WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: Programowanie

Uladzislau Bainiarovich
Nr albumu studenta w68339

Aplikacja konsolowa Zarządzanie magazynem

Prowadzący: mgr inż. Ewa Żesławska

Praca projektowa programowanie obiekotwe C#

Rzeszów 2025

Spis treści

Wstęp

Jest to projekt systemu zarządzania magazynem obejmujący: język programowania C sharp, bazę danych SQL, zarządzanie danymi (CRUD), ładowanie danych za pomocą interfejsów, obsługę wyjątków, walidację danych.

Rozdział 1

Opis założeń projektu

1.1 Cele projektu

Celem projektu jest: stworzyć program "Zarządzanie magazynem". Wynikiem pracy będzie: aplikacja konsolowa

1.2 Wymagania funkcjonalne i niefunkcjonalne

Definicja:
Wymagania funkcjonalne:

- Określają konkretne zadania, które użytkownik lub system może wykonać.
- Są bezpośrednio związane z działaniem aplikacji i jej funkcjonalnością.
- Zwykle wynikają z potrzeb użytkownika.

  Przykłady wymagań funkcjonalnych:
- Logowanie i rejestracja użytkownika
- Obsługa magazynu
- Generowanie statystyk
- Powiadomienia
- Edycja i usuwanie danych

Wymagania niefunkcjonalne:

- Określają ograniczenia lub cechy związane z działaniem systemu.
- Dotyczą takich aspektów, jak czas odpowiedzi, skalowalność, dostępność, czy zgodność z regulacjami prawnymi.

  Przykłady wymagań niefunkcjonalnych:
- Wydajność
- Bezpieczeństwo
- Dostępność
- Użyteczność

- Skalowalność
- Zgodność
- Niezawodność

Rozdział 2

Opis struktury projektu

2.1 Struktura oraz opis techniczny

Struktura projektu składa się z interfejsa i bazy danych.

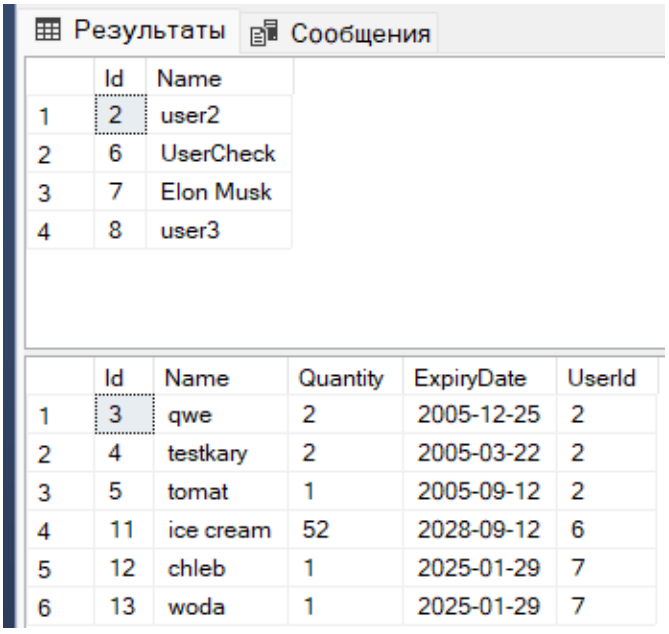
- Baza danych umożliwia przechowywanie oraz dostęp do informacji.
- Interfejs użytkownika umożliwia łatwe zarządzanie danymi.

2.1.1 Interfejs użytkownika

Interfejs użytkownika w tym projekcie jest tekstowy (konsolowy) i zapewnia podstawową funkcjonalność w przejrzysty sposób. Pozwala użytkownikowi na interakcję z systemem zarządzania magazynem poprzez wprowadzanie odpowiednich komend i danych.

2.1.2 Baza Danych

Baza danych przechowuje informacje o użytkownikach i produktach. Jest również natychmiast aktualizowana po wprowadzeniu zmian za pomocą programu konsoli.


Результаты Сообщения

	Id	Name
1	2	user2
2	6	UserCheck
3	7	Elon Musk
4	8	user3

	Id	Name	Quantity	ExpiryDate	UserId
1	3	qwe	2	2005-12-25	2
2	4	testkary	2	2005-03-22	2
3	5	tomat	1	2005-09-12	2
4	11	ice cream	52	2028-09-12	6
5	12	chleb	1	2025-01-29	7
6	13	woda	1	2025-01-29	7

2.2 Wykorzystany język, narzędzia oraz minimalne wymagania sprzętowe

2.2.1 Języki

- System został stworzony w języku **C#**
- Baza danych została stworzona przy użyciu języka **SQL**

2.2.2 Narzędzia

- Microsoft Visual Studio. Community Edition 2022
- Microsoft Server Management Studio 20

2.2.3 Minimalne wymagania sprzętowe

- RAM: 2Gb
- Procesor: 2GHz
- OS: Windows 7 (i wyższy)
- Dysk: 100mb wolnego miejsca

2.3 Hierarchia klas i opis metod

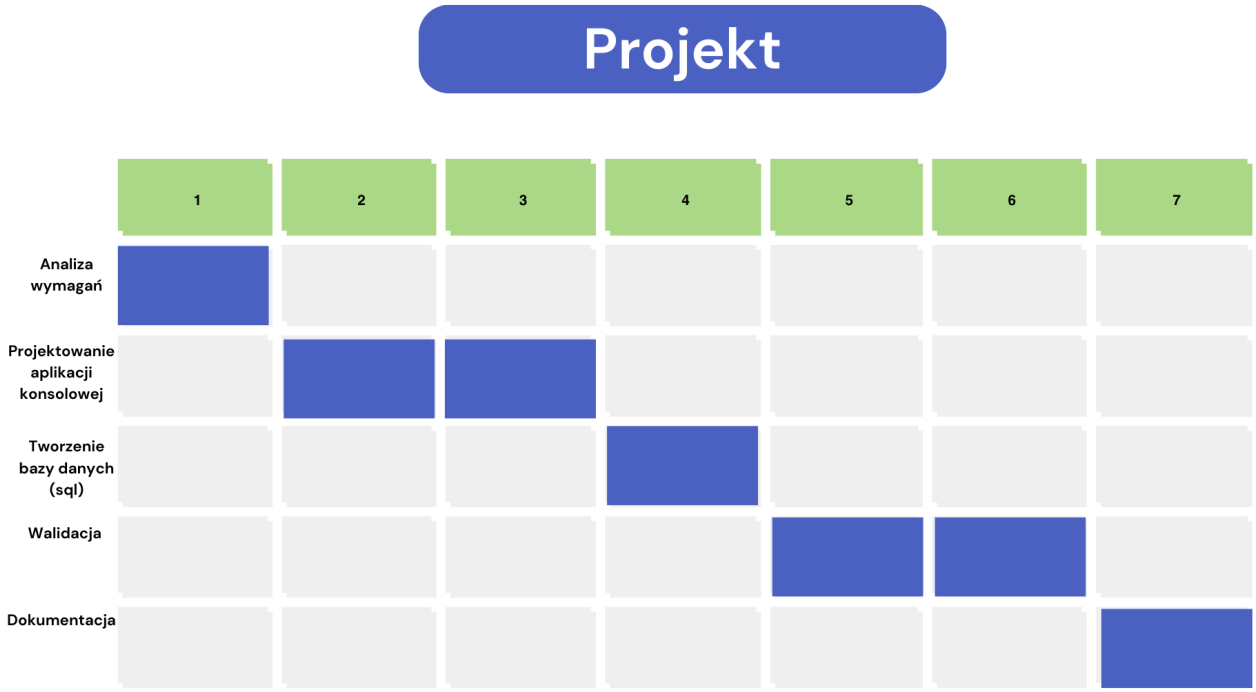
- **Program** - klasa główna, która inicjalizuje i uruchamia aplikację.
  - `Main()` - metoda główna programu, obsługuje logikę interfejsu użytkownika.
- **DatabaseManager** - klasa odpowiedzialna za komunikację z bazą danych.
  - `ExecuteNonQuery(string query, Dictionary<string, object> parameters)` - wykonuje zapytania SQL, które nie zwracają wyników (np. `INSERT`, `UPDATE`).
  - `ExecuteQuery(string query, Dictionary<string, object> parameters = null)` - wykonuje zapytania SQL, które zwracają wyniki (np. `SELECT`).
- **Validator** - klasa odpowiedzialna za walidację danych wejściowych.
  - `ValidateUserInput(string input, string fieldName)` - sprawdza, czy dane wejściowe nie są puste.
  - `ValidateDateInput(string input)` - sprawdza, czy podana data jest w poprawnym formacie.
- **FileManager** - klasa zarządzająca zapisem i odczytem danych z plików.
  - `SaveData(string filePath, List<User> users)` - zapisuje dane użytkowników i produktów do pliku tekstowego.
  - `LoadData(string filePath)` - odczytuje dane z pliku i odtwarza strukturę użytkowników oraz ich produktów.
- **StatisticsManager** - klasa generująca statystyki na podstawie danych z magazynu.

  - `GenerateUserStatistics(Warehouse warehouse)` - generuje statystyki użytkowników.
  - `GenerateOverdueProducts(Warehouse warehouse)` - generuje listę przeterminowanych produktów.
- **PenaltyCalculator** - klasa obliczająca kary za przeterminowane produkty.
  - `CalculatePenalty(Product product)` - oblicza wysokość kary na podstawie liczby dni po terminie.
- **Warehouse** - klasa reprezentująca magazyn i zarządzająca użytkownikami oraz produktami.
  - `LoadUsersFromDatabase()` - ładuje listę użytkowników z bazy danych.
  - `AddUser(string name)` - dodaje nowego użytkownika do bazy danych.
  - `AddProductToUser(int userId, string productName, int quantity, DateTime expiryDate)` - dodaje produkt do konkretnego użytkownika.
  - `GetAllUsers()` - zwraca listę wszystkich użytkowników.
  - `GetProductsOfUser(int userId)` - zwraca listę produktów przypisanych do użytkownika.
- **User** - klasa reprezentująca użytkownika w systemie.
  - `AddProduct(Product product)` - dodaje produkt do listy produktów użytkownika.
  - `ToString()` - zwraca reprezentację tekstową użytkownika (ID, imię, liczba produktów).
- **Product** - klasa reprezentująca produkt w systemie.
  - `ToString()` - zwraca reprezentację tekstową produktu (ID, nazwa, ilość, data ważności).
- **IValidator** - interfejs definiujący metody walidacji.
  - `ValidateUserInput(string input, string fieldName)` - metoda do walidacji tekstu.
  - `ValidateDateInput(string input)` - metoda do walidacji daty.
- **IFileManager** - interfejs definiujący operacje na plikach.
  - `SaveData(string filePath, List<User> users)` - metoda do zapisywania danych.
  - `LoadData(string filePath)` - metoda do ładowania danych.
- **IUserManager** - interfejs definiujący operacje na użytkownikach.
  - `AddUser(User user)` - metoda do dodawania użytkownika.
  - `AddProductToUser(int userId, Product product)` - metoda do dodawania produktu do użytkownika.
  - `GetAllUsers()` - metoda do pobierania listy użytkowników.
  - `GetUserById(int userId)` - metoda do pobierania użytkownika na podstawie ID.

Rozdział 3

Harmonogram realizacji projektu

3.1 Harmonogram



3.2 Repozytorium

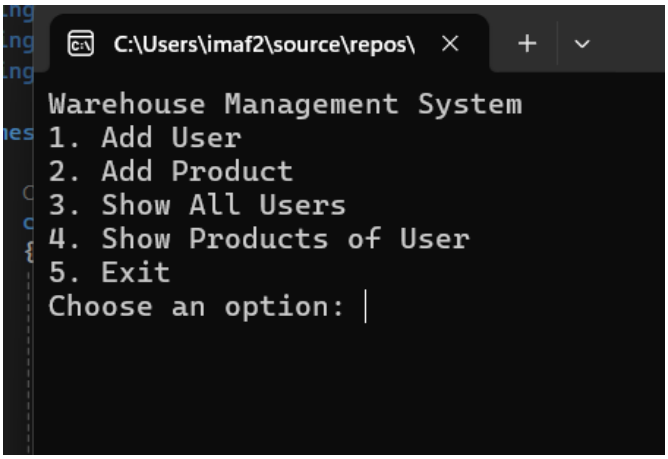
Wszystkie pliki źródłowe zostały zamieszczone w repozytorium, dostępnym pod linkiem: `https://github.com/Vlad16d/Warehouse-Management-System`

Pliki źródłowe obejmują:
Pliki systemowe
Folder SQL z plikami dla bazy danych
Pliki dokumentacji
Folder Latex z plikami dla LaTeX

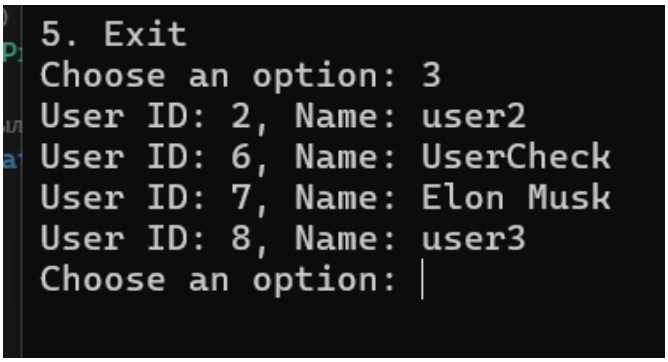
Rozdział 4

Prezentacja warstwy użytkowej projektu

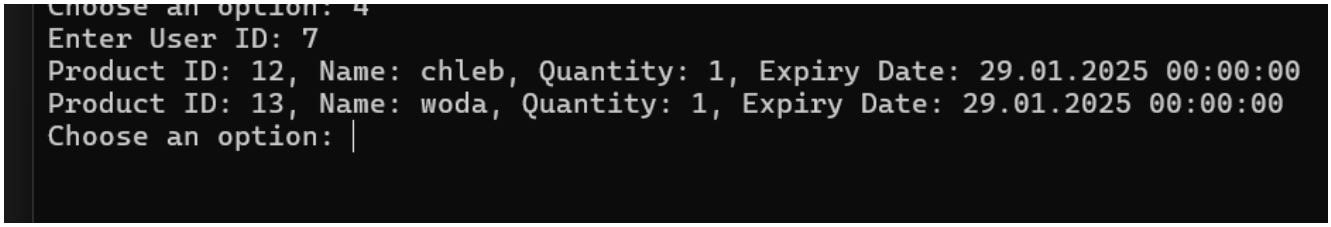
Po włączeniu aplikacji jest główne menu:



Jest możliwość sprawdzić wszystkich użytkowników



Jest możliwość sprawdzenia produktów użytkownika



Jest możliwość dodawania użytkownika i produktów do niego

```
Warehouse Management System
1. Add User
2. Add Product
3. Show All Users
4. Show Products of User
5. Exit
Choose an option: 1
Enter User Name: Latex
User added successfully.
Choose an option: 2
Enter User ID: 9
Enter Product Name: latex_test
Enter Quantity: 643
Enter Expiry Date (yyyy-MM-dd): 2026-01-28
```

Sprawdzamy nowego użytkownika w Bazie danych:

	Id	Name
1	2	user2
2	6	UserCheck
3	7	Elon Musk
4	8	user3
5	9	Latex

	Id	Name	Quantity	ExpiryDate	UserId
1	3	qwe	2	2005-12-25	2
2	4	testkary	2	2005-03-22	2
3	5	tomat	1	2005-09-12	2
4	11	ice cream	52	2028-09-12	6
5	12	chleb	1	2025-01-29	7
6	13	woda	1	2025-01-29	7
7	14	latex_test	643	2026-01-28	9

Rozdział 5

Podsumowanie

W ramach projektu zrealizowano system zarządzania magazynem, który umożliwia efektywne zarządzanie użytkownikami oraz ich produktami. Kluczowe funkcjonalności obejmują dodawanie użytkowników i produktów, generowanie statystyk użytkowników oraz monitorowanie przeterminowanych produktów. Aplikacja integruje się z bazą danych SQL, co pozwala na trwałe przechowywanie danych oraz ich łatwe przetwarzanie. Dodatkowo, system oferuje możliwość eksportu i importu danych z plików tekstowych, co ułatwia ich wymianę oraz kopie zapasowe.

W trakcie realizacji projektu stworzono rozbudowaną strukturę klas, w której zaimplementowano wzorce projektowe i zasady programowania obiektowego. System jest skalowalny i może być łatwo rozbudowywany o dodatkowe funkcje, takie jak obliczanie kar za przeterminowane produkty czy automatyczne powiadomienia o kończącej się ważności produktów. Zadbano również o podstawowe walidacje danych wejściowych, co minimalizuje błędy użytkownika podczas korzystania z systemu.

5.1 Planowane dalsze prace

W przyszłości projekt można rozwinąć o następujące funkcje:

1. **Interfejs graficzny** – zastąpienie obecnego interfejsu konsolowego nowoczesnym graficznym interfejsem użytkownika (GUI), co poprawi doświadczenie użytkownika.

2. **Powiadomienia e-mail** – automatyczne wysyłanie wiadomości e-mail do użytkowników z przypomnieniami o przeterminowanych produktach lub kończącym się terminie ważności.

3. **Rozbudowa statystyk** – dodanie bardziej szczegółowych raportów, takich jak analiza popularności produktów czy podsumowania miesięczne.

4. **Obsługa wielu magazynów** – możliwość zarządzania wieloma magazynami w ramach jednego systemu.

5. **Integracja z systemami ERP** – połączenie systemu z większymi platformami zarządzania zasobami, co pozwoli na lepsze planowanie i zarządzanie w przedsiębiorstwach.

Zrealizowany projekt stanowi solidną podstawę do dalszego rozwoju i wdrożenia w rzeczywistym środowisku biznesowym. Dzięki modularnej architekturze oraz wysokiej czytelności kodu możliwa jest szybka adaptacja systemu do nowych wymagań oraz wdrażanie dodatkowych funkcji.

Bibliografia

[1] https://learn.microsoft.com/

[2] https://is.umk.pl/ grochu/wiki/doku.php?id=zajecia:npr:wyklad:documentation

[3] https://forum.pasja-informatyki.pl/520906/dokumentacja-c