Koncepcja wykonania
Systemu Zarządzania Zapasami
Autor: Mateusz Mazur

1. Przypadki użycia
A) Dodawanie produktu do bazy danych:
Aktorzy: Sprzedawca

Opis:
Sprzedawca wprowadza nowy produkt do bazy danych, podając takie informacje jak nazwa, kod, kategoria, cena i ilość.

Przypadek podstawowy:
1. Sprzedawca loguje się do systemu.
2. Sprzedawca wybiera opcję "Dodaj produkt".
3. Sprzedawca wprowadza wymagane informacje o produkcie.
4. System zapisuje nowe dane produktu w bazie danych.
5. System potwierdza dodanie produktu.

Przypadki alternatywne:
3a. Sprzedawca wprowadza niekompletne dane -> System wyświetla komunikat o błędzie i prosi o uzupełnienie brakujących informacji.

B) Określanie minimalnego poziomu zapasów:
Aktorzy: Sprzedawca

Opis:
Sprzedawca ustala minimalny poziom zapasów dla wybranego produktu.

Przypadek podstawowy:
1. Sprzedawca loguje się do systemu.
2. Sprzedawca wybiera produkt z listy.
3. Sprzedawca wprowadza minimalny poziom zapasów dla produktu.
4. System zapisuje nowy poziom minimalny.
5. System potwierdza ustawienie minimalnego poziomu zapasów.

Przypadki alternatywne:
3a. Sprzedawca wprowadza nieprawidłową wartość -> System wyświetla komunikat o błędzie i prosi o podanie poprawnej wartości.

C) Monitorowanie poziomu zapasów i generowanie raportów:
Aktorzy: Sprzedawca

Opis:
System monitoruje poziom zapasów i generuje regularne raporty.

Przypadek podstawowy:
1. System automatycznie monitoruje poziom zapasów w czasie rzeczywistym.
2. System generuje codzienne raporty dotyczące stanu magazynu.
3. Sprzedawca loguje się do systemu.
4. Sprzedawca przegląda dostępne raporty.

Przypadki alternatywne:
2a. System napotka błąd podczas generowania raportu -> System wyświetla komunikat o błędzie i powiadamia administratora.

D) Śledzenie statusu zamówień:
Aktorzy: Sprzedawca

Opis:
Sprzedawca śledzi status zamówień złożonych u dostawców.

Przypadek podstawowy:
1. Sprzedawca loguje się do systemu.
2. Sprzedawca wybiera zakładkę "Zamówienia".
3. Sprzedawca przegląda listę zamówień i ich statusy.
4. System wyświetla szczegóły wybranego zamówienia.

Przypadki alternatywne:
3a. Brak zamówień w systemie -> System wyświetla komunikat informacyjny.

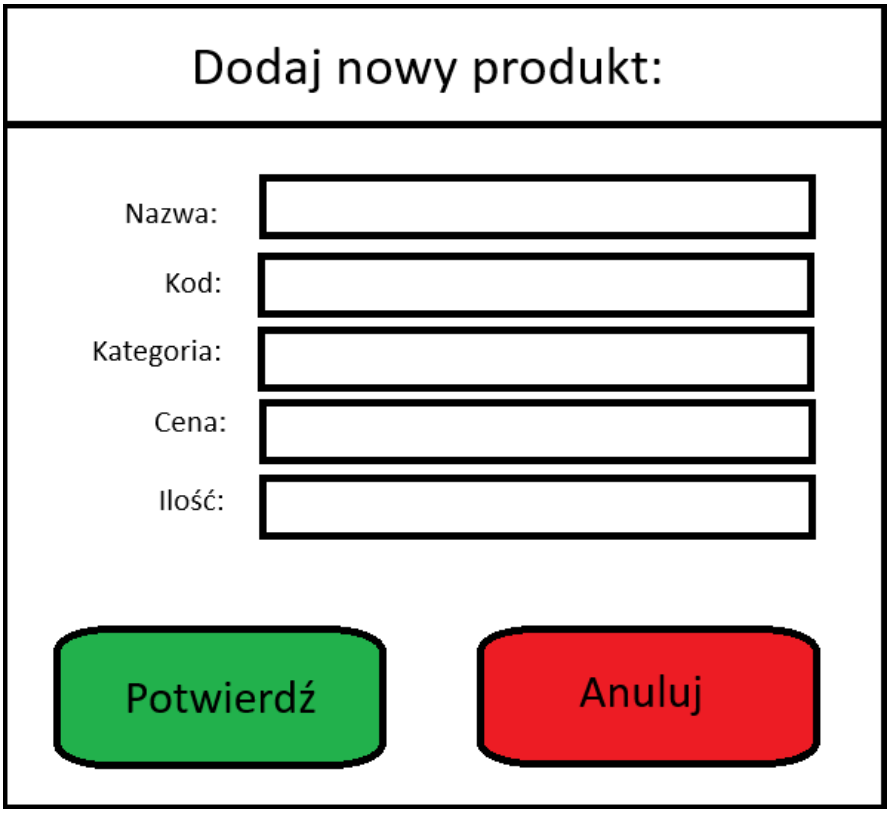
E) Przewidywanie poziomu popytu:
Aktorzy: Sprzedawca

Opis:
System analizuje dane historyczne i przewiduje poziom popytu na produkty.

Przypadek podstawowy:
1. System regularnie analizuje dane sprzedaży.
2. System generuje prognozy popytu na podstawie analizy danych.
3. Sprzedawca loguje się do systemu.
4. Sprzedawca przegląda raporty z prognozami popytu.

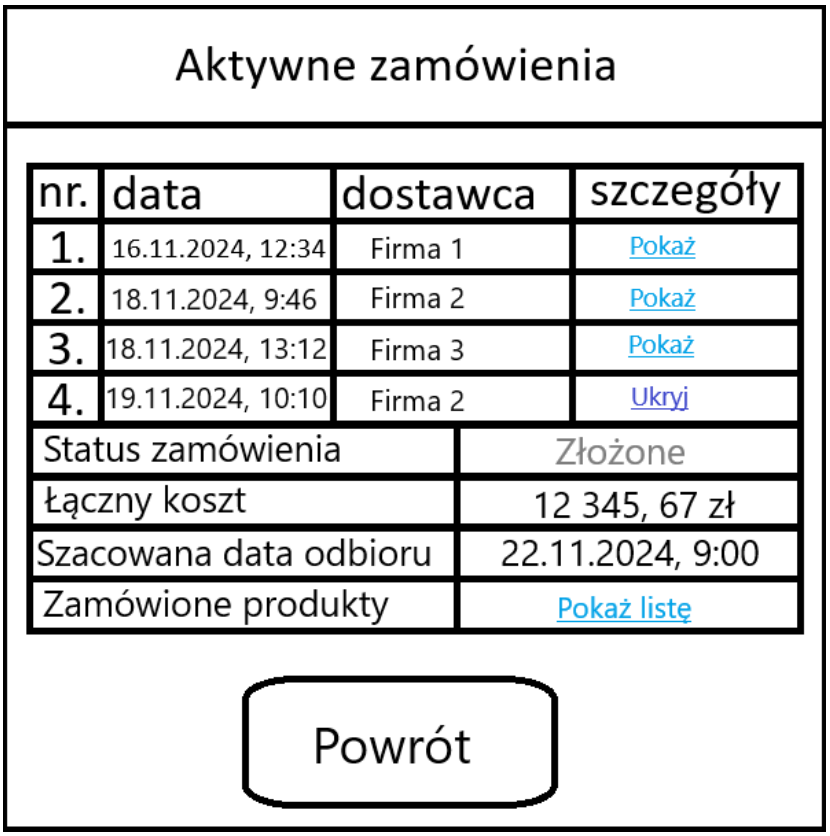
Przypadki alternatywne:
2a. Dane historyczne są niepełne -> System wyświetla komunikat
o konieczności uzupełnienia danych.

2. Projekty dialogów



Strona z dodawaniem nowego produktu (A)

Strona z modyfikowaniem minimalnego poziomu zapasów (domyślnie wynosi on 0) (B)



Strona śledząca status zamówień (D)

3. Zaprojektowanie architektury systemu

A) Przewidywane rozwiązania sprzętowe:
Serwery przechowujące logikę aplikacji i zarządzające operacjami backendowymi.
Serwery odpowiedzialne za przechowywanie danych dotyczących zapasów, zamówień, raportów itp.
Serwery sieciowe obsługujące ruch sieciowy i komunikację z użytkownikami.
Urządzenia użytkowników końcowych: Komputery używane przez sprzedawców do interakcji z systemem.

B) Oprogramowanie systemowe, bazy danych, narzędzia programistyczne:
System operacyjny Linux/Windows Server na serwerach aplikacji i baz danych.
Relacyjne bazy danych, takie jak MySQL, do przechowywania i zarządzania danymi.

C) Narzędzia programistyczne:
ASP.NET Core (C#) do budowy backendu.
React.js do tworzenia interfejsu użytkownika.
Systemy kontroli wersji: GitHub, GitLab do zarządzania wersjami kodu źródłowego.

D) Oprogramowanie do automatycznego testowania:
Testy jednostkowe i End-to-endy: NUnit (C#).

E) Struktura logiczna oprogramowania (podział kodu na główne komponenty):

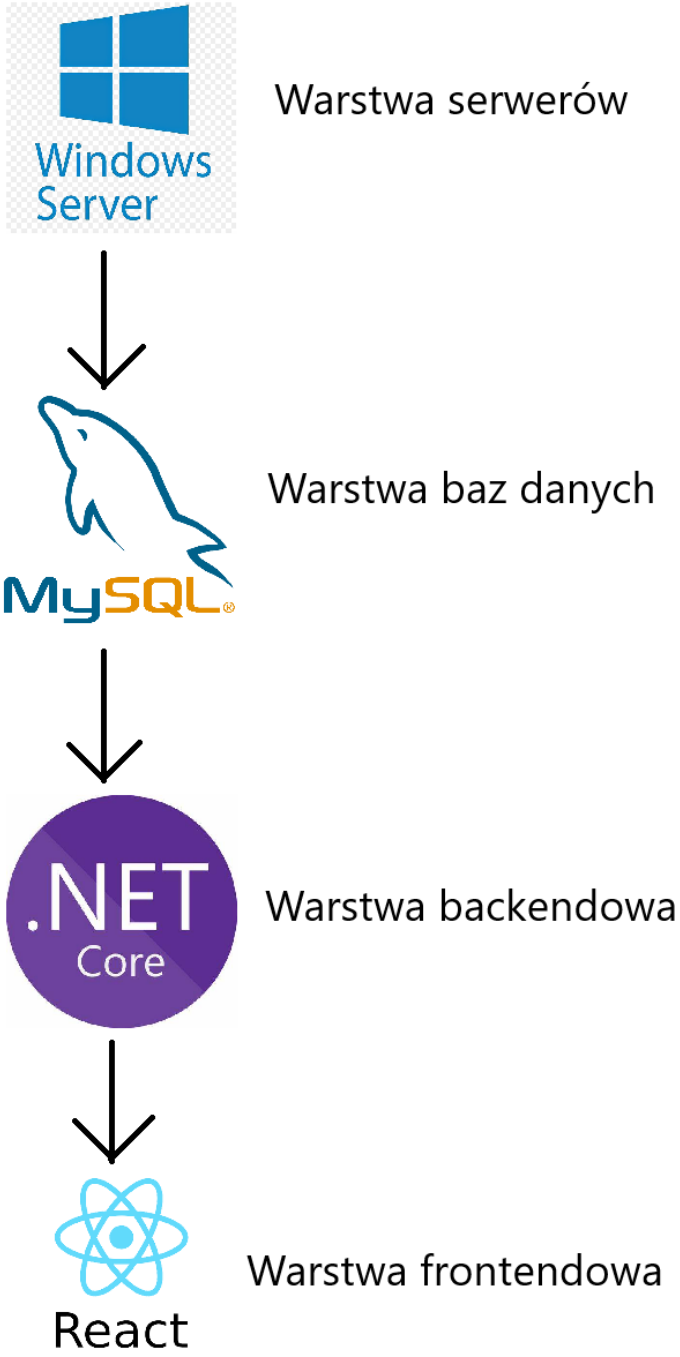
I. Warstwa prezentacji (Frontend):
Komponenty UI - Formularze, tabele, widoki.
Routing - Zarządzanie nawigacją i stronami.
Serwisy - Komunikacja z backendem poprzez API.

II. Warstwa logiki biznesowej (Backend):
Kontrolery: Obsługa żądań użytkowników i przekierowanie do odpowiednich usług.
Serwisy: Logika biznesowa, przetwarzanie danych.

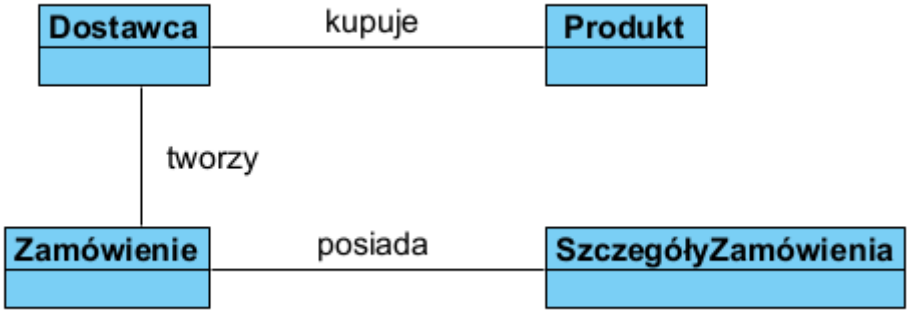
Repozytoria: Interakcja z bazą danych, zarządzanie transakcjami.

III. Warstwa danych:
Modele danych: Definicja struktur danych przechowywanych w bazie danych.
Baza danych: Przechowywanie i zarządzanie danymi.

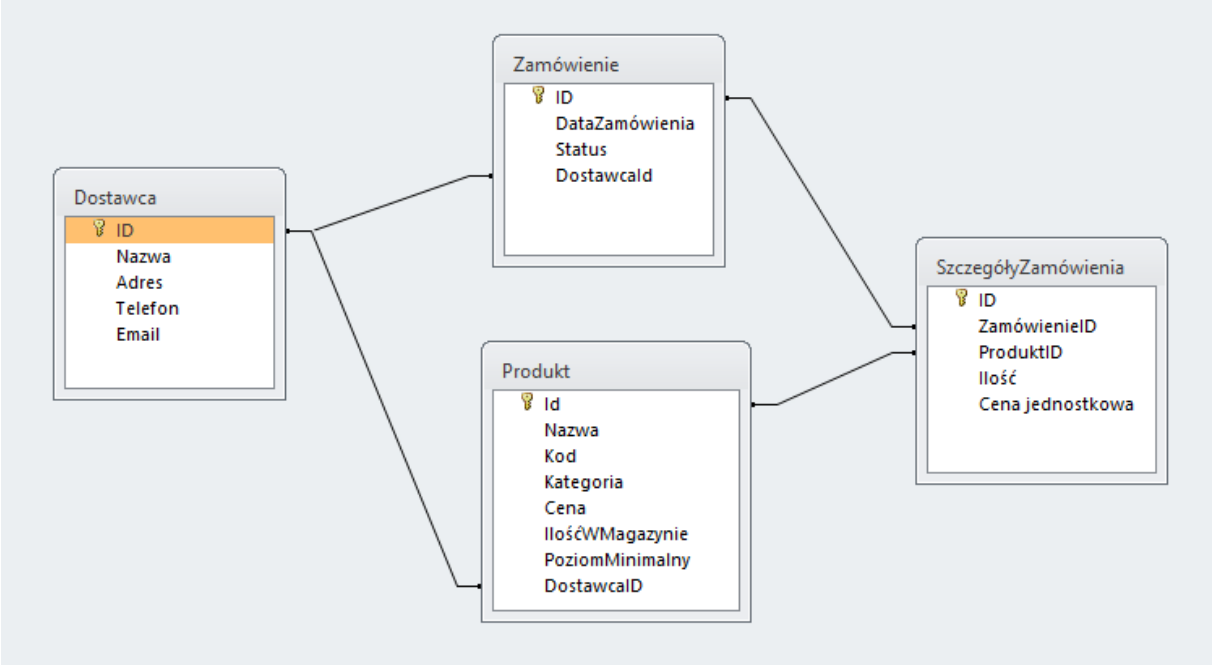
4. Diagram architektury systemu:



5a. Model konceptualny rzeczywistości



5b. Schemat bazy danych



6. Przedstawienie głównych zasad kodowania na podstawie dostępnych materiałów.

- Pisz kod tak, aby był czytelny dla innych programistów. Używaj jasnych i opisowych nazw zmiennych, funkcji i klas.
- DRY - Unikaj duplikowania kodu. Wykorzystuj funkcje, klasy i moduły do organizowania powtarzających się fragmentów kodu.
- Kiedy zauważysz powtarzający się kod, zastanów się nad jego refaktoryzacją i przeniesieniem do osobnej funkcji lub modułu.
- Każda funkcja lub klasa powinna mieć jedną, dobrze określoną odpowiedzialność (Single Responsibility Principle).
- Unikaj tworzenia "monstrów", które robią zbyt wiele rzeczy naraz. Podziel kod na mniejsze, bardziej zarządzalne części.
- Organizuj kod w moduły, które mają jasno określone interfejsy.
- Oddziel logikę biznesową od warstwy prezentacji i interakcji z bazą danych, aby ułatwić testowanie i modyfikowanie kodu.
- Pisz testy jednostkowe dla najważniejszych funkcji i modułów, aby upewnić się, że działają poprawnie.
- Dokumentuj kod, aby inni programiści mogli łatwo zrozumieć jego działanie.

7. Zidentyfikowanie zagrożeń i opracowanie zasad zarządzania ryzykiem
A) Identyfikacja zagrożeń:
I. Zagrożenia związane z bezpieczeństwem:
Nieautoryzowany dostęp - Zagrożenie uzyskania dostępu do systemu przez nieautoryzowane osoby.
Ataki typu SQL Injection - Zagrożenie wynikające z podatności na wstrzykiwanie kodu SQL.
Ataki typu XSS (Cross-Site Scripting) - Zagrożenie polegające na wstrzyknięciu szkodliwego kodu do stron internetowych.

II. Zagrożenia związane z wydajnością:
Wysokie obciążenie serwerów - Zagrożenie spowodowane nadmiernym obciążeniem serwerów, co może prowadzić do spadku wydajności.
Problemy z skalowalnością - Zagrożenie związane z niezdolnością systemu do obsługi zwiększonej liczby użytkowników lub danych.

III. Zagrożenia związane z danymi:
Utrata danych: Zagrożenie utraty danych w wyniku awarii sprzętu, ataku hakerskiego lub błędu użytkownika.
Naruszenie prywatności: Zagrożenie nieautoryzowanego dostępu do danych osobowych użytkowników.

IV. Zagrożenia związane z integracją:
Problemy z integracją z innymi systemami: Zagrożenie wynikające z trudności w integracji z systemami ERP lub CRM.
Niekompatybilność technologiczna: Zagrożenie związane z różnymi technologiami używanymi w systemach zewnętrznych.

B) Zasady zarządzania ryzykiem:
I. Ocena ryzyka:
Regularnie przeprowadzaj ocenę ryzyka, identyfikując potencjalne zagrożenia i oceniając ich wpływ na system oraz prawdopodobieństwo wystąpienia.

II. Zarządzanie dostępem:
Wdrażaj silne mechanizmy uwierzytelniania i autoryzacji, takie jak dwuskładnikowe uwierzytelnianie.
Regularnie przeglądaj i aktualizuj listy uprawnień użytkowników.

III. Bezpieczeństwo aplikacji:
Stosuj praktyki bezpiecznego kodowania, aby zapobiegać atakom SQL Injection i XSS.
Regularnie przeprowadzaj testy penetracyjne i audyty bezpieczeństwa.

IV. Zarządzanie wydajnością:
Monitoruj wydajność systemu i wdrażaj mechanizmy równoważenia obciążenia (load balancing).
Zadbaj o skalowalność systemu, umożliwiając dynamiczne dostosowywanie zasobów do obciążenia.

V. Ochrona danych:
Regularnie twórz kopie zapasowe danych i przechowuj je w bezpiecznym miejscu.
Szyfruj dane przechowywane oraz przesyłane w systemie.

VI. Zgodność z przepisami:
Upewnij się, że system jest zgodny z obowiązującymi przepisami prawnymi dotyczącymi ochrony danych osobowych (np. RODO).
Prowadź regularne szkolenia dla pracowników dotyczące bezpieczeństwa danych i zgodności z przepisami.

VII. Plan awaryjny:
Opracuj plan awaryjny na wypadek wystąpienia krytycznych awarii systemu.
Regularnie testuj i aktualizuj plan awaryjny, aby zapewnić jego skuteczność.

8. Zmiany względem tablicy koncepcyjnej
Użyjemy C# (NET CORE) zamiast C++ do backendu.
Użyjemy MySQL zamiast MongoDB jako bazę danych.