PROJEKT INŻYNIERSKI

„System wspomagający sprzedaż i zarządzanie zasobami firmy”

Karol ZIAJA
Nr albumu kz306786

Kierunek: Informatyka
Specjalność: Bazy Danych i Inżynieria Systemów

PROWADZĄCY PRACĘ
Dr hab. inż. Bożena Małysiak-Mrozek
KATEDRA Systemów Rozproszonych i Urządzeń Informatyki
Wydział Automatyki, Elektroniki i Informatyki

GLIWICE 2026

Tytuł pracy:

System wspomagający sprzedaż i zarządzanie zasobami firmy

Streszczenie:

W pracy opracowano webowy system wspomagający sprzedaż i zarządzanie zasobami przedsiębiorstwa. System obejmuje moduły magazynowy, sprzedażowy i administracyjny, a dostęp do funkcji kontrolowany jest poprzez role użytkowników. Implementację wykonano z użyciem ASP.NET Core 9, Blazor Server oraz bazy SQL Server. Zastosowano algorytm priorytetyzacji zamówień oparty na logice rozmytej, umożliwiający dynamiczną ocenę ważności zamówień. System zweryfikowano pod kątem wymagań funkcjonalnych, niefunkcjonalnych i wydajnościowych. Opracowane rozwiązanie usprawnia realizację procesów biznesowych i stanowi podstawę do dalszej rozbudowy.

Słowa kluczowe:

System zarządzania zasobami, sprzedaż, magazyn, logika rozmyta, ASP.NET Core, Blazor, ERP

Thesis title:

System supporting sales and management of company resources

Abstract:

This thesis presents a web-based system supporting sales processes and company resource management. It includes warehouse, sales and administration modules with role-based access control. The system was implemented using ASP.NET Core 9, Blazor Server and SQL Server database. A fuzzy-logic algorithm was applied to determine order priority dynamically. The solution was validated against functional, non-functional and performance requirements. The system improves sales and resource management workflows and provides a foundation for further development.

Keywords:

Resource management system, sales, warehouse, fuzzy logic, ASP.NET Core, ERP

Spis treści

Rozdział 1

Wstęp

Współczesne firmy coraz częściej wykorzystują systemy informatyczne do usprawnienia procesów sprzedaży, zarządzania zasobami oraz komunikacji pomiędzy poszczególnymi oddziałami firmy. Wzrost liczby zamówień i zróżnicowanie oferty produktowej powoduje, że automatyzacja i integracja danych staje się coraz bardziej znacząca. Brak takiego systemu może prowadzić do opóźnień w realizacji zamówień, niezgodności stanów magazynowych oraz bezpośredniego następstwa – utrudnionej analizy wyników sprzedaży.

Kluczowym zagadnieniem staje się więc projektowanie zintegrowanych systemów, wspomagających zarządzanie procesami magazynowymi oraz obsługę sprzedaży. Tematyka ta wpisuje się w szerszą dziedzinę systemów informatycznych klasy **ERP** (Enterprise Resource Planning). Systemy te łączą ze sobą funkcjonalności administracyjne, logistyczne oraz analityczne – w jednym środowisku podzielonym na odpowiednie moduły. Wykorzystanie technologii webowych i odpowiednie zaprojektowanie systemu pozwalają na udostępnienie tych funkcji wielu użytkownikom jednocześnie, z zachowaniem spójności danych i kontroli uprawnień.

Celem pracy inżynierskiej było zaprojektowanie i implementacja systemu wspomagającego sprzedaż oraz zarządzanie zasobami firmy, który pozwoli na bieżące monitorowanie stanów magazynowych, realizację zamówień oraz wystawianie dokumentów sprzedaży. Projekt miał na celu stworzenie intuicyjnej, webowej aplikacji korzystającej ze zintegrowanej bazy danych. Aplikacja powinna umożliwiać pracę wielu użytkownikom jednocześnie z różnymi poziomami dostępów do modułów programu.

Projektowany system będzie zawierał trzy główne moduły:

- **Moduł magazynowy** – obsługujący ewidencję i podgląd stanów towarów w poszczególnych oddziałach (magazynach) firmy. Jego zadaniem będzie umożliwienie składania zamówień do zewnętrznych kontrahentów oraz generowanie alertów przy niskim poziomie zapasów.
- **Moduł sprzedaży –** umożliwiający realizację zamówień na podstawie dostępnych zasobów, dostęp do panelu analitycznego oraz generowanie i archiwizację dokumentów sprzedaży.
- **Moduł administracyjny** – dostępny dla administratora moduł umożliwiający zarządzanie użytkownikami, rolami czy dostępnymi kontrahentami.

Dodatkowo zaimplementowany zostanie system ról użytkowników (Administrator, Kierownik Oddziału, Pracownik Magazynu, Pracownik Handlowy), który definiował będzie uprawnienia do poszczególnych funkcji systemu.

Część pisemna pracy została podzielona na 7 głównych rozdziałów:

1. Wprowadzenie – omówienie problematyki, celu oraz zakresu projektu.
2. Analiza tematu – sformułowanie problemu, opis istniejących rozwiązań oraz wykorzystywanych algorytmów.
3. Wymagania i narzędzia – wymagania funkcjonalne i niefunkcjonalne, diagramy UML, stos technologiczny, a także metodyka pracy nad projektowaniem i implementacją.
4. Specyfikacja zewnętrzna – wymagania sprzętowe i programowe, sposób instalacji oraz aktywacji programu, instrukcja korzystania z poszczególnych funkcjonalności
5. Specyfikacja wewnętrzna – przedstawienie idei, architektura systemu, organizacja bazy danych, opis algorytmów i implementacji fragmentów kodu.
6. Weryfikacja i walidacja – opis przyjętej metodyki testowania, wyniki przeprowadzonych testów funkcjonalnych i niefunkcjonalnych oraz zestawienie wykrytych i naprawionych błędów
7. Podsumowanie i wnioski – ocena stopnia zrealizowania założonych celów, omówienie problemów napotkanych w trakcie realizacji projektu, wskazanie potencjalnych kierunków dalszego rozwoju systemu

Rozdział 2

Analiza tematu

Współczesne przedsiębiorstwa funkcjonują w warunkach wysokiej konkurencyjności i muszą mierzyć się z dynamicznymi zmianami na rynku. Skuteczne zarządzanie wewnętrznymi procesami stanowi kluczowy element przewagi konkurencyjnej przedsiębiorstwa [1]. W wielu przypadkach można zaobserwować rozproszenie informacji lub niespójność danych pomiędzy różnymi działami firmy. Prowadzi to do oczywistych błędów w ewidencji stanów magazynowych lub opóźnień w realizacji zamówień. Literatura łączy te zjawiska z brakiem integracji procesów i danych oraz problemami koordynacji między funkcyjnej [1], [2]. Problem jest szczególnie widoczny w małych i średnich firmach, które często nie korzystają ze zintegrowanych systemów klasy ERP, a jedynie z rozproszonych narzędzi (arkusze kalkulacyjne, aplikacje magazynowe, systemy fakturowania) – co potwierdzają badania adopcji ERP w MŚP [3], [6]. Aby rozwiązać ten problem konieczne jest opracowanie spójnego systemu informatycznego, który połączy funkcje obsługi sprzedaży, zamówień i stanów magazynowych, jednocześnie zapewniający intuicyjny interfejs użytkownika oraz kontrolę dostępu opartą na rolach [1].

Zagadnienie integracji procesów biznesowych w przedsiębiorstwie jest szeroko omawiane w literaturze naukowej i branżowej. Systemy ERP, będące podstawowym narzędziem informatyzacji przedsiębiorstw, pozwalają na kompleksowe zarządzanie zasobami firmy – od finansów i kadr, przez produkcję, aż po sprzedaż i logistykę. Ich korzyści wynikają m.in. z integracji danych w czasie zbliżonym do rzeczywistego [2].

Na rynku istnieje wiele systemów klasy ERP, takich jak SAP ERP, Microsoft Dynamics 365, Comarch ERP XL czy Subiekt GT. Charakteryzują się one wysokim poziomem integracji, jednak ich wdrożenie i utrzymanie wiążą się z dużymi kosztami, co potwierdzają badania o barierach i wyzwaniach wdrożeniowych [4], [7].

Na rynku funkcjonuje wiele komercyjnych systemów wspomagających zarządzanie firmą. Krótka charakterystyka, w postaci zestawienia, najpopularniejszych z nich została przedstawiona w tabeli 2.1.

Tabela 2.1: Zestawienie komercyjnych systemów ERP

System	Opis i zalety	Wady i ograniczenia
SAP ERP [12]	Kompleksowy system do zarządzania przedsiębiorstwem. Integruje procesy finansowe, produkcyjne i logistyczne.	Bardzo wysoki koszt wdrożenia, specjalistycznego szkolenia użytkowników.
Comarch ERP XL [13]	Popularny w Polsce system dedykowany dużym firmom. Dobrze dopasowany do wymogów księgowych i handlowych.	Ograniczony dostęp przez przeglądarkę, konieczność zakupu licencji i serwera.
Microsoft Dynamics 365 [14]	Rozwiązanie chmurowe charakteryzujące się wysoką integracją.	Wysoka złożoność konfiguracji, mało elastyczny interfejs dla MŚP.
Subiekt GT [15]	Intuicyjny program sprzedażowy dla małych firm.	Brak funkcjonalności webowych, ograniczone raportowanie i współpraca międzyoddziałowa.

Przeanalizowane systemy oferują bogatą funkcjonalność, jednak ich wspólną wadą pozostaje duża złożoność wdrożenia oraz brak dostosowania do specyficznych procesów mniejszych przedsiębiorstw [7].

Wśród darmowych rozwiązań wyróżniają się projekty takie jak Odoo ERP [16], ERPNext [17] oraz inoERP [18]. Ich zaletą jest modularna budowa, dzięki której można wdrażać jedynie potrzebne funkcje, oraz możliwość samodzielnego hostowania systemu.

Mimo wielu zalet, systemy te często wymagają zaawansowanej konfiguracji, znajomości języków programowania oraz integracji z zewnętrznymi narzędziami bazodanowymi. W efekcie stają się one mało przystępne dla firm, które nie dysponują zespołem IT [11].

Równolegle rozwijane są wewnętrzne, dedykowane aplikacje, projektowane z myślą o określonych procesach w danej firmie [10], [5]. W odróżnieniu od komercyjnych systemów ERP, rozwiązania tego typu pozwalają na precyzyjne odwzorowanie struktury organizacji oraz jej wewnętrznych procesów. W połączeniu z nowoczesnymi technologiami webowymi – takimi jak ASP.NET Core [19] – możliwe staje się stworzenie elastycznego, skalowalnego systemu z przyjaznym interfejsem użytkownika i dostępem przeglądarkowym.

Coraz częściej w literaturze spotyka się zastosowanie metod inteligentnych w procesie podejmowania decyzji, w tym logiki rozmytej [8], [9]. W systemach wspierających sprzedaż takie podejście może być użyte m.in. do oceny priorytetu realizacji zamówień, określania poziomu zapasów magazynowych czy diagnozowania popytu.

To co wyróżnia realizowany projekt w porównaniu z przedstawionymi wcześniej systemami to zastosowanie logiki rozmytej do wyznaczania priorytetu zamówień w zależności od daty ich złożenia oraz produktów wchodzących w jego skład.

Algorytm wyznaczania priorytetu zamówień składa się z dwóch etapów: (1) wyznaczenia skalarnego priorytetu p jako średniej ważonej kilku cech zamówienia oraz (2) zamiana priorytetu p na rozmyte klasy ważności z wykorzystaniem funkcji przynależności μ_{trap} o kształcie trapezowym.

$$p = \frac{w_d \tilde{d} + w_m \tilde{m} + w_q \tilde{q}}{w_d + w_m + w_q}, \quad p \in [0,1] \tag{1}$$

gdzie:

w_d, w_m, w_q – wagi ważności

$\tilde{d}, \tilde{m}, \tilde{q}$ – znormalizowane wartości cech

$$\mu_{trap}(x,a,b,c,d) = \begin{cases} 0, & x \le a, \\ \dfrac{x-a}{b-a}, & a < x \le b, \\ 1, & b < x < c, \\ \dfrac{d-x}{d-c}, & c \le x < d, \\ 0, & x \ge d \end{cases} \tag{2}$$

gdzie:

x – priorytet wejściowy

a, b, c, d – punkty definiujące kształt trapezowej funkcji przynależności

Analiza istniejących rozwiązań wskazuje, że mimo szerokiej dostępności systemów ERP, ich złożoność, koszt oraz wymagana infrastruktura stanowią wysoki próg wejścia dla małych i średnich przedsiębiorstw. Z drugiej strony, systemy open-source zapewniają dużą elastyczność, lecz często wymagają specjalistycznej wiedzy technicznej [3], [4], [5], [11]. Opracowany w ramach niniejszej pracy system stanowi kompromis pomiędzy funkcjonalnością, a prostotą. Zapewnia dostęp do kluczowych procesów sprzedażowych i magazynowych w formie aplikacji webowej. Dodatkowo, dzięki zastosowaniu metod nieprecyzyjnego wyszukiwania, system pozwala na bardziej inteligentne zarządzanie procesami sprzedaży i realizacji zamówień [5], [8], [9].

Rozdział 3

Wymagania i narzędzia

W niniejszym rozdziale przedstawione zostaną wymagania funkcjonalne i niefunkcjonalne stawiane systemowi, diagram przypadków użycia oraz opis narzędzi. Całość wzbogaca zaprezentowana metodyka pracy nad projektowaniem i implementacją.

3.1 Wymagania funkcjonalne

Do wymagań funkcjonalnych, które muszą być zaimplementowane w systemie należą:

- Rejestracja i logowanie użytkowników
- Autoryzacja systemu oparta o role
- Podział systemu na funkcjonalne moduły
- Ewidencja towarów i stanów magazynowych w poszczególnych oddziałach
- Rezerwacje towarów pod zamówienia sprzedaży
- Alerty stanów krytycznych / minimalnych
- Generowanie dokumentów sprzedaży
- Priorytetyzacja zamówień
- Filtrowanie oraz sortowanie aktywnych zamówień
- Panel kierowniczy z raportami sprzedaży

3.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne zostały podzielone na kilka części, opisanych w poszczególnych podrozdziałach.

3.2.1 Wydajność

Cel wymagania

Operacje listujące i wyszukujące: czas odpowiedzi P95 ≤ 400 ms dla bazy zawierającej ~ 10 000 zamówień i ~ 100 000 pozycji.

Metoda pomiaru

- metryki aplikacyjne – wyznaczenie P95

Kryterium akceptacji

- raport k6 [20] potwierdzający P95 ≤ 400 ms dla operacji listujących

3.2.2 Skalowalność

Cel wymagania

Warstwa API pracuje w trybie stateless (bez sesji w pamięci procesu), umożliwiając uruchomienie 2 ≥ replik bez utraty funkcjonalności. Autoryzacja użytkowników realizowana w oparciu o JWT (JSON Web Token) [21].

Metoda pomiaru

Uruchomienie dwóch replik API. Wykonanie scenariusza logowania oraz pobierania danych - brak regresji.

Kryterium akceptacji

Scenariusz przechodzi przy 2 ≥ replikach, brak błędów związanych z utratą stanu sesji.

3.2.3 Bezpieczeństwo

Cel wymagania

a) Uwierzytelnianie: JWT, klucze przechowywane w menadżerze sekretów lub zmiennych środowiskowych
b) Hasła: haszowane algorytmami PBKDF2/BCrypt
c) Autoryzacja: RBAC (Role-based Access Control) – kontrola dostępu oparta na rolach na poziomie endpointów oraz interfejsu użytkownika
d) Ochrona przed nadużyciami: rate-limiting na zasobach uwierzytelniania
e) Walidacja wejścia: adnotacje i walidatory

Metoda pomiaru

- testy negatywne – rola „Sprzedawca” nie uzyskuje dostępu do zasobów „Admin” itd.

- test przeciążeniowy zasobu autoryzacji – w przypadku ≥ 100 żądań/min – odpowiedź 429

Kryterium akceptacji

Próby nieautoryzowane skutkują błędem 403, aktywny rate-limiting zwraca 429.

3.2.4 Zarządzanie konfiguracją i środowiskami

Cel wymagania

Parametry środowiskowe wyniesione poza kod źródłowy. Możliwość uruchomienia środowiska deweloperskiego w oparciu o kontenery.

Metoda pomiaru

Przegląd konfiguracji (plików appsettings.*.json, zmiennych środowiskowych, sekretów).

Kryterium akceptacji

Aplikacja startuje w środowisku deweloperskim bez zmian w kodzie, brak sekretów w repozytorium.

3.3 Diagram przypadków użycia

W podrozdziale zaprezentowano diagram przypadków użycia (Rysunek 3.1). Diagram przedstawia zestaw funkcjonalności dostępnych dla poszczególnych ról systemu. Aktorzy widoczni na diagramie odpowiadają rolom opisanym w podrozdziale *4.4 Role użytkowników w systemie*, a ich relacje z przypadkami użycia odzwierciedlają faktyczne procesy biznesowe zachodzące w systemie.

Użytkownik niezalogowany posiada dostęp wyłącznie do funkcji logowania (U1). Po uwierzytelnieniu system przydziela użytkownika do odpowiedniej roli.

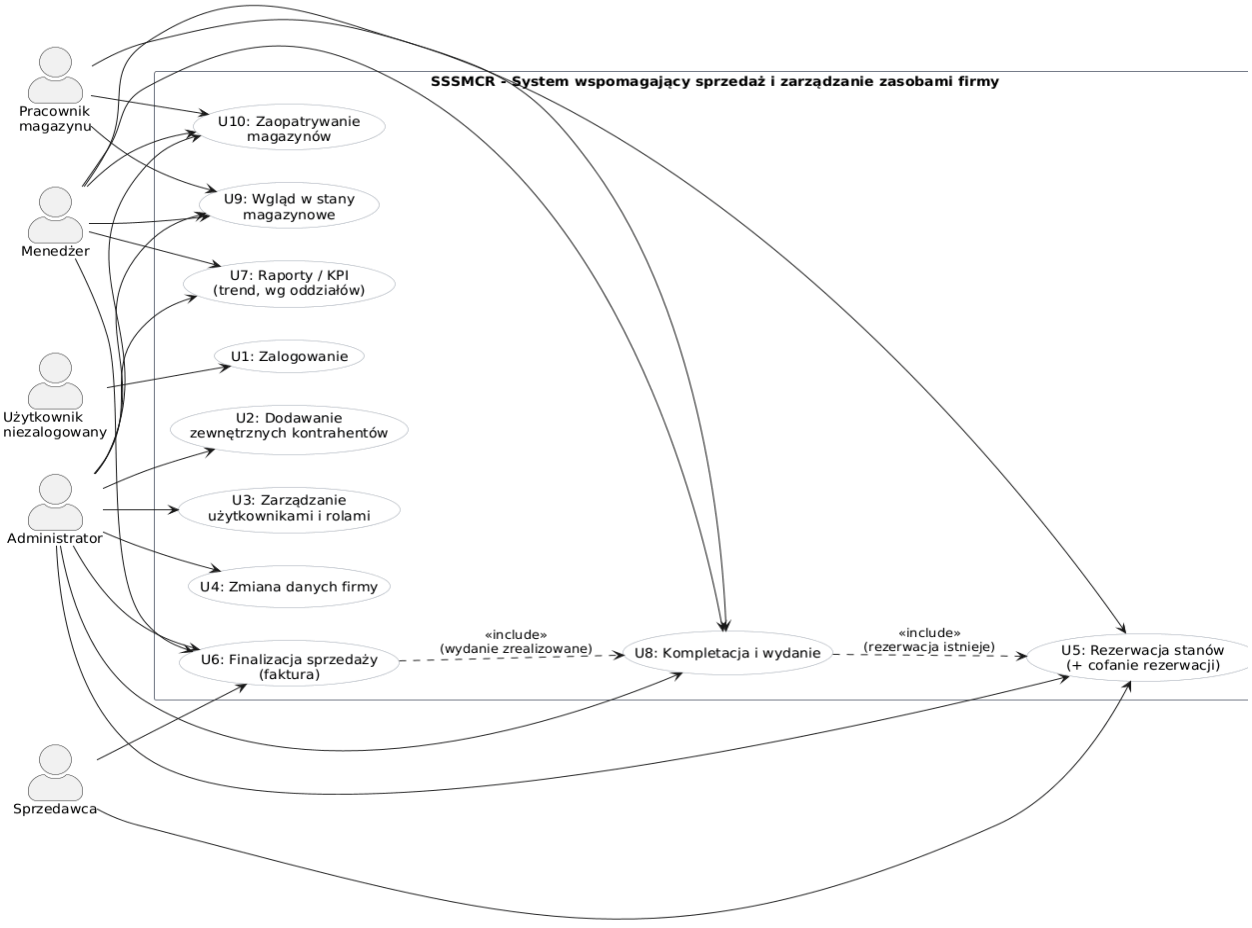
Sprzedawca realizuje proces obsługi zamówień poprzez rezerwację towarów (U5) oraz finalizację sprzedaży i generowanie faktur (U6). Przypadek U5 jest powiązany relacją *include* z U8, ponieważ rezerwacja stanowi podstawę do kompletacji zamówienia.

Magazynier wykonuje operacje logistyczne: podgląd stanów magazynowych (U9), zaopatrzenie (U10) oraz kompletację i wydanie towarów (U8). Relacja *include* między U8, a U6 wskazuje, że finalizacja wymaga wcześniejszego wydania zamówienia.

Menadżer jako rola nadzorcza posiada dostęp do operacji sprzedażowych i magazynowych, a dodatkowo może generować raporty i analizy KPI (U7).

Administrator posiada uprawnienia menadżera, a w dodatku ma możliwość zarządzania konfiguracją oraz danymi firmy w systemie. Do akcji, które może wykonywać należą m.in. zarządzanie użytkownikami i rolami (U3), edycja danych firmy (U4) oraz dodawanie zewnętrznych kontrahentów (U2).

Całość diagramu prezentuje przepływ procesów w systemie, przez rezerwację, kompletowanie i finalizację zamówień, aż po operacje logistyczne i administracyjne, pokazując pełen zakres funkcji dostępnych w systemie.



Rysunek 3.1: Diagram przypadków użycia

3.4 Narzędzia i zastosowane technologie

W podrozdziale zaprezentowane zostały narzędzia i technologie zastosowane w trakcie implementacji systemu, podzielone one zostały pod kątem użycia w odpowiedniej warstwie.

W skład warstwy backend (API) wchodzą:

- ASP.NET Core 9 (Web API) – trzon logiki biznesowej oraz interfejs REST [22]. Zastosowano kontrolery z atrybutami walidacyjnymi oraz podział na warstwy (Kontroler, warstwa mikro-usług, kontekst bazy danych).
- Entity Framework [23] – mapowanie obiektowo-relacyjne (ORM) encji w systemie na obiekty bazodanowe. W zapytaniach odczytowych użyto projekcji do **DTO** (Obiekt transferu danych) w celu minimalizacji transferu danych i kosztu operacji.
- MS SQL Server [24] – relacyjna baza danych systemu.
- Swagger [25] – oprogramowanie służące do testowania interfejsu REST, domyślnie uruchamiane wraz z API.

W skład warstwy frontend (fasady aplikacji) wchodzą:

- Blazor Server [26] – technologia Microsoftu pozwalająca na dostęp do aplikacji dla użytkowników. Umożliwia tworzenie stron internetowych oraz odwoływanie się do API.
- MudBlazor [27] – biblioteka graficzna dedykowana dla projektów Blazor. Pozwala na intuicyjne i responsywne wykorzystanie gotowych graficznych komponentów, aby strona była przejrzysta dla użytkownika.

Środowisko deweloperskie składa się z następujących narzędzi:

- JetBrains Rider [28] – IDE (zintegrowane środowisko deweloperskie). Dedykowane rozwiązanie firmy JetBrains dla projektów .NET. Rozszerza możliwości natywnych narzędzi Microsoftu, oferując dostęp do narzędzi bazodanowych oraz diagramu ERD (struktury bazy danych).
- Git [29] – System pozwalający na kontrolę wersji w projekcie i lepszą organizację pracy.
- GitKraken [30] – aplikacja wykorzystująca Git w celu intuicyjnego i prostego dostępu do repozytorium.

3.5 Metodyka pracy nad projektowaniem i implementacją

W procesie projektowania i implementacji prace prowadzono iteracyjnie, w krótkich cyklach mających na celu dostarczenie pojedynczych, kompletnych funkcjonalności. Każdy cykl rozpoczynał się od sformułowania problemu/pomysłu, a kończył wdrożeniem działającego i zabezpieczonego endpointu API, przetestowanego lokalnie.

Przebieg pojedynczej iteracji:

1. Rejestracja zadania – pomysł zapisywany jako nowe zgłoszenie w repozytorium (krótki opis, kryteria akceptacji).
2. Wydzielenie gałęzi – utworzenie w repozytorium nowej gałęzi dla zadania.
3. Implementacja funkcjonalności / naprawienie błędów zadania.
4. Testowanie funkcjonalności za pomocą narzędzia Swagger.
5. Zabezpieczenie funkcjonalności i ponowny test, tym razem jako zalogowany użytkownik.
6. Porządki i dokumentacja w kodzie.
7. Scalenie gałęzi i zamknięcie zadania.

Rozdział 4

Specyfikacja zewnętrzna

W rozdziale przedstawiono specyfikację zewnętrzną, w szczególności wymagania sprzętowe i programowe, sposób instalacji i aktywacji, sposób obsługi systemu oraz system bezpieczeństwa dostępu do aplikacji.

4.1 Wymagania sprzętowe i programowe

Wymagania sprzętowe i programowe zostały podzielone na część dotyczącą serwera i klienta i przedstawione w podrozdziałach 4.1.1 oraz 4.1.2.

4.1.1 Serwer aplikacyjny / bazy danych

- System: Windows 10/11 lub Windows Server 2019+
- CPU/RAM: min. 2 vCPU, 4-8GB RAM (zalecane 8GB)
- Dysk: 1GB na aplikację + rozmiar bazy danych
- Platforma: .NET 9 (ASP.NET Core)
- Baza danych: Microsoft SQL Server 2019+
- Porty sieciowe: 5000, 5001, 5002, 5003

4.1.2 Stanowisko klienckie

- Przeglądarka: Chrome/Firefox
- Sieć: dostęp do serwera hostującego API

4.2 Sposób instalacji

Proces instalacji obejmuje przygotowanie środowiska, publikację i konfigurację aplikacji, migrację bazy danych oraz uruchomienie.

4.2.1 Przygotowanie środowiska

Przygotowanie środowiska obejmuje następujące kroki:

- Zainstalowanie .NET 9 oraz SQL Server.
- Pobranie aplikacji z repozytorium.
- Upewnienie się, że systemowa usługa MSSQLSERVER jest uruchomiona.

4.2.2 Publikacja aplikacji

Jeśli istnieje folder *publish* z plikiem wykonywalnym *.exe* – należy pominąć budowanie solucji i przejść do konfiguracji. W przeciwnym razie w folderze z repozytorium należy uruchomić komendę w terminalu (Rysunek 4.1).

```
dotnet publish .\SSSMCR.ApiService\SSSMCR.ApiService.csproj -c Release -o
.\publish
```

Rysunek 4.1: Publikacja API

Jeśli istnieje już folder publish-web z analogicznym plikiem wykonywalnym .exe – należy pominąć build i przejść do konfiguracji. W przeciwnym razie należy uruchomić w terminalu komendę (Rysunek 4.2).

```
dotnet publish .\SSSMCR.Web\SSSMCR.Web.csproj -c Release -o .\publish-web
```

Rysunek 4.2: Publikacja fasady aplikacji

4.2.3 Konfiguracja

W celu konfiguracji należy ustawić zestaw zmiennych środowiskowych w systemie Windows z poziomu PowerShell (Rysunek 4.3).

```
[System.Environment]::SetEnvironmentVariable("Jwt__Key",
"super_long_random_dev_secret_at_least_64_chars", "User")
[System.Environment]::SetEnvironmentVariable("Jwt__Issuer", "SSSMCR",
"User")
[System.Environment]::SetEnvironmentVariable("Jwt__Audience",
"SSSMCRUsers", "User")
[System.Environment]::SetEnvironmentVariable("Jwt__ExpiresMinutes", "60",
"User")
[System.Environment]::SetEnvironmentVariable(
"ConnectionStrings__DefaultConnection",
"Server=localhost;Database=SssmcrDb;Trusted_Connection=True;TrustServerCert
ificate=True",
"User"
)
```

Rysunek 4.3: Ustawienie zmiennych środowiskowych

4.2.4 Migracje bazy danych

Aplikacja uruchomiona w trybie Produkcji może wykonać migracje automatycznie lub z terminalu (Rysunek 4.4).

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Rysunek 4.4: Migracja i aktualizacja bazy danych

Po wykonaniu przedstawionych komend baza danych utworzy się samoistnie. Nazwa migracji musi być unikatowa. W przypadku ponownego tworzenia migracji należy usunąć istniejącą *InitialCreate* lub stworzyć nową o innej nazwie.

4.2.5 Uruchomienie API

W celu uruchomienia API należy wpisać komendę (Rysunek 4.5).

```
cd .\publish
.\SSSMCR.ApiService.exe
```

Rysunek 4.5: Uruchomienie API

4.2.6 Uruchomienie fasady aplikacji

Aby uruchomić fasadę aplikacji należy wykonać komendę (Rysunek 4.6).

```
cd .\publish-web
.\SSSMCR.Web.exe
```

Rysunek 4.6: Uruchomienie fasady aplikacji

Przedstawione komendy można zastąpić uruchomieniem pliku wykonywalnego z poziomu eksploratora plików w odpowiednich folderach. Po starcie API powinno być dostępne pod adresem *http://localhost:5000* oraz *https://localhost:5001*. Fasada aplikacji powinna być dostępna pod adresem *http://localhost:5002 oraz https://localhost:5003*.

4.3 Sposób aktywacji

W podrozdziale przedstawiono sposób aktywacji systemu z uwzględnieniem ról i kont startowych oraz zmiany danych firmy.

4.3.1 Role i konta startowe

Przy pierwszym uruchomieniu aplikacji tworzeni są użytkownicy początkowi oraz role: Administrator, Menadżer (eng. Manager), Sprzedawca (eng. Salesman), Magazynier (eng. WarehouseWorker). Domyślne hasło dla każdego konta to *hashed123*. Zalecana jest natychmiastowa zmiana haseł lub usunięcie niepotrzebnych użytkowników.

4.3.2 Zmiana danych firmy

Użytkownik zalogowany jako administrator ma dostęp do panelu, w którym zdefiniowane są dane firmy. Zalecana jest ich zmiana na wartości rzeczywiste przed generowaniem dokumentów sprzedaży dla zrealizowanych zamówień. Po dokonaniu powyższych zmian możliwe jest korzystanie ze wszystkich funkcji aplikacji.

4.4 Role użytkowników w systemie

Zwiększając bezpieczeństwo systemu wydzielono następujące role użytkowników:

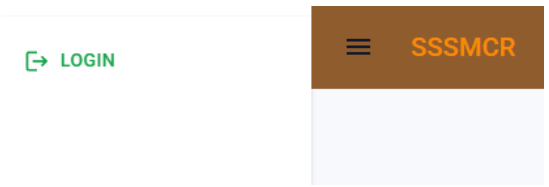
- Administrator – użytkownik mogący zarządzać użytkownikami, danymi firmy, zewnętrznymi kontrahentami oraz oddziałami firmy. Dodatkowo ma dostęp do wszystkich funkcji systemu.
- Sprzedawca – użytkownik mający dostęp do panelu obsługi zamówień. Może on rezerwować produkty w magazynach lub zwalniać niepoprawne rezerwacje. Dodatkowo ma dostęp do modułu generowania dokumentów sprzedaży dla zrealizowanych zamówień.
- Magazynier – użytkownik mający dostęp do modułu magazynowego. Może kompletować rezerwacje, a także składać zamówienia u zewnętrznych kontrahentów (w przypadku niskiego stanu zasobów w magazynie).
- Menadżer – użytkownik nadzorujący. Ma dostęp do wszystkich modułów Sprzedawcy oraz Magazyniera. Dodatkowo jest w stanie generować raporty sprzedaży w dedykowanym do tego panelu.

4.5 Sposób obsługi

Niniejszy podrozdział prezentuje sposób obsługi zaimplementowanego systemu.

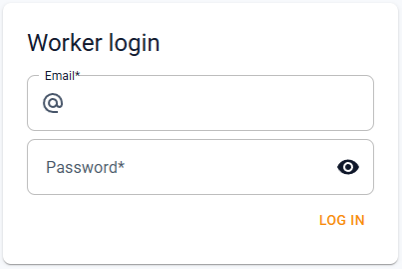
4.5.1 Logowanie

Po uruchomieniu fasady aplikacji należy wybrać przycisk *LOGIN* znajdujący się w lewym górnym rogu rozwijanego menu (Rysunek 4.7). Użytkownik zostanie przeniesiony na podstronę *login*.



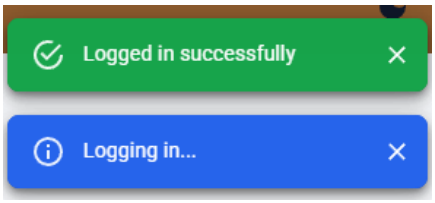
Rysunek 4.7: Umiejscowienie przycisku logowania

Na ekranie pojawi się okno logowania (Rysunek 4.8). Użytkownik musi podać adres email oraz hasło skojarzone z jego kontem. Aplikacja nie przewiduje ręcznej rejestracji użytkownika. Zamiast tego administrator ma możliwość dodawania nowych kont w systemie.



Rysunek 4.8: Panel logowania

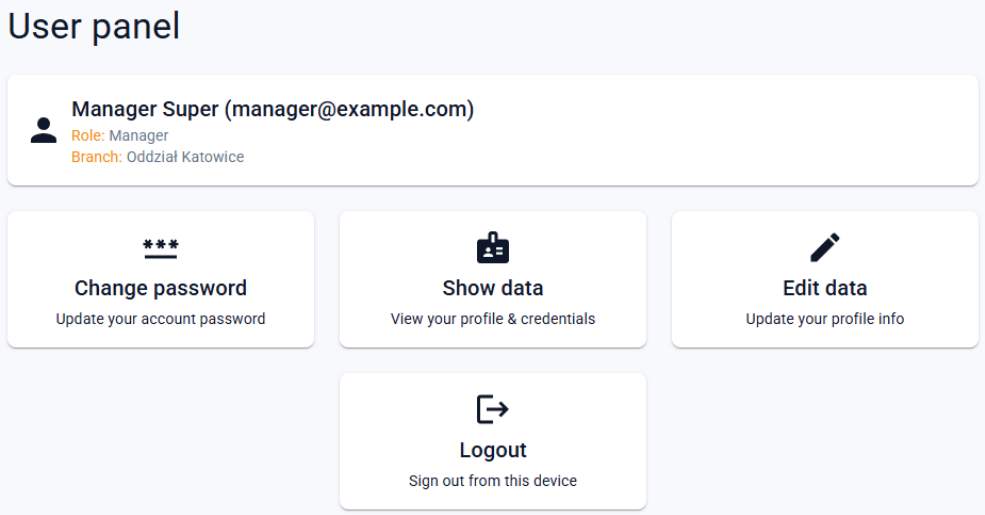
Po udanym logowaniu w prawym górnym rogu okna aplikacji pojawi się odpowiednie powiadomienie (Rysunek 4.9). Użytkownik zostanie również przeniesiony na panel z danymi jego konta.



Rysunek 4.9: Powiadomienie świadczące o poprawnym zalogowaniu

4.5.2 Zmiana danych konta

Na podstronie *userpanel* zalogowany użytkownik ma możliwość zmiany hasła swojego konta, a także podstawowych danych takich jak imię i nazwisko (Rysunek 4.10). Adres email oraz rola w systemie należą do danych, które aktualizować może wyłącznie administrator.

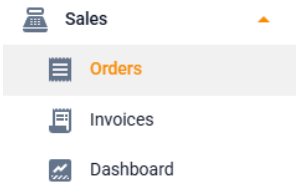


Rysunek 4.10: Panel użytkownika

4.5.3 Rezerwacja produktów

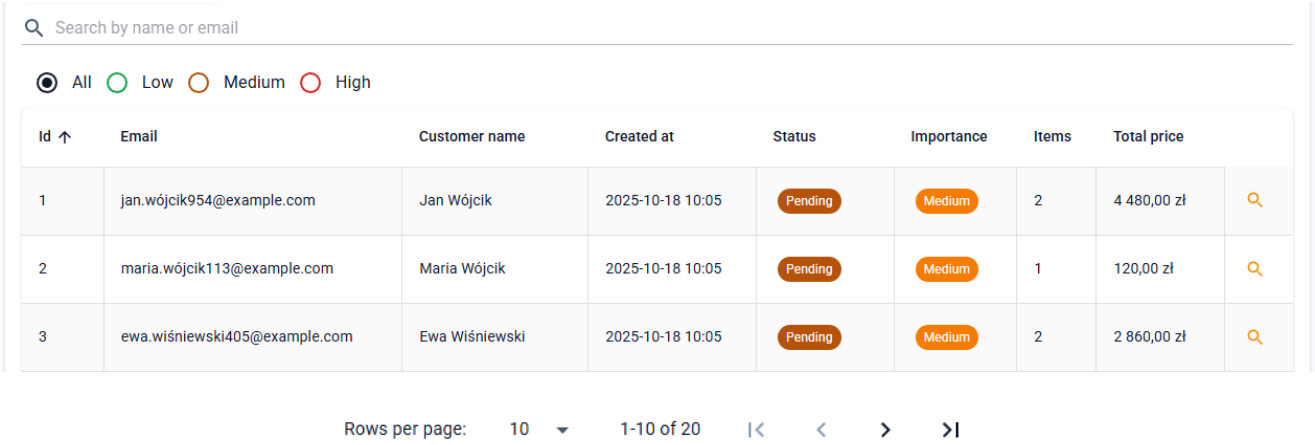
Do tej jak i do kolejnych funkcjonalności dostęp mają użytkownicy określeni w podrozdziale *4.4 Role użytkowników w systemie*.

Aby przejść do zakładki z zamówieniami należy nacisnąć przycisk *Orders* znajdujący się w rozwijanym menu (Rysunek 4.11).



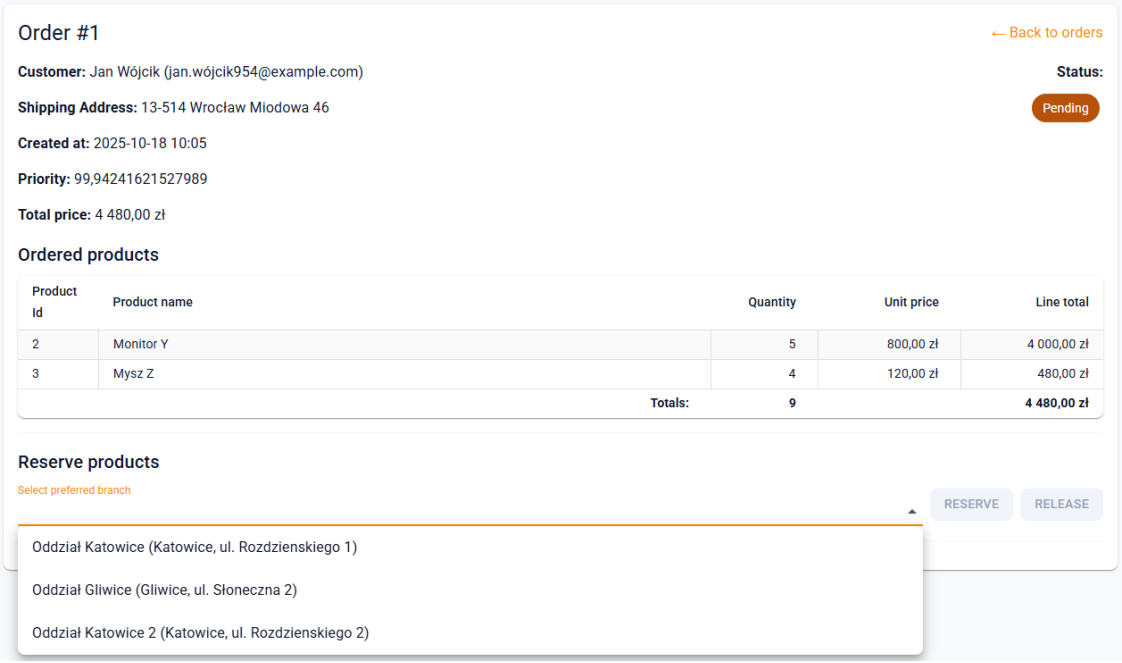
Rysunek 4.11: Umiejscowienie zakładki zamówień

Użytkownikowi ukaże się tabela zawierająca zamówienia widniejące w systemie. Użytkownik ma możliwość filtracji zamówienia oraz podglądu jego szczegółów. Aby to zrobić należy wybrać ikonę lupy w odpowiednim wierszu. Tabela ma również funkcjonalność paginacji. Kolejne strony mogą zostać wyświetlone za pomocą przycisków w prawym dolnym rogu (Rysunek 4.12).

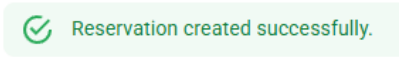


Rysunek 4.12: Tabela zamówień

Sprzedawca ma możliwość wybrania preferowanego oddziału, który zrealizuje zamówienie (Rysunek 4.13). Jeśli stan magazynowy jednak na to nie pozwala, zamówienie zostanie przejęte przez inny magazyn. Poprawne przyjęcie zamówienia skutkuje powiadomieniem (Rysunek 4.14).



Rysunek 4.13: Szczegóły zamówienia



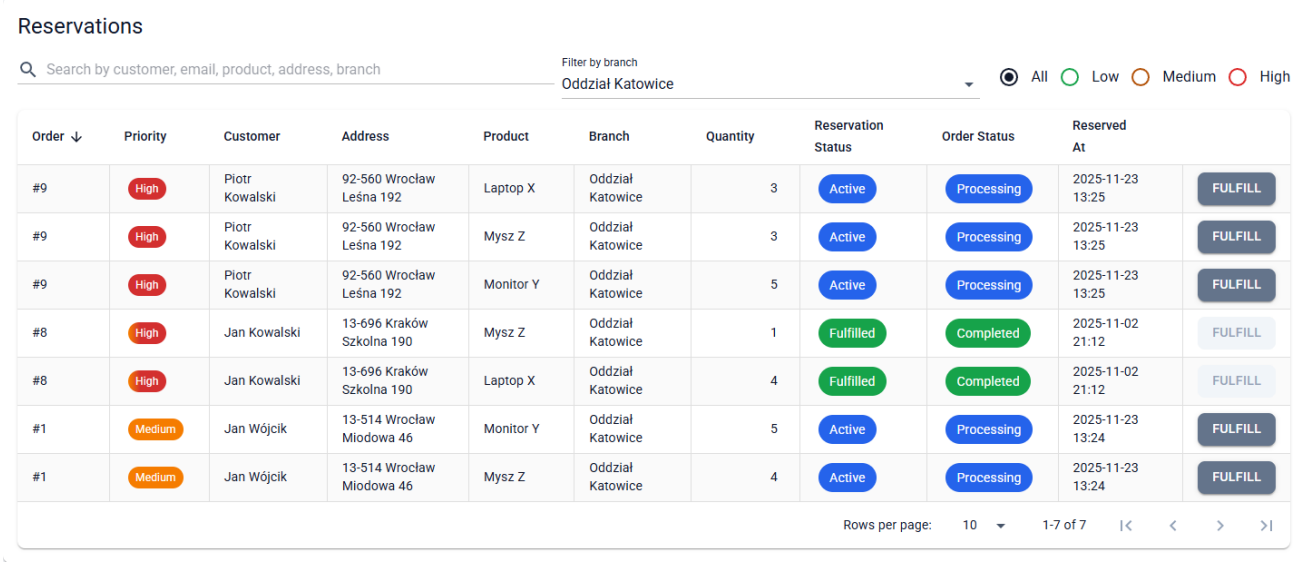
Rysunek 4.14: Powiadomienie świadczące o poprawnym przyjęciu zamówienia

4.5.4 Kompletowanie i wydawanie zamówień

Na kompletowanie zamówień w magazynie pozwala podstrona, do której można przejść naciskając przycisk *Reservations* (Rysunek 4.15).

Rysunek 4.15: Umiejscowienie zakładki kompletacji zamówień

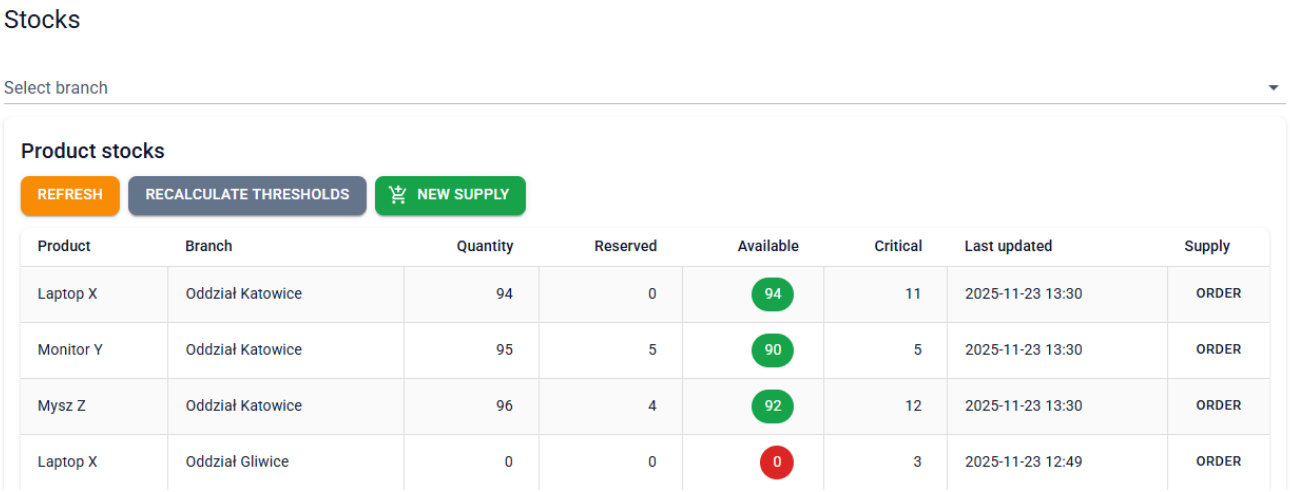
Widnieje na niej tabela analogiczna jak dla zamówień (Rysunek 4.16). Użytkownik ma możliwość podglądu rezerwacji w danym magazynie oraz wypełnienia ich przyciskiem *FULFILL*. Po wypełnieniu wszystkich rezerwacji dla danego zamówienia, całe zamówienie zmienia status na *Completed*, co jest widoczne również na podstronie *Orders* (Rysunek 4.12).



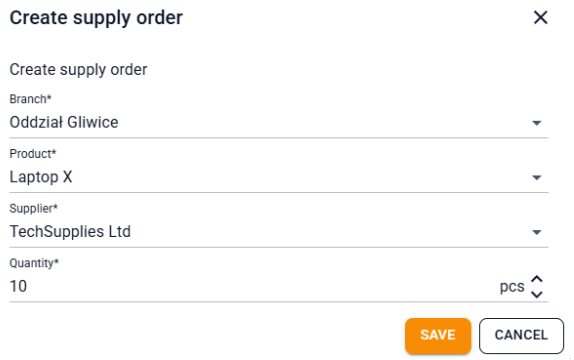
Rysunek 4.16: Tabela rezerwacji

4.5.5 Zwiększanie stanów magazynowych

Zamawianie produktów realizowane jest na podstronie *Stocks* (Rysunek 4.17). Magazynier ma możliwość podglądu stanów magazynowych oraz zwiększania ich za pomocą *NEW SUPPLY* lub *ORDER* w wierszach tabeli (Rysunek 4.18). Dodatkowo funkcja *RECALCULATE THRESHOLDS* aktualizuje krytyczne wartości każdego z produktów o średnią sprzedaży z ostatniego miesiąca.

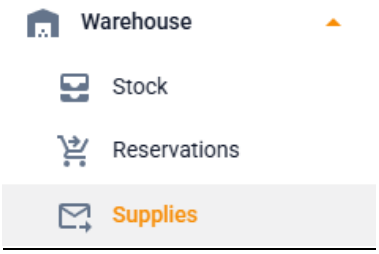


Rysunek 4.17: Tabela stanów magazynowych



Rysunek 4.18: Tworzenie nowego zamówienia

Zamówione produkty muszą być odpowiednio oznaczone w momencie otrzymania, aby mieć wpływ na stan magazynowy. Realizuje się to na podstronie *Supplies*, do której należy przejść zgodnie z Rysunkiem 4.19. Na tej stronie są widoczne wszystkie przeszłe zamówienia (Rysunek 4.20).

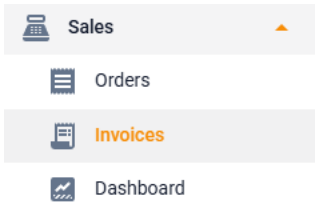


Rysunek 4.19: Umiejscowienie zakładki dostaw

Rysunek 4.20: Historia dostaw

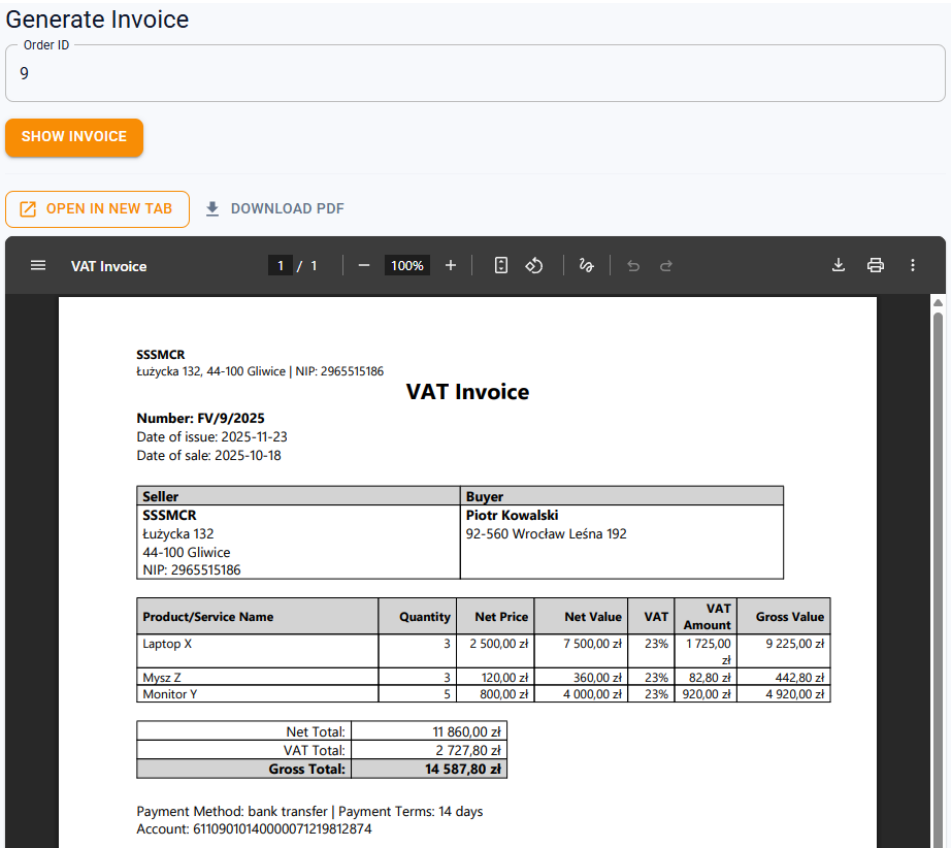
4.5.6 Finalizacja sprzedaży

Zamówienia oznaczone jako *Completed* pozwalają na wygenerowanie faktury. Aby to zrobić należy przejść do zakładki *Invoices* (Rysunek 4.21) lub bezpośrednio - ze szczegółów danego zamówienia (Rysunek 4.22). Przykład wygenerowanej faktury zaprezentowano na Rysunku 4.23.



Rysunek 4.21: Umiejscowienie zakładki fakturowania

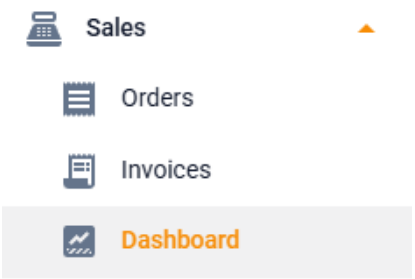
Rysunek 4.22: Przycisk fakturowania zamówienia



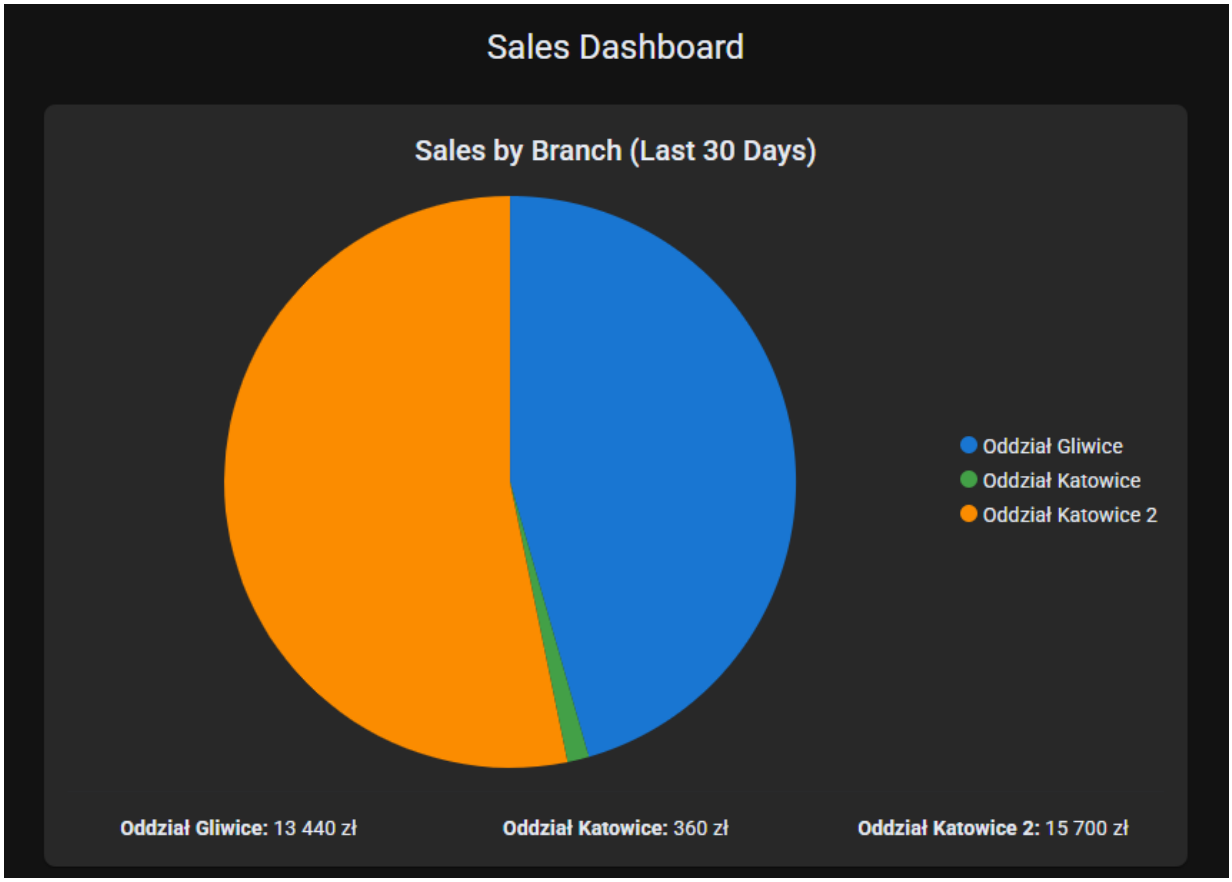
Rysunek 4.23: Przykładowa wygenerowana faktura

4.5.7 Raporty/KPI

Generowanie raportów realizowane jest na podstronie dostępnej pod przyciskiem *Dashboard* (Rysunek 4.24). Dla użytkownika widoczne staną się wykresy trendów sprzedażowych (Rysunek 4.25).



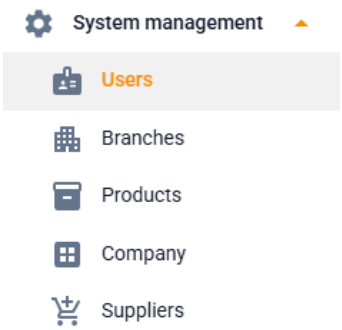
Rysunek 4.24: Umiejscowienie zakładki analitycznej



Rysunek 4.25: Wykres sprzedaży

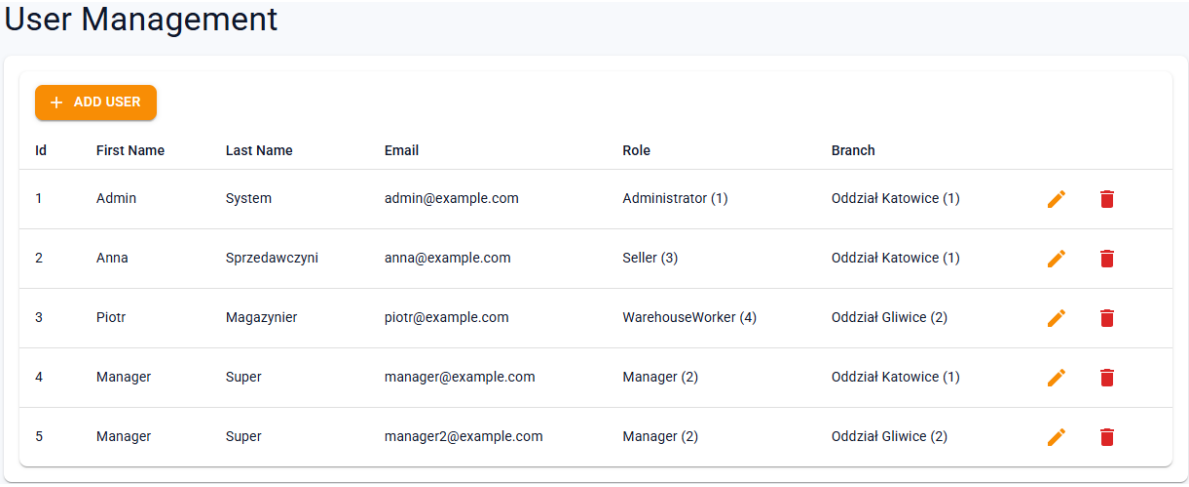
4.5.8 Zarządzanie użytkownikami i oddziałami

Zarządzanie użytkownikami i oddziałami dostępne są w zakładkach *Users* oraz *Branches* (Rysunek 4.26).



Rysunek 4.26: Umiejscowienie zakładek administracyjnych

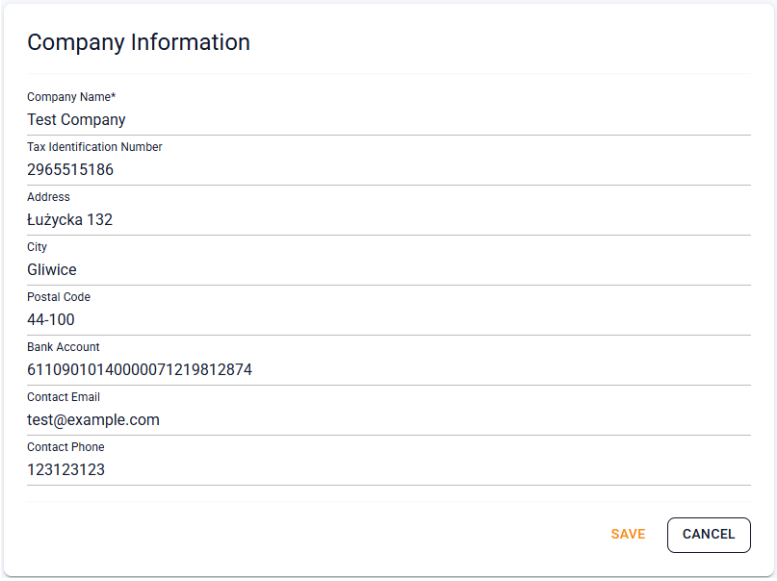
Każda z zakładek zawiera tabelę edycyjną umożliwiającą dodawanie, usuwanie lub edytowanie rekordów (Rysunek 4.27).



Rysunek 4.27: Tabela użytkowników

4.5.9 Zmiana danych firmy

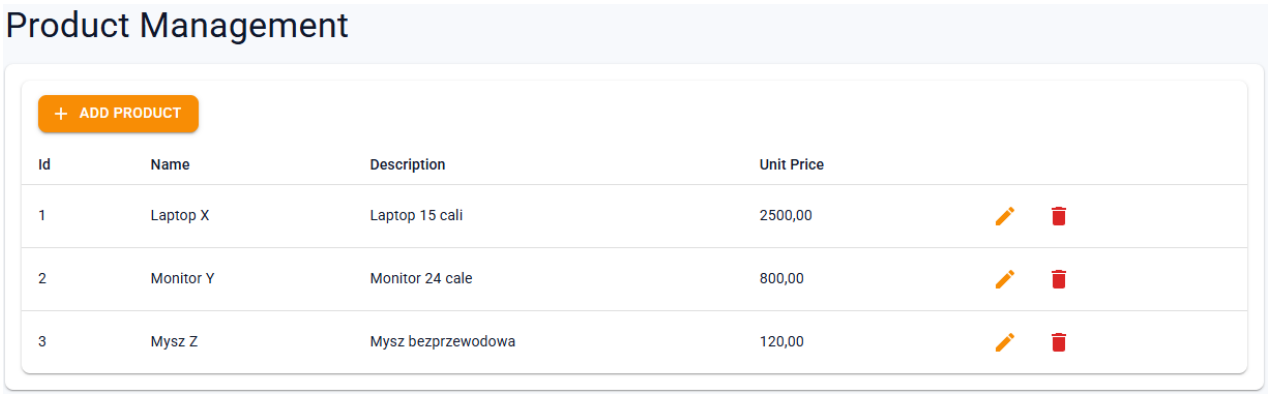
Dane firmy mogą być zaktualizowane na podstronie *Company* (Rysunek 4.28).



Rysunek 4.28: Formularz edycji danych firmy

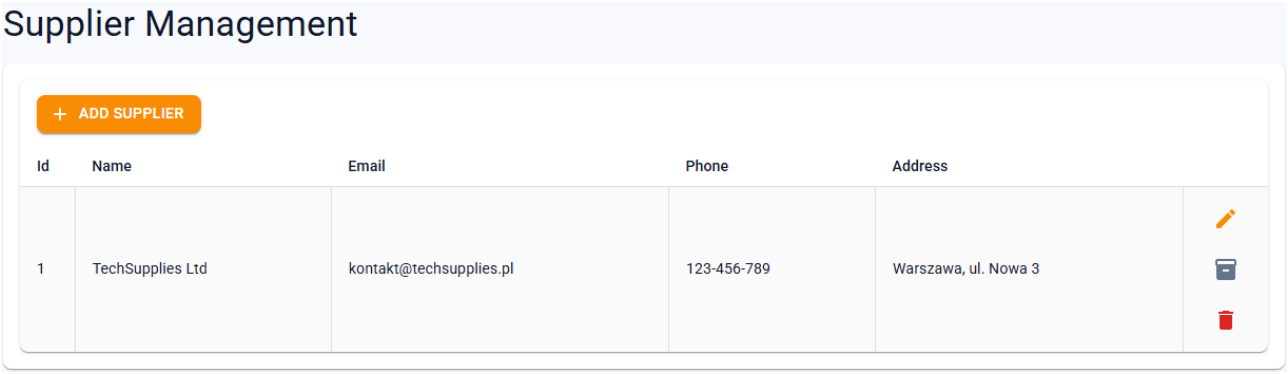
4.5.10 Dodawanie kontrahentów i produktów

Produkty widniejące w systemie mogą zostać dodane na podstronie *Products* (Rysunek 4.29).



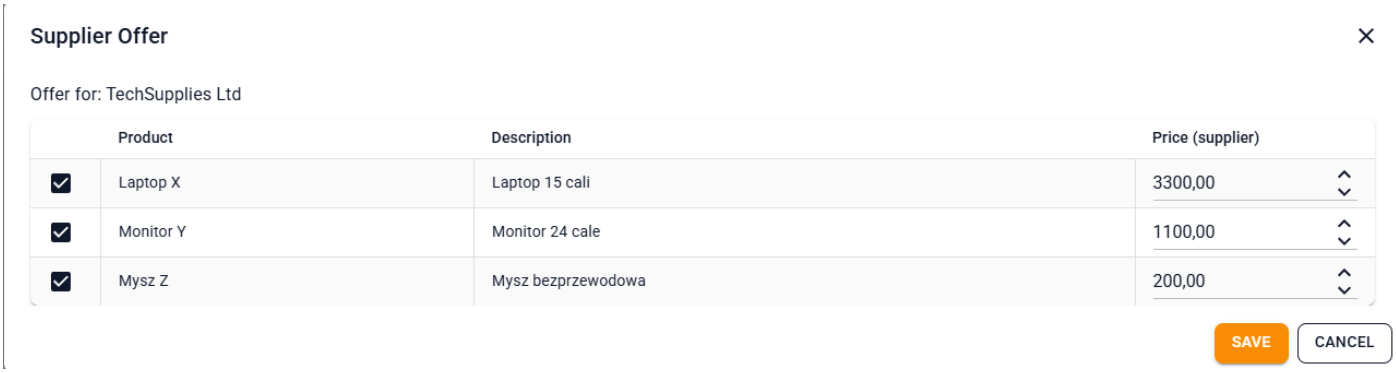
Rysunek 4.29: Tabela produktów

Analogicznie w przypadku dostawców – na podstronie *Suppliers* (Rysunek 4.30).



Rysunek 4.30: Tabela dostawców

Możliwe jest również przypisywanie konkretnych produktów zewnętrznym dostawcom (Rysunek 4.31). Aby przejść do tej funkcjonalności należy nacisnąć ikonę szarego koszyka (Rysunek 4.30) w menu akcji dla dostawcy.



Rysunek 4.31: Powiązania dostawców z produktami

Rozdział 5

Specyfikacja wewnętrzna

W niniejszym rozdziale przedstawiono ogólną ideę systemu, jego architekturę, strukturę bazy danych, przegląd ważniejszych algorytmów i metod.

5.1 Przedstawienie idei

Ideą systemu jest stworzenie spójnego rozwiązania webowego wspomagającego zarządzanie procesami w małych i średnich przedsiębiorstwach. System ma eliminować problemy wynikające z rozproszenia informacji, opóźnień w realizacji oraz trudności w realizacji sprzedaży. Główną zasadą projektową było modułowe podejście – każda funkcjonalność stanowi niezależny komponent odzwierciedlający rzeczywiste procesy w firmie: magazynowanie, sprzedaż, generowanie dokumentów oraz zarządzanie administracyjne. Całość została oparta na architekturze klient-serwer z podziałem na warstwy: warstwę prezentacji (frontend), warstwę logiki biznesowej (backend) oraz warstwę danych (relacyjna baza danych). Modułowy podział pozwala na wygodną implementację mechanizmu ról i uprawnień (RBAC), umożliwiający ograniczenie dostępu do wybranych funkcji – tak aby poszczególni pracownicy mieli dostęp tylko z dedykowanych dla siebie narzędzi. Dodatkowo system wyróżnia się zastosowaniem elementów logiki rozmytej, co umożliwia bardziej inteligentną i dynamiczną obsługę procesów sprzedażowych.

5.2 Architektura systemu

System został zaprojektowany jako trójwarstwowa aplikacja webowa.

5.2.1 Warstwa prezentacji (frontend)

Warstwa prezentacji wykorzystuje technologie **Blazor Server** oraz **MudBlazor**. Odpowiada za interfejs użytkownika oraz interakcję z API. Blazor Server to model hostingu dla frameworka Blazor, w którym logika aplikacji i komponenty UI (interfejsu użytkownika) są wykonywane na serwerze ASP.NET Core. Renderowanie po stronie serwera zapewnia minimalne wymagania sprzętowe po stronie klienta. W opracowanym systemie mechanizm uwierzytelniania został oparty na tokenach JWT:

- Po poprawnym zalogowaniu backend generuje token JWT,
- Token jest przechowywany po stronie klienta w pamięci aplikacji
- Każde żądanie wysyłane z Blazor do API zawiera nagłówek autoryzacji

Dzięki temu system łączy wygodę renderowania Blazor Server z elastycznością klasycznego uwierzytelniania JWT używanego w aplikacjach SPA (Single Page Application).

5.2.2 Warstwa logiki biznesowej (backend)

Warstwa wykorzystuje technologię ASP.NET Core 9 Web API. Odpowiada za obsługę logiki biznesowej i komunikację z bazą danych. Jest to aplikacja REST, całkowicie bezstanowa, co umożliwia łatwe skalowanie poziome (uruchamianie wielu replik API pod równomiernym rozkładem obciążenia). Całość zrealizowano w podejściu modularnym inspirowanym architekturą mikroserwisową.

Wstrzykiwanie zależności

Backend w pełni wykorzystuje wbudowany w ASP.NET Core mechanizm **Dependency Injection (DI)** [31]. Dzięki temu:

- Logika biznesowa została oddzielona od kontrolerów
- Serwisy są rejestrowane jako zależności (AddScoped)
- Warstwa dostępu do danych jest wstrzykiwana do serwisów
- Testowanie jednostkowe jest uproszczone
- Interfejsy serwisów umożliwiają łatwą wymianę informacji

Modularne podejście do logiki biznesowej

Chociaż system nie jest fizycznie złożony z osobnych mikroserwisów, każdy moduł został zaprojektowany jako wyizolowana jednostka funkcjonalna, przypominająca mikroserwis pod względem struktury. Każdy moduł posiada:

- Własny serwis (lub kilka) odpowiedzialny za logikę biznesową
- Własny kontroler API

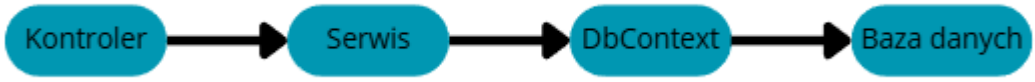
To podejście powoduje, że projekt jest skalowalny, czytelny i łatwy do rozbudowy.

Przepływ danych

Logika biznesowa nie jest implementowana w kontrolerach – zamiast tego kontrolery delegują zadania do dedykowanych serwisów.

Typowy przepływ danych przedstawiono na Rysunku 5.1.

1. Kontroler odbiera żądanie i wykonuje walidację danych wejściowych.
2. Serwis przetwarza logikę biznesową obsługiwanej funkcjonalności.
3. **DbContext** (kontekst bazy danych) komunikuje się z bazą danych używając mapowania obiektowo-relacyjnego.
4. Wynik wraca do kontrolera jako DTO.



Rysunek 5.1: Struktura przepływu danych logiki biznesowej

5.2.3 Warstwa danych

W aplikacji wykorzystano relacyjną bazę danych **SQL Server.** Baza składa się z tabel powiązanych relacjami poprzez klucze główne i klucze obce. Baza danych nie jest tworzona ręcznie, ale z wykorzystaniem mapowania obiektowo-relacyjnego (ORM). Technologia **Entity Framework** pozwala na prostą zamianę obiektów (w języku programowania wysokiego poziomu) na encje bazy danych. W praktyce wymaga to jedynie umieszczenia odpowiednych adnotacji przed definicją klasy obiektu, jak i przed jej polami. Następnie należy wykonać migrację projektu i zaktualizować bazę danych – tak aby zmiany wprowadzone w klasach miały wpływ na strukturę bazy danych.

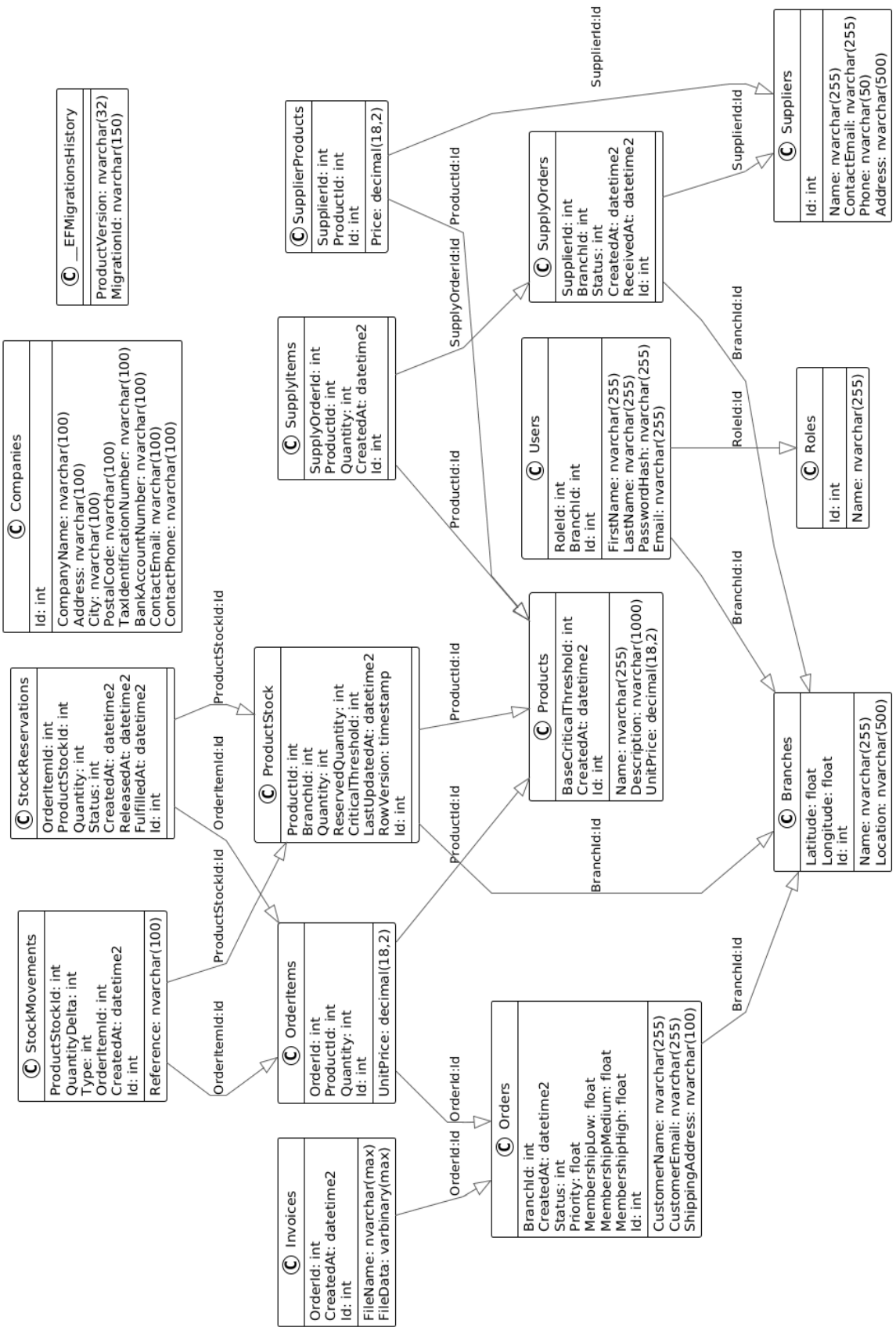
5.3 Struktury danych i organizacja bazy danych

Baza danych została zorganizowana zgodnie z zasadami normalizacji oraz logiką procesów biznesowych. Jej strukturę przedstawiono na diagramie (Rysunek 5.2).

W jej skład wchodzą następujące tabele:

- **Branch** – definiuje oddziały firmy wraz z lokalizacją i współrzędnymi geograficznymi. Stanowi podstawę podziału stanów magazynowych.
- **Company** – przechowuje dane konfiguracyjne firmy, takie jak adres, identyfikator podatkowy czy informacje kontaktowe, używane przy generowaniu dokumentów sprzedaży.
- **Invoice** – odwzorowuje wygenerowane faktury sprzedaży, przechowuje ich nazwę, binarną zawartość oraz powiązanie z zamówieniem. Pełni rolę archiwum dokumentów sprzedaży.
- **Order** – reprezentuje zamówienia klientów. Zawiera dane kontaktowe, status oraz wartości funkcji przynależności dla logiki rozmytej. Powiązana jest z oddziałem realizującym oraz listą pozycji (przedmiotów) w zamówieniu.
- **OrderItem** – przechowuje szczegółowe pozycje zamówienia, obejmując produkt, cenę jednostkową oraz liczbę. Stanowi podstawę procesu rezerwacji i kompletacji zamówień.
- **Product** – tabela przechowuje podstawowe informacje o produktach dostępnych w firmie, takie jak nazwa, opis, cena jednostkowa oraz bazowy próg krytyczny. Stanowi centralną encję powiązaną z rezerwacjami, stanami magazynowymi oraz zamówieniami.
- **ProductStock** – reprezentuje liczbę produktów w poszczególnych oddziałach firmy, obejmując stan rzeczywisty, liczbę zarezerwowanych oraz dynamiczny próg krytyczny.
- **Role** – określa role użytkowników w systemie, będąc podstawą mechanizmu RBAC. Tabela powiązana jest relacją jeden-do-wielu z użytkownikami.
- **StockMovement** – rejestrowana jest w niej każda zmiana stanów magazynowych danego produktu, wraz z informacją o typie operacji (przyjęcie, wydanie, korekta) oraz powiązaniu z rezerwacją lub zamówieniem. Pełni funkcję audytową dla ruchów magazynowych i jest podstawą dla tworzenia wykresów analitycznych.

- **StockReservation** – odwzorowuje rezerwacje produktów pod konkretne pozycje zamówień sprzedaży, śledząc liczbę, status oraz moment zwolnienia i realizacji. Zapewnia spójność stanów pomiędzy procesem sprzedażowym, a magazynowym.
- **Supplier** – zawiera dane kontaktowe oraz adresowe kontrahentów, od których firma zamawia towary. Stanowi część modułu zakupowego i jest powiązana z ofertami produktów oraz zamówieniami dostaw.
- **SupplierProduct** – łączy produkty z dostawcami, umożliwiając modelowanie ofert towarowych w systemie zgodnych z rzeczywistymi ofertami dostawców. Realizuje relację wiele-do-wielu.
- **SupplyOrder** – reprezentuje zamówienia składane do zewnętrznych dostawców, obejmujące status, wskazany oddział oraz daty utworzenia i przyjęcia. Zawiera listę pozycji dostawy i służy do odtwarzania historii zakupów.
- **SupplyItem** – stanowi listę pozycji w zamówieniu dostawy, analogicznie do tabeli OrderItem
- **User** – przechowuje dane użytkowników systemu, w tym hasło (w postaci hasza), dane identyfikacyjne i przypisanie do oddziału.



Rysunek 5.2: Diagram bazy danych

5.4 Implementacja wybranych fragmentów

W niniejszym podrozdziale przedstawiono implementację najważniejszych elementów systemu, które stanowią kluczowe komponenty oraz pokazują realizację przyjętych założeń projektowych. Wybrane fragmenty obejmują algorytm oceny priorytetu zamówień oparty o logikę rozmytą, implementację funkcji przynależności trapezowej, integrację algorytmu z modułem zamówień oraz przykładowe elementy warstwy API realizowane z wykorzystaniem wstrzykiwania zależności.

5.4.1 Algorytm oceny priorytetu zamówień

Jednym z kluczowych elementów systemu jest algorytm służący do określania priorytetu realizacji zamówienia. Został on omówiony w rozdziale 2. Analiza tematu. Celem tej funkcjonalności jest wsparcie pracowników działu sprzedaży oraz magazynu w szybkim identyfikowaniu zamówień, które powinny zostać zrealizowane w pierwszej kolejności.

Algorytm wykorzystuje dwa etapy przetwarzania danych:

1. **Wyznaczenie priorytetu skalarnego** na podstawie cech zamówienia, takich jak liczba pozycji, wiek zamówienia oraz łączna cena zamówienia. Wartości są wcześniej normalizowane i agregowane z użyciem wag (Rysunek 5.3).
2. **Przekształcenie priorytetu** na przynależność do rozmytych kategorii ważności za pomocą funkcji trapezowej (Rysunek 5.4).

Takie podejście pozwala na elastyczną ocenę zamówienia w sytuacji, w których klasyczne metody sortowania nie odzwierciedlają w pełni ważności zamówień.

```
public double CalculatePriority(Order order, IEnumerable<Order> allOrders)
{
    var maxAge = allOrders.Max(o => (DateTime.Now - o.CreatedAt).Total-
Days);
    var maxValue = allOrders.Max(o => o.Items.Sum(i => i.TotalPrice));
    var maxItems = allOrders.Max(o => o.Items.Count);

    var ageFactor = (DateTime.UtcNow - order.CreatedAt).TotalDays / maxAge;
    var valueFactor = (double)order.Items.Sum(i => i.TotalPrice) / (do-
uble)maxValue;
    var itemFactor = (double)order.Items.Count / maxItems;

    const double wAge = 0.5;
    const double wValue = 0.4;
    const double wItem = 0.1;

    var priority = wAge * ageFactor + wValue * valueFactor + wItem * item-
Factor;

    return priority * 100.0;
}
```

Rysunek 5.3: Implementacja metody obliczającej priorytet zamówienia

5.4.2 Trapezowa funkcja przynależności

Trapezowa funkcja przynależności pełni kluczową rolę w procesie przemapowania wartości priorytetu na rozmyte kategorie (Rysunek 5.4). Jest to typowa funkcja stosowana w systemach opartych na logice rozmytej, charakteryzująca się czterema punktami granicznymi, które określają jej kształt. Funkcja ta umożliwia łagodne przejścia pomiędzy poszczególnymi poziomami przynależności, co pozwala uniknąć skokowych zmian klasyfikacji zamówień. Wzór funkcji został przedstawiony w rozdziale 2. Analiza tematu.

```
public class FuzzyPriorityEvaluatorService
{
    private double Trapezoidal(double x, double a, double b, double c, do-
uble d)
    {
        if (x <= a || x >= d) return 0;
        else if (x >= b && x <= c) return 1.0;
        else if (x > a && x < b) return (x - a) / (b - a);
        else return (d - x) / (d - c);
    }

    public (double Low, double Medium, double High) Evaluate(double prio-
rity)
    {
        var low = (a: 0, b: 0, c: 30, d: 50);
        var medium = (a: 30, b: 50, c: 70, d: 85);
        var high = (a: 70, b: 85, c: 100, d: 100);

        var uLow = Trapezoidal(priority, low.a, low.b, low.c, low.d);
        var uMedium = Trapezoidal(priority, medium.a, medium.b, medium.c,
medium.d);
        var uHigh = Trapezoidal(priority, high.a, high.b, high.c, high.d);

        return (uLow, uMedium, uHigh);
    }
}
```

Rysunek 5.4: Implementacja serwisu obliczającego przynależność do zbioru

5.4.3 Integracja algorytmu z modułem zamówień

Aby algorytm mógł być wykorzystywany w praktyce, został zintegrowany z modułem obsługującym zamówienia sprzedażowe. W momencie pobrania istniejącego zamówienia system oblicza priorytet i zwraca użytkownikowi przynależności do poszczególnych klas (Rysunek 5.5).

Metoda *CalculatePriority* (Rysunek 5.3) jest wywoływana bezpośrednio, gdyż jej implementacja znajduje się w tym samym serwisie. Wywołanie funkcji trapezoidalnej wykorzystuje wstrzykiwanie zależności.

```
var orders = await query.ToListAsync(ct);

var allOrders = await GetAllAsync(ct);
foreach (var order in orders)
{
    order.Priority = CalculatePriority(order, allOrders);

    var fuzzy = fuzzyService.Evaluate(order.Priority);
    order.MembershipLow = fuzzy.Low;
    order.MembershipMedium = fuzzy.Medium;
    order.MembershipHigh = fuzzy.High;
}

if (!string.IsNullOrEmpty(importance))
{
    orders = importance.ToLower() switch
    {
        "low" => orders.Where(o => o.MembershipLow > 0.5).ToList(),
        "medium" => orders.Where(o => o.MembershipMedium > 0.5).ToList(),
        "high" => orders.Where(o => o.MembershipHigh > 0.5).ToList(),
        _ => orders
    };
}
```

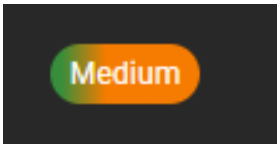
Rysunek 5.5: Obliczenie priorytetu pobranego zamówienia

Następnie API wykorzystuje DTO, aby zwrócić dane do fasady aplikacji. Zostało to przedstawione na Rysunku 5.6.

```
public record OrderListItemDto
{
    public int Id { get; init; }
    public string CustomerEmail { get; init; } = string.Empty;
    public string CustomerName { get; init; } = string.Empty;
    public DateTime CreatedAt { get; init; }
    public string Status { get; init; } = string.Empty;
    public string Importance { get; init; } = string.Empty;
    public double ULow { get; set; }
    public double UHigh { get; set; }
    public double UMedium { get; set; }
    public int ItemsCount { get; init; }
    public decimal TotalPrice { get; init; }
}
```

Rysunek 5.6: Zwracane DTO zamówienia

API zwraca zarówno oszacowany priorytet, jak i przynależności do poszczególnych klas. Służy to estetyce fasady aplikacji, dzięki temu użytkownik poza przybliżonym priorytetem widzi gradient (Rysunek 5.7), który symbolizuje rozkład klas przynależności. Jak widać na Rysunku 5.8, proporcje gradientu odpowiadają zwracanym wartościom uLow, uMedium oraz uHigh. Metoda tworząca gradient przedstawiona została na Rysunku 5.9.



Rysunek 5.7: Gradient ważności zamówienia


```
{
    "id": 23,
    "customerEmail": "maria.kamiński752@example.com",
    "customerName": "Maria Kamiński",
    "createdAt": "2025-11-02T21:10:20.1516096",
    "status": "Pending",
    "importance": "Medium",
    "uLow": 0.2213258504689531,
    "uHigh": 0,
    "uMedium": 0.778674149531047,
    "itemsCount": 1,
    "totalPrice": 4000
}
```


Rysunek 5.8: Szczegóły zamówienia

```
private const string ULowColor = "#388E3C";
private const string UMediumColor = "#F57C00";
private const string UHighColor = "#D32F2F";
public string GetFuzzyGradient(dynamic ctx)
{
    var entries = new List<(string color, double w)>
    {
        (ULowColor,    (double)ctx.ULow),
        (UMediumColor, (double)ctx.UMedium),
        (UHighColor,   (double)ctx.UHigh)
    };

    const double EPS = 1e-9;
    var nonZero = entries.Where(e => e.w > EPS).ToList();

    if (nonZero.Count == 0)
        return "background: transparent; color: inherit;";

    if (nonZero.Count == 1)
        return $"background: {nonZero[0].color}; color: white;";

    if (nonZero.Count > 2)
        nonZero = nonZero.OrderByDescending(e => e.w).Take(2).ToList();

    var (color1, w1) = nonZero[0];
    var (color2, w2) = nonZero[1];

    var sum = w1 + w2;
    if (sum <= EPS) sum = 1;
    w1 /= sum; w2 /= sum;

    var split = w1 * 100.0;

    const double BLEND_WIDTH = 30.0;
    var half = BLEND_WIDTH / 2.0;

    double Clamp(double v, double lo, double hi) => v < lo ? lo : (v > hi ?
hi : v);

    var startBlend = Clamp(split - half, 0, 100);
    var endBlend   = Clamp(split + half, 0, 100);

    var sb = new System.Text.StringBuilder();
    sb.Append("background: linear-gradient(90deg, ");
    sb.Append($"{color1} 0%, ");
    sb.Append($"{color1} {startBlend.ToString("F2",
CultureInfo.InvariantCulture)}%, ");
    sb.Append($"{color2} {endBlend.ToString("F2",
CultureInfo.InvariantCulture)}%, ");
    sb.Append($"{color2} 100%); color: white;");

    return sb.ToString();
}
```

Rysunek 5.9: Metoda tworząca gradient na podstawie przynależności do klas

5.4.4 Wybrane mechanizmy API

W celu zachowania przejrzystości architektury systemu oraz ułatwieniu dalszego rozwoju, projekt został oparty na mechanizmie wstrzykiwania zależności. Każda funkcjonalność biznesowa jest zorganizowana w postaci osobnych serwisów, które są rejestrowane w kontenerze DI (Rysunek 5.11) i wykorzystywane w kontrolerach (Rysunek 5.12). Elementy przedstawione poniżej pełnią spójną strukturę, w której kontrolery odpowiadają wyłącznie za obsługę żądań HTTP, natomiast faktyczna logika znajduje się w warstwie serwisowej (Rysunek 5.10).

```
public interface IAuthService
{
    Task<TokenResponse?> LoginAsync(LoginRequest req, CancellationToken ct
= default);
}
public sealed class AuthService(IUserService users, IJwtTokenGenerator
tokens) : IAuthService
{
    public async Task<TokenResponse?> LoginAsync(LoginRequest req,
CancellationToken ct = default)
    {
        if (string.IsNullOrWhiteSpace(req.Email) ||
string.IsNullOrWhiteSpace(req.Password))
            return null;

        var isValid = await users.VerifyPasswordAsync(req.Email,
req.Password, ct);
        if (!isValid) return null;

        var u = await users.GetByEmailAsync(req.Email, ct);
        if (u is null) return null;

        var role = await users.GetRoleAsync(u.Id, ct);

        return tokens.Generate(u.Id, u.Email, $"{u.FirstName}
{u.LastName}", role.Name);
    }
}
```

Rysunek 5.10: Implementacja serwisu autoryzacyjnego

```
builder.Services.AddScoped<IAuthService, AuthService>();
```

Rysunek 5.11: Rejestracja serwisu w kontenerze DI

```
[ApiController]
[Route("api/auth")]
public sealed class AuthController(IAuthService auth) : ControllerBase
{
    [HttpPost("login")]
    public async Task<ActionResult<TokenResponse>> Login([FromBody]
LoginRequest req, CancellationToken ct)
    {
        var token = await auth.LoginAsync(req, ct);
        if (token is null) return Unauthorized(new { message = "Invalid
credentials." });
        return Ok(token);
    }
}
```

Rysunek 5.12: Kontroler korzystający z wstrzykniętego serwisu

Rozdział 6

Weryfikacja i walidacja

W niniejszym rozdziale przedstawiono sposób testowania opracowanego systemu, organizację eksperymentów oraz wyniki weryfikacji testów niefunkcjonalnych. Celem testów było potwierdzenie, że aplikacja spełnia wymagania określone w rozdziale 3, oraz działa poprawnie w rzeczywistych scenariuszach użytkowania.

6.1 Zakres i metodyka testowania

Testy zostały przeprowadzone zgodnie z podejściem zbliżonym do modelu V – wymagania zdefiniowane w początkowych etapach projektu stały się podstawą do opracowania przypadków testowych. Testy podzielono na:

- Testy funkcjonalne – sprawdzające poprawność implementacji odpowiednich endpointów
- Testy niefunkcjonalne – obejmujące wydajność, bezpieczeństwo i skalowalność
- Testy integracyjne – walidujące poprawność współpracy warstw API, bazy danych i interfejsu użytkownika
- Testy negatywne – weryfikujące odporność systemu na błędne dane, brak uprawnień, czy niepoprawny token JWT

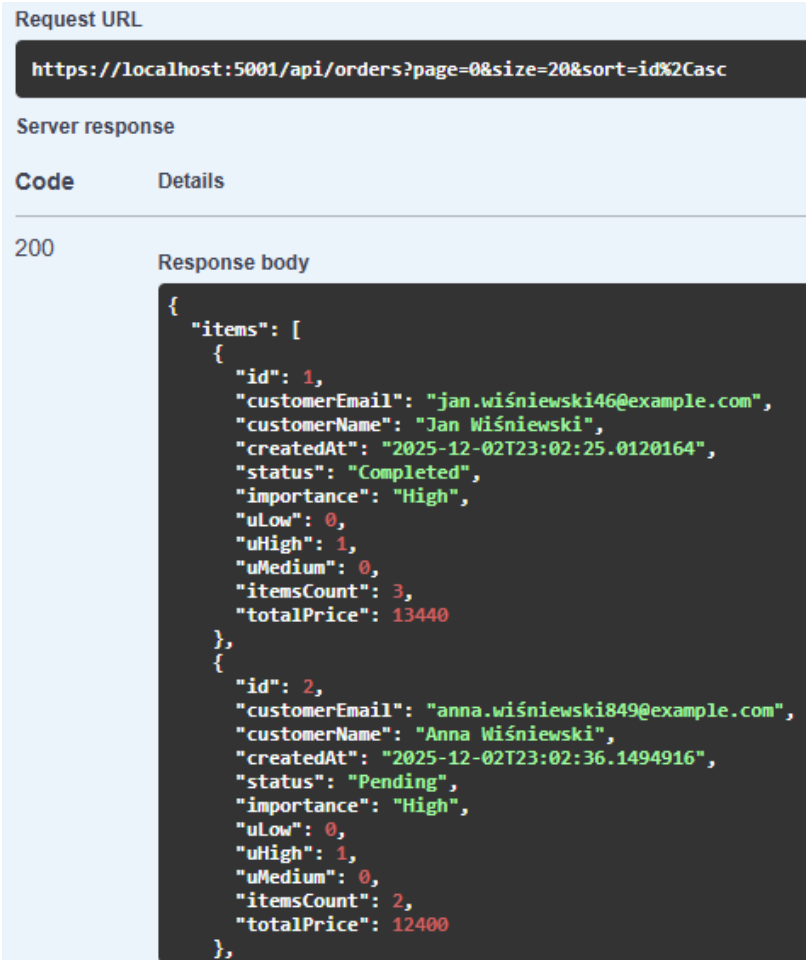
Środowisko testowe stanowiła lokalna instalacja systemu. Wykorzystywała ona SQL Server oraz publiczne endpointy API uruchomione w trybie produkcyjnym.

6.2 Testy funkcjonalne

Testy funkcjonalne objęły wszystkie kluczowe moduły systemu:

- Moduł logowania i ról użytkowników
- Moduł sprzedaży
- Moduł magazynowy
- Moduł administracyjny

W wymienionych modułach sprawdzono poprawność zwracanych danych, jak i zapytań tworzących nowe rekordy w bazie danych, takich jak rezerwowanie towarów, czy zarządzanie użytkownikami. Testy przeprowadzano za pomocą narzędzia Swagger, co zostało przedstawione na Rysunkach 6.1, 6.2 oraz 6.3. Zwracane kody *200* oraz *201* sugerują poprawność wywołania zapytania.



Rysunek 6.1: Poprawność endpointu pobierającego zamówienia

Request URL

```
https://localhost:5001/api/warehouse/orders/2/reserve
```

Server response

Code	Details
200	Response body


```
{
  "perItemReport": [
    {
      "productName": "Monitor Y",
      "branchName": "Oddział Gliwice",
      "reservedQuantity": 3,
      "missingQuantity": 0
    },
    {
      "productName": "Laptop X",
      "branchName": "Oddział Gliwice",
      "reservedQuantity": 4,
      "missingQuantity": 0
    }
  ],
  "isPartial": false
}
```


Rysunek 6.2: Poprawność endpointu rezerwującego produkty w magazynie

Request URL

```
https://localhost:5001/api/users
```

Server response

Code	Details
201 *Undocumented*	Response body


```
{
  "id": 1002,
  "firstName": "Testowy",
  "lastName": "Uzytkownik",
  "email": "test@example.com",
  "roleId": 1,
  "branchId": 1,
  "branchName": "Oddział Katowice",
  "roleName": "Administrator",
  "fullName": "Testowy Uzytkownik"
}
```


Rysunek 6.3: Poprawność endpointu tworzącego użytkownika

6.3 Testy niefunkcjonalne

W niniejszym podrozdziale przetestowano system pod kątem wymagań niefunkcjonalnych opisanych w podrozdziale *3.2 Wymagania niefunkcjonalne*. Testy objęły:

6.3.1 Wydajność

Wydajność mierzono za pomocą narzędzia k6, testując operacje pobierania zamówień wraz z filtracją, sortowaniem i paginacją. Dla bazy testowej obejmującej:

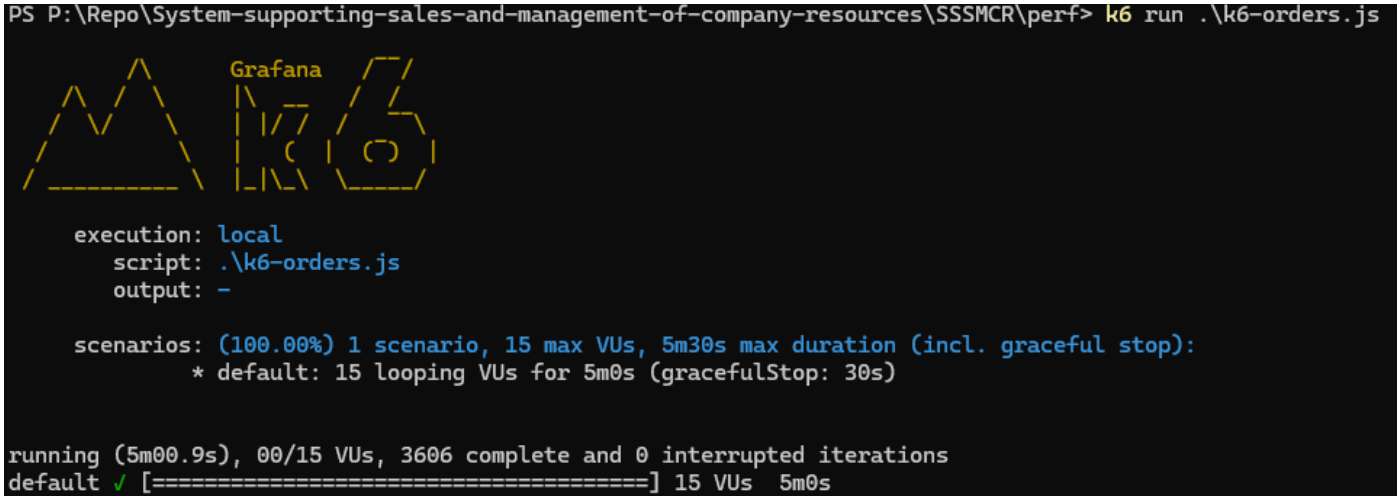
- ~10 000 zamówień
- ~100 000 pozycji zamówień,

Uzyskano wyniki przedstawione w Tabeli 6.1.

Tabela 6.1: Wyniki testów k6

Metryka	Wymaganie	Wynik
P95 czasu odpowiedzi	≤ 400 ms	~363 ms
Mediana czasu odpowiedzi	Nie narzucone	~125 ms

Wyniki potwierdzają spełnienie wymagań wydajnościowych. Wywołanie narzędzia i otrzymane wyniki zostały zaprezentowane na Rysunkach 6.4 oraz 6.5.



Rysunek 6.4: Uruchomienie narzędzia k6

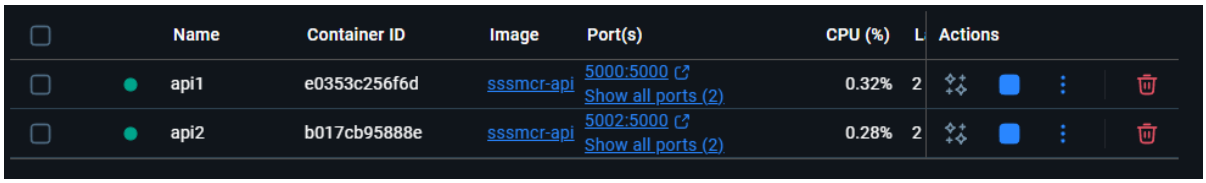

```
"http_req_duration": {
      "thresholds": {
        "p(95)<400": {
          "ok": true
        }
      },
      "type": "trend",
      "contains": "time",
      "values": {
        "med": 125.3526,
        "p(90)": 294.7924,
        "p(95)": 363.43981,
        "p(99)": 561.0108019999993,
        "max": 954.8184,
        "avg": 123.96928186858935,
        "min": 0
      }
    }
```


Rysunek 6.5: Metryki zwracane przez narzędzie k6

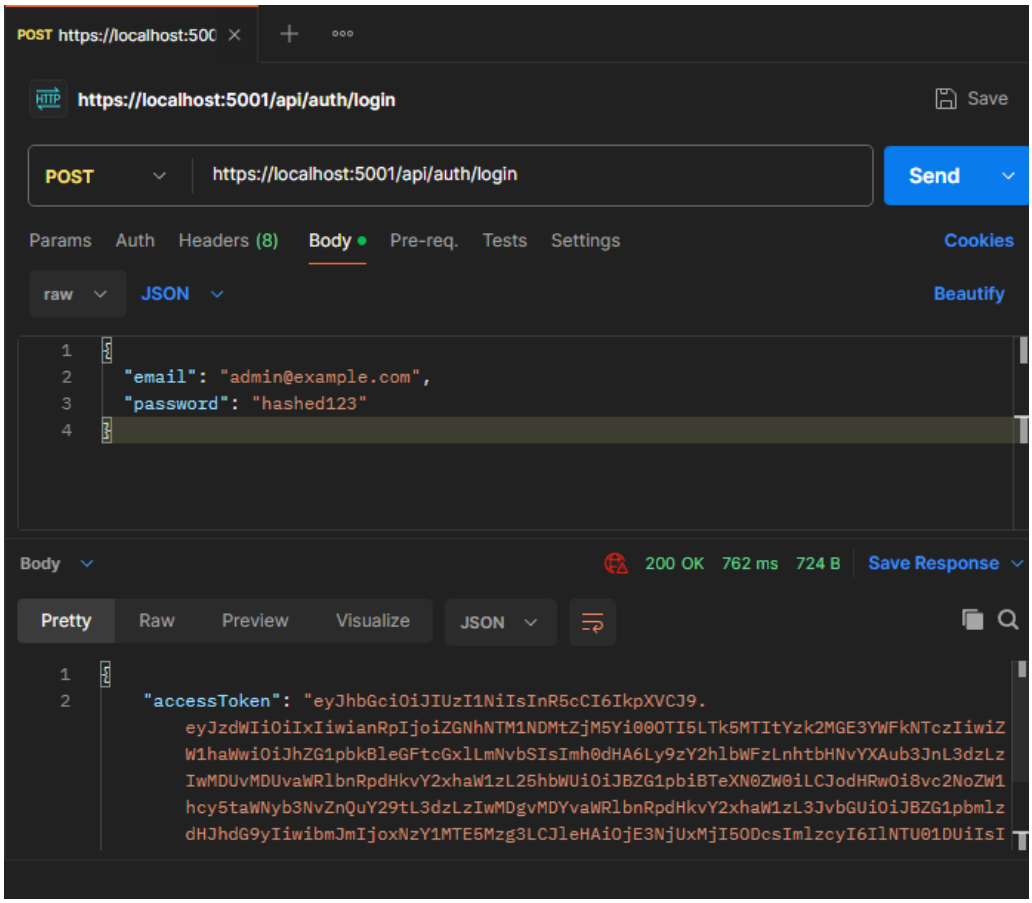
6.3.2 Skalowalność

Zweryfikowano pracę API w trybie stateless poprzez uruchomienie dwóch replik korzystając z narzędzia Docker [32] (Rysunek 6.6). W wyniku otrzymano:

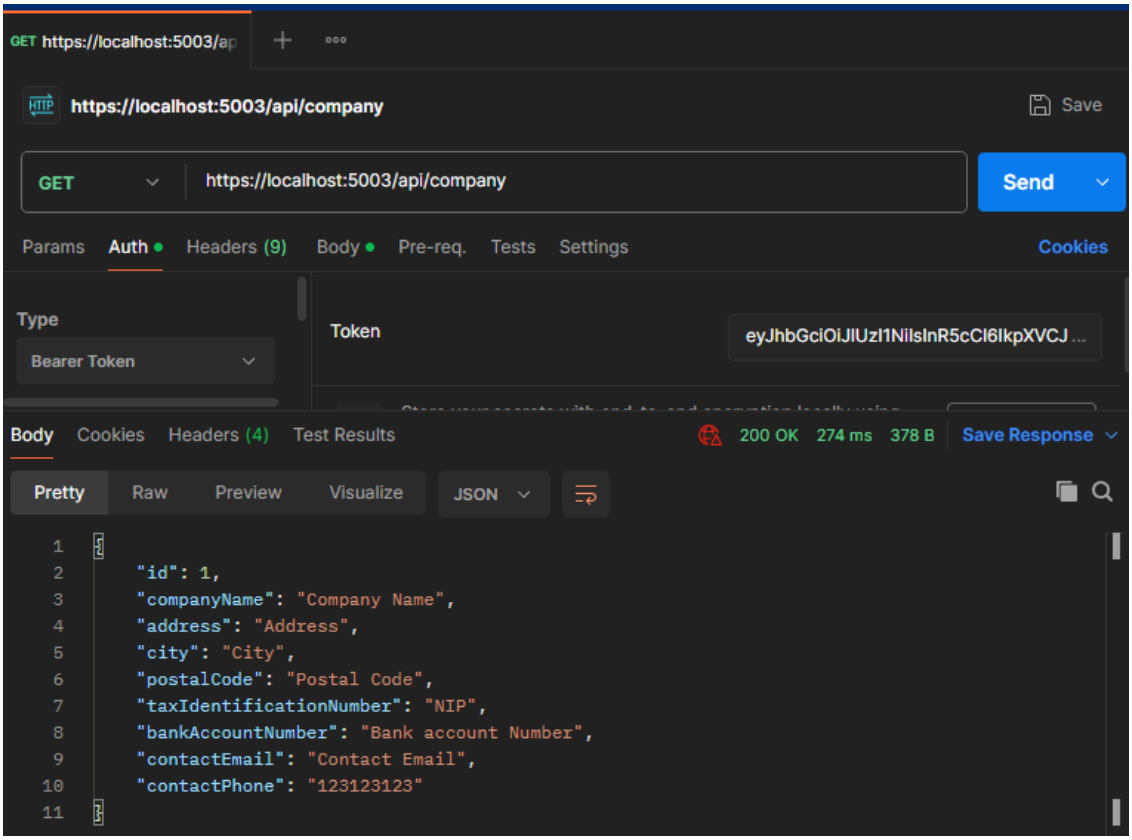
- Brak błędów związanych z utratą stanu,
- Poprawne działanie zabezpieczonych funkcjonalności po przekazaniu tokenu JWT (Rysunek 6.8),
- Pełną poprawność realizacji scenariusza logowania (Rysunek 6.7)



Rysunek 6.6: Uruchomienie dwóch replik API



Rysunek 6.7: Logowanie korzystając z repliki „api1”



Rysunek 6.8: Użycie tokenu JWT w replice „api2”

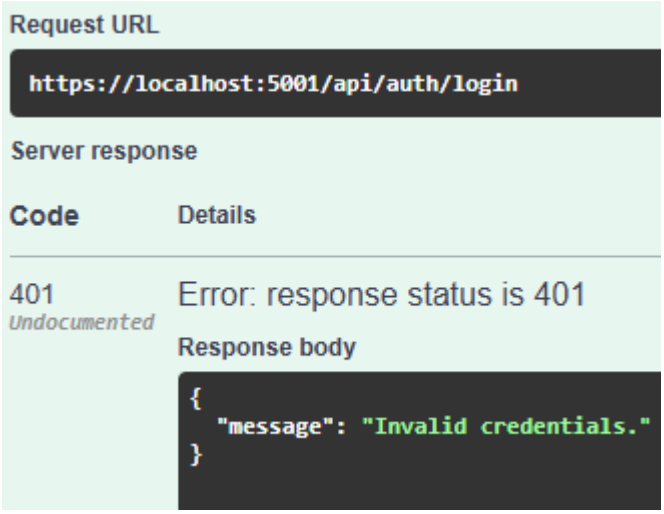
Wyniki testu potwierdzają, że system spełnia wymagania bezstanowości, a co za tym idzie – skalowalności poziomej.

6.3.3 Bezpieczeństwo

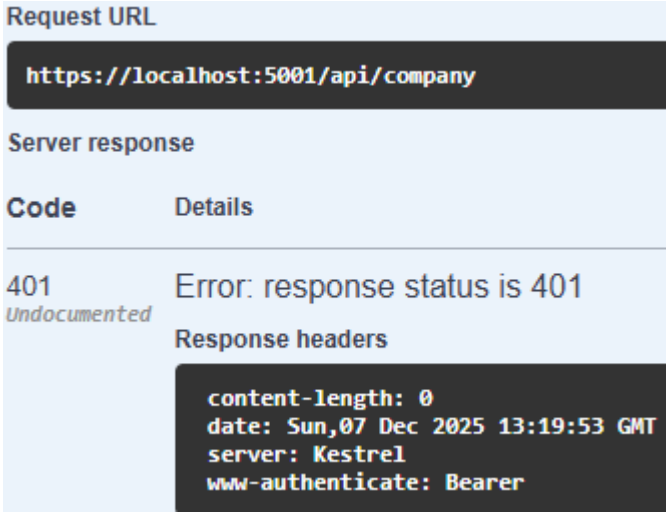
W celu przetestowania bezpieczeństwa systemu przeprowadzono testy obejmujące:

- Niepoprawne logowanie (spodziewane *401 Unauthorized*, Rysunek 6.9)
- Wywołanie endpointów bez tokena lub z tokenem o niewłaściwej roli (błąd *403*, Rysunek 6.10)
- Test przeciążeniowy zasobu autoryzacji

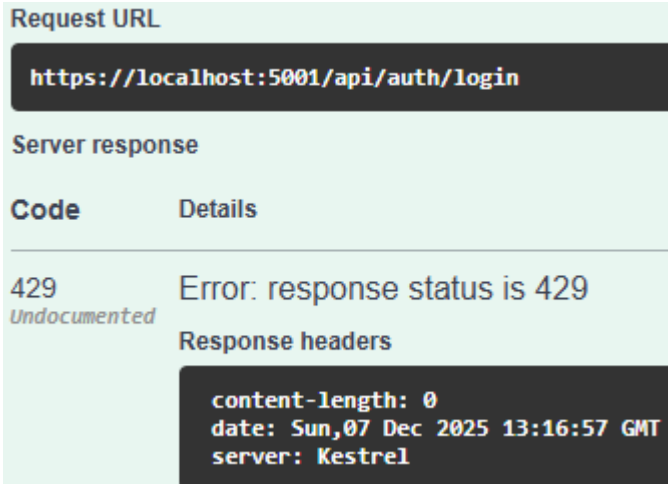
W teście obciążeniowym dla 10+ żądań na minutę do endpointu logowania zwrócono oczekiwany kod *429 Too Many Requests*, co potwierdza poprawną konfigurację rate-limitingu (Rysunek 6.11).



Rysunek 6.9: Niepoprawne logowanie



Rysunek 6.10: Błędny token JWT



Rysunek 6.11: Error „Too many requests”

6.4 Wykryte i usunięte błędy

Podczas testów wykryto i naprawiono między innymi:

- Niepoprawne obliczanie i wyświetlanie priorytetów zamówień po masowym imporcie danych (takim jak seedowanie dużej, testowej bazy) – rozwiązano poprzez wymuszenie aktualizacji cache
- Błąd w kodzie pobierania zamówień, przez który priorytet obliczany był każdorazowo dla wszystkich zamówień. Powodowało to czas dostępu do danych rzędu wielu sekund (dla symulowanej bazy). Po optymalizacji kodu, dzięki któremu priorytet jest obliczany tylko dla części pobieranych danych, czas dostępu zmalał do rzędu milisekund, co pozwoliło na poprawny wynik testu k6.
- Błąd walidacji tokenów JWT. W trakcie testów autoryzacji pojawił się błąd związany z walidacją JWT – system odrzucał poprawne tokeny lub zwracał *401*, mimo, że użytkownik był poprawnie zalogowany. Przyczyną okazał się zbyt krótki klucz symetryczny umieszczony w zmiennych środowiskowych. Problem rozwiązano wydłużając klucz powyżej 256 bitów.

Rozdział 7

Podsumowanie i wnioski

Głównym celem niniejszej pracy było zaprojektowanie i zaimplementowanie nowoczesnego systemu informatycznego, wspomagającego sprzedaż oraz zarządzanie zasobami w małych i średnich przedsiębiorstwach. Analiza tematu przeprowadzona w początkowej fazie projektu wykazała, że wiele firm boryka się brakiem integracji między działami magazynu i sprzedaży, a próg wejścia dla istniejących systemów ERP jest wysoki i kosztowny. Opracowane rozwiązanie stanowi odpowiedź na te wyzwania, integrując kluczowe procesy biznesowe w ramach jednej, spójnej aplikacji.

Zrealizowany projekt stanowi w pełni funkcjonalny system, który obejmuje podział na funkcjonalne moduły: magazynowy, sprzedażowy oraz administracyjny. Kluczowym osiągnięciem jest zastosowanie architektury opartej na platformie ASP.NET Core, z wykorzystaniem Web API w warstwie logiki biznesowej oraz frameworka Blazor Server w warstwie prezentacji. Takie podejście pozwoliło na zachowanie wysokiej interaktywności interfejsu użytkownika przy jednoczesnym odciążeniu urządzeń klienckich. Zastosowanie wzorca wstrzykiwania zależności oraz podziału na warstwy serwisowe i kontrolery zapewniło przejrzystość kodu oraz ułatwiło testowanie.

Istotnym elementem wyróżniającym opracowany system na tle konkurencyjnych rozwiązań jest algorytm priorytetyzacji zamówień oparty na logice rozmytej. Wykorzystanie trapezoidalnej funkcji przynależności pozwoliło na dynamiczną ocenę ważności zamówień w oparciu o wiele kryteriów jednocześnie. Dzięki temu rozwiązaniu, pracownicy otrzymują inteligentne wsparcie w procesie decyzyjnym, co bezpośrednio przekłada się na efektywność realizacji zamówień.

Ważnym etapem prac była weryfikacja spełnienia wymagań niefunkcjonalnych, w szczególności dotyczących wydajności i skalowalności. Przeprowadzone testy obciążeniowe wykazały, że system zachowuje wysoką responsywność nawet przy dużej liczbie rekordów. Ponadto potwierdzono działanie aplikacji w środowisku rozproszonym. Dzięki bezstanowej architekturze API oraz zastosowaniu tokenów JWT do autoryzacji, system poprawnie funkcjonuje przy uruchomieniu wielu replik serwera, co zweryfikowano w środowisku kontenerowym Docker.

Proces implementacji nie był pozbawiony wyzwań. W trakcie prac napotkano na problemy, które wymagały ponownej analizy i optymalizacji kodu. Jednym z nich była niska wydajność pobierania zamówień przy dużym wolumenie danych. Problem wynikał z każdorazowego przeliczania priorytetu zamówień dla całej bazy, nawet w przypadku pobierania jej części (paginacja). Stał się kluczowym błędem systemu, gdyż powodował brak spełnialności jednego z narzuconych wymagań niefunkcjonalnych. Problem ten rozwiązano poprzez optymalizację zapytań i ograniczenie obliczeń wyłącznie do aktualnie przetwarzanej partii danych.

Opracowany system stanowi solidną bazę do dalszego rozwoju. Modułowa budowa aplikacji pozwala na elastyczną rozbudowę o nowe funkcjonalności. Kolejnym naturalnym krokiem rozwoju byłaby integracja z zewnętrznymi systemami płatności oraz firmami kurierskimi, co pozwoliłoby na wdrożenie aplikacji do branży e-commerce. Możliwa jest również integracja z narzędziem *Power BI* w celu generowania bardziej rozbudowanych raportów sprzedaży. Ponadto, zgromadzone dane historyczne mogłyby posłużyć do trenowania modeli uczenia maszynowego, w celu prognozowania popytu na towary i automatycznego generowania zamówień do dostawców.

Podsumowując, praca inżynierska zakończyła się sukcesem, a postawione cele zostały osiągnięte. Powstało kompletne, zabezpieczone i wydajne narzędzie, które rozwiązuje realne problemy zarządzania zasobami w przedsiębiorstwie, wykorzystując przy tym nowoczesne technologie programistyczne jak i efektywne rozwiązania algorytmiczne.

Bibliografia

[1] Yen, H. J. R., Sheu, C., & Chae, B. (2004). Aligning ERP implementation with competitive priorities of manufacturing firms: An exploratory study. International Journal of Production Economics, 92(3), 207–220.

[2] Basoglu, N., Daim, T., & Kerimoglu, O. (2007). Organizational adoption of enterprise resource planning systems: A conceptual framework. Journal of Enterprise Information Management, 20(3), 337–359.

[3] Gessa, A., Jiménez, A., & Sancha, P. (2023). Exploring ERP systems adoption in challenging times. Insights of SMEs stories. Technological Forecasting & Social Change, 195, 122795.

[4] Paulsson, V., et al. (2023). Cloud ERP systems architectural challenges on digital transformation. Procedia Computer Science, 217, 253–260.

[5] Prakash, V., et al. (2022). Cloud- and Edge-based ERP systems for Industrial Internet of Things and Smart Factory. Procedia Computer Science, 200, 1338–1347.

[6] Antoniadis, I., et al. (2015). Adoption and usage of ERP systems by SMEs. Procedia – Social and Behavioral Sciences, 175, 299–304.

[7] Teittinen, H., Pellinen, J., & Järvenpää, M. (2013). ERP in action—Challenges and benefits for management control in SME context. Management Accounting Research, 24(4), 292–307.

[8] Kumar, D., et al. (2013). A fuzzy logic based decision support system for evaluation of suppliers. Applied Soft Computing, 13(1), 193–200.

[9] Nakandala, D., Lau, H., et al. (2013). A fuzzy-based decision support model for monitoring on-time delivery performance. European Journal of Operational Research, 225(3), 507–517.

[10] Lee, H.-Y., et al. (2019). Cloud-based enterprise resource planning with elastic deployment. Robotics and Computer-Integrated Manufacturing, 57, 12–24.

[11] Maican, C., & Lixandroiu, R. (2016). A system architecture based on open source enterprise applications. Information Systems, 57, 33–46.

[12] SAP SE. (2025). SAP ERP – Product Overview. Dostęp online: https://www.sap.com/products/erp.html

[13] Comarch S.A. (2025). Comarch ERP XL. Dostęp online: https://www.comarch.pl/erp/xl

[14] Microsoft (2025). Microsoft Dynamics 365. Dostęp online: https://www.microsoft.com/en-us/dynamics-365

[15] Insert S.A. (2025). Subiekt GT – opis programu. Dostęp online: https://www.insert.com.pl/programy_dla_firm/sprzedaz/subiekt_gt/opis.html

[16] Odoo S.A. (2025). Odoo ERP. Dostęp online: https://www.odoo.com

[17] Frappe Technologies (2025). ERPNext. Dostęp online: https://frappe.io/erpnext

[18] inoERP Association (2025). inoERP Documentation. Dostęp online: https://docs.inoerp.com/

[19] Microsoft (2025). *ASP.NET Core Documentation*. Microsoft Learn. Dostęp online: https://learn.microsoft.com/en-us/aspnet/core/

[20] Grafana Labs (2025). *k6 Documentation*. Dostęp online: https://k6.io/docs/

[21] Auth0 (2025). *JSON Web Tokens (JWT) Introduction*. Dostęp online: https://auth0.com/docs/secure/tokens/json-web-tokens

[22] Microsoft (2025). *REST API Design Guidelines*. Azure Architecture Center. Dostęp online:https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

[23] Microsoft (2025). *Entity Framework Core Documentation*. Dostęp online: https://learn.microsoft.com/en-us/ef/core/

[24] Microsoft (2025). *Microsoft SQL Server Documentation*. Dostęp online: https://learn.microsoft.com/en-us/sql/sql-server

[25] SmartBear Software (2025). *Swagger Documentation*. Dostęp online: https://swagger.io/

[26] Microsoft (2025). *ASP.NET Core Blazor hosting models*. Microsoft Learn. Dostęp online: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models

[27] MudBlazor (2025). *MudBlazor Component Library Documentation*. Dostęp online: https://mudblazor.com/

[28] JetBrains (2025). *Rider: The Cross-Platform .NET IDE*. Dostęp online: https://www.jetbrains.com/rider/

[29] Git (2025). *Git - Reference Documentation*. Dostęp online: https://git-scm.com/doc

[30] GitKraken (2025). *GitKraken Client Documentation*. Dostęp online: https://help.gitkraken.com/

[31] Microsoft (2025). *Dependency Injection in ASP.NET Core*. Microsoft Learn. Dostęp online: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection

[32] Docker Inc. (2025) *Docker Documentation*. Dostęp online: https://docs.docker.com/

Dodatki

Spis skrótów i symboli

ERP	ang. Enterprise Resource Planning – zintegrowane oprogramowanie służące do koordynacji kluczowych procesów przedsiębiorstwa
MŚP	firmy należące do sektora małych i średnich przedsiębiorstw
API	interfejs aplikacji pozwalający na wymianę danych pomiędzy aplikacjami lub usługami
SQL	ang. Structured Query Language – język zapytań wykorzystywany do tworzenia, modyfikowania i odczytu danych w relacyjnych bazach danych
JWT	ang. JSON Web Token – standard bezpiecznego przekazywania informacji w postacji podpisanego tokenu, używany do uwierzytelniania użytkowników i autoryzacji dostępu do zasobów
KPI	ang. Key Performance Indicators – kluczowe wskaźniki efektywności wykorzystywane do oceny wyników działalności firmy
RBAC	ang. Role-Based Access Control - kontrola dostępu oparta na rolach
IDE	ang. Integrated Development Environment - zintegrowane środowisko deweloperskie
ERD	ang. Entity-Relationship Diagram - graficzne przedstawienie struktury bazy danych
DTO	ang. Data Transfer Object – obiekt transferu danych służący do wymiany informacji pomiędzy warstwami aplikacji
REST	ang. Representational State Transfer – styl architektury komunikacyjnej oparty na protokole HTTP, pozwalający na tworzenie skalowalnych interfejsów API
SPA	ang. Single Page Application – aplikacja jednostronnicowa, w której interfes użytkownika jest dynamicznie aktualizowany po stronie klienta
ORM	ang. Object-Relational Mapping - mapowanie obiektowo-relacyjne, proces zamiany klas w systemie na obiekty bazodanowe
DI	ang. Dependency Injection – wstrzykiwane zależności
UI	ang. User Interface – interfejs użytkownika

Spis rysunków

Spis tablic