

Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни «Веб-технології та веб-дизайн»

Тема «Розробка веб-додатка з ведення інвентаризації»

Студента 3 курсу AI-221 групи

Спеціальності 122 – «Комп'ютерні науки»

Попазова П.І.

Керівник: д. Червоненко П.П.

Національна шкала _____

Кількість балів: _____

Оцінка: ECTS _____

Члени комісії

(підпис) _____ (прізвище та ініціали)

(підпис) _____ (прізвище та ініціали)

(підпис) _____ (прізвище та ініціали)

м. Одеса – 2024 рік

ЗМІСТ

Вступ.....	7
1 Актуальність, аналіз аналогів.....	8
1.1 Актуальність веб-додатка.....	8
1.2 Аналіз додатка-аналога Zoho Inventory	9
1.3 Аналіз додатка-аналога Usrasy.....	11
1.4 Бачення програмного продукту	12
2 Алгоритмічне забезпечення веб-сайту.....	14
2.1 Реєстрація, авторизація користувача	14
2.2 Бізнес-логіка застосунка.....	18
3 Проектування веб-сайту.....	21
3.1 Мета та завдання ІС	21
3.2 Кінцева аудиторія ІС.....	21
3.3 Функціональні вимоги ІС	22
3.4 Моделювання прецедентів	25
3.5 Нефункціональні вимоги	27
3.6 Architecture Constraints.....	29
3.7 UI View	31
3.8 Logical View	37
3.9 Deployment View	39
3.10 Design View	40
3.11 Process View	40
3.12 Data View	42
3.13 Interface View.....	45
3.14 Security View	47
3.15 Infrastructure View	47
3.16 Delivery Strategy View.....	48
4 Реалізація веб сайту	49

4.1 API Gateway	49
4.2 Inventory Service	53
4.3 User Service	58
4.4 Метрика застосунка	61
5 Забезпечення якості веб-сайту	63
5.1 Модульне тестування	63
5.2 Функціональне тестування	69
5.3 Інструкція користувача	75
Висновки	83
Перелік використаних джерел	84
Додаток А – СПОВІЩЕННЯ ПРО НЕДОСТАТНЮ КІЛЬКІСТЬ ТОВАРІВ	86
Додаток Б – Форма для замовлення товарів	87

Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ

студенту Попазову Петру Івановичу

група АІ-221

1. Тема роботи «Розробка веб-додатка з ведення інвентеризації»

2. Термін здачі студентом закінченої роботи

10.12.2024

3. Початкові дані до проекту (роботи). Користувачі мають змогу зареєструватися, увійти у систему та виконати двуфакторну авторизацію. Користувачі створюють команди, додають користувачів у команди з певними ролями. Користувачі додають товари до системи, вказуючи дані такі, як назва, опис, ціна, мінімальний та максимальний рівень на складі і т.д. При зменшенні рівня товару нижче зазначеного, надсилається лист на електронну адресу із формою для замовлення поставки, з'являється відповідне повідомлення на сторінці. Всі дії над товарами фіксуються та відображаються у таблиці. Користувачі бачать статистику по продуктам, командам, доходам.

4. Зміст розрахунково-пояснювальної записки: Розділ 1 – Актуальність, аналіз аналогів. Розділ 2 – Алгоритмічне забезпечення веб-сайту. Розділ 3 – Проектування веб-сайту. Розділ 4 – Реалізація веб сайту. Розділ 5 – Забезпечення якості веб-сайту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): ER-діаграма, діаграма логічного представлення.

Завдання видано
(підпис викладача)

01.09.2024

Завдання прийнято до виконання
(підпис студента)

10.09.2024

АНОТАЦІЯ

Попазов П.І. Розробка програмного забезпечення для управління складськими запасами» : курсова робота за спеціальністю «122 Комп'ютерні науки» керівник Червоненко Петро Петрович. – Одеса : Нац. ун-т «Одес. Політехніка», 2024. – 86 с.

Метою роботи є створення системи, яка дозволяє ефективно управляти складськими запасами, моніторити їх рівень, попереджати про низькі залишки, аналізувати рух товарів і генерувати статистику для підтримки бізнес-процесів.

У ході роботи використано мікросервісну архітектуру з реалізацією сервісів на базі Spring Boot. Для збереження даних застосовано PostgreSQL та MongoDB, а для комунікації між мікросервісами – Apache Kafka. Для клієнтської частини використано React і Tailwind CSS, що забезпечують сучасний і адаптивний інтерфейс користувача. Методи розробки включали проектування REST API, роботу з базами даних, налаштування систем сповіщень та використання методологій тестування.

У результаті роботи створено програмне забезпечення, яке дозволяє додавати та редагувати інформацію про товари, відстежувати рух товарів між складами, автоматично генерувати сповіщення про низький рівень запасів, а також надавати статистику руху товарів у вигляді звітів.

Рекомендована галузь використання: системи управління запасами для малого та середнього, включаючи роздрібну торгівлю, логістику, виробництво та електронну комерцію. Рішення спрямоване на оптимізацію бізнес-процесів, підвищення точності обліку та зниження витрат, пов'язаних з управлінням складськими запасами.

Ключові слова: управління складськими запасами, система сповіщень, моніторинг рівня запасів, мікросервісна архітектура, електронна комерція.

ABSTRACT

Petro I. Popazov. Development of software for warehouse stock management": coursework in the specialty "122 Computer Science"; supervisor Chervonenko Petro Petrovych. – Odesa: Odesa Polytech. Nat. Univ. 2024. – 86 p.

The goal of work is to create a system that allows users to effectively manage warehouse stocks, monitor their level, warn about low balances, analyze the movement of goods and generate statistics to support business processes.

In the course of work, a microservice architecture is used with the implementation of services based on Spring Boot. PostgreSQL and MongoDB are used for data storage, and Apache Kafka is used for communication between microservices. For the client part, React and Tailwind CSS are used, which provides a modern and responsive user interface. The development methods included designing a REST API, working with databases, setting up a notification system, and using a testing methodology.

As a result of the work, software was created that allows users to add and edit information about products, warehouses, track the movement of products, automatically generate summaries of low inventory levels and send them via email, as well as provide statistics on the movement of goods in reports.

Recommended Industry: Small and medium-sized inventory management systems, including retail, logistics, manufacturing and e-commerce. The solution is aimed at optimizing business processes, increasing accounting accuracy and reducing costs associated with warehouse stock management.

Keywords: inventory management, notification system, inventory level monitoring, microservice architecture, e-commerce.

ВСТУП

У сучасному бізнес-середовищі, де конкуренція та технології швидко змінюються, розробка веб-застосунку для інвентаризації стає надзвичайно важливою. Такий застосунок автоматизує процеси управління запасами, що зменшує ризик помилок і економить час. Він забезпечує доступ до актуальної інформації про запаси в реальному часі, що дозволяє приймати обґрунтовані рішення. Мобільний доступ до даних та гнучкість у масштабуванні відповідають потребам бізнесу, а покращене обслуговування клієнтів і зменшення витрат підвищують конкурентоспроможність.

Веб-застосунки для інвентаризації мають ряд важливих функцій, які роблять їх незамінними для ефективного управління запасами. По-перше, вони забезпечують відстеження запасів, що дозволяє моніторити кількість товарів на складі в реальному часі та їхню історію руху. По-друге, функція управління замовленнями автоматизує процеси створення, обробки та виконання замовлень, включаючи повернення та зміни. Також важливою є аналітика та звітність, яка генерує звіти про запаси, продажі та прогнози попиту, що допомагає приймати обґрунтовані рішення. Крім того, такі веб-застосунки дозволяють зберігати інформацію про постачальників, а саме – контактні дані та історію закупівель.

Мобільний доступ до системи дозволяє управляти запасами на ходу, а функція керування цінами допомагає оптимізувати цінову політику, включаючи знижки та акції. Нарешті, захист даних забезпечує безпеку інформації, шифрування даних та регулярне резервне копіювання, що є критично важливим для захисту від втрат.

Всі вищенаведені функції в комплексі дозволяють компаніям ефективно управляти своїми запасами, оптимізувати бізнес-процеси та підвищувати задоволеність клієнтів.

1 АКТУАЛЬНІСТЬ, АНАЛІЗ АНАЛОГІВ

1.1 Актуальність веб-додатка

Розробка веб-застосунку для інвентаризації залишається надзвичайно актуальною в сучасному бізнес-середовищі, яке швидко змінюється під впливом сучасних технологій. У 2024 році компанії стикаються з необхідністю оптимізації своїх процесів управління запасами, щоб залишатися конкурентоспроможними та задовольняти потреби клієнтів.

Однією з основних причин для розробки такого застосунку є підвищення ефективності. Автоматизація процесів інвентаризації дозволяє компаніям значно зменшити час, витрачений на ручне введення даних і обробку інформації, що не тільки знижує ризики помилок, але й дозволяє співробітникам зосередитися на більш важливих завданнях, таких як стратегічне планування та покращення обслуговування клієнтів.

Крім того, веб-застосунки для інвентаризації надають можливість моніторингу запасів у реальному часі, що дозволяє компаніям оперативно реагувати на зміни в попиті та забезпечувати наявність товарів на складі, що є особливо важливо в умовах нестабільного ринку, де швидкість реакції може стати ключовим фактором успіху. Застосунки також дозволяють генерувати детальні звіти та аналітику, що сприяє прийняттю обґрунтованих рішень на всіх рівнях управління.

В умовах зростаючої конкуренції та змінюваних споживчих уподобань компанії також повинні звертати увагу на можливості масштабування своїх систем. Веб-застосунки легко адаптуються до зростання бізнесу, дозволяючи додавати нові функції або інтегрувати нові платформи за потреби, що забезпечує гнучкість, необхідну для швидкого реагування на зміни на ринку.

Веб-застосунки для інвентаризації використовуються різними типами організацій у багатьох галузях. По-перше, роздрібні магазини—як малі, так і великі мережі—застосовують ці рішення для управління запасами товарів, обробки

замовлень і відстеження продажів. Оптові постачальники потребують ефективних інструментів для моніторингу великих обсягів товарів і управління постачаннями. Виробничі підприємства контролюють сировину та готову продукцію, а також запаси на складах. Логістичні компанії використовують ці системи для моніторингу вантажів, оптимізації складу та управління транспортними процесами.

У сфері електронної комерції такі застосунки використовують для обробки замовлень та генерації електронних чеків. Медичні заклади, такі як лікарні та клініки, контролюють рівень медичних запасів. Готелі та ресторани використовують ці рішення для управління запасами продуктів харчування, напоїв та інших матеріалів. Будівельні компанії потребують інструментів для моніторингу матеріалів та обладнання, необхідного для виконання проектів.

Також університети та освітні установи можуть використовувати інвентаризаційні рішення для управління запасами навчальних матеріалів і обладнання. Навіть неприбуткові організації застосовують ці рішення для управління запасами товарів, призначених для благодійних програм.

Веб-застосунки для інвентаризації є універсальними інструментами, що підходять для різних галузей і типів бізнесу, допомагаючи покращувати ефективність управління запасами та оптимізувати бізнес-процеси.

1.2 Аналіз додатка-аналога Zoho Inventory

Zoho Inventory [1] — це потужне рішення для управління запасами, яке входить до складу екосистеми Zoho, розробленої для допомоги бізнесам у відстеженні, управлінні та оптимізації їхніх запасів. Zoho Inventory дозволяє відстежувати рівні запасів у режимі реального часу, що допомагає компаніям уникати переповнення складів або нестачі товарів. Користувачі можуть налаштовувати автоматичні сповіщення про низькі запаси, щоб вчасно поповнювати запаси.

Платформа дозволяє управляти як закупівлею, так і продажем товарів, включаючи створення та відстеження замовлень постачальників і клієнтів. Інтерфейс дозволяє легко управляти історією замовлень, що забезпечує повний огляд діяльності. Zoho Inventory інтегрується з популярними платформами електронної комерції, такими як Shopify, Amazon, eBay та Etsy. Це забезпечує синхронізацію запасів та автоматизацію процесу продажу. Користувачі можуть легко управляти замовленнями та відстежувати продажі на різних каналах з одного місця. Платформа дозволяє керувати кількома складами, що корисно для бізнесів з різними локаціями. Користувачі можуть відстежувати товарні запаси в різних місцях та оптимізувати розподіл товарів. Zoho Inventory надає розширені можливості для створення звітів та аналітики. Користувачі можуть отримувати статистику про продажі, запаси, замовлення та постачальників, що дозволяє приймати обґрунтовані рішення на основі даних. Інструменти автоматизації, такі як автоматичне створення замовлень, допомагають зменшити трудозатрати і підвищити ефективність. Користувачі можуть налаштовувати автоматичні робочі процеси відповідно до своїх потреб.

Переваги Zoho Inventory:

- комплексна функціональність;
- інтуїтивно зрозумілий інтерфейс;
- автоматизація процесів;
- гнучка інтеграція;
- потужні аналітичні інструменти;
- підтримка кількох валют і мов.

Недоліки Zoho Inventory

- обмежена безкоштовна версія;
- складність у налаштуванні;
- вартість підписки;
- обмеження у кастомізації.

Отже, Zoho Inventory є потужним інструментом для управління запасами, який підходить для бізнесів різного розміру. Його багатфункціональність, простота використання та інтеграція з іншими платформами роблять його привабливим вибором для компаній, які прагнуть оптимізувати свої процеси управління запасами, підвищити ефективність і забезпечити якісне обслуговування клієнтів.

1.3 Аналіз додатка-аналога Uspasy

Uspasy [2] — це український інноваційний веб-застосунок, розроблений для ефективного управління запасами та обліку товарів у бізнесах, що займаються роздрібною торгівлею, оптовими постачаннями та електронною комерцією. Цей інструмент стає все більш популярним серед підприємців завдяки своїй функціональності та простоті використання, що дозволяє значно спростити процеси управління запасами.

Однією з основних функцій Uspasy є управління запасами в режимі реального часу. Це означає, що користувачі можуть миттєво відстежувати наявність товарів на складі, що допомагає уникнути проблем, пов'язаних з нестачею товарів або переповненням складів. Система автоматично оновлює інформацію про запаси, що забезпечує точність даних і дозволяє підприємствам бути в курсі своїх ресурсів.

Автоматизація процесів є ще однією важливою перевагою Uspasy. Застосунок дозволяє автоматично створювати замовлення, обробляти постачання і генерувати звіти, що значно зменшує трудозатрати та підвищує ефективність бізнес-процесів. Автоматизація також допомагає зменшити ризик помилок, які можуть виникнути під час ручного введення даних.

Аналіз даних — це ще одна ключова функція, яка робить Uspasy корисним інструментом для прийняття обґрунтованих рішень. Застосунок надає можливості для створення детальних звітів та аналітики, що дозволяє підприємствам

отримувати цінну інформацію про продажі, запаси та ринкові тенденції. Це, в свою чергу, сприяє оптимізації управління запасами та підвищенню прибутковості.

Usrasyu також підтримує інтеграцію з іншими платформами та сервісами, такими як електронна комерція та системи управління бізнесом за допомогою REST API.

Переваги Uspasy:

- інтуїтивно зрозумілий інтерфейс;
- управління запасами в реальному часі;
- автоматизація процесів, що знижує трудозатрати та підвищує ефективність;
- аналітика та звітність;
- інтеграція з іншими платформами;
- гнучкість.

Недоліки Uspasy:

- обмежена функціональність безкоштовної версії;
- необхідність в Інтернет-з'єднанні;
- потреба в навчанні для більш складних функцій.

Отже, Uspasy є потужним інструментом для управління запасами, який допомагає бізнесам оптимізувати їхні процеси, зменшити витрати та підвищити ефективність. Його функціональність, простота використання та можливості інтеграції роблять його привабливим вибором для компаній, які прагнуть вдосконалити управління запасами та забезпечити якісне обслуговування клієнтів.

1.4 Бачення програмного продукту

У книзі «Перетинаючи прірву» маркетолога Джеффри А. Мур (Crossing the Chasm, Geoffrey A. Moore) описується, як формалізація бізнес (часткова) вимог може привабити перших прихильників, які з ентузіазмом приймуть продукт і можуть самі почати рекламувати та переконати або навчити інших людей

користуватися цим продуктом. Існує багато різних шаблонів опису бачення продукту. Один з них описано у вказаній книзі у вигляді 7-ми елементів [3]:

- For: «target customer» (Для кого: цільовий споживач);
- Who: «needs» (Хто: потреби);
- The: «product name» (назва продукту);
- Is: «product category» (Є: категорія продукту);
- That: «product benefit. Reason to buy» (Що: переваги продукту, причина купувати);
- Unlike: «competitors» (на відміну: продукти-конкуренти);
- Our product: «differentiation or value proposition» (наш продукт: позитивні відмінності від інших продуктів або суттєві пропозиції).

Бачення програмного продукту: «FOR споживача, який є повнолітньою працюючою особою з середнім або вище середнього рівнем доходу, WHO має потребу в ефективному управлінні запасами, точній інформації про наявні товари та оптимізації процесів обліку, THE Easy Store App IS веб-додаток THAT розроблений для забезпечення користувачів зручними інструментами для ведення обліку товарів, відстеження запасів та автоматизації процесів замовлень. UNLIKE інші програмні продукти, такі як Zoho Inventory та Uspasy, які є повністю комерційними рішеннями та мають заскладний інтерфейс користувача, OUR продукт має повністю безоплатний доступ до усіх функцій та має простий, інтуїтивно зрозумілий інтерфейс користувача».

2 АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ВЕБ-САЙТУ

2.1 Реєстрація, авторизація користувача

Детальний опис алгоритма реєстрації користувачів.

Вхідні параметри:

- email: рядок, що представляє email користувача;
- password: рядок, що представляє пароль користувача.

Вихідні параметри:

- унікальний ідентифікатор користувача (Integer userId) після його успішного збереження в базі даних.

Блок-схема алгоритму реєстрації (рис. 2.1):

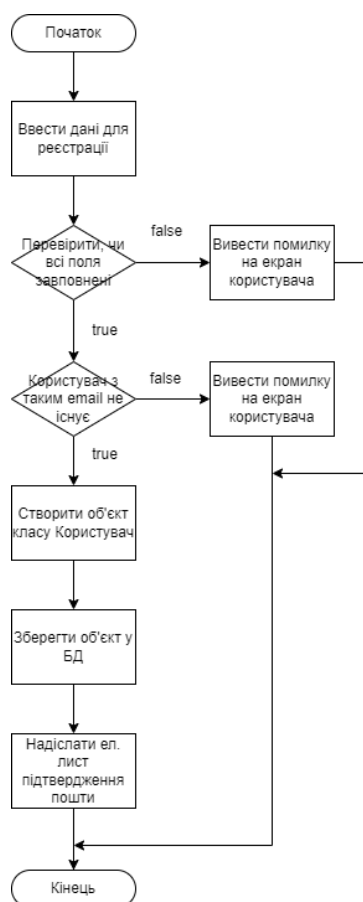


Рисунок 2.1 – Реєстрація користувача

Алгоритм реєстрації:

- перевірка і валідація вхідних даних: перед збереженням користувача виконується валідація введених полів (email та пароль), що гарантує коректність введених даних;
- хешування паролю: пароль користувача хешується за допомогою PasswordEncoder для забезпечення безпеки зберігання;
- створення об'єкта користувача: на основі вхідних даних створюється об'єкт AppUser;
- збереження користувача в базі даних: користувач зберігається через репозиторій appUserRepository;
- надсилання верифікаційного листа: генерується токен активації, який надсилається користувачеві для підтвердження акаунта через email;
- логування реєстрації: генерується повідомлення про реєстрацію, яке надсилається іншому мікросервісу через userRegisteredProducer.

Трудомісткість (Time Complexity):

- хешування пароля: $O(1)$ — операція хешування паролю займає постійний час;
- збереження користувача в базі даних: $O(1)$ — збереження нового запису до бази даних;
- генерація токена активації: $O(n)$ — де n — це довжина токена, яку можна вважати константною, отже, складність генерації також близька до $O(1)$;
- надсилання email: $O(1)$ — це виклик асинхронного процесу, тому не враховується в основній логіці;

Залежність використання пам'яті (Space Complexity):

- збереження паролю: $O(1)$ — для збереження хешу пароля;
- об'єкт користувача: $O(1)$ — структура даних користувача займає постійний обсяг пам'яті;
- токен активації: $O(1)$ — токен займає фіксовану кількість пам'яті;

- база даних: $O(n)$, де n — кількість користувачів, оскільки кожен новий користувач потребує місця для зберігання даних у базі;
- загальна залежність від пам'яті — $O(1)$ для кожного окремого користувача.

Діаграма (рис. 2.2) демонструє архітектуру аутентифікації в додатку на основі Spring Security з використанням JWT (JSON Web Token).

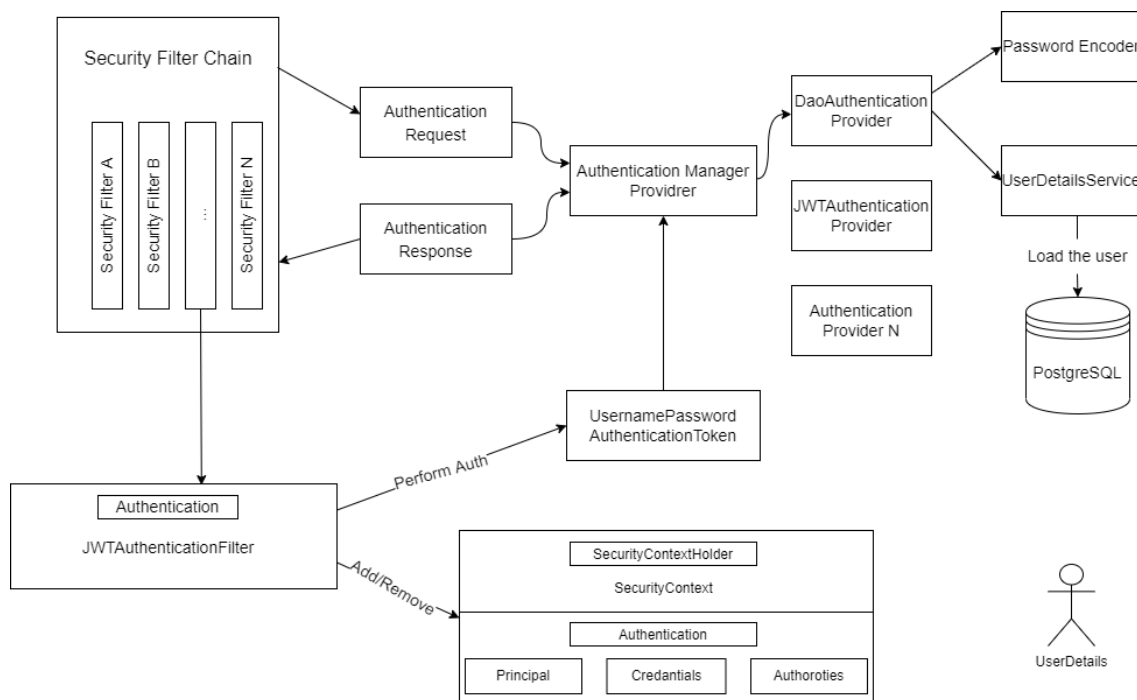


Рисунок 2.2 – Архітектура аутентифікації у Spring Security

Security Filter Chain (Ланцюг фільтрів безпеки) складається з кількох фільтрів безпеки (Security Filter A, B, ..., N). Фільтри взаємодіють з запитами, зокрема, перевіряють наявність токена або валідність аутентифікаційних даних.

Authentication Request & Authentication Response (Запит та відповідь аутентифікації): - запит аутентифікації передається через фільтри до менеджера аутентифікації. Відповідь надсилається назад після успішної або неуспішної аутентифікації.

Authentication Manager Provider (Менеджер аутентифікації) – це ключовий елемент, що делегує аутентифікацію до різних провайдерів (DaoAuthenticationProvider, JWTAuthenticationProvider, і AuthenticationProvider N).

`DaoAuthenticationProvider` – використовує `Password Encoder` та `UserDetailsService` для аутентифікації на основі даних із бази. Після отримання даних користувача з бази PostgreSQL через `UserDetailsService` перевіряє пароль користувача за допомогою `Password Encoder`.

`JWTAuthenticationProvider` відповідає за перевірку та обробку JWT-токену. Якщо токен валідний, користувач вважається аутентифікованим.

`Authentication Providers (N)`: можливі інші провайдери аутентифікації для різних типів методів (LDAP, OAuth і т.д.).

`UsernamePasswordAuthenticationToken` – Використовується для передачі аутентифікаційних даних (логін і пароль) від користувача до менеджера аутентифікації.

`JWTAuthenticationFilter` – фільтр, який перехоплює запити, що містять JWT, і передає їх для подальшої обробки. Встановлює користувача у `SecurityContext` після успішної аутентифікації.

`SecurityContextHolder` – центральний компонент, який зберігає контекст безпеки для поточного потоку. Включає інформацію про аутентифікованого користувача (`Principal`), його облікові дані (`Credentials`) та права доступу (`Authorities`).

За авторизацію користувача відповідає метод `authenticate`. Опис алгоритму:

Вхідні параметри: `request` (типу `AuthUserRequest`): містить email та пароль користувача для аутентифікації. Вихідні параметри: `AuthenticationResponse`: об'єкт з JWT-токеном, якщо аутентифікація успішна. Метод працює лише з валідними об'єктами `AuthUserRequest`. Трудомісткість: $O(1)$ для генерації токена, $O(n)$ для пошуку користувача в базі, де n — кількість користувачів. Обґрунтована залежність споживання пам'яті від розміру вхідних параметрів: використовує постійну кількість пам'яті для `claims` та `AuthenticationResponse`, що оцінюється як $O(1)$. Блок-схема алгоритму (рис. 2.3).

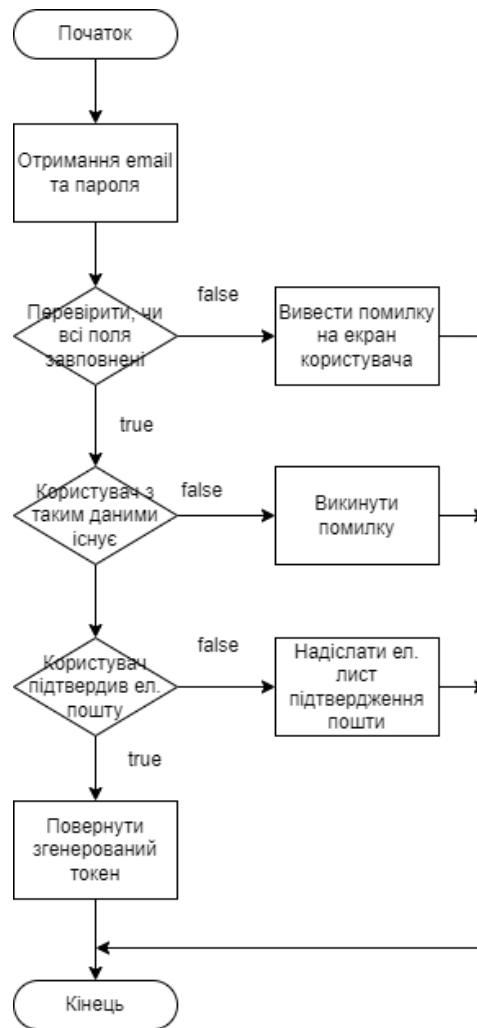


Рисунок 2.3 – Авторизація користувача

2.2 Бізнес-логіка застосунка

Метод `createProduct` призначений для зберігання нового товару. Вхідні параметри: `productRequest` (типу `CreateProductRequest`): містить дані для створення продукту (ім'я, код, баркод, опис, ціна, кількість, тощо), `userId` (типу `String`): ідентифікатор користувача, який створює продукт. Вихідні параметри: `ServerResponse<Integer>`: об'єкт, що містить ID нового продукту. Пошук категорії: $O(n)$, де n — кількість категорій у репозиторії. Створення нового продукту: $O(1)$ (фіксовані операції). Метод використовує фіксовану кількість пам'яті для об'єктів (`Category`, `Inventory`, `Product`), що оцінюється як $O(1)$. Блок-схема алгоритму (рис. 2.4).



Рисунок 2.4 – Створення нового товару

Метод `transfer` відповідає за перенаправлення товару з одного складу в інший. Вхідні параметри: `request` (типу `TransferRequest`): містить дані для передачі продукту, такі як ідентифікатори продукту, інвентарю та складів, `userId` (типу `String`): ідентифікатор користувача, який виконує передачу. Вихідні параметри: `ServerResponse<String>`: об'єкт, що містить повідомлення про успіх або помилку передачі. Трудомісткість: пошук продукту: $O(1)$ (ідентифікатор продукту), пошук інвентарю: $O(1)$ (ідентифікатор інвентарю). Операції з оновленнями: $O(1)$. Використовує постійну кількість пам'яті для об'єктів (`Product`, `Inventory`), що оцінюється як $O(1)$. Блок-схема алгоритму (рис. 2.5).

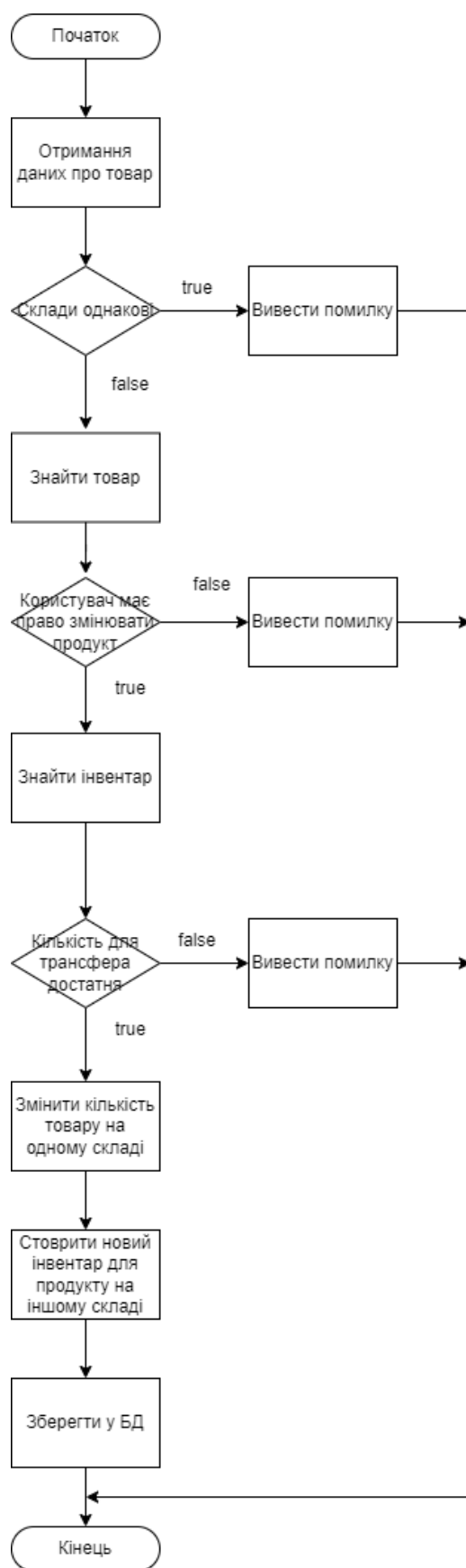


Рисунок 2.4 – Трансфер товару

3 ПРОЕКТУВАННЯ ВЕБ-САЙТУ

3.1 Мета та завдання ІС

Ціль ІС: створення веб-додатка, що забезпечує ефективне управління обліком товарів на складах, автоматизацію процесів обліку, передачі та управління запасами, підвищуючи точність, швидкість і зручність роботи користувачів.

Мета веб-додатка:

1. Забезпечити користувачів зручними інструментами для автоматичного обліку товарів, що надходять та відправляються зі складів, знижуючи ймовірність людських помилок.
2. Надати можливість в режимі реального часу відстежувати наявність товарів, щоб уникнути дефіциту або надлишків.
3. Спрощення процесів передачі товарів між складами, обліку поставок і контролю за термінами зберігання, що дозволить зекономити час і ресурси.
4. Аналіз даних: Забезпечити аналітичні звіти щодо обігу товарів, використання складів та ефективності управлінських рішень, що допоможе у стратегічному плануванні.

3.2 Кінцева аудиторія ІС

Інформаційна система для ведення обліку товарів на складах орієнтована на задоволення потреб користувачів, забезпечуючи зручність, доступ до необхідної інформації та інструменти для ефективного управління запасами та обробки замовлень. Можливі користувачі системи:

1. Менеджери складів: ефективний моніторинг запасів на складах; можливість швидко отримувати звіти про наявність товарів і їх обіг; інструменти для планування поповнення запасів.
2. Співробітники складу: інтуїтивно зрозумілий інтерфейс для обліку надходжень і відправлень товарів; можливість швидкої обробки замовлень і

перевірки наявності товарів; системи для сканування штрих-кодів для швидшого внесення даних.

3. Керівництво компанії: стратегічна інформація про запаси, обсяги продажів і тренди для прийняття рішень; інструменти для моніторингу ефективності роботи складу та управлінських процесів.

3.3 Функціональні вимоги ІС

FR1. Реєстрація нового користувача:

FR1.1 Користувач повинен ввести обов'язкові дані: електронну пошту, пароль.

FR1.2 Пароль повинен відповідати критеріям безпеки: не менше 8 символів, містити щонайменше одну велику літеру, одну цифру та один спеціальний символ.

FR1.3 У разі введення некоректних або неповних даних система повинна видати повідомлення: «Будь ласка, заповніть усі обов'язкові поля коректно».

FR1.4 Якщо електронна пошта вже зареєстрована, система повинна видати повідомлення: «Ця електронна пошта вже використовується».

FR1.5 При успішній реєстрації користувач повинен отримати повідомлення: «Реєстрація успішна. Ви можете увійти до системи».

FR1.6 Після успішної реєстрації система повинна надіслати електронний лист на вказану адресу для підтвердження реєстрації.

FR1.7 Користувач повинен перейти за посиланням у листі для завершення процесу реєстрації.

FR2. Авторизація користувача

FR2.1 Користувач повинен ввести електронну пошту та пароль для авторизації.

FR2.2 У разі некоректного введення електронної пошти або пароля система повинна видати повідомлення: «Некоректний логін або пароль», без уточнення, що саме було введено неправильно.

FR2.3 При успішній авторизації користувач повинен бути перенаправлений до робочого кабінету.

FR2.4 Система повинна зберігати сесію користувача, щоб він не вимагав повторного входу протягом певного часу (наприклад, 30 хвилин бездіяльності).

FR3 Управління командою

FR3.1 Система повинна дозволяти адміністратору створювати нову команду, вказуючи її назву та опис (той, хто створює команду – автоматично адміністратор).

FR3.2 Система повинна дозволяти адміністратору додавати нових учасників за електронною поштою до команди з відповідними ролями.

FR3.3 У разі, якщо такий користувач ще не зареєстрований – надати відповідне повідомлення.

FR3.4 Адміністратор команди повинен мати можливість видаляти учасників із команди.

FR4. Управління товаром

FR4.1 Користувач повинен ввести всю необхідну інформацію про товар: назву, код, штрих-код, опис, ціну, кількість доступних одиниць на відповідному складі, та категорію.

FR4.2 У разі відсутності обов'язкових полів або введення некоректних даних система повинна видати повідомлення: «Будь ласка, заповніть всі обов'язкові поля коректно».

FR4.3 При успішному додаванні товару система повинна відобразити повідомлення: «Товар успішно додано».

FR4.4 Користувач повинен мати можливість вибрати товар зі списку для редагування.

FR4.5 Після вибору товару користувач може редагувати будь-які з полів: назву, код, штрих-код, опис, ціну, кількість доступних одиниць, та категорію.

FR4.6 У разі некоректного введення даних система повинна видати повідомлення: «Будь ласка, виправте помилки у формах».

FR4.7 При успішному редагуванні інформації система повинна відобразити повідомлення: «Інформацію про товар успішно оновлено».

FR4.8 Користувач повинен мати можливість вибрати товар зі списку для видалення.

FR4.8 Система повинна підтвердити видалення товару через діалогове вікно: «Ви впевнені, що хочете видалити цей товар?».

FR4.9 У разі успішного видалення товару система повинна відобразити повідомлення: «Товар успішно видалено».

FR4.10 Користувач може ввести назву товару для перевірки його наявності.

FR4.11 Система повинна відобразити такий товар або – повідомлення «Товар недоступний», залежно від результату перевірки.

FR5. Перегляд аналітики товарів

FR5.1 Система повинна відображати загальні дані про кількість товарів на складі, кількість проданих одиниць та їх загальну вартість за вибраний період.

FR5.2 Система повинна генерувати графіки або діаграми, що відображають динаміку продажів товарів за вибраний період.

FR5.3 Користувач може завантажити усі доступні товари у форматі JSON.

FR6. Моніторинг залишків товарів

FR6.1 Система повинна автоматично оновлювати дані про залишки товарів у режимі реального часу при кожній операції (додавання, видалення, передача), а також за графіком, кожні 30 хв.

FR6.2 Користувач отримує повідомлення про нестачу товару на складі у вигляді електронного листа.

FR6.3 Користувач отримує повідомлення про нестачу товару на складі у вигляді сповіщення на веб-застосунку.

FR6.4 Користувач повинен мати можливість замовляти товари у вигляді PDF форми, що надсилається йому на пошту

FR7. Створення нового замовлення

FR7.1 Користувач повинен мати можливість вибрати товари для замовлення зі списку доступних товарів.

FR7.2 Система повинна дозволяти користувачу вказати кількість кожного товару, яку потрібно замовити.

FR7.3 При спробі замовлення кількості товару, яка перевищує наявний запас, система повинна видати повідомлення: «Недостатня кількість товару на складі для замовлення».

FR8 Перегляд отриманих змін про товари

FR8.1 Користувач повинен мати можливість переглядати останні зміни про товари.

FR8.1 Користувач повинен бачити, що саме спричинило зміну у товарі, наприклад, співробітник оновив дані, система замовила товар або покупець купив товар.

3.4 Моделювання прецедентів

Діаграма прецедентів (use case diagram, створені у застосунку draw.io [4]) (рис. 3.5) відображає взаємодії користувачів із системою в контексті управління товарами, створення замовлень, та аналітики. Основні елементи діаграми:

Актори:

- незареєстрований користувач може зареєструватися та авторизуватися в системі. Під час реєстрації проходить верифікація електронної пошти.
- співробітник має доступ до таких функцій:
- адміністратор відповідає за управління командою, включаючи додавання та видалення учасників, призначення ролей.
- менеджер займається створенням замовлень для поповнення товарів або виконання клієнтських запитів.

- замовник (зовнішній користувач) може створювати замовлення на товари.

Основні прецеденти (Use Cases):

- реєстрація: користувач може зареєструватися в системі;
- верифікація пошти: підтвердження електронної пошти після реєстрації;
- авторизація: процес входу в систему після реєстрації;
- управління товарами (рис. 2.6);

- управління командою (рис. 2.7);
- перегляд аналітики (рис. 2.8);
- моніторинг залишків;
- створити замовлення для співробітника;
- створити замовлення для замовника;

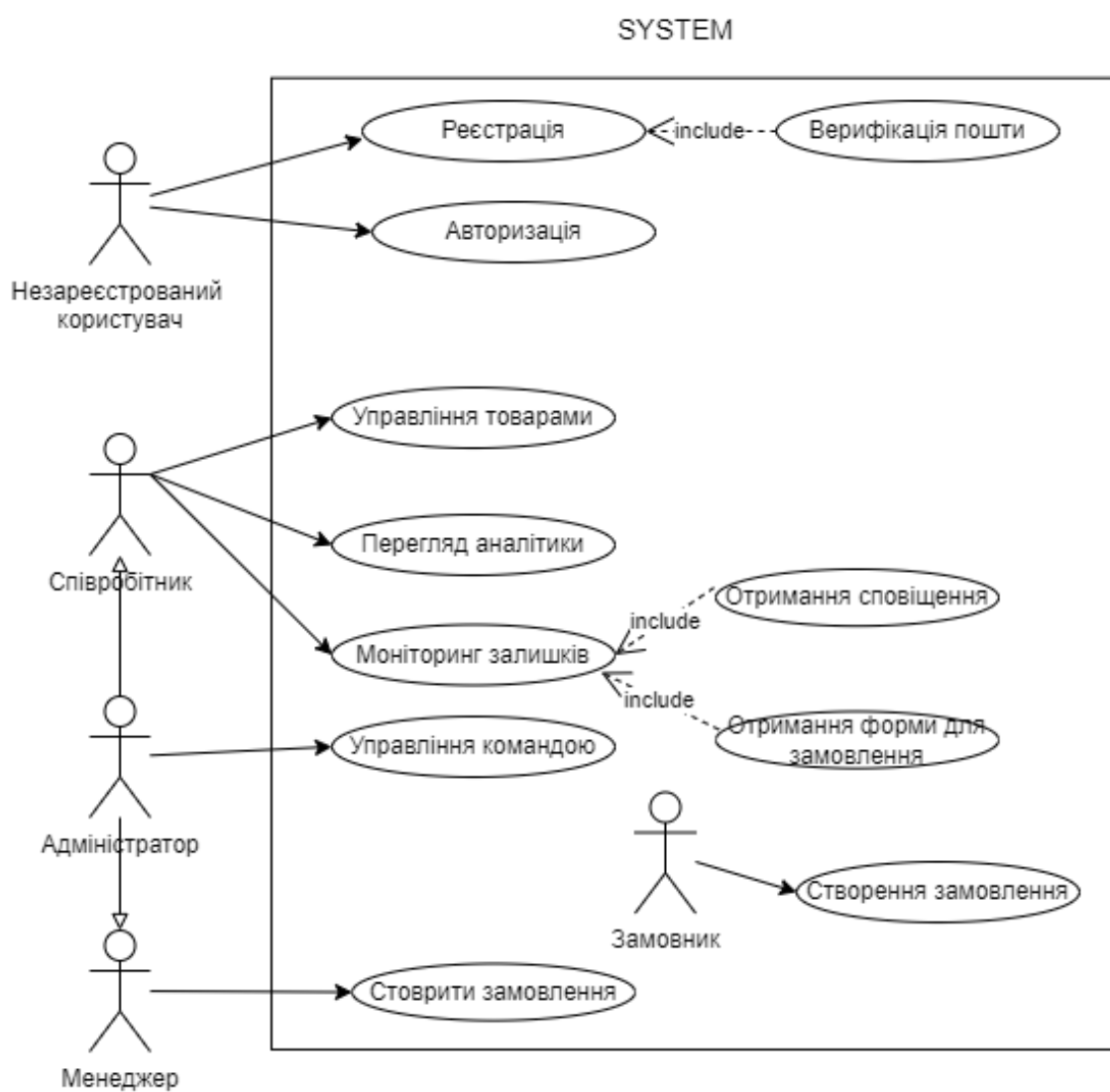


Рисунок 2.5 – Діаграма прецедентів



Рисунок 2.6 – Прецедент управління товарами



Рисунок 2.7 – Прецедент управління командою

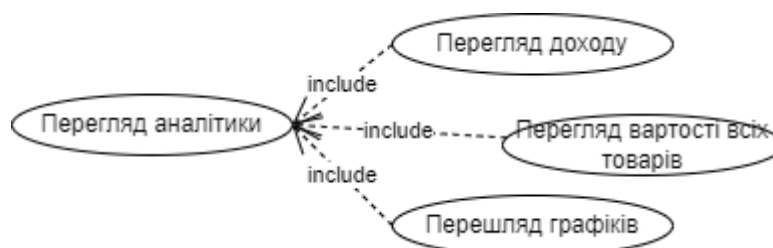


Рисунок 2.8 – Прецедент перегляд аналітики

3.5 Нефункціональні вимоги

NFR1. Вимоги платформи:

NFR1.1 Додаток повинен підтримувати основні веб-браузери:

- Google Chrome (версія не нижче 90);

- Mozilla Firefox (версія не нижче 85);
- Microsoft Edge (версія не нижче 90);
- Safari (версія не нижче 14).

NFR1.2 Додаток повинен коректно працювати на операційних системах Windows 10, macOS 11, і Linux (версія Ubuntu 20.04 і вище).

NFR2. Надійність:

NFR2.1 У разі збою підключення до бази даних система повинна автоматично спробувати повторне підключення протягом 5 секунд.

NFR3. Безпека

NFR3.1 Усі дані, що передаються між клієнтом і сервером, повинні бути зашифровані з використанням протоколу HTTPS.

NFR3.2 Система повинна використовувати алгоритми шифрування AES з довжиною ключа не менше 256 біт.

NFR1.3 Система шифрує конфіденційні дані такі, як пароль користувача.

NFR4. Продуктивність

NFR4.1 Максимальний час відповіді системи на запит користувача не повинен перевищувати ~3 секунди.

NFR4.2 Система повинна витримувати одночасно до 1000 активних користувачів.

NFR4.3 Додаток повинен підтримувати до 5000 відвідувань користувачів на добу.

NFR4.4 Система повинна обробляти до 200 транзакцій в секунду.

NFR5. Обсяг даних

NFR5.1 Система повинна підтримувати зберігання даних не менше ніж 100,000 користувачів.

NFR5.2 Максимальний обсяг даних, який система повинна обробляти, становить 1 ТБ.

NFR6. Доступність

NFR6.1 Система повинна бути доступною 24/7.

NFR6.2 Максимально допустиме час простою системи не повинно перевищувати 1 години на рік (99.99% доступності).

NFR6.3 Під час планового обслуговування система повинна сповіщати користувачів заздалегідь, не менш ніж за 24 години.

3.6 Architecture Constraints

Застосунок для ведення інвентаризації можна віднести до архітектурного патерну Мікросервісної архітектури WA (Web Application) архітепу.

1. Мова програмування: Java [5].

Джерело: Вимога замовника (дипломний керівник).

Опис: Основна мова програмування для розробки всіх компонентів додатку, забезпечує сумісність з існуючими рішеннями та вимагається для використання в проєкті.

2. Фреймворк: Spring Boo [6]t.

Джерело: Вимога замовника (дипломний керівник).

Опис: Вибір Spring Boot як основного фреймворку для розробки серверної частини додатку обумовлений вимогою до створення сучасного, кросплатформеного та масштабованого веб-застосунку.

3. Система управління базами даних: PostgreSQL [7].

Джерело: Інтеграція з існуючою інформаційною системою (IC).

Опис: Використання PostgreSQL як системи управління базами даних для зберігання структурованих даних додатку через наявність сумісності з існуючими IC, що також використовують PostgreSQL.

4. Система управління NoSQL базами даних: MongoDB [8].

Джерело: Інтеграція з IC та вимога замовника (дипломний керівник).

Опис: MongoDB використовується для зберігання неструктурованих даних та масштабованих об'єктів, що не підпадають під реляційну модель даних.

5. Система обміну повідомленнями: Apache Kafka [9].

Джерело: Вимога замовника (дипломний керівник).

Опис: Kafka використовується для передачі повідомлень між мікросервісами, забезпечуючи надійну та масштабовану систему обробки потоків даних у реальному часі.

6. Формат передачі даних: JSON.

Джерело: Вимога замовника (дипломний керівник).

Опис: Формат JSON обрано для обміну даними між клієнтською та серверною частинами додатку через його легкість та широке використання в сучасних веб-застосунках.

7. Інтерфейс взаємодії між сервісами: REST API.

Джерело: Інтеграція з існуючими ІС.

Опис: Взаємодія між мікросервісами та іншими системами відбувається через REST API, що забезпечує стандартизовану передачу даних та легкість інтеграції.

8. Протокол безпеки: JWT (JSON Web Tokens) [10].

Джерело: Вимога замовника (дипломний керівник).

Опис: Використання JWT для аутентифікації та авторизації користувачів. Цей протокол забезпечує безпечну передачу даних між клієнтом та сервером, що відповідає вимогам до безпеки додатку.

9. Фронтенд: React [11].

Джерело: Вимога замовника (дипломний керівник).

Опис: Вибір React для побудови клієнтської частини додатку обумовлений необхідністю створення динамічного, швидкого та інтерактивного інтерфейсу користувача.

10. UI бібліотека: ShadCN UI [12].

Джерело: Вимога замовника (дипломний керівник).

Опис: Для побудови інтерфейсу користувача використовується ShadCN UI, яка надає готові компоненти для швидкої розробки сучасних інтерфейсів.

11. Контейнеризація: Docker [13].

Джерело: Вимога інтеграції з існуючими ІС.

Опис: Використання Docker для контейнеризації компонентів додатку з метою спрощення деплоюменту та забезпечення кросплатформеності рішення.

3.7 UI View

Сторінка реєстрації дозволяє користувачу, ввівши валідні дані зареєструватися у системі (рис. 2.9).

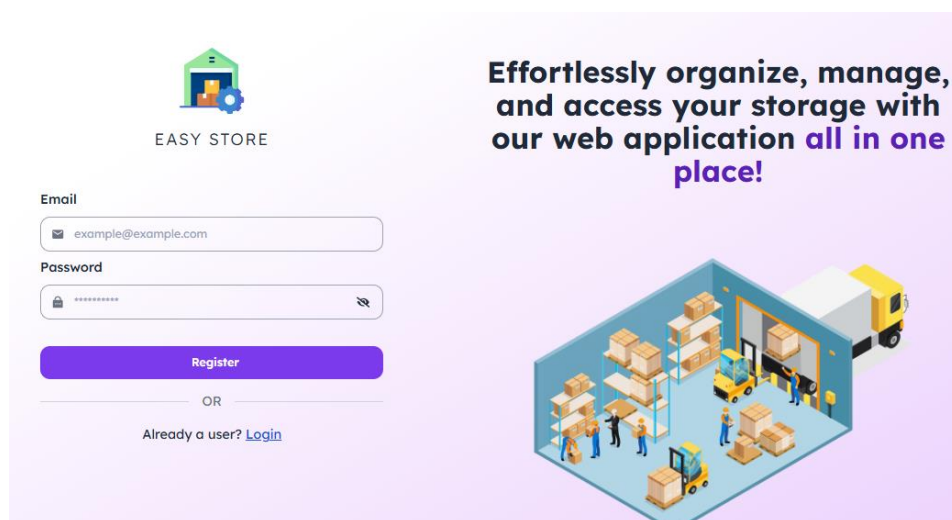


Рисунок 2.9 – Сторінка реєстрації

Сторінка авторизації дозволяє користувачу, ввівши валідні дані увійти у систему (рис. 2.10).

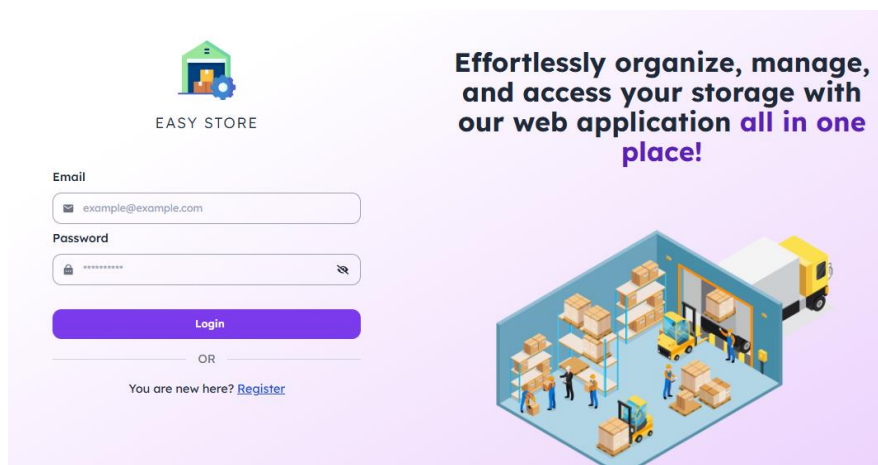


Рисунок 2.9 – Сторінка входу

Сторінка «Інформаційна панель» є центральним центром для користувачів, що забезпечує огляд ключової інформації та швидкий доступ до важливих дій (рис. 2.10). Ключові розділи:

- огляд системних показників;
- прибуток;
- вартість товарів на складах;
- останні переміщення операції над товарами;
- графки.
- загальна кількість товарів;
- кількість користувачів у команді.

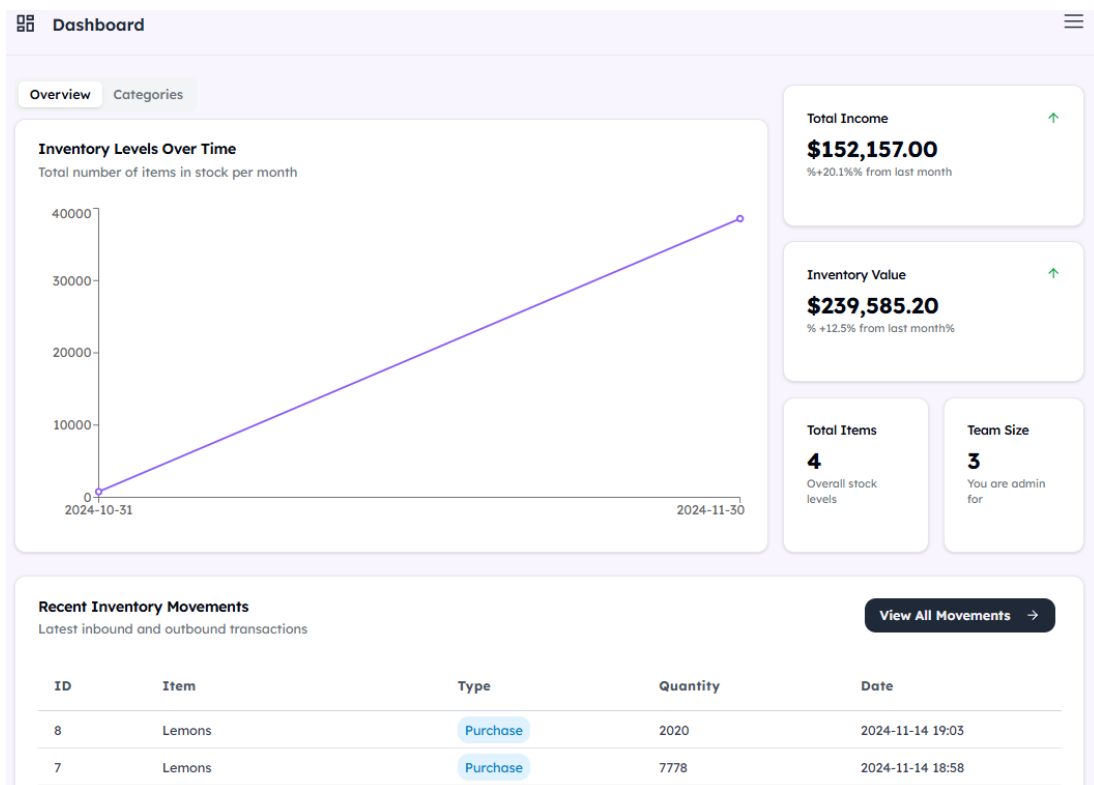


Рисунок 2.10 – Сторінка «Інформаційна панель»

На рисунку (рис. 2.11) показано сторінку керування командою зі зрозумілим і простим інтерфейсом користувача. Сторінка складається з трьох основних вкладок: «Створити команду» (рис. 2.12), «Запросити учасників» (рис. 2.13) і «Керувати командою» (рис. 2.14).

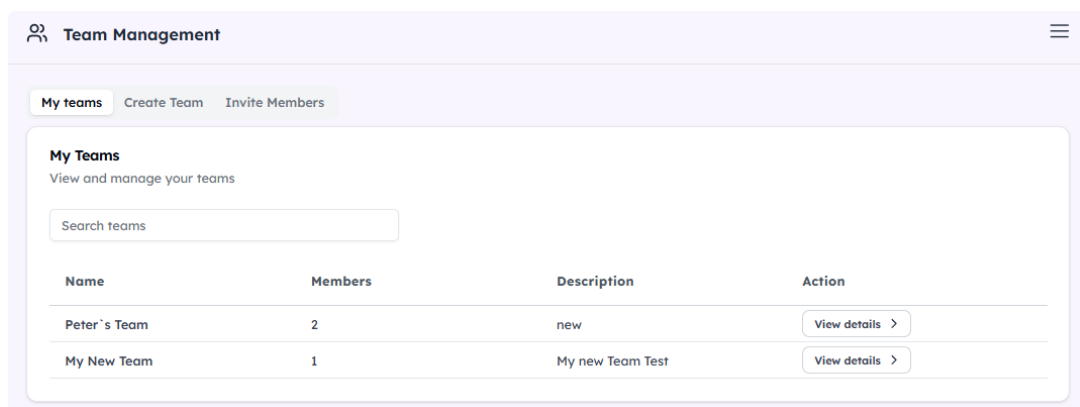


Рисунок 2.11 – Сторінка «Команда»

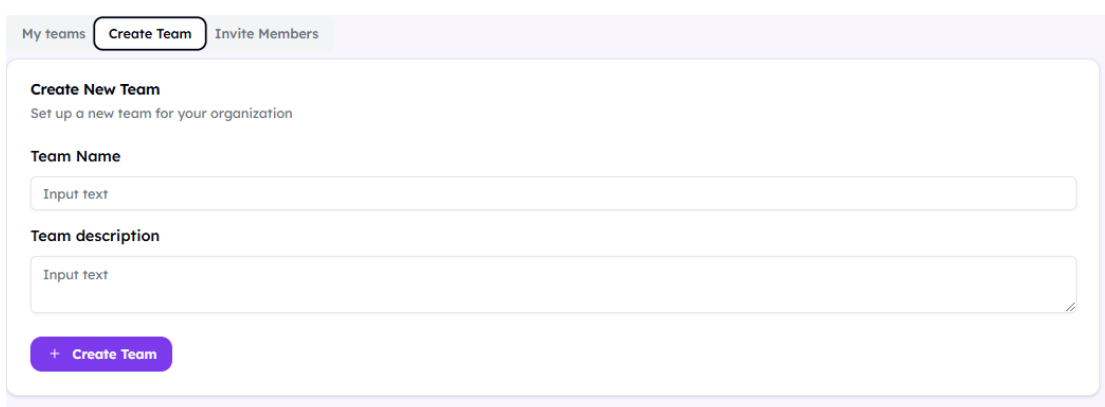


Рисунок 2.12 – Створити команду



Рисунок 2.13 – Запросити учасників

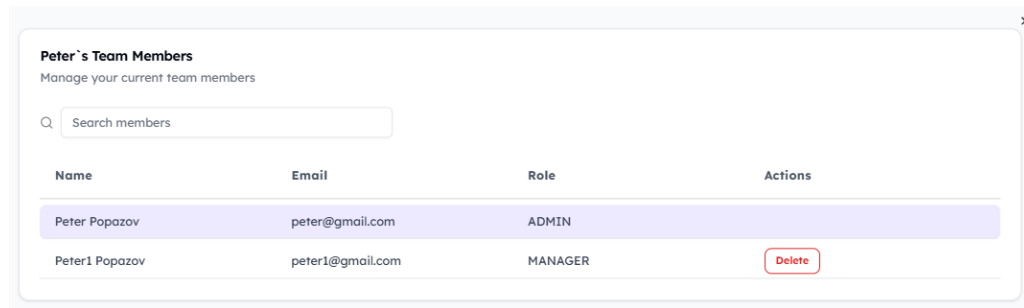


Рисунок 2.14 – Керувати командою

На зображенні (рис. 2.15) відображена сторінка Інвентаризації з чистим і організованим інтерфейсом користувача. Сторінка містить такі елементи:

Пошуковий рядок у верхній частині, де можна вводити назву товару для пошуку.

Таблиця з кількома колонками, які містять наступну інформацію:

- Product (Товар);
- Price (Ціна);
- Quantity (Кількість);
- Category (Категорія);
- Reorder Level (Рівень поповнення);
- і т.д.

У верхньому правому куті сторінки розташовані кнопки:

- Add Item (Додати товар) (рис. 2.16);
- Export (Експортувати): для експорту даних з таблиці.

Inventory						
<input type="text" value="Name"/> + Add Item 📄 Export						
SKU	Product	Price	Quantity	Category	Reorder Level	Provider
12345	Laptop	\$1200	10	Electronics	5	TechProvider Inc.
67890	Smartphone	\$800	50	Mobile Devices	20	MobileWorld
11223	Desk Chair	\$150	210	Furniture	50	FurniCo
44556	Wireless Headphones	\$300	705	Accessories	30	SoundGear
77889	Monitor	\$400	250	Electronics	10	DisplayTech
99001	Office Desk	\$600	400	Furniture	15	Workspace Solutions
PAGINATION						

Рисунок 2.15 – Сторінка «Інвентаризація»

Add new item [X]

Item name: Input text

Product SKU: Input text

Barcode: Input text

Price: Input text

Description: Input text

Warehouse: Select warehouse

Category: Select a category

Provider: Select a provider

+ Add Warehouse

+ Add Category

Warehouse stock: Input number

Total Stock: Warehouse stock

Min stock level: Input text

Max stock level: Input text

Add Discard

Рисунок 2.16 – Модальне вікно «Додати новий товар»

Для роботи із товаром створено спеціальну панель (рис. 2.17), яка містить кнопки «Змінити» - (рис. 2.18), «Видалити» - (рис. 2.19).

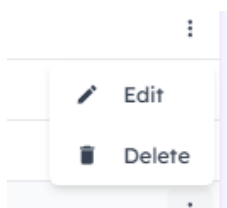


Рисунок 2.17 – Панель роботи з товаром

Edit item [X]

Item name: iPhone

Product SKU: 343423

Barcode: 34798

Price: 678

Description: iPhone 12 Pro 64Gb

Warehouse: W2

Category: Phones

Provider: Select a provider

+ Add Warehouse

+ Add Category

Warehouse stock: 20

Total Stock: 20

Min stock level: 1

Max stock level: 20

Edit Discard

Рисунок 2.18 – Модальне вікно «Змінити товар»

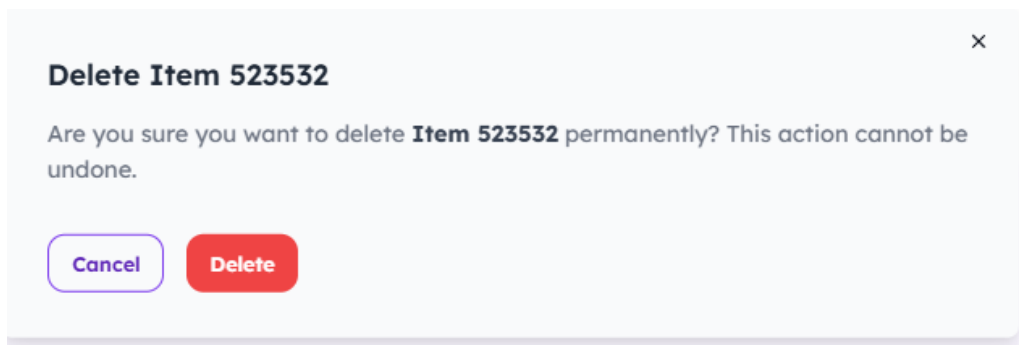


Рисунок 2.19 – Модальне вікно «Видалити товар»

Сторінка про рухи товарів створена для зручного відображення останніх транзакцій здійснені над товарами (рис. 2.20).

ID	Item	Type	Quantity	Date
8	Lemons	Purchase	2020	2024-11-14 19:03
7	Lemons	Purchase	7778	2024-11-14 18:58
6	Apples	Purchase	1013	2024-11-14 18:56
5	Apples	Sale	920	2024-11-12 16:31
4	Apples	Purchase	11220	2024-11-11 16:11
1	Apples	Sale	110	2024-11-11 12:05
2	Apples	Sale	220	2024-11-10 16:07
3	Oranges	Sale	1220	2024-10-10 16:11

Рисунок 2.20 – Сторінка про рухи товарів

Сторінка «Панель сповіщень» (рис. 2.21) служить для наступних цілей:

1. Відстеження низьких запасів: виділяє товари, які потрібно незабаром поповнити, щоб уникнути їх вичерпання.
2. Кнопку для отримання форми для замовлення товарів на ел. пошту.
3. Інформаційна панель надає приблизну загальну вартість поповнення запасів усіх товарів, які наразі позначені як низькі, що допомагає спланувати бюджет для поповнення.

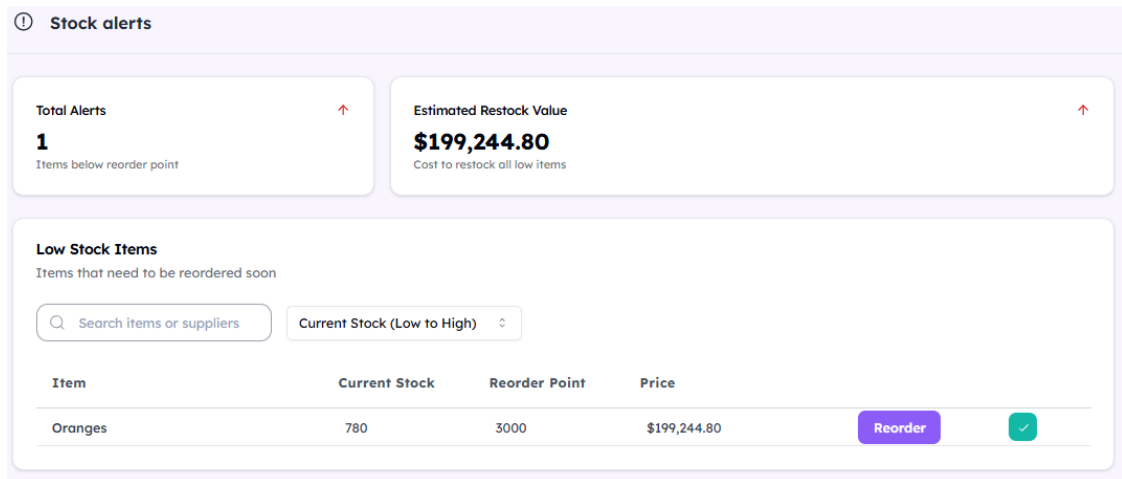


Рисунок 2.21 – Сторінка «Панель сповіщень»

3.8 Logical View

Зображено логічне представлення (рис. 2.22). Користувач взаємодіє з клієнтом (веб), який надсилає HTTP запити на API Gateway, який у свою чергу аутентифікує користувача та додає у заголовки HTTP запиту необхідну інформацію. API Gateway знає про всі інстанси мікросервісів та має load balancer, який здатен автоматично направляти запити до потрібного мікросервісу. Комунікація між мікросервісами реалізована у приватній мережі (клієнт не може надсилати запити на мікросервіс, тільки через API Gateway). Присутній Message Broker Apache Kafka та мікросервіси, які реєструють повідомлення та ті, що слухають. Наявні дві СУБД PostgreSQL – таблицна (RDBMS) та MongoDB – документноорієнтована.

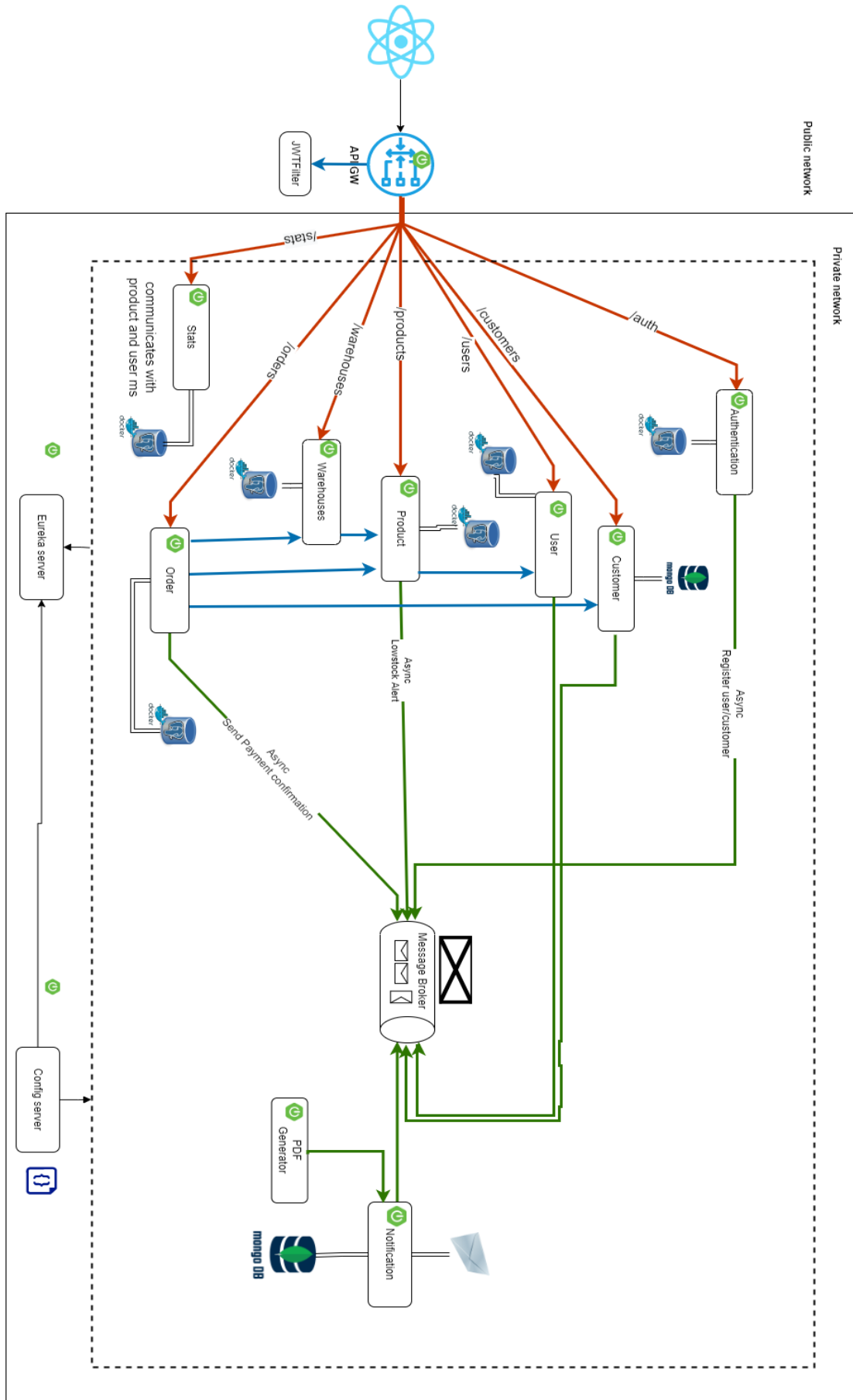


Рисунок 2.22 – Logical View

3.9 Deployment View

Кожен мікросервіс запускається на окремому сервері. Кожен мікросервіс використовує свою СУБД, що запускається в окремому контейнері. СУБД не були позначені на діаграмі для візуального спрощення, адже кожен мікросервіс використовує СУБД і загальна кількість контейнерів перевищувала 20 (рис. 2.23).

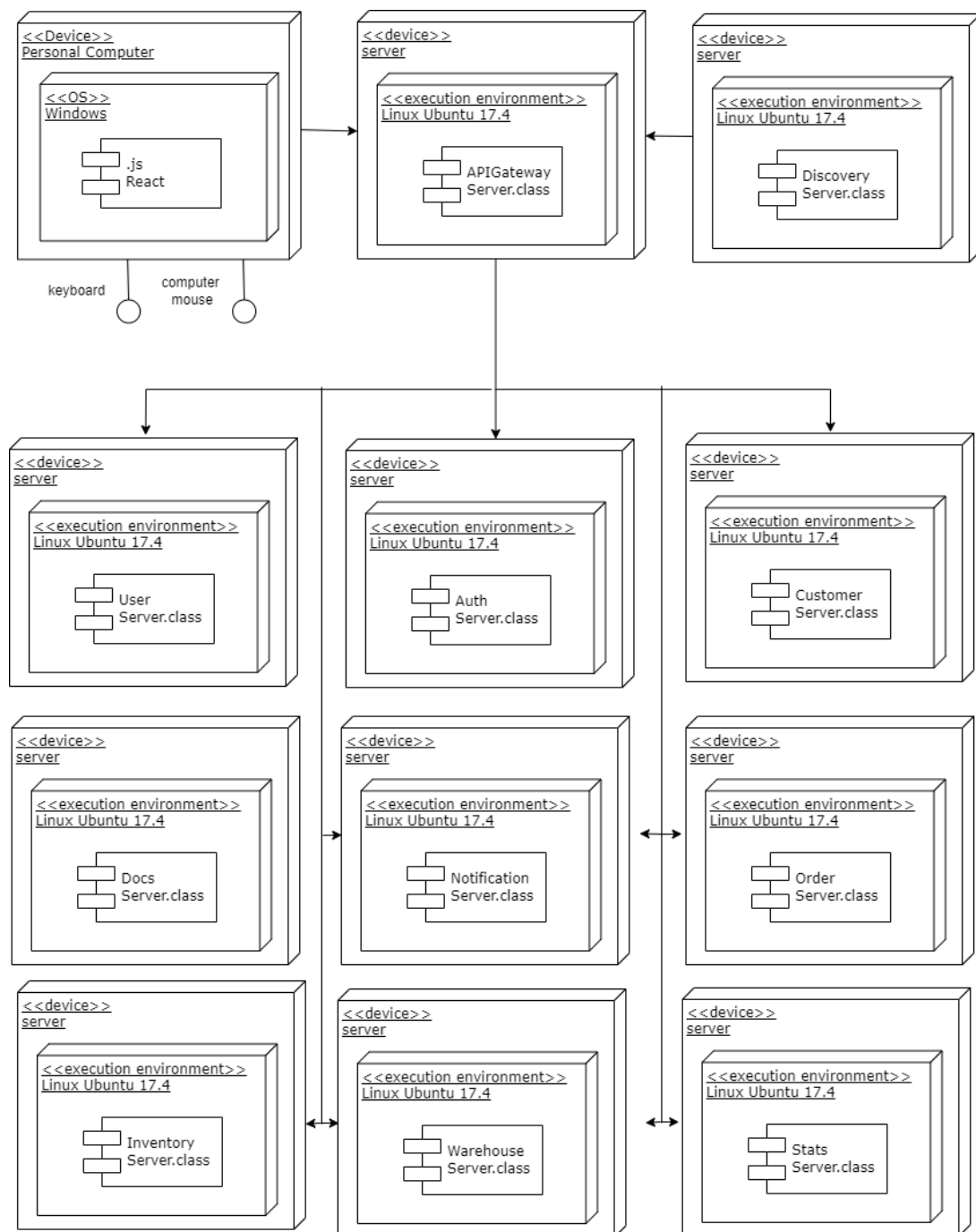


Рисунок 2.22 – Deployment View

3.10 Design View

UML діаграма Design View (рис. 2.23). Рівні пов'язані з логічним розподілом компонентів і функціональних можливостей і не враховують фізичне розташування компонентів, тоді як рівні описують фізичний розподіл функціональних можливостей і компонентів на окремих серверах, комп'ютерах, мережах або віддалених локації. Модель додатка показує кілька рівнів: рівень презентації, Presentation Layer, Business Layer, Data Layer.

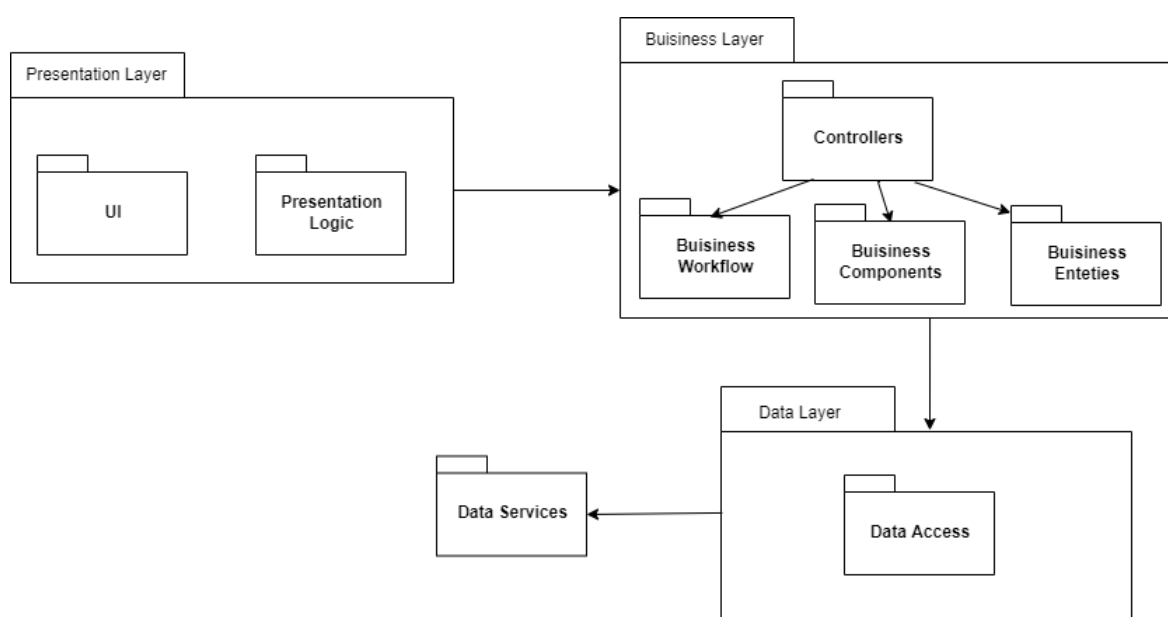


Рисунок 2.23 – Design View

3.11 Process View

Нижче наведено схему процесу додавання нового продукту до системи (рис. 2.24). Діаграма процесу візуально представляє послідовність дій і моментів прийняття рішень, які беруть участь у процесі додавання продукту. Діаграми створені у веб-ресурсі, що дозволяє за методологією «Docs as Codes» створювати діаграми, використовуючи код, замість графічного інтерфейсу, як от у застосунку draw.io – PlantUML [14].

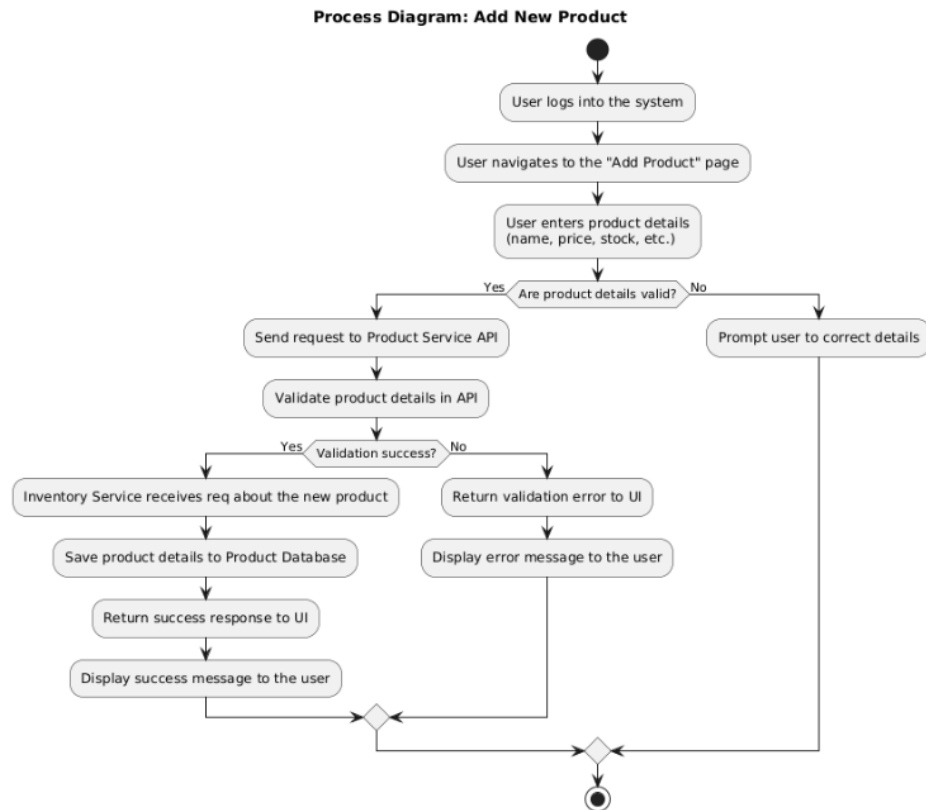


Рисунок 2.24 – Додавання нового продукту до системи

Нижче наведено діаграму процесу, яка представляє алгоритм для перевірки недостатнього запасу (рис. 2.25).

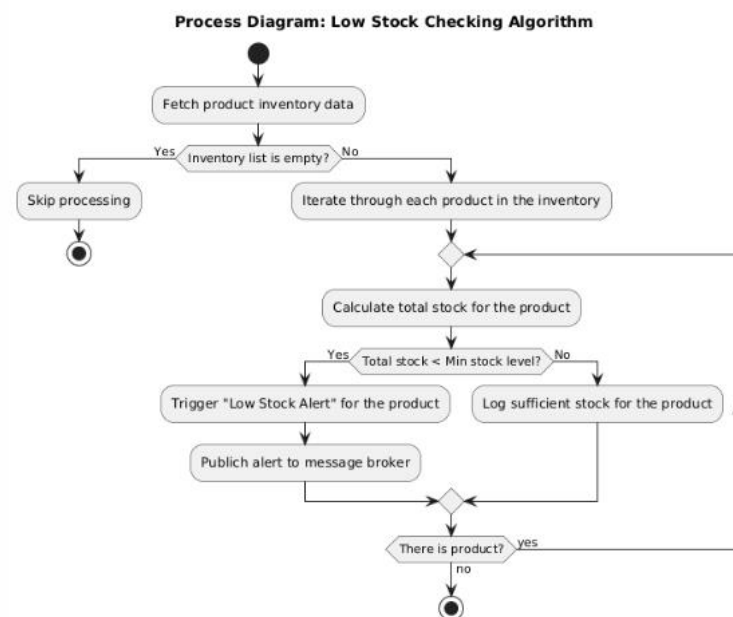


Рисунок 2.24 – Алгоритм для перевірки недостатнього запасу

Нижче наведено діаграму процесу, яка представляє процес додавання нового користувача до команди (рис. 2.26).

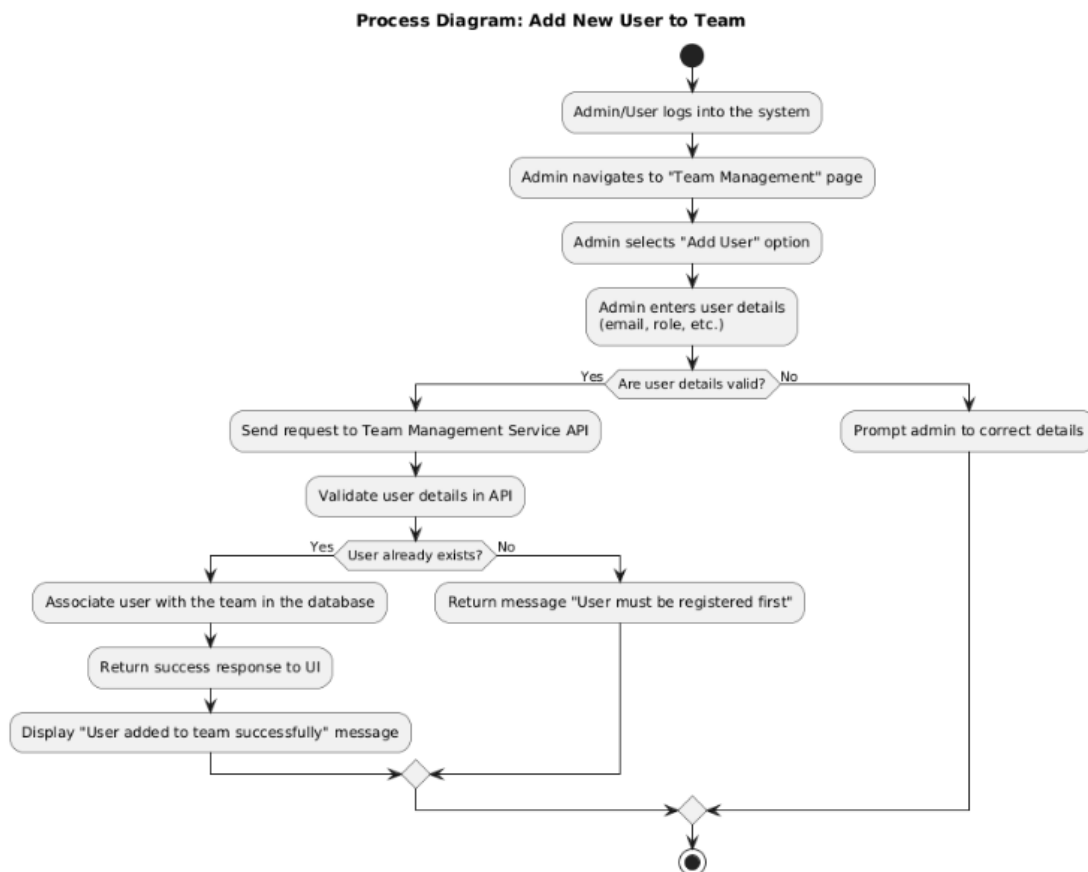


Рисунок 2.24 – Додавання нового користувача до команди

3.12 Data View

При плануванні розробки застосунка було обрано використовувати дві СУБД різного типу - SQL (PostgreSQL) та NoSQL (MongoDB). Розроблено ER-діаграму із зазначеним первинними та вторинними ключами у веб-застосунку lucidchart [15] (рис. 2.25). Так як архітектура застосунка передбачає використання мікросервісів, усю діаграму було розбито на домени за технікою (Domain Driven Development). Було створено індекси для полів, за якими найчастіше планується здійснювати пошук, а саме у таблиці User (email), Category (name).

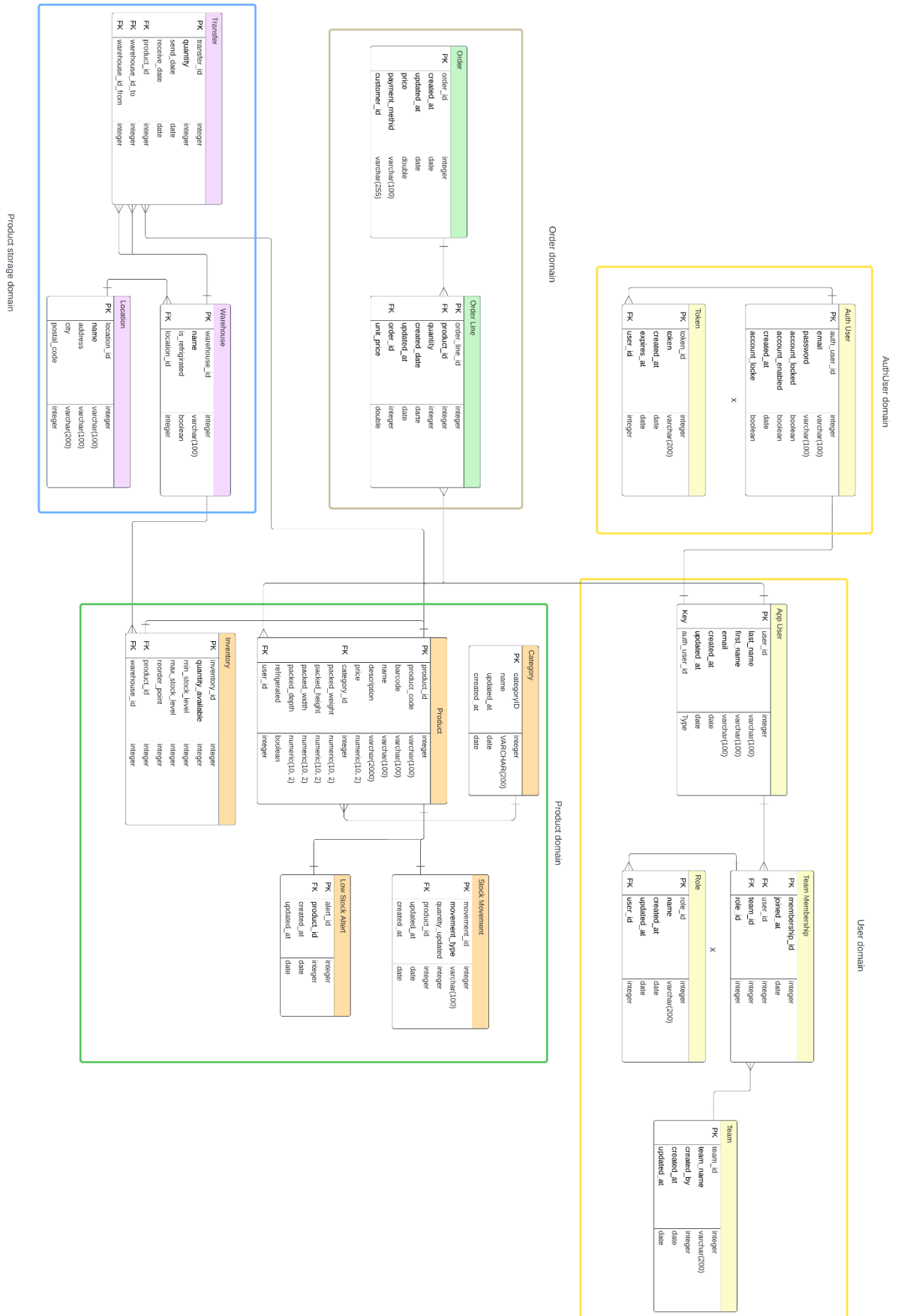


Рисунок 2.25 – ER-діаграма

Наведено діаграми для документоорієнтованої СУБД. Призначення — зберігання даних про надіслані електронні листи (рис. 2.26) зберігання даних про покупців (рис. 2.27).

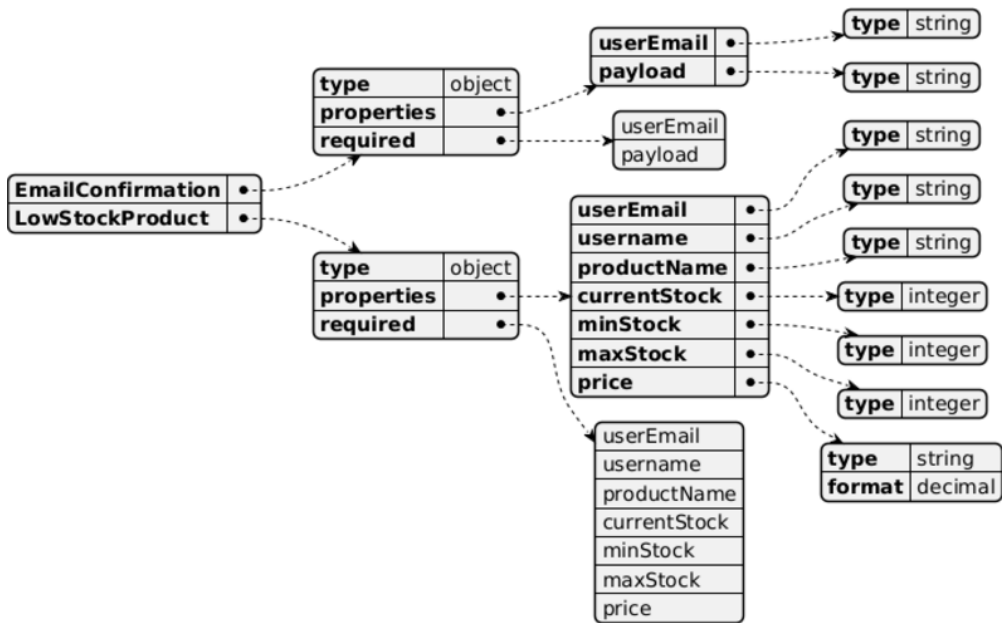


Рисунок 2.26 – Зберігання даних про надіслані електронні листи

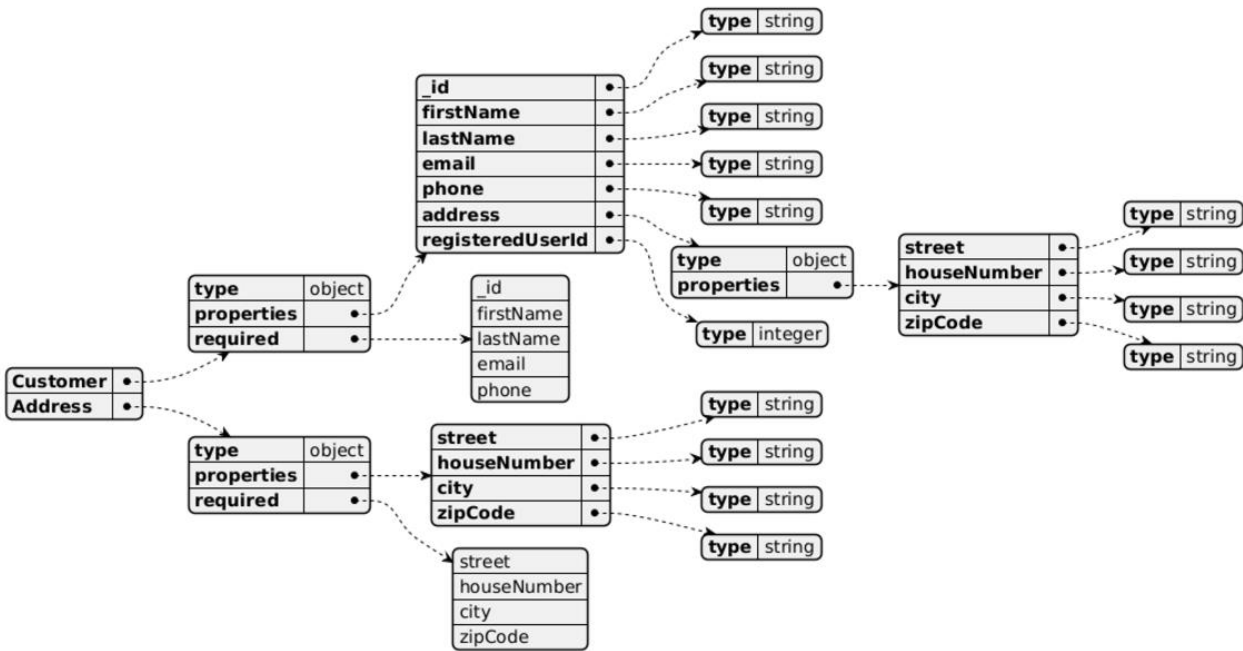


Рисунок 2.27 – Зберігання даних про покупців

3.13 Interface View

Система Smart Inventory Management System (рис. 2.28) розроблена з використанням архітектури мікросервісів із API Gateway, який діє як центральна точка входу для зв'язку між інтерфейсом і сервером. Frontend, створений як React SPA, надає інтерфейси для керування сповіщеннями про низькі запаси, керування продуктами та автентифікації користувачів (частина функціоналу). Зовнішні компоненти надсилають запити REST API до шлюзу API, який потім направляє ці запити до відповідних серверних служб. Для автентифікації шлюз API пересилає запити користувачів на вхід до служби автентифікації, де автентифікація JWT забезпечує безпечний доступ. Після автентифікації користувачі можуть взаємодіяти з іншими серверними службами, такими як Product Service для операцій CRUD над продуктами та Alert Service, щоб ініціювати та керувати сповіщеннями про низькі запаси.

Сервіс складається з різних мікросервісів Spring Boot, включаючи Product Service, який взаємодіє з таблицею Products у базі даних PostgreSQL для обробки даних продукту. Служба сповіщень обробляє сповіщення про низький запас, зберігаючи їх у таблиці сповіщень і використовуючи службу електронної пошти для надсилання сповіщень. Служба електронної пошти також взаємодіє зі службою PDF Generator Service, щоб додавати звіти до електронних листів і зберігати пов'язані з електронною поштою дані в MongoDB. Крім того, Служба продукту публікує сповіщення для брокера повідомлень Kafka, який прослуховує Служба сповіщень для обробки сповіщень у реальному часі. Ця система обміну повідомленнями допомагає роз'єднати служби та забезпечити асинхронну обробку. Усі серверні служби підключено до відповідних баз даних, таких як PostgreSQL для продуктів, сповіщень і даних автентифікації, що забезпечує надійну та масштабовану архітектуру. API Gateway спрощує взаємодію інтерфейсу з кількома мікросервісами, пропонуючи єдину точку доступу для маршрутизації запитів, одночасно вирішуючи такі проблеми, як автентифікація та балансування навантаження.

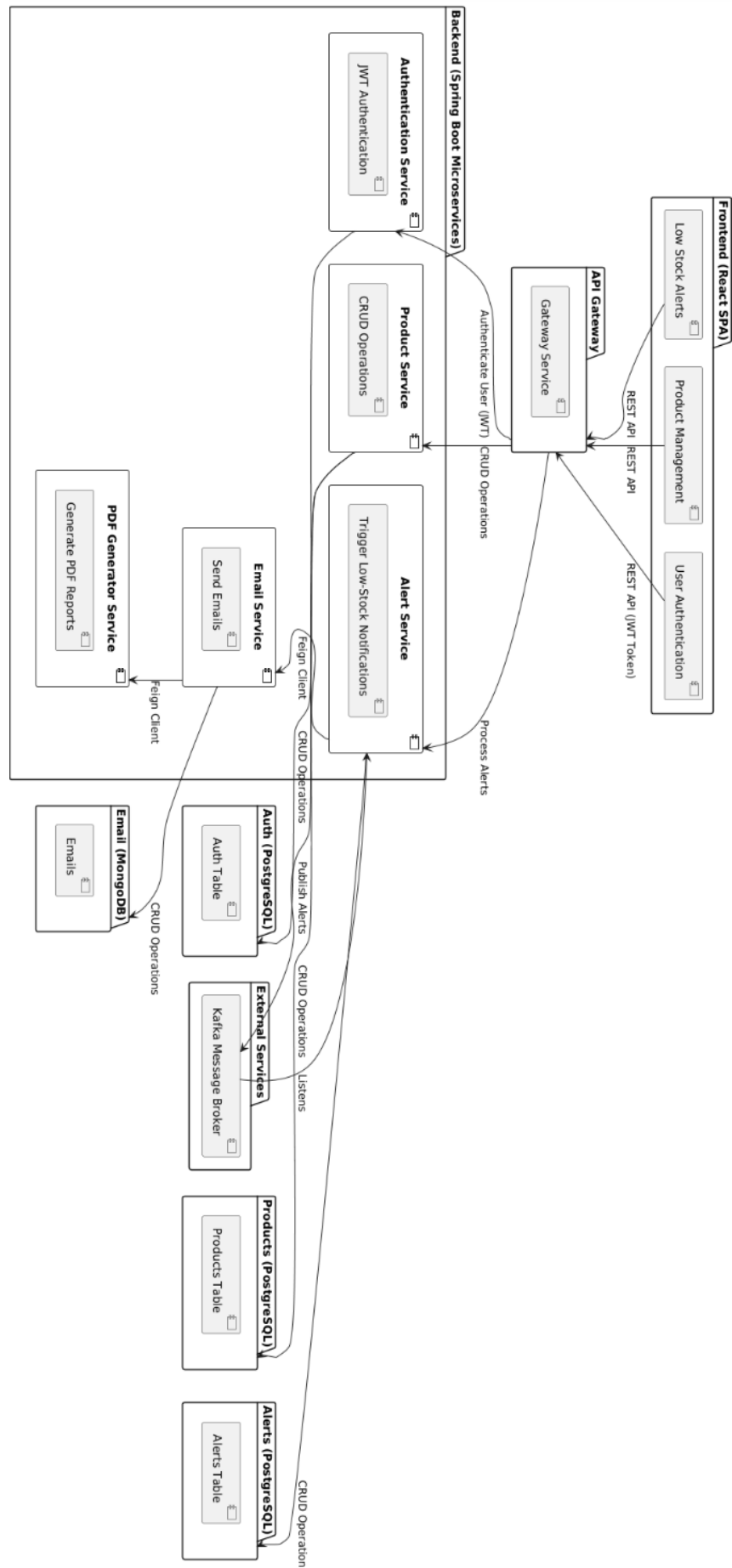


Рисунок 2.28 – Interface View

3.14 Security View

Для системи, побудованої на архітектурі мікросервісів з використанням Spring Boot, API Gateway та окремого мікросервісу для аутентифікації, використані наступні кроки забезпечення безпеки:

1. Аутентифікація та авторизація. Використовується аутентифікація за допомогою токенів JWT — JSON Web Token [16]. Клієнт передає токен у заголовок запиту HTTP (Authorization Header), який перевіряється на API Gateway. Для реалізації ролей та дозволів застосовується Spring Security з підтримкою механізмів авторизації на рівні API Gateway. Використання HTTPS (SSL/TLS) для захищеного передавання даних між клієнтом і сервером, а також між мікросервісами.

2. Захист чутливих даних у базі даних. Паролі користувачів зберігаються у зашифрованому вигляді, використовуючи алгоритм BCrypt

3. Захищені з'єднання. Всі канали зв'язку між клієнтом і сервером, а також між мікросервісами, повинні бути зашифровані за допомогою SSL/TLS для захисту даних від перехоплення та підміни. API Gateway та інші мікросервіси забезпечують підтримку TLS 1.2 або 1.3 для безпечного з'єднання.

4. Запобігання ін'єкціям. Використання параметризованих запитів (prepared statements) та ORM-бібліотек, таких як JPA або Hibernate [17], для захисту від SQL-ін'єкцій. Використання механізмів захисту від CSRF, таких як токени CSRF (вони повинні передаватися у кожному запиті від клієнта і перевірятися сервером). Spring Security підтримує захист від CSRF за замовчуванням.

3.15 Infrastructure View

Infrastructure View (Інфраструктурне представлення) — це архітектурне представлення інформаційної системи, яке описує її фізичну та програмну інфраструктуру (табл. 2.1, табл. 2.2). Його основна мета — деталізувати технічне середовище, в якому працює система, щоб забезпечити її ефективність, надійність та масштабованість.

Таблиця 2.1 – Application Infrastructure View

Сервер додатків	База даних (PostgreSQL):	AWS (Amazon Web Services)[17]	ПЗ
Оперативна пам'ять: 16 ГБ ЦП: 8 ядер (3,0 ГГц) GPU: відсутній	Оперативна пам'ять: 32 ГБ ЦП: 12 ядер (2,8 ГГц) Хранилище: SSD 1 ТБ	EC2 для серверів програм, RDS для бази даних	Java 17 (Spring Boot 3.3) Kafka 3.0

Таблиця 2.2 – Users Infrastructure View

PCs	Mobiles	Веб-браузери
Оперативна пам'ять: мінімум 4 ГБ ЦП: 2 ядра (2.0 ГГц) GPU: базовий рівень, інтегрована графіка	Оперативна пам'ять: мінімум 2 ГБ ОС: Android 8.0+ або iOS 13+ Підключення: 4G або Wi-Fi (мінімум 10 Мбіт/с)	Google Chrome 90+, Firefox 88+, Safari 14+

Мережа:

- пропускна здатність: 1 Гбіт/с;
- зовнішнє підключення до Інтернету: мінімум 100 Мбіт/с.

3.16 Delivery Strategy View

Система побудована на мікросервісній архітектурі, де кожен сервіс (управління продуктами, складами, сповіщеннями тощо) контейнеризовано через Docker. Для доступу користувачів використовуються React SPA та REST API. Комунікація між сервісами реалізована через Kafka (асинхронно) та API (синхронно). Захист даних забезпечується JWT-токенами, SSL/TLS та ролями (RBAC). Масштабованість досягається завдяки горизонтальному масштабуванню сервісів і Load Balancer.

4 РЕАЛІЗАЦІЯ ВЕБ САЙТУ

4.1 API Gateway

Для зберігання програмного коду, було обрано веб-сервіс GitHub [21], було створено моно репозиторій для зберігання як клієнтської, так і серверної частини веб-додатка [22].

Діаграма (рис. 4.1) представляє архітектуру програми API Gateway, яка виконує втентифікацію, перевірку запитів та перенапряння їх на відповідний мікросервіс. Система розроблена для обробки та перевірки запитів через AuthFilter, забезпечуючи як перевірку маршруту, так і автентифікацію на основі токенів. AuthFilter покладається на RouteValidator, щоб підтвердити, чи дозволений запитаний шлях, і він інтегрується з JwtService для обробки операцій JSON Web Token (JWT), таких як перевірка токенів, витягування claims і підписання ключів для автентифікації.

Обробкою помилок керує GlobalExceptionHandler, який централізує обробку різних винятків, включаючи помилки автентифікації (BadCredentialsException) і прострочені JWT (ExpiredJwtException). Коли виникає виключення, GlobalExceptionHandler взаємодіє з ErrorResponseBuilder, щоб створити послідовну та детальну відповідь на помилку. Конструктор дозволяє об'єднувати численні повідомлення про помилки, описи та специфічні для бізнесу деталі в структурований ErrorResponse. Модульний підхід гарантує, що помилки не тільки обробляються, але й ефективно повідомляються користувачам або іншим службам.

У класах Config та AppConfig забезпечують необхідні залежності, наприклад, RestTemplate для зв'язку HTTP. Діаграма ілюструє чіткий розподіл завдань, де кожен клас має чітко визначену роль, наприклад перевірку запитів (RouteValidator), керування токенами (JwtService) і створення відповіді на помилки (ErrorResponseBuilder).

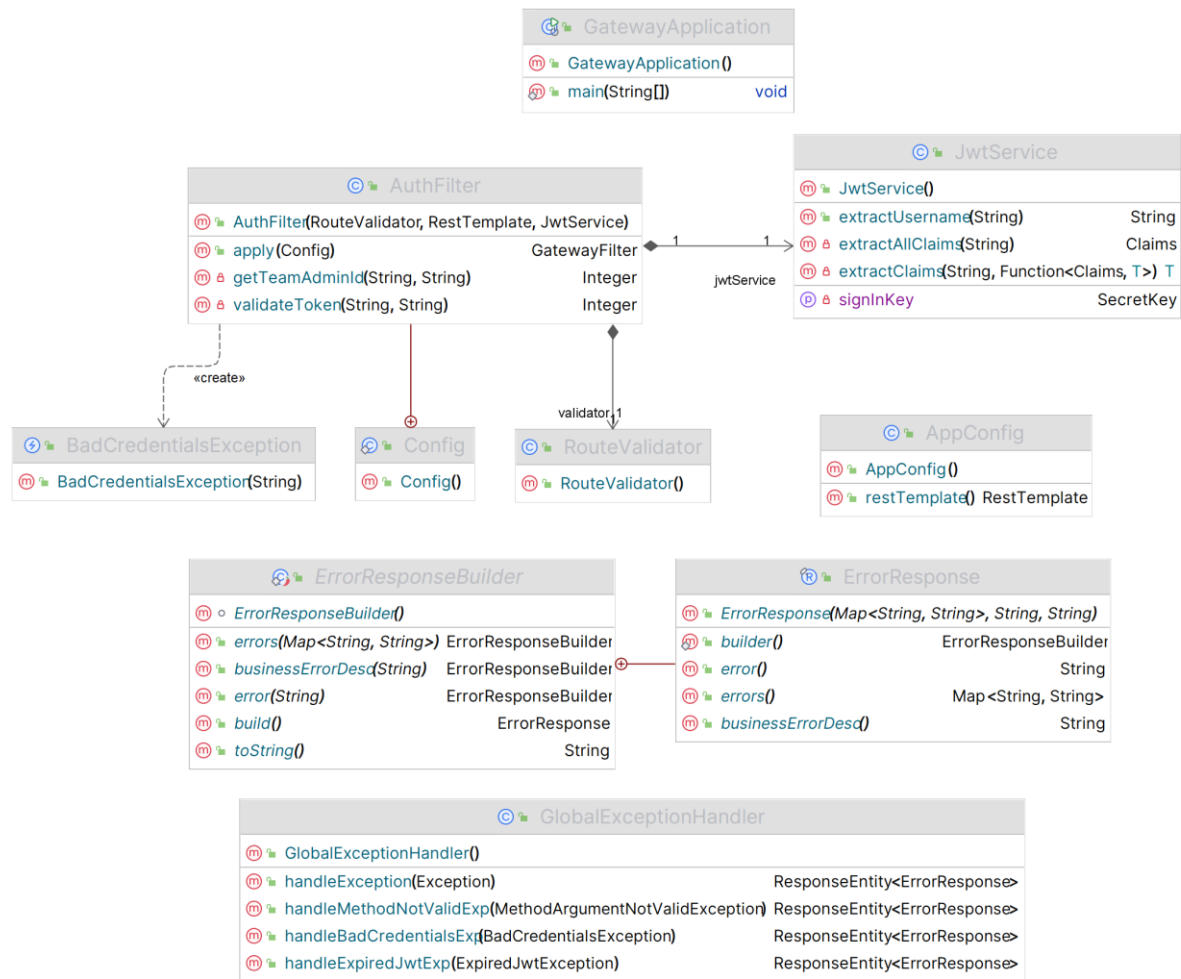


Рисунок 4.1 – Діаграма класів

Клас `AuthFilter` є частиною фільтрів `Spring Cloud Gateway` і використовує анотацію `@Component` для автоматичного реєстрування фільтра в контексті `Spring`. Основні методи класу:

1. `apply(Config config)` – основний метод, який виконується при обробці запиту. Використовує валідатор `RouteValidator`, щоб визначити, чи є запит захищеним (потрібна авторизація). Перевіряє наявність заголовка `Authorization` і витягує токен `Bearer`. Якщо токен дійсний, додаються додаткові заголовки (`userId`, `teamAdminId`) до запиту. Після цього передає запит наступному фільтру в ланцюжку.

2. `validateToken(String token, String email)` – виконує GET-запит до зовнішнього сервісу для перевірки дійсності токена. Повертає ідентифікатор користувача (`userId`), якщо токен є дійсним.

3. `getTeamAdminId(String token, String userId)` – виконує GET-запит до сервісу для отримання ідентифікатора адміністратора команди для поточного користувача. Повертає `teamAdminId`, якщо є.

Клас `RouteValidator` є допоміжним компонентом для визначення, чи є запит захищеним (потрібна автентифікація). Основні компоненти:

1. `endpoints` – це статичний список шляхів API, які не потребують автентифікації (тобто відкриті для всіх користувачів). Наприклад, `"/api/v1/auth/register"`, `"/api/v1/auth/authenticate"`.

2. `isSecured` – це предикат (умова), який перевіряє, чи є запит захищеним, виконується для кожного HTTP-запиту і визначає, чи потрібно застосовувати автентифікацію. Якщо шлях запиту не містить один із шляхів з `endpoints`, то це вважається, що запит є захищеним, і йому потрібно проходити перевірку на авторизацію. Використовується метод `noneMatch`, щоб перевірити, чи жоден із зазначених шляхів не є частиною шляху запиту.

Коли приходить HTTP-запит, метод `isSecured` перевіряє, чи містить шлях запиту один із відкритих шляхів. Якщо такий шлях знайдений, запит вважається відкритим і не потребує автентифікації. Якщо жоден з шляхів не збігається, це означає, що запит потребує автентифікації (потрібен токен), і фільтр `AuthFilter` має його перевірити.

Клас `JwtService` відповідний за витягування та перевірки інформації з JWT токenu, зокрема, для автентифікації користувачів:

1. `extractUsername(String token)` – отримує ім'я користувача (`subject`) з JWT токenu, викликаючи метод `extractClaims` для отримання інформації з токenu.

2. `extractClaims(String token, Function<Claims, T> claimsResolver)` – загальний метод, що використовує функцію `claimsResolver` для витягування конкретних даних з `Claims` токenu. Повертає значення, яке визначається функцією `claimsResolver` (наприклад, ім'я користувача).

3. `extractAllClaims(String token)` – витягує всі Claims з JWT токена, використовуючи бібліотеку `Jwts` для перевірки підпису і парсингу токена. Повертає об'єкт `Claims`, який містить інформацію з токена.

4. `getSignInKey()` – генерує секретний ключ для підпису токена за допомогою значення `secretKey`, яке зчитується з конфігурації. Повертає об'єкт `SecretKey` для використання при перевірці підпису токена.

Конфігурація для `Spring Cloud Gateway`, яка задає параметри маршрутизації та фільтри для різних мікросервісів знаходиться на окремому мікросервісі. Опис головних налаштувань:

1. `globalcors` – налаштування `CORS` (`Cross-Origin Resource Sharing`) для глобальних маршрутів.

2. `discovery.locator.enabled` – включає автоматичне виявлення маршрутів через `Spring Cloud Service Discovery` (наприклад, `Eureka`), що дозволяє з'єднуватися з сервісами за їх іменами.

4. `routes` – список маршрутів, який визначає, як з'єднуватися з різними сервісами. Кожен маршрут має:

- `id` – унікальний ідентифікатор маршруту;
- `uri` – URI сервісу, наприклад, `lb://SERVICE_NAME` (використовується балансування навантаження);
- `predicates` – умови для маршруту (наприклад, шлях для конкретного запиту);
- `filters` – фільтри, що застосовуються до запитів, тут використовується фільтр `AuthFilter` для аутентифікації.

Мікросервіси, на які перенаправляються запити:

1. `app-user` — відповідає за управління користувачами, включаючи їх реєстрацію, оновлення даних і обробку ролей;

2. `auth-service` — надає функції аутентифікації та авторизації користувачів, включаючи валідацію токенів;

3. `customer` — опрацьовує інформацію про клієнтів, зберігаючи їхні дані й взаємодіючи з іншими сервісами;

4. `notification` — обробляє та надсилає повідомлення (наприклад, email або push-сповіщення);

5. `order` — відповідає за управління замовленнями, включаючи їх створення, обробку та оновлення;

6. `product` — забезпечує функціонал для роботи з продуктами, їх описами, категоріями й доступністю;

7. `stats` — агрегує та надає статистичні дані, необхідні для аналізу або звітності;

8. `warehouse (warehouse-service)` — займається управлінням складськими приміщеннями, зокрема переміщенням і зберіганням товарів.

Інші мікросервіси:

1. `config-server` — централізовано керує конфігурацією для всіх мікросервісів;

2. `discovery` — відповідає за сервісну реєстрацію і знаходження, забезпечуючи динамічний зв'язок між мікросервісами;

3. `document-generator-server` — генерує документи, імовірно для звітів чи інших завдань, пов'язаних із системою;

4. `gateway` — виступає як точка входу до системи, маршрутизує запити до відповідних мікросервісів.

4.2 Inventory Service

Діаграма класів (рис. 4.2, рис. 4.3) представляє архітектуру системи, призначеної для керування категоріями, продуктами та запасами, містить контролери, сервіси та сховища, показуючи, як ці компоненти взаємодіють у програмі. Архітектура дотримується стандартного шаблону Model-View-Service-Repository, забезпечуючи поділ завдань і модульність.

1. `CategoryController` і `CategoryService`:

- `createCategory` дозволяє створювати нову категорію шляхом прийняття об'єкта `CreateCategoryRequest`. Сервісний рівень обробляє логіку та перевірку цієї операції;
- `updateCategory` оновлює деталі існуючої категорії. Це вимагає перевірки, щоб переконатися, що категорія існує;
- `deleteCategory` видаляє категорію за її ідентифікатором, гарантуючи, що пов'язані продукти не залежать від категорії;
- `getAllCategories` отримує список усіх категорій. Сервісний рівень отримує дані з `CategoryRepository`;
- `getCategoryByName` отримує категорію на основі її назви, що забезпечує легкий доступ і фільтрацію;

2. ProductController і ProductService

- `createProduct`: керує створенням продукту з використанням деталей із `ProductRequest`. Служба обробляє бізнес-правила, такі як ініціалізація запасів і асоціація категорій;
- `updateProduct` оновлює деталі певного продукту, наприклад назву, опис, ціну або категорію;
- `deleteProduct` видаляє продукт на основі його унікального ідентифікатора. Служба забезпечує відповідне оновлення будь-яких записів інвентаризації;
- `getAllProducts` отримує всі продукти в системі для загального або адміністративного використання;

3. InventoryController і InventoryService

- `updateInventory` оновлює деталі інвентарю для продукту, приймаючи `productId` і нові дані про інвентар. Це забезпечує точність рівня запасів на складах;
- `getInventoryForProduct` отримує деталі запасів для даного продукту, надаючи інформацію про наявність і рівень запасів;

4. `WarehouseClient` – `getWarehouseById`: інтегрується з `Warehouse Microservice`, щоб отримати деталі конкретного складу, які використовуються під час оновлення інвентаризації або призначення місць розташування запасів.

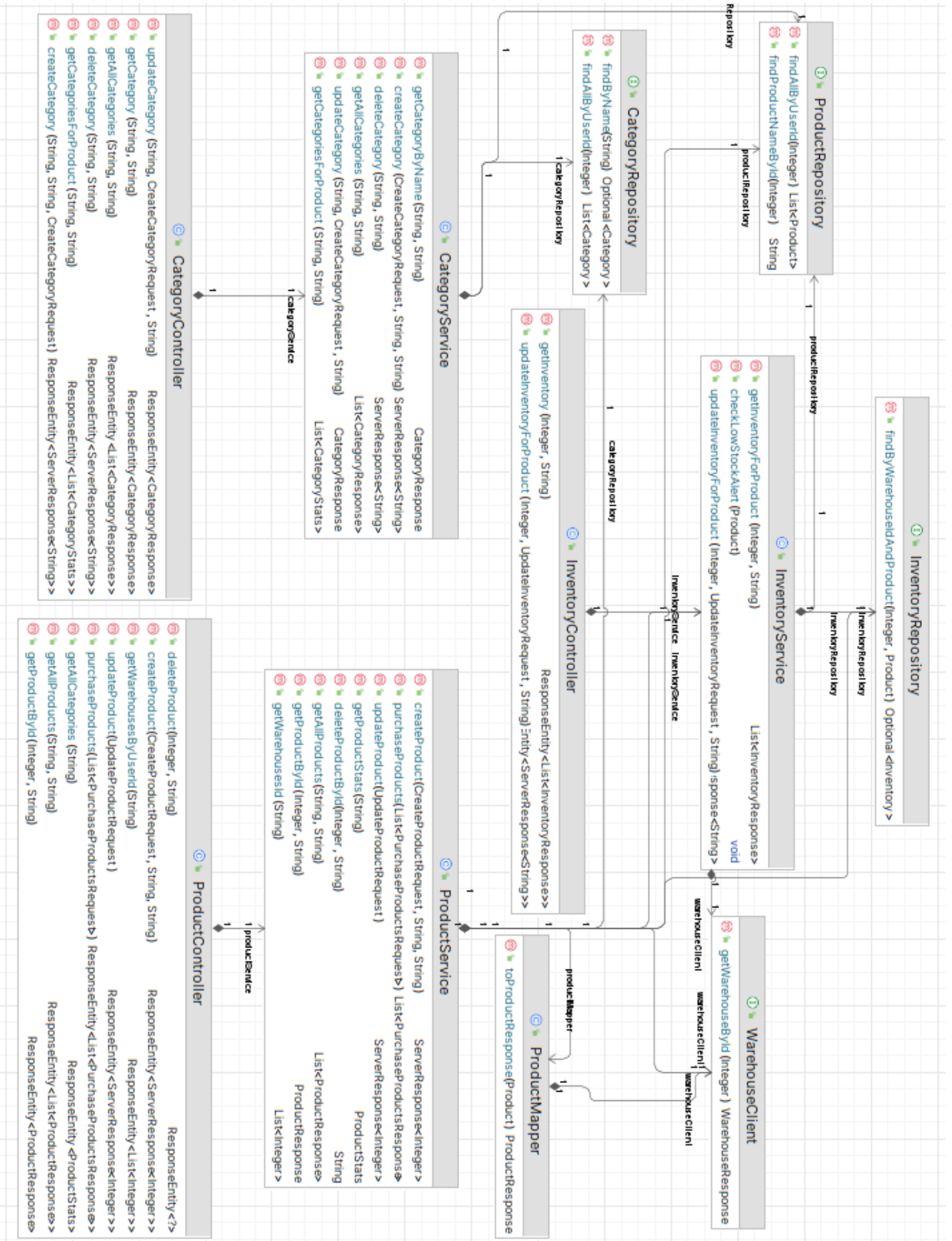


Рисунок 4.2 – Діаграма класів

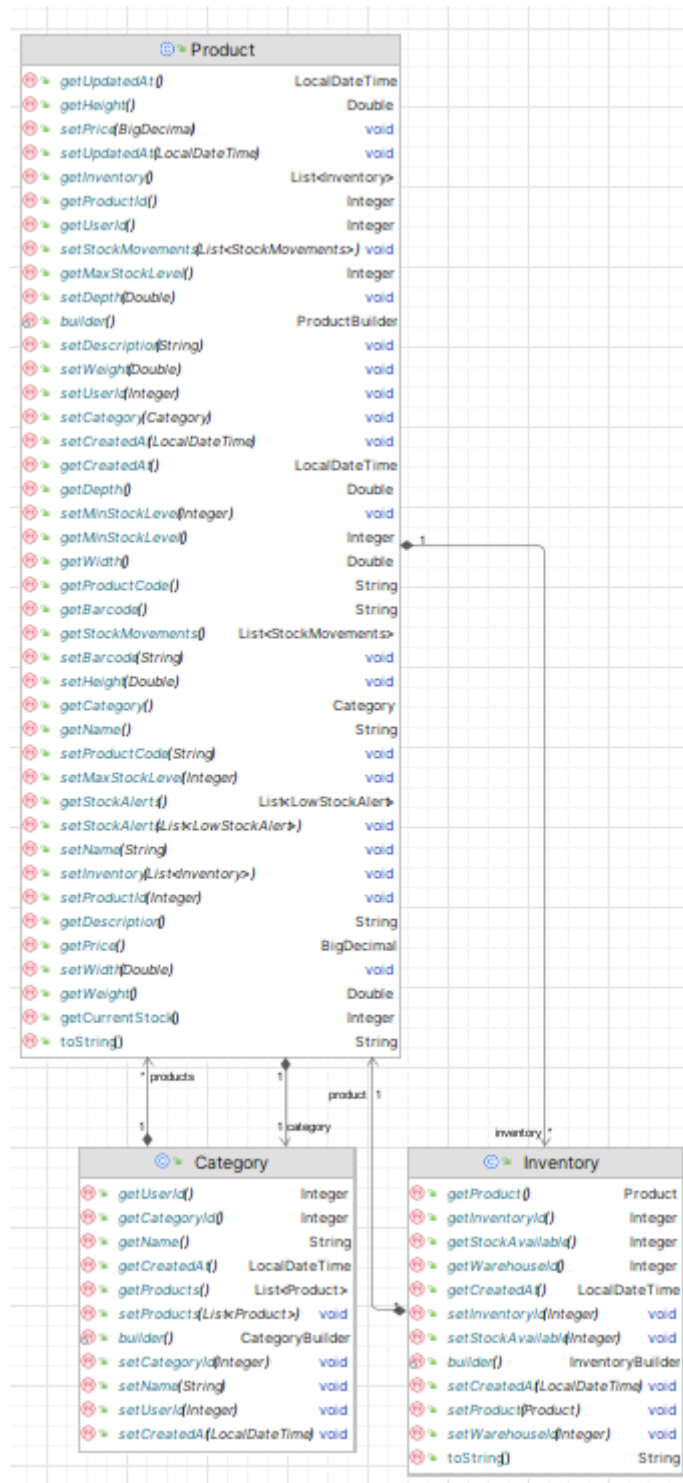


Рисунок 4.3 – Даграма класів

Діаграма класів (рис. 4.4) представляє систему, орієнтовану на управління рухом запасів, сповіщення про низькі запаси та уточнення запасів. Він відповідає сервісно-орієнтованій архітектурі, де контролери обробляють вхідні запити, служби реалізують бізнес-логіку, а репозиторії керують збереженням даних.

Діаграма ілюструє тісно інтегровану систему, спрямовану на відстеження руху запасів, виявлення ситуації з низькими запасами та сповіщення відповідних зацікавлених сторін.

StockMovementsController обробляє HTTP-запити для операцій руху товарів, як-от отримання, додавання або аналіз рухів. StockMovementsService містить бізнес-логіку для керування рухом запасів, покладаючись на репозиторії (StockMovementRepository, InventoryRepository) і мапер (StockMapper) для обробки та отримання даних про запаси. StockMovementRepository безпосередньо взаємодіє з базою даних, щоб отримати або зберегти записи про рух акцій.

LowStockAlertController керує запитом сповіщень про низькі запаси та надає відповіді з деталями сповіщень. LowStockAlertService реалізує основну логіку для створення, об'єднання та обробки сповіщень про низькі запаси, залежить від StockMovementsService, LowStockAlertRepository та RefillmentService. LowStockAlertRepository: взаємодіє з базою даних для зберігання та отримання даних про низькі запаси.

RefinementService обробляє логіку для уточнення запасів і сповіщень електронною поштою. Взаємодіє з:

- EmailRefinementProducer надсилає сповіщення електронною поштою (наприклад, про низькі запаси чи зміни в інвентарі) за допомогою інтеграції Kafka;
- UserClient отримує інформацію про користувача, як-от інформацію про адміністратора або команду, щоб визначити одержувачів сповіщень;

LowStockScheduler періодично перевіряє стан запасів за допомогою InventoryService та ініціює сповіщення.

Система відстежує рух запасів, оцінює ситуацію з нестачею запасів і надсилає автоматичні сповіщення, щоб забезпечити підтримку рівня запасів. Запити на дані запасів або сповіщень надходять від контролерів до служб і сховищ. Завдання сповіщень, наприклад надсилання електронних листів, обробляються асинхронно за допомогою виробників Kafka, таких як EmailRefinementProducer.



Рисунок 4.4 – Даграма класів

4.3 User Service

Діаграма класів (рис. 4.5) ілюструє систему для керування командами та користувачами, показуючи зв'язки між декількома ключовими класами, відповідальними за різні операції. TeamService обробляє бізнес-логіку для управління командою. Він взаємодіє з TeamMembershipRepository, щоб керувати даними членства в команді, включаючи методи перевірки права власності на команду, додавання або видалення учасників і отримання розмірів команди. Клас

TeamController надає ці функції через кінцеві точки HTTP, дозволяючи створювати та видаляти команди, а також керувати членством у них.

Клас AppUserService керує пов'язаними з користувачами операціями, включаючи реєстрацію користувача, оновлення інформації про користувача та перевірку наявності користувача електронною поштою. Цей клас взаємодіє з AppUserController, щоб розкрити функціональні можливості користувача через RESTful API. Крім того, MessageConsumer обробляє сповіщення електронною поштою про підтвердження реєстрації користувача. Класи Team і TeamMembership представляють основні об'єкти в системі, де Team містить список об'єктів TeamMembership, кожен з яких пов'язує користувача з певною командою разом із його роллю.

Основні методи:

- countTotalTeamMembersByUserId(Integer userId) повертає загальну кількість членства в команді для певного ідентифікатора користувача;
- validateUserTeamOwnership(List<TeamMembership>, Integer teamId) перевіряє, чи має користувач право власності на команду;
- addTeamMember(AddTeamMemberRequest запит, String userId: додає нового учасника до вказаної команди;
- deleteTeamMember(String teamId, String userId) видаляє учасника з команди;
- createTeam(CreateTeamRequest запит, String adminId) створює нову команду;
- getTeamForUse(String teamId) отримує команду для подальшого використання за її ідентифікатором;
- registerUser(Integer userId, String email) реєструє нового користувача;
- updateUser(запит UpdateUserRequest, String userId) оновлює інформацію про користувача;
- deleteUser(String userId, String email) видаляє користувача за його ідентифікатором або електронною поштою;

- getUsersByTeamId(String teamId) отримує всіх користувачів у вказаній команді;
- consumeEmailConfirmationNotification(UserRegisteredRequest userRegisteredRequest) споживає сповіщення про підтвердження електронною поштою та обробляє запит на реєстрацію;

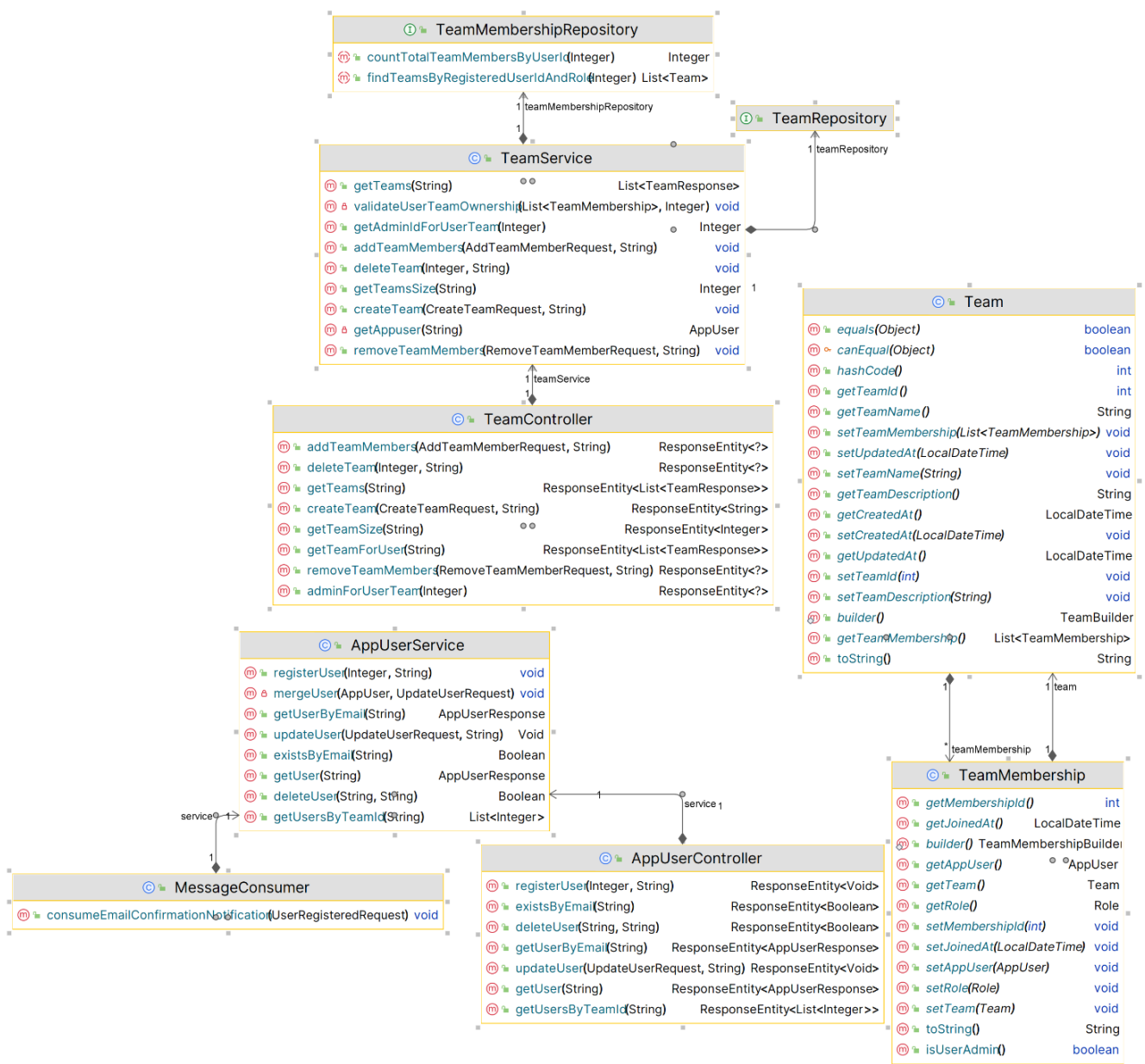


Рисунок 4.5 – Даграма класів

4.4 Метрика застосунка

При розробці додатка, було використано GitHub репозиторії для збереження ті підтримки програмного коду, метрики (табл. 1).

Таблиця 4.1 – Метрики програмного коду додатка у GitHub

Метрика	Значення
Кількість комотів в master бранчі	45
Кількість бранчів	1
Кількість закритих PR	1

Наведено метрики коду програмного продукту (табл. 4.2). Ці значення можуть допомогти знайти потенційні прогалини у дотриманні code conventions обраної мови програмування.

Таблиця 4.2 – Метрики програмного коду додатка

Метрика	Значення
Загальна кількість строк коду в проєкті	10545
Середня кількість строк коду в одному класі	70
Максимальна кількість строк коду в одному класі	267
Середня кількість строк коду в одному методі	23
Максимальна кількість строк коду в одному методі	68
Максимальна глибина дерева наслідування	2
Середня циклометрична складність метода	6
Максимальна циклометрична складність метода	14
Коментування коду	4%
Покриття коду модульними тестами	80% (покрита основна частина бізнес логіки)

Наведена таблиця контрольних питань, що оцінюють якість реалізації ІС (табл. 4.3).

Таблиця 3.3 – Контрольний список з якості реалізації додатка

Твердження	Відповідь	Обґрунтування у разі негативної відповіді
Використання Dependency Injection	Так	
Використання логування	Так	
Використання модульного тестування	Так	
Захист від SQL-ін'єкцій	Ні	Клієнт не взаємодіє з БД.
Захист від Javascript/XSS ін'єкцій	Так	
Використання валідації в усіх полях вводу форм для інтерфейсу	Так	
Використання інсталяторів або інтернет-магазинів	Ні	Застосунок не є мобільним додатком
Використання засобів синхронізації даних у разі використання багатопотоковості	Ні	Багатопотоковість не запроваджена.
Підтримка глобалізації додатку	Так	
Відсутність зашитих до програмного коду конфігураційних параметрів додатку	Так	
Використання UI патернів під час розробки UI	Так	
Враховані code style guide lines для обраної мови програмування	Так	
Враховані guide lines для розробки UI обраної ОС	Так	

5 ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ВЕБ-САЙТУ.

5.1 Модульне тестування

Модульні тести призначені для перевірки функціональності окремих одиниць коду, таких як методи або класи, окремо, тести зосереджені на тому, щоб переконатися, що кожна частина коду поводить себе належним чином, враховуючи набір вхідних даних, і що вона створює правильні результати. Модульні тести допомагають розробникам виявляти проблеми на ранніх стадіях процесу розробки, покращувати якість коду та гарантувати, що зміни (наприклад, рефакторинг) не порушують існуючу функціональність. Вони мають важливе значення для підтримки надійності коду без помилок, швидшого циклу розробки та полегшення регресійного тестування. Тести, як правило, автоматизовані, швидкі та виконуються незалежно від зовнішніх залежностей, часто з використанням фіктивних об'єктів для імітації взаємодії з базами даних, API або іншими службами. Ізоляція забезпечує ретельну перевірку логіки пристрою, який тестується, без втручання з боку інших частин програми. Модульні тести також забезпечують безпеку під час рефакторингу коду, гарантуючи, що нові зміни випадково не спричинять збоїв в існуючій функціональності.

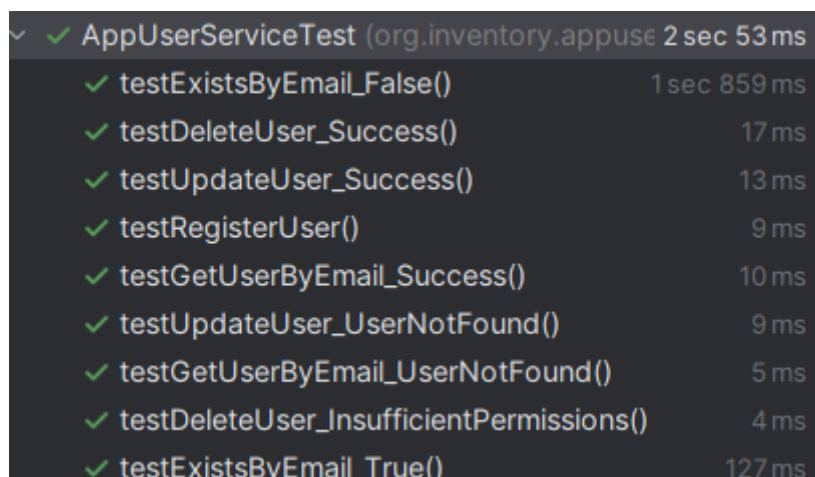
Виконано модульне тестування, використовуючи SpringBootTest [18] та Junit [19] із таким анотаціями:

- @Mock для залежностей репозиторіїв;
- @InjectMocks для створення екземпляра сервісу.

Виконано модульне тестування для сервісу, що обробляє запити про користувачів – AppUserService (рис. 5.1):

- testRegisterUser – перевіряє реєстрацію користувача, гарантуючи, що метод repository.save() викликається з правильними даними;

- testUpdateUser – забезпечує можливість оновлення інформації про користувача, якщо він існує, а оновлені поля відображають зміни. Включає сценарії для не знайдених користувачів або неавторизованих оновлень;
- testGetUserByEmail – підтверджує, що система отримує дані користувача електронною поштою, включно з його роллю. Тестує обробку випадків, коли користувач не існує;
- testExistsByEmail – перевіряє, чи система правильно визначає, чи існує користувач, на основі його електронної пошти.
- testDeleteUser – перевіряє видалення користувача, якщо він існує, і перевіряє дозволи на видалення. Включає сценарії недостатніх дозволів або відсутності користувачів.

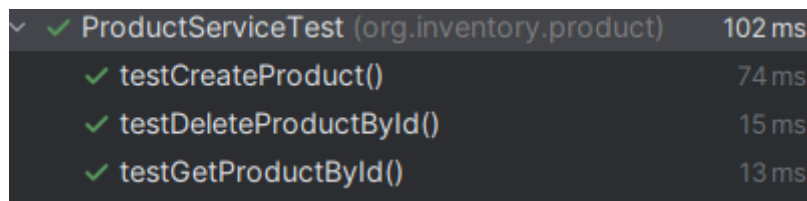


✓ AppUserServiceTest (org.inventory.appuse	2 sec 53 ms
✓ testExistsByEmail_False()	1 sec 859 ms
✓ testDeleteUser_Success()	17 ms
✓ testUpdateUser_Success()	13 ms
✓ testRegisterUser()	9 ms
✓ testGetUserByEmail_Success()	10 ms
✓ testUpdateUser_UserNotFound()	9 ms
✓ testGetUserByEmail_UserNotFound()	5 ms
✓ testDeleteUser_InsufficientPermissions()	4 ms
✓ testExistsByEmail_True()	127 ms

Рисунок 5.1 – Тестування AppUserService

Виконано модульне тестування для сервісу, що обробляє запити про команди – ProductService (рис. 5.2):

- testGetProductById – перевіряє, чи продукт отримано за ідентифікатором, і перевіряє, чи має користувач дозвіл на доступ до нього;
- testCreateProduct – перевіряє, чи правильно створено та збережено новий продукт;
- testDeleteProductById – перевіряє, чи продукт видалено, і чи повертається правильне повідомлення.



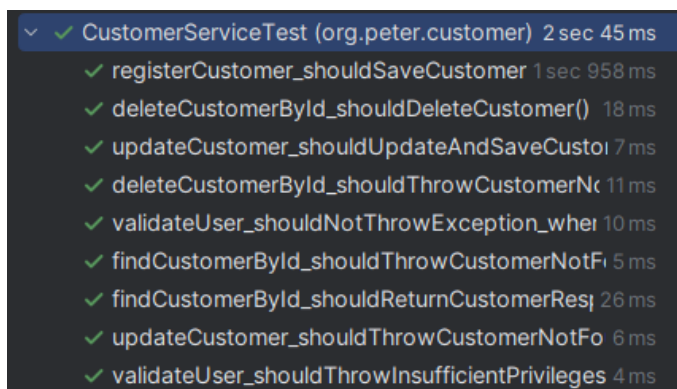
A screenshot of a test runner interface showing the results for ProductServiceTest. The test suite is expanded, showing four individual test methods, each with a green checkmark indicating success and a duration in milliseconds.

✓ ProductServiceTest (org.inventory.product)	102 ms
✓ testCreateProduct()	74 ms
✓ testDeleteProductById()	15 ms
✓ testGetProductById()	13 ms

Рисунок 5.2 – Тестування ProductService

Виконано модульне тестування для сервісу, що обробляє запити про клієнтів – CustomerService (рис. 5.3). Протестовано наступні методи:

- registerClient – перевіряє, що customerRepository.save() викликається з правильним об’єктом Customer;
- updateCustomer – тестує успішне оновлення, гарантуючи збереження змін. Перевіряє помилку за допомогою CustomerNotFoundException, коли клієнт не існує;
- findCustomerById – тестує успішне отримання CustomerResponse. Перевіряє CustomerNotFoundException, якщо клієнт не існує;
- deleteCustomerById – перевіряє успішне видалення, забезпечує обробку винятків, коли клієнт не знайдено;
- validateUser – забезпечує роботу перевірки привілеїв, перевіряючи невідповідні ідентифікатори користувачів, підтверджує відсутність виключення, коли ідентифікатори користувачів збігаються.



A screenshot of a test runner interface showing the results for CustomerServiceTest. The test suite is expanded, showing nine individual test methods, each with a green checkmark indicating success and a duration in milliseconds.

✓ CustomerServiceTest (org.peter.customer)	2 sec 45 ms
✓ registerCustomer_shouldSaveCustomer	1 sec 958 ms
✓ deleteCustomerById_shouldDeleteCustomer()	18 ms
✓ updateCustomer_shouldUpdateAndSaveCusto	7 ms
✓ deleteCustomerById_shouldThrowCustomerNc	11 ms
✓ validateUser_shouldNotThrowException_whe	10 ms
✓ findCustomerById_shouldThrowCustomerNotFi	5 ms
✓ findCustomerById_shouldReturnCustomerResj	26 ms
✓ updateCustomer_shouldThrowCustomerNotFo	6 ms
✓ validateUser_shouldThrowInsufficientPrivileges	4 ms

Рисунок 5.3 – Тестування CustomerService

Виконано модульне тестування для сервісу, що обробляє запити про переміщення товарів, наприклад, клієнт купив товар, було здійснене замовлення або співробітник самостійно змінив кількість – StockMovementService (рис. 5.4). Протестовано наступні методи:

- getMovementsForProduct – правильно отримано рух запасів для певного ідентифікатора продукту зі сховища, перевіряється, чи повернутий список рухів запасів відсортовано за полем дати в порядку зростання;
- addMovementsForProduct – рух зберігається в StockMovementRepository, перевіряється, чи метод створює виключення ProductNotFoundException, якщо наданий ідентифікатор продукту не існує, а також, що InventoryNotFoundException викидається, якщо не знайдено запасів для наданого ідентифікатора складу та продукту. Це підтверджує здатність служби обробляти відсутні інвентаризації. Підтверджується, що метод створює виключення IllegalArgumentException, коли відбувається спроба продажу, але запитана кількість перевищує доступний запас;
- getMovements – перевіряється, чи служба правильно отримує рух запасів із сховища на основі ідентифікатора користувача/адміністратора, перевіряється, чи отримані рухи точно відображаються в об'єктах StockMovementsResponse за допомогою засобу відображення.
- findStockStatistics – гарантує, що рух акцій згруповано за роком-місяць і правильно агреговано для певного користувача/адміністратора, перевіряється чи метод повертає порожній список, якщо не знайдено продуктів для користувача чи адміністратора. Перевіряється сортування.

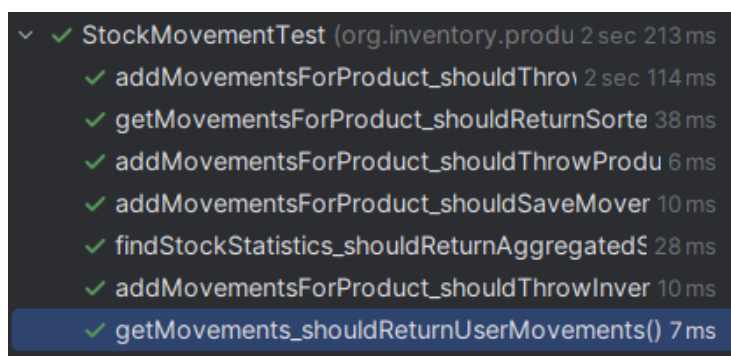
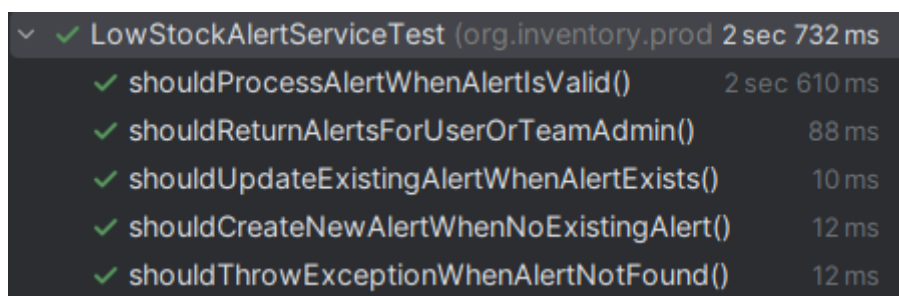


Рисунок 5.4 – Тестування StockMovementService

Виконано модульне тестування для сервісу, що обробляє запити про нестачу товарів на складі– `LowStockAlertService` (рис. 5.5). Створено наступні тести:

- `shouldReturnAlertsForUserOrTeamAdmin()` – гарантує, що метод `getAlerts` правильно отримує сповіщення про низькі запаси для користувача або адміністратора команди;
- `shouldCreateNewAlertWhenNoExistingAlert()` – перевіряє, що якщо для продукту не існує активного сповіщення, створюється нове сповіщення та надсилається електронний лист;
- `shouldUpdateExistingAlertWhenAlertExists()` – підтверджує, що якщо активне сповіщення вже існує, воно оновлюється замість створення нового, і електронний лист не надсилається;
- `shouldProcessAlertWhenAlertIsValid()` – гарантує, що під час обробки дійсного сповіщення про низький запас створюються руху запасів для поповнення запасу, а сповіщення деактивується;
- `shouldThrowExceptionWhenAlertNotFound()` – перевіряє, що під час спроби обробити сповіщення, яке не існує або неактивне, створюється виняткова ситуація `RuntimeException` із повідомленням «Попередження не знайдено».



✓ LowStockAlertServiceTest (org.inventory.prod	2 sec 732 ms
✓ shouldProcessAlertWhenAlertIsValid()	2 sec 610 ms
✓ shouldReturnAlertsForUserOrTeamAdmin()	88 ms
✓ shouldUpdateExistingAlertWhenAlertExists()	10 ms
✓ shouldCreateNewAlertWhenNoExistingAlert()	12 ms
✓ shouldThrowExceptionWhenAlertNotFound()	12 ms

Рисунок 5.5 – Тестування `LowStockAlertService`

Виконано модульне тестування для сервісу, що обробляє запити про склади, на який зберігаються товари на складі– `WarehouseService`(рис. 5.6). Створено наступні тести:

- `shouldCreateWarehouse()` – тестує створення сховища, імітує репозиторій місцезнаходжень, повертаючи порожнє розташування, і перевіряє, чи правильно збережено склад і розташування;
- `shouldReturnWarehouseById()` – перевіряє метод `getWarehouseById`;
- `shouldReturnWarehouseByIdWithUserId()` – варіант попереднього тесту, де метод приймає ідентифікатор користувача, щоб переконатися, що він працює з цим додатковим параметром;
- `shouldUpdateWarehouse()` – тестує метод `updateWarehouse`, перевіряє, чи правильно оновлено склад;
- `shouldThrowWarehouseNotFoundExceptionWhenUserDoesNotOwnWarehouse()` – перевіряє, що `WarehouseNotFoundException` створено, якщо користувач не є власником сховища, яке оновлюється;
- `shouldDeleteWarehouse()` – перевіряє, чи метод `deleteWarehouse` правильно видаляє склад;
- `shouldReturnTrueIfWarehouseExistsForUser()` – перевіряє, чи метод `existsById` повертає `true`, якщо склад існує та належить користувачеві;
- `shouldReturnFalseIfWarehouseDoesNotBelongToUser()` – перевіряє, чи метод `existsById` повертає `false`, якщо склад існує, але не належить користувачеві;
- `shouldReturnAllWarehouses()` – тестує метод `getAllWarehouses`, перевіряючи, що він повертає список складів.

✓ WarehouseServiceTest (org.peter.warehouse)	2 sec 131 ms
✓ shouldUpdateWarehouse()	2 sec 69 ms
✓ shouldReturnFalseIfWarehouseDoesNotBelongToUser()	5 ms
✓ shouldReturnTrueIfWarehouseExistsForUser()	5 ms
✓ shouldCreateWarehouse()	17 ms
✓ shouldReturnAllWarehouses()	10 ms
✓ shouldReturnWarehouseByIdWithUserId()	6 ms
✓ shouldThrowWarehouseNotFoundExceptionWhenUserDoesNotOwnWarehouse()	8 ms
✓ shouldReturnWarehouseById()	4 ms
✓ shouldDeleteWarehouse()	7 ms

Рисунок 5.6 – Тестування WarehouseService

5.2 Функціональне тестування

Функціональне тестування — це метод тестування програмного забезпечення, який перевіряє, чи система або її окремі компоненти виконують свої функції відповідно до заданих вимог. Основною метою цього тестування є переконатися, що програмне забезпечення працює так, як це було визначено в специфікаціях, і відповідає очікуванням користувачів.

Функціональне тестування проводиться для досягнення кількох ключових цілей. По-перше, воно допомагає перевірити, чи відповідає програмне забезпечення всім функціональним вимогам, визначеним у документації. По-друге, це тестування дозволяє виявити дефекти та помилки, які можуть виникати у процесі реалізації функціональності. Також важливою метою є перевірка взаємодії між компонентами системи, наприклад, між фронтендом і бекендом, або інтеграція із зовнішніми сервісами.

Окрім цього, функціональне тестування спрямоване на підтвердження того, що програмне забезпечення відповідає очікуванням кінцевих користувачів. Це забезпечує високу якість продукту, підвищує його зручність використання, а також створює довіру до системи як у замовників, так і у кінцевих користувачів.

Проведено функціональне тестування для функціональних вимог з реєстрації та авторизації (табл. 5.1).

Таблиця 5.1 – Функціональне тестування для функціональних вимог з реєстрації та авторизації

ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
1	Система вимагає введення електронної пошти та пароля.	1. Відкрити форму реєстрації. 2. Натиснути кнопку "Зареєструватись" без заповнення полів.	Система показала помилку про обов'язкові поля.	Фактичний результат відповідає очікованому

Продовження таблиці 5.1

ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
2	Система відхиляє пароль, що не відповідає критеріям безпеки	1. Ввести email. 2. Ввести пароль, що не відповідає критеріям. 3. Натиснути кнопку "Зареєструватись".	Система показала помилку про невідповідність пароля.	Фактичний результат відповідає очікованому
3	Система видає повідомлення: «Будь ласка, заповніть усі обов'язкові поля коректно» за некоректного вводу.	1. Ввести неповні або некоректні дані (наприклад, email без домену). 2. Натиснути кнопку "Зареєструватись".	Система показала відповідне повідомлення.	Фактичний результат відповідає очікованому
4	Система видає повідомлення: «Ця електронна пошта вже використовується» при спробі реєстрації на існуючий email.	1. Ввести вже зареєстрований email. 2. Ввести пароль. 3. Натиснути "Зареєструватись".	Система показала повідомлення про існуючий email.	Фактичний результат відповідає очікованому
5	Система видає повідомлення: «Реєстрація успішна. Ви можете увійти до системи».	1. Ввести коректні дані. 2. Натиснути кнопку "Зареєструватись".	Система показала повідомлення про успішну реєстрацію.	Фактичний результат відповідає очікованому
6	Система надсилає лист для підтвердження реєстрації після успішної реєстрації.	1. Завершити успішну реєстрацію. 2. Перевірити вказану поштову скриньку.	Лист із підтвердженням отримано.	Фактичний результат відповідає очікованому

Продовження таблиці 5.1

ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
7	Користувач переходить за посиланням у листі, і його реєстрація завершується.	1. Відкрити лист із підтвердженням реєстрації. 2. Перейти за надісланим посиланням	Реєстрація завершилась успішно.	Фактичний результат відповідає очікованому
8	Система вимагає введення електронної пошти та пароля для авторизації.	1. Відкрити форму входу. 2. Натиснути "Увійти" без введення даних	Система показала помилку про обов'язкові поля.	Фактичний результат відповідає очікованому
9	Система видає повідомлення: «Некоректний логін або пароль» при помилці в email або паролі.	1. Ввести некоректний email або пароль. 2. Натиснути "Увійти".	Система показала відповідне повідомлення.	Фактичний результат відповідає очікованому
10	При успішній авторизації користувач перенаправляється до робочого кабінету.	1. Ввести коректний email та пароль. 2. Натиснути "Увійти".	Користувач перенаправлений до кабінету	Фактичний результат відповідає очікованому

Проведено функціональне тестування для функціональних вимог з управління командою (табл. 5.2).

Таблиця 5.2 – Функціональне тестування для функціональних вимог з управління командою

ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
11	Система створює нову команду з	1. Авторизуватися як адміністратор.	. Система успішно	Фактичний результат

	указаною назвою та описом. Користувач, що створив команду, стає адміністратором.	2. Перейти до розділу "Команди". 3. Натиснути "Створити команду". 4. Ввести назву та опис команди. 5. Натиснути "Зберегти".	створила нову команду. Користувач, що створив команду, автоматично отримав роль адміністратора.	відповідає очікованому
12	Система додає нового учасника до команди з указаною роллю.	1. Авторизуватися як адміністратор. 2. Відкрити сторінку команди. 3. Натиснути "Додати учасника". 4. Ввести електронну пошту користувача та вибрати роль. 5. Натиснути "Зберегти".	Система успішно додала учасника до команди. Його роль відповідає указаній під час додавання.	Фактичний результат відповідає очікованому
13	Система виводить повідомлення: "Користувач ще не зареєстрований"	1. Авторизуватися як адміністратор. 2. Спробувати додати до команди користувача з неіснуючою електронною поштою.	Система видала повідомлення: "Користувач ще не зареєстрований".	Фактичний результат відповідає очікованому
14	Адміністратор успішно видаляє учасника з команди.	1. Авторизуватися як адміністратор. 2. Перейти до розділу "Команди". 3. Вибрати команду. 4. Вибрати учасника зі списку. 5. Натиснути "Видалити".	Учасник був успішно видалений із команди, і його більше не видно в списку учасників.	Фактичний результат відповідає очікованому

Проведено функціональне тестування для функціональних вимог з управління товарами (табл. 5.3).

Таблиця 5.3 – Функціональне тестування для функціональних вимог з управління товарами

ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
15	Система додає товар, якщо заповнені всі необхідні поля коректно.	1. Увійти в систему. 2. Перейти до розділу "Товари". 3. Натиснути "Додати товар". 4. Заповнити всі обов'язкові поля. 5. Натиснути "Зберегти".	Система успішно додала товар і відобразила повідомлення: «Товар успішно додано».	Фактичний результат відповідає очікуваному
16	Система не дає зберегти товар.	1. Увійти в систему. 2. Перейти до розділу "Товари". 3. Натиснути "Додати товар". 4. Залишити одне або кілька полів незаповненими. 5. Натиснути "Зберегти".	Система не зберігає товар	Фактичний результат відповідає очікуваному
17	Користувач може вибрати товар для редагування зі списку.	1. Увійти в систему. 2. Перейти до розділу "Товари". 3. Вибрати товар зі списку.	Система відкрила форму редагування обраного товару.	Фактичний результат відповідає очікуваному
18	Користувач може редагувати будь-яке з доступних полів товару.	1. Вибрати товар зі списку. 2. Змінити будь-яке поле (наприклад, опис). 3. Натиснути "Зберегти".	Система зберегла змінені дані товару.	Фактичний результат відповідає очікуваному

Продовження таблиці 5.3

ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
19	Система відображає повідомлення: «Інформацію про товар успішно оновлено».	1. Вибрати товар для редагування. 2. Змінити кілька полів з коректними даними. 3. Натиснути "Зберегти".	Система зберегла зміни та видала повідомлення: «Інформацію про товар успішно оновлено».	Фактичний результат відповідає очікованому

Проведено функціональне тестування для функціональних вимог з управління залишками товарів (табл. 5.4).

Таблиця 5.4 – Функціональне тестування для функціональних вимог з управління залишками товарів

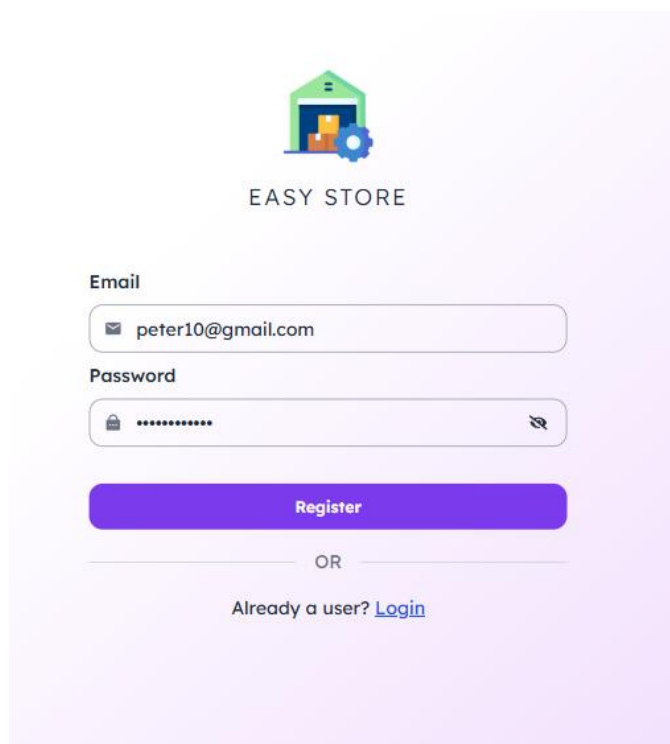
ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
20	Система автоматично оновлює залишки товарів після операції додавання.	1. Увійти в систему. 2. Додати товар до складу. 3. Перевірити залишки товару.	Залишки товару оновлено у системі в реальному часі.	Фактичний результат відповідає очікованому
21	Система оновлює залишки товарів кожні 30 хвилин, якщо операцій не відбувається.	1. Не виконувати жодних операцій протягом 30 хвилин. 2. Перевірити залишки товару.	Залишки товару автоматично оновлено за графіком.	Фактичний результат відповідає очікованому
22	Користувач отримує електронного листа з повідомленням про нестачу товару на складі.	1. Виконати операцію, що призводить до нестачі товару. 2. Перевірити електронну пошту.	Користувач отримав електронного листа з повідомленням про нестачу товару.	Фактичний результат відповідає очікованому

Продовження таблиці 5.4

ТС id	Очікуваний результат	Виконані кроки	Фактичний результат	Коментар
23	Користувач отримує сповіщення у веб-застосунку про нестачу товару на складі.	1. Виконати операцію, що призводить до нестачі товару. 2. Перевірити сповіщення у веб-застосунку.	Сповіщення з'явилося у веб-застосунку.	Фактичний результат відповідає очікованому
24	Користувач може замовити товар, і система надсилає PDF форму замовлення на електронну пошту.	1. Вибрати товар для замовлення. 2. Натиснути "Замовити". 3. Перевірити електронну пошту на отримання PDF форми.	Система створила PDF форму та надіслала її на електронну пошту користувача.	Фактичний результат відповідає очікованому
25	Система відображає повідомлення про успішне створення замовлення	1. Виконати замовлення товару. 2. Перевірити, чи з'явилося повідомлення про успішне створення замовлення.	Система відобразила повідомлення: «Замовлення успішно створено».	Фактичний результат відповідає очікованому

5.3 Інструкція користувача

Користувач має змогу зареєструватися, якщо він до цього це не зробив, ввівши необхідні дані (рис. 5.7). При успішній реєстрації, користувач отримує повідомлення (рис. 5.8) та лист на електронну пошту про необхідність активації акаунту (рис. 5.9). Після активації акаунту користувач може увійти у систему (рис. 5.10).



The image shows a registration form for 'EASY STORE'. At the top is a logo of a green house with a blue gear and orange boxes. Below the logo is the text 'EASY STORE'. The form has two input fields: 'Email' with the value 'peter10@gmail.com' and 'Password' with masked characters. Below these is a blue 'Register' button. Under the button is the text 'OR' and a link 'Already a user? Login'.

Рисунок 5.7 – Реєстрація користувача

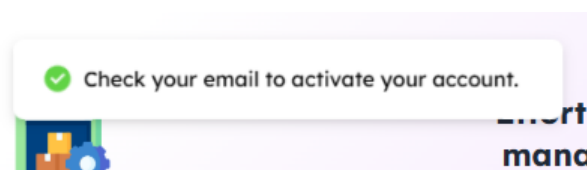
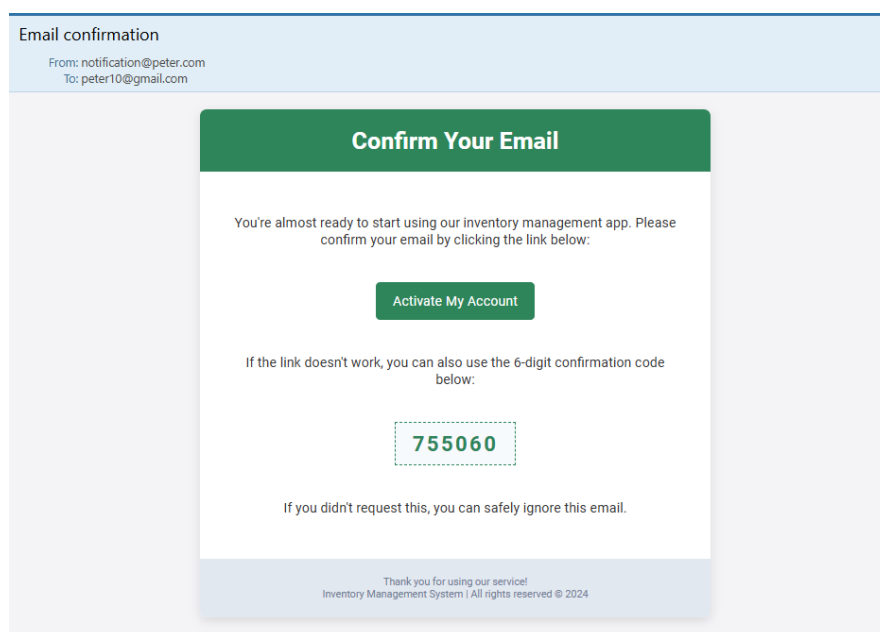
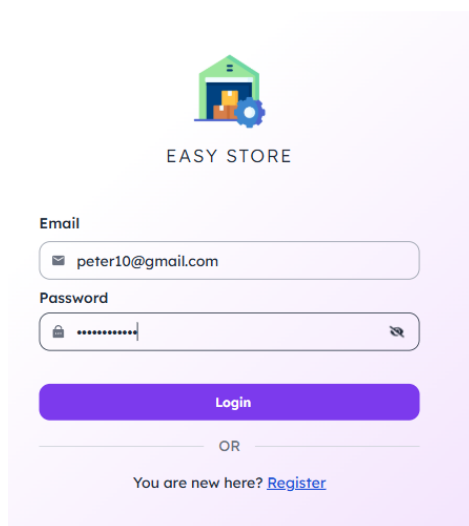



Рисунок 5.8 – Повідомлення, отримане після успішної реєстрації



The image shows an email confirmation page. At the top is a header 'Email confirmation' with the text 'From: notification@peter.com' and 'To: peter10@gmail.com'. Below the header is a green box with the text 'Confirm Your Email'. Inside the green box is a white box with the text 'You're almost ready to start using our inventory management app. Please confirm your email by clicking the link below:'. Below the text is a green button 'Activate My Account'. Below the button is the text 'If the link doesn't work, you can also use the 6-digit confirmation code below:'. Below the text is a green box with the code '755060'. Below the code is the text 'If you didn't request this, you can safely ignore this email.' At the bottom of the page is the text 'Thank you for using our service! Inventory Management System | All rights reserved © 2024'.

Рисунок 5.9 – Електронний лист про необхідність активації акаунту




 EASY STORE

Email

Password

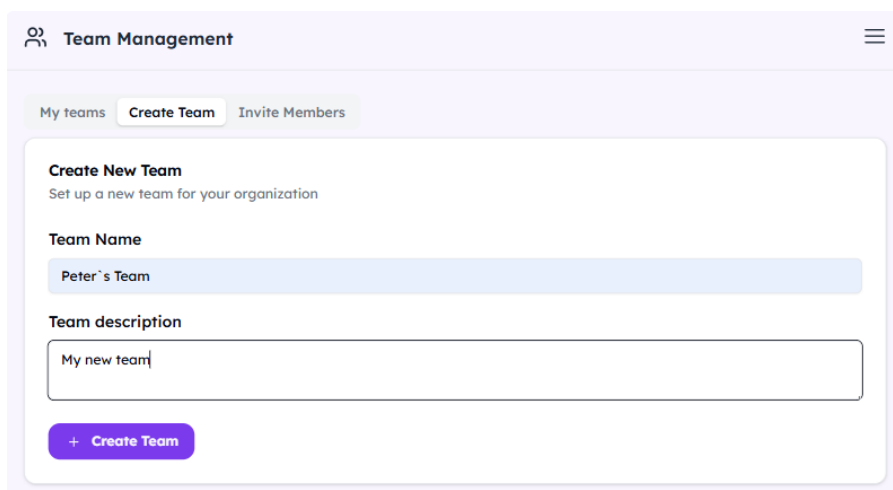
[Login](#)

OR

You are new here? [Register](#)

Рисунок 5.10 – Вхід у систему

Після успішної реєстрації, користувач може створити свою команду, заповнивши відповідні поля (рис. 5.11). Якщо деякі поля були залишені пустими, впливає повідомлення (рис. 5.12). Користувач додає користувачів собі у команду (рис. 5.13). Користувач може управляти командою (рис. 5.14).



Team Management

My teams **Create Team** Invite Members

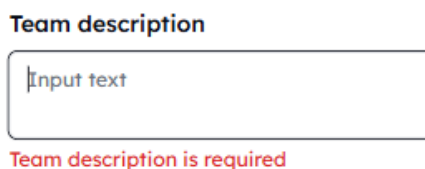
Create New Team
Set up a new team for your organization

Team Name

Team description

[+ Create Team](#)

Рисунок 5.11– Створення команди



Team description

Team description is required

Рисунок 5.12 – Повідомлення з помилкою

Team Management

My teams Create Team **Invite Members**

Invite Team Members
Add new members to your team

Email Address
peter1@gmail.com

Team
Peter`s Team

User role
Manager

+ Invite

Рисунок 5.13 – Додавання користувача у команду

Peter`s Team Members
Manage your current team members

Search members

Name	Email	Role	Actions
Peter Popazov	peter10@gmail.com	ADMIN	
Peter1 Popazov	peter1@gmail.com	MANAGER	Delete

Рисунок 5.14 – Управління командою

Успішно створивши команду, користувач може приступити до додавання нових товарів у систему (рис. 5.15). Якщо невірно були вказані дані, користувач може натиснути на кнопку контекстного меню (рис. 5.16) та обрати: видалити (рис. 5.17) або змінити (рис. 5.18).

Add new item

Item name

Product SKU

Barcode

Price

iPhone

2324523

34798

1119.00

Description

new iPhone

Warehouse

Category

Warehouse1

Phone

+ Add Warehouse

+ Add Category

Warehouse stock

Total Stock

Min stock level

Max stock level

50

50

5

100

Add

Discard

Рисунок 5.14 – Додавання товарів у систему

SKU	Product	Price	Quantity	Category	Reorder Level	Warehouses
2324523	iPhone	\$1,119.00	50	Phone	5	Warehouse1
PAGINATION						<div>Edit</div> <div>Delete</div>

Рисунок 5.16 – Кнопка контекстного меню

Delete Item 2324523

Are you sure you want to delete **Item 2324523** permanently? This action cannot be undone.

Cancel

Delete

Рисунок 5.17 – Видалення товару

Edit item

Item name

Product SKU

Barcode

Price

iPhone

2324523

34798

1119

Description

new iPhone

Warehouse

Category

Warehouse1

Phone

+ Add Warehouse

+ Add Category

Warehouse stock

Total Stock

Min stock level

Max stock level

50

50

5

100

Edit

Discard

Рисунок 5.18 – Змінення даних про товар

При ході роботи із системою кількість відповідного товару на складі може стати менше вказаного, тоді користувач отримає повідомлення у системі (рис. 5.19) та отримає електронний лист (рис 5.20) з PDF файлом (рис 5.21).

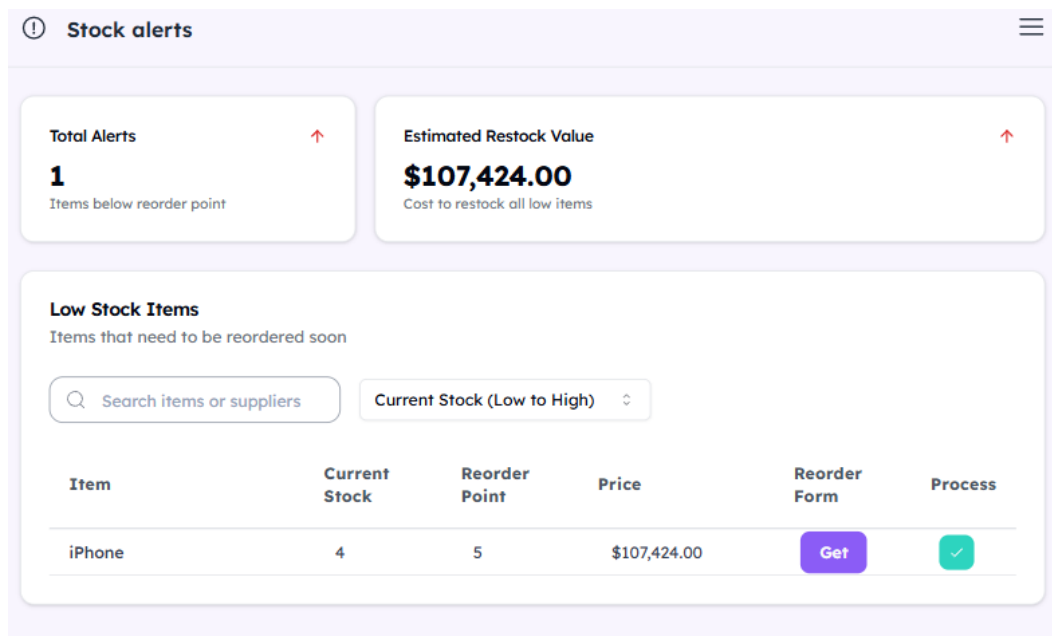


Рисунок 5.19 – Повідомлення про нестачу товарів

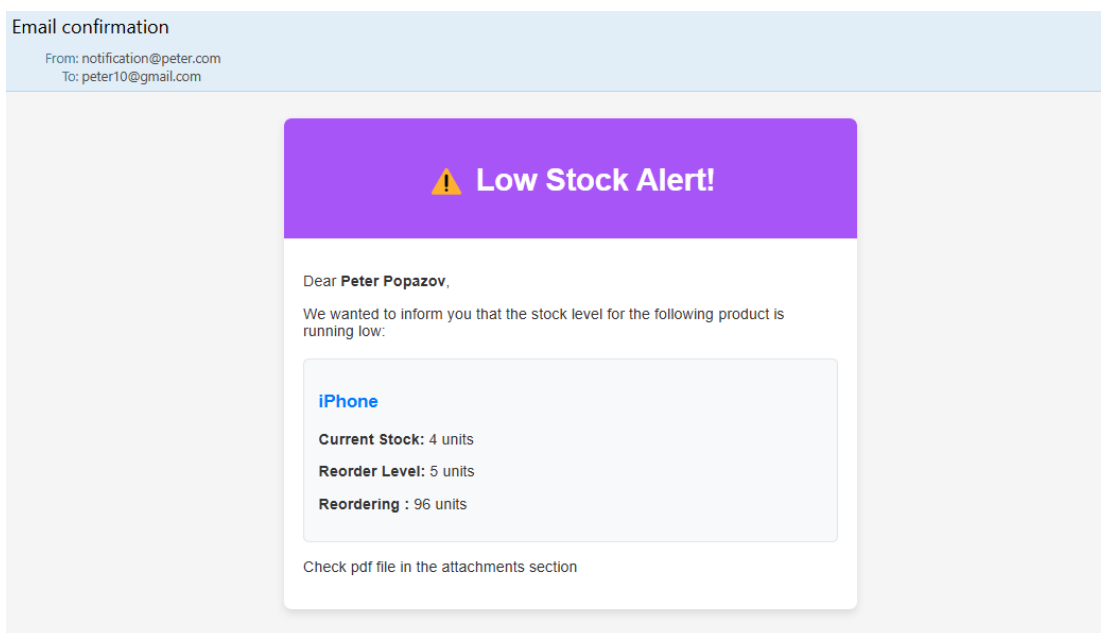


Рисунок 5.20 – Електронний лист про нестачу товарів

Product Order Form

Customer Information

Name: Peter Popazov

Email: peter10@gmail.com

Address:

Supplier Information

Name:

Email:

Address:

Order Details:

Product Name	Quantity	Unit Price	Subtotal Price
iPhone	96	1119.00	107424.00

Discount:

Total Price:

Date: 2024-11-24 00:30:58

Signature:

Рисунок 5.21 – PDF форма

Користувач може самостійно замовити PDF форму із інтерфейсу системи (рис. 5.22). Після успішного оформлення замовлення, користувач може позначити відповідне повідомлення, як оброблене (рис. 5.23).

Item	Current Stock	Reorder Point	Price	Reorder Form
iPhone	4	5	\$107,424.00	<div>Get</div>

Рисунок 5.22 – Самостійно замовити форму

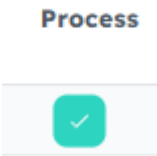


Рисунок 5.23 – Кнопка позначення повідомленн, як обробленого

Після деякого часу роботи із системою, користувачі бачитимуть заповнену статистику, наприклад, графіки на головній сторінці (рис. 5.24), заповнена таблиця з переміщеннями товарів (замовлення, продаж або ручне змінення) (рис. 5.25).

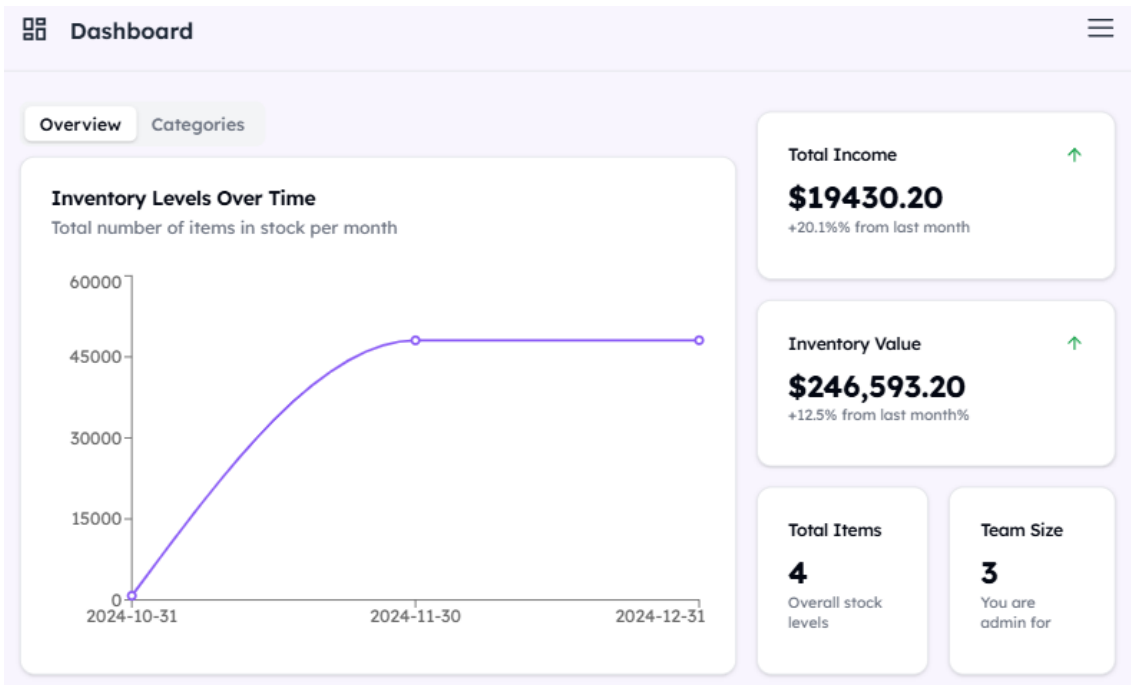


Рисунок 5.24 – Головна сторінка

All Movements				
Recent Inventory Movements				
Latest inbound and outbound transactions				
ID	Item	Type	Quantity	Date
11	iPhone	Sale	2	2024-12-02 21:53
10	Apples	Adjustment	2900	2024-11-18 18:34
9	Apples	Adjustment	900	2024-11-18 18:33
8	Lemons	Purchase	2020	2024-11-14 19:03
7	Lemons	Purchase	7778	2024-11-14 18:58
6	Apples	Purchase	1013	2024-11-14 18:56
5	Apples	Sale	920	2024-11-12 16:31
4	Apples	Purchase	11220	2024-11-11 16:11
1	Apples	Sale	110	2024-11-11 12:05
2	Apples	Sale	220	2024-11-10 16:07
3	Oranges	Sale	1220	2024-10-10 16:11

Рисунок 5.25 – Заповнена таблиця з переміщеннями товарів

ВИСНОВКИ

У ході виконання курсового проєкту, було розроблено веб-додаток для ведення івентаризації, який забезпечує управління запасами продуктів та моніторинг їх рівнів. Основна функціональність системи включає створення команд, створення нових продуктів, управління їх розташуванням, відстеження руху товарів, сповіщення про низький рівень запасів, а також створення статистики щодо руху товарів. Користувачі мають можливість додавати нові товари, оновлювати їхні дані, переглядати інформацію про запаси та отримувати попередження про необхідність поповнення запасів.

Для реалізації було використано мікросервісну архітектуру, що дозволяє розділити функціонал на незалежні сервіси, такі як управління продуктами, складами, та сповіщеннями. У проєкті застосовувалися технології Spring Boot, що забезпечує швидку розробку серверної частини, а також PostgreSQL та MongoDB для збереження даних. Для обміну повідомленнями між мікросервісами використовувався Kafka, що гарантує надійну доставку подій і взаємодію компонентів, а також FeignClient – це інструмент для розробників, що спрощує створення підключень до веб-служб, дозволяючи визначити інтерфейс. Також у роботі реалізовано валідацію даних і роботу з ролями користувачів для захисту доступу до функцій системи.

На рівні клієнтського інтерфейсу застосовувалися React і Tailwind CSS, що надали сучасний і зручний дизайн інтерфейсу. Система підтримує REST API для інтеграції з іншими сервісами. Крім того, були реалізовані функції сповіщення через електронну пошту про необхідність поповнення запасів за допомогою Thymeleaf і інтеграції з поштовими сервісами.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zoho Inventory URL: <https://www.zoho.com/inventory/> (дата звернення: 24.11.2024)
2. Uspasy URL: <https://uspasy.ua/> (дата звернення: 24.11.2024)
3. Олександр Блажко. Дисципліна ТСПП. Лабораторна робота №1 – Формалізований опис бізнес-вимог та вимог користувача до програмного продукту
4. Draw.io URL: <https://app.diagrams.net/> (дата звернення: 24.11.2024)
5. Java URL: <https://www.java.com/en/> (дата звернення: 24.11.2024)
6. Spring Boot URL: <https://spring.io/projects/spring-boot> (дата звернення: 24.11.2024)
7. PostgreSQL Documentation URL: <https://www.postgresql.org/docs/> (дата звернення: 24.11.2024)
8. MongoDB Community URL: <https://www.mongodb.com/try/> (дата звернення: 24.11.2024)
9. Apache Kafka URL: <https://kafka.apache.org/> (дата звернення: 24.11.2024)
10. JWON Web Token Documentation URL: <https://jwt.io/> (дата звернення: 24.11.2024)
11. React Documentation URL: <https://react.dev/> (дата звернення: 24.11.2024)
12. Shadcn Documentation URL: <https://ui.shadcn.com/docs/> (дата звернення: 24.11.2024)
13. Docker Documentation URL: <https://docs.docker.com/> (дата звернення: 24.11.2024)
14. PlantUml URL: <https://plantuml.com/> (дата звернення: 24.11.2024)
15. Lucidchart URL: <https://www.lucidchart.com/> (дата звернення: 24.11.2024)
16. Як користуватися Hibernate. Стаття URL: <https://dou.ua/lenta/articles/how-to-use-hibernate/> (дата звернення: 24.11.2024)
17. AWS: cloud computing services URL: <https://aws.amazon.com/> (дата звернення: 24.11.2024)

18. Testing in Spring Boot. Baeldung URL: <https://www.baeldung.com/spring-boot-testing> (дата звернення: 24.11.2024)

19. Junit Documentation URL: <https://junit.org/junit5/> (дата звернення: 24.11.2024)

20. GitHub Documentation URL: <https://github.com/> (дата звернення: 24.11.2024)

21. GitHub репозиторій курсового проєкту URL: <https://github.com/peterporazov/smart-inventory-system> (дата звернення: 24.11.2024)

ДОДАТОК А – Сповіщення про недостатню кількість товарів

Сповіщення, що отримує користувач системи, коли кількість відповідного товару впала за відповідне значення (рис. А.1). Сповіщення надсилається користувачу на пошту, яку було вказано при реєстрації.

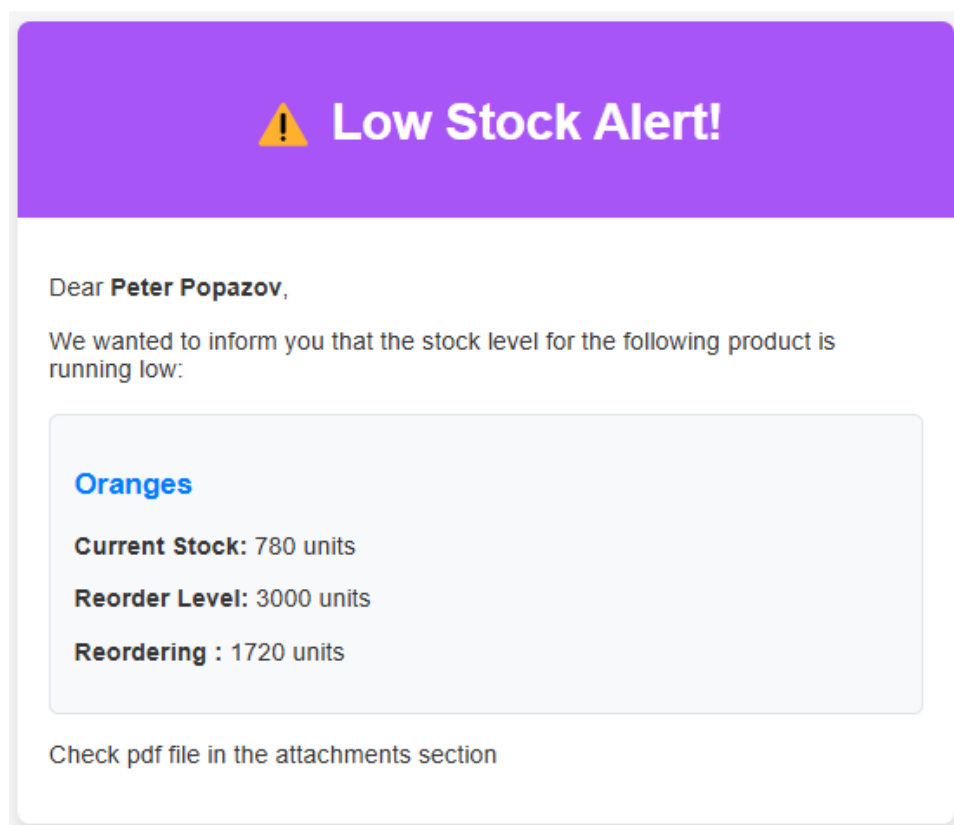


Рисунок А.1 – Сповіщення на електронну пошту

ДОДАТОК Б – Форма для замовлення товарів

Форма, якуотримує користувач системи, коли кількість відповідного товару впала за відповідне значення (рис. Б.1). Коистувач Може самостійно замовити таау форму, на сторінці «Сповіщення», натиснувчи на відповідну кнопку. Сповіщення надсилається користувачу на пошту, яку було вказано приреєстрації.

Product Order Form

Customer Information	Supplier Information
Name: Peter Popazov	Name: _____
Email: peter@gmail.com	Email: _____
Address: _____	Address: _____

Order Details:

Product Name	Quantity	Unit Price	Subtotal Price
Oranges	1720	115.84	199244.80

Discount: _____

Total Price: _____

Date: 2024-11-18 18:45:47

Signature: _____

Рисунок Б.1 – Форма для замовлення товарів