

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA
SPECJALNOŚĆ: SYSTEMY I SIECI KOMPUTEROWE

PRACA DYPLOMOWA
INŻYNIERSKA

Zaawansowany system zarządzania
magazynami

An advanced warehouse management system

AUTOR:
Michał Drelich

PROWADZĄCY PRACĘ:
Dr inż. Jarosław Mierzwa, W4/K9

OCENA PRACY:

Spis treści

1. Wstęp	4
2. Analiza zagadnienia	5
2.1. Istniejące rozwiązania	5
2.1.1. LoMag.....	5
2.1.2. SMART system	5
2.2. Wymagania funkcjonalne	6
2.3. Aspekty prawne	6
3. Opis wykorzystanych technologii	7
3.1. Język programowania C#	7
3.2. MySQL	7
3.3. Visual Studio 2017	8
3.4. iTextSharp	8
3.5. Toolkit	8
4. Przebieg procesu implementacji.....	9
4.1. Baza danych.....	9
4.2. Integracja MySQL z C#	10
5. Prezentacja aplikacji.....	14
5.1. Ekran logowania	14
5.2. Ekran zarządzania magazynami	15
5.3. Główny menadżer magazynu	15
5.4. Ekran zarządzania produktami	16
5.5. Edytor kontrahentów	17
5.6. Edytor dokumentów	17
5.7. Generator dokumentów	18
5.8. Statystyki działalności	19
5.9. Inwentaryzacja.....	20
5.10. Produkcja	21
5.11. Przegląd towarów na dokumentach.....	21
6. Testy aplikacji	23
7. Podsumowanie.....	24
Literatura	25
Spis rysunków	26
Spis listingów	27

1. Wstęp

Rozwój technologiczny w ostatnich latach spowodował wzrost handlu wewnętrznego, jak i międzynarodowego co wymusiło na firmach działających w tej branży zwiększenia ilości przyjmowanych oraz wydawanych towarów. Konsekwencją tego mogą stać się problemy z monitorowaniem ilości towaru na stanie czy zwiększenie częstotliwości pomyłek przy wystawianiu dokumentów. Rozwiązaniem dla tych niedogodności mogą okazać się programy magazynowe mające za zadanie pomóc firmom, które swoją działalnością obejmują składowanie towaru. Takie oprogramowanie przejmując zadania pełnione przez człowieka co korzystnie wpływa na szybkość procesu finalizacji transakcji.

Celem niniejszej pracy jest opracowanie desktopowej aplikacji pozwalającej na zarządzanie wieloma magazynami prowadzonych przez polskie przedsiębiorstwo. Użytkownik będzie miał możliwość dodawać kontrahentów, definiować produkty, tworzyć dokumenty magazynowe oraz przeglądać dostarczone statystyki. Program wykorzystuje system bazodanowy do operowania na dostarczonych danych.

Układ pracy jest następujący:

- Rozdział 2 – przegląd istniejących rozwiązań na rynku
- Rozdział 3 – opis wykorzystanych technologii
- Rozdział 4 – opis procesu implementacji, szczegóły dotyczące bazy danych, przykłady użyte w kodzie
- Rozdział 5 – prezentacja kolejnych okien używanych w aplikacji wraz z ich zastosowaniem
- Rozdział 6 – podsumowanie

2. Analiza zagadnienia

2.1. Istniejące rozwiązania

Aktualnie na rynku można znaleźć kilkanaście programów pozwalających na usprawnianie działania swojego magazynu. Dostępne są one zarówno w formie desktopowej, jak i webowej oraz charakteryzują się zróżnicowanym stopniem zaawansowania. Większość z nich jest płatna, ale niektórzy producenci dają możliwość przetestowania swojego produktu przez określony czas lub na pewną ilość uruchomień.

2.1.1. LoMag

Wydany ponad 10 lat temu program firmy Longint to narzędzie do obsługi firmy handlowej średniej wielkości. Aplikacja pozwala na zarządzanie kilkoma magazynami, tworzenia dla nich produktów lub importowania z innego magazynu. Oprócz obsługi dokumentów magazynowych program daje też możliwość generowania faktur.

Warto zwrócić uwagę na to, że producent wprowadził kilka przydatnych opcji takich jak możliwość korzystania ze skanerów kodów kreskowych, ustawianie alarmów przy zagrożeniu osiągnięcia wcześniej zdefiniowanego stanu minimalnego towaru lub zbliżającym się terminie opłacenia faktury.

Program ten można uznać za dobrze przemyślany i użyteczny, ponieważ jak dotąd zakupiło go ponad 10 000 osób [1].

2.1.2. SMART system

Program wydany przez SOFT-STUDIO został podzielony na kilka mniejszych modułów odpowiedzialnych za różne dziedziny. W ich skład wchodzi moduł magazynu, który daje możliwość tworzenia dokumentów, definiowania ich według własnych potrzeb oraz scalania kilku w jeden. Składowanym produktom można określić termin ważności, co pozwala programowi na ostrzeżenie użytkownika z odpowiednim wyprzedzeniem. Obsługa numerów seryjnych i numerów partii produktów również została wprowadzona [2].

Kolejną możliwością tej aplikacji jest drukowanie zestawień o nadwyżkach i zapotrzebowaniach na dany towar oraz wycenianie wartości magazynu.

2.2. Wymagania funkcjonalne

Opracowana aplikacja ma umożliwiać:

- zarządzanie wieloma magazynami oraz zezwalać na przemieszczanie towarów między nimi
- definiowanie produktów, ustalanie dla nich nazwy, ceny, stawki VAT oraz jednostki
- definiowanie produktów złożonych z innych produktów prostych
- dodawanie i edycję kontrahentów
- tworzenie dokumentów magazynowych, zgodnych z polskim prawem, dotyczących przekazywania towaru
- tworzenie statystyk

2.3. Aspekty prawne

Tworzony program będzie umożliwiać generowanie dokumentów magazynowych. Ich minimalne wymagania co do zawartości wypisane są w ustawie z dnia 29. września 1994 r. [3]. Dowody księgowe dzieli się na dwa typy.

Przychodowe:

- PZ – przyjęcie zewnętrzne
- PW – przyjęcie wewnętrzne
- (MM+) – przesunięcie międzymagazynowe do magazynu

Rozchodowe:

- WZ – wydanie zewnętrzne
- RW – rozchód wewnętrzny
- (MM-) – przesunięcie magazynowe z magazynu

Każdy z wyżej wymienionych rodzajów dokumentów powinien zawierać informację o rodzaju dokumentu i jego numerze, stronach transakcji, wartości transakcji, dacie operacji oraz podpisy osób upoważnionych.

3. Opis wykorzystanych technologii

3.1. Język programowania C#

Wydany w 2002 r. przez firmę Microsoft początkowo bardzo przypominał język Java. Głównym celem jego powstania była chęć utworzenia języka prostego, nowoczesnego oraz zorientowanego obiektowo [4]. Kompilacja programów napisanych w tym języku odbywa się w dwóch etapach. Pierwszym z nich jest tłumaczenie kodu na język pośredni IL (ang. Intermediate Language), który jest wspólny dla wszystkich języków działających w oparciu o platformę .NET. Kod maszynowy produkowany jest dopiero w czasie wykonywania programu przez kompilator JIT (ang. Just In Time) i dotyczy on jedynie fragmentów wymaganych w danym momencie [5].

Język ten został wybrany do zaprojektowania aplikacji głównie z powodu doświadczenia autora oraz ze względu na możliwości, jakie daje. Aby program mógł komunikować się z użytkownikiem stworzono interfejs w oparciu o WPF (ang. Windows Presentation Foundation). Framework ten używa języka znaczników XAML (ang. Extensible Application Markup Language) aby określić jego wygląd i strukturę.

3.2. MySQL

MySQL to system zarządzania bazami danych stworzony przez Oracle Corporation. Jest to otwarte oprogramowanie, które może być dowolnie zmieniane w celu dostosowania do własnych potrzeb. Został zaprojektowany, aby w pełni korzystać z zalet wielowątkowości co pozytywnie wpływa na szybkość wykonywanych zapytań.

Przechowywane dane są w postaci kolumn i wierszy co ułatwia ich organizację. Takie tabele mogą zostać ze sobą powiązane za pomocą relacji typu jeden do jednego, czy jeden do wielu [6].

System ten został wybrany do przechowywania wszystkich informacji potrzebnych w firmie takich jak atrybuty produktów, zawartości magazynów, dane kontrahentów i szczegóły transakcji. Ponadto serwer dla baz danych może być w łatwy sposób założony na jednym komputerze uniezależniając klienta od zewnętrznych hostów. Dodatkowym atutem jest możliwość łatwego zintegrowania z językiem C#.

3.3. Visual Studio 2017

Visual Studio to narzędzie stworzone przez firmę Microsoft do rozwoju oprogramowania. Środowisko przeznaczone jest do pisania programów w takich językach jak C++, C#, Visual Basic. Najważniejszymi funkcjami tego programu jest możliwość edycji kodu, budowanie projektu, jego debugowanie oraz uruchamianie. Dzięki funkcji IntelliSense programista dostaje podpowiedzi dotyczące pisanych linii kodu ograniczając w ten sposób możliwości popełnienia literówek lub zastosowania niewłaściwej struktury danych. Ponadto, nawet po skończeniu pisania, środowisko może proponować zmiany w kodzie w celu jego uproszczenia lub sprawienia by spełniał konwencje obowiązujące w danym języku.

Kolejną funkcją dostarczającą przez VS są pakiety NuGet, czyli narzędzie do tworzenia i dzielenia się kodem z innymi programistami. W przypadku zainstalowania takiej paczki, do projektu zostaje dołączona biblioteka, z której od razu można korzystać.

3.4. iTextSharp

iTextSharp to biblioteka pozwalająca na wygenerowanie plików PDF. Pozwala ona na dowolne ułożenie tekstu w pliku oraz na manipulację jego ustawieniami takimi jak rozmiar czcionki i pogrubianie tekstu.

W przygotowywanej aplikacji użyto tej biblioteki do generowania dokumentów magazynowych. Wybrano ją dlatego, że pozwala na pisanie tekstu niezaczynającego się od krawędzi dokumentu co przy wypisywaniu danych firmy oraz kontrahentów nadało dokumentom estetyczności. Ponadto, skorzystano z możliwości rysowania tabel, które w przejrzysty sposób pozwalają na przedstawienie użytkownikom informacji o produktach i pełnej wartości transakcji.

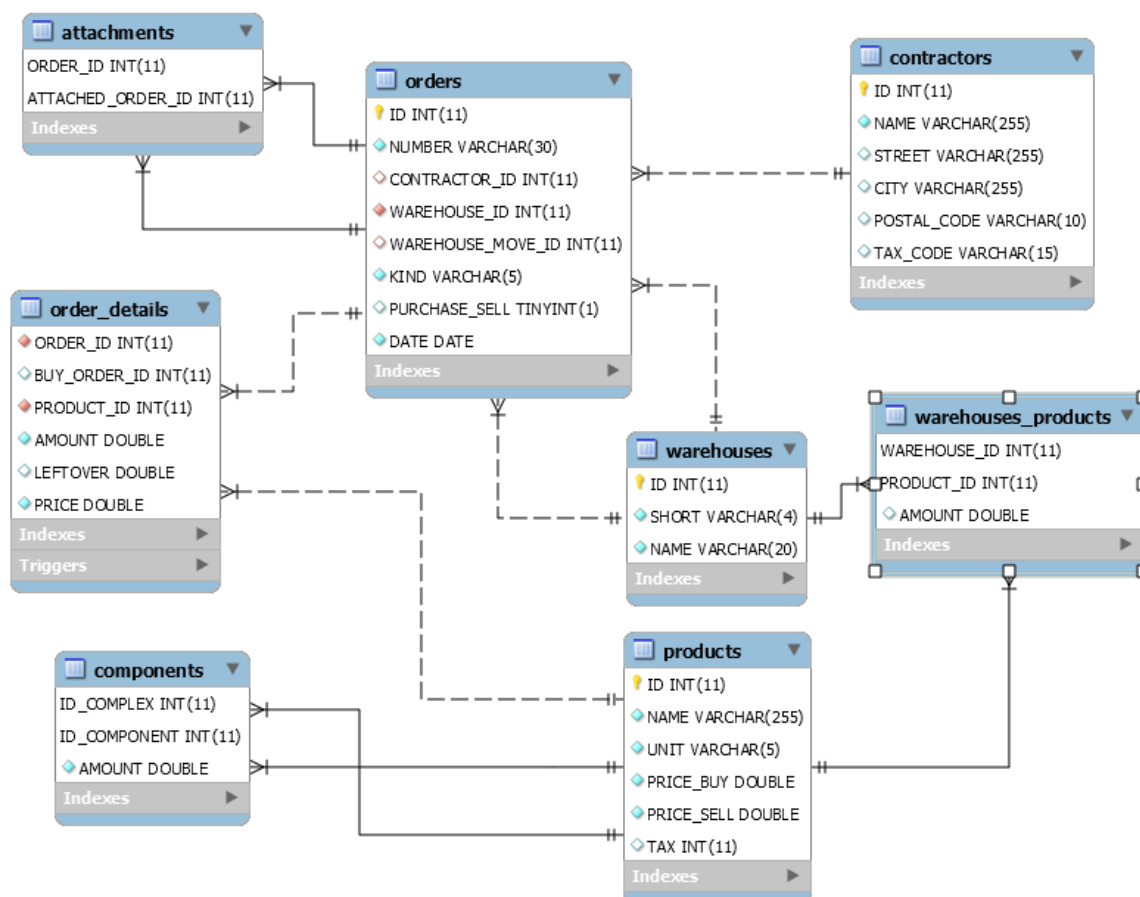
3.5. Toolkit

Toolkit to zestaw dodatkowych narzędzi dla programisty, który rozszerza i dodaje nowe funkcjonalności do jego projektu. W przygotowywanej aplikacji wykorzystano tę bibliotekę do rysowania grafów służących do przedstawienia statystyk dotyczących firmy.

4. Przebieg procesu implementacji

4.1. Baza danych

Wszystkie dane wykorzystywane dotyczące produktów, kontrahentów, zamówień, magazynów czy samej nawet samej firmy zapisywane są w bazie danych.



Rys. 4.1: Diagram ERD tworzonej bazy danych

Na powyższym rysunku pokazany jest diagram bazy o której działanie będzie opierać się cały program.

Tabela *products* zawiera informacje o produktach, którymi operuje firma. Atrybutami dla tej encji jest identyfikator, nazwa produktu, jednostka, w której ilość produktu jest mierzona, sugerowane ceny netto zakupu i sprzedaży oraz stawka podatku. Produkt może mieć dodatkową cechę dotyczącą swojej receptury. Jeżeli dany produkt składa się z innych produktów w magazynie, czyli jest tak zwanym produktem złożonym, informacje o tym będą zawarte w tabeli *components*. Tabela ta, zawiera indeksy produktów złożonych, indeksy towarów składowych oraz ich ilość potrzebną do receptury. Oznacza to, że im więcej różnych

składników dany produkt złożony posiada, tym więcej rekordów w tabeli *components* się pojawi.

Tabela *warehouses* dotyczy magazynów, które są w posiadaniu danej firmy. Jej atrybutami jest identyfikator magazynu, nazwa oraz jej skrót. W rzeczywistości, między produktami a magazynami zachodzi relacja wiele do wielu, tzn. każdy magazyn może przechowywać wiele różnych produktów oraz każdy produkt może znaleźć się w kilku magazynach. Z tego powodu w bazie danych znalazła się tabela łącznikowa *warehouses_products*, która przechowuje informacje o ilości danego produktu w danym magazynie.

Encja *contractors* zawiera informacje o stronach, które mogą występować na dokumentach, czyli kontrahentach i prowadzonej firmy. Tabela może przechować informacje takie jak identyfikator, nazwa, ulica, miasto, kod pocztowy i numer NIP.

Ostatnie trzy tabele służą do obsługi dokumentów. Pierwsza z nich - *orders* przechowuje informacje o numerze zamówienia, rodzaju dokumentu i dacie a także magazynu, którego dotyczy. Stroną, która będzie rozumiana jako dostawca w transakcjach typu przychodowych lub odbiorca w rozchodowych może być jednocześnie tylko kontrahent lub inny magazyn. Atrybut *purchase_sell* wykorzystywany będzie przez wyzwalacz do automatycznego dodawania lub usuwania produktów z magazynów w zależności od tego, czy jest to transakcja przychodowa czy rozchodowa. Tabela *order_details* jest kolejną tabelą łącznikową dla zamówień i produktów. Każdej transakcji przypisuje towar z określeniem jego ilości. Pole *buy_order_id* służy dokumentom wychodzącym na oznaczenie, z jakiej partii towaru korzystają. Z kolei dokumenty przychodzące korzystają z pola *leftover*, w którym zapamiętywana jest ilość niewydanego towaru z danej partii. Tabela *attachments* pokazuje relacje między dokumentami.

4.2. Integracja MySQL z C#

Do zarządzania bazą danych stworzono klasę *SqlHandler*, której metody pozwalają na modyfikowanie jej zawartością.

Listing 4.1: Metody do komunikacji z bazą

```
public DataSet ExecuteCommand(String command)
{
    var dataSet = new DataSet();
    if (Connect())
    {
        var dataAdapter = new MySqlDataAdapter(command, connection);
        dataSet = new DataSet();
        try
        {
            dataAdapter.Fill(dataSet);
        }
        catch (Exception e)
        {
            MessageBox.Show(e.Message, "Błąd", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}
```

```

        Disconnect();
    }
    return dataSet;
}

private bool Connect()
{
    BuildConnection();
    try
    {
        connection.Open();
        return true;
    }
    catch (Exception e)
    {
        MessageBox.Show("Polaczenie sie nie udalo (" + e.Message + ")", "Błąd",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
}

private void BuildConnection()
{
    var builder = new MySqlConnectionStringBuilder()
    {
        Server = Server,
        UserID = Uid,
        Password = Password,
        Database = Database,
        ConnectionTimeout = timeout,
        CharacterSet = charset,
        AllowUserVariables = true,
    };

    connection = new MySqlConnection(builder.ToString());
}

```

Na listing 4.1. pokazane są trzy funkcje używane do wywoływania komend SQL na serwerze. Funkcja *ExecuteCommand* przeznaczona jest dla poleceń typu *SELECT*, ponieważ ich wynikiem są tabele. Efektem całej tej funkcji jest obiekt typu *DataSet*, który w późniejszym etapie projektu będzie wykorzystywany do prezentacji otrzymanych wyników. Metoda *BuildConnection* przygotowuje ustawienia połączenia. Pierwsze cztery właściwości podawane są przez użytkownika zaraz po uruchomieniu programu. Kolejne zaś, są stałe. System kodowania ustawiany jest na utf8, aby można było operować na polskich znakach a następnie umożliwiające jest tworzenie i używanie własnych zmiennych we wszystkich poleceniach. Jest to przydatna opcja, jeżeli wartości zwracane przez jedną kolumnę zależą od wyniku innej. Dla poleceń typu *INSERT*, *UPDATE*, *DELETE* napisano bardzo podobną funkcję do *ExecuteCommand* - *ExecuteNonQuery*. Jej kod nie został zamieszczony, ponieważ ich jedyną różnicą jest to, że zwraca wartość logiczną oznaczającą, czy jakiegokolwiek rekordy w tablicach zostały zmienione.

Przy tak przygotowanych metodach, aby wpłynąć na zawartość bazy wystarczy przygotować komendę i wywołać odpowiednią funkcję. W tym celu przygotowano kilkadziesiąt prostych funkcji, których jedynym zadaniem jest zwrócenie odpowiedniego zapytania. Poniżej znajduje się przykładowa funkcja z komendą tworzącą tabelę

kontrahentów. Warto zauważyć, że jest to jedna z kilku funkcji wywoływanych przy zalogowaniu na podany przez użytkownika serwer w ramach przygotowania programu do działania.

Listing 4.2: Funkcja zwracająca komendę tworzenia tablicy

```
public static String ContractorsTable()
{
    return
        "CREATE TABLE IF NOT EXISTS `contractors` (" +
        "ID int PRIMARY KEY AUTO_INCREMENT," +
        "NAME varchar(255) NOT NULL UNIQUE," +

        "STREET varchar(255)," +
        "CITY varchar(255)," +
        "POSTAL_CODE varchar(10)," +
        "TAX_CODE varchar(15)" +
        ");";
}
```

Tworzona aplikacja musi dostosowywać się do działań użytkownika, dlatego nie zawsze da się w pełni opracować zapytanie, które wykona polecenie użytkownika. W takim przypadku metoda tworzenia komend za pomocą funkcji okazuje się bardzo przydatna, ponieważ można zmienić jej treść nie znając wcześniej intencji klienta.

Listing 4.3: Tworzenie zapytania o produkty podlegające pod dany dokument

```
public static String ShowProductsInDocument(String orderNumber)
{
    return
        $"SELECT p.name, p.unit, p.tax, p.price_buy, p.price_sell, details.amount FROM " +
        "products p " +
        $"INNER JOIN order_details details ON product_id = p.id " +
        $"INNER JOIN orders ord ON details.order_id = ord.id " +
        $"WHERE ord.number = '{orderNumber}';";
}
```

W celu zmniejszenia ilości kodu w kodzie źródłowym aplikacji, do bazy danych został dodany wyzwalacz (ang. trigger), który po wprowadzeniu do tabeli *order_details* nowego wiersza, automatycznie zmieni ilość towaru w danym magazynie.

Listing 4.4: Tworzenie wyzwalacza dla wprowadzanych szczegółów zamówienia

```
public static String CreateUpdateWarehousesProductTrigger()
{
    return
        "DROP TRIGGER IF EXISTS updateWarehouses;" +
        "CREATE TRIGGER updateWarehouses AFTER INSERT ON order_details " +
        "FOR EACH ROW " +
        "BEGIN " +
        "    DECLARE war_id int; " +
        "    DECLARE doctype bool; " +
        "    SELECT orders.purchase_sell INTO doctype FROM orders INNER JOIN order_details
            ON order_details.order_id = orders.id WHERE order_details.order_id =
            new.order_id GROUP BY orders.purchase_sell; " +
        "    SELECT o.warehouse_id INTO war_id FROM orders o INNER JOIN order_details o_d
            ON o.id = o_d.ORDER_ID WHERE o_d.ORDER_ID = new.order_id GROUP BY
            o.warehouse_id; " +
        "    IF(doctype = '0') THEN " +
        "        IF(EXISTS(SELECT product_id FROM warehouses_products w_p WHERE
            w_p.PRODUCT_ID = NEW.product_id AND warehouse_id = war_id)) THEN " +
        "            UPDATE warehouses_products w_p SET w_p.amount = w_p.amount + new.amount
            WHERE w_p.product_id = new.product_id AND w_p.warehouse_id = war_id; " +
        "        ELSE " +
        "            INSERT INTO warehouses_products(warehouse_id, product_id, amount) VALUES
            (war_id, new.product_id, new.amount); " +
        "        END IF ;" +
        "    ELSE " +
        "        UPDATE warehouses_products w_p SET w_p.amount = w_p.amount - new.amount
            WHERE w_p.product_id = new.product_id AND w_p.warehouse_id = war_id; " +
        "    END IF; " +
        "END$$";
}
```

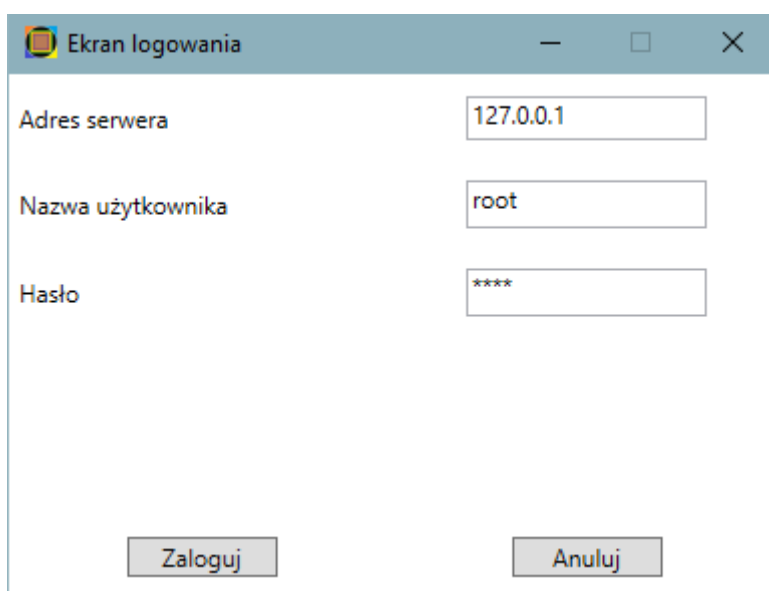
Analizując listing 4.4. widać, że na początku komendy następuje próba usunięcia wyzwalacza a następnie jego ponowne utworzenie, dzieje się tak, ponieważ dla triggerów, w odróżnieniu od tabel, nie istnieje komenda *CREATE IF NOT EXISTS* [6]. Jego działanie można podzielić na dwa etapy. Początkowo następuje próba znalezienia identyfikatora magazynu, którego dotyczy zamówienie wprowadzone do *order_details* oraz ustalenie czy jego zadaniem będzie stan towaru zwiększyć, czy zmniejszyć. Następnie, na podstawie ustalonych wartości, skrypt może zaktualizować stan produktów w magazynie lub stworzyć dla niego nową krotkę, w przypadku jego pierwszorazowego wystąpienia. Warto zwrócić uwagę na znaki „\$\$” kończące cały skrypt. Są one potrzebne, ponieważ w tym przypadku pełnią rolę średnika, który w zwykłym poleceniu oznacza jego zakończenie. Jako że wyzwalacz składa się z kilku mniejszych komend zaistniała potrzeba chwilowej zmiany tego ustawienia. Jest to wyjątkowa sytuacja, dlatego w C# wykorzystano klasę *MySqlScript*, która umożliwia zmianę tego ogranicznika (ang. delimiter).

5. Prezentacja aplikacji

Aplikacja została stworzona w oparciu o framework WPF, który pozwala na rozdzielenie kodu odpowiedzialnego za operację na danych oraz znaczników odpowiedzialnych za wyznaczanie wyglądu danego okna.

5.1. Ekran logowania

Ekran logowania to pierwsze okno dostępne dla użytkownika po uruchomieniu programu. To właśnie w tym miejscu klient ma możliwość zalogowania się do dowolnego serwera MySQL.



The screenshot shows a Windows-style window titled "Ekran logowania". It has a standard title bar with minimize, maximize, and close buttons. The main area contains three labels on the left and corresponding text input boxes on the right. The first label is "Adres serwera" and the box contains "127.0.0.1". The second label is "Nazwa użytkownika" and the box contains "root". The third label is "Hasło" and the box contains four asterisks "****". At the bottom of the window, there are two buttons: "Zaloguj" on the left and "Anuluj" on the right.

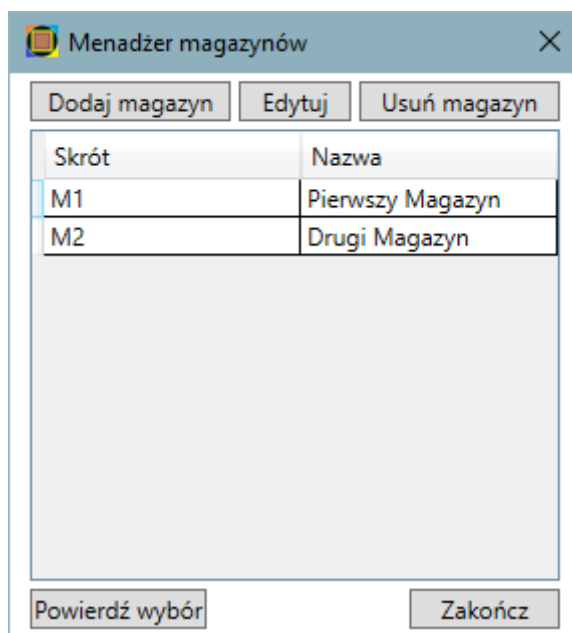
Rys. 5.1: Ekran logowania

Jeżeli, próba skomunikowania się z serwerem zakończyła się sukcesem, wygląd powyższego okna zostanie zmieniony i pojawią się na nim dodatkowa opcja wyboru bazy danych, ale też pozwoli na założenie całkowicie nowej. W przypadku wykrycia braku jakiejkolwiek bazy, aplikacja wymusza na użytkowniku stworzenie nowej. Proces zakładania bazy danych jest bardzo łatwy, ponieważ polega jedynie na podaniu unikatowej nazwy składających się z dużych i małych liter alfabetu łacińskiego [7] oraz wybraniu metody usuwania towarów z magazynu (FIFO lub LIFO) i metody generowania numerów dokumentów (miesięczna lub roczna).

Wybranie serwera i bazy danych dostarcza aplikacji wystarczającą ilość informacji, aby utworzyć obiekt klasy *SqlHandler*, opisany w rozdziale 4.2. który będzie od tego momentu wykorzystywany przez wszystkie inne okna. Na wypadek, gdyby wybrana baza była nowa, wywoływane są funkcje tworzące tabele z rozdziału 4.1.

5.2. Ekran zarządzania magazynami

Jeżeli poprzedni etap zakończył się powodzeniem, program pokaże ekran do zarządzania magazynami. W tym miejscu użytkownik może dodawać nowe magazyny, zmieniać informacje o nich oraz je usuwać.

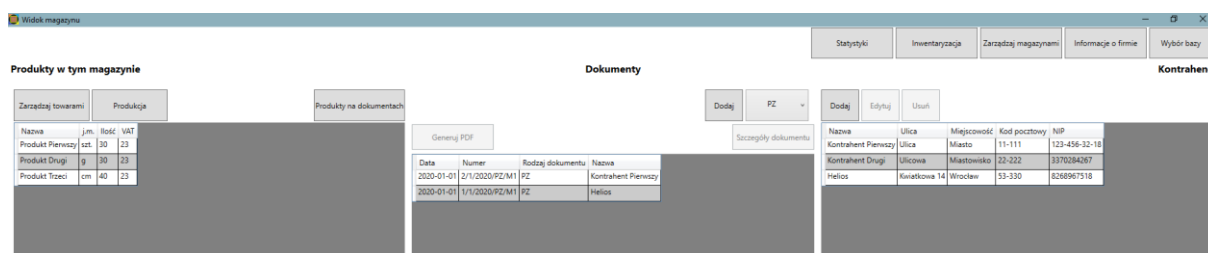


Rys. 5.2: Ekran zarządzania magazynami

W późniejszym etapie użytkowania będzie możliwość powrotu do tego okna w celu zmiany wybranego magazynu.

5.3. Główny menadżer magazynu

Menadżer magazynu to okno, które wypełnione jest informacjami dotyczącymi zarówno wybranego budynku, jak i całej firmy. Z tej pozycji możliwe jest skorzystanie z pozostałych funkcjonalności omawianej aplikacji.



Rys. 5.3: Główny menadżer magazynu

Jak widać na rysunku 5.3, został on podzielony na trzy części zajmującymi się odrębnymi dziedzinami.

Lewą stronę zajmuje panel zarządzający produktami. W podlegającej mu tabeli wyświetlane są informacje o produktach aktualnie znajdujących się w magazynie. Ponadto, znajdujący się tam przycisk „Zarządzaj” otwiera okno pozwalające definiować produkty oraz edytować te znane już wcześniej w firmie. Przycisk „Produkcja” otworzy okno dające możliwość definiowania produktów złożonych jak i ich kompletację/dekompletację. Ostatni przycisk – „Produkty na dokumentach” otwiera okno pozwalające wyfiltrować dokumenty, na których znajduje się wybrany towar.

Środkowa strona dotyczy dokumentów magazynowych oraz transakcji. W znajdującej się tam tabeli wypisane są wszystkie obroty towarem, których jedną ze stron był wybrany magazyn. W tym miejscu możliwe jest tworzenie nowych dokumentów oraz przeglądanie szczegółów tych już utworzonych.

Prawa strona pokazuje znanych firmie kontrahentów oraz pozwala na zarządzanie nimi. Nad tym panelem znajdują się przyciski ogólnego przeznaczenia. „Wybór bazy” ponownie pokazuje okno opisane w rozdziale 5.1. a „Zarządzaj magazynami” to z rozdziału 5.2.

5.4. Ekran zarządzania produktami

Rysunek 5.4. przedstawia narzędzie używane do zarządzania zwykłymi produktami. Zostało ono zaprojektowane tak, aby umożliwić użytkownikowi jednocześnie dodawanie nowych pozycji (prawa strona ekranu) oraz edycję tych już wprowadzonych. Aby wprowadzić zmiany w towarze, należy wybrać jego pozycję z tabeli produktów znajdującej się na środku, zmienić potrzebne wartości a następnie je zaakceptować.

Kontrolka użyta po lewej i prawej stronie tego ekranu jest używana również w przypadku definiowania produktu złożonego. W takim przypadku, pojawia się dodatkowa zakładka „Składniki” pozwalająca na dobór receptury.

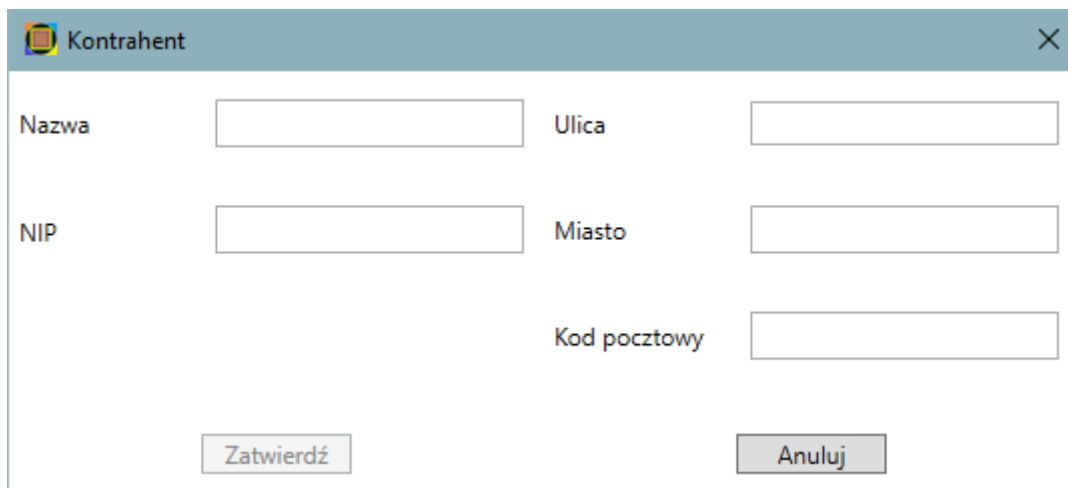
The screenshot shows a software window titled "Okno produktów" (Product Window). It is divided into three main sections:

- Edycja produktu (Edit product):** Located on the left, it contains fields for "Nazwa" (Name), "Jednostka" (Unit), "Stawka VAT" (VAT rate), and "Sugerowane ceny" (Suggested prices) with sub-fields for "Cena zakupu netto" (Net purchase price), "Cena zakupu brutto" (Gross purchase price), "Cena sprzedaży netto" (Net selling price), and "Cena sprzedaży brutto" (Gross selling price). There is an "Akceptuj" (Accept) button at the bottom.
- Przegląd produktów (Product overview):** A central table with columns: "Nazwa", "j.m." (unit), "VAT", "Zakup netto" (Net purchase), and "Sprzedaż netto" (Net selling). The table lists three products: "Produkt Pierwszy" (szt., 23, 1, 2), "Produkt Drugi" (g, 23, 3, 4), and "Produkt Trzeci" (cm, 23, 5, 6). Below the table is a large greyed-out area and a "Usuń produkt" (Delete product) button.
- Nowy produkt (New product):** Located on the right, it has the same form fields as the "Edycja produktu" section, including "Nazwa", "Jednostka", "Stawka VAT", and price fields, with an "Akceptuj" button and a "Zakończ" (Finish) button at the bottom.

Rys. 5.4: Ekran zarządzania produktami

5.5. Edytor kontrahentów

Okno edytora kontrahentów wywoływane jest w razie potrzeby utworzenia nowych odbiorców oraz edycji tych już istniejących. Ponadto, służy do wprowadzania informacji o prowadzonej firmie, której dane będą wykorzystywane przy generowaniu dokumentów.



Kontrahent

Nazwa Ulica

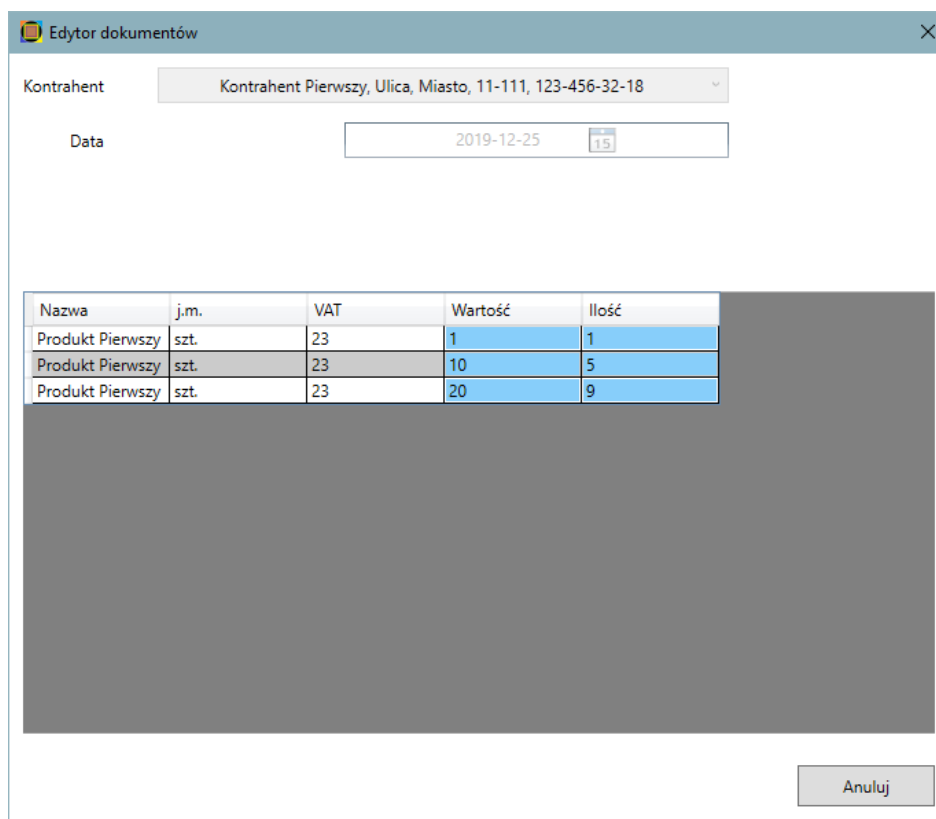
NIP Miasto

Kod pocztowy

Zatwierdź Anuluj

Rys. 5.5: Edytor kontrahentów

5.6. Edytor dokumentów



Edytor dokumentów

Kontrahent: Kontrahent Pierwszy, Ulica, Miasto, 11-111, 123-456-32-18

Data: 2019-12-25

Nazwa	j.m.	VAT	Wartość	Ilość
Produkt Pierwszy	szt.	23	1	1
Produkt Pierwszy	szt.	23	10	5
Produkt Pierwszy	szt.	23	20	9

Anuluj

Rys. 5.6: Edytor dokumentów

Do tworzenia dokumentów magazynowych służy okno pokazane na rysunku 5.6. Pozwala ono na wybranie jednej firmy zapisanej w bazie programu lub dla dokumentów typu przesunięcia międzymagazynowe – inny magazyn prowadzony przez przedsiębiorstwo. Kolejna dana to data transakcji. Poniżej znajduje się tabela, do której można dodawać kolejne produkty, których ilość ma zostać zmieniona.

Prezentowany przykład pokazuje szczegóły już zaakceptowanego dokumentu wychodzącego, dlatego mogą się na nim zwiłokrotnione wiersze z produktami w przypadku, gdy wydawany towar został pobrany z różnych partii.

5.7. Generator dokumentów

Dokument WZ nr 0007/2019

Magazyn: Pierwszy Magazyn

Data wystawienia dokumentu: 2019-11-24

Moja Firma sp. z o.o.

Krakowska 33

50-045 Wrocław

NIP: 7967757074

Kontrahent:

Firma Dąbrowska

Kierunkowa 7

00-021 Warszawa

NIP: 4174417689

Lp.	Nazwa	j.m.	Ilość	Cena netto	Wartość netto	VAT	Kwota VAT	Wartość brutto
1.	Jabłka	kg	5	4	20	23	4,6	24,6
2.	Rabarbar	kg	70	7	490	23	112,7	602,7
3.	Wiśnie	kg	65	7	455	23	104,65	559,65
					Netto	%	VAT	Brutto
					965	23	221,95	1186,95
					965		221,95	1186,95

Wartość dokumentu: 1186,95 zł

.....
Podpis osoby uprawnionej do wystawienia dokumentu

.....
Podpis osoby uprawnionej do odbioru dokumentu

Rys. 5.7: Przykład generowanego dokumentu magazynowego

Aby użytkownik mógł drukować tworzone w poprzednim rozdziale dokumenty, postanowiono wprowadzić tę możliwość do aplikacji pośrednio – poprzez wygenerowania dokumentów PDF. Informacje na nim zawarte to rodzaj dokumentu i kolejny numer w danym roku, informacja o dacie transakcji oraz nazwa magazynu źródłowego (lub docelowego dla kupna). Kolejną część zajmują wypisane dane obu stron uczestniczących w operacji. Poniżej znajduje się tabelka z rozpisanymi produktami, ich ceną oraz wartością.

Dodatkowo została także podana wyliczona wartość brutto. Następnie wypisywane jest podsumowanie dotyczące wartości rozważanego towaru a pod nim wyznaczane są miejsca na odrębne podpisy reprezentantów stron. Przykładowy tekst, tak wygenerowanego dokumentu znajduje się na rysunku poniżej.

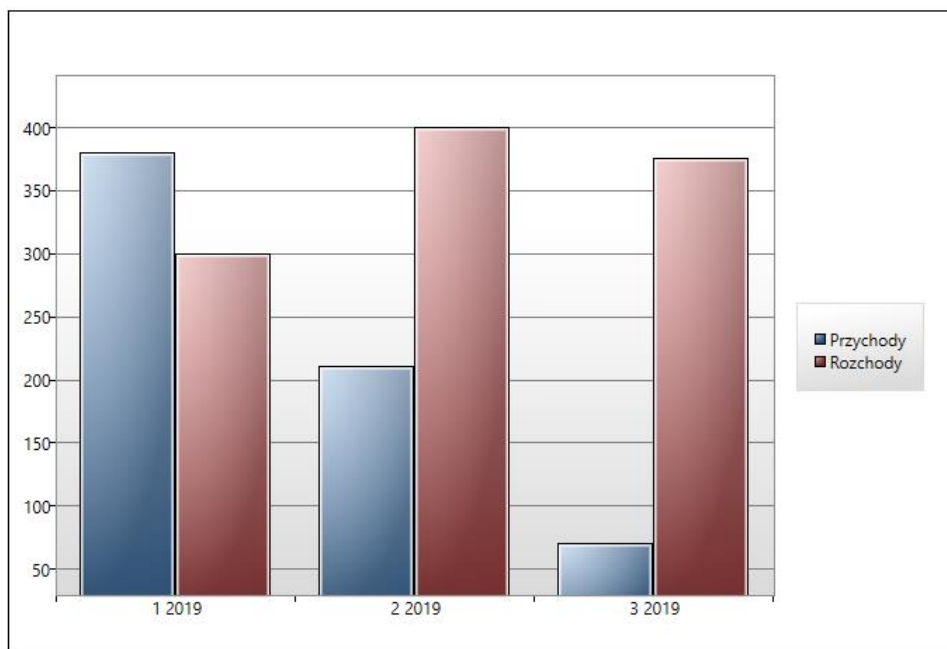
5.8. Statystyki działalności

W prezentowanej aplikacji generowanie statystyk odbywa się poprzez pokazanie klientowi odpowiedniego wykresu. Użytkownik otwierając okno do ich rysowania ma możliwość wyboru z jakiego okresu mają pochodzić dane oraz jakiego rodzaju informacji potrzebuje. Aktualnie do wyboru są dwa typy statystyk.

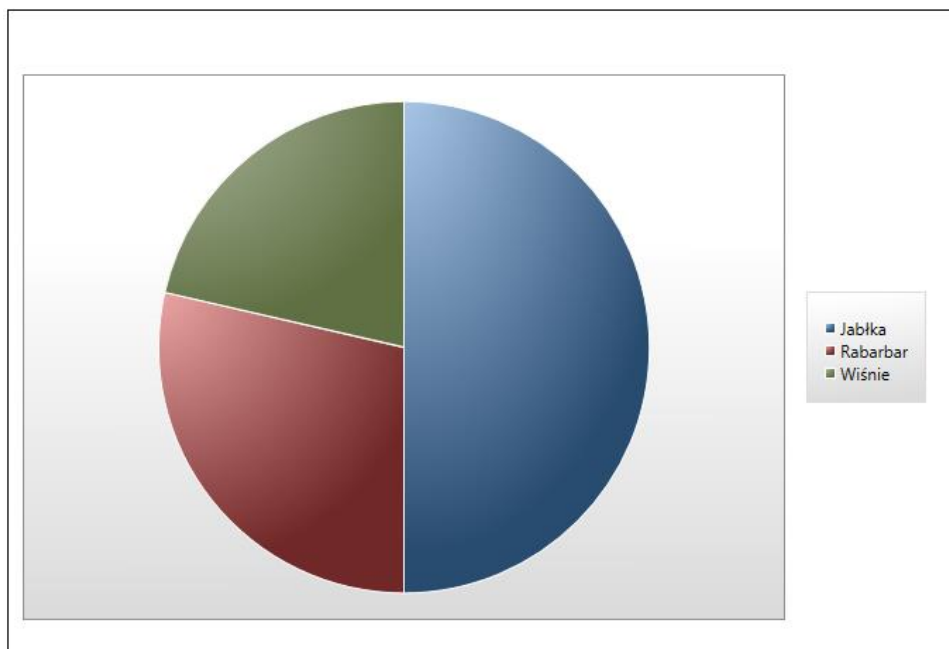
Pierwsza dotyczy porównania przychodów i rozchodów w każdym miesiącu danego przedziału czasowego. Dzięki temu, można przykładowo zauważyć, czy firma odnosi sukces na rynku, czy jej wpływy nieustannie maleją.

Drugi typ wykresu przedstawia częstotliwość wystąpień danego produktu we wszystkich zamówieniach. Informacje dostarczone przez ten wykres mogą zostać wykorzystane przykładowo przy układaniu towaru w magazynie, tak aby dostęp do produktów, na który jest wysoki popyt, był najłatwiejszy.

Rysunek 5.8. przedstawia przykładowy wykres pierwszego typu a 5.9. drugiego.



Rys. 5.8: Wykres przychodów i rozchodów w firmie



Rys. 5.9: Wykres częstości wystąpień produktów na dokumentach magazynowych

5.9. Inwentaryzacja

Okno przedstawione na rysunku 5.10. pozwala na przeprowadzenie przez użytkownika inwentaryzacji poprzez podanie różnic w stanach towarów między tymi znanymi w aplikacji a tymi z rzeczywistości. Jeżeli podana data remanentu nie będzie prowadziła do wadliwych stanów magazynów, program automatycznie zmieni ilość posiadanych towarów.

Okno aplikacji o tytule "Inwentaryzacja" zawiera formularz z następującymi elementami:

- Wpiszanie daty inwentaryzacji: 2019-12-20
- Tabela z danymi o zmianach stanów magazynowych:

Nazwa	j.m.	Przed zmianą	Po zmianie	Wartość
Produkt Pierwszy	szt.	10	5	1
Produkt Drugi	g	20	10	2

Na dole okna znajdują się przyciski "Zapisz" i "Wróć".

Rys. 5.10: Przykład przeprowadzonej inwentaryzacji

5.10. Produkcja

Program pozwala na przeprowadzenie procesu produkcyjnego. Odbywa się to za pośrednictwem okna pokazanego na rysunku 5.11. Użytkownik ma możliwość zdefiniowania nowej receptury dla produktu złożonego oraz taki towar na jej podstawie stworzyć. Operacja odwrotna również jest dozwolona.

W oknie znajdują się również dwie tabele pokazujące odpowiednio: przeprowadzone w przeszłości procesy kompletacji i dekompletacji oraz skład dla wybranej receptury.

Produkcja

Produkt złożony

Produkt Złożony

Ilość na stanie

0

Nowa receptura

Wytwarzaj

Kompletacje i dekompletacje

Data	Nazwa	Czynność	Ilość
2020-01-01	Produkt Złożony	Skompletowano	1
2019-12-19	Produkt Złożony	Rozłożono	1

Składniki

Nazwa	Ilość
Produkt Pierwszy	3

Wydź

Rys. 5.11: Ekran produkcji

5.11. Przegląd towarów na dokumentach

Ostatnie okno z rysunku 5.12. pozwala na filtrowanie dokumentów w zależności od tego, czy zawierają wybrany produkt oraz rodzaju samego dokumentu. Użytkownik ma możliwość sprawdzenia jak dany towar jest używany jest we wszystkich magazynach,

6. Testy aplikacji

Listing 6.1: Test jednostkowy

```
[TestClass]
public class UnitTest
{
    ContractorEditor contractorEditor;

    [TestInitialize]
    public void TestInitialize()
    {
        contractorEditor = new ContractorEditor(new SqlHandler(), false);
    }

    [TestMethod]
    public void ContractorGoodTaxCodeTest()
    {
        String goodEntry = "7625019373";

        try
        {
            contractorEditor.SetTaxCode(goodEntry);
            Assert.AreEqual(contractorEditor.TaxCode, goodEntry);
        }
        catch (Exception)
        {
            Assert.Fail("Good entry failed");
        }
    }

    [TestMethod]
    public void ContractorBadTaxCodeTest()
    {
        String badEntry = "5237605645";

        try
        {
            contractorEditor.SetTaxCode(badEntry);
            Assert.Fail("Exception wasn't thrown for bad entry");
        }
        catch (ArgumentException e)
        {
            StringAssert.Equals(e.Message, "Niepoprawny NIP");
        }
        catch (Exception)
        {
            Assert.Fail("Unknown exception was thrown");
        }
    }
}
```

Listing 6.1. przedstawia przykładowy test jednostkowy dla klasy, która zajmuje się tworzeniem i modyfikowaniem kontrahentów i w szczególnym przypadku danymi własnej firmy. W powyższym teście sprawdzana jest poprawność działania funkcji sprawdzającej, czy poprawnie kontruje wpisywane numery NIP.

7. Podsumowanie

W ramach projektu inżynierskiego zbudowano aplikację pozwalającą na zarządzanie wieloma magazynami o różnej wielkości. System ten pozwala na tworzenie produktów prostych i złożonych, dodawanie kontrahentów, generowanie dokumentów oraz statystyk co świadczy o tym, że wymagania funkcjonalne określone w założeniach zostały spełnione. Stworzone oprogramowanie ma duże szanse stać się alternatywą dla firm potrzebujących tańszego narzędzia niż te oferowane rynku, posiadającego szereg przydatnych funkcji.

Największym atutem tej aplikacji jest to, że wszystkie dane przechowuje na serwerze bazy danych. Takie rozwiązanie pozwala na zwiększenie uniwersalności programu dając wybór klientowi, czy samodzielnie założyć serwer, aby mieć pełną kontrolę nad dostępem do przetrzymywanych informacji, czy zdecydować się na skorzystanie z usług zewnętrznego hosta. Kolejnym plusem takiego rozwiązania jest niemal natychmiastowy przepływ informacji między magazynami. Dzięki ciągłej synchronizacji, każda osoba z dostępem będzie miała możliwość wglądu w aktualne stany magazynowe.

Mimo że aplikacja już na ten moment oferuje dużo możliwości, to została zaprojektowana w taki sposób, aby umożliwić jej dalszą rozbudowę. Należy jednak zwrócić uwagę na to, że dla tego typu oprogramowania, które opiera swoje działanie na obowiązujących przepisach, jego aktualizowanie i dostosowywanie musi istnieć niezależnie od procesu wprowadzania nowych funkcji.

Literatura

- [1] Oficjalna strona programu LoMag. <https://www.lomag.pl/> [dostęp dnia 18.11.2019]
- [2] Oficjalna strona programu Smart System. <https://www.system-smart.pl/> [dostęp dnia 18.11.2019]
- [3] Internetowy System Aktów Prawnych. <http://prawo.sejm.gov.pl/isap.nsf/> [dostęp dnia 20.11.2019]
- [4] Oficjalna dokumentacja Microsoftu. <https://docs.microsoft.com/> [dostęp dnia 18.11.2019]
- [5] Liberty Jesse, C#. Programowanie, Helion, 2005
- [6] Oficjalna dokumentacja MySQL <https://dev.mysql.com/doc/> [dostęp dnia 23.11.2019]
- [7] Celko Joe, SQL for Smarties: Advanced SQL Programming Third Edition, Morgan Kaufmann, 2005

Spis rysunków

Rys. 4.1: Diagram ERD tworzonej bazy danych	9
Rys. 5.1: Ekran logowania	14
Rys. 5.2: Ekran zarządzania magazynami.....	15
Rys. 5.3: Główny menadżer magazynu.....	15
Rys. 5.4: Ekran zarządzania produktami.....	16
Rys. 5.5: Edytor kontrahentów	17
Rys. 5.6: Edytor dokumentów	17
Rys. 5.7: Przykład generowanego dokumentu magazynowego.....	18
Rys. 5.8: Wykres przychodów i rozchodów w firmie.....	19
Rys. 5.9: Wykres częstości wystąpień produktów na dokumentach magazynowych.....	20
Rys. 5.10: Przykład przeprowadzonej inwentaryzacji	20
Rys. 5.11: Ekran produkcji.....	21
Rys. 5.12: Przegląd towarów na dokumentach	22

Spis listingów

Listing 4.1: Metody do komunikacji z bazą.....	10
Listing 4.2: Funkcja zwracająca komendę tworzenia tablicy	12
Listing 4.3: Tworzenie zapytania o produkty podlegające pod dany dokument	12
Listing 4.4: Tworzenie wyzwalacza dla wprowadzanych szczegółów zamówienia.....	13
Listing 6.1: Test jednostkowy	23