

UNIwersYTET RZESZOWSKI
WYDZIAŁ NAUK ŚCIŚLYCH I TECHNICZNYCH
INSTYTUT INFORMATYKI



Yevhen Marchak & Mykhailo Kleban
134945, 134922

Bazy danych

Dokumentacja projektu System zarządzania magazynem

Praca projektowa

Praca wykonana pod kierunkiem
dr inż. Piotr Grochowalski

Rzeszów 2026

Spis treści

1. Specyfikacja tematu projektu	7
1.1. Cel projektu	7
1.2. Zakres projektu	7
1.3. Technologie i narzędzia	7
1.4. Instrukcja uruchomienia projektu	8
2. Aspekt projektowy baz danych	9
2.1. Zdefiniowane struktury tabel	9
2.1.1. Tabela <code>categories</code>	9
2.1.2. Tabela <code>suppliers</code>	9
2.1.3. Tabela <code>clients</code>	9
2.1.4. Tabela <code>employees</code>	10
2.1.5. Tabela <code>locations</code>	10
2.1.6. Tabela <code>shipments</code>	10
2.1.7. Tabela <code>shipment_details</code>	10
2.1.8. Tabela <code>products</code>	11
2.1.9. Tabela <code>inventory</code>	11
2.1.10. Tabela <code>receipts</code>	11
2.1.11. Tabela <code>receipt_details</code>	11
2.2. Diagram związków encji	12
2.3. Powiązania pomiędzy tabelami	12
2.4. Kod realizujący bazę danych	13
2.5. Normalizacja bazy danych	15
2.5.1. Pierwsza postać normalna (1NF)	15
2.5.2. Druga postać normalna (2NF)	16
2.5.3. Trzecia postać normalna (3NF)	16
2.5.4. Podsumowanie	16
3. Zapytania problemowe i ich realizacja	17
3.1. Premia dla pracowników za obsługane wysyłki (zrealizowane w interfejsie użytkownika)	17
3.2. Automatyczne oznaczanie przeterminowanych wysyłek	17
3.3. Wykrywanie braków magazynowych	18
3.4. Obliczenie stopnia realizacji dostawy (zrealizowane w interfejsie użytkownika)	18
3.5. Najaktywniejsi klienci w danym okresie	19
3.6. Kontrola pojemności lokalizacji magazynowej	20
3.7. Aktualizacja stanu magazynu po przyjęciu dostawy	20
3.8. Ocena produktywności pracowników (zrealizowane w interfejsie użytkownika)	21
3.9. Najczęściej wysyłane produkty (zrealizowane w interfejsie użytkownika)	21

3.10. Walidacja poprawności dostawy.....	22
3.11. Wymagania CRUD systemu	23
4. Przedstawienie powstałej bazy danych i wdrożonych mechanizmów	25
4.1. Realizacja mechanizmu CRUD	25
4.1.1. Operacja Create.....	25
4.1.2. Operacja Read.....	26
4.1.3. Operacja Update.....	26
4.1.4. Operacja Delete.....	27
5. Koncepcja dostępu zdalnego do bazy danych	28
5.1. Założenia architektoniczne	28
5.2. Realizacja połączenia z bazą danych.....	28
5.3. Zakres dostępu do aplikacji	28
6. Interfejs użytkownika aplikacji	29
6.1. Główne okno aplikacji.....	29
6.2. Zarządzanie kategoriami produktów	29
6.3. Zarządzanie klientami	30
6.4. Zapytanie problemowe – premia pracowników	30
6.5. Analiza produktywności pracowników	31
6.6. Zarządzanie pracownikami.....	32
6.7. Zarządzanie stanami magazynowymi.....	32
6.8. Zarządzanie lokalizacjami magazynowymi.....	33
6.9. Najczęściej wysyłane produkty	34
6.10. Zarządzanie produktami	34
6.11. Stopień realizacji dostawy	35
6.12. Wynik obliczenia stopnia realizacji dostawy.....	36
6.13. Zarządzanie przyjęciami towaru.....	36
6.14. Zarządzanie wysyłkami	37
6.15. Zarządzanie dostawcami.....	37
7. Repozytorium projektu (GitHub)	38
7.1. Dokumentacja projektu (GitHub).....	38
Bibliografia	39
Spis rysunków	40
Spis listingów	41

1. Specyfikacja tematu projektu

1.1. Cel projektu

Celem projektu było zaprojektowanie oraz implementacja relacyjnej bazy danych dla systemu zarządzania magazynem. Projekt skupia się na stworzeniu spójnej, logicznej oraz znormalizowanej struktury danych, umożliwiającej efektywne przechowywanie i przetwarzanie informacji związanych z funkcjonowaniem magazynu.

Głównym założeniem projektu było opracowanie bazy danych, która umożliwia zarządzanie stanami magazynowymi, produktami, kategoriami, klientami, pracownikami oraz procesami przyjęcia i wydania towarów. System wspiera realizację podstawowych operacji na danych (CRUD), a także zapytań problemowych pozwalających na analizę danych magazynowych.

Celem jest praktyczne zastosowanie wiedzy z zakresu projektowania relacyjnych baz danych, definiowania relacji pomiędzy encjami, normalizacji danych oraz integracji bazy danych z warstwą aplikacyjną.

1.2. Zakres projektu

Zakres projektu obejmuje zaprojektowanie oraz implementację relacyjnej bazy danych systemu zarządzania magazynem. Baza danych umożliwia przechowywanie informacji niezbędnych do obsługi procesów magazynowych oraz zapewnia integralność i spójność danych.

W ramach projektu zrealizowano:

- definicję struktur tabel wraz z kluczami głównymi i obcymi,
- zaprojektowanie relacji pomiędzy encjami systemu,
- implementację operacji CRUD dla kluczowych tabel,
- realizację zapytań problemowych o charakterze analitycznym,
- opracowanie diagramu związków encji (ERD),
- integrację bazy danych z aplikacją posiadającą interfejs graficzny.

Projekt obejmuje warstwę bazodanową oraz aplikacyjną. Interfejs użytkownika umożliwia wygodną obsługę systemu, natomiast logika biznesowa opiera się na zapytaniach SQL wykonywanych na relacyjnej bazie danych.

1.3. Technologie i narzędzia

Do realizacji projektu wykorzystano następujące technologie oraz narzędzia:

- **PostgreSQL** – relacyjny system zarządzania bazą danych,
- **pgAdmin4** – narzędzie administracyjne do zarządzania bazą danych,
- **SQL** – język zapytań do definiowania struktury bazy danych oraz operacji na danych,
- **JetBrains IntelliJ IDEA** – środowisko programistyczne wykorzystane do implementacji aplikacji, logiki systemu oraz interfejsu graficznego użytkownika (GUI),

- **Visual Studio Code** – środowisko używane do tworzenia dokumentacji projektu w systemie \LaTeX .
- **JDBC** – interfejs komunikacji aplikacji Java z bazą danych,

1.4. Instrukcja uruchomienia projektu

Aby poprawnie uruchomić system na lokalnym środowisku, należy postępować zgodnie z poniższymi krokami:

1. Pobranie kodu źródłowego:

- Pobierz projekt z repozytorium GitHub, klikając przycisk *Code*, a następnie wybierając opcję *Download ZIP*.
- Rozpakuj pobrany folder ProjectBazyDanych-2rok-.
- Otwórz rozpakowany folder w wybranym środowisku programistycznym (zalecane: JetBrains IntelliJ IDEA).

2. Przygotowanie relacyjnej bazy danych (PostgreSQL):

- Uruchom narzędzie pgAdmin4 i utwórz nową, pustą bazę danych o nazwie SystemZarzadzaniaMagazynem.
- Kliknij prawym przyciskiem myszy na utworzoną bazę danych i wybierz opcję *Restore*. Wybierz format Plain, i w pole Filename wklej plik "BD_file.sql" i zatem naciśnij na przycisk *Restore*.
- Kliknij prawym przyciskiem myszy na utworzoną bazę danych i wybierz opcję *Refresh*.

3. Konfiguracja połączenia z bazą danych:

- Otwórz projekt aplikacji w środowisku JetBrains IntelliJ IDEA.
- Skonfiguruj parametry połączenia z bazą danych w pliku "DataBaseConnection", takie jak: nazwa bazy danych, użytkownik oraz hasło.
- Upewnij się, że aplikacja posiada poprawny dostęp do serwera PostgreSQL.

4. Uruchomienie aplikacji:

- Upewnij się, że serwer bazy danych PostgreSQL jest uruchomiony.
- Uruchom aplikację z poziomu środowiska JetBrains IntelliJ IDEA. Uruchomić plik Main.
- Po uruchomieniu aplikacji możliwa jest interakcja z systemem poprzez interfejs graficzny użytkownika (GUI).

Po wykonaniu powyższych kroków system zarządzania magazynem jest gotowy do pracy i umożliwia realizację operacji na danych magazynowych.

2. Aspekt projektowy baz danych

2.1. Zdefiniowane struktury tabel

W niniejszym podrozdziale przedstawiono szczegółowy opis tabel wchodzących w skład relacyjnej bazy danych systemu zarządzania magazynem. Każda tabela została zaprojektowana w sposób zapewniający spójność danych oraz poprawne odwzorowanie procesów magazynowych.

2.1.1. Tabela `categories`

Tabela `categories` przechowuje informacje o kategoriach produktów dostępnych w magazynie.

- `category_id` (`BIGSERIAL`, `PK`) — unikalny identyfikator kategorii,
- `name` (`text`, `NOT NULL`) — nazwa kategorii,
- `description` (`text`) — opis kategorii.

2.1.2. Tabela `suppliers`

Tabela `suppliers` zawiera dane dotyczące dostawców towarów.

- `supplier_id` (`BIGSERIAL`, `PK`) — identyfikator dostawcy,
- `company_name` (`text`, `NOT NULL`) — nazwa firmy dostawcy,
- `address` (`text`) — adres dostawcy,
- `phone` (`text`) — numer telefonu,
- `email` (`text`) — adres e-mail,
- `tax_id` (`text`) — numer podatkowy.

2.1.3. Tabela `clients`

Tabela `clients` przechowuje dane klientów systemu magazynowego.

- `client_id` (`BIGSERIAL`, `PK`) — identyfikator klienta,
- `company_name` (`text`, `NOT NULL`) — nazwa firmy klienta,
- `delivery_address` (`text`) — adres dostawy,
- `phone` (`text`) — numer telefonu,
- `email` (`text`) — adres e-mail,
- `tax_id` (`text`) — numer podatkowy.

2.1.4. Tabela **employees**

Tabela `employees` zawiera informacje o pracownikach magazynu.

- `employee_id` (`BIGSERIAL`, `PK`) — identyfikator pracownika,
- `first_name` (`text`, `NOT NULL`) — imię,
- `last_name` (`text`, `NOT NULL`) — nazwisko,
- `position` (`varchar(100)`, `NOT NULL`) — stanowisko,
- `hire_date` (`date`) — data zatrudnienia,
- `phone` (`text`) — numer telefonu,
- `email` (`text`) — adres e-mail.

2.1.5. Tabela **locations**

Tabela `locations` definiuje lokalizacje magazynowe.

- `location_id` (`BIGSERIAL`, `PK`) — identyfikator lokalizacji,
- `location_code` (`text`, `NOT NULL`) — kod lokalizacji,
- `location_type` (`varchar(50)`, `NOT NULL`) — typ lokalizacji,
- `max_capacity` (`numeric`) — maksymalna pojemność lokalizacji.

2.1.6. Tabela **shipments**

Tabela `shipments` rejestruje wydania towarów z magazynu do klientów. Zawiera informacje dotyczące realizowanych wysyłek oraz osób odpowiedzialnych za ich obsługę.

- `shipment_id` (`BIGSERIAL`, `PK`) — identyfikator wydania,
- `client_id` (`bigint`, `NOT NULL`, `FK` → `clients.client_id`) — klient,
- `employee_id` (`bigint`, `FK` → `employees.employee_id`) — pracownik realizujący wydanie,
- `shipment_date` (`date`, `NOT NULL`) — data wydania,
- `status` (`varchar(50)`, `NOT NULL`) — status wydania (np. „oczekujące”, „zrealizowane”).

2.1.7. Tabela **shipment_details**

Tabela `shipment_details` przechowuje szczegółowe informacje dotyczące produktów wydawanych w ramach konkretnego wydania magazynowego. Stanowi tabelę pośrednią realizującą relację wiele do wielu pomiędzy wydaniem a produktami.

- `shipment_id` (`bigint`, `NOT NULL`, `FK` → `shipments.shipment_id`),
- `product_id` (`bigint`, `NOT NULL`, `FK` → `products.product_id`),
- `quantity` (`numeric`, `NOT NULL`) — ilość wydanego produktu.

2.1.8. Tabela **products**

Tabela `products` przechowuje informacje o produktach.

- `product_id` (`BIGSERIAL`, `PK`) — identyfikator produktu,
- `sku` (`text`, `NOT NULL`) — kod SKU produktu,
- `name` (`text`, `NOT NULL`) — nazwa produktu,
- `description` (`text`) — opis produktu,
- `category_id` (`bigint`, `FK` → `categories.category_id`) — kategoria produktu,
- `supplier_id` (`bigint`, `FK` → `suppliers.supplier_id`) — dostawca produktu,
- `weight` (`numeric`) — waga produktu,
- `dimensions` (`text`) — wymiary produktu.

2.1.9. Tabela **inventory**

Tabela `inventory` przechowuje informacje o stanach magazynowych.

- `inventory_id` (`BIGSERIAL`, `PK`) — identyfikator rekordu magazynowego,
- `product_id` (`bigint`, `FK` → `products.product_id`) — produkt,
- `location_id` (`bigint`, `FK` → `locations.location_id`) — lokalizacja magazynowa,
- `quantity` (`numeric`, `NOT NULL`) — ilość produktu,
- `last_updated` (`timestampz`) — data ostatniej aktualizacji.

2.1.10. Tabela **receipts**

Tabela `receipts` rejestruje przyjęcia towarów do magazynu.

- `receipt_id` (`BIGSERIAL`, `PK`) — identyfikator przyjęcia,
- `supplier_id` (`bigint`, `FK` → `suppliers.supplier_id`) — dostawca,
- `employee_id` (`bigint`, `FK` → `employees.employee_id`) — pracownik przyjmujący,
- `receipt_date` (`date`, `NOT NULL`) — data przyjęcia,
- `external_invoice_no` (`text`) — numer faktury zewnętrznej,
- `status` (`varchar(50)`, `NOT NULL`) — status przyjęcia.

2.1.11. Tabela **receipt_details**

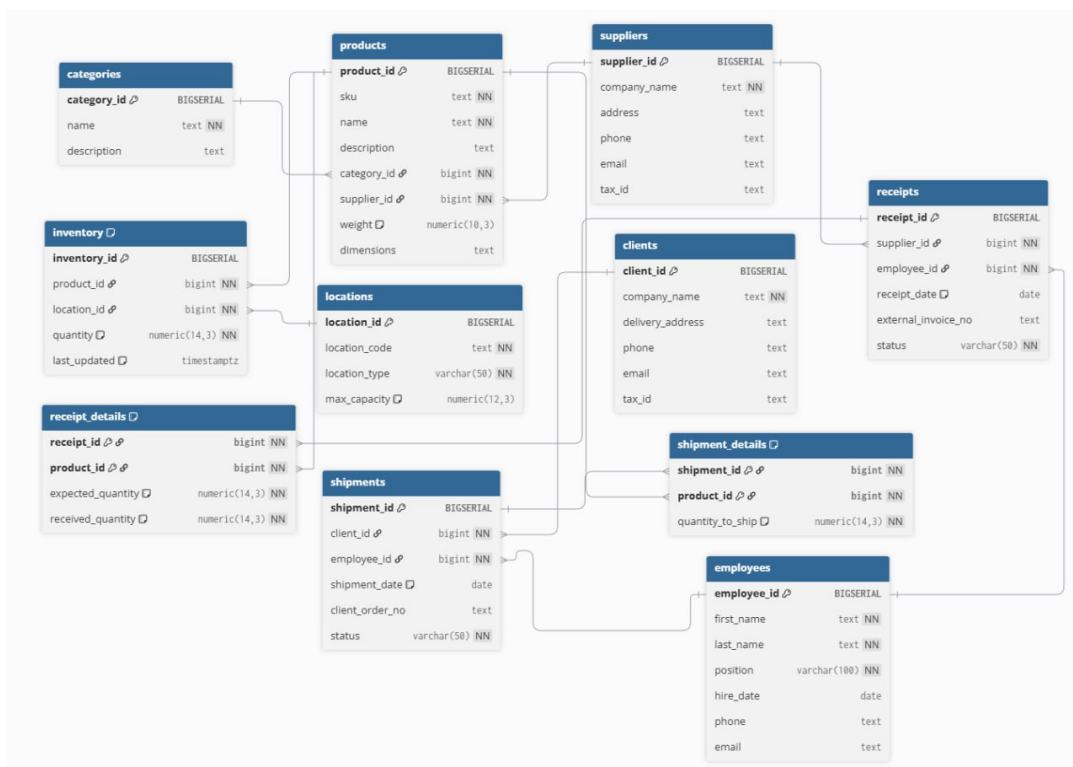
Tabela `receipt_details` zawiera szczegóły przyjęć magazynowych.

- `receipt_id` (`bigint`, `FK` → `receipts.receipt_id`),
- `product_id` (`bigint`, `FK` → `products.product_id`),
- `expected_quantity` (`numeric`, `NOT NULL`) — ilość oczekiwana,
- `received_quantity` (`numeric`) — ilość faktycznie otrzymana.

2.2. Diagram związków encji

Struktura relacyjnej bazy danych systemu zarządzania magazynem została przedstawiona w postaci diagramu związków encji (ERD). Diagram obrazuje encje występujące w systemie oraz relacje pomiędzy nimi, w tym klucze główne oraz obce.

Diagram ERD stanowi graficzne odwzorowanie struktury bazy danych i ułatwia analizę zależności pomiędzy poszczególnymi tabelami.



Rys. 2.1. Diagram związków encji (ERD) systemu zarządzania magazynem

2.3. Powiązania pomiędzy tabelami

Relacje pomiędzy tabelami w systemie zarządzania magazynem zostały zaprojektowane w sposób odzwierciedlający rzeczywiste zależności występujące w procesach magazynowych. Zastosowanie kluczy obcych zapewnia integralność referencyjną danych oraz eliminuje możliwość występowania niespójnych rekordów.

Poniżej przedstawiono najważniejsze relacje pomiędzy encjami systemu:

- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `categories` a tabelą `products`. Jedna kategoria może być przypisana do wielu produktów, natomiast każdy produkt należy do jednej kategorii.
- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `suppliers` a tabelą `products`. Jeden dostawca może dostarczać wiele produktów, natomiast każdy produkt posiada jednego dostawcę.
- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `locations` a tabelą `inventory`. Jedna lokalizacja magazynowa może zawierać wiele rekordów stanów magazynowych.
- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `products` a tabelą `inventory`. Każdy produkt może występować w wielu lokalizacjach magazynowych.

- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `suppliers` a tabelą `receipts`. Jeden dostawca może realizować wiele przyjęć magazynowych.
- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `employees` a tabelą `receipts`. Jeden pracownik może obsługiwać wiele przyjęć towarów.
- Relacja typu wiele do wielu (N:M) pomiędzy tabelą `receipts` a tabelą `products`, zrealizowana za pomocą tabeli pośredniej `receipt_details`. Tabela ta przechowuje informacje o ilościach oczekiwanych oraz faktycznie otrzymanych produktów.
- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `clients` a tabelą `shipments`. Jeden klient może posiadać wiele wydań magazynowych.
- Relacja typu jeden do wielu (1:N) pomiędzy tabelą `employees` a tabelą `shipments`. Jeden pracownik może realizować wiele wydań towarów.
- Relacja typu wiele do wielu (N:M) pomiędzy tabelą `shipments` a tabelą `products`, zrealizowana za pomocą tabeli pośredniej `shipment_details`. Tabela ta przechowuje informacje o ilości wydanych produktów.

Zaprojektowane relacje umożliwiają poprawne odwzorowanie procesów przyjęcia oraz wydania towarów, a także zapewniają spójność i integralność danych w całym systemie.

2.4. Kod realizujący bazę danych

W niniejszym podrozdziale przedstawiono kompletną definicję struktur tabel relacyjnej bazy danych systemu zarządzania magazynem. Zaprezentowane listingi zawierają instrukcje `CREATE TABLE` odpowiedzialne za utworzenie wszystkich encji systemu wraz z kluczami głównymi, kluczami obcymi oraz ograniczeniami integralności.

Listing 2.1. Definicja tabeli `categories`

```
CREATE TABLE categories (  
    category_id BIGSERIAL PRIMARY KEY,  
    name TEXT NOT NULL UNIQUE,  
    description TEXT  
);
```

Listing 2.2. Definicja tabeli `suppliers`

```
CREATE TABLE suppliers (  
    supplier_id BIGSERIAL PRIMARY KEY,  
    company_name TEXT NOT NULL,  
    address TEXT,  
    phone TEXT,  
    email TEXT,  
    tax_id TEXT  
);
```

Listing 2.3. Definicja tabeli `clients`

```
CREATE TABLE clients (  
    client_id BIGSERIAL PRIMARY KEY,  
    company_name TEXT NOT NULL,  
    delivery_address TEXT,
```

```
    phone TEXT,  
    email TEXT,  
    tax_id TEXT  
);
```

Listing 2.4. Definicja tabeli employees

```
CREATE TABLE employees (  
    employee_id BIGSERIAL PRIMARY KEY,  
    first_name TEXT NOT NULL,  
    last_name TEXT NOT NULL,  
    position VARCHAR(100) NOT NULL,  
    hire_date DATE,  
    phone TEXT,  
    email TEXT  
);
```

Listing 2.5. Definicja tabeli locations

```
CREATE TABLE locations (  
    location_id BIGSERIAL PRIMARY KEY,  
    location_code TEXT NOT NULL UNIQUE,  
    location_type VARCHAR(50) NOT NULL,  
    max_capacity NUMERIC CHECK (max_capacity >= 0)  
);
```

Listing 2.6. Definicja tabeli products

```
CREATE TABLE products (  
    product_id BIGSERIAL PRIMARY KEY,  
    sku TEXT NOT NULL UNIQUE,  
    name TEXT NOT NULL,  
    description TEXT,  
    category_id BIGINT NOT NULL,  
    supplier_id BIGINT,  
    weight NUMERIC CHECK (weight >= 0),  
    dimensions TEXT,  
    FOREIGN KEY (category_id) REFERENCES categories(category_id),  
    FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id)  
);
```

Listing 2.7. Definicja tabeli inventory

```
CREATE TABLE inventory (  
    inventory_id BIGSERIAL PRIMARY KEY,  
    product_id BIGINT NOT NULL,  
    location_id BIGINT NOT NULL,  
    quantity NUMERIC NOT NULL CHECK (quantity >= 0),  
    last_updated TIMESTAMPTZ DEFAULT now(),  
    UNIQUE (product_id, location_id),  
    FOREIGN KEY (product_id) REFERENCES products(product_id),  
    FOREIGN KEY (location_id) REFERENCES locations(location_id)  
);
```

Listing 2.8. Definicja tabeli receipts

```
CREATE TABLE receipts (  
    receipt_id BIGSERIAL PRIMARY KEY,
```

```

supplier_id BIGINT NOT NULL,
employee_id BIGINT,
receipt_date DATE DEFAULT CURRENT_DATE,
external_invoice_no TEXT,
status VARCHAR(50) NOT NULL,
FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id),
FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);

```

Listing 2.9. Definicja tabeli receipt_details

```

CREATE TABLE receipt_details (
    receipt_id BIGINT NOT NULL,
    product_id BIGINT NOT NULL,
    expected_quantity NUMERIC NOT NULL CHECK (expected_quantity >= 0),
    received_quantity NUMERIC NOT NULL CHECK (received_quantity >= 0),
    PRIMARY KEY (receipt_id, product_id),
    FOREIGN KEY (receipt_id) REFERENCES receipts(receipt_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

```

Listing 2.10. Definicja tabeli shipments

```

CREATE TABLE shipments (
    shipment_id BIGSERIAL PRIMARY KEY,
    client_id BIGINT NOT NULL,
    employee_id BIGINT,
    shipment_date DATE DEFAULT CURRENT_DATE,
    client_order_no TEXT,
    status VARCHAR(50) NOT NULL,
    FOREIGN KEY (client_id) REFERENCES clients(client_id),
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);

```

Listing 2.11. Definicja tabeli shipment_details

```

CREATE TABLE shipment_details (
    shipment_id BIGINT NOT NULL,
    product_id BIGINT NOT NULL,
    quantity_to_ship NUMERIC NOT NULL CHECK (quantity_to_ship >= 0),
    PRIMARY KEY (shipment_id, product_id),
    FOREIGN KEY (shipment_id) REFERENCES shipments(shipment_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

```

2.5. Normalizacja bazy danych

Proces normalizacji bazy danych został przeprowadzony w celu eliminacji redundancji danych, zapewnienia spójności informacji oraz poprawy integralności logicznej systemu. Projektowana baza danych systemu zarządzania magazynem spełnia założenia trzech pierwszych postaci normalnych: 1NF, 2NF oraz 3NF.

2.5.1. Pierwsza postać normalna (1NF)

Pierwsza postać normalna wymaga, aby:

- wszystkie atrybuty zawierały wartości atomowe,
- w tabelach nie występowały powtarzające się grupy danych,
- każdy rekord był jednoznacznie identyfikowany przez klucz główny.

W zaprojektowanej bazie danych wszystkie tabele posiadają klucze główne (np. `category_id`, `product_id`, `shipment_id`), a każdy atrybut przechowuje pojedynczą, niepodzielną wartość. Nie występują pola wielowartościowe ani listy danych w jednym atrybucie.

2.5.2. Druga postać normalna (2NF)

Druga postać normalna wymaga spełnienia warunków 1NF oraz braku zależności częściowych atrybutów niekluczowych od części klucza głównego.

W bazie danych zależności częściowe zostały wyeliminowane poprzez:

- rozdzielenie danych opisowych do osobnych tabel (np. `categories`, `suppliers`),
- zastosowanie tabel pośrednich z kluczami złożonymi, takich jak `receipt_details` oraz `shipment_details`.

W tabelach posiadających klucz złożony wszystkie atrybuty niekluczowe zależą od całego klucza głównego, a nie od jego fragmentu.

2.5.3. Trzecia postać normalna (3NF)

Trzecia postać normalna wymaga, aby w tabelach nie występowały zależności przechodnie pomiędzy atrybutami niekluczowymi.

W zaprojektowanej bazie danych:

- dane klientów, pracowników, dostawców i produktów zostały umieszczone w oddzielnych tabelach,
- informacje zależne od innych encji są przechowywane wyłącznie poprzez klucze obce,
- nie występują atrybuty, które mogłyby być wyznaczane na podstawie innych atrybutów niekluczowych w tej samej tabeli.

Dzięki temu baza danych spełnia wymagania trzeciej postaci normalnej, zapewniając wysoką spójność danych oraz łatwość dalszej rozbudowy systemu.

2.5.4. Podsumowanie

Zaprojektowana struktura bazy danych systemu zarządzania magazynem została poprawnie znormalizowana do trzeciej postaci normalnej (3NF). Zastosowana normalizacja minimalizuje redundancję danych, ułatwia utrzymanie spójności informacji oraz poprawia wydajność operacji na bazie danych.

3. Zapytania problemowe i ich realizacja

W niniejszym rozdziale przedstawiono zapytania problemowe, które odzwierciedlają rzeczywiste potrzeby biznesowe systemu zarządzania magazynem. Dla każdego problemu zaprezentowano opis zagadnienia, kolejne kroki rozwiązania oraz implementację w postaci funkcji lub procedury składowanej w języku PL/pgSQL.

3.1. Premia dla pracowników za obsłużone wysyłki (zrealizowane w interfejsie użytkownika)

- wyszukanie pracowników realizujących wysyłki w danym miesiącu,
- zliczenie liczby wysyłek dla każdego pracownika,
- wyznaczenie procentu premii na podstawie liczby wysyłek.

Listing 3.1. Funkcja obliczająca premię pracownika

```
CREATE OR REPLACE FUNCTION public.calculate_employee_bonus(  
    p_month INT,  
    p_year INT  
)  
RETURNS TABLE (  
    employee_id BIGINT,  
    shipments_count INT,  
    bonus_percent INT  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    RETURN QUERY  
    SELECT  
        s.employee_id,  
        COUNT(*)::INT AS shipments_count,  
        CASE  
            WHEN COUNT(*) > 100 THEN 30  
            WHEN COUNT(*) BETWEEN 50 AND 100 THEN 15  
            ELSE 0  
        END AS bonus_percent  
    FROM public.shipments s  
    WHERE EXTRACT(MONTH FROM s.shipment_date) = p_month  
        AND EXTRACT(YEAR FROM s.shipment_date) = p_year  
        AND s.employee_id IS NOT NULL  
    GROUP BY s.employee_id;  
END;  
$$;
```

3.2. Automatyczne oznaczanie przeterminowanych wysyłek

- wyszukanie wysyłek starszych niż 7 dni,

- sprawdzenie ich aktualnego statusu,
- aktualizacja statusu na OVERDUE.

Listing 3.2. Procedura oznaczania przeterminowanych wysyłek

```
CREATE OR REPLACE PROCEDURE public.mark_overdue_shipments()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE public.shipments
    SET status = 'OVERDUE'
    WHERE shipment_date < CURRENT_DATE - INTERVAL '7_days'
    AND status <> 'DELIVERED';
END;
$$;
```

3.3. Wykrywanie braków magazynowych

- pobranie aktualnych stanów magazynowych,
- porównanie ilości z progiem minimalnym,
- zwrócenie produktów wymagających uzupełnienia.

Listing 3.3. Funkcja wykrywająca braki magazynowe

```
CREATE OR REPLACE FUNCTION public.find_low_stock_products(
    p_min_quantity NUMERIC
)
RETURNS TABLE (
    product_id BIGINT,
    location_id BIGINT,
    quantity NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        i.product_id,
        i.location_id,
        i.quantity
    FROM public.inventory i
    WHERE i.quantity < p_min_quantity;
END;
$$;
```

3.4. Obliczenie stopnia realizacji dostawy (zrealizowane w interfejsie użytkownika)

- pobranie ilości oczekiwanych i faktycznie odebranych produktów,
- zsumowanie wartości dla całej dostawy,

- obliczenie procentowego stopnia realizacji przyjęcia.

Listing 3.4. Funkcja obliczająca stopień realizacji dostawy

```
CREATE OR REPLACE FUNCTION public.receipt_completion_percentage (
    p_receipt_id BIGINT
)
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$
DECLARE
    expected_sum NUMERIC := 0;
    received_sum NUMERIC := 0;
BEGIN
    SELECT
        COALESCE (SUM(expected_quantity), 0),
        COALESCE (SUM(received_quantity), 0)
    INTO expected_sum, received_sum
    FROM public.receipt_details
    WHERE receipt_id = p_receipt_id;

    IF expected_sum = 0 THEN
        RETURN 0;
    END IF;

    RETURN ROUND((received_sum / expected_sum) * 100, 2);
END;
$$;
```

3.5. Najaktywniejsi klienci w danym okresie

- pobranie wysyłek z wybranego okresu,
- zliczenie liczby wysyłek dla każdego klienta,
- wybranie klientów przekraczających ustalony próg aktywności.

Listing 3.5. Funkcja identyfikująca najaktywniejszych klientów

```
CREATE OR REPLACE FUNCTION public.get_top_clients (
    p_from DATE,
    p_to DATE,
    p_min_shipments INT
)
RETURNS TABLE (
    client_id BIGINT,
    shipments_count INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.client_id,
        COUNT(*)::INT AS shipments_count
    FROM public.shipments s
```

```

WHERE s.shipment_date BETWEEN p_from AND p_to
GROUP BY s.client_id
HAVING COUNT(*) >= p_min_shipments;
END;
$$;

```

3.6. Kontrola pojemności lokalizacji magazynowej

- zsumowanie ilości produktów przypisanych do lokalizacji,
- pobranie maksymalnej pojemności lokalizacji,
- porównanie wartości i zwrócenie wyniku kontroli.

Listing 3.6. Funkcja kontrolująca pojemność lokalizacji

```

CREATE OR REPLACE FUNCTION public.check_location_capacity(
    p_location_id BIGINT
)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
DECLARE
    total_quantity NUMERIC := 0;
    v_max_capacity NUMERIC;
BEGIN
    SELECT COALESCE(SUM(i.quantity), 0)
    INTO total_quantity
    FROM public.inventory i
    WHERE i.location_id = p_location_id;

    SELECT l.max_capacity
    INTO v_max_capacity
    FROM public.locations l
    WHERE l.location_id = p_location_id;

    IF v_max_capacity IS NULL THEN
        RETURN TRUE;
    END IF;

    RETURN total_quantity <= v_max_capacity;
END;
$$;

```

3.7. Aktualizacja stanu magazynu po przyjęciu dostawy

- pobranie danych o faktycznie odebranych ilościach produktów,
- dopasowanie produktów do odpowiednich rekordów magazynowych,
- aktualizacja ilości oraz daty ostatniej modyfikacji.

Listing 3.7. Procedura aktualizująca stany magazynowe po przyjęciu

```

CREATE OR REPLACE PROCEDURE public.update_inventory_after_receipt(

```

```

        p_receipt_id BIGINT
    )
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE public.inventory i
    SET quantity = i.quantity + rd.received_quantity,
        last_updated = NOW()
    FROM public.receipt_details rd
    WHERE rd.receipt_id = p_receipt_id
        AND rd.product_id = i.product_id;
END;
$$;

```

3.8. Ocena produktywności pracowników (zrealizowane w interfejsie użytkownika)

- pobranie danych o wysyłkach realizowanych przez pracowników,
- zliczenie liczby unikalnych klientów,
- określenie poziomu produktywności pracownika.

Listing 3.8. Funkcja oceniająca produktywność pracowników

```

CREATE OR REPLACE FUNCTION public.employee_productivity()
RETURNS TABLE (
    employee_id BIGINT,
    clients_count INT,
    productivity_level TEXT
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.employee_id,
        COUNT(DISTINCT s.client_id)::INT AS clients_count,
        CASE
            WHEN COUNT(DISTINCT s.client_id) > 20 THEN 'HIGH'
            WHEN COUNT(DISTINCT s.client_id) BETWEEN 10 AND 20 THEN 'MEDIUM'
            ELSE 'LOW'
        END AS productivity_level
    FROM public.shipments s
    WHERE s.employee_id IS NOT NULL
    GROUP BY s.employee_id;
END;
$$;

```

3.9. Najczęściej wysyłane produkty (zrealizowane w interfejsie użytkownika)

- pobranie danych dotyczących wysyłanych produktów,

- zsumowanie ilości wysyłek dla poszczególnych produktów,
- wybranie produktów przekraczających ustalony próg.

Listing 3.9. Funkcja identyfikująca najczęściej wysyłane produkty

```
CREATE OR REPLACE FUNCTION public.most_shipped_products (
    p_min_quantity NUMERIC
)
RETURNS TABLE (
    product_id BIGINT,
    total_quantity NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        sd.product_id,
        SUM(sd.quantity_to_ship) AS total_quantity
    FROM public.shipment_details sd
    GROUP BY sd.product_id
    HAVING SUM(sd.quantity_to_ship) >= p_min_quantity;
END;
$$;
```

3.10. Walidacja poprawności dostawy

- pobranie danych dotyczących ilości oczekiwanych i odebranych,
- sprawdzenie poprawności wartości,
- zatwierdzenie lub odrzucenie dostawy.

Listing 3.10. Funkcja walidująca poprawność dostawy

```
CREATE OR REPLACE FUNCTION public.validate_receipt (
    p_receipt_id BIGINT
)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM public.receipt_details
        WHERE receipt_id = p_receipt_id
        AND received_quantity > expected_quantity
    ) THEN
        RETURN FALSE;
    END IF;

    RETURN TRUE;
END;
$$;
```

3.11. Wymagania CRUD systemu

System zarządzania magazynem musi zapewniać pełną obsługę operacji typu CRUD (Create, Read, Update, Delete) dla wszystkich kluczowych encji występujących w bazie danych. Operacje te są niezbędne do prawidłowego funkcjonowania systemu oraz realizacji zapytań problemowych przedstawionych w dalszej części rozdziału.

Create (tworzenie danych)

Operacje tworzenia danych umożliwiają dodawanie nowych rekordów do bazy danych, takich jak:

- nowi klienci i dostawcy,
- produkty oraz ich kategorie,
- lokalizacje magazynowe,
- przyjęcia towarów (receipts) oraz wysyłki (shipments),
- szczegóły przyjęć i wysyłek.

Operacje CREATE realizowane są przy użyciu procedur składowanych, które dodatkowo wykonują walidację danych wejściowych.

Read (odczyt danych)

Operacje odczytu danych umożliwiają pobieranie informacji niezbędnych do analizy stanu magazynu i podejmowania decyzji biznesowych. Obejmują one m.in.:

- przegląd aktualnych stanów magazynowych,
- listę klientów, dostawców i pracowników,
- historię wysyłek i przyjęć towarów,
- dane wykorzystywane w zapytaniach analitycznych.

Operacje READ realizowane są za pomocą funkcji zwracających pojedyncze rekordy lub zbiory danych.

Update (aktualizacja danych)

Operacje aktualizacji danych umożliwiają modyfikację istniejących rekordów w bazie danych, w szczególności:

- zmianę statusów wysyłek i dostaw,
- aktualizację stanów magazynowych,
- korektę danych klientów, produktów i lokalizacji,
- rejestrację faktycznie odebranych ilości towarów.

Operacje UPDATE są realizowane przez procedury składowane, co zapewnia spójność danych i kontrolę poprawności modyfikacji.

Delete (usuwanie danych)

Operacje usuwania danych pozwalają na eliminację nieaktualnych lub błędnie wprowadzonych rekordów, takich jak:

- nieużywane produkty lub kategorie,
- błędnie dodane rekordy magazynowe,
- anulowane wysyłki lub przyjęcia.

Operacje `DELETE` są zabezpieczone mechanizmami integralności referencyjnej, co zapobiega usuwaniu danych powiązanych z innymi encjami systemu.

Spełnienie powyższych wymagań CRUD stanowi podstawę do realizacji zaawansowanych funkcjonalności systemu, w tym zapytań problemowych opisanych w kolejnych podrozdziałach.

4. Przedstawienie powstałej bazy danych i wdrożonych mechanizmów

4.1. Realizacja mechanizmu CRUD

W systemie zarządzania magazynem zaimplementowano mechanizm CRUD (*Create, Read, Update, Delete*), który umożliwia wykonywanie podstawowych operacji na danych przechowywanych w relacyjnej bazie danych PostgreSQL. Mechanizm ten został zrealizowany przy użyciu procedur oraz funkcji składowanych napisanych w języku *PL/pgSQL*.

W celu czytelnego zaprezentowania działania mechanizmu CRUD, jego implementację przedstawiono na przykładzie tabeli **clients**, przechowującej dane klientów obsługiwanych przez system. Dla tej tabeli przygotowano komplet procedur i funkcji umożliwiających dodawanie, odczyt, aktualizację oraz usuwanie danych.

4.1.1. Operacja Create

Operacja *Create* odpowiada za dodawanie nowych rekordów do tabeli **clients**. Zrealizowano ją w postaci procedury składowanej, która przyjmuje dane klienta jako parametry wejściowe. Przed zapisaniem danych wykonywana jest walidacja poprawności najważniejszych pól, takich jak nazwa firmy oraz adres e-mail. Dzięki temu system zapobiega zapisywaniu niepoprawnych danych w bazie.

Listing 4.1. Procedura dodawania klienta

```
CREATE OR REPLACE PROCEDURE public.clients_create(
    IN p_company_name text,
    IN p_delivery_address text,
    IN p_phone text,
    IN p_email text,
    IN p_tax_id text
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF p_company_name IS NULL OR trim(p_company_name) = '' THEN
        RAISE EXCEPTION 'Company_name_cannot_be_empty';
    END IF;

    IF p_email IS NOT NULL AND position('@' IN p_email) = 0 THEN
        RAISE EXCEPTION 'Invalid_email_format:_%', p_email;
    END IF;

    INSERT INTO clients(
        company_name, delivery_address, phone, email, tax_id
    )
    VALUES (
        p_company_name, p_delivery_address, p_phone, p_email, p_tax_id
    );
END;
$$;
```

4.1.2. Operacja Read

Operacja *Read* umożliwia odczyt danych klienta z bazy danych. Została ona zaimplementowana w postaci funkcji składowanej, która na podstawie identyfikatora klienta zwraca kompletny rekord zawierający jego dane. W przypadku próby odczytu nieistniejącego rekordu funkcja zgłasza wyjątek informujący o braku danych.

Listing 4.2. Funkcja odczytu danych klienta

```
CREATE OR REPLACE FUNCTION public.clients_read_one(
    p_client_id bigint
)
RETURNS public.clients
LANGUAGE plpgsql
AS $$
DECLARE
    v_row clients;
BEGIN
    SELECT *
    INTO v_row
    FROM clients
    WHERE client_id = p_client_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Client_with_id_%_not_found', p_client_id;
    END IF;

    RETURN v_row;
END;
$;
```

4.1.3. Operacja Update

Operacja *Update* odpowiada za aktualizację istniejących danych klienta. Została zrealizowana w formie procedury składowanej, która umożliwia modyfikację wybranych pól rekordu. Procedura sprawdza istnienie klienta w bazie danych oraz wykonuje podstawową walidację danych wejściowych przed dokonaniem aktualizacji.

Listing 4.3. Procedura aktualizacji danych klienta

```
CREATE OR REPLACE PROCEDURE public.clients_update(
    IN p_client_id bigint,
    IN p_company_name text,
    IN p_delivery_address text,
    IN p_phone text,
    IN p_email text,
    IN p_tax_id text
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM clients WHERE client_id = p_client_id
    ) THEN
        RAISE EXCEPTION 'Client_with_id_%_does_not_exist', p_client_id;
    END IF;
```



```
UPDATE clients
SET company_name      = COALESCE(p_company_name, company_name),
    delivery_address = COALESCE(p_delivery_address, delivery_address),
    phone             = COALESCE(p_phone, phone),
    email              = COALESCE(p_email, email),
    tax_id             = COALESCE(p_tax_id, tax_id)
WHERE client_id = p_client_id;
END;
$$;
```

4.1.4. Operacja Delete

Operacja *Delete* służy do usuwania danych klienta z systemu. Została ona zaimplementowana w postaci procedury składowanej, która przed usunięciem rekordu sprawdza jego istnienie. Dodatkowo obsługiwana jest sytuacja naruszenia integralności referencyjnej, np. gdy klient jest powiązany z wysyłkami, co uniemożliwia jego usunięcie.

Listing 4.4. Procedura usuwania klienta

```
CREATE OR REPLACE PROCEDURE public.clients_delete(
    IN p_client_id bigint
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM clients WHERE client_id = p_client_id
    ) THEN
        RAISE EXCEPTION 'Client_with_id_%_does_not_exist', p_client_id;
    END IF;

    DELETE FROM clients WHERE client_id = p_client_id;

EXCEPTION
    WHEN foreign_key_violation THEN
        RAISE EXCEPTION
            'Cannot_delete_client_%,_client_is_used_in_shipments',
            p_client_id;
END;
$$;
```

5. Koncepcja dostępu zdalnego do bazy danych

5.1. Założenia architektoniczne

System Zarządzania Magazynem został zrealizowany jako aplikacja desktopowa działająca w architekturze klient–serwer. Aplikacja kliencka została napisana w języku Java i komunikuje się bezpośrednio z serwerem bazy danych PostgreSQL.

Dostęp do bazy danych realizowany jest z wykorzystaniem mechanizmu JDBC (Java Database Connectivity), który umożliwia wykonywanie zapytań SQL oraz wywoływanie procedur i funkcji składowanych.

5.2. Realizacja połączenia z bazą danych

Połączenie aplikacji z bazą danych realizowane jest poprzez klasę odpowiedzialną za konfigurację połączenia JDBC. Wykorzystywany jest adres serwera, numer portu, nazwa bazy danych oraz dane użytkownika.

5.3. Zakres dostępu do aplikacji

Aplikacja została zaprojektowana jako narzędzie administracyjne i przeznaczona jest wyłącznie do użytku przez administratora systemu. Nie przewidziano mechanizmu obsługi wielu ról użytkowników ani logowania końcowych użytkowników.

Administrator posiada pełny dostęp do wszystkich funkcjonalności systemu, w tym do zarządzania danymi magazynowymi, obsługi procesów przyjęć i wysyłek oraz wykonywania zapytań analitycznych. Takie podejście upraszcza architekturę aplikacji oraz umożliwia bezpośrednią kontrolę nad danymi przechowywanymi w bazie.

6. Interfejs użytkownika aplikacji

W niniejszym rozdziale przedstawiono interfejs użytkownika aplikacji zarządzania magazynem. Zaprezentowane ekrany ilustrują najważniejsze funkcjonalności systemu, w tym obsługę danych słownikowych, realizację operacji CRUD oraz wizualizację zapytań problemowych.

6.1. Główne okno aplikacji

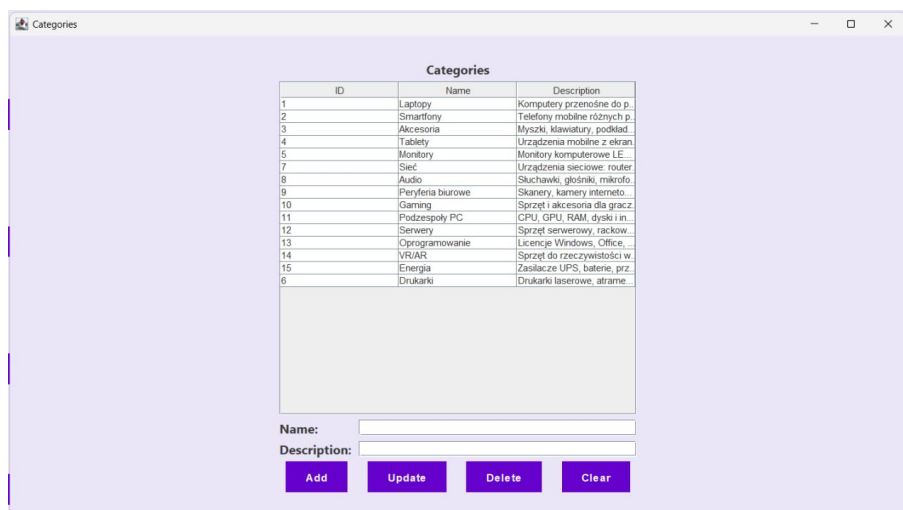
Główne okno aplikacji stanowi centralny punkt dostępu do wszystkich funkcjonalności systemu. Użytkownik ma możliwość przechodzenia do poszczególnych modułów, takich jak zarządzanie kategoriami, klientami, pracownikami, produktami, lokalizacjami magazynowymi oraz realizacja zapytań problemowych.



Rys. 6.1. Główne okno aplikacji

6.2. Zarządzanie kategoriami produktów

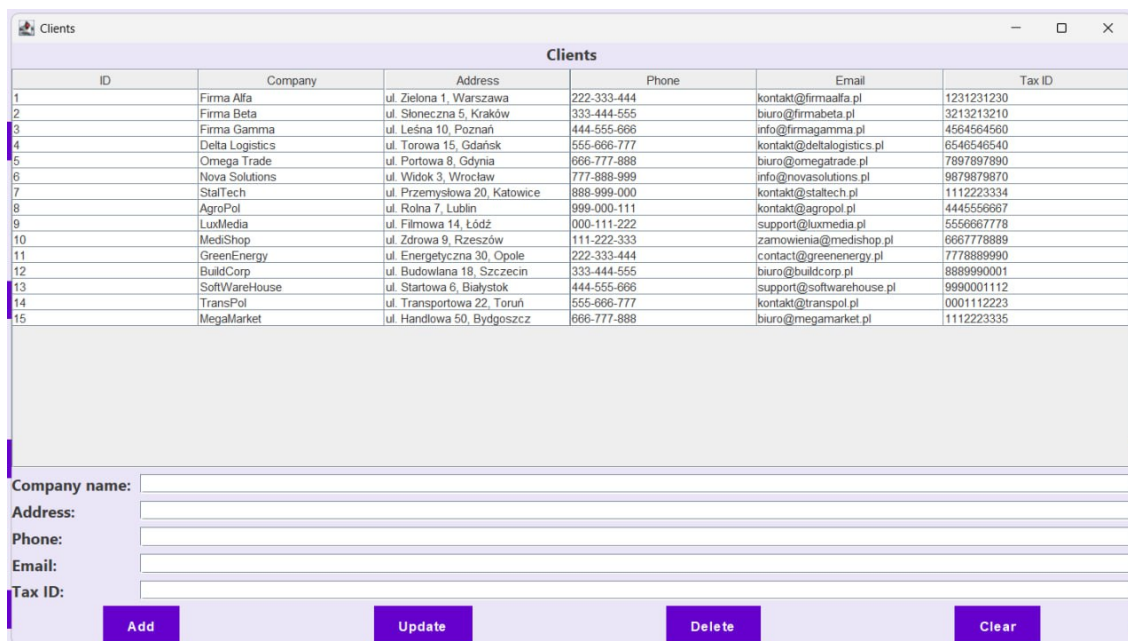
Okno zarządzania kategoriami umożliwia wykonywanie operacji CRUD (Create, Read, Update, Delete) na tabeli kategorii. Użytkownik może przeglądać listę istniejących kategorii, dodawać nowe rekordy, modyfikować istniejące dane oraz usuwać wybrane pozycje.



Rys. 6.2. Okno zarządzania kategoriami

6.3. Zarządzanie klientami

Moduł klientów umożliwia kompleksową obsługę danych kontrahentów. Dostępne są operacje dodawania, edycji oraz usuwania klientów, a także przegląd pełnej listy zapisanej w bazie danych. Dane prezentowane są w formie tabelarycznej, co ułatwia ich analizę i wyszukiwanie.



Rys. 6.3. Okno zarządzania klientami

6.4. Zapytanie problemowe – premia pracowników

Okno premii pracowników umożliwia realizację zapytania problemowego polegającego na obliczeniu wysokości premii w zależności od liczby obsługiwanych wysyłek w wybranym miesiącu i roku. Użytkownik wybiera parametry czasowe, a następnie generuje raport prezentujący wyniki obliczeń zgodnie z ustalonymi progami premiovymi.

Employee Bonus

Month 1 Year 2024 Calculate Bonus

Employee Bonus Report

Select month and year, then click "Calculate Bonus".

Bonus legend

0-49 shipping → 0% | 50-100 shipping → 15% | >100 → 30%

Rys. 6.4. Okno obliczania premii pracowników

6.5. Analiza produktywności pracowników

Na rysunku 6.5 przedstawiono okno aplikacji umożliwiające analizę produktywności pracowników na podstawie liczby obsłużonych klientów. Funkcjonalność ta realizuje zapytanie problemowe, którego celem jest ocena efektywności pracy personelu magazynowego.

Użytkownik może wygenerować raport, klikając przycisk *Calculate Productivity*. Wyniki prezentowane są w formie czytelnego zestawienia tekstowego, zawierającego identyfikator pracownika, liczbę obsłużonych klientów oraz poziom produktywności.

Employee Productivity

Employee productivity analysis based on handled clients Calculate Productivity

Employee Productivity

Click "Calculate Productivity" to generate the report.

Rys. 6.5. Okno analizy produktywności pracowników

6.6. Zarządzanie pracownikami

Na rysunku 6.6 zaprezentowano interfejs zarządzania pracownikami systemu magazynowego. Widok ten realizuje pełny mechanizm CRUD (Create, Read, Update, Delete) dla encji *Employees*.

Górna część okna zawiera tabelę z listą pracowników, obejmującą dane takie jak imię, nazwisko, stanowisko, data zatrudnienia oraz dane kontaktowe. Dolna część formularza umożliwia dodawanie nowych rekordów, edycję istniejących oraz ich usuwanie za pomocą dedykowanych przycisków.

ID	First name	Last name	Position	Hire date	Phone	Email
1	Jan	Kowalski	Magazynier	2021-01-15	111-222-333	jan.kowalski@firma.pl
2	Anna	Nowak	Kierownik Magazynu	2020-03-10	222-333-444	anna.nowak@firma.pl
3	Piotr	Wiśniewski	Specjalista ds. logistyki	2022-06-10	333-444-555	piotr.wisniewski@firma.pl
4	Ewa	Zielńska	Magazynier	2023-02-05	444-555-666	ewa.zielnska@firma.pl
5	Tomasz	Lewandowski	Operator Wózka	2021-11-20	555-666-777	t.lewandowski@firma.pl
6	Karolina	Wójcik	Koordinator ds. wysyłek	2019-09-01	666-777-888	k.wojcik@firma.pl
7	Marcin	Kamiński	Specjalista IT	2020-12-12	777-888-999	m.kaminski@firma.pl
8	Agnieszka	Piotrowska	Magazynier	2022-10-18	888-999-000	a.piotrowska@firma.pl
9	Marek	Mazur	Pracownik biurowy	2023-01-22	999-000-111	m.mazur@firma.pl
10	Natalia	Kaczmarek	Specjalista ds. zakupów	2021-04-14	000-111-222	n.kaczmarek@firma.pl
11	Paweł	Król	Magazynier	2023-03-03	123-111-222	pawel.krol@firma.pl
12	Katarzyna	Sikora	Kontroler jakości	2021-12-11	234-222-333	k.sikora@firma.pl
13	Dominik	Ostrowski	Kierowca	2020-07-07	345-333-444	d.ostrowski@firma.pl
14	Olga	Adamczyk	Analitik logistyki	2019-05-25	456-444-555	o.adamczyk@firma.pl
15	Radosław	Duda	Magazynier	2022-08-09	567-555-666	r.duda@firma.pl

First Name:

Last Name:

Hire Date:

Position:

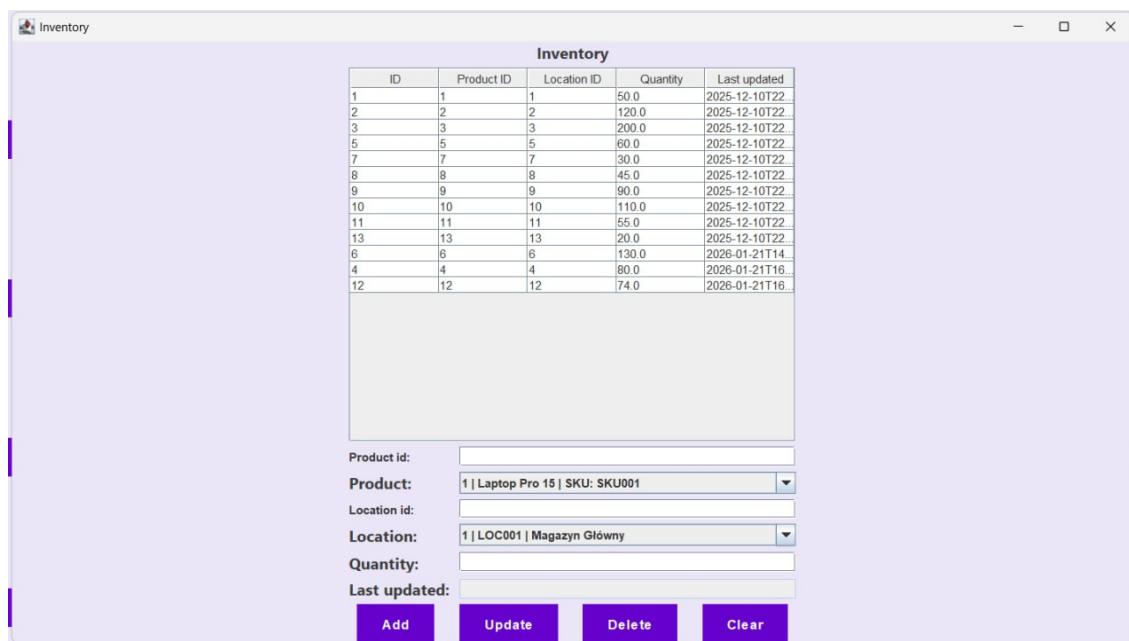
Phone:

Email:

Rys. 6.6. Interfejs zarządzania pracownikami

6.7. Zarządzanie stanami magazynowymi

Widok stanów magazynowych umożliwia przegląd aktualnych ilości produktów przypisanych do poszczególnych lokalizacji magazynowych. Użytkownik może dodawać nowe rekordy, aktualizować istniejące dane oraz usuwać wpisy dotyczące stanów magazynowych.



The screenshot shows a window titled "Inventory" with a table of inventory items and a form for adding or updating items. The table has columns: ID, Product ID, Location ID, Quantity, and Last updated. The form includes fields for Product id, Product (dropdown), Location id, Location (dropdown), Quantity, and Last updated, with buttons for Add, Update, Delete, and Clear.

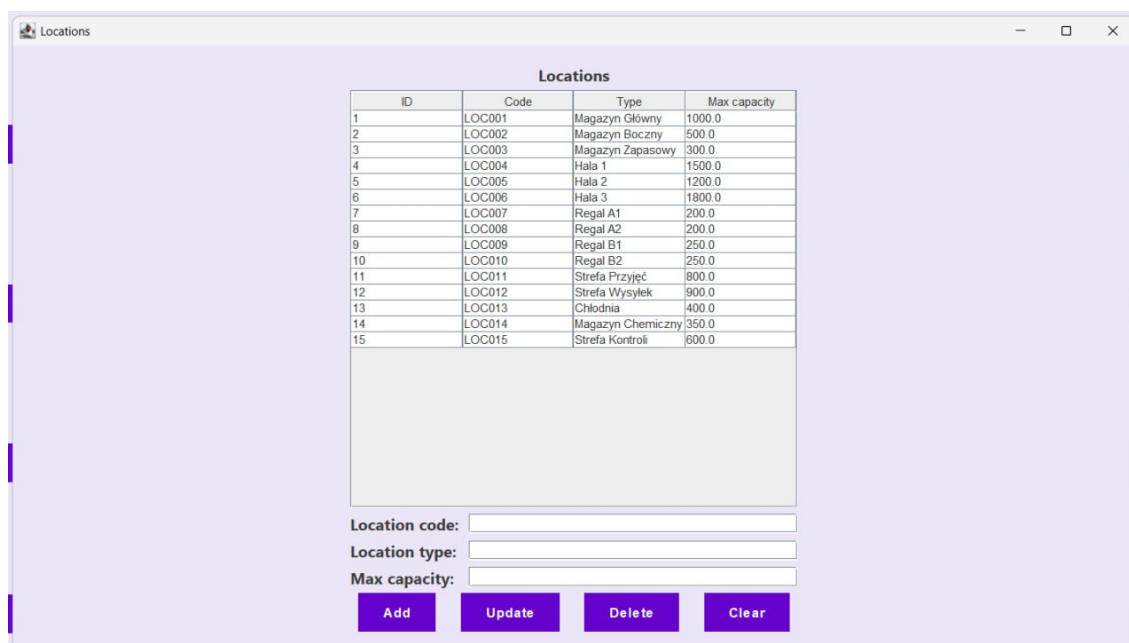
ID	Product ID	Location ID	Quantity	Last updated
1	1	1	50.0	2025-12-10T22
2	2	2	120.0	2025-12-10T22
3	3	3	200.0	2025-12-10T22
5	5	5	60.0	2025-12-10T22
7	7	7	30.0	2025-12-10T22
8	8	8	45.0	2025-12-10T22
9	9	9	90.0	2025-12-10T22
10	10	10	110.0	2025-12-10T22
11	11	11	55.0	2025-12-10T22
13	13	13	20.0	2025-12-10T22
6	6	6	130.0	2026-01-21T14
4	4	4	80.0	2026-01-21T16
12	12	12	74.0	2026-01-21T16

Product id:
Product:
Location id:
Location:
Quantity:
Last updated:

Rys. 6.7. Widok stanów magazynowych

6.8. Zarządzanie lokalizacjami magazynowymi

Widok lokalizacji magazynowych umożliwia zarządzanie strukturą magazynu poprzez definiowanie poszczególnych lokalizacji oraz ich maksymalnej pojemności. Funkcjonalność ta pozwala na kontrolę rozmieszczenia towarów oraz zapewnia poprawne działanie procesów magazynowych.



The screenshot shows a window titled "Locations" with a table of locations and a form for adding or updating locations. The table has columns: ID, Code, Type, and Max capacity. The form includes fields for Location code, Location type, and Max capacity, with buttons for Add, Update, Delete, and Clear.

ID	Code	Type	Max capacity
1	LOC001	Magazyn Główny	1000.0
2	LOC002	Magazyn Boczny	500.0
3	LOC003	Magazyn Zapasowy	300.0
4	LOC004	Hala 1	1500.0
5	LOC005	Hala 2	1200.0
6	LOC006	Hala 3	1800.0
7	LOC007	Regal A1	200.0
8	LOC008	Regal A2	200.0
9	LOC009	Regal B1	250.0
10	LOC010	Regal B2	250.0
11	LOC011	Strefa Przyjęć	800.0
12	LOC012	Strefa Wysyłek	900.0
13	LOC013	Chłodnia	400.0
14	LOC014	Magazyn Chemiczny	350.0
15	LOC015	Strefa Kontroli	600.0

Location code:
Location type:
Max capacity:

Rys. 6.8. Widok lokalizacji magazynowych

6.9. Najczęściej wysyłane produkty

Widok najczęściej wysyłanych produktów umożliwia identyfikację produktów, których łączna ilość wysyłek przekracza zadany próg. Użytkownik wprowadza minimalną wartość ilości, a następnie generuje raport prezentujący produkty spełniające to kryterium.



The screenshot shows a web application window titled "Most Shipped Products". It features a light purple header and footer. In the header, there is a label "Minimum quantity" followed by a text input field containing the value "1000" and a purple "Calculate" button. Below the header, the main content area has a title "Most Shipped Products" and a text instruction "Enter minimum quantity and click 'Calculate'.". The footer contains the text "Shows products with total shipped quantity above the specified threshold."

Rys. 6.9. Zapytanie problemowe – najczęściej wysyłane produkty

6.10. Zarządzanie produktami

Widok zarządzania produktami umożliwia obsługę danych dotyczących asortymentu magazynowego. Użytkownik może dodawać nowe produkty, modyfikować istniejące rekordy oraz usuwać dane produktów. Informacje prezentowane są w formie tabelarycznej, co ułatwia ich przegląd i edycję.

ID	SKU	Name	Description	Category ID	Supplier ID	Weight	Dimensions
1	SKU001	Laptop Pro 15	Laptop 15" i7, 16GB RAM	1	1	2.3	35x24x2 cm
2	SKU002	Smartfon X10	Smartfon 6.5" AMOLED	2	2	0.18	16x7x0.8 cm
3	SKU003	Mysz Gamingowa RGB	Mysz gamingowa 7200 D.	3	3	0.12	12x6x4 cm
4	SKU004	Tablet Max 12	Tablet 12" 8GB RAM, 25.	4	4	0.52	28x19x0.7 cm
5	SKU005	Monitor 27 UltraHD	Monitor 27" 4K HDR	5	5	4.2	62x37x6 cm
6	SKU006	Drukarka Laser A4	Drukarka laserowa mono.	6	6	8.0	40x32x25 cm
7	SKU007	Router WiFi 6 Plus	Dwuzakresowy router Wi.	7	7	0.35	22x15x5 cm
8	SKU008	Sluchawki BassBoost	Sluchawki nauszne z red.	8	8	0.28	18x16x8 cm
9	SKU009	Kamera FullHD	Kamera internetowa 1080.	9	9	0.15	9x5x4 cm
10	SKU010	Klawiatura Mechaniczna	Klawiatura mechaniczna	10	10	0.9	45x15x3 cm
11	SKU011	SSD 1TB NVMe	Dysk SSD PCIe 1TB	11	11	0.04	8x2x0.4 cm
12	SKU012	Serwer RACK 2U	Serwer 2U do szafy rack	12	12	12.5	60x45x10 cm
13	SKU013	Windows 11 Pro	Licencja cyfrowa Window.	13	13	0.001	N/A
14	SKU014	Zestaw VR Pro	Gogle VR + kontrolery	14	14	1.2	35x25x15 cm
15	SKU015	UPS 1200VA	Zasilacz awaryjny 1200VA	15	15	6.5	40x15x20 cm

Sku:
 Name:
 Description:
 Category id:
 Category:
 Supplier id:
 Supplier:
 Weight:
 Dimensions:

Rys. 6.10. Interfejs zarządzania produktami

6.11. Stopień realizacji dostawy

Widok stopnia realizacji dostawy umożliwia obliczenie procentowego poziomu realizacji przyjęcia towaru na podstawie ilości oczekiwanych oraz faktycznie odebranych produktów. Użytkownik podaje identyfikator dostawy, po czym system wykonuje odpowiednie obliczenia.

Receipt ID:

Receipt Completion

—

Status: —

The percentage represents the ratio of received to expected quantities.

Rys. 6.11. Widok obliczania stopnia realizacji dostawy

6.12. Wynik obliczenia stopnia realizacji dostawy

Po wykonaniu obliczeń system prezentuje wynik w postaci procentowej oraz tekstowego statusu realizacji dostawy. Informacja ta pozwala na szybką ocenę kompletności przyjęcia towaru do magazynu.

Receipt ID: Calculate

Receipt Completion

95.00 %

Status: PARTIALLY RECEIVED

The percentage represents the ratio of received to expected quantities.

Rys. 6.12. Wynik obliczenia stopnia realizacji dostawy

6.13. Zarządzanie przyjęciami towaru

Widok przyjęć towaru umożliwia rejestrowanie oraz obsługę dostaw przyjmowanych do magazynu. Użytkownik może dodawać nowe przyjęcia, aktualizować ich dane, usuwać wpisy oraz zarządzać szczegółami dostaw, takimi jak produkty, ilości oczekiwane oraz faktycznie odebrane.

Receipts

ID	Supplier	Employee	Date	Invoice	Status
1	1	1	2025-01-05	FV001	Oczekuje
2	2	3	2025-01-07	FV002	Zrealizowana
3	3	5	2025-01-10	FV003	Oczekuje
5	5	4	2025-01-14	FV005	Zrealizowana
6	6	6	2025-01-15	FV006	Oczekuje

Supplier ID:

Supplier:

Employee ID:

Employee:

Receipt date:

External invoice:

Status:

Add Update Delete Clear

Receipts Details

Product	Expected	Received

Product:

Expected quantity:

Received quantity:

Rys. 6.13. Widok zarządzania przyjęciami towaru

6.14. Zarządzanie wysyłkami

Widok wysyłek umożliwia obsługę procesów wydawania towarów z magazynu do klientów. Użytkownik może rejestrować nowe wysyłki, przypisywać je do klientów oraz pracowników, a także zarządzać listą produktów przeznaczonych do wysyłki.

ID	Client	Employee	Date	Order	Status
1	1	1	2025-02-01	ZAM001	W trakcie
2	2	2	2025-02-02	ZAM002	Zrealizowana
3	3	3	2025-02-03	ZAM003	W trakcie
4	4	4	2025-02-04	ZAM004	W trakcie
5	5	5	2025-02-05	ZAM005	Zrealizowana

Client ID:

Client:

Employee ID:

Employee:

Shipment date:

Client order:

Status:

Shipments Details

Product	Quantity

Product:

Quantity to ship:

Rys. 6.14. Widok zarządzania wysyłkami

6.15. Zarządzanie dostawcami

Widok zarządzania dostawcami umożliwia obsługę danych firm współpracujących z magazynem. Użytkownik może dodawać nowych dostawców, edytować istniejące dane oraz usuwać wpisy z bazy danych, zachowując spójność informacji wykorzystywanych w procesach przyjęć towaru.

ID	Company	Phone	Email
2	Tech Solutions	987-654-321	biuro@techsolutions.pl
3	Gadget World	555-666-777	info@gadgetworld.pl
4	DigitalPoint	222-444-666	kontakt@digitalpoint.pl
5	ElectroHouse	333-555-777	biuro@electrohouse.pl
7	KomputronikX	555-777-999	support@komputronikx.pl
8	HardwareZone	666-888-000	sklep@hardwarezone.pl
10	MegaDevices	888-000-222	info@megadevices.pl
11	SmartParts	999-111-333	zamowienia@smartparts.pl
12	TopComponents	111-333-555	kontakt@topcomponents.pl
13	EuroElectro	222-444-666	biuro@euroelectro.pl
14	NextGenSupply	333-555-777	support@nextgensupply.pl
15	ITWarehouse	444-666-888	kontakt@itwarehouse.pl
9	ProTechSupply	777-999-111	kontakt@protechsupply.pl
1	ABC Electronics	123-456-788	kontakt@abcelectronics.pl
6	MobilePr	444-666-888	kontakt@mobilepro.pl

Company name:

Address:

Phone:

Email:

Tax ID:

Rys. 6.15. Widok zarządzania dostawcami

7. Repozytorium projektu (GitHub)

Kod źródłowy projektu Systemu Zarządzania Magazynem został umieszczony w publicznym repozytorium GitHub. Repozytorium zawiera pełną strukturę projektu, w tym definicje bazy danych, skrypty SQL, kod aplikacji oraz pliki konfiguracyjne.

Repozytorium projektu dostępne jest pod adresem:

<https://github.com/YevhenMarchak/ProjectBazyDanych-2rok->

7.1. Dokumentacja projektu (GitHub)

Dokumentacja projektu została przygotowana w systemie \LaTeX i udostępniona w publicznym repozytorium GitHub. Repozytorium zawiera kompletną strukturę dokumentacji, w tym pliki źródłowe pracy, konfigurację środowiska oraz materiały niezbędne do poprawnej kompilacji dokumentu.

Dokumentacja projektu dostępna jest pod adresem:

<https://github.com/YevhenMarchak/ProjectBazyDanych-2rok->

Bibliografia

- [1] JetBrains. IntelliJ idea documentation. *<https://www.jetbrains.com/idea/documentation/>*, 2025.
- [2] PostgreSQL Global Development Group. Postgresql documentation. *<https://www.postgresql.org/docs/>*, 2024.

Spis rysunków

2.1	Diagram związków encji (ERD) systemu zarządzania magazynem	12
6.1	Główne okno aplikacji	29
6.2	Okno zarządzania kategoriami	30
6.3	Okno zarządzania klientami	30
6.4	Okno obliczania premii pracowników	31
6.5	Okno analizy produktywności pracowników	31
6.6	Interfejs zarządzania pracownikami	32
6.7	Widok stanów magazynowych	33
6.8	Widok lokalizacji magazynowych	33
6.9	Zapytanie problemowe – najczęściej wysyłane produkty	34
6.10	Interfejs zarządzania produktami	35
6.11	Widok obliczania stopnia realizacji dostawy	35
6.12	Wynik obliczenia stopnia realizacji dostawy	36
6.13	Widok zarządzania przyjęciami towaru	36
6.14	Widok zarządzania wysyłkami	37
6.15	Widok zarządzania dostawcami	37

Spis listingów

2.1	Definicja tabeli categories	13
2.2	Definicja tabeli suppliers	13
2.3	Definicja tabeli clients	13
2.4	Definicja tabeli employees	14
2.5	Definicja tabeli locations	14
2.6	Definicja tabeli products	14
2.7	Definicja tabeli inventory	14
2.8	Definicja tabeli receipts	14
2.9	Definicja tabeli receipt_details	15
2.10	Definicja tabeli shipments	15
2.11	Definicja tabeli shipment_details	15
3.1	Funkcja obliczająca premię pracownika	17
3.2	Procedura oznaczania przeterminowanych wysyłek	18
3.3	Funkcja wykrywająca braki magazynowe	18
3.4	Funkcja obliczająca stopień realizacji dostawy	19
3.5	Funkcja identyfikująca najaktywniejszych klientów	19
3.6	Funkcja kontrolująca pojemność lokalizacji	20
3.7	Procedura aktualizująca stany magazynowe po przyjęciu	20
3.8	Funkcja oceniająca produktywność pracowników	21
3.9	Funkcja identyfikująca najczęściej wysyłane produkty	22
3.10	Funkcja walidująca poprawność dostawy	22
4.1	Procedura dodawania klienta	25
4.2	Funkcja odczytu danych klienta	26
4.3	Procedura aktualizacji danych klienta	26
4.4	Procedura usuwania klienta	27