ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

Факультет комп'ютерно-інформаційних технологій
Кафедра комп'ютерних інформаційних систем і технологій

Воровченко Ігор Романович
(прізвище, ім'я, по-батькові студента)

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«Розробка веб-інформаційної системи управління замовленнями для інтернет-магазину»**

Група: ІК-9-22-Б1КНТ(4.0д)

Спеціальність: Комп`ютерні Науки

Рівень вищої освіти: перший (бакалавр)

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

_____ Воровченко Ігор Романович
(підпис) (ПІБ студента)

Науковий керівник _____ Дуднік Андрій Сергійович
(підпис) (ПІБ)

Допущено до захисту ЕК

Завідувач кафедри _____ Сергій КАВУН , д.е.н., професор
(підпис)

Київ 2026

АНОТАЦІЯ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 59 с., 39 рис., 6 додатків, 28 джерел.

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ЗАМОВЛЕННЯМИ, ІНТЕРНЕТ-МАГАЗИН, МУЛЬТИСКЛАД, FIREBASE, ВЕБ-ДОДАТОК, JAVASCRIPT, АНАЛІТИКА, ЛОГУВАННЯ, ЧАТ.

Об’єктом дослідження є процес управління замовленнями в інтернет-магазині та організація взаємодії між користувачами системи.

Метою роботи є розробка веб-інформаційної системи для обробки замовлень, яка забезпечує автоматизацію основних бізнес-процесів, підвищує швидкість виконання операцій та зменшує кількість помилок. Система повинна підтримувати роботу з кількома складами, забезпечувати облік товарів, контроль залишків, а також повний цикл обробки замовлення.

Методи розробки базуються на використанні веб-технологій, зокрема HTML5, CSS, JavaScript, а також хмарної платформи Firebase для зберігання даних, автентифікації користувачів та роботи в режимі реального часу.

Результатом роботи є розробка функціональної інформаційної системи управління замовленнями з підтримкою мультискладської логіки, аналітичного модуля, системи логування дій та внутрішнього чату для співробітників. Розроблена система може бути використана для автоматизації процесів в інтернет-магазинах різного масштабу.

ORDER MANAGEMENT INFORMATION SYSTEM, ONLINE STORE, MULTI-WAREHOUSE, FIREBASE, WEB APPLICATION, JAVASCRIPT, ANALYTICS, LOGIN, CHAT.

The object of the study is the process of order management in an online store and the organization of interaction between system users.

The purpose of the work is to develop a web information system for order processing, which provides automation of basic business processes, increases the speed of operations and reduces the number of errors. The system should support work with multiple warehouses, provide accounting of goods, control of balances, as well as a full order processing cycle.

Development methods are based on the use of web technologies, in particular HTML5, CSS, JavaScript, as well as the Firebase cloud platform for data storage, user authentication and real-time operation.

The result of the work is the development of a functional information system for order management with support for multi-warehouse logic, an analytical module, an action logging system, and an internal chat for employees. The developed system can be used to automate processes in online stores of various scales.

ЗМІСТ

ВСТУП

Сьогодні більшість бізнес-процесів у компаніях так чи інакше пов’язані з цифровими технологіями. Особливо це помітно в електронній комерції: інтернет-магазини вже давно стали не просто додатковим каналом продажів, а основним джерелом доходу для багатьох компаній. Разом із цим зростає і навантаження — більше замовлень, більше клієнтів, більше очікувань щодо швидкості обробки та якості сервісу. У таких умовах питання ефективного управління замовленнями виходить на перший план.

На практиці цей процес часто організований доволі просто - через Excel-таблиці або навіть паперові записи. Працює це нестабільно: інформація губиться, дані не завжди актуальні, виникають затримки. Як наслідок — помилки в замовленнях, складнощі у взаємодії між співробітниками, проблеми з контролем. Усе це прямо впливає на якість обслуговування клієнтів і змушує шукати більш надійні рішення.

Один із варіантів — перехід на спеціалізовані веб-системи. Вони дозволяють зібрати всі дані в одному місці, автоматизувати ключові операції та спростити комунікацію між користувачами. Плюс — доступ до системи з будь-якого пристрою, що значно полегшує роботу. У підсумку це дає більш швидку обробку замовлень, менше помилок і кращу керованість процесами.

Метою дипломної роботи є розробка веб-інформаційної системи для управління замовленнями в інтернет-магазині, яка допоможе спростити роботу з замовленнями та зробити внутрішні процеси більш ефективними.

Для досягнення цієї мети було поставлено такі завдання:

- розібратися в тому, як працюють інтернет-магазини та як організований процес обробки замовлень;
- проаналізувати існуючі підходи до автоматизації;
- визначити їхні сильні та слабкі сторони;
- сформувати вимоги до майбутньої системи;

- продумати архітектуру та структуру бази даних;
- реалізувати основний функціонал;
- перевірити роботу системи та оцінити результат.

Об’єкт дослідження — процес управління замовленнями в інтернет-магазині.

Предмет — веб-інформаційна система, яка автоматизує цей процес.

Під час виконання роботи використовувалися різні методи. Аналіз допоміг розібрати структуру процесу обробки замовлень і зрозуміти, як це реалізується в інших системах. Порівняння дало можливість оцінити різні підходи та вибрати найбільш доцільні. Системний підхід дозволив розглядати інтернет-магазин як єдину систему, де всі елементи пов’язані між собою. Також застосовувалося моделювання — для опису структури системи та її компонентів.

Робота складається з трьох розділів. У першому розглядається теорія: як працюють інтернет-магазини, як організовується обробка замовлень і яку роль у цьому відіграють інформаційні системи. Другий розділ присвячений аналізу існуючих рішень, їхніх обмежень і проблем. У третьому розділі описано створення власної системи - від вимог і структури до реалізації функціоналу.

У підсумку вдалося сформувати цілісне бачення процесу управління замовленнями та показати, як його можна покращити за допомогою сучасних веб-технологій.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ЗАМОВЛЕННЯМИ В ІНТЕРНЕТ-МАГАЗИНІ

1.1 Опис предметної області

Останні кілька років електронна комерція росте дуже швидко — і це вже ні для кого не новина.[1] Люди звикли купувати онлайн: це зручно, швидко і не залежить від того, де ти знаходишся.[4] У результаті інтернет-магазини все більше витісняють класичну торгівлю. Але разом із цим зростають і вимоги до

“внутрішньої кухні” таких систем, особливо до того, як обробляються замовлення.[5]

У цій роботі розглядається саме ця частина — як інтернет-магазин приймає, обробляє і доводить замовлення до завершення. Фактично замовлення — це центр усього процесу. Саме через нього відбувається взаємодія між клієнтом і системою. Користувач обирає товар, оформлює покупку, а система вже “підхоплює” це і запускає далі весь ланцюг дій. При цьому формується набір даних: що саме замовили, скільки одиниць, хто покупець, як оплачувати і куди доставляти.

Як це виглядає на практиці. Спочатку клієнт заходить на сайт, додає товари в кошик і оформлює замовлення. Нічого складного — стандартний сценарій, який всі знають. Після цього інформація передається в систему, де замовлення фіксується і отримує свій статус (наприклад, “нове” або “в обробці”).

Далі починається етап обробки. Тут вже підключаються працівники або автоматизовані механізми: перевіряється, чи є товар у наявності, чи все правильно заповнено, чи немає помилок. Якщо чогось не вистачає — можуть зв’язатися з клієнтом або скоригувати замовлення. Після підтвердження воно передається на склад.

На складі відбувається комплектація. По суті, це збір замовлення: працівник бере потрібні товари відповідно до списку. І тут важливий момент — будь-яка помилка(не той товар, не та кількість) одразу впливає на клієнта. Після збору замовлення перевіряють ще раз, пакують і готують до відправки.

Фінальний етап — доставка. Замовлення передається службі доставки, а в системі змінюється його статус, щоб і клієнт, і працівники розуміли, що відбувається.

Окремо варто згадати про інформацію, яка “ходить” між усіма цими етапами. Дані постійно передаються: від клієнта до системи, далі на склад, потім до служби доставки. Якщо немає єдиної системи, починаються типові проблеми

— затримки, плутанина, різні версії одних і тих самих даних. У підсумку це ускладнює контроль і гальмує роботу.

До речі, у багатьох інтернет-магазинах досі використовують досить прості або навіть застарілі підходи. Наприклад, ведуть облік у Excel, передають інформацію вручну або користуються кількома окремими програмами, які між собою не “спілкуються”. Це створює додаткове навантаження на працівників і підвищує ризик помилок. Як наслідок — замовлення обробляються довше, а якість сервісу падає.

Саме тому автоматизація тут відіграє ключову роль. Якщо всі процеси об’єднані в одній веб-системі, стає набагато простіше: дані зберігаються централізовано, оновлюються в реальному часі, менше шансів щось загубити або переплутати. Плюс це економить час і робить роботу більш прозорою.

Ще один важливий момент — це користувачі системи. З одного боку є клієнти, які оформлюють замовлення. З іншого — працівники (наприклад, адміністратори або комплектувальники), які їх обробляють.

Саме сучасні веб-системи дозволяють привести це все до ладу і зробити сервіс для клієнта справді зручним.

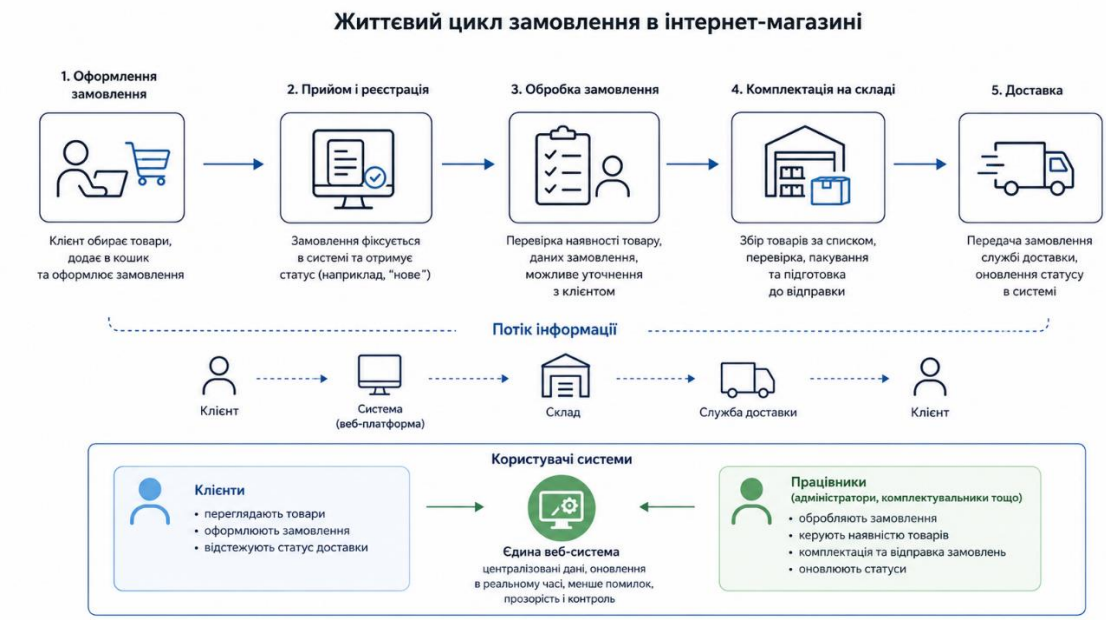


Рис. 1.1 - Процес управління замовленнями в інтернет магазині

Отже, можна зазначити, що управління замовленнями є складним багатогранним процесом, що поєднує інформаційні та фізичні операції. Його ефективність залежить від узгодженості дій усіх учасників і рівня автоматизації процесів. Впровадження сучасних веб-інформаційних систем дозволяє значно покращити організацію роботи інтернет-магазину та створює основу для підвищення якості обслуговування клієнтів.

1.2 Організація процесу обробки замовлень

Обробка замовлень — це по суті “серце” будь-якого інтернет-магазину. Саме тут звичайний клік користувача перетворюється на реальну покупку. І від того, наскільки цей процес налагоджений, залежить дуже багато: і швидкість роботи, і кількість помилок, і те, чи залишиться клієнт задоволений.

Як усе відбувається на практиці. Спочатку клієнт оформлює замовлення через сайт: обирає товари, додає їх у кошик, вводить свої дані, обирає спосіб доставки й оплати. У цей момент у системі створюється запис — фактично повна “картка” замовлення з усією потрібною інформацією.

Далі замовлення реєструється і отримує свій статус. Це може бути щось типу “нове”, “в обробці” тощо. Такі статуси — не просто формальність, вони дозволяють нормально відстежувати, що зараз відбувається із замовленням і на якому воно етапі.

Після цього йде перевірка. Тут дивляться, чи правильно введені дані, чи є товар на складі, чи все ок з оплатою (якщо вона вже проведена). Якщо знаходяться якісь неточності — наприклад, товару немає або щось не сходиться в даних — доводиться уточнювати. Іноді навіть зв’язуються з клієнтом, щоб не допустити помилки далі.[3]

Коли все перевірено, замовлення передається на склад. Там починається комплектація — тобто збір товарів. Звучить просто, але на практиці це один із

найвідповідальніших моментів. Якщо переплутати товар або кількість — клієнт це одразу помітить. Тому тут важлива уважність і нормальна організація процесу.

Після збору замовлення пакують: перевіряють ще раз, складають, маркують. Далі воно передається службі доставки. У системі при цьому оновлюється статус, щоб і клієнт міг подивитися, де зараз його посилка.

Фінал — це доставка і отримання замовлення. Після цього процес фактично завершується. Хоча іноді додається ще один крок — зворотний зв’язок. Наприклад, клієнт може залишити відгук або оцінити сервіс. Це дрібниця, але дає розуміння, що можна покращити.

Важливий момент, який часто недооцінюють — це рух даних між усіма цими етапами. Інформація постійно передається: від сайту до сервера, далі до бази даних, на склад, у доставку. Якщо ця взаємодія не налагоджена, починаються типові проблеми: щось загубилося, щось продублювалося, десь затримка. І вся система починає “гальмувати”.

У реальності багато магазинів досі працюють неідеально. Десь використовують Excel, десь частину процесів роблять вручну, десь програми між собою не пов’язані. У підсумку працівники витрачають більше часу, ніж потрібно, і частіше помиляються.

Саме тому автоматизація тут реально вирішує. Коли всі етапи об’єднані в одній системі, усе працює значно простіше: дані не губляться, статуси оновлюються автоматично, менше ручної роботи. Плюс з’являється нормальний контроль за всім процесом — від моменту оформлення до доставки.

Якщо коротко, добре організований процес обробки замовлень — це коли все рухається без зайвих затримок, без плутанини і з мінімумом ручного

втручання. І саме це напряму впливає на те, наскільки ефективно працює інтернет-магазин і чи захоче клієнт повернутися ще раз.

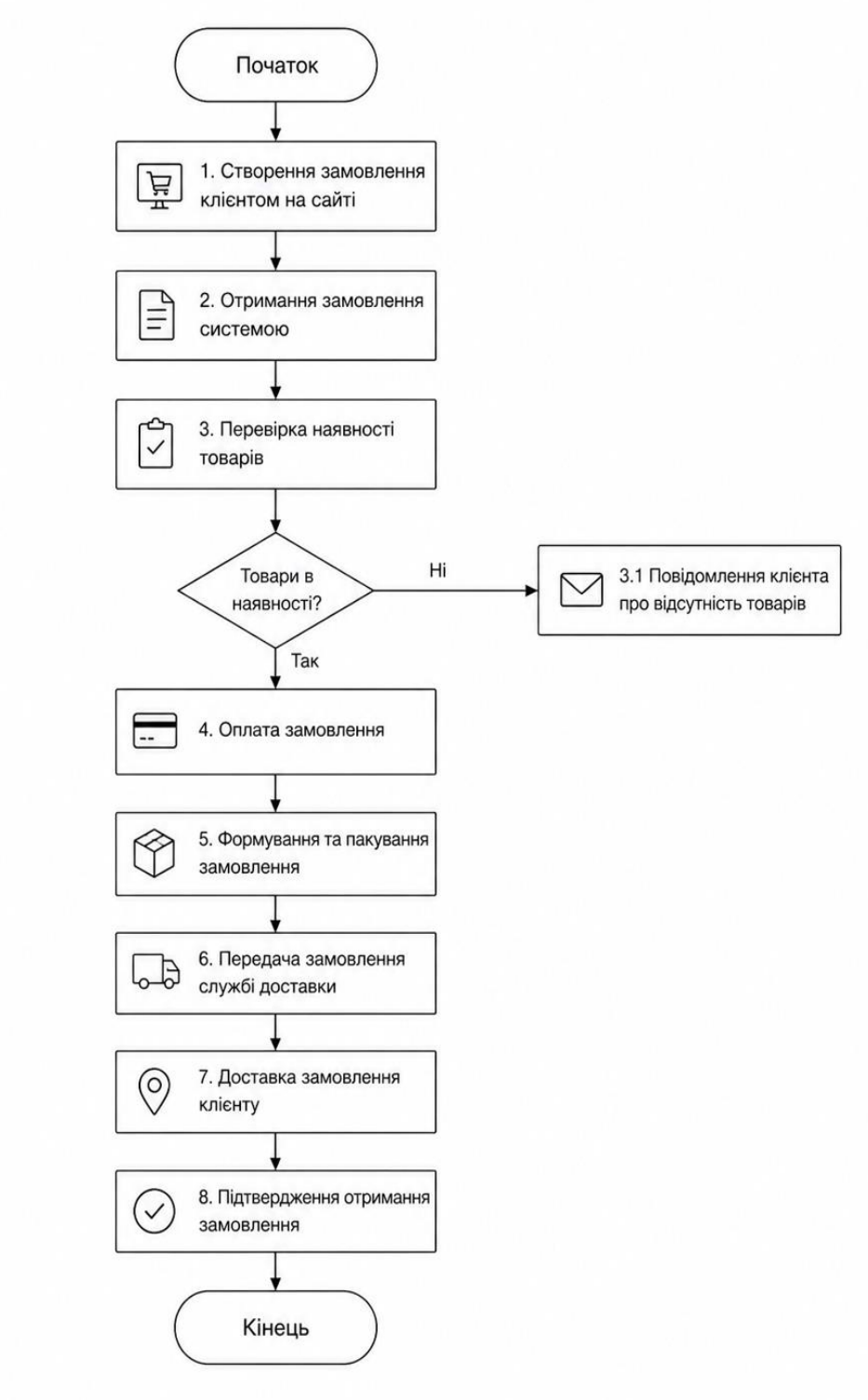


Рисунок 1.2 – Блок-схема процесу обробки інтернет замовлення

1.3 Інформаційні системи управління замовленнями

У сучасній електронній комерції вже складно уявити нормальну роботу інтернет-магазину без інформаційних систем. Кількість замовлень росте, клієнти очікують швидкої реакції, і “вручну” це все просто не витягнути. Тому автоматизація тут — не якась опція, а скоріше необхідність. Саме такі системи дозволяють зібрати всі етапи роботи із замовленнями в одному місці і не губитися в процесах. [2]

Якщо говорити простіше, інформаційна система управління замовленнями — це набір інструментів, які беруть на себе рутину: приймають замовлення, обробляють їх, зберігають дані і допомагають контролювати, що відбувається на кожному етапі. Вона фактично зв’язує між собою користувачів, базу даних і всі інші частини системи. За рахунок цього зникає хаос із даними — менше дублювання, менше помилок, легше щось знайти.

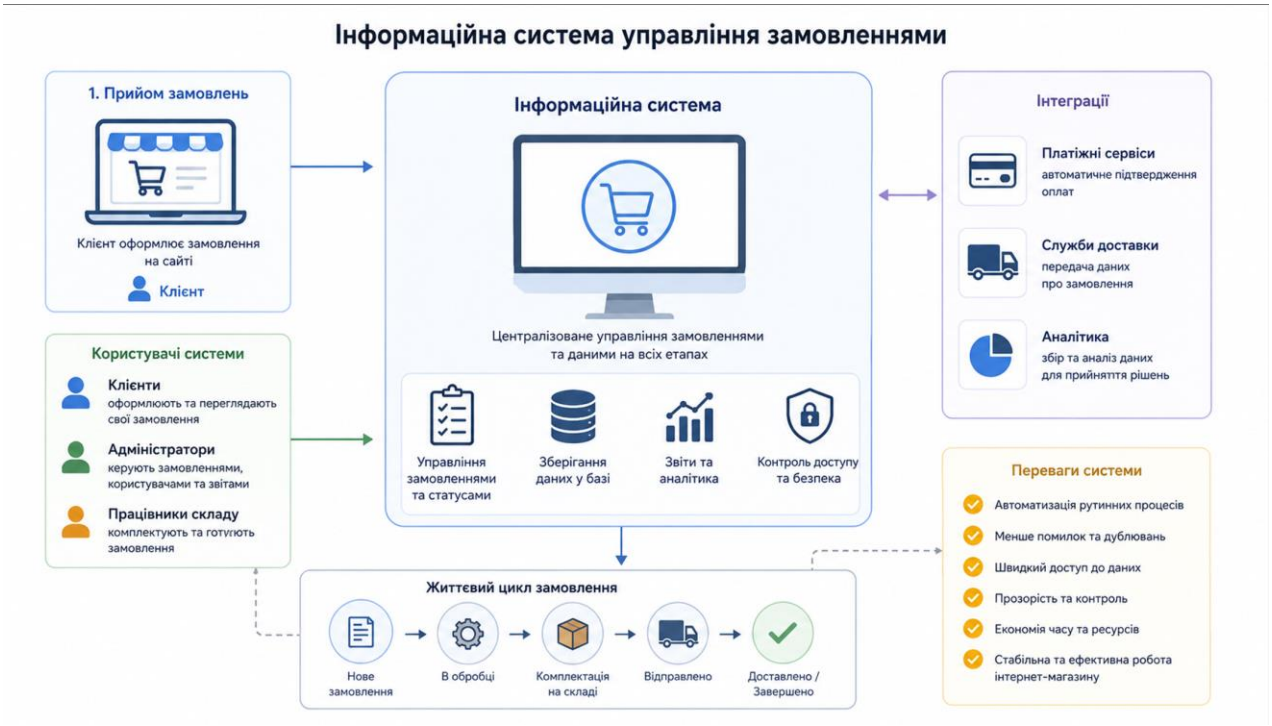


Рис. 1.3 – Діаграма інформаційних систем управління замовленнями

Що така система робить на практиці? По-перше, фіксує замовлення: хто замовив, що саме, куди доставити. Далі — дозволяє керувати статусами, щоб було зрозуміло, чи замовлення ще нове, вже в обробці або, наприклад,

відправлене. Плюс контроль усіх дій і можливість подивитися звіти — це особливо корисно, коли замовлень стає багато і “на око” вже нічого не зрозумієш.

Окрема історія — це база даних. Саме там зберігається вся інформація: товари, клієнти, замовлення. І тут є нюанс: якщо база зроблена абияк, система починає гальмувати, щось губиться або працює нестабільно. А якщо все продумано нормально — доступ до даних швидкий, працювати зручно, і шансів на помилки значно менше.

Ще один плюс — інтеграції. Сучасні системи не існують “самі по собі”, вони підключаються до платіжних сервісів, служб доставки, аналітики. Наприклад, оплата підтягнулась автоматично — і не потрібно вручну це перевіряти. Або дані по доставці одразу передалися в потрібну службу. Це економить час і знімає частину навантаження з працівників.

Щодо типів систем — є локальні і веб-орієнтовані. Локальні ставляться на конкретні комп’ютери і мають обмежений доступ. Веб-системи працюють через браузер, і до них можна зайти з будь-якого місця. Для інтернет-магазинів логічніше використовувати саме веб-варіант — це і гнучкіше, і зручніше, особливо якщо команда працює не в одному місці.

Не менш важливий момент — ролі користувачів. У системі зазвичай є клієнти, адміністратори і, наприклад, працівники складу. І кожен бачить тільки те, що йому потрібно для роботи. Це не лише про зручність, а й про безпеку — зайві дані нікому не відкриваються.

Отже, впровадження таких систем — це не завжди просто. Потрібен час на розробку, налаштування, підтримку. Плюс іноді доводиться підлаштовувати систему під реальні бізнес-процеси, які можуть змінюватися. Але в підсумку це виправдовується: робота стає швидшою, зрозумілішою і значно стабільнішою. І для інтернет-магазину це, по суті, база, без якої далеко не заїдеш.

1.4 Висновки до першого розділу

У першому розділі розібрано базові речі щодо того, як працює управління замовленнями в інтернет-магазині. По суті, це дає розуміння всієї логіки процесу — від моменту, коли клієнт оформлює покупку, до того, як товар реально потрапляє до нього. І стає видно, що вся електронна комерція тримається не тільки на сайті, а на тому, як всередині організована робота із замовленнями.

Як показав аналіз, замовлення проходить кілька етапів: спочатку оформлення, далі перевірка, потім збір товару, пакування і вже після цього доставка. На кожному з цих кроків постійно рухається інформація, і якщо десь щось “випадає”, це одразу тягне за собою проблеми далі по ланцюгу. Через це навіть дрібні помилки можуть впливати на весь результат.

Окремо розглянуто саму організацію цього процесу. У реальній роботі найчастіше складнощі з’являються там, де все тримається на ручних діях або розрізнених інструментах. Це виглядає ніби дрібниця, але саме тут виникають затримки, плутанина і зайве навантаження на працівників. У підсумку частина часу витрачається не на реальну роботу, а на виправлення дрібних помилок.

Коли мова йде про інформаційні системи, їх роль стає досить очевидною. Вони просто прибирають хаос із процесів і зводять усе в одну систему, де дані не губляться і не дублюються. Особливо це помітно у веб-рішеннях, бо вони не прив’язані до одного комп’ютера і нормально працюють у різних умовах, плюс легко підключаються до інших сервісів.

Також важливим моментом є розподіл доступу між користувачами. У системі є різні ролі, і кожна з них бачить тільки те, що потрібно для роботи. Це банально зменшує кількість помилок і робить процес більш контрольованим. Паралельно з цим працює база даних, де зберігається вся інформація про замовлення, товари і клієнтів — без цього система просто не могла б нормально функціонувати.

Як підсумок, видно, що без автоматизованої системи управління замовленнями інтернет-магазин швидко впирається в обмеження: ручна робота, помилки, затримки. Перехід до веб-системи тут виглядає не як покращення “для зручності”, а як необхідний крок, щоб процес взагалі працював стабільно і без постійних збоїв.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПРОЦЕСУ УПРАВЛІННЯ ЗАМОВЛЕННЯМИ

2.1 Аналіз діяльності інтернет-магазину

Інтернет-магазини зараз фактично стали базовим інструментом у сфері електронної комерції. Вони дозволяють продавати товари без прив’язки до місця і значно спрощують взаємодію з клієнтом. Але вся ця “зручність ззовні” тримається на доволі складній внутрішній системі процесів — від показу товару на сайті до моменту доставки.

Як правило, робота інтернет-магазину складається з кількох частин. Це веб-платформа, склад, обробка замовлень і доставка. Веб-сайт тут — це не просто сторінка з товарами, а точка входу, через яку проходить усе: вибір товару, оформлення замовлення і передача даних далі в систему.

Склад відповідає за просту, але критичну річ — щоб товар реально існував і був готовий до відправки. Далі включається обробка замовлень: перевірка, підтвердження, контроль статусів. І вже фінальний етап — доставка, яка закриває весь цикл.

На практиці майже вся ця система зав’язана на інформаційних технологіях. Без них інтернет-магазин просто не міг би працювати в нормальному темпі. Але проблема в тому, що не всюди ці процеси автоматизовані повністю. Часто частина роботи все ще виконується вручну, і саме тут починаються затримки та помилки.

Найважливіший елемент у всій цій структурі — обробка замовлень. Вона запускається в момент, коли клієнт натискає “оформити”, і закінчується тільки

тоді, коли товар уже доставлено. По дорозі це замовлення проходить через кілька етапів і різні підрозділи, і кожна передача даних — це потенційне місце для помилки, якщо система не узгоджена.

Більшість проблем в інтернет-магазинах якраз і виникає на цьому рівні. Це можуть бути затримки в обробці, неправильна комплектація, або ситуації, коли статус замовлення не відповідає реальному стану. Усе це напряму б’є по якості сервісу і, по суті, по довірі клієнтів.

Окремо варто виділити роботу з товарними залишками. Якщо інформація про склад не оновлюється вчасно, виникають типові ситуації — замовлення оформлено, а товару вже немає. Коли немає нормальної системи обліку, це доводиться перевіряти вручну, що тільки збільшує час обробки і створює зайве навантаження.

Ще один критичний момент — це рух даних між усіма частинами системи. Інформація про клієнтів, товари і замовлення повинна оновлюватися і бути доступною без затримок.[21] Якщо цього немає, починаються розбіжності: різні підрозділи працюють з різними даними, і процес управління замовленнями стає нестабільним.

2.2 Аналіз процесу обробки замовлень

Процедура опрацювання замовлень в інтернет-магазині є однією з ключових частин усієї системи електронної комерції. Саме тут звичайний вибір товару перетворюється на реальне замовлення, яке далі проходить кілька етапів до моменту доставки. Від того, як це організовано, залежить швидкість роботи магазину, точність виконання замовлень і те, як клієнт загалом оцінює сервіс.

Все розпочинається з оформлення замовлення через вебсайт. Користувач додає товари які цікавлять у кошик, вводить дані для доставки і підтверджує купівлю. Після цього система формує запис із інформацією про замовлення. На цьому етапі важливо, щоб дані були введені без помилок, бо навіть невеликі неточності потім можуть ускладнити опрацювання.

Далі замовлення потрапляє в систему і реєструється зі своїм статусом. Після цього воно проходить верифікацію: дивляться наявність товарів, правильність введеної інформації, іноді перевіряється платіж. Якщо щось не сходиться, доводиться уточнювати деталі, і це вже напряму впливає на тривалість виконання.

Після верифікації починається комплектація. Це стадія, коли на складі збирають замовлення. Працівники відбирають товари згідно зі списком і готують їх до відвантаження. Тут часто виникають негаразди, якщо інформація про залишки неактуальна або якщо частина процедури здійснюється вручну.

Далі йде пакування та підготовка до відправлення. Замовлення перевіряють ще раз, упаковують і передають службі доставки. На цьому етапі важливо, щоб не було помилок у документах і щоб передача даних відбувалась без затримок, інакше це впливає на строки доставки.

Завершальний етап — доставка клієнту. Тут важливу роль відіграє можливість відстеження замовлення, щоб було зрозуміло, на якому воно етапі. Якщо інтеграції зі службами доставки немає, інформація часто оновлюється із затримкою або взагалі неповна.

Під час аналізу процесу видно типові проблеми: затримки в обробці, помилки при введенні або передачі даних, неактуальні залишки на складі та слабка координація між підрозділами. Усе це впливає на швидкість роботи і якість обслуговування.[25]

Окремо варто згадати інформаційні потоки. Дані постійно передаються між етапами, і якщо немає єдиної системи, вони починають дублюватися або втрачати актуальність. Через це складніше контролювати процес і частіше виникають помилки.

У результаті видно, що основне вдосконалення можливе через впровадження веб-інформаційної системи. Вона дозволяє об’єднати всі процеси,

зменшити кількість ручних операцій та зробити обробку замовлень більш стабільною та передбачуваною.

2.3 Порівняння існуючих систем та підходів

Коли розглядаються підходи до управління замовленнями, має сенс не обмежуватись загальним описом, а подивитися на конкретні приклади. Саме так краще видно, як усе працює на практиці — де зручно, а де починаються проблеми.

Найпростіший та найдешевший варіант — ручний підхід. Його зазвичай використовують на старті або в невеликих інтернет-магазинах, де обсяг замовлень ще невеликий. Найтиповіший приклад — звичайні таблиці, наприклад **Microsoft Excel** або **Google Sheets**.[8] У такому випадку вся інформація заноситься вручну: клієнти, товари, статуси. І так само вручну все оновлюється.

Плюс тут очевидний — нічого складного. Не потрібно впроваджувати окрему систему, достатньо базових інструментів. Але це працює рівно до того моменту, поки замовлень мало. Як тільки їх стає більше, починаються типові проблеми: помилки в даних, плутанина зі статусами, затримки в обробці. І контролювати все це стає все складніше.

Якщо трохи детальніше подивитися на Google Sheets, то він дійсно дає базовий набір можливостей. Можна налаштувати таблицю під замовлення, додати сортування, фільтри, формули, навіть підсвічувати статуси кольорами. Плюс є спільний доступ — кілька людей можуть працювати одночасно, і зміни видно одразу. Також легко встановити доступ.

Але на цьому, по суті, переваги закінчуються. Жодної вбудованої логіки для роботи із замовленнями там немає. Наприклад, система не перевірить залишки на складі, не підкаже, що товар закінчився, і не зв’яжеться з доставкою чи оплатою. Усе це доводиться робити вручну: вводити, перевіряти, виправляти.

Через це зростає кількість помилок і сповільнюється робота. Особливо це помітно, коли замовлень стає більше і таблиця починає розростатися. У якийсь момент такий підхід просто перестає справлятися з навантаженням.

Тому використання таблиць має сенс тільки на початковому етапі або при дуже невеликій кількості замовлень. Далі, як правило, виникає потреба перейти на більш автоматизоване рішення, де частина процесів уже не залежить від ручного введення і перевірок.

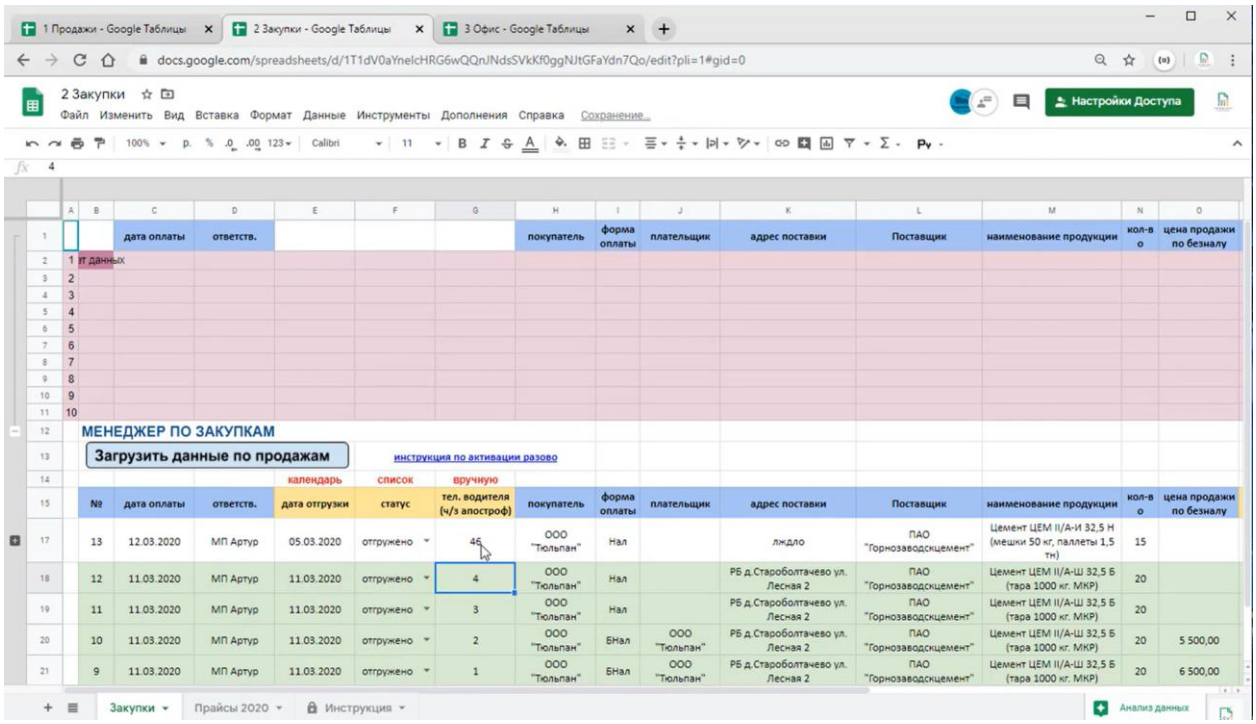


Рис.2.1 - Приклад ручної системи обліку Google Sheets

Другий варіант — це вже використання повноцінних веб-систем, які спочатку створені саме для роботи з замовленнями. До таких рішень можна віднести, наприклад, **Shopify**, **WooCommerce** або **OpenCart**.

У цьому випадку багато процесів вже автоматизовані. Замовлення оформлюється на сайті — далі система сама його обробляє: створює запис, змінює статуси, дозволяє працювати з оплатою і доставкою. Тобто немає потреби вручну вести облік, як це робиться в таблицях.

Такі платформи зазвичай уже мають інтеграції з платіжними сервісами і службами доставки. Це означає, що не потрібно окремо “зв’язувати” різні системи між собою. Також є інструменти для роботи з товарами — додавання, редагування, контроль залишків. Плюс базова аналітика, щоб розуміти, що продається і як іде процес.

Головний плюс тут — швидкість і менше помилок. Дані обробляються автоматично, усе знаходиться в одному місці, і за рахунок цього легше контролювати ситуацію.

Але не все так просто. Впровадження таких систем часто потребує витрат — як фінансових (як правило – сервіси працюють по підписці), так і по часу. Особливо якщо потрібно щось налаштовувати під конкретні задачі. Іноді стандартного функціоналу вистачає, а іноді доводиться дороблятиабо підключати додаткові модулі. [6]

Shopify

Як приклад більш готового рішення для роботи із замовленнями можна взяти Shopify. Цю платформу часто використовують, коли потрібно не писати все з нуля, а просто запустити магазин і одразу почати працювати.

По суті, там вже багато чого зроблено за користувача. Є управління товарами, замовленнями, оплатами, доставкою. Не потрібно окремо думати, як це зв’язати між собою — воно вже працює разом. Це зручно, особливо на старті.

Коли клієнт оформлює замовлення, система сама його створює і далі веде по етапах. Статуси змінюються, можна бачити, що вже зроблено, що ще ні. Оплата теж обробляється всередині платформи, якщо підключені відповідні сервіси. Тобто багато процесів відбуваються автоматично, без ручного втручання.

Є і робота зі складом, але вона не дуже глибока. Можна подивитись залишки, відстежити продажі, зрозуміти, що закінчується. Для невеликого магазину цього вистачає. Для чогось більш складного — вже не завжди.

Ще один плюс — швидкий старт. Є шаблони, можна зібрати сайт буквально за короткий час. Але тут є нюанс: поки все стандартно — працює добре. Як тільки потрібно щось нестандартне, починаються складнощі.

Shopify працює по підписці. Тобто потрібно платити регулярно. Крім цього, є певна залежність від платформи — дані і логіка знаходяться не локально, а у сервісі. Це нормально, але не завжди зручно.

Ще момент — обмеження в налаштуванні. Не всі речі можна змінити під себе. Якщо потрібна своя логіка обробки замовлень або якісь специфічні процеси, доводиться або обходити це, або підключати додаткові додатки. Інколи це ускладнює систему.

Якщо підсумувати, Shopify — це хороший варіант, коли потрібно швидко запустити інтернет-магазин і не витрачати час на розробку. Але при більш складних задачах може не вистачати гнучкості. [7]

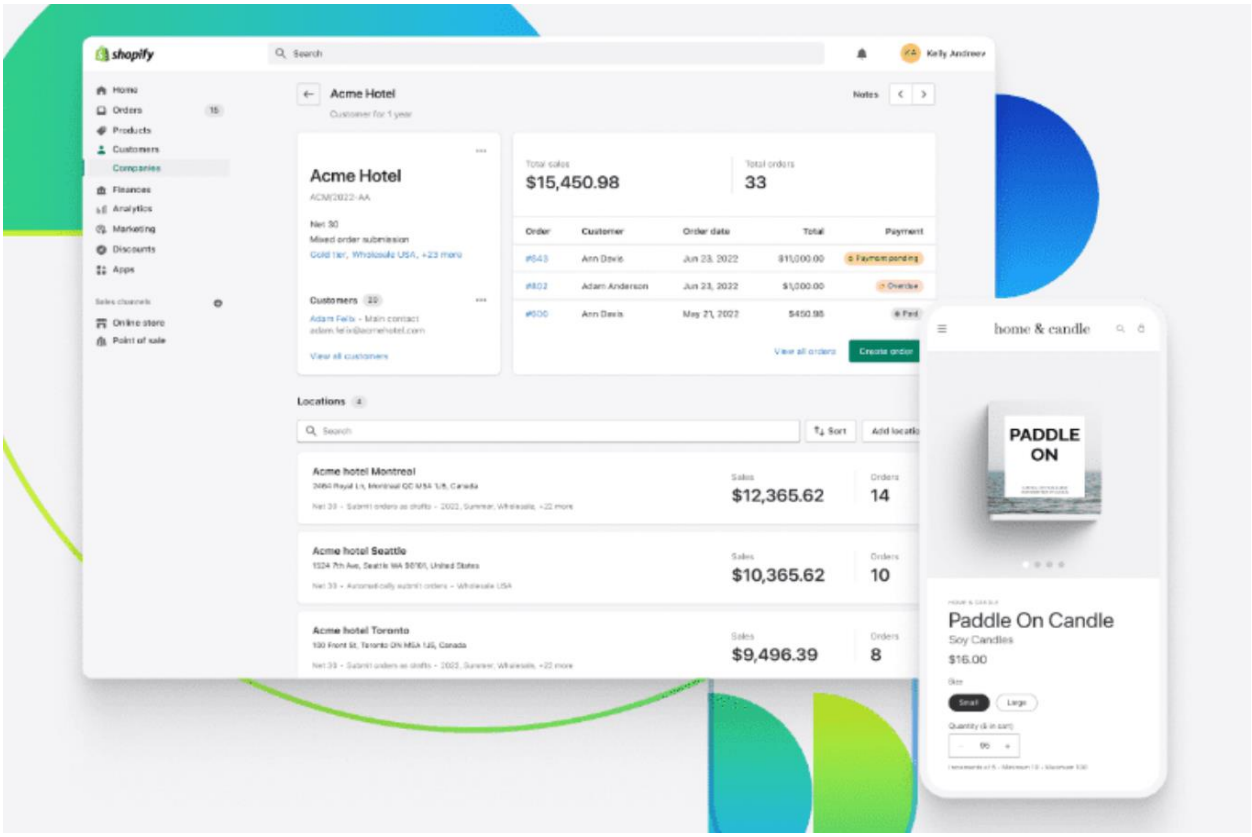


Рис.2.2 - Демонстрація платформи Spopify

Однак навіть такі системи можуть мати певні обмеження. Зокрема, готові платформи не завжди дозволяють повністю адаптувати функціонал під специфіку конкретного підприємства. У деяких випадках виникає необхідність у додатковій розробці або інтеграції сторонніх сервісів.

Внаслідок цього підприємства можуть обирати шлях створення власної веб-інформаційної системи, яка максимально відповідає їхнім потребам.

Критерій	Ручний підхід	Повноцінна веб-система
Швидкість обробки	Низька	Висока
Ймовірність помилок	Висока	Низька
Контроль виконання	Ускладнений	Повний
Актуальність даних	Низька	Висока (у реальному часі)
Масштабованість	Відсутня	Висока
Інтеграція з іншими системами	Відсутня	Повна
Зручність використання	Низька	Висока
Витрати на впровадження	Мінімальні	Середні\високі

Таблиця 2.1 - Порівняння існуючих підходів управління замовленнями

Порівняльний аналіз показує, що ручні підходи не здатні забезпечити необхідний рівень ефективності в умовах зростання обсягів замовлень. Спеціалізовані системи, у свою чергу, значно покращують організацію процесів, проте можуть мати обмеження у гнучкості та адаптації. Веб-інформаційні системи поєднують у собі переваги автоматизації та можливість індивідуального налаштування, що робить їх найбільш доцільним варіантом для сучасних інтернет-магазинів.

2.4 Висновок до другого розділу

Отже, після аналізу різних підходів до управління замовленнями стало зрозуміло, що для нормальної роботи інтернет-магазину вже недостатньо простих таблиць або окремих сервісів, які не пов’язані між собою. Коли замовлень мало — це ще працює. Але зі зростанням навантаження починають з’являтися затримки, плутанина в даних і банальні помилки, які потім напряму впливають на клієнтів.

Якщо говорити про ручний підхід, то найчастіше це звичайні таблиці в Google Sheets або Microsoft Excel. На старті цього цілком вистачає: можна вести список замовлень, змінювати статуси, дивитися залишки товарів. Але проблема в тому, що все тримається фактично на уважності працівника. Дані потрібно оновлювати вручну, перевіряти, контролювати. І чим більше замовлень — тим важче це робити без помилок.

Частково автоматизовані рішення виглядають уже краще. Наприклад, CRM-системи на кшталт Pipedrive, Odoo або KeyCRM дозволяють трохи впорядкувати процес: вести клієнтів, контролювати статуси замовлень, бачити історію взаємодії. Це справді спрощує роботу, особливо коли магазин починає рости. Але тут теж є свої нюанси. Часто складський облік, доставка або аналітика знаходяться в окремих сервісах, через що інформація починає “розходитись” між системами. У результаті доводиться постійно щось синхронізувати або дублювати вручну.

Найбільш зручними виглядають уже повноцінні веб-системи, такі як Shopify, WooCommerce чи OpenCart. У них більшість процесів об’єднана в одному місці: замовлення, оплата, склад, доставка, статистика. Через це система працює більш цілісно, а кількість ручної роботи помітно зменшується. До того ж дані оновлюються швидше, і легше контролювати, що відбувається на кожному етапі.

Але навіть готові рішення не завжди закривають усе, що потрібно конкретному магазину. Десь не вистачає гнучкості, десь складно змінити логіку роботи під свої процеси. Плюс є залежність від сторонніх сервісів і додаткові витрати — наприклад, на підписки, модулі або інтеграції. Через це в багатьох випадках виникає потреба у власній веб-системі, яку можна адаптувати саме під свої задачі, а не підлаштовуватись під обмеження готової платформи.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ЗАМОВЛЕННЯМИ В ІНТЕРНЕТ-МАГАЗИНІ

3.1 Проєктування системи

З урахуванням результатів аналізу існуючих підходів, наведених у попередньому розділі, було встановлено необхідність створення власної веб-інформаційної системи, яка забезпечить комплексну автоматизацію бізнес-процесів та усуне недоліки існуючих рішень.

Основною задачею розробки є створення програмної системи, що дозволить автоматизувати процес обробки замовлень, починаючи від моменту їх оформлення і завершуючи етапом доставки товару. Система повинна забезпечувати централізоване зберігання даних, їх обробку в режимі реального часу, а також підтримку взаємодії між учасниками процесу, включаючи, адміністраторів та комплектувальників.[17]

У межах поставленої задачі визначено такі основні функціональні вимоги до системи:

1. Авторизація користувачів із розмежуванням прав доступу;
2. Формування та обробка замовлень;
3. Управління товарами на складі (додавання, редагування, видалення, переміщення);
4. Облік складських залишків;
5. Формування звітності та аналітики;
6. Логування усіх дій які відбувається в межах системи;

7. Зміна пріорітету замовлень;
8. Робочий чат співробітників;
9. Система мульти-складів: обробка, відправка замовлень з різних складів підприємства.

Окрім функціональних, система повинна відповідати низці нефункціональних вимог, які визначають якість її роботи. Зокрема, до них належать:

- **продуктивність** - здатність обробляти значну кількість запитів без втрати швидкодії;
- **масштабованість** - можливість розширення функціоналу та збільшення обсягів даних;
- **надійність** - забезпечення стабільної роботи та збереження даних;
- **безпека** - захист персональних даних користувачів та запобігання несанкціонованому доступу;
- **зручність використання** - інтуїтивно зрозумілий інтерфейс для різних категорій користувачів.

3.2 Архітектура системи та вибір технологій

Для реалізації задач у роботі було обрано клієнт-серверний підхід із використанням **HTML, CSS, JavaScript і Firebase**. Якщо говорити простіше, то це стандартний варіант для веб-застосунків, але з акцентом на те, щоб не піднімати власний сервер. Основна причина такого вибору — необхідність працювати з даними в реальному часі. У системі, де є замовлення і склад, це досить важливо, бо інформація постійно змінюється.

Firebase у цьому випадку використовується як вже готовий бекенд. Тобто частина, яка відповідає за серверну логіку, фактично винесена в хмару. Це зручно, тому що не потрібно окремо налаштовувати сервер, писати API “з нуля” і думати про розгортання. Базові речі вже є — і їх достатньо для більшості задач.[20]

Наприклад, для зберігання даних використовується база, яка працює в реальному часі. Це означає, що будь-які зміни одразу відображаються у всіх користувачів. Якщо хтось змінив статус замовлення або кількість товару — це видно без оновлення сторінки. На практиці це реально спрощує роботу, особливо коли декілька людей працюють одночасно.

Ще один момент — авторизація. У Firebase вона вже реалізована, тому не потрібно окремо продумувати механізм входу. Можна використати стандартні варіанти (email, пароль і т.д.), і цього вистачає. Це економить час, хоча іноді обмежує в гнучкості, якщо потрібні якісь нестандартні сценарії.

Якщо дивитись у цілому, Firebase дозволяє не відволікатись на технічні деталі бекенду. Замість цього більше уваги приділяється логіці системи і тому, як користувач із нею взаємодіє. І це, мабуть, одна з головних причин, чому він тут використаний.

З боку клієнта все більш звично. **HTML5** використовується для структури сторінок. Тобто всі елементи інтерфейсу — кнопки, форми, таблиці — задаються саме через нього. В двох словах – це скелет сторінки. Нічого складного, але без цього нікуди. [28]

CSS (каскадні таблиці стилів) — відповідає за вигляд. Через нього задається, як саме виглядає сторінка: кольори, шрифти, відступи, розташування елементів. Також через CSS реалізована адаптація під різні екрани. Це не суперскладно, але необхідно, бо користувач може зайти як з комп’ютера, так і з телефону. [23, 14]

JavaScript — це високорівнева мова програмування, виконує основну роботу. Саме через нього реалізована логіка: обробка дій користувача, перевірка введених даних, взаємодія з базою. Наприклад, при зміні статусу замовлення дані одразу відправляються в базу і синхронізуються з іншими користувачами. Без JavaScript система була б просто набором статичних сторінок. [22]

Іноді виникали ситуації, коли потрібно було обробляти дані асинхронно, наприклад при отриманні інформації з бази. Це вирішується стандартними засобами JavaScript[27], але на початку можуть бути затримки, поки не підвантажаться дані. Це не критично, але такі моменти варто враховувати.

Окремо можна згадати Node.js. Він не використовувався напряму, але сам факт, що JavaScript може працювати і на сервері, робить його універсальним. У цьому випадку це не було необхідно, бо Firebase вже закриває серверну частину.

У процесі розробки стало зрозуміло, що вибраний стек не є складним, але цього достатньо. Він дозволяє швидко зробити робочу систему без зайвих витрат часу. Звісно, у більш складних проєктах можуть знадобитися інші рішення, але для цієї задачі такий підхід виглядає цілком виправданим.

У підсумку можна сказати, що поєднання HTML, CSS, JavaScript і Firebase дозволяє реалізувати систему, яка працює стабільно і при цьому не потребує складної інфраструктури. Це особливо важливо, коли потрібно швидко отримати результат і не витрачати ресурси на додаткові налаштування.

3.3 Проєктування бази даних

На початковому етапі було визначено основні сутності предметної області, які відображають процеси інтернет-магазину. До них належать користувачі, товари, замовлення, позиції замовлення та статуси замовлень. Виділення цих сутностей дозволяє структуровано представити інформацію та забезпечити її логічну організацію.

Сутність «**Користувач**» призначена для зберігання даних про осіб, які взаємодіють із системою. Вона містить такі атрибути, як унікальний ідентифікатор, ім’я, електронна пошта, пароль та роль. Роль визначає рівень доступу до функціоналу системи та забезпечує розмежування прав користувачів.

Сутність «**Товар**» включає інформацію про продукцію, що пропонується в інтернет-магазині. До її атрибутів належать назва, ціна та кількість на складі.

Сутність «**Замовлення**» містить інформацію про оформлені покупки. Вона включає дані про користувача, дату створення, загальну суму та поточний статус. Для відображення складу замовлення використовується додаткова сутність «Позиція замовлення», яка реалізує зв’язок між замовленням і товарами. Вона містить інформацію про товар, його кількість та вартість у межах замовлення.

Особливу увагу приділено встановленню зв’язків між сутностями. Зокрема, між користувачем і замовленням реалізовано зв’язок типу «один до багатьох», оскільки один користувач може мати декілька замовлень. Між замовленням і товарами використано зв’язок «багато до багатьох», який реалізується через сутність «**Позиція замовлення**». Це дозволяє зберігати інформацію про декілька товарів у межах одного замовлення.

Для забезпечення цілісності даних використано первинні та зовнішні ключі. Первинні ключі забезпечують унікальність записів у таблицях, а зовнішні ключі встановлюють зв’язки між ними. Це дозволяє уникнути появи некоректних або «висячих» даних у системі.

У процесі проєктування також було враховано принципи нормалізації. Структура бази даних приведена до третьої нормальної форми, що дозволяє усунути надлишковість даних та мінімізувати ризик виникнення аномалій при їх оновленні. У результаті забезпечується ефективне зберігання інформації та підвищується надійність роботи системи.

Окрім цього, було передбачено можливість розширення бази даних. Зокрема, структура дозволяє додавати нові сутності або атрибути без значних змін у вже існуючих таблицях. Це є важливим у випадку розвитку інтернет-магазину та розширення його функціональних можливостей.

Отже, було сформовано структуру бази даних на основі Firebase [9, 19], яка забезпечує ефективне зберігання та обробку інформації про замовлення, товари та користувачів. Це створює надійну основу для реалізації веб-інформаційної системи управління замовленнями та забезпечує її стабільне функціонування.

3.4 Визначення та логіка взіємодії ролей користувачів

У системі передбачено дві ролі: адміністратор та комплектувальник. Кожна з них виконує чітко визначені функції та має відповідний рівень доступу до даних.[13]

Адміністратор є ключовим користувачем системи, який відповідає за створення та управління замовленнями. До його основних функцій належать додавання нових замовлень, внесення інформації про склад замовлення, редагування даних та контроль загального стану системи. Адміністратор також має можливість переглядати всі замовлення, перегляд логів та аналітики

Комплектувальник у свою чергу виконує функції обробки замовлень. Йому доступний перегляд списку замовлень, які потребують виконання для його складу. Також комплектувальник наприклад зі складу “Дніпро” не зможе виконати замовлення зі складу “Львів”. Основним завданням є комплектація замовлення відповідно до наданої інформації та оновлення його статусу після завершення роботи. У результаті комплектувальник забезпечує фізичне виконання замовлення в межах системи.

Такий розподіл ролей дозволяє чітко розмежувати функції створення та виконання замовлень, що спрощує організацію процесу та зменшує ймовірність помилок.

Логіка взаємодії ролей є надважливим аспектом, оскільки дозволяє визначити, яким чином різні категорії користувачів взаємодіють із системою та між собою. Чіткий розподіл ролей забезпечує ефективну організацію роботи, зменшує ймовірність помилок та підвищує рівень безпеки даних.

У процесі проєктування було визначено основні ролі користувачів (адміністратор та комплектувальники), сформовано сценарії їх взаємодії з системою та встановлено обмеження доступу до функціоналу (наприклад комплектувальник не може видалити щось зі складу або видалити замовлення).

Такий підхід дозволяє забезпечити логічну узгодженість роботи системи та її стабільне функціонування.

3.5 Підключення Firebase

Для початку потрібно створити проект у Firebase, після цього налаштувати базові параметри системи та отримати конфігураційні дані (**API key, projectId , authDomain**, тощо.) [12]

Далі потрібно додати Веб-Додаток у розділі **Project settings**. Після цього Firebase згенерував конфігураційний об’єкт, який містить ключі для підключення проекту.

Для того, щоб встановити базу у наш застосунок, у наш локальний JS застосунок пишемо: **npm install firebase**, це дозволить використовувати модулі Firebase безпосередньо у коді застосунку.

На всіх .html сторінках підключено наш головний JS файл (app.js), у ньому додаємо ініціалізацію Firebase:

```
const firebaseConfig = {
    apiKey: "AIzaSyDqT6wTo1NeA5eNJ3kiHGUN1nu2QQlkSuI",
    authDomain: "orderflowbase.firebaseapp.com",
    databaseURL: "https://orderflowbase-default-rtdb.firebaseio.com",
    projectId: "orderflowbase",
    storageBucket: "orderflowbase.firebasestorage.app",
    messagingSenderId: "345696988928",
    appId: "1:345696988928:web:a625151459aaa17eff0de5"
};
```

Рис. 3.1 - Підключення конфігу Firebase у JS

Після підключення було ініційовано необхідні сервіси – авторизація користувачів, та зберігання даних. Це потрібно для подальшої правильної роботи з сервісом Firebase.

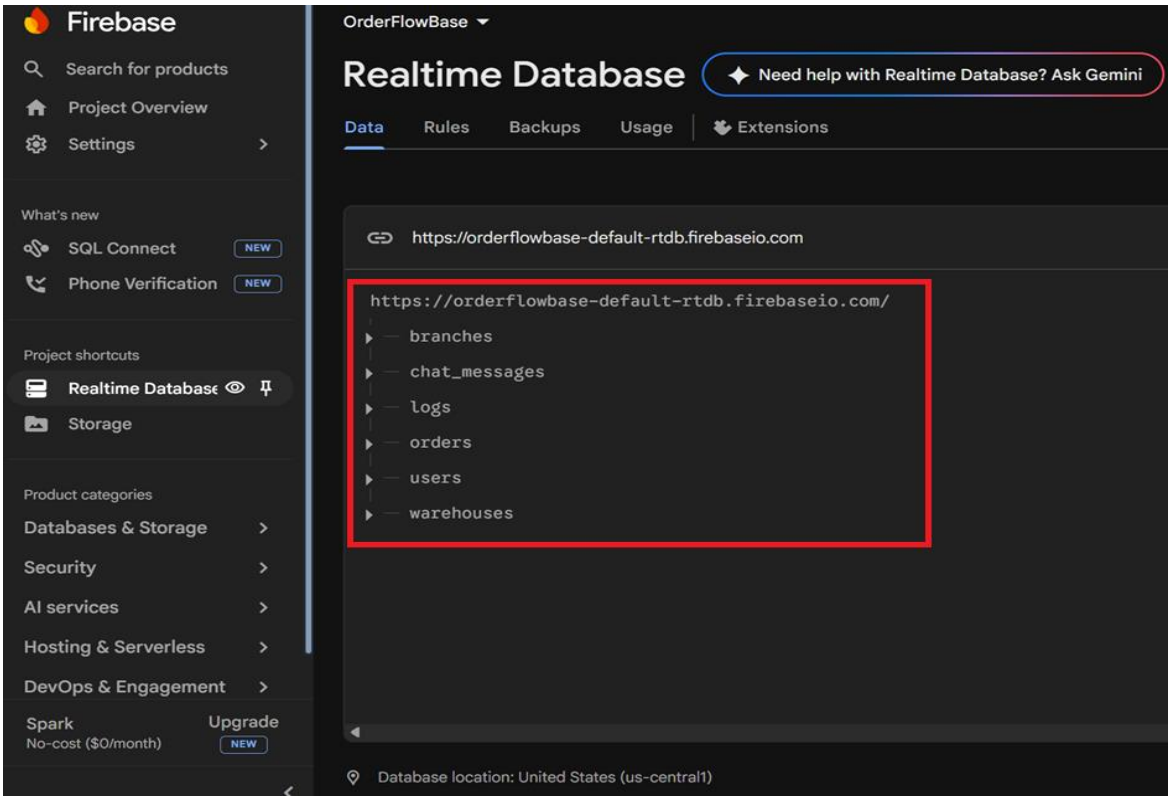


Рис 3.2 - Структура даних у базі реального часу Firebase

3.6 Розробка авторизації

У даному проєкті авторизація реалізована з використанням сервісу **Firebase Authentication**, який відповідає за перевірку облікових даних користувачів та надання доступу до системи. Вхід у систему здійснюється за допомогою електронної пошти та пароля, після чого користувач отримує сесію доступу.[24, 10]

Після успішної автентифікації додатково виконується перевірка ролі користувача, яка зберігається в базі даних Firestore у відповідному документі користувача. Саме поле ролі (наприклад, **role: admin** або **role: picker**) визначає рівень доступу в системі.

Алгоритм роботи авторизації виглядає наступним чином:

1. Користувач вводить логін та пароль у форму входу.
2. **Firebase Authentication** перевіряє коректність даних.

3. Після успішного входу система звертається до Firestore та отримує роль користувача.
4. На основі отриманої ролі виконується перенаправлення:
   - адміністратор отримує доступ до панелі управління замовленнями;
   - комплектувальник переходить до інтерфейсу виконання замовлень.
5. У разі відсутності або невідповідності ролі доступ до функціоналу блокується.

Додатково реалізовано захист маршрутів (route protection), що запобігає доступу до сторінок системи без авторизації. Навіть при спробі прямого переходу за URL система перевіряє стан входу користувача та його роль, якщо авторизація не пройшла перевірку, нас викине на початкову сторінку і зправа внизу покаже помилку.

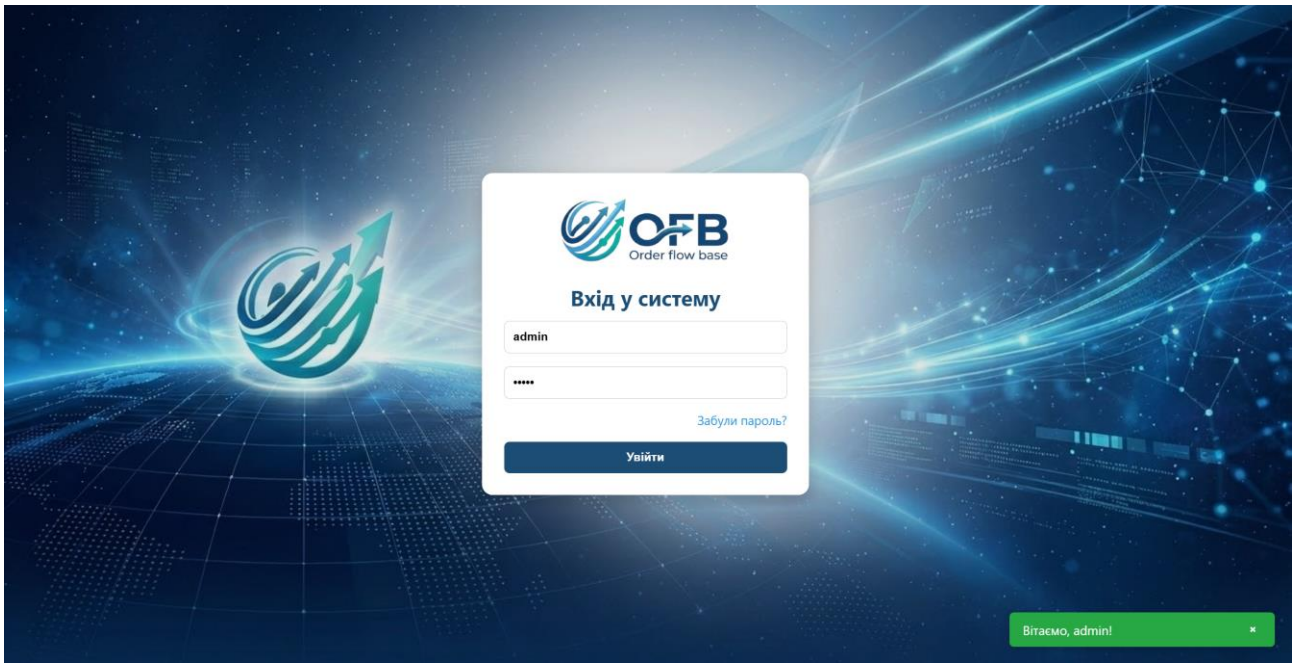
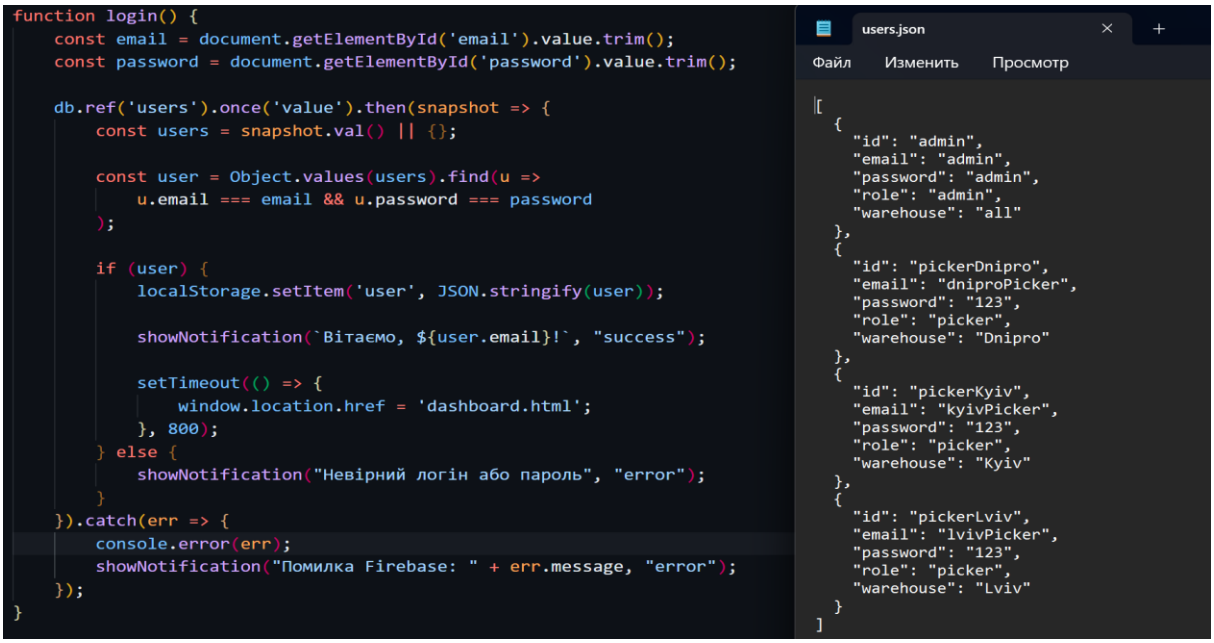


Рис. 3.3 - Форма авторизації користувача



```
function login() {
    const email = document.getElementById('email').value.trim();
    const password = document.getElementById('password').value.trim();

    db.ref('users').once('value').then(snapshot => {
        const users = snapshot.val() || {};

        const user = Object.values(users).find(u =>
            u.email === email && u.password === password
        );

        if (user) {
            localStorage.setItem('user', JSON.stringify(user));

            showNotification(`Вітаємо, ${user.email}!`, "success");

            setTimeout(() => {
                window.location.href = 'dashboard.html';
            }, 800);
        } else {
            showNotification("Невірний логін або пароль", "error");
        }
    }).catch(err => {
        console.error(err);
        showNotification("Помилка Firebase: " + err.message, "error");
    });
}
```

users.json

Файл Изменить Просмотр

```
[
  {
    "id": "admin",
    "email": "admin",
    "password": "admin",
    "role": "admin",
    "warehouse": "all"
  },
  {
    "id": "pickerDnipro",
    "email": "dniproPicker",
    "password": "123",
    "role": "picker",
    "warehouse": "Dnipro"
  },
  {
    "id": "pickerKyiv",
    "email": "kyivPicker",
    "password": "123",
    "role": "picker",
    "warehouse": "Kyiv"
  },
  {
    "id": "pickerLviv",
    "email": "lvivPicker",
    "password": "123",
    "role": "picker",
    "warehouse": "Lviv"
  }
]
```

Рис. 3.4 - Логіка авторизації

Після успішної авторизації ми потрапляємо на головну панель, де розташовані замовлення, а також інші функції [25]:

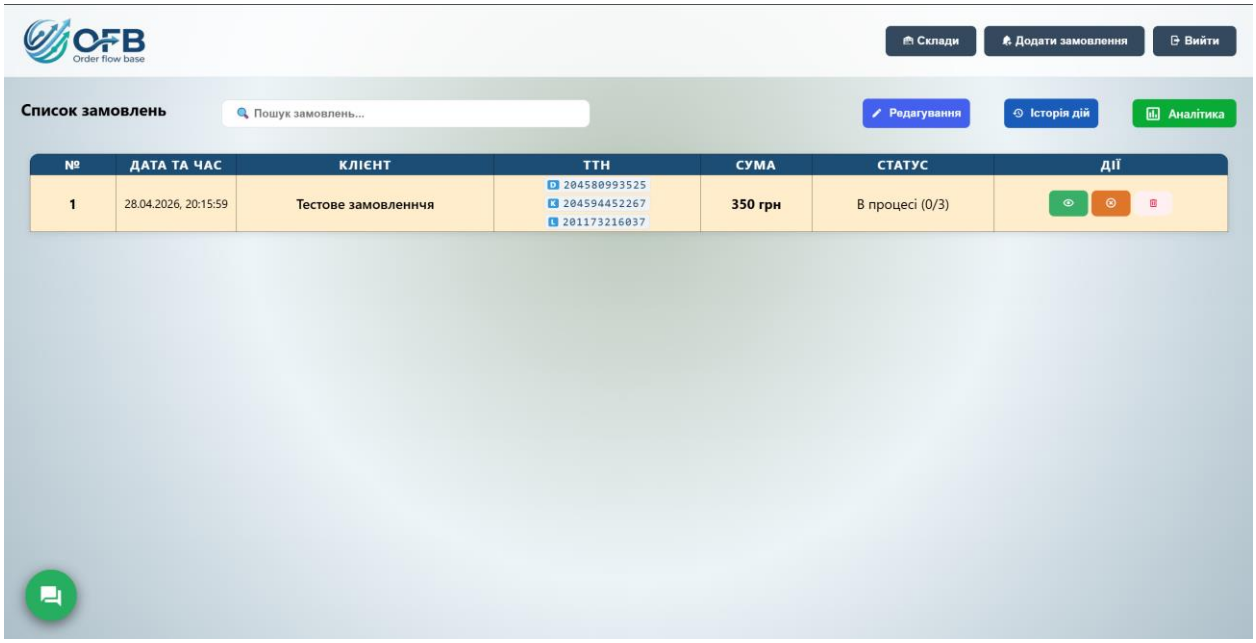


Рис. 3.5 - Головна сторінка панелі

3.7 Реалізація Multi-Warehouses (мульти-склади та логіка обліку залишків)

Мульти-склади передбачають собою створення адміністратором нових складів, наприклад з різних міст, та наповнення товарами потрібних складів

потрібним товаром, що є дуже зручним інструментом для керування товарами. Ключовою функцією є переміщення товарів на складах, з правильною логікою списання залишків у реальному часі без затримок.

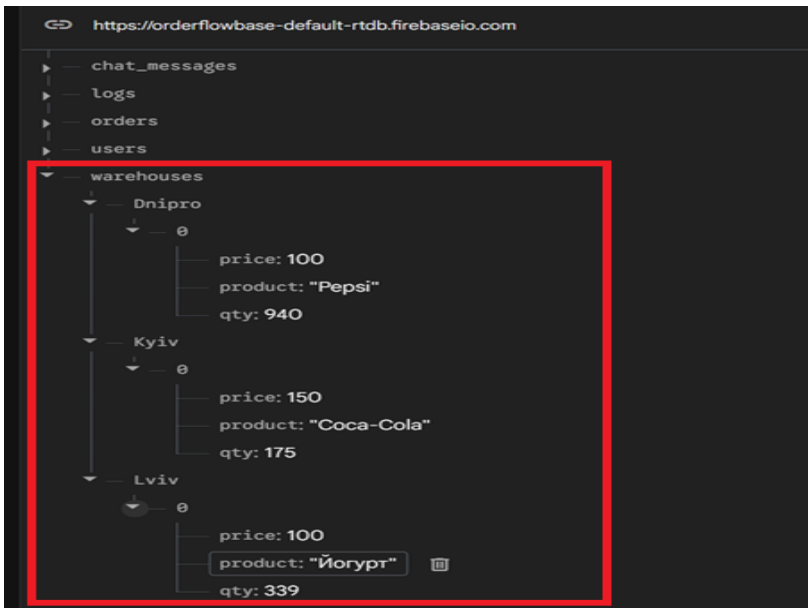


Рис. 3.6 - Структура зберігання складських залишків у Firebase Realtime Database

```
function createNewWarehouse() {
    const name = document.getElementById('newWarehouseName').value.trim();
    if (!name) return showNotification("Введіть назву складу", "error");

    // порожній запис у базі для нового складу
    db.ref('warehouses/' + name).set([{ product: "Тестовий товар (видаліть)", price: 1, qty: 1 }])
    .then(() => {
        showNotification(`Склад "${name}" створено`, "success");
        document.getElementById('newWarehouseName').value = '';
        loadWarehouse(); // рестарт інтерфейсу
    });
}
```

Рис. 3.7 - Створення нових складів

Після створення складу інтерфейс автоматично оновлюється, що забезпечує відображення нового складу у списку доступних, у вигляді **tab-buttons** зверху **popup** складу, що є зручним для користування навіть для тих, хто перший раз користується подібними системами. Надалі користувач може додавати, редагувати, видаляти або переміщати товари в межах створених складів.

```
function loadWarehouse(activeWarehouseName) {
    const user = JSON.parse(localStorage.getItem('user') || '{}');
    const role = checkRole();

    db.ref('warehouses').once('value', (snapshot) => {
        const allWarehouses = snapshot.val() || {};
        const tabsContainer = document.getElementById('warehouseTabs');
        if (!tabsContainer) return;

        const warehouseNames = Object.keys(allWarehouses);
        tabsContainer.innerHTML = '';
```

Рис. 3.8 - Отримання та динамічне відображення списку складів у системі (функція loadWarehouse)

```
function renderWarehouseContent(activeWarehouseName) {
    const role = checkRole();
    const user = JSON.parse(localStorage.getItem('user') || '{}');

    const warehouseTitle = document.getElementById('currentWarehouseName');
    if (warehouseTitle) warehouseTitle.textContent = activeWarehouseName;

    // --- ЛОГІКА ТОВАРІВ НА СКЛАДІ ---
    db.ref(`warehouses/${activeWarehouseName}`).on('value', (snapshot) => {
        const data = snapshot.val();
        const tbody = document.querySelector('#warehouse-table tbody');
        if (!tbody) return;
        tbody.innerHTML = '';

        let items = Array.isArray(data) ? data : (data ? Object.values(data) : []);
        const filteredItems = items.filter(item => item && item.product && item.product !== "PL

        if (filteredItems.length === 0) {
            tbody.innerHTML = '<tr><td colspan="4" style="text-align:center;">Склад порожній</t
        } else {
            filteredItems.forEach((item, index) => {
                const tr = document.createElement('tr');
                const actionButtons = role === 'admin'
                    ? `<button class="admin-btn delete" onclick="deleteProduct(${index})"><i cl
                    : '-';
                tr.innerHTML = `
                    <td>${item.product}</td>
```

Рис. 3.9 - Динамічне відображення товарів обраного складу

```
function addProduct() {
    const name = document.getElementById('newProduct').value.trim();
    const price = parseFloat(document.getElementById('newPrice').value);
    const qty = parseInt(document.getElementById('newQty').value);

    if (!name || isNaN(price) || isNaN(qty)) {
        return showNotification("Заповніть всі поля!", "error");
    }

    db.ref(`warehouses/${currentActiveWarehouse}`).once('value').then(snap => {
        let items = snap.val() || [];
        items.push({ product: name, price: price, qty: qty });
        return db.ref(`warehouses/${currentActiveWarehouse}`).set(items);
    }).then(() => {
        showNotification(`Товар додано на склад ${currentActiveWarehouse}`, "success");
        document.getElementById('newProduct').value = '';
        document.getElementById('newPrice').value = '';
        document.getElementById('newQty').value = '';
    }).catch(err => showNotification(err.message, "error"));
}
```

Рис. 3.10 - Реалізація додавання товару до складу

Також у системі важливо нормально продумати момент зі списанням товарів. На перший погляд це дрібниця, але саме тут часто виникають проблеми зі складським обліком. Наприклад, адміністратор скасовує замовлення — клієнт передумав, не оплатив покупку або просто помилився під час оформлення. У такій ситуації товари не повинні залишатися “списаними” із системи. Їх потрібно автоматично повернути назад на склад, а саме замовлення позначити як «Скасоване».

Без цього починається плутанина із залишками. Товар фактично є на полиці, але в системі його вже немає. Через це інші працівники можуть бачити неправильну кількість продукції або взагалі думати, що позиція закінчилась. У результаті доводиться вручну перевіряти склад і виправляти дані, а це займає час і створює зайву роботу.

Особливо помітно це стає тоді, коли замовлень багато. Якщо кілька разів забути повернути товари після скасування, залишки поступово починають “роз’їжджатися” з реальною кількістю на складі. Через це можуть виникати дивні ситуації: система показує нуль товару, хоча він насправді лежить у коробках, або навпаки — товар уже проданий, але в обліку ще рахується доступним.

Ще один момент — робота з кількома складами. Якщо товар списувався з конкретного складу, то й повертатися він має туди ж. Інакше інформація стає неточною, а працівники починають витрачати час на пошук товарів або перевірку залишків вручну.

Також важливо, щоб після скасування змінювався статус самого замовлення. Це потрібно хоча б для того, щоб працівники одразу бачили, що воно більше не обробляється. Плюс така інформація може знадобитися пізніше — наприклад, для перегляду історії замовлень або аналізу причин скасування.

По суті, нормальна логіка повернення товарів потрібна не тільки для зручності, а й для того, щоб складський облік взагалі залишався актуальним.

Інакше навіть невеликі помилки з часом накопичуються й починають впливати вже на всю роботу магазину.

```
// 2. групуємо повернення по складах
const promises = itemsToReturn.map(item => {
    const whName = item.warehouse || order.warehouse || "Dnipro"; // Дефолтний склад

    return db.ref(`warehouses/${whName}`).once('value').then(whSnap => {
        let whData = whSnap.val() || [];
        let inventory = Array.isArray(whData) ? whData : Object.values(whData);


        const productOnWh = inventory.find(p =>
            p.product && p.product.trim().toLowerCase() === item.product.trim().toLowerCase()
        );

        if (productOnWh) {
            productOnWh.qty = (Number(productOnWh.qty) || 0) + (Number(item.qty) || 0);
        } else if (item.product !== "PLACEHOLDER_EMPTY") {
            // якщо товару немає, додаємо його (якщо це не пустий маркер)
            inventory.push({
                product: item.product,
                price: item.price || 0,
                qty: Number(item.qty)
            });
        }
```

Рис. 3.11 - Повернення товарів на склад при скасуванні замовлення

Так виглядає popup-вікно складу. У ньому реалізовано основні функції для роботи із залишками товарів та керування складом. Зокрема, передбачено створення нових складів, додавання товарів, оновлення інформації про залишки, а також подальшу роботу з ними без перезавантаження сторінки.

Окрему увагу було приділено саме логіці обліку товарів, оскільки від правильності цих даних напряму залежить коректна робота всієї системи. Наприклад, під час оформлення або скасування замовлення кількість товару автоматично змінюється у відповідному складі. Це дозволяє уникнути ситуацій, коли інформація про залишки не відповідає реальному стану товарів.

Уся взаємодія працює динамічно [16]. Дані оновлюються практично в реальному часі завдяки використанню Firebase та реалізації клієнтської логіки на JavaScript. За рахунок цього зміни одразу відображаються в інтерфейсі без

необхідності повторного завантаження сторінки, що робить роботу із системою швидшою та зручнішою для користувача.

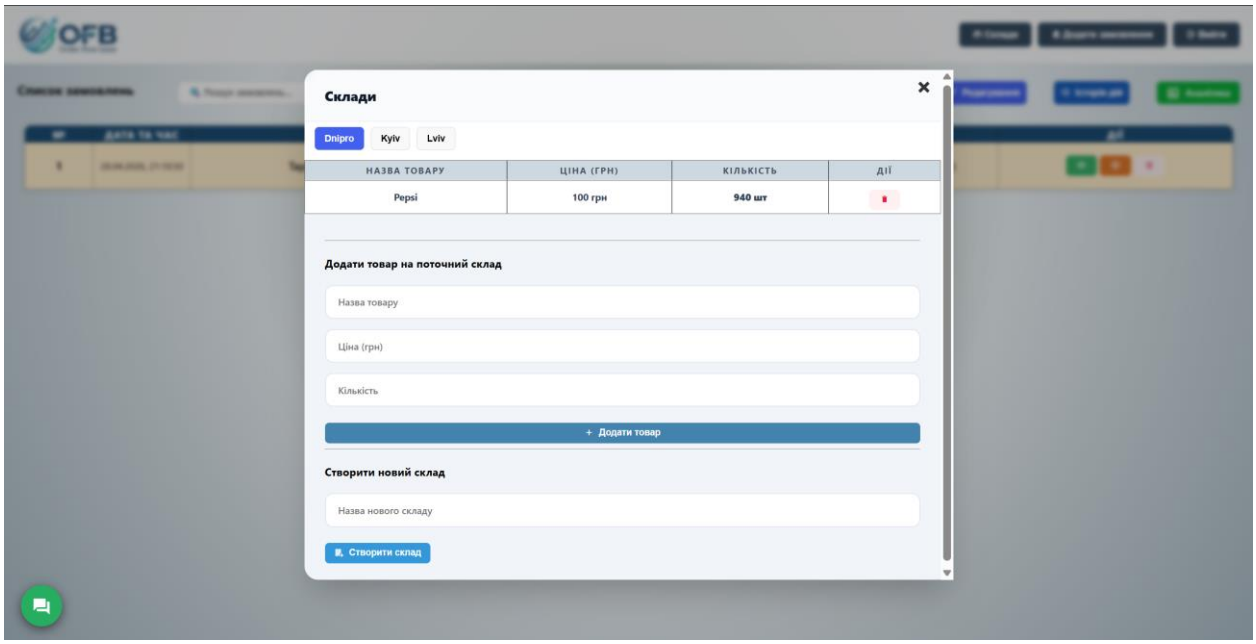


Рис. 3.12 - Вигляд popup(спливаюче вікно) складу

Переміщення товарів на складах

Переміщення товарів між складами або всередині одного складу — це одна з тих речей, яка на практиці здається простою, але сильно впливає на порядок у системі. По суті, це звичайна операція зміни “місця” товару, але з обов’язковим оновленням усіх даних про залишки.

У реальній роботі це виглядає так: є певний товар, який спочатку знаходиться на одному складі або в одній його частині, і виникає потреба перемістити його в інший склад або іншу категорію зберігання. Причини можуть бути різні — наприклад, перерозподіл запасів між складами, оптимізація логістики або підготовка товару під конкретні замовлення. І тут важливо, щоб система не просто “перенесла” товар, а правильно оновила всі пов’язані дані.

При переміщенні обов’язково змінюється прив’язка до складу. Тобто кількість товару зменшується в одному місці й одночасно додається в іншому. Це здається очевидним, але якщо цей процес не автоматизований, легко отримати розбіжності в обліку. Наприклад, товар фізично вже лежить на новому складі, а в

системі все ще рахується на старому. Або навпаки — він списаний з обох складів через помилку в ручному оновленні.

У добре продуманій системі переміщення відбувається як єдина операція. Тобто користувач просто вказує, який товар і куди потрібно перенести, а далі система сама оновлює всі залишки. При цьому бажано, щоб зберігалася історія таких дій: коли саме було переміщення, хто його виконав і в якій кількості. Це корисно не тільки для контролю, але й для розуміння руху товарів у межах складу.

Окремий момент — це часткові переміщення. Буває, що переноситься не весь залишок, а тільки його частина. Наприклад, із 50 одиниць товару переносять лише 20. У такому випадку система повинна коректно розділити кількість: на першому складі залишається 30, а на другому з’являється 20. Тут важлива точність, бо будь-яка неточність одразу впливає на подальші замовлення.

Ще одна деталь — це синхронізація з іншими процесами. Якщо товар бере участь у активних замовленнях або вже зарезервований, система має враховувати це перед переміщенням. Інакше може виникнути ситуація, коли товар формально “перенесли”, але він уже був закріплений за іншим процесом.

Правильно реалізоване переміщення товарів дозволяє підтримувати порядок на складі без зайвих ручних перевірок. Дані залишаються актуальними, працівникам не потрібно постійно звіряти залишки вручну, а система краще відображає реальну картину руху товарів.

```
async function executeTransfer() {
    const productSelect = document.getElementById('transferProductSelect');
    const targetWhSelect = document.getElementById('targetWarehouseSelect');
    const qtyInput = document.getElementById('transferQty');
    const reasonInput = document.getElementById('transferReason');

    if (!productSelect || !targetWhSelect || !qtyInput || !reasonInput) {
        return showNotification("Помилка: Елементи інтерфейсу не знайдені", "error");
    }

    const productName = productSelect.value;
    const targetWhName = targetWhSelect.value;
    const qtyToMove = parseInt(qtyInput.value);
    const reason = reasonInput.value.trim() || "Не вказано";
    const sourceWhName = currentActiveWarehouse;

    const user = JSON.parse(localStorage.getItem('user') || '{}');

    const userName = user.name || user.displayName || user.username || user.email || "Співробітник";

    if (!productName || !targetWhName || isNaN(qtyToMove) || qtyToMove <= 0) {
        return showNotification("Заповніть усі поля коректно", "error");
    }

    try {
        const snap = await db.ref('warehouses').once('value');
        const allWh = snap.val() || {};

        let sourceItems = allWh[sourceWhName] || [];
        let targetItems = allWh[targetWhName] || [];
```

Рис. 3.13 – Логіка переміщення товарів на склад(1)

```
        //шукаємо товар на складі-відправнику
        const itemInSource = sourceItems.find(i => i.product === productName);

        if (!itemInSource || itemInSource.qty < qtyToMove) {
            return showNotification(`Недостатньо товару (є ${itemInSource ? itemInSource.qty : 0})`, "error");
        }

        // списуємо у відправника
        itemInSource.qty -= qtyToMove;

        // додаємо отримувачу
        const itemInTarget = targetItems.find(i => i.product === productName);
        if (itemInTarget) {
            itemInTarget.qty += qtyToMove;
        } else {
            targetItems.push({
                product: productName,
                price: itemInSource.price || 0,
                qty: qtyToMove
            });
        }

        // чистимо масиви від порожніх записів
        allWh[sourceWhName] = sourceItems.filter(i => i.qty > 0 || i.product === "PLACEHOLDER_EMPTY");
        allWh[targetWhName] = targetItems.filter(i => i.product !== "PLACEHOLDER_EMPTY");


        await db.ref('warehouses').set(allWh);

        const logEntry = {
            user: userName,
            action: "Переміщення",
```

Рис. 3.14 – Логіка переміщення товарів на склад(2)

```
        const logEntry = {
            user: userName,
            action: "Переміщення",
            details: `Товар: ${productName} (${qtyToMove} шт.) | ${sourceWhName} → ${targetWhName} | Причина: ${reason}`,
            date: new Date().toLocaleString()
        };

        await db.ref('logs').push(logEntry);


        showNotification("Переміщення виконано успішно", "success");


        reasonInput.value = '';
        qtyInput.value = '1';


        loadWarehouse(sourceWhName);

    } catch (error) {
        console.error("Помилка при переміщенні:", error);
        showNotification("Помилка бази даних", "error");
    }
}
```

Рис. 3.15 – Логіка переміщення товарів на склад(3)

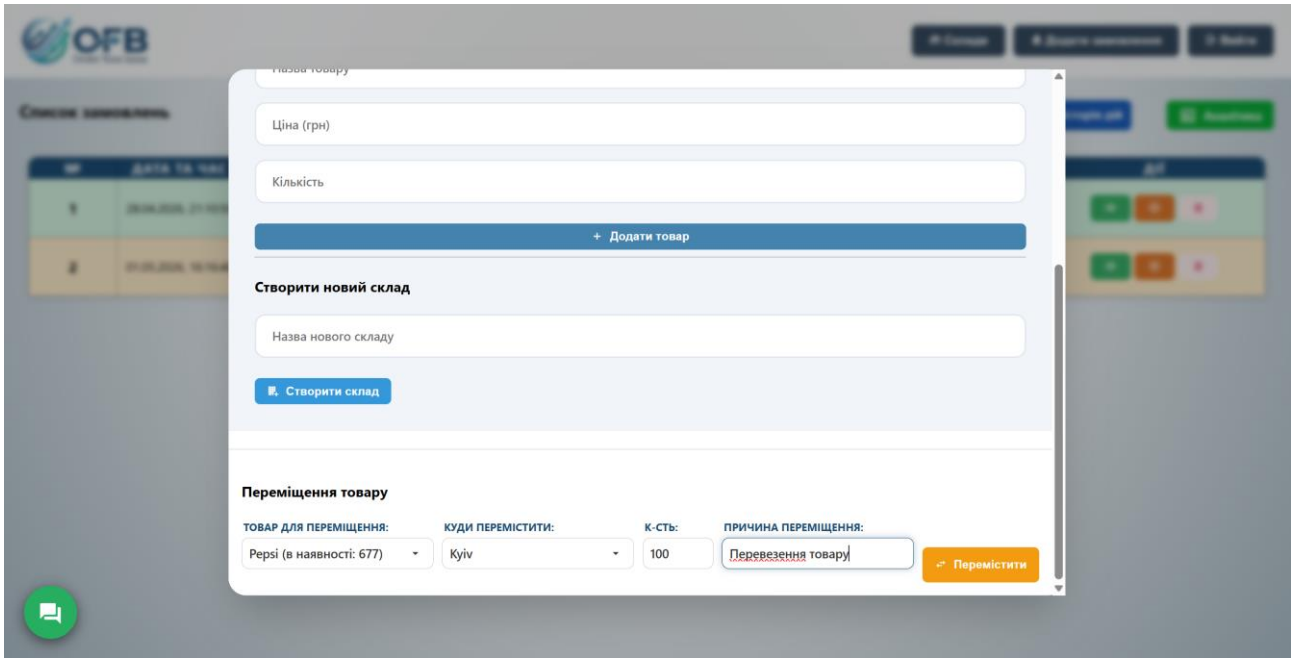


Рис. 3.16 – Вигляд переміщення товарів на склад

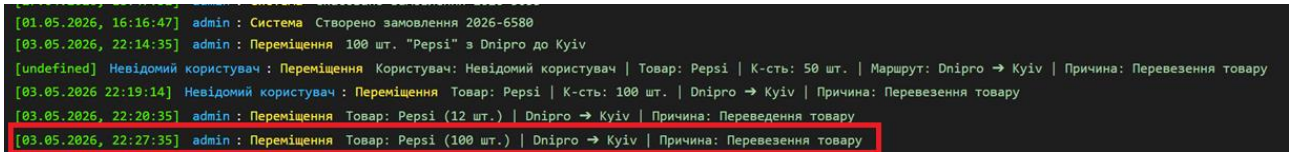


Рис. 3.17 – Логування цього переміщення

3.8 Створення та обробки замовлень

3.8.1 Створення замовлень

Процес створення замовлення фактично є основною точкою входу в систему, бо саме з нього все й починається. Користувач обирає товари, формує кошик і підтверджує покупку — після цього система має швидко і без затримок перетворити ці дії у нормальний запис у базі даних.

У реалізації цей процес побудований на асинхронній взаємодії клієнтської частини з **Firebase**, зокрема з його **Realtime Database**. Це означає, що дані не “завмирають” на одному етапі, а одразу передаються в базу й синхронізуються між усіма частинами системи. Користувач натиснув кнопку — і замовлення вже створене в системі без потреби в перезавантаженні сторінки.

Після відправки даних формується структурований запис, у якому зберігається вся необхідна інформація: список товарів, їх кількість, дані клієнта, час створення та початковий статус замовлення. Зазвичай це статус на кшталт “нове”, який означає, що замовлення ще не потрапило в обробку.

Важливий момент тут — це перевірка даних перед записом. Система має переконатися, що всі обов’язкові поля заповнені, товари дійсно існують у базі, а кількість не перевищує доступні залишки. Інакше можуть виникнути ситуації, коли в систему потрапляють некоректні замовлення, що потім ускладнює роботу складу.

Також варто врахувати, що створення замовлення одразу запускає подальші процеси — наприклад, резервування товарів або оновлення складських залишків. Тобто це не ізольована дія, а частина загального ланцюга обробки. Саме тому важливо, щоб цей етап працював стабільно і без затримок, інакше це одразу впливає на всі наступні операції в системі.

Логіка роботи модуля:

1. **Збір даних**: Система зчитує введені дані (ПІБ та номер клієнта, вибрані товари, склад, коментар для комплектувальника, відділення пошти - УкрПошта або НоваПошта).
2. **Формування об'єкта**: Створюється JSON-об'єкт, що містить мітку часу (timestamp), статус замовлення (за замовчуванням «Нове») та унікальний ідентифікатор.
3. **Запис у БД**: Використовується метод **push()** для генерації унікального ключа замовлення в гілці **/orders.**

Візуалізація процесу:

З точки зору інтерфейсу, цей процес супроводжується динамічним оновленням таблиці замовлень. Оскільки Firebase працює за протоколом **WebSockets**, адміністратор бачить нове замовлення миттєво, без оновлення сторінки.

Особливості реалізації:

1. **Автоматизація**: При створенні замовлення система автоматично підтягує дані про доступність товару на вибраному складі.
2. **Цілісність**: Перед відправкою запиту проводиться валідація полів на стороні клієнта (перевірка заповнення обов’язкових полів).

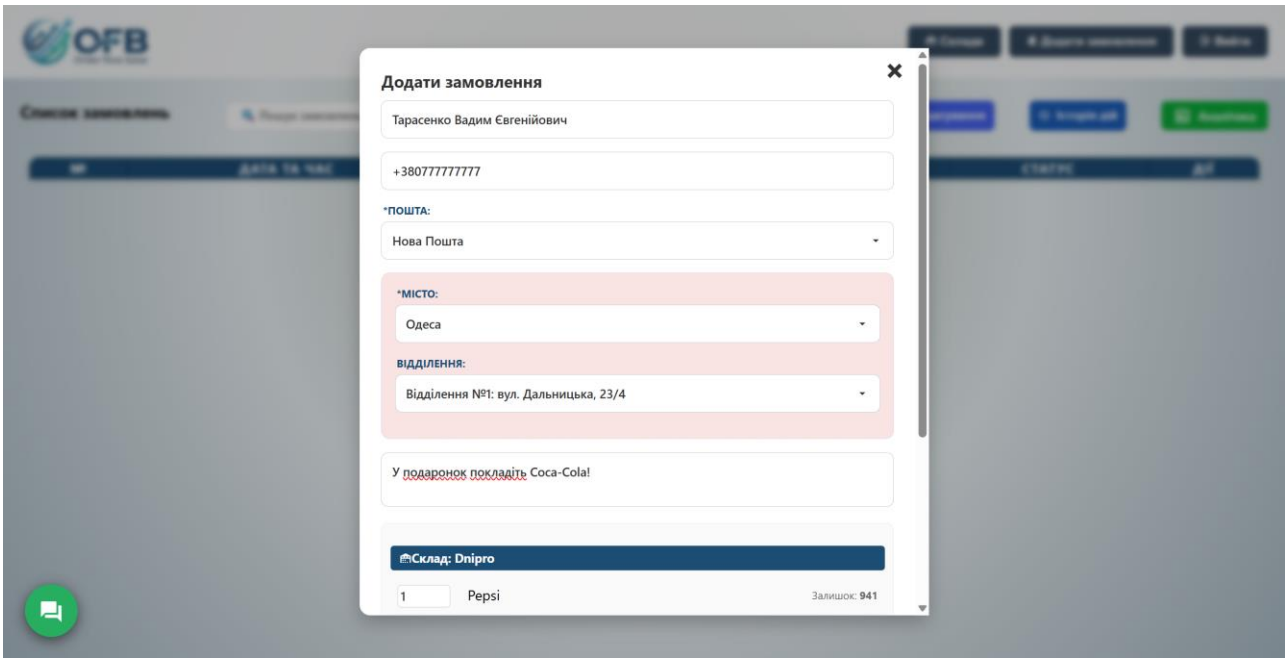


Рис. 3.18 - Додавання замовлення адміністратором (1 частина)

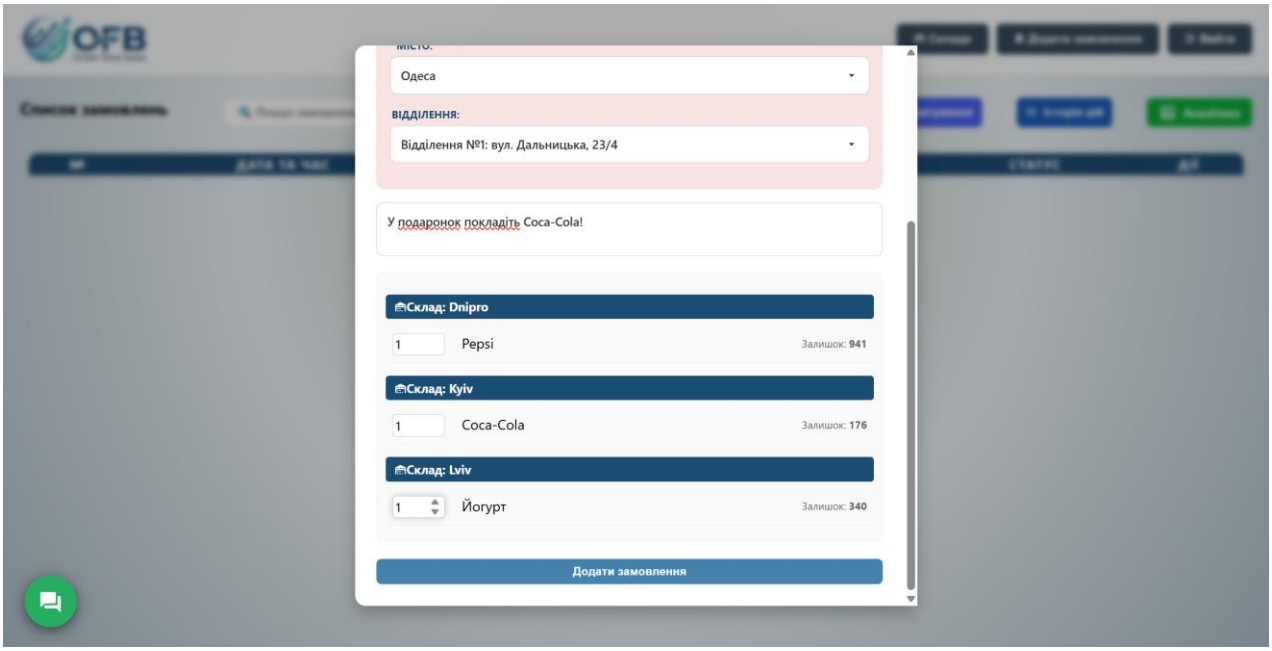


Рис. 3.19 - Додавання замовлення адміністратором (2 частина)

У результаті на головній панелі бачимо тільки що створене замовлення, яке отримало статус нове (0/3). Воно має зібратися комплектувальниками з трьох різних складів з різних міст Дніпро, Київ, Львів (3 посилки).

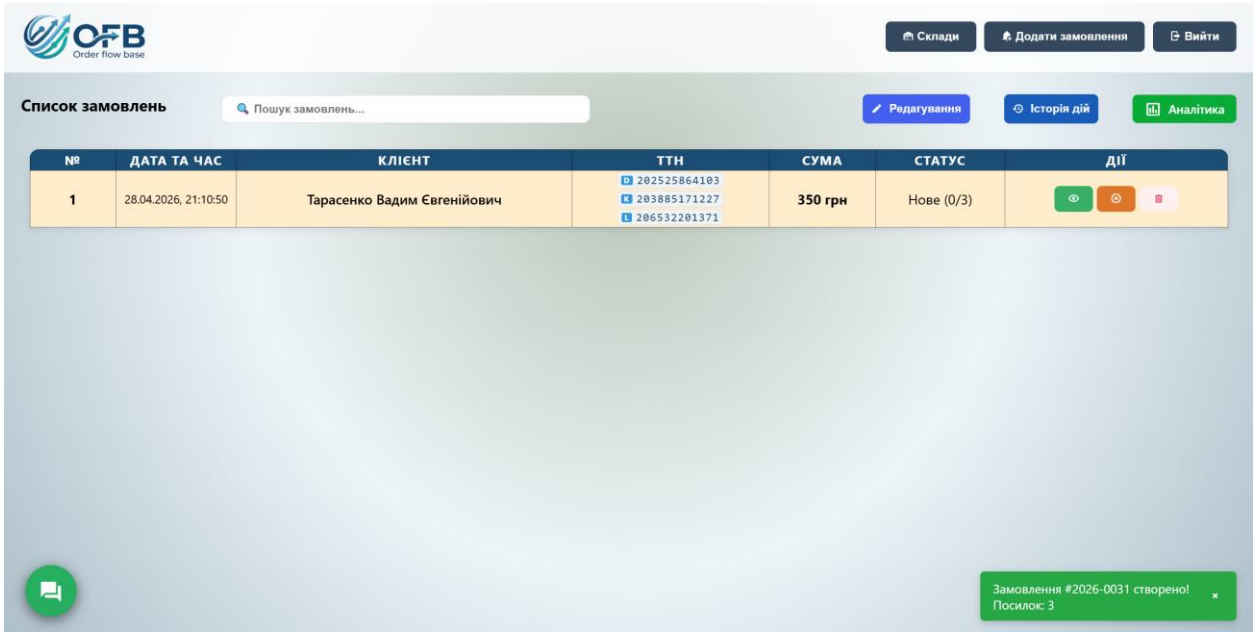


Рис. 3.20 - Нове замовлення створене Адміном

3.8.2 Обробка замовлення

Кожен комплектувальник у системі бачить нові замовлення практично одразу після того, як їх створює адміністратор. Оновлення відбувається в реальному часі, тому немає потреби вручну перезавантажувати сторінку чи постійно перевіряти список замовлень. Це особливо зручно, коли одночасно працює кілька складів і замовлення надходять регулярно протягом дня.

При цьому доступ комплектувальника обмежується лише тим складом, до якого він прив’язаний. Наприклад, працівник складу в Києві не зможе працювати із замовленнями львівського складу: змінювати їх статус, закривати або друкувати маркування для відправки. Такий підхід допомагає уникнути плутанини між складами та випадкових помилок під час обробки замовлень. Особливо це помітно, коли один і той самий товар може бути на кількох складах одночасно.

Після того як комплектувальник бере замовлення в роботу, система автоматично фіксує це. Біля інформації про клієнта одразу відображається email або логін працівника, який зараз займається цим замовленням. Завдяки цьому

інші користувачі бачать, що замовлення вже обробляється, і не беруть його повторно. Фактично це працює як проста внутрішня координація між працівниками без окремих повідомлень чи дзвінків. [15]

Додатково така логіка спрощує контроль роботи всередині системи. Адміністратор у будь-який момент може подивитися, хто саме працював із конкретним замовленням, на якому етапі воно знаходиться та чи не виникло затримок під час комплектації або відправлення.

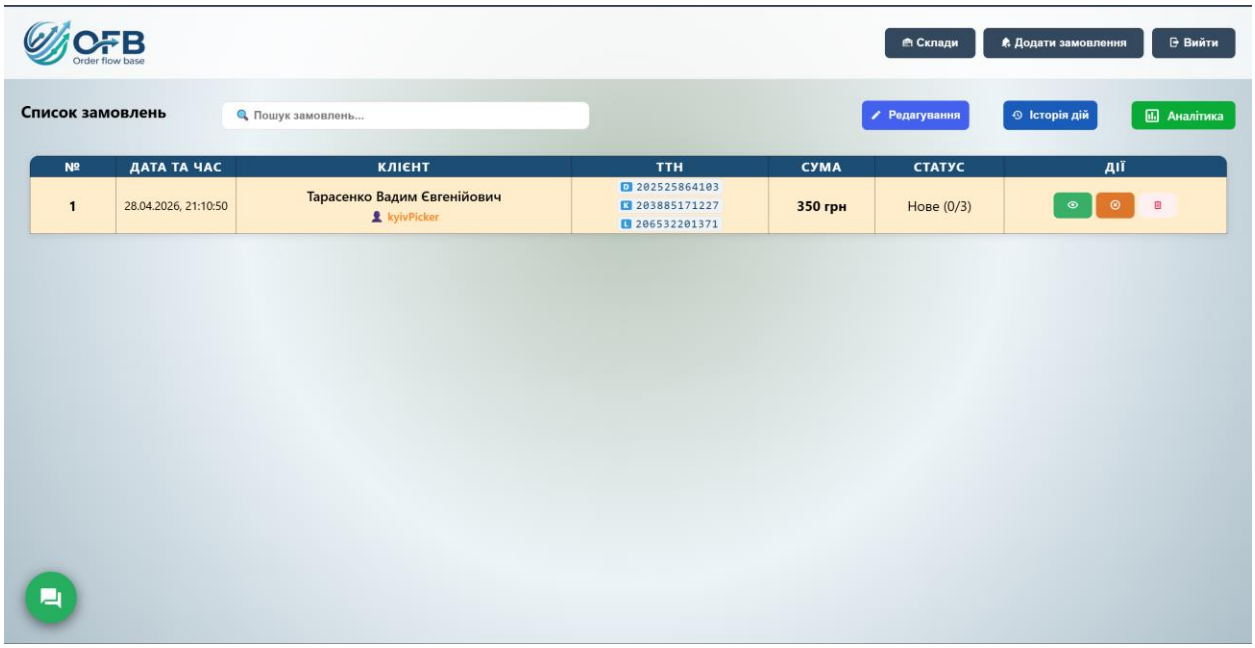


Рис. 3.21 – Вигляд dashboard з аккаунту Адміна, коли комплектувальник з Києва взяв замовлення в роботу

Подивимось на прикладі збиральника зі складу Київ (аккаунт kyivPicker), він бачить тільки замовлення та ТТН зі свого складу, тому і зможе зібрати тільки його, товари спишуться з відповідного складу, а після друкування маркування Пошти – замовлення перейде у статус “В процесі 1\3”, тобто один з трьох складів закрили замовлення зі своєї сторони.

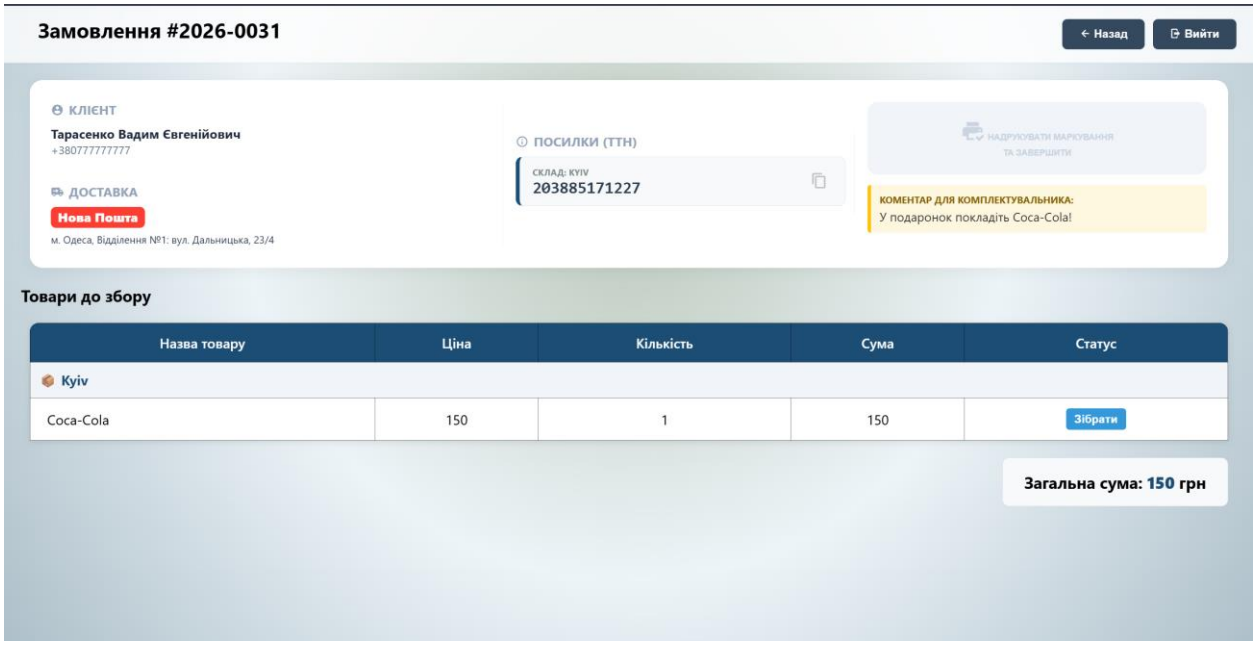


Рис. 3.22 - Замовлення з аккаунта kyivPicker

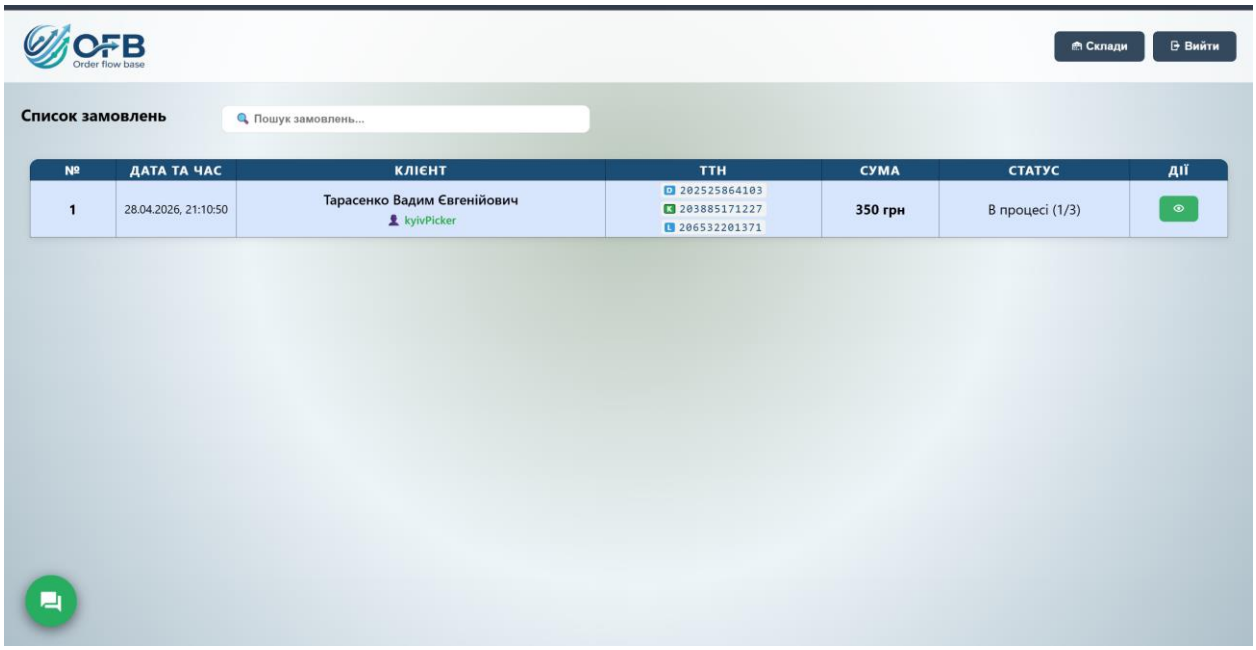


Рис. 3.23 - Комплектувальник зі складу Kyiv завершив замовлення

Як тільки усі склади успішно виконають складання замовлень зі своїх сторін, замовлення перейде у статус “Закритий (3/3)” та буде помічатися зеленим кольором, що візуально допомагає не плутатися у списку.

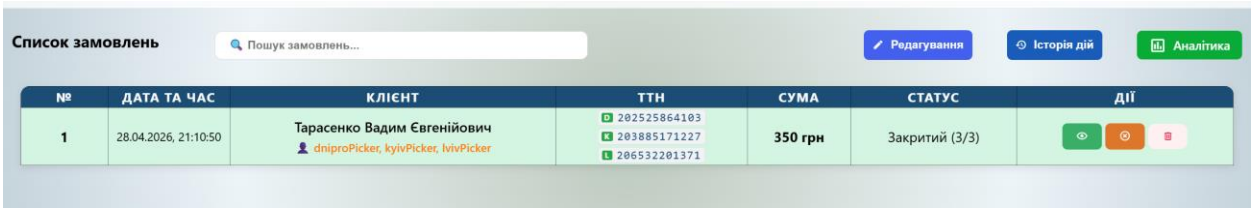


Рис. 3.24 - Повністю завершене замовлення

3.9 Реалізація аналітичного модуля системи

У межах розробленої інформаційної системи реалізовано аналітичний модуль, який забезпечує збір, обробку та відображення статистичних даних щодо роботи із замовленнями. Даний модуль дозволяє оцінювати ефективність роботи системи та приймати обґрунтовані управлінські рішення.

Призначення аналітики

Основною метою аналітичного модуля є:

- контроль кількості оброблених замовлень;
- оцінка продуктивності комплектувальників;
- аналіз швидкості виконання замовлень;
- відстеження фінансових показників.

Аналітика формується на основі даних, що зберігаються у базі Firebase, а саме: інформація про замовлення, їх статуси та час обробки.

Основні показники системи

У систему інтегровано обчислення таких ключових метрик:

1. **Загальна кількість замовлень** — відображає загальний обсяг оброблених заявок;

2. **Кількість замовлень за поточний день** — дозволяє оцінити поточне навантаження;
3. **Середній час виконання замовлення** — розраховується як різниця між моментом створення та завершення;
4. **Загальний дохід** — формується на основі вартості товарів у замовленнях.
5. **Навантаження складів** — малюється діаграма, де показано, які склади завантаженні у порівнянні.
6. **Рейтинг комплектувальників** — хто з комплектувальників зібрав більше замовлень, за що компанія може виписати премію.

Обчислення цих показників здійснюється шляхом ітерації по масиву замовлень та агрегування відповідних значень.

```
const stats = {
    totalOrders: 0,
    todayOrders: 0,
    totalRevenue: 0,
    totalTime: 0,
    completedOrders: 0
};
const today = new Date().toDateString();
ordersArray.forEach(order => {
    stats.totalOrders++;
    const orderDate = new Date(order.createdAt).toDateString();
    if (orderDate === today) {
        stats.todayOrders++;
```

```
    }

    stats.totalRevenue += Number(order.totalPrice) || 0;

    if (order.isFinished && order.createdAt && order.finishedAt) {
        const time = new Date(order.finishedAt) - new Date(order.createdAt);
        stats.totalTime += time;
        stats.completedOrders++;
    }
});
const avgTime = stats.completedOrders
    ? Math.round(stats.totalTime / stats.completedOrders / 1000 / 60)
    : 0;
```

Логіка розрахунку

Аналітичні дані формуються динамічно при завантаженні або оновленні даних. Для кожного замовлення:

- перевіряється його статус;
- визначається дата створення;
- у разі завершення обчислюється тривалість виконання;
- значення додаються до загальних підсумків.

Такий підхід дозволяє отримувати актуальну інформацію в режимі реального часу.

Візуалізація даних

Результати аналітики відображаються на окремій сторінці (**analytics.html**), до якої доступ має тільки адмін, показано це у вигляді інформаційних блоків, які містять ключові показники. Інтерфейс побудований таким чином, щоб користувач міг швидко оцінити стан системи без необхідності глибокого аналізу даних, що є зручним інструментом.

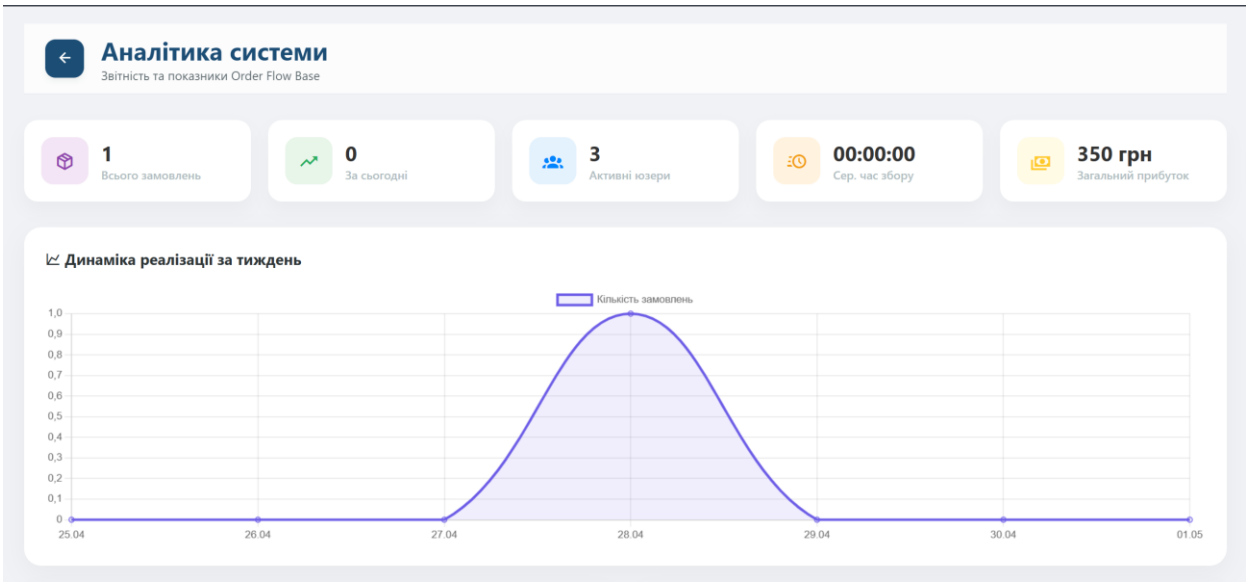


Рис. 3.25 – Графік динаміки реалізації за тиждень

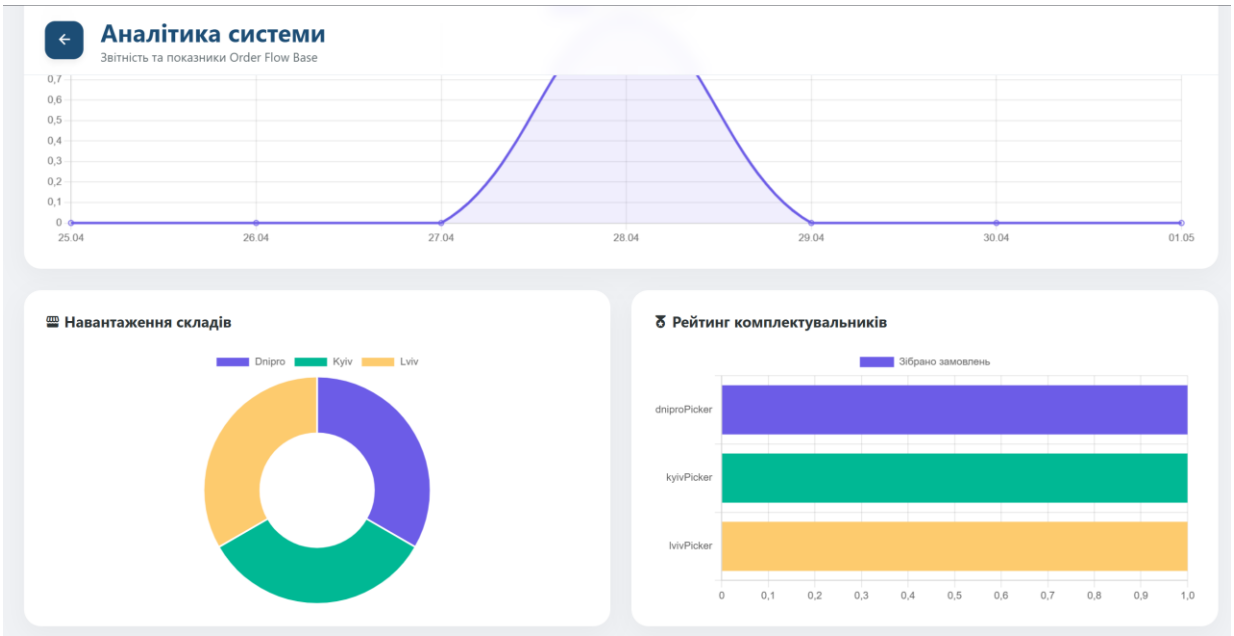


Рис. 3.26 – Навантаження складів та рейтинг всіх комплектувальників

Практичне значення

Інтегрований аналітичний модуль дозволяє:

- оперативно відстежувати навантаження на систему;
- оцінювати ефективність роботи персоналу;
- виявляти вузькі місця у процесі обробки замовлень;
- приймати рішення щодо оптимізації роботи складу.

Аналітичний модуль є важливим для підприємств різного роду, він забезпечує прозорість процесів та підвищує ефективність управління. Його використання дозволяє перетворити накопичені дані у корисну інформацію для майбутнього прийняття рішень.

3.10 Реалізація системи логування дій користувачів

Реалізовано підсистему логування, яка призначена для фіксації дій користувачів та контролю змін у системі. Логування є важливим елементом забезпечення прозорості роботи та дозволяє відстежувати історію виконаних операцій.[18]

Призначення системи логування

Основними завданнями підсистеми логування є:

- фіксація дій користувачів (створення, зміна, видалення даних);
- контроль виконання операцій із замовленнями;
- можливість аналізу помилок та спірних ситуацій;
- підвищення рівня безпеки системи.

Структура логів

Дані логування зберігаються у базі даних Firebase у вигляді окремої колекції logs. Кожен запис містить:

- дата та час виконання операції;

- ідентифікатор користувача;
- опис виконаної дії;
- додаткові дані (за потреби).

Такий підхід дозволяє швидко отримувати інформацію про будь-які зміни у системі або для вирішення будь-яких непорозумінь у роботі.

Реалізація логування

Для спрощення роботи з логами було створено окрему функцію, яка викликається при виконанні ключових дій у системі (створення замовлення, скасування, зміна статусу, тощо).

```
function addLog(action, details = "") {
    const user = JSON.parse(localStorage.getItem('user') || '{}');
    const log = {
        user: user.email || "system",
        action,
        details,
        date: new Date().toLocaleString('uk-UA')
    };

    // вивід у консоль на сторінці
    const container = document.getElementById('logs-console');
    if (container) {
        const div = document.createElement('div');
        div.className = 'log-entry';
        div.innerHTML = `
            <span class="log-date">[${log.date}]</span>
            <span class="log-user">${log.user}</span>:
            <span class="log-action">${log.action}</span>
            <span class="log-details">${log.details ? "- " + log.details : ""}</span>
        `;
        container.appendChild(div);
        container.scrollTop = container.scrollHeight; // автоскрол вниз
    }

    db.ref('logs').push(log).catch(err => console.error("Не вдалося додати лог:", err));
}
```

Рис. 3.27 – Функція додавання логування

Використання у системі

Функція логування викликається у ключових точках бізнес-логіки. **Наприклад:**

1. При збиранні товару:

```
function pickItem(orderId, index) {
    const itemRef = db.ref(`orders/${orderId}/items/${index}`);
    itemRef.update({ picked: true });

    db.ref(`orders/${orderId}`).once('value').then(snap => {
        const order = snap.val();
        if (order.status === "Новий") {
            db.ref(`orders/${orderId}`).update({ status: "В процесі" });
        }
        addLog("Зібрано товар", `${order.items[index].product} (замовлення ${orderId})`);
        showNotification(`Товар "${order.items[index].product}" зібрано!`, "success");
    });
}
```

Рис. 3.28 – Додавання логів функцію збирання товарів

2. При створенні замовлення:

```
775  function addOrder() {
841      db.ref('warehouses').once('value').then(snap => {
851          });
852
853          return db.ref('warehouses').set(allWH);
854      }).then(() => {
855          return db.ref('orders/' + orderId).set(newOrder);
856      }).then(() => {
857          if (typeof addLog === 'function') addLog("Система", `Створено замовлення ${orderId}`);
858          showNotification(`Замовлення #${orderId} створено! Посилок: ${parcelsArray.length}`, "success");
859          closePopup('orderPopup');
860          if (typeof initDashboard === 'function') initDashboard();
861      }).catch(err => {
862          console.error(err);
863          showNotification("Помилка збереження", "error");
864      });
865  }
```

Рис. 3.29 – Додавання логів функцію створення замовлення

3. При скасуванні замовлення:

```
// допоміжна функція для зміни статусу
function updateOrderStatus(orderId) {
    return db.ref(`orders/${orderId}`).update({
        status: 'Скасований',
        assignedTo: ""
    }).then(() => {
        showNotification("Замовлення скасовано, товари повернуто", "success");
        if (typeof addLog === 'function') addLog("Система", `Скасовано замовлення ${orderId}`);
    });
}
```

Рис. 3.30 – Додавання логів функцію скасування замовлення

Відображення логів

У системі окремо реалізований перегляд логів через панель адміністратора. По суті, це журнал дій, у який автоматично записується все важливе, що відбувається під час роботи: створення або скасування замовлень, зміна статусів, додавання товарів, редагування даних складу та інші операції.

Усі записи відображаються у вигляді звичайного списку. Для кожної дії можна побачити короткий опис, ім’я користувача та точний час виконання. Це зручно, коли потрібно швидко перевірити, хто саме щось змінив у системі. Наприклад, якщо випадково змінився залишок товару або замовлення отримало неправильний статус, адміністратор може відкрити логи й буквально за кілька секунд зрозуміти, що сталося.

Оновлення відбувається динамічно, тому нові записи з’являються майже одразу після виконання дії. Не потрібно перезавантажувати сторінку чи вручну оновлювати список. Через це логування працює більше як “жива історія” системи, а не просто архів старих записів.

Окремо була додана можливість очищення логів. З часом записів накопичується багато, особливо якщо система активно використовується щодня. Через це список стає перевантаженим і в ньому складніше знайти щось потрібне. Саме тому адміністратор може видалити старі записи, які вже не мають практичного значення. При цьому доступ до очищення має тільки адміністратор,

оскільки логи містять службову інформацію про роботу системи та дії користувачів.

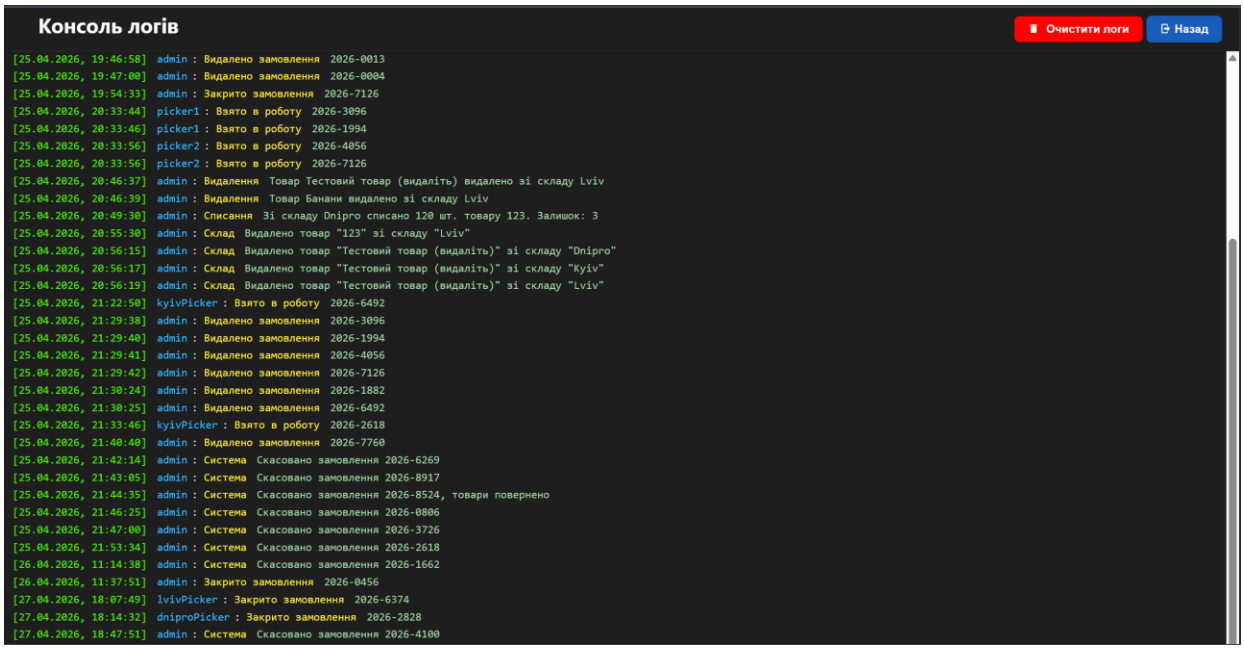


Рис. 3.31 – Відображення всіх логів на сторінці

Практичне значення

Інтегрована система логування дозволяє:

- відстежувати всі ключові дії користувачів;
- аналізувати роботу системи;
- швидко знаходити причини помилок;
- підвищувати контроль над процесами.

3.11 Реалізація чату для співробітників

Нормальна комунікація між працівниками напряму впливає на те, наскільки стабільно працюють бізнес-процеси. Особливо це помітно в системах, де замовлення обробляються постійно і будь-яка затримка з передачею інформації може створити проблему. Наприклад, склад може не побачити зміни в замовленні, або адміністратор — не отримати відповідь вчасно. У результаті все це впливає і на швидкість роботи, і на кількість помилок.

На практиці для зв’язку часто використовують телефон або сторонні месенджери. Це зручно, бо всі до цього звикли. Але є і мінуси. Частина інформації губиться в повідомленнях, щось передається усно, а потім складно зрозуміти, хто і що саме погоджував. Крім того, коли комунікація винесена за межі системи, вона ніяк не пов’язана із самими замовленнями чи задачами. Через це працівникам доводиться постійно перемикатися між різними сервісами.

Саме тому в межах проєкту було вирішено зробити окрему вбудовану систему повідомлень. Вона працює прямо всередині веб-системи і дозволяє користувачам швидко обмінюватися інформацією без переходу в сторонні додатки. Повідомлення оновлюються в реальному часі, тому нові дані з’являються практично одразу.

Такий підхід спрощує координацію між працівниками, особливо коли над замовленням одночасно працюють кілька людей. Плюс уся комунікація залишається всередині системи, і при потребі можна швидко повернутися до попередніх повідомлень або уточнень.

Архітектура зберігання повідомлень

Повідомлення в системі зберігаються у Firebase у вигляді окремої структури (гілки) chat_messages. Такий підхід дозволяє тримати весь чат в одному місці, але при цьому кожне повідомлення існує як окремий запис із власними даними.

Зазвичай у кожного повідомлення є набір базових полів: текст, час відправлення, а також службова інформація — наприклад, хто саме його відправив або до якого чату воно належить. Завдяки цьому система може не просто показувати список повідомлень, а й правильно їх сортувати, фільтрувати або розділяти між різними діалогами.

Структура у вигляді окремої гілки зручна тим, що Firebase одразу синхронізує зміни в реальному часі. Тобто як тільки з’являється новий запис у

chat_messages, він одразу стає доступним для всіх підключених клієнтів. Це і є причина, чому чат оновлюється миттєво без додаткових запитів чи перезавантаження сторінки.

Окремо це спрощує подальшу роботу з даними. Наприклад, можна легко отримати останні повідомлення, побудувати історію переписки або реалізувати додаткову логіку — на кшталт сповіщень чи логування активності користувачів.

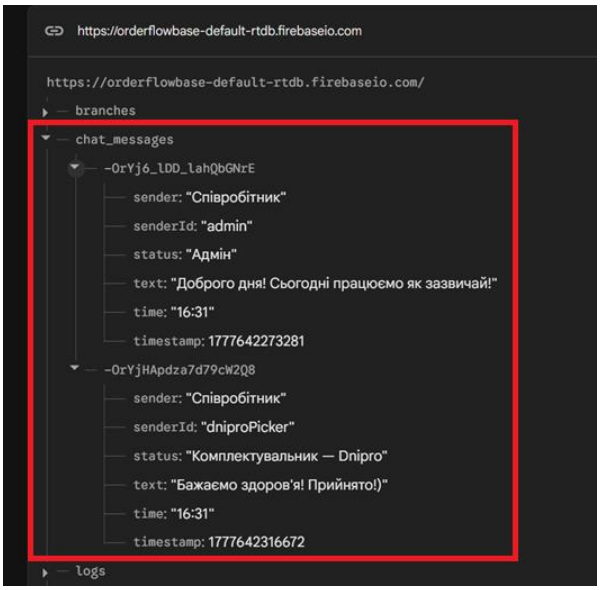


Рис. 3.32 - Зберігання повідомлень у Firebase

Усі повідомлення у собі містять:

- текст повідомлення;
- ім’я користувача та його посада;
- часову мітку (timestamp).

Це дозволяє організувати обмін повідомленнями в режимі реального часу.

Надсилання повідомлень

Надсилання повідомлень у системі реалізоване досить простою логікою: кожне нове повідомлення не “живе” окремо в інтерфейсі, а одразу записується в

базу даних як новий елемент. Після цього воно вже стає доступним для всіх частин системи, які відповідають за його відображення або подальшу обробку.

Фактично процес виглядає так: користувач вводить текст і натискає кнопку відправки, після чого запускається функція, яка формує об’єкт повідомлення. У цьому об’єкті зазвичай зберігається сам текст, час створення, а також додаткова інформація — наприклад, відправник або ідентифікатор чату. Далі цей об’єкт додається в базу даних як новий запис.

Після запису в базу повідомлення автоматично підхоплюється іншими частинами системи, які слухають зміни в реальному часі. Саме тому воно одразу з’являється в інтерфейсі отримувача без додаткового оновлення сторінки або ручних запитів. Такий підхід дозволяє підтримувати швидку і безперервну комунікацію.

Окремо варто відзначити, що така структура робить систему більш гнучкою. Одне й те саме повідомлення може використовуватися не тільки для відображення в чаті, але й для інших задач — наприклад, логування дій, тригерів

сповіщень або аналітики. Тобто запис у базі виступає як універсальна одиниця даних, яка далі може оброблятися різними модулями системи.

```
// надсилання смс з перевіркою ролі
function sendChatMessage() {
    const input = document.getElementById('chatInput');
    const text = input.value.trim();
    if (!text) return;

    const user = JSON.parse(localStorage.getItem('user') || '{}');

    const myId = user.email || user.uid || user.name;

    const now = new Date();
    const timeString = now.getHours().toString().padStart(2, '0') + ':' +
                       now.getMinutes().toString().padStart(2, '0');

    const msgData = {
        text: text,
        sender: user.name || "Співробітник",
        senderId: myId,
        status: (user.role === 'admin') ? "Адмін" : `Комплектувальник — ${user.warehouse}`,
        time: timeString,
        timestamp: firebase.database.ServerValue.TIMESTAMP
    };

    db.ref('chat_messages').push(msgData).then(() => {
        input.value = '';
    });
}
```

Рис. 3.33 – Функція надсилання повідомлення

Відображення повідомлень

Повідомлення після надходження одразу підхоплюються системою і виводяться у вигляді простого списку в чаті. Все оновлюється без перезавантаження сторінки — новий текст просто з’являється внизу, ніби сам по собі, і користувачу не потрібно нічого додатково робити.

Сам чат зроблений як popup-вікно, тобто він не відкритий постійно і не заважає роботі з основним інтерфейсом. Його можна викликати кнопкою внизу зліва. Натиснув — і вікно плавно виїжджає, закрив — так само акуратно зникає. Це зроблено більше для зручності, щоб чат не “висів” перед очима постійно.

Всередині повідомлення розміщуються послідовно, як звичайна переписка. Нові йдуть в кінець, старі залишаються вище, і чат автоматично

підкручується до останнього повідомлення, щоб не доводилось шукати актуальний текст вручну. Якщо ж користувач прокрутив історію, система це не ламає — просто додає нові повідомлення нижче.

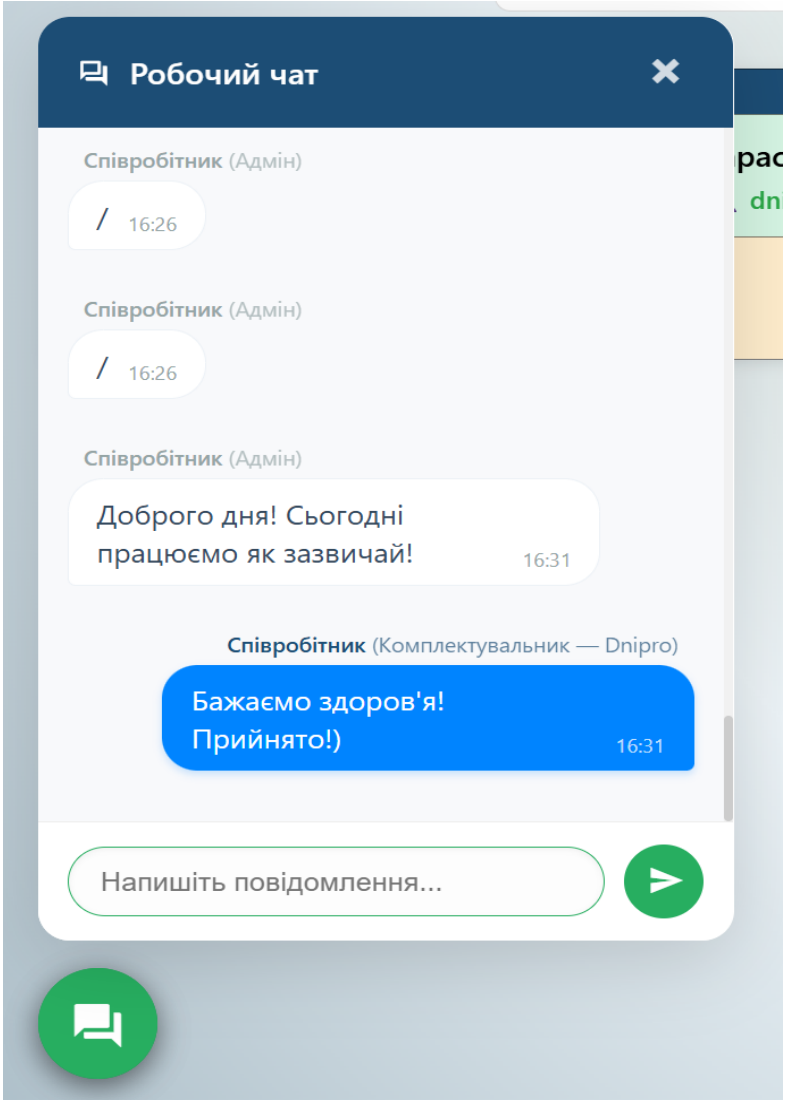


Рис. 3.34 – Вигляд робочого чату співробітників

Уся логіка працює в динамічному режимі: дані оновлюються одразу після появи в системі. Це дає відчуття живого спілкування, без затримок і ручного оновлення сторінки. У результаті чат більше схожий на звичайний месенджер, ніж на класичну форму з повідомленнями.

Автоматичне оновлення повідомлень без необхідності перезавантаження сторінки є ключовим досягненням. Завдяки використанню механізмів роботи з даними в режимі реального часу. Повідомлення відображаються у

хронологічному порядку, що забезпечує їх коректне сприйняття користувачами та логічну послідовність діалогу. Крім того, кожне повідомлення прив’язується до конкретного користувача, що дозволяє ідентифікувати автора та підвищує прозорість комунікації в системі. Сукупність цих рішень забезпечує зручність використання та наближує функціональні можливості чату до сучасних месенджерів, таких як Viber, Telegram тощо.

Дана реалізація дозволяє оперативно вирішувати робочі питання в процесі виконання завдань. Використання внутрішнього каналу комунікації сприяє зменшенню кількості помилок під час обробки замовлень, оскільки всі уточнення можуть бути швидко передані відповідним користувачам. Крім того, інтеграція чату в систему покращує взаємодію між співробітниками та підвищує загальну ефективність роботи.

ВИСНОВКИ

Якщо дивитися на різні підходи до управління замовленнями, то добре видно, що між ними є досить велика різниця — не тільки по функціоналу, а й загалом по зручності роботи. На перших етапах багато інтернет-магазинів використовують максимально прості рішення, бо це дешевше і не потребує окремої розробки. Але з часом, коли замовлень стає більше, такі підходи починають створювати більше проблем, ніж користі.

Найпростіший варіант — це ручне ведення замовлень через таблиці. Найчастіше для цього використовують Google Sheets або Microsoft Excel. По суті, працівники самі вносять інформацію про клієнтів, товари, статуси замовлень і залишки. Для невеликого магазину, де замовлень небагато, це може працювати нормально. Особливо на старті, коли немає сенсу вкладати кошти у складні системи.

Але проблема такого підходу стає помітною досить швидко. Дані потрібно постійно оновлювати вручну, а будь-яка неуважність одразу створює помилки. Наприклад, працівник може забути змінити статус замовлення або випадково вказати неправильну кількість товару. Через це виникає плутанина, а клієнти отримують неправильну інформацію. Коли замовлень кілька на день — це ще не критично. Але якщо їх десятки або більше, контролювати все вручну стає складно.

Крім цього, таблиці погано підходять для одночасної роботи великої кількості людей. Навіть якщо використовується спільний доступ, як у Google Sheets, все одно виникають ситуації, коли дані змінюються некоректно або дублюються. У результаті працівники витрачають час не на обробку замовлень, а на перевірку того, чи все правильно внесено.

Через це багато магазинів переходять до CRM-систем. Це вже більш організований підхід, де частина процесів автоматизується. Як приклади можна назвати Pipedrive, Odoo, KeyCRM або інші схожі рішення. Такі системи

дозволяють вести базу клієнтів, відстежувати статуси замовлень і контролювати частину процесів без постійного ручного втручання.

На практиці це справді спрощує роботу. Менше часу йде на рутинні дії, простіше відслідковувати, на якому етапі знаходиться замовлення, і легше працювати з клієнтами. Але навіть тут не все ідеально. Часто CRM використовується тільки для частини задач, а інші процеси все одно залишаються в окремих сервісах. Наприклад, склад може вестися в іншій програмі, доставка — через окремий кабінет служби доставки, а аналітика — ще десь окремо.

Через це інформація починає “розкидуватись” між різними системами. Працівникам доводиться постійно переключатись між вкладками, копіювати дані або дублювати інформацію вручну. Це вже краще, ніж звичайні таблиці, але все одно не вирішує проблему повністю.

Тому найбільш ефективними виглядають повноцінні веб-системи для електронної комерції, наприклад Shopify, WooCommerce або OpenCart. У таких рішеннях більшість процесів уже працює як єдина система. Замовлення, оплата, склад, доставка і навіть базова аналітика знаходяться в одному середовищі.

За рахунок цього значно менше шансів втратити інформацію або зробити помилку через людський фактор. Дані оновлюються швидше, працівники бачать актуальний стан замовлень, а керувати процесами стає простіше. Особливо це помітно, коли магазин працює з великою кількістю замовлень щодня.

Але навіть готові платформи мають свої мінуси. Найчастіше проблема виникає тоді, коли бізнесу потрібна якась нестандартна логіка роботи. Не всі процеси можна змінити “під себе”, а іноді для цього доводиться купувати окремі модулі або взагалі залучати розробників. Крім цього, є постійна залежність від самої платформи — її тарифів, оновлень і обмежень.

Отже, саме через це в окремих випадках більш доцільною стає розробка власної веб-системи. Особливо коли потрібно реалізувати специфічні процеси, які складно або незручно вписати в уже готові рішення.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Statista. Global retail e-commerce sales 2014-2027. 2023. URL: https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/ (дата звернення: 20.03.2026)

2. OMS vs ERP for Suppliers: Why You Need a True Order Management Layer 2023. URL: https://www.orderease.com/community/order-management-software-for-suppliers (дата звернення: 20.03.2026)

3. Oracle. Order Management Business Process Guide. 2023. URL: https://docs.oracle.com/en/cloud/saas/supply-chain-and-manufacturing/ (дата звернення: 21.03.2026)

4. Think with Google. Retail consumer insights and trends. 2023. URL: https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/ (дата звернення: 21.03.2026)

5. BigCommerce. Essential Guide to Order Management. 2023. URL: https://support.bigcommerce.com/s/article/Guide-to-Order-Management?language=en_US (дата звернення: 22.03.2026)

6. Топ-5 CMS для інтернет-магазину в 2025 році: порівняння популярних платформм. URL: https://brander.ua/blog/top-5-cms-dlya-internet-mahazynu-v-2025-rotsi (дата звернення: 25.03.2026)

7. What Is Shopify and How Does It Work? (2026). URL: https://www.shopify.com/blog/what-is-shopify (дата звернення: 25.03.2026)

8. Google Sheets Introduction. URL: https://www.techtarget.com/whatis/definition/Google-Spreadsheets (дата звернення: 20.03.2026)

9. Firebase. Cloud Firestore Data Model. 2024. URL: https://firebase.google.com/docs/firestore/data-model (дата звернення: 18.04.2026)

10. MDN Web Docs. JavaScript: Working with objects. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects (дата звернення: 26.03.2026)

11. MDN Web Docs. Asynchronous JavaScript (Promises, async/await). 2024.URL:https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous (дата звернення: 28.03.2026)

12. Firebase. Get started with Cloud Firestore Security Rules. 2024. URL: https://firebase.google.com/docs/firestore/security/get-started (дата звернення: 24.03.2026)

13. OWASP Foundation. Top 10 Web Application Security Risks. 2021. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 01.05.2026)

14. W3C. Cascading Style Sheets (CSS) Snapshot 2023. URL: https://www.w3.org/TR/css-2023/ (дата звернення: 21.03.2026)

15. Dashboards: Making Charts and Graphs Easier to Understand. 2017. URL: https://www.nngroup.com/articles/dashboards-preattentive/ (дата звернення: 17.04.2026)

16. Web.dev. Core Web Vitals and Performance. 2024. URL: https://web.dev/articles/vitals (дата звернення: 20.03.2026)

17. ДСТУ ISO/IEC 12207:2015. Інженерія систем і програмного забезпечення. Процеси життєвого циклу ПЗ. 2015. (дата звернення: 30.04.2026)

18. Google Cloud. Firestore Best Practices for Scalability. 2023. URL: https://cloud.google.com/firestore/docs/best-practices (дата звернення: 28.03.2026)

19. DZone. Advantages of NoSQL for Web Applications. 2022. URL: https://dzone.com/articles/top-nosql-databases-and-use-cases (дата звернення: 17.04.2026)

20. Що таке Firebase? URL: https://avada-media.ua/blog/firebase/#scho-take-firebase (дата звернення: 11.04.2026)

21. IBM. Inventory and Warehouse Management Concepts. 2022. URL: https://www.ibm.com/topics/inventory-management (дата звернення: 27.04.2026)

22. Eloquent JavaScript. Asynchronous Programming in JS. 2024. URL: https://eloquentjavascript.net/11_async.html (дата звернення: 27.03.2026)

23. CSS-Tricks. A Complete Guide to Flexbox. 2023. URL: https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (дата звернення: 25.03.2026)

24. Firebase Authentication. URL: https://firebase.google.com/docs/auth/users (дата звернення: 26.03.2026)

25. Design better data tables (The ingredients of a successful data table UI). URL: https://coyleandrew.medium.com/design-better-data-tables-4ecc99d23356 (дата звернення: 14.04.2026)

26. Microsoft Learn. Single Page Application Architecture. 2023. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/ (дата звернення: 15.04.2026)

27. V8 Engine Blog. Performance of Modern JavaScript Engines. 2023. URL: https://v8.dev/features (дата звернення: 28.03.2026)

28. HTML5 Introduction. URL: https://www.w3schools.com/Html/html_intro.asp (дата звернення: 25.03.2026)

ДОДАТОК А

Лістинг коду Авторизації та розмежування ролей (перевірка прав доступу)

```
document.addEventListener('DOMContentLoaded', () => {
    const user = JSON.parse(localStorage.getItem('user') || '{}');
        if (user.role === 'admin') {
        const adminBtn = document.getElementById('analytics-btn');
        if (adminBtn) adminBtn.style.display = 'flex';
    }
});
```

ДОДАТОК Б

Лістинг коду Асинхронне завантаження данних через проміси Firebase(Orders)

```
async function loadOrders() {
    const ordersContainer = document.getElementById('orders-list');
    const snapshot = await db.ref('orders').once('value');
    const orders = snapshot.val();
    if (!orders) {
        ordersContainer.innerHTML = '<p>Замовлень немає</p>';
        return;
    }
    renderOrders(orders);
}
```

ДОДАТОК В

Лістинг коду Зміна статусу замовлення та Audit Trail

```
async function updateOrderStatus(orderId, newStatus) {
    const user = JSON.parse(localStorage.getItem('user'));
        await db.ref(`orders/${orderId}`).update({
        status: newStatus,
        updatedBy: user.name
    });
    const logEntry = {
        user: user.name,
        action: "Зміна статусу",
        details: `Замовлення ${orderId} змінено на: ${newStatus}`,
        date: new Date().toLocaleString()
    };
    await db.ref('logs').push(logEntry);
}
```

ДОДАТОК Г

Лістинг коду Логіка мультискладовості (Переміщення товарів)

```
async function moveProductBetweenWarehouses(sourceWhName, targetWhName, productName, qtyToMove, reason) {
    const snapshot = await db.ref('warehouses').once('value');
    const allWh = snapshot.val();
    const sourceItems = allWh[sourceWhName] || [];
    const targetItems = allWh[targetWhName] || [];
    const itemInSource = sourceItems.find(i => i.product === productName);
    if (itemInSource && itemInSource.qty >= qtyToMove) {
        itemInSource.qty -= qtyToMove;


        const itemInTarget = targetItems.find(i => i.product === productName);
        if (itemInTarget) {
            itemInTarget.qty += qtyToMove;
        } else {
            targetItems.push({ product: productName, qty: qtyToMove, price: itemInSource.price });
        }


        await db.ref('warehouses').set(allWh);
    }
}
```

ДОДАТОК Ґ

Лістинг коду Пошук та фільтрація в реальному часі

```
function searchOrders(query) {
    const allRows = document.querySelectorAll('.order-row');
    const lowerQuery = query.toLowerCase();

    allRows.forEach(row => {
        const text = row.innerText.toLowerCase();
        row.style.display = text.includes(lowerQuery) ? '' : 'none';
    });
}
```

ДОДАТОК Д

Лістинг коду Логіка чату для співробітників

```
function toggleChat() {
    const chat = document.getElementById('chatPopup');
    if (chat.style.display === 'flex') {
        chat.style.display = 'none';
    } else {
        chat.style.display = 'flex';
        loadChatMessages();
        // скролл
        setTimeout(() => {
            const container = document.getElementById('chat-messages-area');
            if(container) container.scrollTop = container.scrollHeight;
        }, 100);
    }
}

// обробка Enter
function handleChatKey(e) {
    if (e.key === 'Enter') sendChatMessage();
}
```

```
// надсилання
function sendChatMessage() {
    const input = document.getElementById('chatInput');
    const text = input.value.trim();
    if (!text) return;

    const user = JSON.parse(localStorage.getItem('user') || '{}');
    const senderName = user.name || "Співробітник";
    const senderWH = user.warehouse || "Адмін";

    const msgData = {
        text: text,
        sender: senderName,
        warehouse: senderWH,
        timestamp: firebase.database.ServerValue.TIMESTAMP
    };

    db.ref('chat_messages').push(msgData).then(() => {
        input.value = '';
    });
}

// надсилання смс з перевіркою ролі
```

```
function sendChatMessage() {
    const input = document.getElementById('chatInput');
    const text = input.value.trim();
    if (!text) return;

    const user = JSON.parse(localStorage.getItem('user') || '{}');
    const myId = user.email || user.uid || user.name;

    const now = new Date();
    const timeString = now.getHours().toString().padStart(2, '0') + ':' +
                       now.getMinutes().toString().padStart(2, '0');

    const msgData = {
        text: text,
        sender: user.name || "Співробітник",
        senderId: myId,
        status: (user.role === 'admin') ? "Адмін" : `Комплектувальник —
${user.warehouse}`,
        time: timeString,
        timestamp: firebase.database.ServerValue.TIMESTAMP
    };

    db.ref('chat_messages').push(msgData).then(() => {
```

```
        input.value = '';
    });
}

function loadChatMessages() {
    const container = document.getElementById('chat-messages-area');
    if (!container) return;

    const user = JSON.parse(localStorage.getItem('user') || '{}');
    const myId = user.email || user.uid || user.name;

    db.ref('chat_messages').limitToLast(50).on('value', (snapshot) => {
        container.innerHTML = '';
        const msgs = snapshot.val();
        if (!msgs) return;

        Object.values(msgs).forEach(m => {
            const isMy = m.senderId === myId;

            const msgDiv = document.createElement('div');
            msgDiv.className = `msg ${isMy ? 'my' : 'others'}`;

            msgDiv.innerHTML = `
```

```
            <div class="msg-info">${m.sender} <span style="font-weight:normal;
opacity:0.7;">(${m.status || ''})</span></div>
            <div class="msg-bubble">
              <div class="msg-text">${m.text}</div>
              <div class="msg-time">${m.time || ''}</div>
            </div>
          `;
          container.appendChild(msgDiv);
        });
        container.scrollTop = container.scrollHeight;
    });
}
```