|  |  |
| --- | --- |
| Obraz zawierający szkic, rysowanie, clipart, Grafika liniowa  Zawartość wygenerowana przez AI może być niepoprawna. | **POLITECHNIKA ŚLĄSKA**  **Wydział Inżynierii Materiałowej** |
| **Tworzenie aplikacji internetowych i bazodanowych** | |
| **TEMAT:**  **Tokyo Garden Restaurant — Dokumentacja Projektu** | |
| **Uwagi prowadzącego:** | |
| **Data ostatniej modyfikacji:**  **16.09.2025**    **Podpis prowadzącego:** | **Imię i nazwisko:**  **1.** Jakub Majer  **2.** Wojciech Kin  **3.** Szymon Czapla  **4**. Dominik Deszczka |
| **Grupa dziekańska: TP 4.0** |

[0. Streszczenie wykonawcze 4](#_Toc208946864)

[Cel biznesowy 4](#_Toc208946865)

[Zakres systemu 4](#_Toc208946866)

[Korzyści i mierniki 4](#_Toc208946867)

[1. Specyfikacja 4](#_Toc208946868)

[Kontekst i interesariusze 4](#_Toc208946869)

[Przypadki użycia (skrót) 4](#_Toc208946870)

[Wymagania niefunkcjonalne 4](#_Toc208946871)

[2. Model danych i domeny 5](#_Toc208946872)

[CDM – encje i relacje 5](#_Toc208946873)

[PDM – tabele i klucze 5](#_Toc208946874)

[Konwencje i integralność 5](#_Toc208946875)

[Diagram klas EF (logika mapowania) 5](#_Toc208946876)

[3. Architektura i warstwy 6](#_Toc208946877)

[Przegląd 6](#_Toc208946878)

[Wzorce i decyzje projektowe 6](#_Toc208946879)

[IDAL – interfejsy 6](#_Toc208946880)

[DAL – EF Core 6](#_Toc208946881)

[IBL/BL – logika biznesowa 6](#_Toc208946882)

[API – Web 6](#_Toc208946883)

[UI – Angular 6](#_Toc208946884)

[4. Testy (podejście i zakres) 7](#_Toc208946885)

[Zasady ogólne 7](#_Toc208946886)

[Doubles własne i Moq 7](#_Toc208946887)

[Testy DAL z InMemory 7](#_Toc208946888)

[Testy kontrolerów 7](#_Toc208946889)

[5. Interfejs Web API (kontrakt i zasady) 7](#_Toc208946890)

[Konwencje REST 7](#_Toc208946891)

[Przykładowe zasoby i operacje 7](#_Toc208946892)

[Walidacja i błędy 8](#_Toc208946893)

[Paginacja/sortowanie/filtrowanie 8](#_Toc208946894)

[6. Frontend (Angular) 8](#_Toc208946895)

[Struktura i wzorce 8](#_Toc208946896)

[UX i walidacje 8](#_Toc208946897)

[Mapowanie DTO 8](#_Toc208946898)

[7. Bezpieczeństwo i zgodność 8](#_Toc208946899)

[Autoryzacja i role 8](#_Toc208946900)

[CORS i ochrona danych 8](#_Toc208946901)

# 0. Streszczenie wykonawcze

## Cel biznesowy

Celem systemu Tokyo Garden Restaurant jest wsparcie pełnego cyklu życia zarządzania ofertą i zamówieniami w restauracji: od definicji kategorii i pozycji menu (ze składnikami i alergenami), przez zbieranie zamówień klientów, po administrację danymi użytkowników i adresów dostaw. System ma ułatwić codzienną pracę obsłudze i zapewnić spójność danych.

## Zakres systemu

System obejmuje backend (ASP.NET Web API + warstwa BL + DAL/EF Core) oraz frontend (Angular). Backend udostępnia RESTowe endpointy do CRUD-ów i operacji domenowych. Frontend zapewnia panel administracyjny oraz widoki list/szczegółów.

## Korzyści i mierniki

• Redukcja błędów w danych (wymuszone relacje i walidacje po stronie BL). • Szybsza obsługa katalogu menu (kategorie, alergeny, zdjęcia). • Skalowalność: separacja warstw i wzorzec repozytorium. • Mierniki: czas wprowadzania zmiany w pozycji menu, liczba błędnych zamówień wykrytych walidacją, czasy odpowiedzi API.

# 1. Specyfikacja

## Kontekst i interesariusze

Interesariusze: Klient (przegląda menu i składa zamówienie), Administrator/Pracownik (utrzymuje dane), Właściciel (raporty i nadzór). System płatności może zostać zintegrowany w przyszłości przez bramkę płatniczą.

## Przypadki użycia (skrót)

• Klient: przeglądanie menu, filtrowanie po kategoriach i alergenach, składanie zamówienia.

• Administrator: dodawanie/edycja/usuwanie pozycji menu, przypinanie kategorii i alergenów, podgląd i aktualizacja statusów zamówień.

• System: walidacja danych i spójność relacji; obliczanie wartości zamówień (suma pozycji × ilości).

## Wymagania niefunkcjonalne

• Wydajność: endpointy listowe z paginacją/sortowaniem/filtrowaniem. • Bezpieczeństwo: autoryzacja ról (Admin/User) — do rozbudowy; walidacje wejścia; zasady CORS. • Utrzymywalność: warstwowa architektura, DI, jasno zdefiniowane kontrakty. • Jakość: testy jednostkowe dla BL/DAL i kontrolerów.

# 2. Model danych i domeny

## CDM – encje i relacje

Encje główne: Kategorie, Alergeny, Pozycje\_Menu, Zamówienia, Pozycje\_Zamówienia, Użytkownicy, Adresy.

Relacje: Kategorie 1..N Pozycje\_Menu; Alergeny N..M Pozycje\_Menu; Zamówienia 1..N Pozycje\_Zamówienia; Użytkownicy 1..N Zamówienia i 1..N Adresy.

## PDM – tabele i klucze

Przykładowe tabele i klucze:

• Pozycje\_Menu(id PK, nazwa, opis, składniki, cena, image\_data, kategoria\_id FK)

• Kategorie(id PK, nazwa)

• Alergeny(id PK, nazwa, opis)

• Pozycje\_Menu\_Alergeny(pozycja\_id FK, alergen\_id FK, PK(pozycja\_id, alergen\_id))

• Zamowienia(id PK, data, status, łączna\_cena, metoda\_płatności, opcje\_zamówienia, uzytkownik\_id FK, dodatkowe\_info)

• Pozycje\_Zamowienia(id PK, ilosc, cena, pozycja\_menu\_id FK, zamowienie\_id FK)

• Uzytkownicy(id PK, nazwa, haslo, telefon, typ)

• Adresy(id PK, miasto, ulica, nr\_domu, nr\_mieszkania, uzytkownik\_id FK)

## Konwencje i integralność

• PK: Id jako tożsamość. • FK z kaskadą ograniczoną wg potrzeb (np. nie usuwać zamówień przy usuwaniu użytkownika). • Indeksy: po nazwa\_pozycji, data\_zamowienia; indeksy dla kluczy FK. • Walidacje: nazwy i ceny wymagane, ceny > 0, ilości > 0. • N:M dla Alergeny–Pozycje realizowane przez tabelę łączącą.

## Diagram klas EF (logika mapowania)

Klasy POCO odpowiadają tabelom; konfiguracje relacji w Fluent API/atrybutach. Kolekcje nawigacyjne dla relacji 1..N i N..M. Mapowanie pól binarnych (image\_data) oraz ograniczeń długości tekstów.

# 3. Architektura i warstwy

## Przegląd

Architektura warstwowa: Model (POCO), IDAL (interfejsy repozytoriów + UnitOfWork), DAL (EF Core), IBL (interfejsy usług), BL (implementacje), API (ASP.NET Web API), UI (Angular). Zasada: BL zna IDAL, API zna IBL; brak zależności wstecznych.

## Wzorce i decyzje projektowe

• Repository + Unit of Work: ujednolicony kontrakt CRUD i transakcje.

• DTO vs Model: API pracuje na DTO, BL operuje na Modelu domenowym (mapowanie).

• DI/IoC: kontrolery wstrzykują IBL; BL wstrzykuje IDAL.

• Walidacja: logika w BL, wstępna walidacja atrybutami w DTO.

## IDAL – interfejsy

• IRepository<T>: Add, Update, Remove, GetById, Query (z IQuerable/IAsyncEnumerable), paginacja.

• Specjalizacje: PozycjeMenu, Kategorie, Alergeny, Zamowienia, PozycjeZamowienia, Uzytkownicy, Adresy.

• IUnitOfWork: SaveChanges/Commit, dostęp do repozytoriów.

## DAL – EF Core

• DbContext (DbTokyoGarden) eksponuje DbSety encji. • Repozytorium generyczne z AsNoTracking w zapytaniach odczytowych. • Transakcje przez UnitOfWork; kontrola śledzenia i ładowania powiązań.

## IBL/BL – logika biznesowa

• Serwisy BL egzekwują reguły: poprawność cen/ilości, obecność kategorii/alergenów, przeliczanie łącznej wartości zamówienia. • Obsługa błędów: wyjątki domenowe → mapowane na odpowiedzi HTTP 400/404/409 w API.

## API – Web

• Kontrolery CRUD dla encji z REST konwencjami, kody 200/201/204/400/404/409. • Query endpoints: filtrowanie po kategorii/alergenach, paginacja. • ProblemDetails dla błędów.

## UI – Angular

• Moduł Admin Panel z widokami list i formularzy edycji dla kategorii, alergenów, pozycji menu. • Serwisy HTTP, interceptory błędów, formularze reaktywne, walidacje.

# 4. Testy (podejście i zakres)

## Zasady ogólne

Testy w xUnit, nomenklatura AAA (Arrange–Act–Assert), izolacja przypadków testowych. Focus na regułach biznesowych i spójności danych.

## Doubles własne i Moq

• Dummy: parametry obowiązkowe bez użycia.

• Stub: deterministyczny zwrot danych (np. repozytorium zwracające stałą listę pozycji).

• Fake: prosta implementacja (np. in-memory kolekcje) do szybkich testów BL.

• Mock: weryfikacja interakcji (czy BL wywołało Save/Update na repozytorium).

• Spy: rejestracja liczby wywołań/metod.

W Moq konfigurujemy Setup/Returns i Verify, aby odwzorować te role.

## Testy DAL z InMemory

Każdy test tworzy unikalny DbContext z nazwą bazy, aby nie współdzielić stanu. Scenariusze: CRUD, kaskady, relacje N:M, ograniczenia unikalności i walidacje na poziomie EF.

## Testy kontrolerów

Testy jednostkowe z zamockowaną BL (sprawdzenie statusów/kontraktów) oraz testy integracyjne (minimalny host), które weryfikują poprawność routingu, serializacji i walidacji ModelState.

# 5. Interfejs Web API (kontrakt i zasady)

## Konwencje REST

Rzeczowniki w liczbie mnogiej, wersjonowanie ścieżką (np. /api/v1/pozycje-menu), kody statusu zgodne z rezultatem operacji, Location header po POST 201.

## Przykładowe zasoby i operacje

• /api/v1/kategorie — GET (lista z paginacją), POST, GET/{id}, PUT/{id}, DELETE/{id}

• /api/v1/alergeny — jak wyżej

• /api/v1/pozycje-menu — GET z filtrowaniem po kategorii/alergenach; CRUD

• /api/v1/zamowienia — GET (status), POST (utworzenie wraz z pozycjami), PUT/PATCH (aktualizacja statusu)

• /api/v1/uzytkownicy — CRUD; /api/v1/uzytkownicy/{id}/adresy — CRUD

## Walidacja i błędy

Walidacja DTO na wejściu; błędy jako ProblemDetails (tytuł, status, szczegóły). Konflikty (409) przy naruszeniu integralności; 404 gdy byt nie istnieje; 400 dla nieprawidłowych danych.

## Paginacja/sortowanie/filtrowanie

Parametry zapytań np. page, pageSize, sortBy, sortDir, filters[]. BL/DAL mapuje na zapytania LINQ/EF Core.

# 6. Frontend (Angular)

## Struktura i wzorce

Struktura modułowa (np. admin, shared). Serwisy API kapsułkują wywołania HTTP; komponenty korzystają z formularzy reaktywnych (walidacje min/max, wymagane pola).

## UX i walidacje

Pola wymagane oznaczone, formaty liczbowe dla cen, komunikaty o błędach przy focus out i submit. Widoki list obsługują paginację, filtrowanie po kategorii/alergenach; edycja w formularzach dialogowych.

## Mapowanie DTO

Formularze → DTO request → API; odpowiedzi API → modele widoków. Idempotencja aktualizacji przez PUT, częściowe zmiany przez PATCH tam gdzie wskazane.

# 7. Bezpieczeństwo i zgodność

## Autoryzacja i role

Wariant MVP: role Admin/User; kontrolery zabezpieczone atrybutami autoryzacji. Docelowo JWT Bearer z krótkim TTL i odświeżaniem.

## CORS i ochrona danych

CORS ograniczony do domen frontendu. Sekrety w zmiennych środowiskowych. Walidacja rozmiarów uploadów (np. image\_data). Sanitizacja danych tekstowych.