нових або оновлювати існуючих лише власник та менеджер. Особливо важливо врахувати логіку роботи з ролями, власник може створювати та оновлювати всіх користувачів, менеджер у свою чергу лише операторів, оператори можуть лише переглядати користувачів системи.

На цій сторінці як і на інших знаходиться доступ до навігаційного меню, форма для фільтрації та пошуку, де можна фільтрувати користувача за іменем, електронною поштою, ролями, статусами та активністю в системі на даний

момент.

Також є форма для створення користувачів, вона вимагає введення логіну, електронної пошти, ролі, статусу, часового поясу і паролю. Ця ж форма дозволяє редагувати користувачів, для цього необхідно вибрати користувача зі списку та натиснути кнопку редагування.

Кожне поле має валідації узгоджені з логікою серверної частини, обмеження довжини логіну та паролю, формат електронної пошти, у випадку порушення цих правил ініціація запита буде заблокована, а відповідні проблеми буде відображено на сторінці для користувача, з очікуванням їх виправлення.

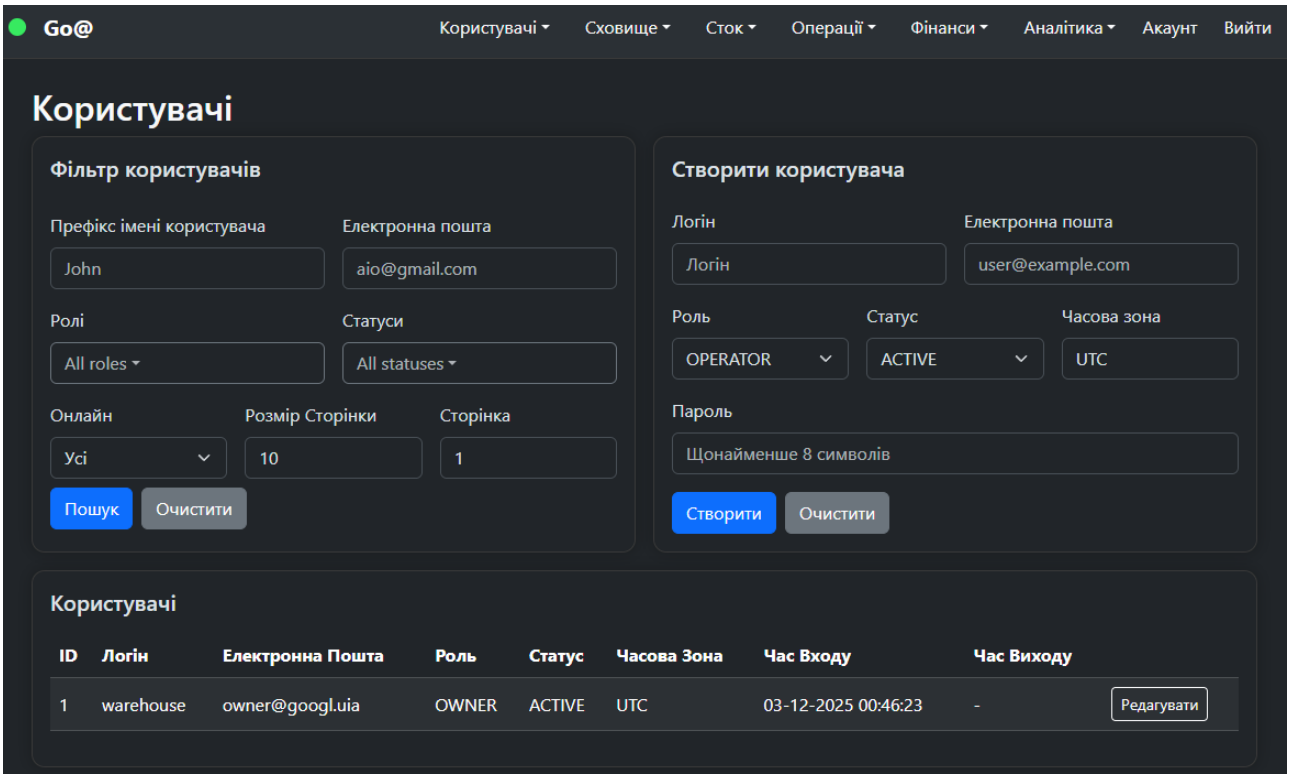


Рисунок 4.8 — Керування користувачами

Розглянемо сторінку роботи зі складами 4.9.

Ця сторінка як і всі інші зберігають загальну стилістику: окрема форма для пошуку, окрема для створення та редагування і під всіма формами відображення списку сутностей, у даному випадку складів.

Багато сутностей мають велику кількість атрибутів, через це реалізовано функцію модального вікна для перегляду конкретного елементу, як і на цій

сторінці зображеній на рисунку 4.10, щоб переглянути детальні необхідно натиснути на відповідний рядок у списку і вікно з’явиться автоматично у верхній частині сторінки. Додатково сторінка забезпечує навігацію по записах та валідацію введених даних, що спрощує роботу зі складською структурою навіть при великій кількості елементів.

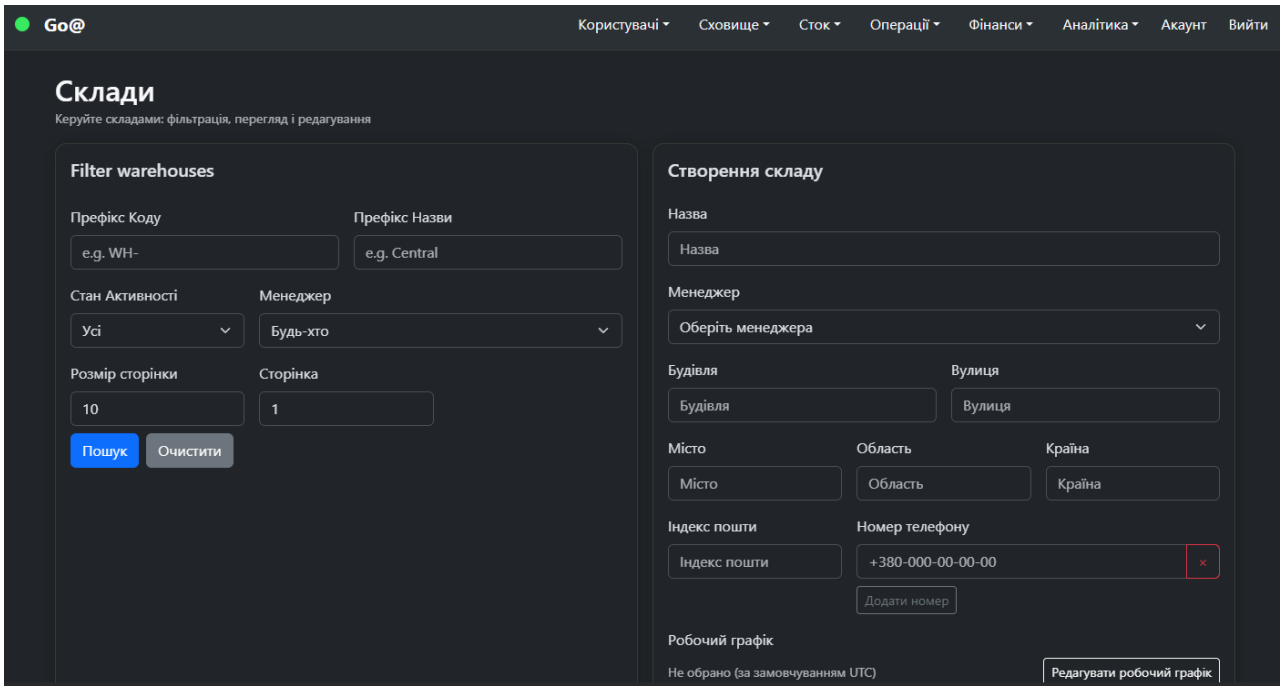


Рисунок 4.10 — Робота зі складами

Завдяки модальному вікну зображеному на рисунку 4.11, можна оперативно переглядати повну інформацію про вибраний склад без необхідності переходу на інші сторінки або повторного пошуку, що значно підвищує зручність взаємодії та покращує досвід роботи користувача з інтерфейсом.

Додатково, модальний формат забезпечує фокусоване відображення даних, мінімізуючи візуальний шум і дозволяючи користувачу швидко приймати рішення на основі актуальної інформації.

Це сприяє більш ефективному виконанню операцій та знижує ризик помилок під час роботи з великою кількістю складських записів. Також використання модального вікна дозволяє легко розширювати функціональність — наприклад, додавати швидке редагування, перегляд історії змін чи виконання

службових операцій без перезавантаження основної сторінки.

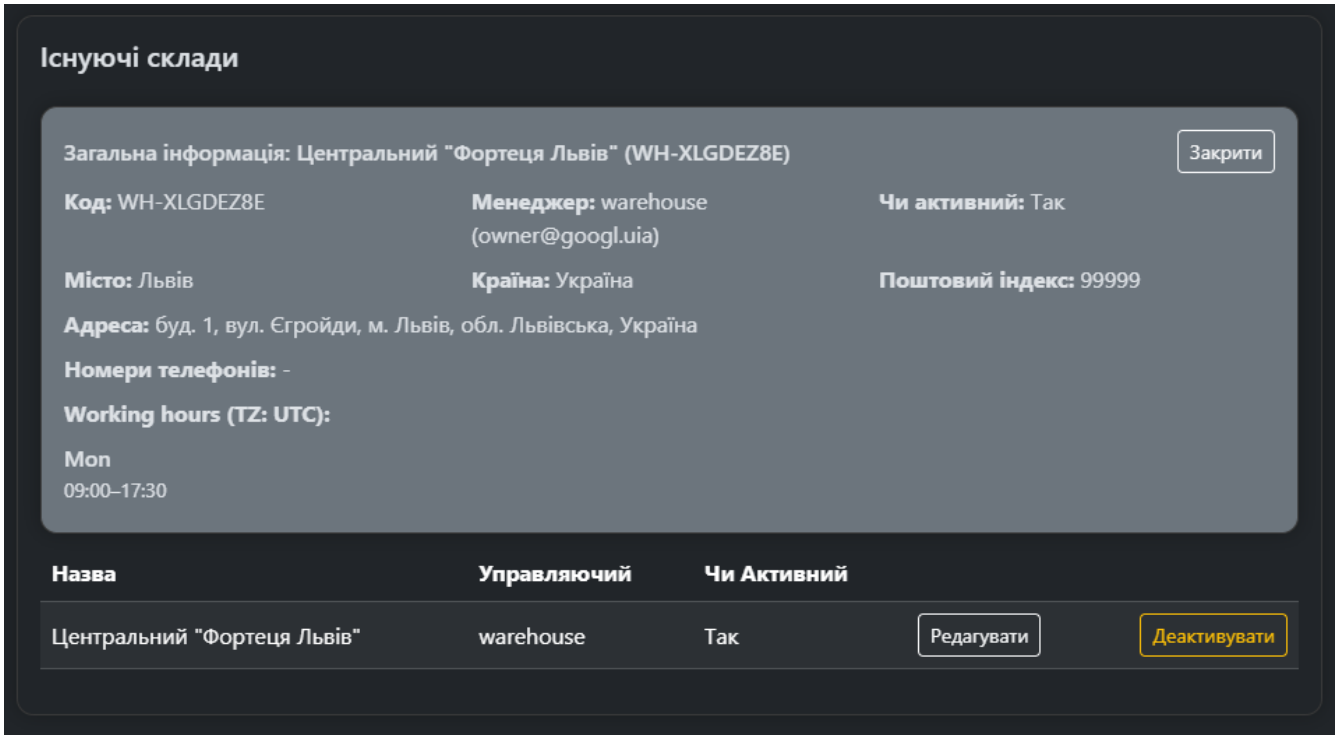


Рисунок 4.11 — Інформація про склади

Щоб покращити логістичну організацію товарів на складах запроваджено окрему сутність, що відображає секції зберігання, кожна секція має власний унікальний ідентифікатор і належить конкретному складу, кожне підприємство може запровадити власні правила вибору тієї чи іншої секції для товару, хоч за розміром, хоч за ціною, ці регуляції варто винести до окремого довідника для конкретного складу, або ж логістичного ланцюга.

Ця сторінка надає можливість перегляду, створення та редагування секцій, водночас склад як і товар не обов’язково має секції, зображено на рисунку 4.12.

Ці секції є опціональними та призначені для покращення основних можливостей підприємства, що працює з системою.

Наприклад для сортування, або ж інтеграції з поштовими відділеннями де така практика є дуже поширеною, тощо.

Крім того, секції можуть використовуватися для більш точного зонування простору складу, що полегшує навігацію персоналу та прискорює виконання

логістичних операцій. У результаті підприємство отримує більш впорядковану логістичну інфраструктуру, що сприяє підвищенню продуктивності та якості роботи персоналу.

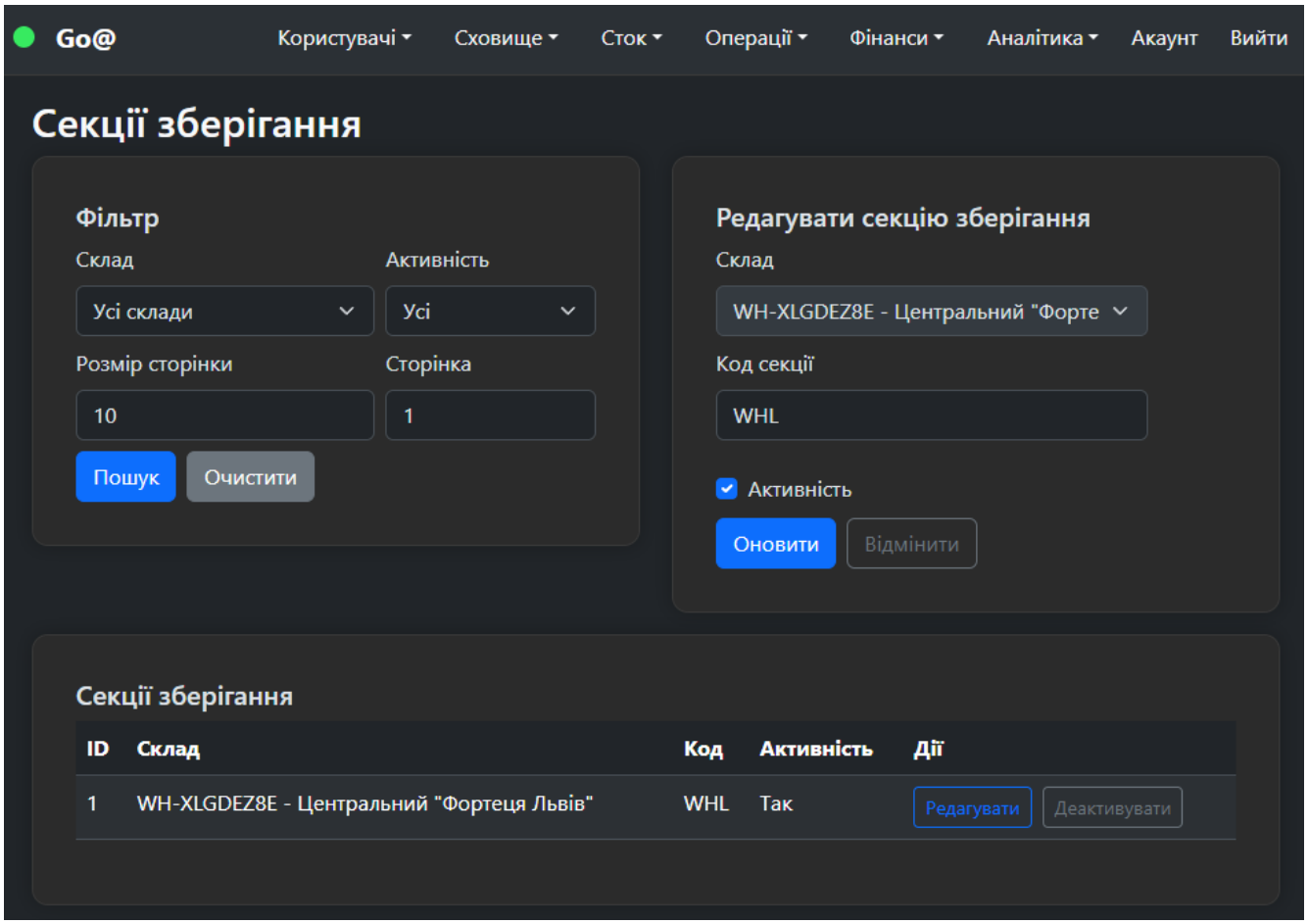


Рисунок 4.12 — Робота з секціями зберігання

Наступним є функціонал роботи з товарами, а саме створення груп товарів, товарів, тегів продуктів та самих продуктів і перегляд історії змін товарів, яка фіксується автоматично у разі відповідних змін. Розглянемо сторінку роботи з товарними групами зображено на рисунку 4.13. Як і решта вона має фільтрацію за параметрами, пагінацію та форму для створення і редагування груп товарів, змінювати можна лише назву, у даній сутності застосовано функціонал генерації унікального коду як і в більшості інших сутностей. Є валідація на полі для нової назви групи та індикація активності, якщо необхідно прибрати її з використання або ж деактивувати товари цієї групи.

Окрім базових операцій, сторінка забезпечує зручне відображення списку всіх груп товарів з можливістю швидкого пошуку потрібного запису. Усі зміни виконуються інтерактивно та одразу відображаються у таблиці, що дозволяє користувачу контролювати актуальний стан даних. Форма редагування побудована таким чином, щоб мінімізувати ризик помилок. Користувач може змінювати лише ті атрибути, які допускають модифікацію, тоді як код групи залишається незмінним для збереження цілісності.

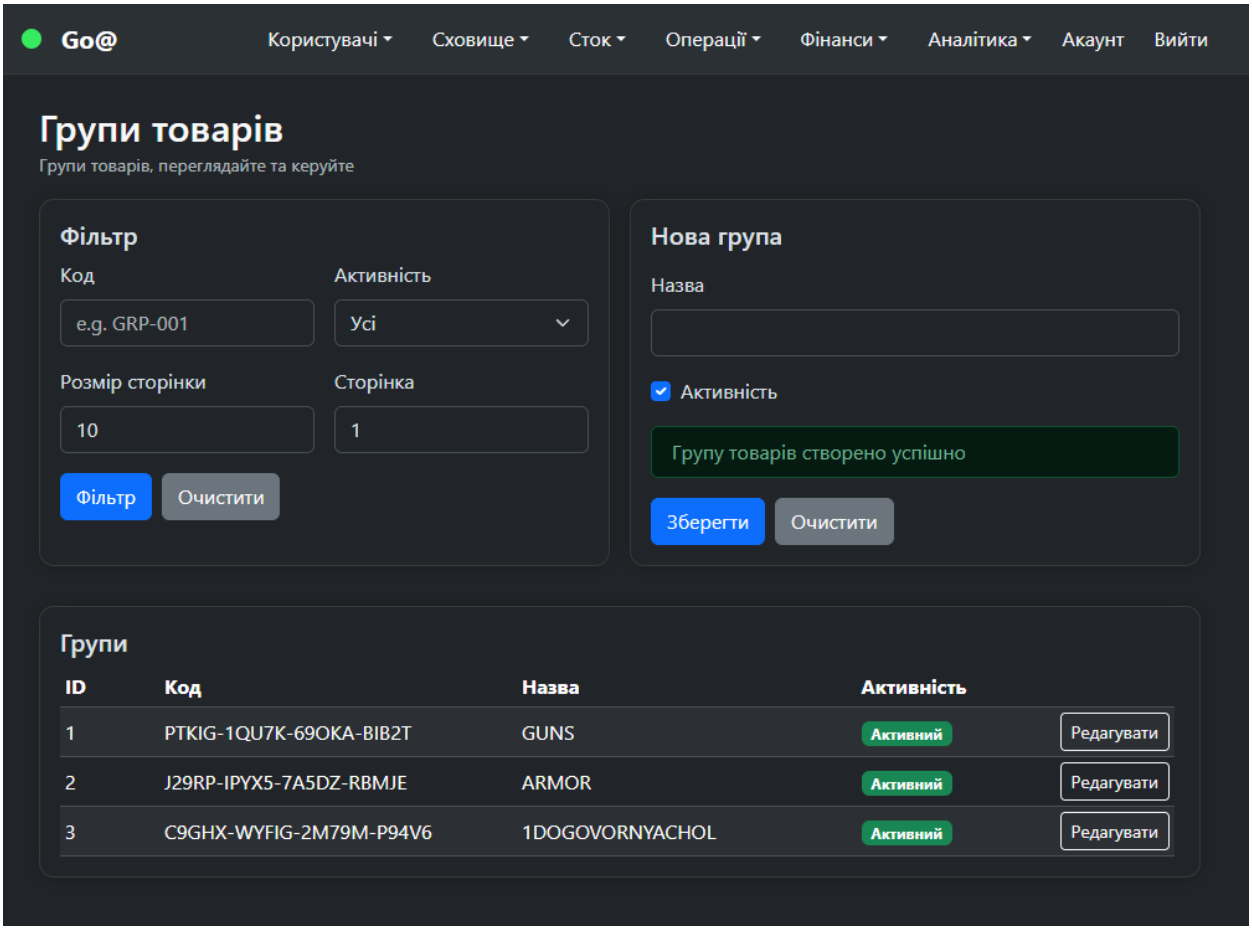


Рисунок 4.13 — Робота з групами товарів

Розглянемо сторінку роботи з тегами зображено на рисунку 4.14. На ній реалізовано повний набір інструментів для керування тегами продуктів: від фільтрації та пошуку до створення, редагування й підтримання актуальності набору тегів у системі. Форма фільтрації дає змогу обмежити вибірку за назвою тегу, його активністю, а також визначити параметри пагінації для зручного

перегляду великих списків.

Основний функціональний блок містить форму створення нового тегу та редагування існуючого, де користувач може змінювати назву, перемикати статус активності та отримувати повідомлення про успішні або помилкові операції. Важливим елементом є окремий механізм очищення невикористаних тегів, який автоматично видаляє теги, що не прив'язані до жодного продукту, тим самим підтримуючи довідник у впорядкованому стані та запобігаючи накопиченню непотрібних записів. Такий підхід забезпечує зручність роботи та дозволяє ефективно підтримувати структуру даних у модулі товарів.

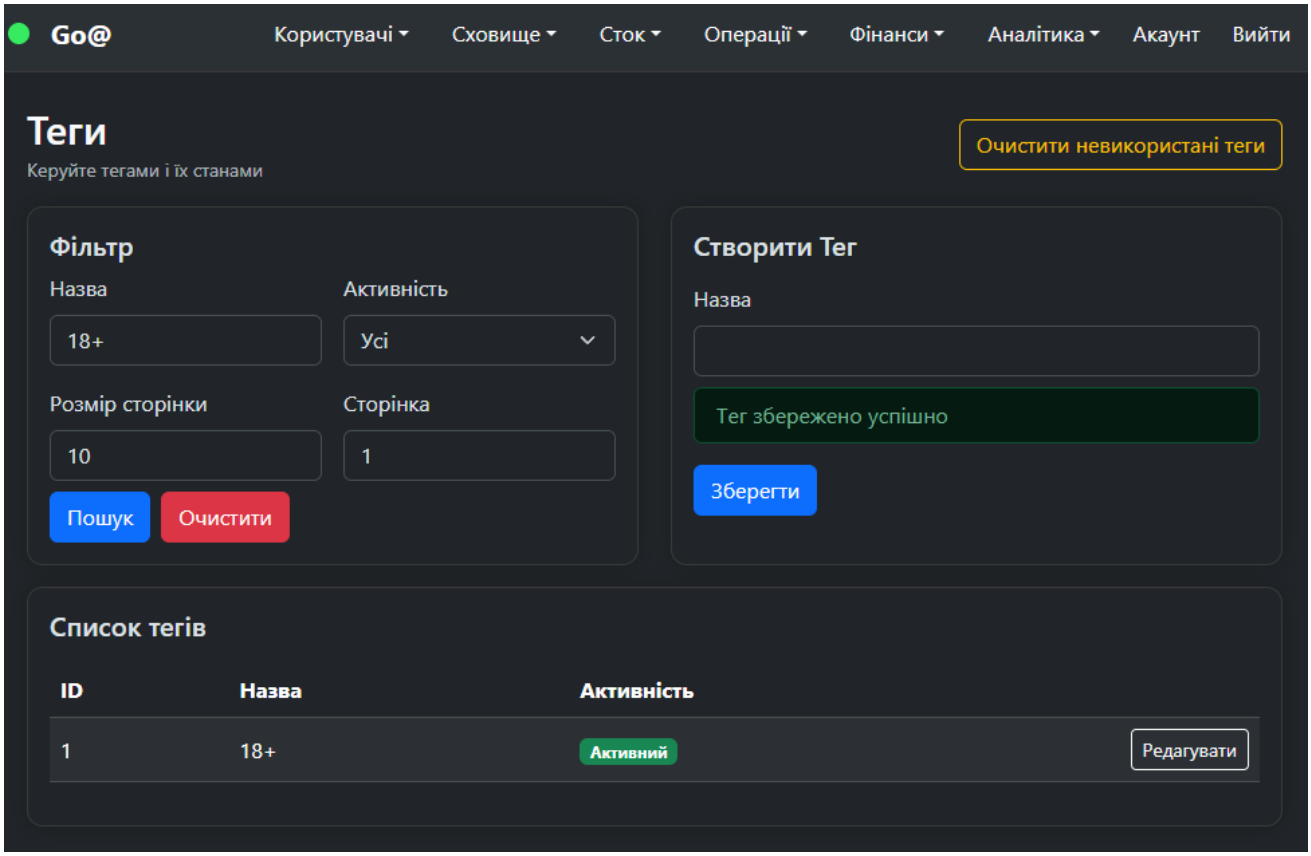


Рисунок 4.14 — Робота з тегами

Сторінка роботи з продуктами забезпечує повний цикл керування даними про товари, на основі яких у подальшому створюються товарні одиниці зображено на рисунку 4.15.

Інтерфейс підтримує фільтрацію за назвою, тегами, групою товару або активністю, а також має вбудовану пагінацію для зручного перегляду великої

кількості продуктів. Форма створення та редагування дозволяє вказувати назву, опис, ціну, валюту, вагово-габаритні параметри та прив’язувати теги й фотографії.

Генерація унікального коду продукту відбувається автоматично, що забезпечує уніфікацію даних та зручність пошуку.

Продукт виступає базовою одиницею, на основі якої формуються конкретні товарні позиції на складі, тому інтерфейс акцентує увагу на повноті та коректності його атрибутів. Крім того, продумана структура полів унеможливлює виникнення неузгодженостей між різними елементами системи, забезпечуючи стабільність роботи сервісу та точність подальших операцій із товарними позиціями.

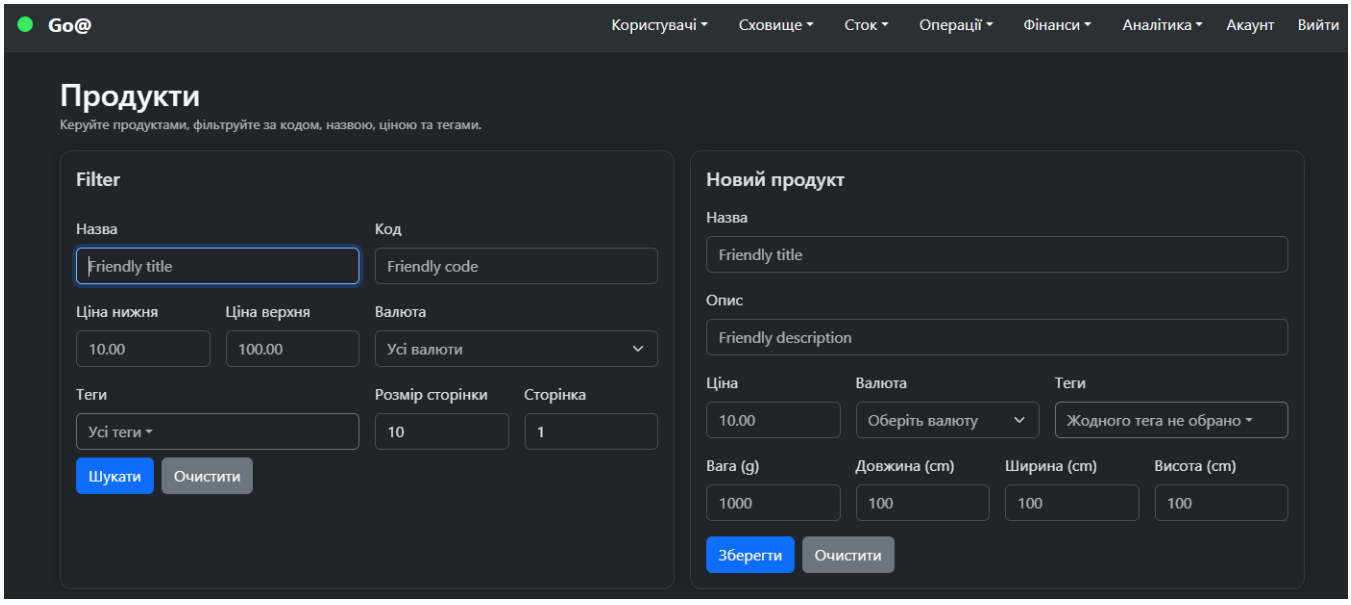


Рисунок 4.15 — Робота з продуктами

Робота з товарами реалізована на окремій сторінці, яка дозволяє формувати реальні складські позиції на основі вже створених продуктів зображено на рисунку 4.16. Для кожного товару обов’язково вказується група товару, склад, секція зберігання, кількість, статус і за потреби дата придатності.

Інтерфейс містить фільтри для швидкого пошуку товарних позицій за складом, кодом, статусом або активністю, а також підтримує пагінацію для навігації великими обсягами даних.

Створення та редагування товарів доступне через форму, яка динамічно

перевіряє введені значення, зберігає зв’язки з продуктом та групою і забезпечує відображення помилок або успішних операцій. При кожній зміні параметрів автоматично формується запис в історії змін товарів, що дозволяє відстежувати всі операції та забезпечує прозорість обліку на складі. Окрім базових операцій, сторінка також пропонує можливість перегляду детальної інформації про кожну товарну позицію за допомогою модального вікна, що відкривається при натисканні на відповідний рядок у списку.

Це дозволяє користувачу швидко оцінити ключові характеристики товару, включно з пов’язаним продуктом, історією змін та параметрами зберігання, не перемикаючись між окремими формами або вкладками. Такий підхід значно полегшує роботу з великою кількістю товарів і забезпечує швидкий доступ до всієї необхідної інформації в одному інтерфейсі.

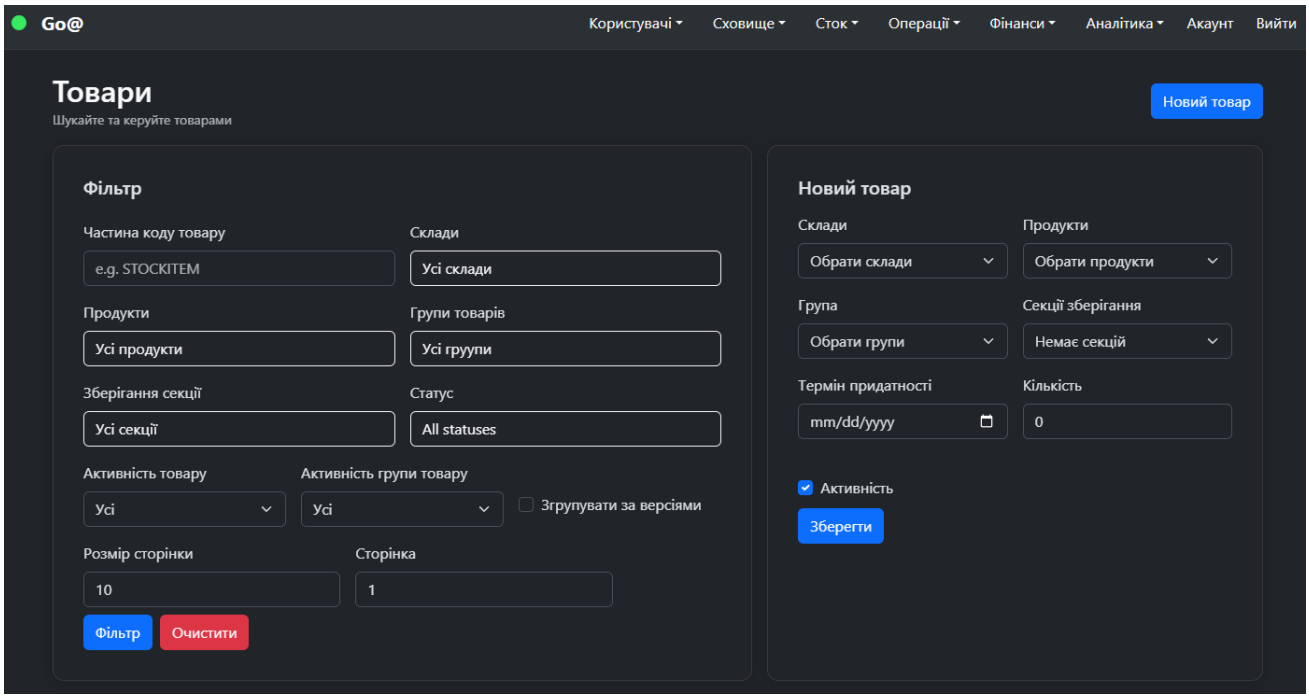


Рисунок 4.16 — Робота з товарами

І також розглянемо роботу з історією зміни товарів зображено на рисунку 4.17. Для отримання потрібних даних користувачу необхідно обрати конкретний товар і проміжок часу, за який потрібно відобразити історію, що дає змогу аналізувати дії у певному контексті та за конкретний період. Історія автоматично

групується за днями, що робить її структурованою та простою для перегляду навіть при великій кількості подій. Оскільки змін може бути багато, сторінка підтримує пагінацію з можливістю гнучко переглядати списки записів без втрати продуктивності.

Додатково інтерфейс дозволяє швидко визначити характер змін завдяки відображенню ключових атрибутів кожної операції — попереднє значення, нове значення, час фіксації та тип зміни. Такий підхід забезпечує повну прозорість складських процесів, дозволяє проводити аудит змін та оперативно реагувати на потенційні помилки або нестачі, що є критично важливим для підтримання точності інвентаризації у складській системі. Завдяки чіткому структуруванню записів користувач може швидко знаходити потрібні події та аналізувати причини зміни запасів.

Інтерфейс також передбачає відображення як ручних, так і автоматичних змін, що дозволяє повноцінно відслідковувати усі операції над товаром. У поєднанні з пошуком за датами така система історії забезпечує високий рівень контролю та надійності складського обліку.

Додатково механізм роботи з історією змін інтегрований із загальною системою аналітики, що дає змогу використовувати зібрані дані для виявлення тенденцій у роботі складу.

На основі накопичених подій можна аналізувати частоту та характер змін, визначати повторювані операції, оцінювати навантаження на окремі складські зони та виявляти можливі аномалії в процесах. Такий підхід підвищує прозорість операційної діяльності та дозволяє не лише фіксувати факти змін, але й формувати підґрунтя для подальшої оптимізації логістичних процесів. Завдяки цьому історія змін перетворюється на важливе джерело даних для ухвалення управлінських рішень.

Крім того, історія змін може використовуватися як інструмент контролю якості даних, оскільки дозволяє швидко виявити некоректні або підозрілі операції. Завдяки наочному відображенню подій користувачі можуть оперативно відстежувати відхилення та своєчасно коригувати записи, що сприяє підвищенню

точності складського обліку.

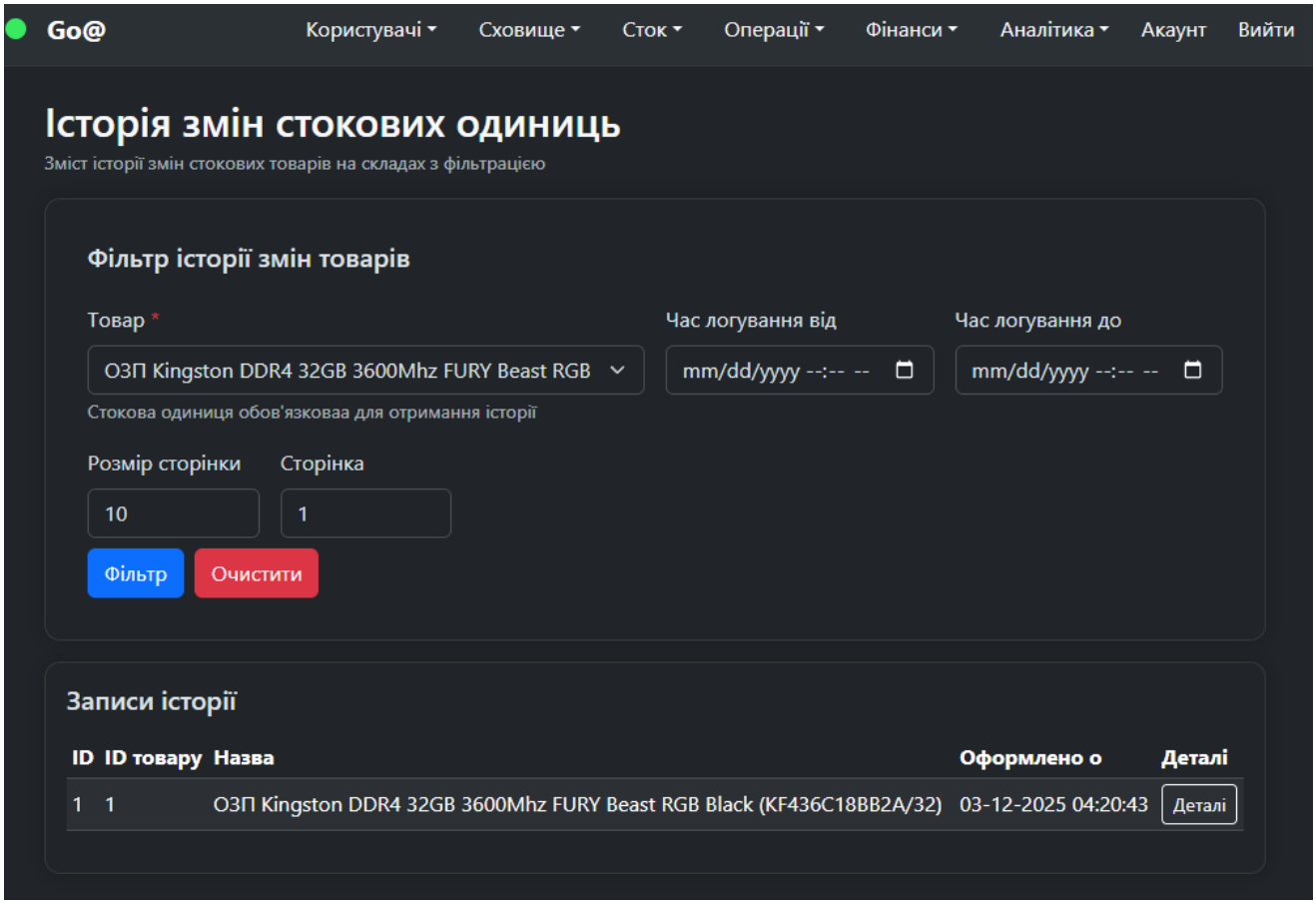


Рисунок 4.17 — Робота з історією товарів

Це є основні сторінки для роботи з системою логістики та обліку товарів на складах, які забезпечують повний цикл взаємодії з даними: від створення довідників і формування складських позицій до контролю стану запасів та відстеження історії змін.

Така структура дозволяє ефективно управляти товарними ресурсами, підтримувати актуальний стан складів і швидко виконувати ключові операції, необхідні для щоденної роботи персоналу.

Окрім базового функціоналу управління складською інфраструктурою, система включає модулі роботи з відправленнями зображено на рисунку 4.18, які дають змогу формувати нові, переглядати та контролювати існуючі переміщення товарів між складами чи зовнішніми точками з можливістю вказати точну адресу отримувача. Алгоритм роботи з цим функціоналом наведено у додатку И.

Інтерфейс підтримує фільтрацію, сортування, відображення деталей кожного відправлення та відстеження статусів, що підвищує прозорість логістичних процесів і зменшує кількість помилок при транспортуванні.

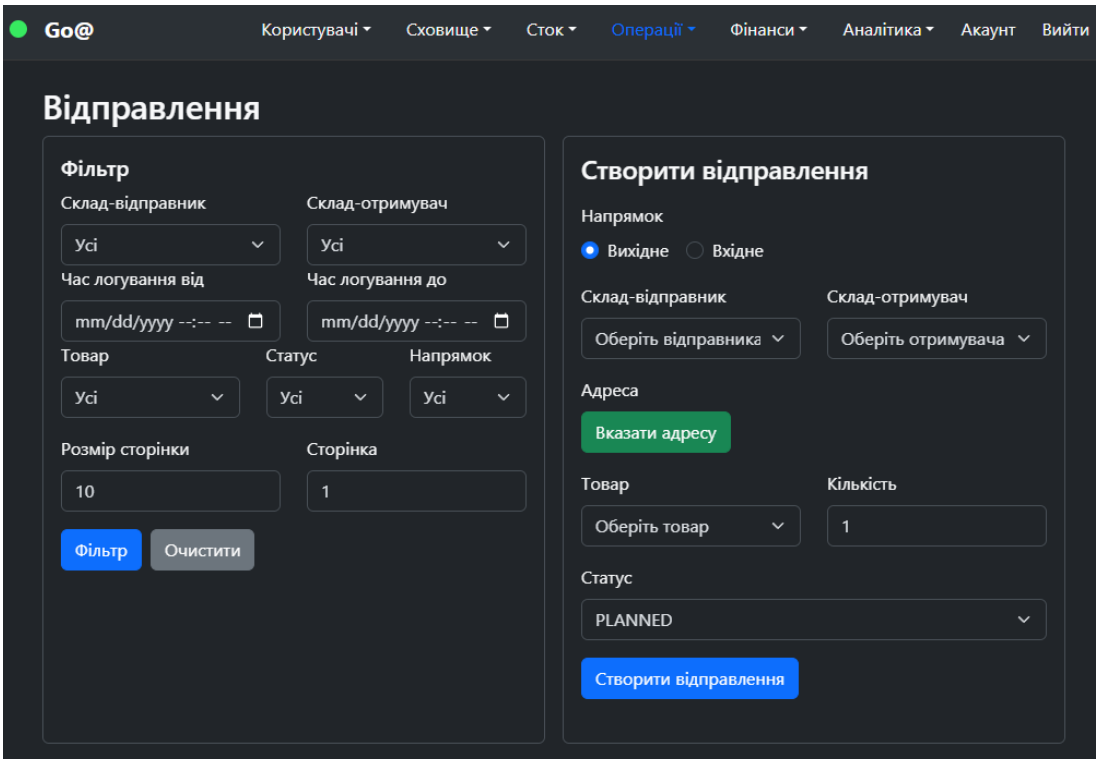


Рисунок 4.18 — Робота з відправленнями

Також реалізовано окремі інструменти управління фінансовим обігом та аналітикою, які дозволяють контролювати транзакції, курси валют, взаємодію з бенефіціарами та формувати зведені звіти. Аналітичні сторінки забезпечують узагальнення великих масивів даних у зручному форматі, що допомагає оперативно приймати управлінські рішення, виявляти тенденції й оптимізувати складські та логістичні процеси в межах всієї системи.

Окремо розглянемо сторінку присвячену роботі з бенефіціарами на рисунку 4.19, що дозволяє керувати реквізитами отримувачів коштів та пов’язаними фінансовими операціями. Сторінка містить фільтри для пошуку за ім’ям, IBAN, SWIFT-кодом та станом активності, а також форму для створення й редагування записів. Для кожного бенефіціара зберігаються детальні платіжні дані, включно з банківськими рахунками і картками, що забезпечує коректність і швидкість

проведення різних фінансових транзакцій.

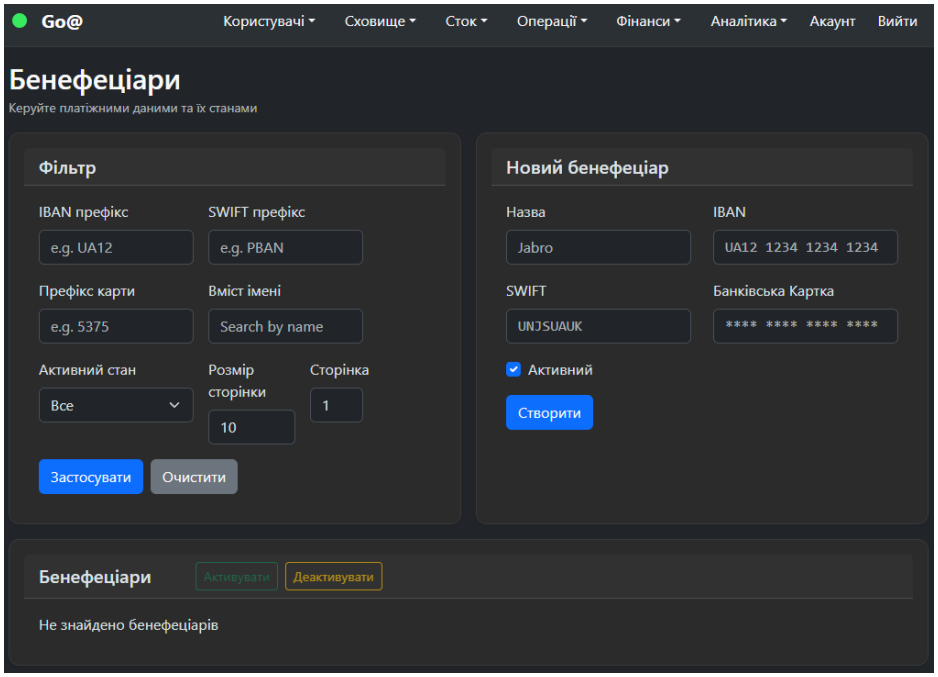


Рисунок 4.19 — Робота з бенефеціарами

Транзакції та відповідна сторінка дозволяє реєструвати платежі в системі на рисунку 4.20.

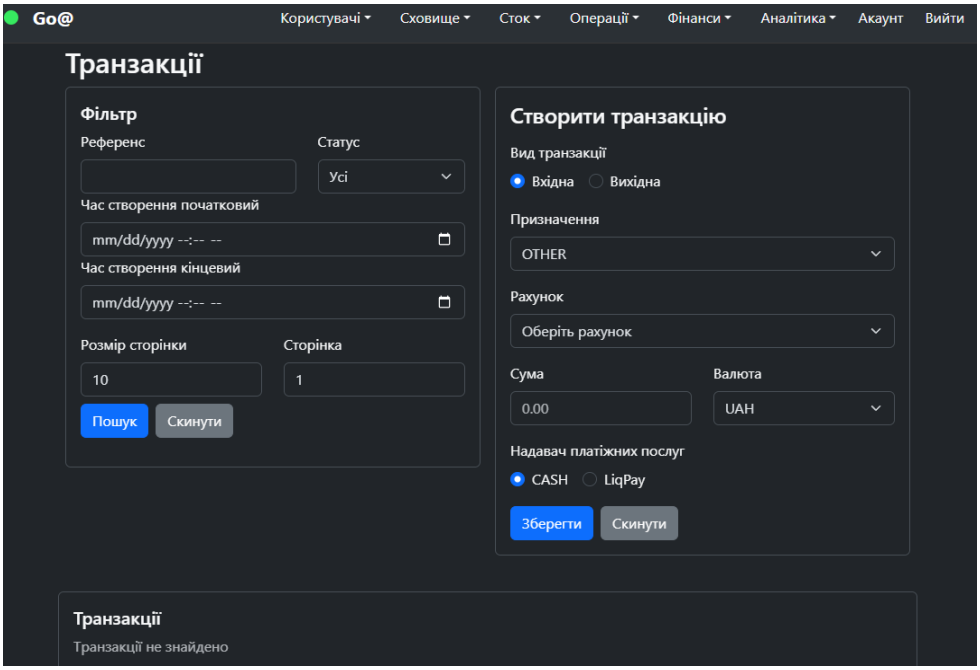


Рисунок 4.20 — Робота з транзакціями

Як і решта сторінок, ця містить фільтр, форму для створення і оновлення

транзакцій. Інтегровано декілька платіжних провайдерів, що дозволяє проводити перекази електронно.

Інтеграція з LiqPay [18] дозволяє отримувати електронні перекази, система генерує відповідне посилання на рисунку 4.21 за яким платник може ініціювати платіж, сума отримувач та інші необхідні деталі вже вбудовані на рисунку 4.22.

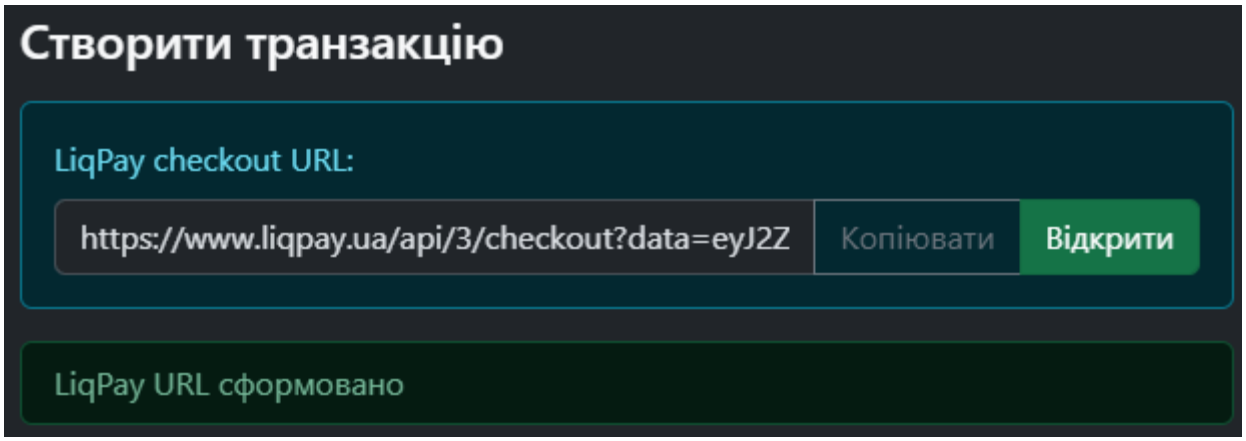


Рисунок 4.21 — Отримання посилання на оплату при ініціації транзакції

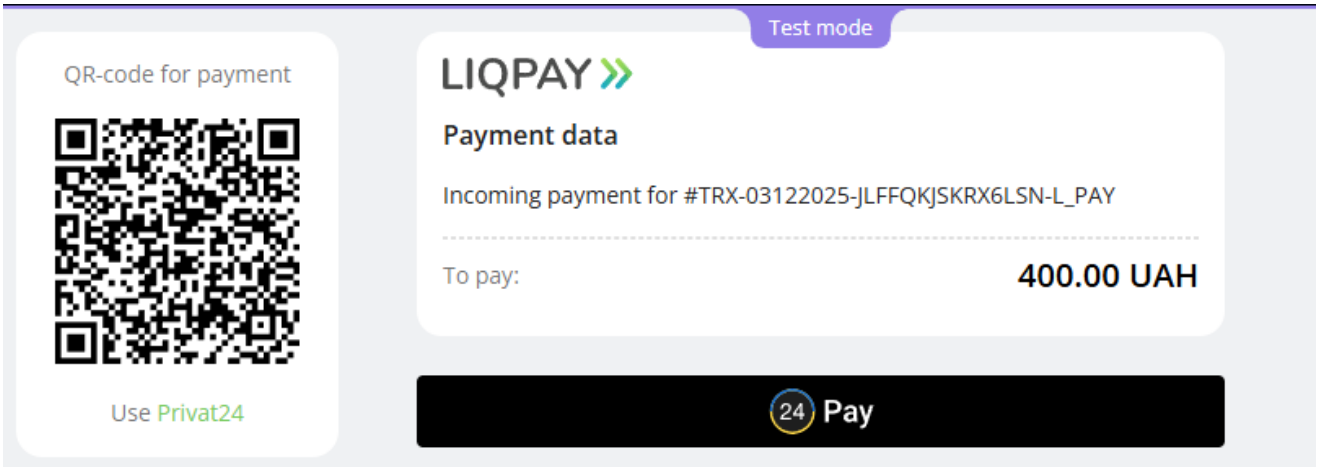
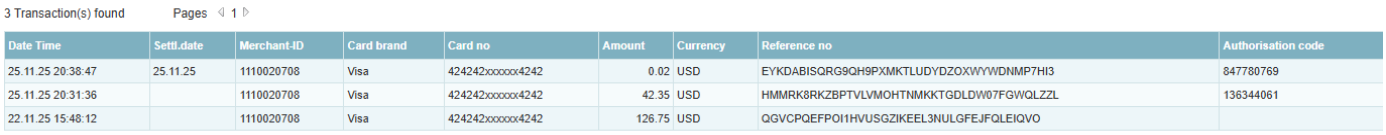


Рисунок 4.22 — Оформлення транзакції через LiqPay форми

Інтеграція з DataTrans [19] дозволяє проводити вихідні електронні платежі, аналогічно до LiqPay ці сервіси надають інформаційні дашборди, у яких можна керувати всією необхідною інформацією.

Розглянемо особистий кабінет користувача DataTrans зі списком існуючих транзакцій на рисунку 4.23. Цей платіжний провайдер надає достатній функціонал

навіть для найбільших підприємств. Логування усіх фінансових транзакцій, синхронізація, відміна або підтвердження, звіти, налаштування облікових записів для різних типів платників, як то онлайн через сайт, форму або через телефонний застосунок.



3 Transaction(s) found Pages 1

Date Time	Settl.date	Merchant-ID	Card brand	Card no	Amount	Currency	Reference no	Authorisation code
25.11.25 20:38:47	25.11.25	1110020708	Visa	424242xxxxxx4242	0.02	USD	EYKDABISQRG9QH9PXMKTLUDYDZOXWYWDNMP7HI3	847780769
25.11.25 20:31:36		1110020708	Visa	424242xxxxxx4242	42.35	USD	HMMRK8RKZBPTVLVMOHTNMKKTGDLDW07FGWQLZZL	136344061
22.11.25 15:48:12		1110020708	Visa	424242xxxxxx4242	126.75	USD	QGVCPQEFPOI1HVUSGZIKEEL3NULGFEJFQLEIQVO	

Рисунок 4.23 — Робота з DataTrans транзакціями

Розроблена система також підтримує мультивалютні операції, є окрема сторінка для роботи з курсами валют, є автоматична конвертація між валютами при ініціалізації транзакції через платіжного провайдера на рисунку 4.24.

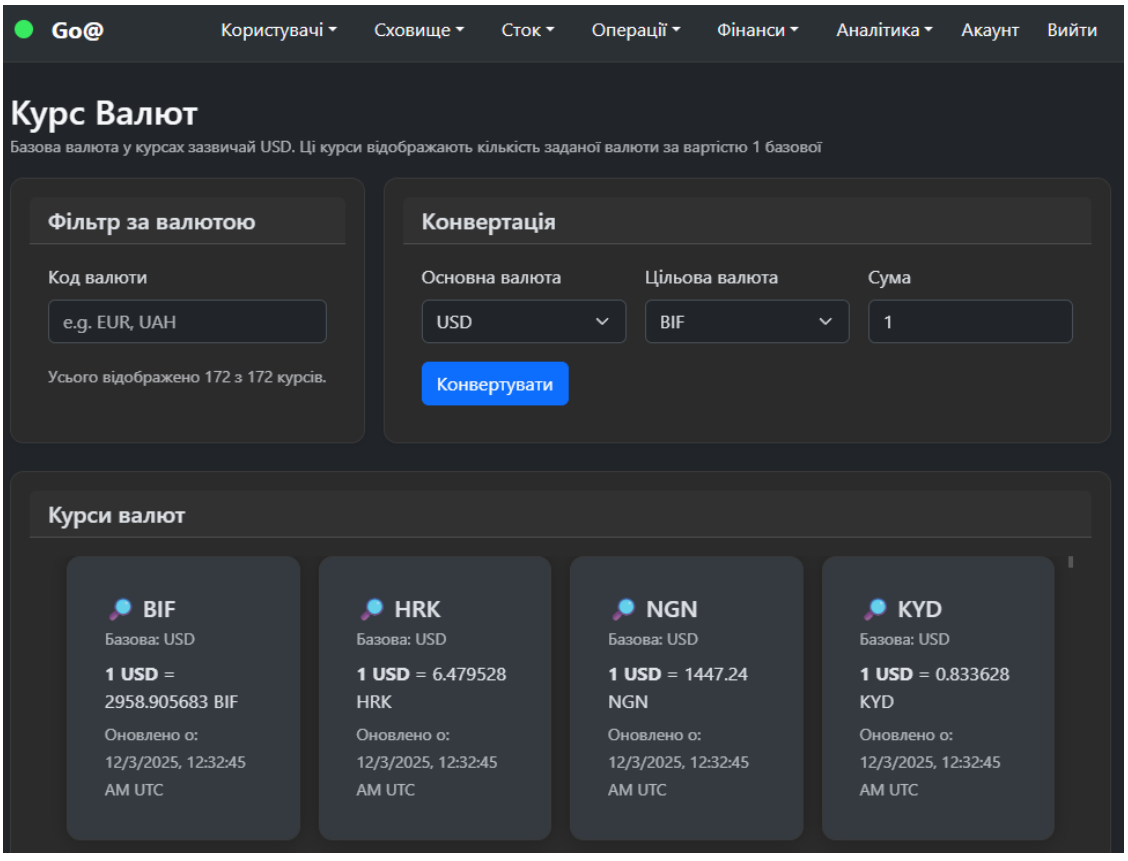


Рисунок 4.24 — Робота з курсами валют

Ці дані отримуються з відкритого API для всіх існуючих світових валют. Ця

інформація оновлюється раз у добу, зберігається у кеші на стороні сервера, оновлюється та зберігається локально на стороні клієнта.

Аналітика є важливою частиною системи, в даному випадку це реалізовано у достатньому простому об’ємі. Є можливість переглянути загальний фінансовий обіг для конкретного рахунку, загальна сума вхідних та вихідних транзакцій для кожної валюти, якщо застосовувалися різні на рисунку 4.25.

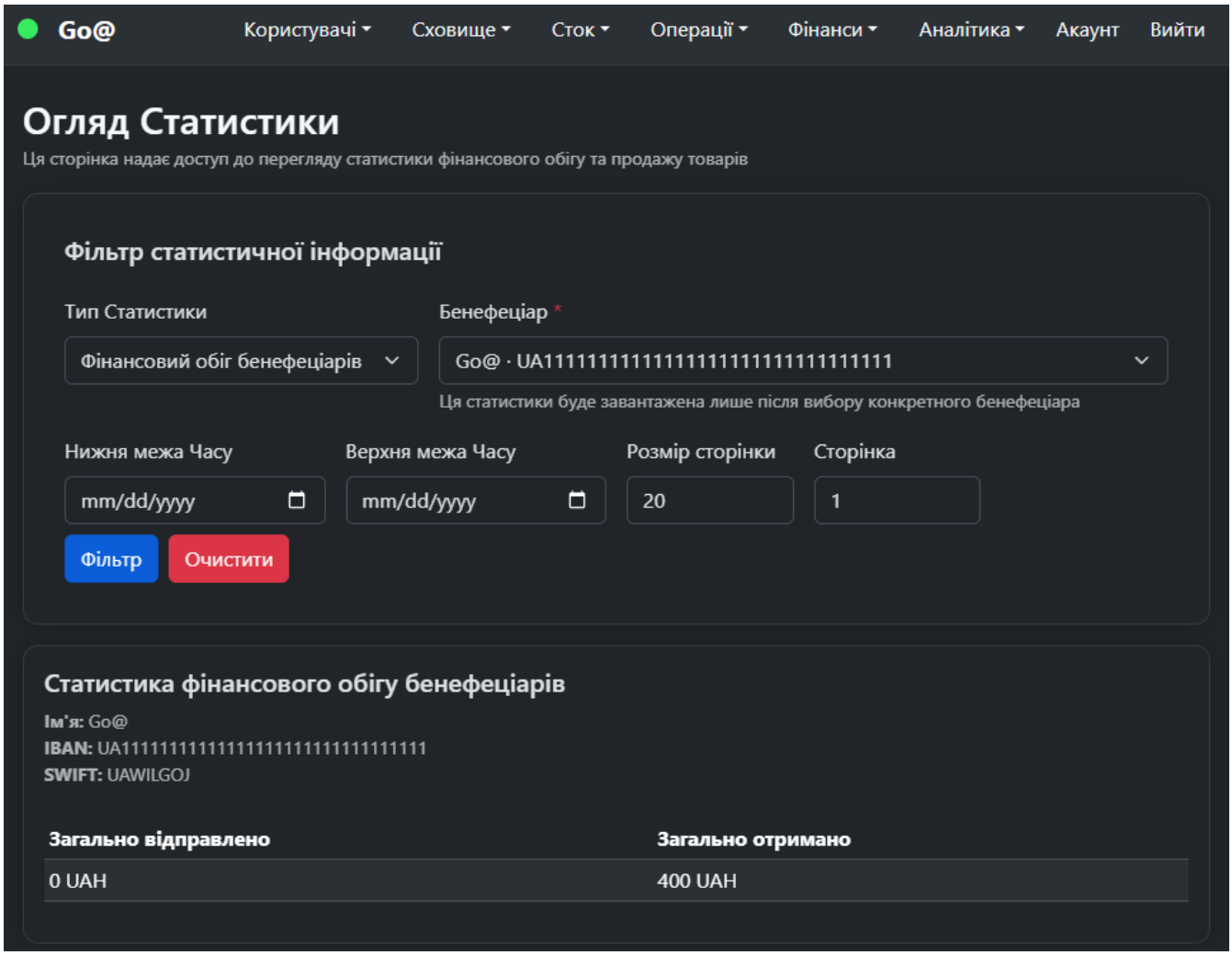


Рисунок 4.25 — Робота з фінансовою статистикою

І також є статистика продажів товарів на рисунку 4.26, загальна сума і кількість та валюта. Користувач може фільтрувати ці дані за часовими проміжками, що дозволяє формувати деталізовані звіти за день, тиждень, місяць або будь-який інший період. Такий функціонал дає змогу відстежувати динаміку продажів, визначати найбільш популярні позиції та аналізувати фінансову

ефективність роботи складу чи всієї логістичної системи загалом.

Завдяки гнучким параметрам фільтрації та інтуїтивному простому відображенню результатів модуль дозволяє швидко отримувати потрібні аналітичні дані та приймати обґрунтовані управлінські рішення на основі фактичних показників.

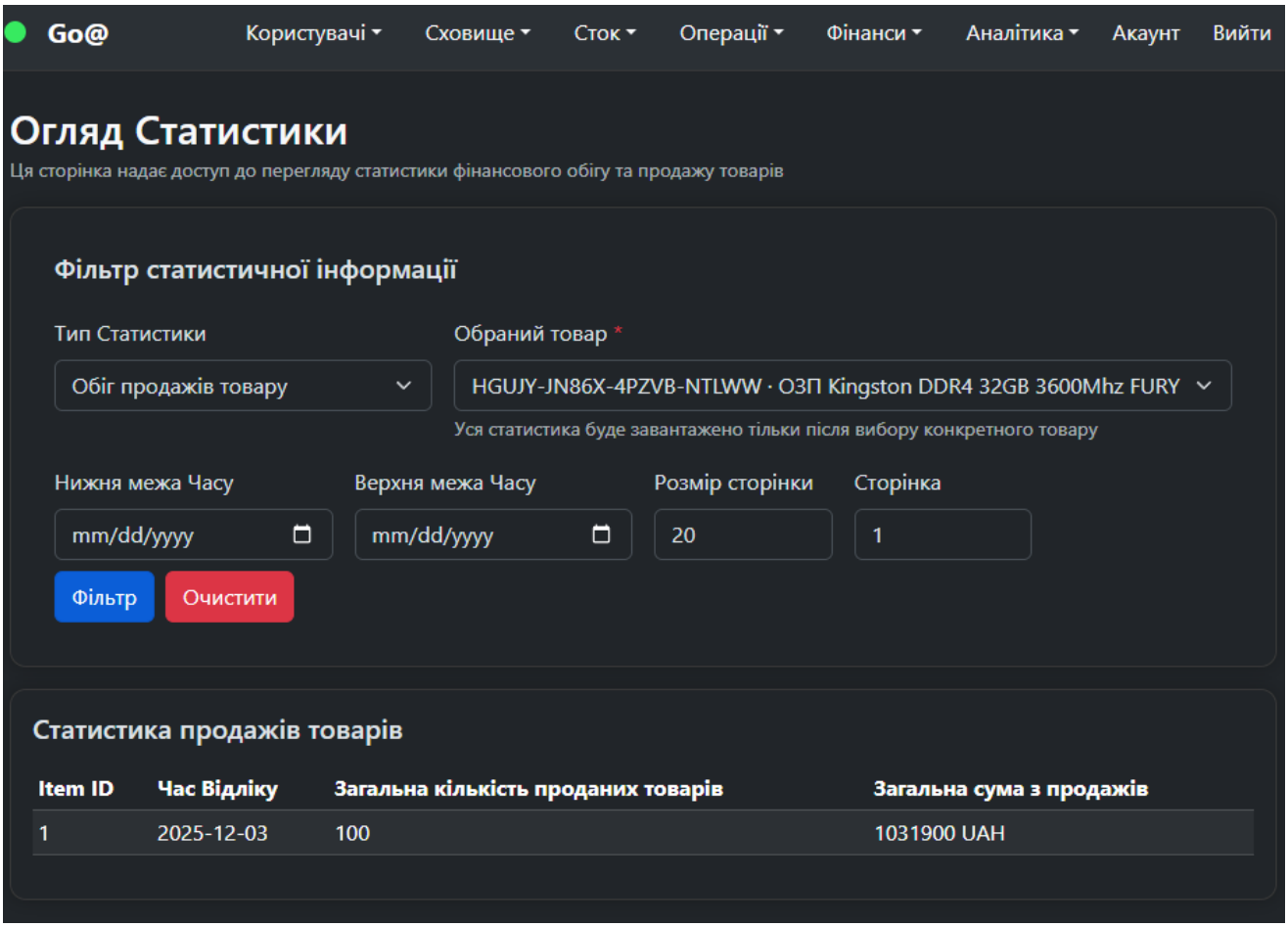


Рисунок 4.26 — Робота зі статистикою продажів товарів

Таким чином було розглянуто всі основні сторінки клієнтського інтерфейсу, це задовольняє потреби користувача для більшості задач по роботі з логістикою та отриманням необхідної інформація для бізнес-аналізу та покращення робочого процесу в компанії.

Крім того, гнучкість інтерфейсу забезпечує можливість подальшого масштабування системи відповідно до зростаючих потреб компанії.

4.5 Тестування системи

Для перевірки працездатності та коректності роботи системи було проведено комплексне тестування, яке охоплює як автоматизовані, так і ручні методи перевірки. Реальні модульні та інтеграційні тести забезпечили перевірку ключових сервісів і бізнес-логіки на рівні серверної частини. Додатково виконано ручне тестування через користувацький інтерфейс та запити в Postman, що дало змогу оцінити роботу всіх API-ендпоінтів, взаємодію модулів та поведінку системи в реальних сценаріях використання.

4.5.1 Модульне тестування

Модульне тестування в системі реалізовано із використанням стеку Spring Test, JUnit та Mockito, що дозволяє ізолювати окремі компоненти та перевіряти їх поведінку без залежності від зовнішніх ресурсів. Для роботи з базою даних застосовано Testcontainers [20], завдяки чому тести виконуються у реальному середовищі з піднятими контейнерами PostgreSQL, що гарантує коректність перевірки репозиторіїв та SQL-логіки. Аналогічно піднімаються контейнери Redis та Kafka [21], які дозволяють перевіряти кешування, черги повідомлень, подієву взаємодію та механізми обміну даними між сервісами. Такий підхід дає змогу впевнено тестувати сервіси, контролери, валідацію та бізнес-правила у максимально наближених до бойових умов, забезпечуючи стабільність і передбачуваність роботи системи під час подальшого розгортання та розвитку.

Основним тестовим класом є AbstractIT на рисунку 4.27, він містить усю необхідну логіку, яка наслідується всіма тестовими класами. Цей клас містить спільну конфігурацію, ініціалізацію Testcontainers, налаштування контейнерів PostgreSQL, Redis та Kafka, а також механізми підготовки контексту Spring перед виконанням тестів. Завдяки цьому кожен дочірній тестовий клас успадковує повністю готове та узгоджене середовище, що значно спрощує розробку, усуває дублювання коду та забезпечує єдині принципи тестування для всієї системи.

Таке рішення підвищує стабільність інтеграційних тестів, робить їх передбачуваними та полегшує масштабування тестової інфраструктури у майбутньому.

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@ActiveProfiles({
    "actuator",
    "database",
    "external",
    "redis",
    "default",
    "test"
})
@Testcontainers
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableRetry
@WithUserDetails(value = AbstractIT.OWNER, userDetailsServiceBeanName = "userDetailsSecurityService", setupBefore = TestExecutionEvent.TEST_EXECUTION)
public abstract class AbstractIT {
    protected static final String OWNER = "warehouse";
    protected static final RandomStringUtils GENERATOR = RandomStringUtils.secure();

    @ServiceConnection
    protected static final PostgreSQLContainer < ? > postgres =
        new PostgreSQLContainer < > ("postgres:16-alpine")
        .withConnectTimeoutSeconds(5)
        .withStartupTimeoutSeconds(30);

    @ServiceConnection
    protected static final GenericContainer < ? > redis =
        new GenericContainer < > ("redis:7-alpine")
        .withExposedPorts(6379);
```

Рисунок 4.27 — Основний тестовий клас системи

Кожен існуючий сервіс та контролер покрито тестами на перевірку успішного відпрацювання та виняткових ситуацій з відповідними очікуваними помилками, тощо. Для більшості випадків такого тестування має бути достатньо, ці тести імітують усі дії, які робив би кінцевий користувач використовуючи інтерфейс застосунку, тестуваннння використовує реальний контекст з сховищами даних та іншими елементами зовнішньої інфраструктури. Тестування побудовано за методологією інтеграційних, це передбачає використання оточення, що імітує реальне, використовує юніт та модуль тести. Java передбачає JUnit [22] та Mockito [23] фреймворки, які дозволяють реалізувати це тестування. Також використано Spring Test для імітації реального робочого циклу застосунку запускаючи Spring Bean Context та всі інші компоненти цього фреймворку для створення застосунків, результати якого на рисунку 4.28. Поєднання інтеграційних, модульних та юніт-тестів дає змогу виявляти потенційні проблеми на ранніх етапах розробки та значно зменшує ризик появи критичних помилок у продакшені. Повноцінне відтворення робочого середовища через Spring Test та Testcontainers забезпечує перевірку взаємодії між сервісами, коректної роботи з даними та узгодженості

всієї бізнес-логіки. Такий підхід створює міцний фундамент для подальшого масштабування системи та спрощує супровід у довгостроковій перспективі.

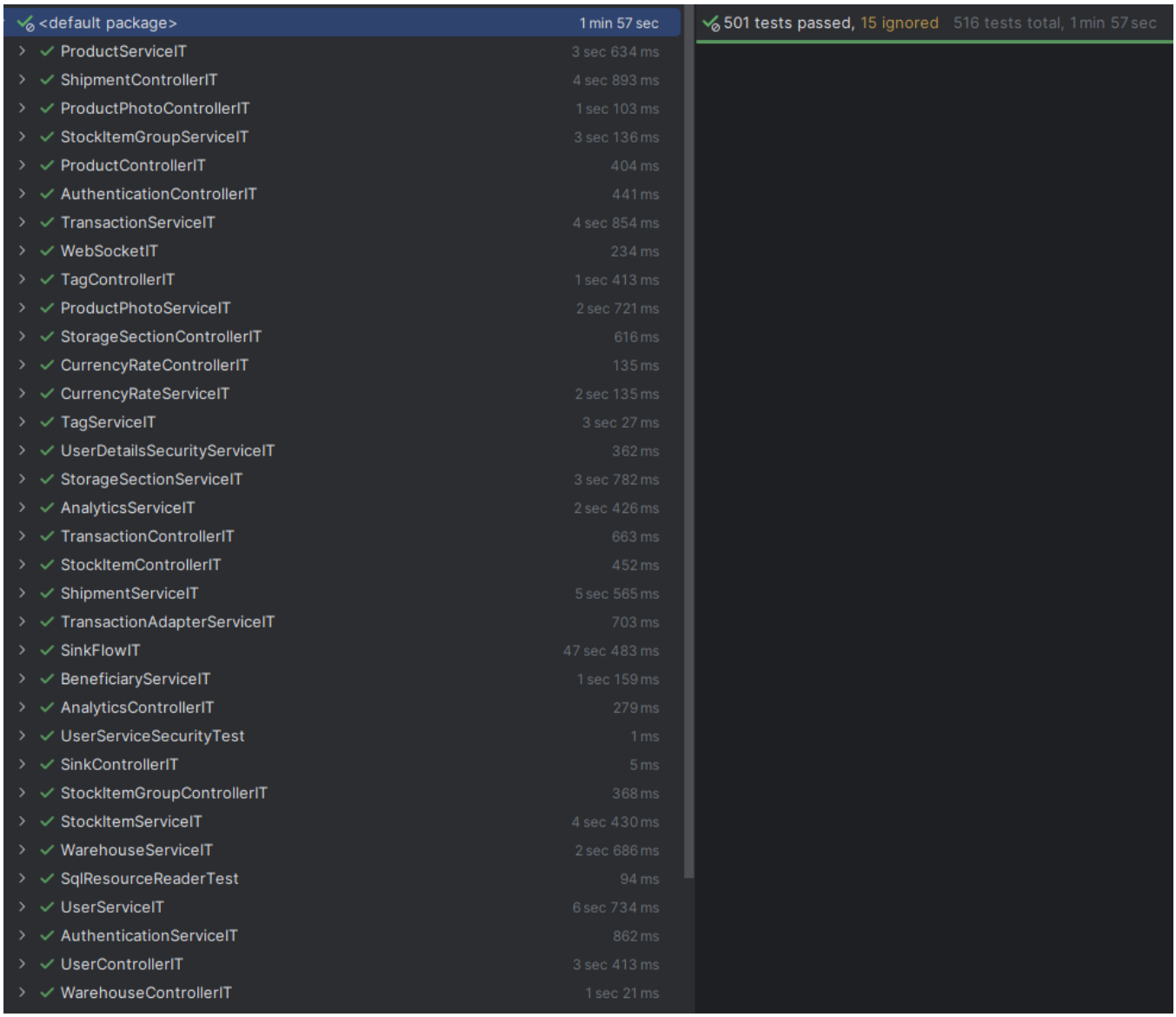


Рисунок 4.28 — Результати модульного тестування

Ці тести покривають роботу усіх бізнес-сервісів та контролерів, усього розроблено 501 тест та ще 15, які дозволяють протестувати інтеграцію з зовнішніми сервісами, але так як вони виконують запити до реальних зовнішніх сервісів, ці тести виключені із загального тестового контексту за винятком цілеспрямованого запуску цих тестових класів вручну, це також вимагатиме налаштуванння оточення.

4.5.2 Мануальне тестування

Тестування застосунку також було проведено за допомогою зовнішніх інструментів та безпосередньо з залученням розробленого інтерфейсу за принципом виконання усіх можливих дій через інтерфейс, також було використано інструмент для виконання запитів Postman. Це дозволяє протестувати логіку роботи з застосунком не маючи інтерфейсу.

Розглянемо тестування засобами Postman [24] детальніше. За своєю специфікою це тестування не має повноцінного контексту і призначене для перевірки конкретного функціоналу через метод контролерів застосунку.

Це тестування передбачає виконання запиту до застосунку ініційованого вручну, необхідно внести відповідні облікові дані та отримати відповідь від сервера, яка має відповідати очікуванням. Отримати успішну відповідь, якщо облікові дані вірні, отримати помилку з 401 статусом, якщо дані невірні, або 404 якщо такого користувача не знайдено в системі.

Таке тестування було проведено для всіх існуючих можливих запитів на сторону сервера. Деякі запити потребують результатів з інших, наприклад для доступу до захищених ресурсів необхідно авторизуватися та передати JWT у запиті.

Однією з переваг Postman є можливість використання оточення та спеціальних скриптів, що автоматично будуть запускатися до або ж після запитів, наприклад можна зберігати токен з відповіді після авторизації, щоб не оновлювати його в усіх запитах кожен раз вручну, коли завершиться термін його дії, також можна визначити адресу для запитів до сервера та інші корисні дані. Також Postman дозволяє групувати запити у колекції, що значно спрощує повторне тестування та систематизацію роботи з API.

Завдяки можливості зберігати змінні, параметри та тіла запитів, розробник може швидко переключатися між різними сценаріями, тестувати обробку помилок, моделювати нестандартні ситуації та перевіряти поведінку системи під різними умовами. Інструмент також підтримує автоматичне виконання

послідовностей запитів, що дає змогу імітувати реальні бізнес-процеси та перевіряти коректність їх проходження від початку до кінця.

4.6 Інструкція користувача

Інструкція користувача описує порядок роботи з системою, вимоги до середовища та особливості взаємодії з інтерфейсом у різних режимах. Оскільки система підтримує як онлайн-, так і офлайн-режим, користувач повинен розуміти принципи збереження даних локально, механізм синхронізації та правила роботи з основними модулями. Даний розділ допоможе швидко ознайомитись з можливостями системи та зрозуміти, як правильно виконувати ключові операції для забезпечення коректного обліку та управління складською логістикою.

4.6.1 Вимоги до робочого середовища

Для коректної роботи системи необхідно попередньо підготувати робоче середовище, яке забезпечить стабільне виконання всіх серверних і клієнтських компонентів. Система розгортається у контейнеризованому вигляді та використовує Docker як основну платформу для запуску, що дозволяє уникнути налаштування інфраструктури вручну та гарантує однакову поведінку у будь-якому середовищі. Для роботи достатньо встановити Docker Engine або Docker Desktop залежно від операційної системи. Після встановлення Docker [25] система здатна автоматично розгорнути всі необхідні складові: сервер застосунку, базу даних PostgreSQL, кеш Redis, брокер повідомлень Kafka разом із ZooKeeper, стек Elasticsearch [26] для журналювання, а також допоміжні інструменти на кшталт Kafka UI та Kibana. Наявність контейнерної інфраструктури повністю усуває потребу встановлювати PostgreSQL, Redis, Java чи додаткові бібліотеки вручну, тому вимоги до налаштування зводяться лише до Docker та мінімального набору системних ресурсів.

Для роботи системи рекомендується мати комп’ютер з операційною

системою Windows, Linux або macOS, який підтримує запуск Docker. У більшості випадків для стабільної роботи всіх контейнерів необхідно не менше 8 ГБ оперативної пам’яті, а для швидшої індексації даних та обробки логів оптимальним є обсяг від 15 ГБ. На диску потрібно передбачити не менше 5-10 ГБ вільного простору для зберігання даних PostgreSQL, журналів Elasticsearch та внутрішніх шарів контейнерів. Необхідність доступу до інтернету залежить від початкового етапу: під час першого запуску Docker завантажить всі образи сервісів із зовнішнього репозиторію, після чого система може працювати повністю локально. Якщо образи були завантажені раніше, робота можлива навіть у середовищі без доступу до мережі.

Для запуску системи потрібно клонувати репозиторій з вихідним кодом або отримати архів з конфігурацією, після чого перейти у директорію проекту та виконати стандартну команду запуску Docker Compose. Зазвичай цього достатньо щоб всі сервіси автоматично було запущено у правильній послідовності, оскільки в конфігурації визначено залежності, перевірки станів та очікування готовності. Після старту контейнерів серверна частина стає доступною через HTTP API за вказаним портом, а користувацький інтерфейс може бути відкритий у браузері без потреби у локальній збірці. Внутрішня мережа Docker ізольовує всі сервіси один від одного, але водночас забезпечує коректну комунікацію між ними, тому від користувача не потрібно додаткових дій щодо налаштування мережевих підключень або портів.

Для роботи користувачеві необхідно мати сучасний браузер з підтримкою JavaScript, локального сховища та IndexedDB, що забезпечує функціональність офлайн-режиму. Завдяки цьому застосунок може працювати навіть у момент відсутності мережі, а синхронізація з сервером відбувається автоматично після її відновлення. Стандартні браузери на Windows, Linux та macOS повністю підтримують необхідні функції, тому не потрібно встановлювати додаткові розширення чи інструменти. Вебзастосунок працює автономно і не вимагає окремого встановлення: достатньо відкрити адресу, на якій запущено клієнтську частину, і здійснити авторизацію.

При роботі з Docker важливо, щоб у системі був увімкнений механізм віртуалізації, оскільки він необхідний для запуску контейнерів. На Windows використовується вбудована технологія WSL, на macOS — власний механізм віртуальних машин, а на Linux Docker працює безпосередньо з ядром. У разі проблем із запуском найчастіше потрібно лише увімкнути віртуалізацію у BIOS або перевстановити Docker Desktop.

Після того як усі сервіси запущено, система готова до повноцінної роботи. Вебінтерфейс дозволяє виконувати управління складами, товарами, продуктами, тегами, відправленнями та фінансами. Серверна частина обробляє всі запити, зберігає дані у базі, відправляє повідомлення через Kafka, кешує запити через Redis та логює події у Elasticsearch. Усі ці процеси працюють автоматично та не потребують від користувача знань з адміністрування чи налаштування інфраструктури. Використання Docker робить систему портативною та незалежною від конкретного середовища, а також дозволяє легко переносити її на інші машини та сервери без повторної конфігурації. Діаграма розгортання наведена у додатку К.

4.6.2 Робота в офлайн-режимі

Окремої уваги вартий найбільш особливий функціонал застосунку, а саме робота в режимі без доступу до сервера. Це має свої обмеження і особливості.
Ідея полягає в тому, щоб у випадку зникнення з’єднання частина функціоналу залишалася активною, хоча б та яка є критично важливою для роботи і може зупинити роботу всього підприємства у разі проблеми з підключеннями.

Щоб реалізувати це необхідно визначитися з функціоналом, який повинен підтримуватися і чи можливо це зробити. Виконання оновлень інформації майже гарантовано призведе до конфліктів або втрачених оновлень, концептуально можливо вести лог змін, зберігати ці зміни у чергу після чого обробляти їх у порядку створення у реальному часі. Таким чином кожна задача повинна мати час створення, але навіть з таким підходом головна проблема, це чи актуально робити

цю зміну тут і зараз. Якщо це вже було оновлено іншим користувачем, тощо. Через це оновлення не є раціональними в першу чергу з логічної точки зору. Щоб забезпечити працездатність ми використаємо лише запити на створення нових сутностей. Деякі з них не мають залежностей від інших, а саме: бенефеціари, групи товарів, продуктові теги, користувачі, склади і продукти. Усі сутності з цього списку окрім користувачів підтримують створення в режимі офлайн, у випадку з користувачами це заборонено, так як для створення користувача потрібен пароль і зберігати ці дані в локальному сховищі є ризиковано.

Окрім цих сутностей є й інші, щоправда вони відрізняються від інших, так як вони залежать від інших. Товар будується на основі продукту, секція зберігання повинна бути призначена до якогось складу, відправлення робиться на базі існуючого товару, для оформлення транзакцій потрібен рахунок бенефеціара. Ця проблема вирішена за рахунок локального сховища на стороні клієнтського застосунку. Кожен запит на отримання сутностей зі сторони сервера, або ж створення та оновлення в режимі офлайн зберігає та оновлює всі існуючі дані в IndexedDB. Це база даних у пам’яті вебbrowser. Також в системі відсутнє видалення сутностей, окрім продуктових тегів, таким чином ми забезпечуємо що з моменту переривання зв’язку всі сутності які знаходяться в локальному сховищі будуть в базі даних незалежно від інших користувачів та їх дій. Це гарантує безпечність у використанні даних з локального сховища для створення інших сутностей, які залежать від інших на рівні бази даних, тощо.

Це також забезпечує функціональність фільтрації та пошуку записів, у режимі офлайн клієнт просто починає отримувати записи з локальної бази даних замість виконання запиту до сервера. Така логіка забезпечує функціонування 70% системи у режимі офлайн, за умови наявності необхідних даних. Однак цей підхід має обмеження, окрім проблем з оновленням сутностей та синхронізацією цих змін також для ініціації системи обов’язково повинен бути зв’язок. Без цього неможливо авторизуватися в системі або отримати актуальні дані з сервера, це відбувається автоматично після успішної авторизації. Це спричинено безпековими ризиками, винесення логіки авторизації в зону відповідальності клієнтської

частини застосунку компрометує надійність цього процесу. Однак, через збільшення часу дії авторизаційного токена можна забезпечити високу тривалість сесії користувача.

На стороні інтерфейсу існує декілька індикаторів зміни у стані підключення до сервера.

У меню застосунку є індикатор стану підключення у вигляді маленького кола біля логотипу, червоний колір означає офлайн режим і недоступність сервера, зелений колір означає активне з’єднання та можливість виконання запитів. Клієнтська частина кожні 5 секунд автоматично перевіряє доступність сервера і у разі зміни стану показує спливаюче повідомлення у випадку зміни статуса..

Натискання на індикатор також відкриває спливаюче сповіщення з поясненням поточного стану. Розглянемо приклад сторінки в офлайн-режимі на рисунку 4.30.

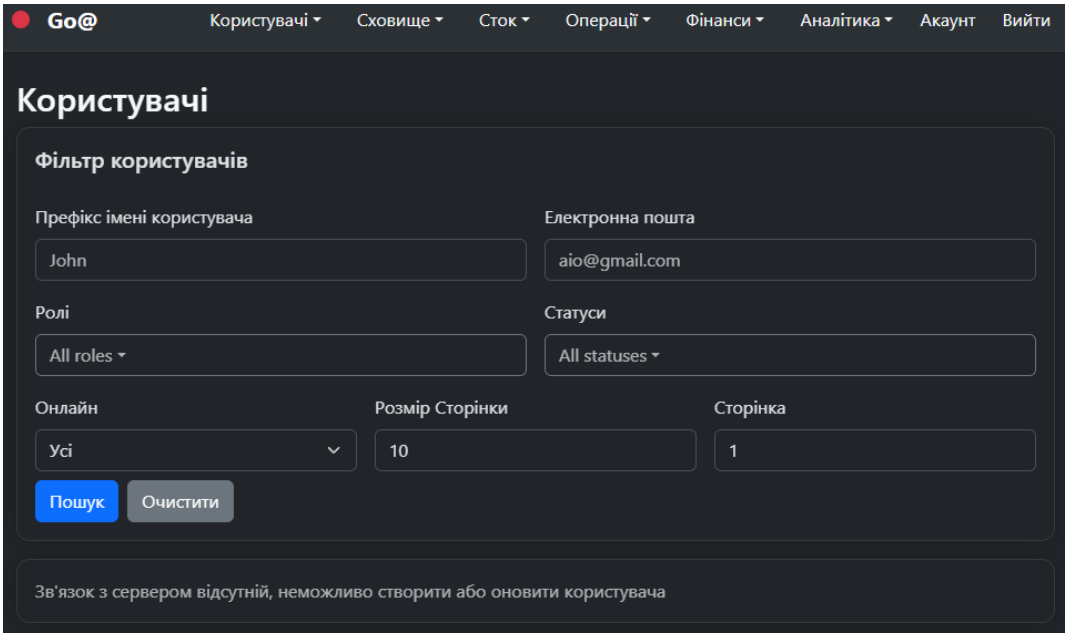


Рисунок 4.30 — Робота з користувачами в офлайн-режимі

У разі відсутнього з’єднання створення чи оновлення користувача стає недоступне. У разі втрати підключення система переходить у офлайн режим, але користувач все одно може створювати окремі сутності. Усі такі дії не

надсилаються одразу на сервер, а зберігаються у локальній базі IndexedDB і автоматично додаються до черги завдань.

Це означає, що створення відбувається повністю локально, а запис очікує моменту, коли з’явиться підключення. Після відновлення доступу до мережі черга обробляється автоматично, і всі заплановані створення надсилаються на сервер без повторного введення даних користувачем.

Такий механізм дозволяє працювати без перерв навіть у ситуаціях, коли інтернет тимчасово недоступний, і забезпечує збереження введеної інформації до моменту синхронізації.

Формат збереження даних передбачає усю необхідну інформацію для відкладеного запиту до сервера, адресу, заголовки контент, тощо. Якщо авторизаційний токен завершив термін дії, в такому разі система автоматично використає новий з авторизаційних даних користувача.

4.6.3 Синхронізація даних після відновлення мережі

Завершальним етапом реалізації системи є логіка синхронізації відкладених завдань у локальній черзі на стороні клієнта з серверними даними.
Було розроблено спеціальну сторінку, де можна переглянути всі заплановані задачі та їх деталі на рисунку 4.31.

Ця сторінка також дозволяє створювати і переглядати історію автоматизованого виконання завдань з черги.

Це дозволяє своєчасно контролювати стан кожного завдання, виявляти можливі помилки та забезпечувати передбачувану узгодженість даних між клієнтом і сервером. Цей підхід до обробки відкладених завдань система гарантує, що жодна операція не буде втрачена, а всі дії користувача коректно відобразяться на сервері після відновлення зв’язку.

Також, журнал виконання дозволяє простежити історію кожного завдання, аналізувати час виконання та реагувати на потенційні збої.

Така архітектура значно підвищує надійність застосунку та створює прозорий механізм контролю синхронізації даних.

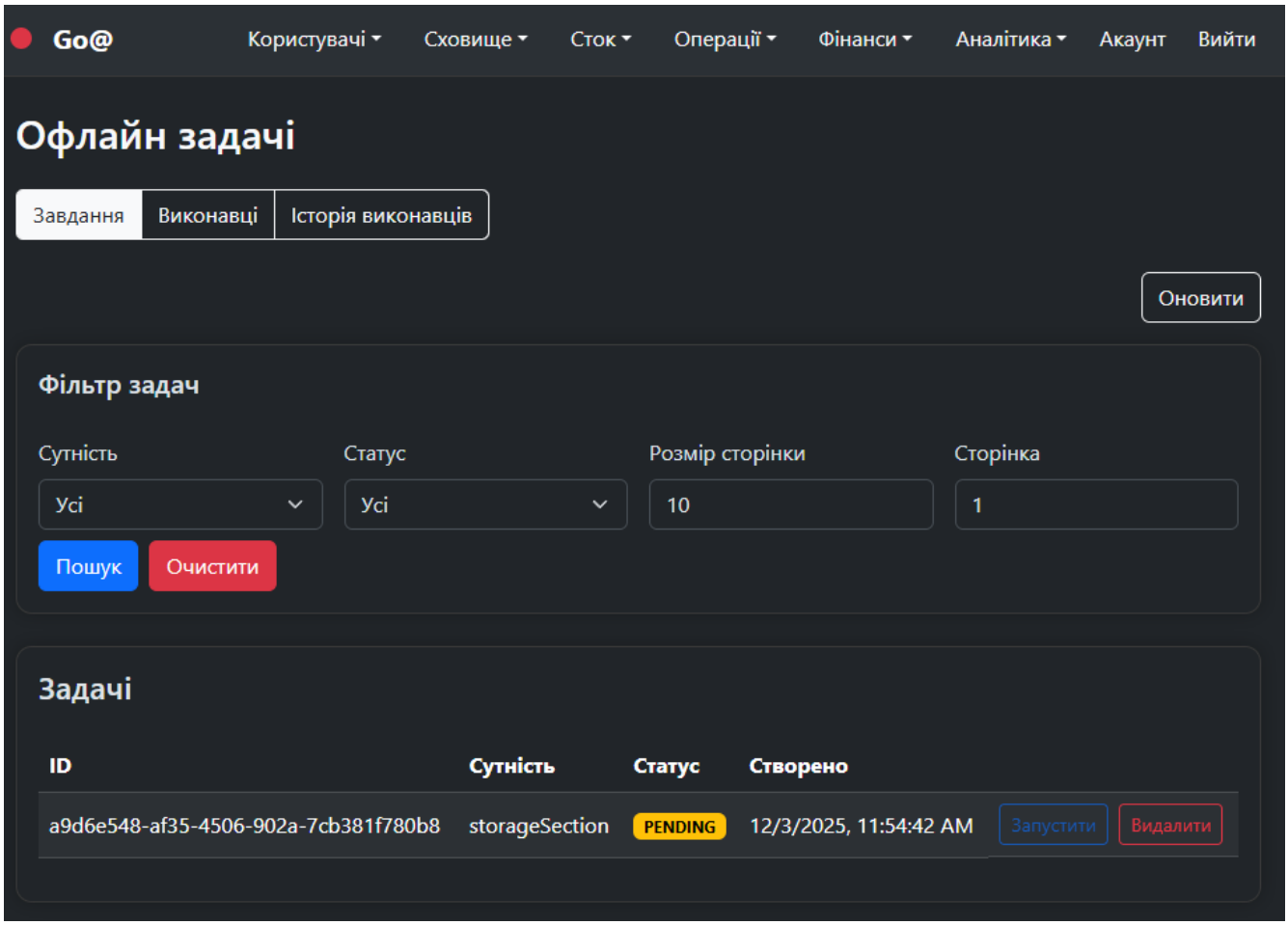


Рисунок 4.31 — Робота з чергою синхронізації

Надається можливість запускати завдання вручну та створити окремі виконавці завдань, що будуть читати чергу та виконувати усі завдання доки вони там є. Таким чином досягнуто гнучкість у підході до синхронізації, користувач може визначити порядок самостійно, або ж якщо після відновлення зв'язку виявилося, що зміни вже непотрібні, задачу можна видалити з черги.

Логіка відтворення виконання задач передбачає надійну систему перевірки та реакції на відповідь сервера, якщо відповідь інформує про клієнтську помилку, наприклад некоректний запит, така задача перейде у фінальний статус завершеної з помилкою, якщо ж існують певні проблеми зі сторони сервера, то для такого випадку було розроблено спеціальний статус, що дозволяє повторне виконання.

Наостанок, на випадок коректного виконання задачі розроблено статус успішності.

Зі сторони сервера реалізовано окремі адреси для збереження та обробки завдань з залученням Apache Kafka. Завдання зберігаються у відповідні топіки, один на кожну сутність, після чого повертається відповідь про прийняття завдання. Обробка ж відбудеться системою самостійно у процесі читання повідомлень consumer-групою. Цей механізм реалізовано для уникнення проблем з великим навантаженням у моменти відновлення зв’язку з сервером, адже мова йдеться про велику кількість завдань надісланих на виконання одночасно з різних клієнтів.

Цей підхід застосовано лише для виконання задач планувальником, у випадку ініціювання обробки задачі самостійно відбувається запит напряму з отриманням відповіді безпосередньо по факту обробки. Це дозволено через контрольований режим навантаження, так як система здатна не лише обробляти задачі з офлайн-сесії, але й продовжувати працювати в звичайному режимі.

Висновки до розділу 4

У цьому розділі було розглянуто реалізацію системи, починаючи від структури бази даних і бізнес-логіки та завершуючи інтеграцією інтерфейсу, механізмами офлайн-роботи та інструментами тестування. Детально описано доменну модель, що включає ключові сутності для обліку логістичних процесів, товарів, фінансових операцій, користувачів та інших об’єктів системи, а також визначено їх взаємозв’язки, способи зберігання та правила взаємодії між ними.

На основі розглянутих прикладів показано, як правильно організована структура сховища дозволяє підтримувати складні сценарії роботи, зберігати історію змін та забезпечувати надійність у роботі з великими обсягами даних.

У межах серверної частини детально описано застосування багатошарової архітектури, де контролери відповідають за маршрути та REST-взаємодію, сервіси містять бізнес-логіку, валідацію та доступ до зовнішніх ресурсів, а репозиторії

реалізують безпосередню роботу з базою даних.

Розглянуто конфігураційні файли, профілі та середовище Docker, у якому працює застосунок разом з базою даних, Redis, Kafka та стеком для логування. Окрему увагу приділено клієнтській частині, побудованій на React, яка включає маршрутизацію, авторизацію, компоненти відображення даних, форми редагування, таблиці, модальні вікна й механізми взаємодії з сервером.Особливе місце у реалізації системи займає офлайн-режим, що дозволяє працювати навіть за відсутності мережі, використовуючи локальну чергу завдань та збереження даних у IndexedDB з подальшою синхронізацією після відновлення доступу до сервера.

Описано індикатори стану мережі, логіку сповіщень та сценарії переходу між режимами роботи. Також розглянуто модульне та мануальне тестування, включаючи використання Testcontainers, Postman та інструментів Spring Test, що забезпечують перевірку коректності всіх частин системи. Сукупність розглянутих рішень демонструє цілісність архітектури, узгодженість внутрішніх компонентів і повноцінну готовність системи до експлуатації в реальних умовах складської логістики.

Окремо варто підкреслити, що система була ретельно та всебічно протестована, що дозволило підтвердити стабільність, коректність і надійність усіх її компонентів. Модульні тести охопили бізнес-логіку, валідацію, доступ до даних та поведінку критичних сервісів, використання Testcontainers забезпечило перевірку в умовах, максимально наближених до реального середовища роботи. Мануальне тестування через інтерфейс та Postman дозволило відтворити реальні сценарії користувача, перевірити точність відображення інформації, роботу форм, обробку помилок, механізми авторизації й офлайн-режим. Результати тестування підтвердили, що система функціонує коректно, витримує високий рівень навантаження, забезпечує цілісність даних, якісну синхронізацію та є повністю готовою до практичного використання.

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проекту

Таблиця 5.1 — Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Автономна система керування логістикою та зберіганням товарів	Торгівельний бізнес	Зручний та простий інтерфейс керування системою
		Автоматизація головних логістичних процесів
		Підтримка роботи критичного функціоналу в режимі офлайн
		Підтримка фінансових транзакцій
		Підтримка логіки оформлення перевезень товарів
		Інтегрована з рядом необхідних сторонніх сервісів
		Автоматизована звітність фінансового та товарного обігу
		Масштабованість для великих підприємств
	Перевезення вантажів	Гнучка організація логістичного функціоналу
		Адаптивна система ведення обліку зберігання
		Легка інтеграція з новими сервісами
		Формування та перегляд статистики
		Легка інтеграція зі сторонніми сервісами
		Функціонал відслідковування статусу відправлень

Таблиця 5.2 — Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Продукти-конкуренти			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Shopify Stocky	Loho Inventory			
1	Інтерфейс та UX	Висока	Середня	Висока		+	
2	Автоматизація логістичних процесів	Повна	Часткова	Повна	+		
3	Швидкість розгортання	Швидка	Швидка	Повільна		+	
4	Кастомізація	Часткова	Обмежена	Повна			+
5	Інтуїтивність інтерфейсу	Висока	Середня	Низька	+		
6	Масштабованість	Висока	Висока	Висока		+	
7	Гнучкість ведення обліку товарів та звітності	Розширена	Розширена	Розширена		+	
8	Доступність API	Повна	Часткова	Повна	+		
9	Підтримка мобільних пристроїв	Веб	Веб	Веб		+	

№	Техніко-економічні характеристики ідеї	Продукти-конкуренти			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Shopify Stocky	Loho Inventory			
10	Офлайн режим	Повний	Відсутній	Відсутній	+		
11	Безпека даних	Висока	Висока	Середня		+	
12	Можливості аналітики	Базова	Базова	Розширена			+
13	Система сповіщень	Базова	Базова	Базова		+	
14	Споживання ресурсів сервера	Низьке	Середнє	Високе	+		
15	Час на розгортання системи	<10 хв	<59 хв	>2 год		+	
16	Можливість розширення функціоналу	Висока	Обмежена	Через плагіни	+		
17	Вимоги до апаратного забезпечення	Мінімальні	Мінімальні	Середні	+		

Цей детальний аналіз показує, що проект має значну кількість сильних сторін, особливо в аспектах, пов'язаних з технічною реалізацією та інтеграцією зі сторонніми сервісами. Виявлені слабкі сторони можуть бути посилені в процесі оновлень інформаційної системи.

5.2 Технологічний аудит ідеї проекту

Таблиця 5.3 — Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Інформаційна система управління складською логістикою з підтримкою офлайн-доступу та аналітики	Серверна частина: Java та Spring Boot для REST API	Наявні	Доступні, безкоштовні
		Асинхронна обробка: CompletableFuture, WebFlux	Наявні	Доступні, безкоштовні
		База даних: PostgreSQL	Наявні	Доступні, безкоштовні
2	Інтеграція зі сторонніми сервісами	LiqPay, DataTrans, OpenCurrency SDKs	Наявні	Доступні, безкоштовні
3	Автоматизоване тестування	Власні алгоритми інтеграційного тестування	Розроблені	Доступні після розробки
4	Система сповіщень	React Toast	Наявні	Доступні, безкоштовні
5	Взаємодія з користувачем	Інтерактивний вебінтерфейс	Наявний	Доступний після розробки
6	Збереження даних	PostgreSQL, Redis	Наявні	Доступні, безкоштовні

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
7	Безпека та авторизація	JWT-stateless авторизація	Наявні	Доступна після розробки
8	Аналітика та звітність	Алгоритми аналізу	Розроблені	Доступні після розробки

Більшість застосованих технологій є відкритими, доступними та не потребують додаткових фінансових вкладень, що робить розробку системи економічно доцільною. Також передбачено використання сторонніх інтеграцій з економічно вигідним плануванням за рахунок низьких тарифів або функціоналу, що покриває більшу кількість потреб за ціною конкурентів на ринку у даній області програмного забезпечення

Частина функціональних можливостей реалізована за допомогою власних програмних компонентів, однак це не створює істотних технологічних бар’єрів і не ускладнює подальший розвиток проекту через адаптивний та легкий до інтеграції з новим функціоналом інтерфейс розробленого програмного забезпечення та основі для інтеграції зі сторонніми сервісами, що необхідні потенційній клієнтській базі

Лімітація розробленого за стосунку на даний момент виражена на рівні інтеграції зовнішніх сервісів або інфраструктури, проте вони легко долаються масштабуванням, використанням альтернативних рішень або переходом на розширені тарифи відповідних платформ та розробкою функціоналу з можливістю обробки виключно існуючими технічними засобами на стороні за стосунку без залучення зовнішніх рішень..

Загалом система є технічно реалізованою, повністю сумісною з обраною технологічною базою та не потребує значних додаткових ресурсів для завершення та подальшої підтримки.

5.3Аналіз ринкових можливостей запуску стартап-проекту

Таблиця 5.4 — Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	5-7 (Warehouse Mgt, Oracle WMS , Odoo Inventory, Loho Inventory, AShopify Warehouse)
2	Загальний обсяг продажів, грн/ум.од	Світовий ринок WMS (Warehouse Management Systems) оцінюється в $3.4 млрд (2024 рік) [24,25]
3	Динаміка ринку (якісна оцінка)	Зростає. За прогнозами, середньорічний темп зростання складе 19.5% до 2030 року
4	Наявність обмежень для входу (вказати характер обмежень)	Висока конкуренція з боку існуючих рішень
		Необхідність відповідності вимогам безпеки даних та GDPR
		Вимоги до надійності та стабільності системи
		Необхідність початкових інвестицій у розробку
№	Показники стану ринку (найменування)	Характеристика

5	Специфічні вимоги до стандартизації та сертифікації	Відповідність стандартам захисту персональних даних (GDPR, CCPA)
		Відповідність галузевим стандартам (GS1, EDI, ISO 9001)
		Сертифікація безпеки для роботи з корпоративними та логістичними системами (ISO 27001, SOC 2)
		Відповідність вимогам доступності (WCAG 2.1)
6	Середня норма рентабельності в галузі (або по ринку), %	19-30%

Варто зазначити, що банківський відсоток на вкладення (19%) є нижчим за середню норму рентабельності в галузі (19-30%), також зайняття ніші офлайн-підтримки повинно привабити інвесторів вкладати кошти у даний проект з розрахунком на відчутне повернення коштів.

Додатково слід підкреслити, що зростаючий попит на автоматизацію складських процесів, особливо з урахуванням потреби безперервної роботи систем у разі нестабільного інтернет-з’єднання, створює сприятливі умови для впровадження рішень з офлайн-підтримкою.

Така функціональність не лише підвищує надійність і конкурентоспроможність програмного продукту, також зменшує операційні ризики для бізнесу, що позитивно впливає на економічну привабливість проєкту та посилює зацікавленість потенційних інвесторів у довгостроковій перспективі.

Таблиця 5.5 — Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Потреба в ефективній та автономній організації логістичних процесів	Торговельні підприємства	Тривалі цикли прийняття рішень	Відповідність акредитаційним вимогам
			Потреба в інтеграції з існуючими системами	Можливість масштабування
			Важливість відповідності логістичним стандартам	Надійне зберігання даних
				Розширена система звітності
				Гнучкі інструменти керування логістикою
			Необхідність безперервної роботи системи	Підтримка офлайн-режиму роботи та синхронізації даних

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
2	Необхідність організації переміщень товарів	Організація перевезень вантажів	Швидкі цикли прийняття рішень	Інтеграція з існуючою IT-інфраструктурою
			Фокус на практичних результатах	Адаптація під корпоративні вимоги
			Важливість інтеграції з корпоративними системами	Аналітика товарного та фінансового обігу підприємства
			Потреба в швидкому масштабуванні	Захист корпоративних даних
			Важливість користувацького досвіду	Простота налаштування
				Гнучкість у налаштуванні
				Інтерактивні інструменти

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
				Автоматизація рутинних процесів
				Доступна цінова політика

Такий аналіз цільової аудиторії демонструє, що попри відмінності у вимогах різних груп користувачів, для всіх них існує спільний набір ключових функцій, які залишаються актуальними незалежно від масштабу чи напрямку діяльності. Це створює можливість розробити універсальну систему, яка забезпечує базову функціональність для всіх сегментів і водночас легко розширюється та адаптується під індивідуальні потреби кожного конкретного користувача.

Таблиця 5.6 — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Наявність сильних гравців на ринку	Впровадження інноваційних функцій
		Поява нових конкурентів	Гнучка цінова політика
		Демпінг конкурентами цін	Розробка програм лояльності
		Наявність на ринку подібних або функціонально дублюючих рішень	Акцент на унікальних перевагах продукту та диференціація функціоналу

№	Фактор	Зміст загрози	Можлива реакція компанії
2	Ринкові умови	Економічна нестабільність	Розробка нових функцій під запити ринку
		Зміна потреб користувачів	Пропозиція гнучких умов оплати, оновлення функціоналу
		Зниження платоспроможності клієнтів	Пропозиція гнучких умов оплати, оновлення функціоналу
		Диверсифікація цінових пропозицій	
3	Законодавчі зміни	Нові вимоги до захисту даних	Моніторинг законодавчих змін
		Зміни в освітньому законодавстві	Швидка адаптація продукту
		Регуляторні обмеження	Отримання необхідних сертифікатів
			Консультації з юристами
4	Кадрові ризики	Складність пошуку кваліфікованих спеціалістів	Створення привабливих умов праці
		Несприятливі фінансові умови роботи	Розширення клієнтської бази, адаптація цін на користування та розробка нового унікального функціоналу для залучення фінансових вкладів
		Висока вартість розробки	Програми підтримки малого бізнесу
		Плинність кадрів	Партнерство з логістичними компаніями

Таблиця 5.7 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростання попиту на логістичні послуги	Збільшення кількості потенційних клієнтів	Розширення функціоналу
		Розширення сфер застосування	Створення спеціалізованих рішень
		Нові ніші на ринку	Активна маркетингова кампанія
2	Технологічний розвиток	Поява нових технологій навчання	Впровадження нових технологій
		Розвиток AI та ML	Розробка інноваційних функцій
			Автоматизація процесів
3	Зміни в сфері логістики	Перехід на автономний режим роботи	Адаптація продукту під нові формати та реалізація рішень для потреб
		Нові формати логістичної організації	Розробка специфічних рішень
		Потреба в сховищі, переміщенні товарів	Партнерство з службами доставки
4	Корпоративний сектор	Зростання потреби в корпоративному навчанні	Розробка корпоративних рішень
		Збільшення бюджетів на навчання	Створення спеціальних тарифів
		Тренд на постійне навчання	Інтеграція з корпоративними системами

№	Фактор	Зміст можливості	Можлива реакція компанії
5	Міжнародний ринок	Можливість виходу на глобальний ринок	Локалізація продукту
		Зростання попиту в різних регіонах	Розвиток партнерської мережі
		Нові партнерства	Адаптація під регіональні особливості

Аналіз зовнішніх можливостей і загроз свідчить, що ринок демонструє високий потенціал зростання, однак потребує безперервного відстеження тенденцій та оперативної адаптації до нових викликів. Більшість ризиків може бути знижена завдяки стратегічному плануванню, своєчасному оновленню функціоналу та гнучкому реагуванню на зміни у технологічному та конкурентному середовищі.

Таблиця 5.8 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (реалізація конкурентоспроможності)
1.Тип конкуренції олігополістична	Ринок поділений між кількома великими гравцями	Фокус на унікальних особливостях та перевагах системи
	Високі бар'єри входу	Створення та розвиток інноваційних рішень
	Значна диференціація продуктів	Гнучка цінова політика
	Високий рівень конкуренції	Постійне вдосконалення функціоналу та підвищення якості сервісу

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (реалізація конкурентоспроможності)
2.За рівнем конкурентної боротьби — глобальний	Конкуренція ведеться на міжнародному рівні	Локалізація продукту
	Відсутність географічних обмежень	Адаптація до місцевих вимог
	Різні вимоги локальних ринків	Створення регіональних представництв
3.За галузевою ознакою — внутрішньогалузева	Конкуренція між компаніями в межах освітньої галузі	Моніторинг галузевих тенденцій
	Схожий базовий функціонал	Впровадження найкращих практик
	Різні підходи до реалізації	Розвиток унікальних особливостей
4.Конкуренція за видами товарів — товарно-видова	Конкуренція між системами управління навчанням	Розширення функціоналу
	Різні підходи до організації навчального процесу	Покращення користувацького досвіду
	Різний набір функціональних можливостей	Оптимізація процесів навчання
	Орієнтація продуктів на різні цільові аудиторії	Адаптація функціоналу під потреби користувачів

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (реалізація конкурентоспроможності)
5.За характером конкурентних переваг — нецінова	Конкуренція за рахунок технічних переваг	Постійне вдосконалення системи
	Важливість функціональності та зручності	Розвиток технічної підтримки
	Значення якості підтримки	Покращення якості сервісу
6. За інтенсивністю — марочна	Важливість репутації розробника	Розвиток бренду
	Значення впізнаваності бренду	Формування позитивного іміджу
	Довіра до постачальника рішення	Програми лояльності

Таблиця 5.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари- замінники
SAP EWM	Інші корпоративні WMS-системи	Хмарні WMS-стартапи	Провайдери інфраструктури (AWS, Azure, GCP)	Великі логістичні оператори, дистриб’ютори	ERP-модулі з базовим складським функціоналом

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари- замінники
Fishbowl Infor WMS	Корпоративні WMS-системи	Розробники автоматизованих складських комплексів	Постачальники роботизованих рішень	Виробничі підприємства, складські комплекси	Спрощені системи обліку, інтегровані в ERP
Висновки	Висока конкуренція між існуючими гравцями	Низькі бар'єри входу для базових рішень	Широкий вибір постачальників	Вимоги до якості та функціоналу	Часткова заміна функцій
	Диференціація за функціоналом та цільовою аудиторією	Високі вимоги до якості та функціоналу	Помірна залежність від постачальників	Висока чутливість до ціни	Необхідність комплексного рішення
	Боротьба за частку ринку	Необхідність значних інвестицій	Можливість зміни постачальників	Потреба в кастомізації	Різні цілі використання

Проведений аналіз демонструє, що попри значну конкуренцію галузь залишається перспективною для виходу нового продукту, особливо якщо він запропонує чітко сформовану цінність та фокусується на особливостях, які відрізняють його від наявних рішень.

Таблиця 5.10 — Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Модульна архітектура	Можливість легкої інтеграції з різними сторонніми сервісами
		Гнучкість у виборі компонентів системи
		Спрощене масштабування та оновлення
		Можливість поступового розширення функціоналу
2	Автоматизація процесів	Зменшення рутинної роботи працівників логістичної компанії
		Автоматичне ведення статистики
		Автоматизація роботи з товарообігом та фінансовою складовою
		Система сповіщень працівників
3	Технічна надійність	Висока доступність системи
		Ефективна обробка навантаження
		Захищеність даних
4	Інтуїтивний інтерфейс	Мінімальний час на освоєння системи
		Зрозуміла навігація
		Адаптивний дизайн
		Доступність з різних пристроїв
5	Гнучкість налаштування	Можливість адаптації під різні логістичні вимоги
		Кастомізація під потреби конкретного підприємства
		Налаштування ролей та прав доступу

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
6	Аналітичні можливості	Детальна статистика успішності
		Відстеження активності користувачів
		Генерація звітів
		Аналіз ефективності навчання
7	Економічна ефективність	Оптимальне співвідношення ціна/якість
		Відсутність прихованих витрат
8	Розширюваність системи	Відкритий API для інтеграцій
		Можливість розробки власних модулів
		Легка інтеграція нових функцій
9	Якість підтримки	Професійна технічна підтримка
		Регулярні оновлення
		Навчальні матеріали
		Швидке реагування на проблеми
10	Безпека даних	Відповідність стандартам захисту даних
		Шифрування важливої інформації
		Контроль доступу

Таблиця 5.11 — Порівняльний аналіз сильних та слабких сторін "Інформаційна система управління складською логістикою з підтримкою офлайн-доступу та аналітики"

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з розроблюваною системою						
			-3	-2	-1	0	+1	+2	+3
1	Модульна архітектура	19			+				

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з розроблюваною системою						
			-3	-2	-1	0	+1	+2	+3
2	Автоматизація процесів	17				+			
3	Технічна надійність	15			+				
4	Інтуїтивний інтерфейс	19	+						
5	Гнучкість налаштування	16			+				
6	Аналітичні можливості	15				+			
7	Економічна ефективність	14					+		
8	Розширюваність системи	18		+					
9	Якість підтримки	16						+	
10	Безпека даних	17				+			

Таблиця 5.12 — SWOT-аналіз стартап-проекту

Сильні сторони:	Слабкі сторони:
Модульна архітектура з можливістю інтеграції різних платформ Високий рівень автоматизації логістичних процесів Інтуїтивно зрозумілий інтерфейс Гнучка система налаштування Розширена аналітика логістики Надійний захист даних Відкритий API для розширень Автономний режим роботи з налаштованою синхронізацією	Необхідність початкових інвестицій Залежність від надійності зовнішніх платформ Відсутність сформованої репутації на ринку Обмежений початковий функціонал Потреба у формуванні технічної підтримки Невелика команда розробників Необхідність підтримки офлайн-логіки для імплементації нового функціоналу

Можливості:	Загрози:
Зростання ринку освітніх технологій Збільшення попиту на дистанційне навчання Тренд на автоматизацію та реалізацію автономності у логістиці Можливість виходу на міжнародний ринок Розширення функціоналу через нові інтеграції Залучення інвестицій для розвитку Партнерство з логістичними компаніями	Висока конкуренція на ринку Економічна нестабільність Зміни в законодавстві щодо логістики Швидка зміна технологій Проблеми з масштабуванням Складність залучення кваліфікованих кадрів Можлива недовіра до нового продукту

Таблиця 5.13 — Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Поступовий вихід на ринок з базовим функціоналом Фокус на одній платформі інтеграції Залучення пілотних навчальних закладів Збір зворотного зв'язку та доопрацювання продукту	Висока	6-8 місяців
2	Швидкий вихід на ринок з повним функціоналом Одночасна підтримка декількох платформ Активна маркетингова кампанія Створення партнерської мережі	Середня	12-15 місяців

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
3	Партнерство з існуючими освітніми платформами Інтеграція як додатковий модуль Використання партнерської бази клієнтів Спільний маркетинг	Низька	10-12 місяців
4	Фокус на корпоративному сегменті Розробка специфічних корпоративних рішень Прямі продажі великим компаніям Створення галузевих рішень	Середня	14-17 місяців

Після аналізу зазначених альтернатив обираємо альтернативу 1 — "Поступовий вихід на ринок з базовим функціоналом", оскільки:

а) має високу ймовірність отримання ресурсів;

б) дозволяє швидко вийти на ринок та отримати зворотний зв'язок;

в) мінімізує початкові ризики та інвестиції;

г) дозволяє поступово розвивати продукт на основі реальних потреб користувачів;

д) має найбільш оптимальне співвідношення термінів реалізації та ризиків;

е) забезпечує гнучкість у коригуванні стратегії розвитку проєкту з урахуванням змін ринкових умов та результатів первинної експлуатації системи.

ж) сприяє раціональному використанню наявних людських і технічних ресурсів без перевантаження команди на початковому етапі.

и) Підвищує привабливість проєкту для потенційних інвесторів завдяки наявності працюючого прототипу та підтвердженого попиту з боку перших користувачів.

5.4 Розроблення ринкової стратегії проекту

Таблиця 5.14 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Логістичні компанії	Середня	Високий	Висока	Середня
2	Корпоративний сектор	Висока	Високий	Середня	Низька
3	Торговельні компанії	Висока	Середній	Низька	Висока
4	Індивідуальні підприємства у сфері торгівлі або транспортації	Середня	Низький	Низька	Висока
5	Оператори складів та 3PL-провайдери	Середня	Середня	Середній	Висока
6	Малі та середні підприємства у сфері логістики та дистрибуції	Висока	Середня	Середній	Середня
Які цільові групи обрано: корпоративний сектор — як сегмент з високою готовністю до впровадження, значним попитом та платоспроможністю; торговельні компанії— як сегмент з високою гнучкістю та готовністю до інновацій.Стратегія охоплення ринку: стратегія диференційованого маркетингу.					

Таблиця 5.15 — Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту цільової групи потенційних клієнтів	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Поступовий вихід на ринок з базовим функціоналом	Стратегія диференційованого маркетингу	Фокус на автоматизації та роботі в офлайн-режимі Інтуїтивний інтерфейс Якісна технічна підтримка Конкурентна цінова політика Автоматизація ключових логістичних процесів із підтримкою роботи в офлайн-режимі, що забезпечує безперервність функціонування системи. Інтуїтивно зрозумілий інтерфейс та гнучке налаштування функціоналу відповідно до потреб різних категорій користувачів. Конкурентна цінова політика з урахуванням існуючої конкуренції	Стратегія диференціації

Таблиця 5.16 — Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Не повністю, на ринку вже присутні аналогічні рішення, проте розроблена система має офлайн підтримку роботи, що є інноваційним у цій сфері	Компанія буде використовувати комбіновану стратегію: Пошук нових споживачів серед організацій, які тільки починають впроваджувати системи логістики Залучення існуючих користувачів конкурентних продуктів через пропозицію кращого рішення	Базовий функціонал системи управління логістикою буде подібним до конкурентів: Управління користувачами Фінансовий та товарний обіг зі звітністю Модульна архітектура Гнучка інтеграція з іншими платформами Розширена автоматизація процесів Наявність ряду інтеграцій	Стратегія заняття конкурентної ніші

Таблиця 5.17 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Надійність роботи системи Захист даних користувачів Висока доступність сервісу	Стратегія диференціації	Модульна архітектура системи Сучасні технології розробки Надійна інфраструктура	Надійність Безпека Стабільність
2	Простота використання Зрозумілий інтерфейс Мінімальний час на навчання	Стратегія диференціації	Інтуїтивний користувацький інтерфейс Адаптивний дизайн Детальна технічна документація	Простота Зручність Доступність
3	Гнучкість налаштування Можливість інтеграції Масштабованість	Стратегія диференціації	Відкритий API Підтримка різних інтеграцій Модульна структура	Гнучкість Інтеграція Масштабован ість

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
4	Автоматизація процесів Аналітика та звітність Контроль навчального процесу	Стратегія диференціації	Розвинуті інструменти автоматизації Розширена аналітика Система моніторингу	Ефективність Контроль Аналітика

5.5 Розроблення маркетингової програми стартап-проекту

Таблиця 5.18 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Ефективна організація логістичного процесу	Автоматизація рутинних завдань Єдина система управління Контроль обігу товарів та фінансів Автономний режим роботи та синхронізації даних	Модульна архітектура Можливість легкої інтеграції з різними фінансовими, логістичними сервісами та системами обліку Розширені інструменти автоматизації та інтеграції нового функціоналу

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
2	Зручна взаємодія користувача з системою	Інтеграція з популярними сторонніми сервісами Швидкий обмін інформацією Єдиний простір для комунікації	Гнучкість у виборі каналів комунікації Зрозумілий інтерфейс взаємодії Підтримка різних форматів спілкування
3	Відстеження прогресу навчання	Детальна аналітика успішності Автоматична оцінка результатів Формування звітності	Розширені можливості аналітики Автоматизоване оцінювання тестів Гнучкі інструменти формування звітів
4	Безпека та надійність	Захист персональних даних Стабільна робота системи Резервне копіювання	Сучасні методи захисту даних Надійна інфраструктура Регулярне оновлення системи безпеки
5	Масштабованість рішення	Підтримка великої кількості користувачів Можливість розширення функціоналу Адаптація під зростаючі потреби	Оптимізована архітектура для високих навантажень Можливість додавання нових модулів Гнучке масштабування ресурсів

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
6	Інтеграція з існуючими платформами	API для інтеграції Підтримка стандартних форматів даних Можливість розширення	Відкритий API з детальною документацією Підтримка популярних стандартів інтеграції Можливість створення власних розширень
7	Підтримка та супровід	Технічна підтримка Навчальні матеріали Регулярні оновлення	Професійна служба підтримки Детальна документація Постійне вдосконалення системи

Таблиця 5.19 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Інформаційна система управління складською логістикою з підтримкою офлайн-доступу та аналітики		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	а) модульна архітектура системи; б) підтримка різних інтеграцій; в) автоматизоване тестування; г) система аналітики та звітності; д) управління користувачами та ролями;	Нм М М М М	Тл Тх Тх Тл Тх

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td rowspan="4"></td><td>е. API для інтеграції;
ж. захист даних;
и. масштабованість;
к. інтуїтивний інтерфейс.</td><td>Нм М М
М</td><td>Тх Тх Тх
Е</td></tr>
<tr><td colspan="3">Якість: відповідність стандартам безпеки даних</td></tr>
<tr><td colspan="3">Пакування: розгортання через хмарний сервіс</td></tr>
<tr><td colspan="3">Марка: Go@</td></tr>
<tr><td>ІІІ. Товар із підкріпленням</td><td colspan="3">До продажу:
презентація можливостей системи;
тестовий період використання;
допомога у розгортанні;
базове навчання користувачів.</td></tr>
<tr><td></td><td colspan="3">Після продажу:
технічна підтримка 24/7;
оновлення системи;
розширена аналітика;
консультації щодо оптимізації процесів.</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання:
– комплексний захист інтелектуальної власності через патентування ключових технологічних рішень;
ліцензування програмного забезпечення;
унікальні алгоритми обробки даних;
постійне вдосконалення та оновлення системи;
захист серверної частини через закритий код;
складність відтворення модульної архітектури;
унікальний користувацький досвід.</td></tr>
</table>

Таблиця 5.20 — Визначення меж встановлення ціни

№	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	Loho Inventory — $79-$249/місяць Fishbowl — від $4 395 одноразово QuickBooks Commerce — $39-$249/місяць AShopify Warehouse — $150-$500/місяць Hulu Warehouse Management — $50-$150/місяць	Oracle WMS Cloud — від $7 000/користувач/рік SAP Extended Warehouse Management — $10 000-$50 000/рік Manhattan WMS — $20 000-$250 000/рік Infor WMS — $15 000-$100 000/рік LogiChech — $15 000 - $45 000/рік AWS Wareouse Enterprise — $400 000 - $1 500 000/рік	$400k-1 млн/рік обороту Середній бізнес: $1-30 млн/рік Логістичні компанії: $5-300 млн/рік Бюджет на WMS: $4 000-150 000/рік	$10-50/користувач/місяць Професійний: $99-199/користувач/місяць Enterprise: від $300/користувач/місяць Вартість впровадження: $1 000-50 000

Таблиця 5.21 — Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Корпоративний сегмент: Тривалий цикл прийняття рішень Потреба в демонстрації продукту Важливість технічної експертизи Необхідність індивідуального підходу	Презентація продукту Проведення демонстрацій Технічні консультації Допомога у впровадженні Навчання користувачів	Нульового рівня (прямі продажі)	Власний відділ продажів Прямі контакти з клієнтами Індивідуальний підхід
2	Освітні заклади: Формальні процедури закупівлі Обмежений бюджет Потреба в офіційній документації Важливість відповідності стандартам	Участь у тендерах Підготовка документації Відповідність вимогам закупівель Інтеграція з існуючими системами	Однорівневий (через партнерів)	Партнерська мережа Сертифіковані реселери Системні інтегратори

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
3	Малий бізнес та стартапи: Швидке прийняття рішень Фокус на ціні Потреба в швидкому старті Самостійне впровадження	Онлайн-продажі Автоматизована реєстрація Базова технічна підтримка Документація та навчальні матеріали	Нульового рівня (онлайн-продажі)	Вебсайт з можливістю замовлення система активації Онлайн-підтримка

Обрана модель збуту ґрунтується на поєднанні трьох ключових каналів, що дозволяє охопити різні сегменти ринку та забезпечити гнучкість у взаємодії з клієнтами. Перший канал — прямі продажі, які виступають основним способом залучення великих замовників. У цьому напрямі працює внутрішній відділ продажів, що супроводжує ключових клієнтів, а також персональні менеджери, які забезпечують індивідуальні консультації, адаптацію рішень і побудову довгострокових партнерських відносин.

Другим елементом системи є партнерська мережа, що включає співпрацю з інтеграторами, впроваджувальними компаніями та сертифікованими партнерами. Завдяки такому підходу можливе розширення ринку, участь у спільних проєктах і підвищення довіри за рахунок зовнішніх експертів, які мають власну клієнтську базу й досвід роботи у сфері логістичних рішень.

Третій канал — онлайн-продажі, орієнтовані насамперед на малий та середній бізнес. Цей сегмент забезпечується за рахунок автоматизованої системи самообслуговування на вебсайті, що дозволяє швидко оформити підписку, отримати консультацію та доступ до документації. Онлайн-підтримка допомагає клієнтам розпочати роботу без залучення менеджера, що значно пришвидшує

процес продажу.

Поєднання цих трьох каналів забезпечує комплексний підхід до збуту: ефективну роботу з різними групами клієнтів, можливість персоналізації для великих компаній, розширення охоплення ринку через партнерів та масштабування продажів за рахунок онлайн-каналу.

Таблиця 5.22 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Корпоративний сегмент: Пошук рішень для оптимізації логістики Увага до ROI Важливість безпеки та надійності Потреба в масштабованості	Професійні B2B майданчики LinkedIn Галузеві конференції Бізнес-форуми Спеціалізовані видання	Автоматизація логістичних процесів Підвищення ефективності логістичних підприємств Надійність та безпека Інтеграція з існуючими системами	Демонстрація економічної ефективності Підкреслення надійності рішення Акцент на масштабованості	"Ефективний логістичний бізнес: автоматизація, контроль, результат, автономність"

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
2	Торговельні та логістичні компанії: Консервативність у виборі рішень Обмежений бюджет Відповідність стандартам Потреба в простоті використання Орієнтація на надійність та безперебійну роботу системи	Логістичні конференції Профільні видання Логістичні портали Професійні спільноти Email-розсилки Різні соціальні мережі з акцентом на професійну комунікацію та розширення інформаційного поля потенційної клієнтської аудиторії	Відповідність логістичним стандартам Простота впровадження Доступність рішення Підтримка офлайн-режиму роботи	Акцент на простоті використання Демонстрація відповідності стандартам Підкреслення доступності	“Надійний та автономний інструмент для логістичного підприємства "

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
3	Малий бізнес та стартапи: Швидке прийняття рішень Орієнтація на ціну Потреба в швидкому старті Самостійне впровадження	Соціальнімережі Онлайн-медіа Тематичні блоги YouTube Email-маркетинг	Швидкий старт Гнучкі тарифи Простота використання Масштабованість рішення	Підкреслення простоти початку роботи Акцент на доступності Демонстрація швидких результатів	"Логістика не блокується навіть без доступу до мережі"

Висновки до розділу 5

Аналіз ринку WMS-рішень показав, що розроблена система управління складською логістикою має високі перспективи впровадження та здатна конкурувати з наявними продуктами у своєму сегменті. Сфера автоматизації логістики продовжує активно розвиватися, а підприємства різних масштабів мають стабільну потребу в інструментах, які забезпечують точний облік, оптимізацію операцій та можливість швидкого масштабування. Розроблена система повністю відповідає цим тенденціям і пропонує функціональні можливості, які є актуальними для сучасних складських процесів.

Ринок складських інформаційних систем демонструє привабливу динаміку,

характеризується стійким попитом та високим потенціалом для розвитку нових гравців. Значна частина компаній переходить від ручного обліку або застарілих систем до сучасних вебрішень із гнучкою архітектурою, що відкриває можливості для впровадження продуктів, які поєднують доступність, інтуїтивність та масштабованість. Рентабельність галузі залишається високою, а різні групи користувачів — від малих складів до великих логістичних операторів — демонструють готовність до впровадження таких систем.

Важливою конкурентною перевагою розробленої системи є модульна архітектура та можливість роботи в офлайн-режимі з подальшою автоматичною синхронізацією даних, що значно підвищує надійність у реальних умовах із нестабільним інтернет-з'єднанням. Інтерфейс побудований так, щоб користувачі швидко адаптувалися без потреби у тривалому навчанні, а внутрішня логіка дозволяє легко інтегрувати систему з іншими сервісами та розширювати можливості за потреби. Такий підхід створює основу для подальшої диференціації на ринку, а також дозволяє ефективно зайняти нішу середніх та малих підприємств, де особливо важлива доступність і простота впровадження.

Обрана стратегія розвитку передбачає акцент на унікальних властивостях системи та поступове розширення функціональних можливостей відповідно до потреб ринку. Комбінована модель збуту, що включає прямі продажі, онлайн-канали та партнерську мережу, дає змогу охоплювати різні сегменти та забезпечує гнучкість у масштабуванні. У межах аналізу було визначено ключові групи користувачів: логістичні компанії, виробничі підприємства, ритейл, e-commerce і склади малого бізнесу — кожна з яких має специфічні вимоги, але водночас користується спільним базовим функціоналом.

Фінансово-економічні характеристики проекту свідчать про конкурентоспроможну модель ціноутворення, що дозволяє запропонувати доступне рішення для невеликих компаній і при цьому забезпечити розширений функціонал для середніх та великих складів. Гнучка система тарифів та передбачувана модель монетизації створюють умови для стабільного розвитку продукту та подальших інвестицій у його вдосконалення.

Узагальнюючи результати дослідження, можна зробити висновок, що система має реальні перспективи комерціалізації. Це підтверджується стабільним попитом на WMS-рішення, динамічним розвитком логістичного ринку та чіткими конкурентними перевагами продукту. Подальший розвиток системи є доцільним за умови дотримання визначеної стратегії, постійного поліпшення функціоналу та забезпечення високої технічної надійності. За правильного підходу продукт може зайняти помітну частку ринку та стати ефективним інструментом для цифровізації складської логістики. У перспективі система може стати основою для створення повноцінної екосистеми логістичних інструментів, що охоплюватимуть не лише складські операції, але й суміжні бізнес-процеси, такі як транспортна логістика, управління замовленнями чи інтеграція з e-commerce-платформами. Це відкриває додаткові можливості для розширення функціоналу, формування партнерських рішень та виходу на нові ринкові сегменти, що підсилює довгострокову конкурентоспроможність продукту.

Таким чином, розроблена система не лише відповідає поточним вимогам ринку, але й має достатній технологічний та архітектурний резерв для подальшого розвитку. Завдяки гнучкості, модульності та орієнтації на реальні потреби користувачів вона може адаптуватися до змін бізнес-середовища та підтримувати розширення функціональності без суттєвих витрат. Це робить її перспективним рішенням, здатним забезпечити довгострокову цінність для підприємств різного масштабу.

ВИСНОВКИ

У результаті виконання магістерської роботи було розроблено повноцінну інформаційну систему для управління складською логістикою з підтримкою офлайн-режиму, автоматичної синхронізації даних та розширеною аналітикою. В межах проекту було проведено аналіз предметної області та вивчення існуючих WMS-рішень, розроблено архітектуру, програмної реалізації, тестування та розроблення ринкової стратегії проекту як стартапу.

Під час дослідження було виконано ґрунтовний аналіз логістичних процесів, особливостей роботи складів, вимог до автономності та відмовостійкості інформаційних систем, а також проведено детальний огляд корпоративних та хмарних WMS-платформ. Це дозволило визначити ключові функціональні та нефункціональні вимоги до системи, сформувати перелік основних сутностей і визначити проблеми, на які має відповідати розроблене рішення, зокрема необхідність роботи при втраті мережі та потребу в прозорому обліку операцій.

У ході здійснення архітектурного проєктування розроблено клієнт-серверну модель із застосуванням сучасних технологій: Spring Boot на серверній частині та React на клієнтській. Обґрунтовано вибір бази даних PostgreSQL, засобів кешування Redis, брокера повідомлень Kafka та засобів централізованого логування Elasticsearch. Було сформовано модульну архітектуру, що забезпечує масштабованість, розширюваність і високу відмовостійкість. Запроваджено механізм офлайн-роботи, який включає локальне кешування даних, черги синхронізації та алгоритми відкладеного виконання операцій.

У рамках реалізації створено серверну частину, яка відповідає за бізнес-логіку, перевірку даних, авторизацію, взаємодію з БД та публікацію подій у Kafka. реалізовано повний набір сутностей, необхідних для управління складом: товари, продукти, групи товарів, склади, секції, транзакції, відправлення, тегування, історію змін та інші. Значна увага приділена захисту даних через використання JWT-авторизації, шифрування конфіденційної інформації та сегрегації доступів.

Клієнтська частина реалізована з використанням React та забезпечує повноцінний інтерфейс для роботи користувача: фільтри, пагінацію, модальні вікна, таблиці, форми створення та редагування сутностей, індикатор офлайн-режиму та систему сповіщень. Було реалізовано механізм локальної черги операцій для офлайн-режиму, що дозволяє продовжувати роботу при втраті підключення до сервера. Система автоматично синхронізує дані після відновлення мережі, забезпечуючи цілісність і узгодженість інформації.

Реалізовано масштабне тестування проекту, яке включало модульні тести на основі JUnit, Mockito та Testcontainers із реальними інстансами PostgreSQL, Redis та Kafka, а також детальне мануальне тестування через Postman та інтерфейс системи. Результати тестування підтвердили коректність функціонування бізнес-логіки, стабільність роботи у складних сценаріях та працездатність офлайн-режиму.

У межах розроблення стартап-проекту виконано аналіз технологічних можливостей, ринку WMS-систем та конкурентного середовища. Проведено оцінку ринкової динаміки, визначено основні групи споживачів, бар'єри входу та переваги розробленого продукту. Проєкт демонструє високу ринкову перспективність, оскільки поєднує функціонал корпоративних WMS із доступністю та простотою впровадження, а підтримка офлайн-режиму є важливою конкурентною перевагою у сегменті малого та середнього бізнесу. Розроблено ринкову стратегію, модель збуту, пропозиції щодо ціноутворення та канали комунікації з клієнтами.

Усі поставлені цілі магістерської дисертації були успішно досягнуті. Створена система відповідає функціональним та нефункціональним вимогам, є технологічно сучасною, масштабованою, безпечною та готовою до подальшого розвитку. Проект має чітку архітектурну базу, практичну цінність і реальні перспективи комерціалізації.

Система має значний потенціал для впровадження у промислових умовах та подальшого масштабування як комерційний продукт. Отримані результати мають суттєве практичне значення, оскільки створена система може бути інтегрована в

операційну діяльність складів малого та середнього бізнесу без значних витрат на інфраструктуру. Гнучкість архітектурних рішень дозволяє легко адаптувати систему під специфічні процеси конкретного підприємства.

Перспективами подальшого розвитку системи є впровадження мобільного застосунку, модулів машинного навчання для прогнозування залишків, оптимізації маршруту комплектування та автоматичного виявлення аномалій у логістичних процесах. Також можливо інтегрувати систему з обладнанням складу, таким як сканери штрихкодів, вагові станції та роботизовані комплекси.

Таким чином, розроблена система є завершеним та конкурентоспроможним рішенням для автоматизації складських процесів, здатним забезпечити цифровізацію логістики, скорочення витрат, підвищення продуктивності, зменшення кількості помилок та ефективну підтримку підприємств різного масштабу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Object-Relational Mapping [Електронний ресурс] — Режим доступу до ресурсу: https://martinfowler.com/eaaCatalog/objectRelationalMapping.html

2. Client—Server Architecture [Електронний ресурс] — Режим доступу до ресурсу: https://www.techtarget.com/searchnetworking/definition/client-server

3. Java Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://docs.oracle.com/javase/

4. JavaScript Guide [Електронний ресурс] — Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

5. Spring Framework Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://spring.io/projects/spring-framework

6. Hibernate ORM Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://hibernate.org/orm/documentation/

7. Document Object Model (DOM) Specification [Електронний ресурс] — Режим доступу до ресурсу: https://dom.spec.whatwg.org/

8. IndexedDB API [Електронний ресурс] — Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

9. React Official Documentation [Електронний ресурс] — Режим доступу до ресурсу : https://react.dev/

10. REST Architectural Constraints [Електронний ресурс] — Режим доступу до ресурсу : https://restfulapi.net/

11. PostgreSQL Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.postgresql.org/docs/

12. ACID Transactions Explained [Електронний ресурс] — Режим доступу до ресурсу : https://www.databricks.com/glossary/acid-transactions

13. Redis Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://redis.io/documentation

14. Bcrypt Password Hashing Function [Електронний ресурс] — Режим доступу до ресурсу: https://pkg.go.dev/golang.org/x/crypto/bcrypt

15. JSON Web Token (JWT) Introduction [Електронний ресурс] — Режим доступу до ресурсу: https://jwt.io/introduction

16. Entity—Relationship Diagram (ERD) Basics [Електронний ресурс] — Режим доступу до ресурсу: https://vertabelo.com/blog/what-is-entity-relationship-diagram/

17. HTML Living Standard [Електронний ресурс] — Режим доступу до ресурсу: https://html.spec.whatwg.org/

18. LiqPay API Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.liqpay.ua/documentation/en

19. DataTrans Payment Gateway Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.datatrans.ch/en/developers/

20. Testcontainers Java Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.testcontainers.org/

21. Apache Kafka Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://kafka.apache.org/documentation/

22. JUnit 5 User Guide [Електронний ресурс] — Режим доступу до ресурсу: https://junit.org/junit5/docs/current/user-guide/

23. Mockito Framework Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://site.mockito.org/

24. Postman API Platform [Електронний ресурс] — Режим доступу до ресурсу: https://www.postman.com/

25. Docker Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://docs.docker.com/

26. Elasticsearch Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html