WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: PROGRAMOWANIE

Viktor Pylypenko
Nr albumu studenta W68166

Aplikacja z interfejsem GUI do zarządzania sklepem z częściami samochodowymi

Prowadzący: mgr inż. Ewa Żesławska

Praca projektowa programowanie obiekotwe C#

Rzeszów 2025

Spis treści

Wstęp

Projekt „CarParts” ma na celu stworzenie aplikacji służącej do zarządzania częściami samochodowymi, ich kategoriami oraz dostawcami. Głównym celem jest zapewnienie użytkownikom intuicyjnego i prostego w obsłudze interfejsu umożliwiającego zarządzanie informacjami o częściach samochodowych w sposób efektywny i uporządkowany.

Aplikacja wykorzystuje technologie takie jak C# i ASP.NET Core, co zapewnia stabilność oraz możliwość łatwego skalowania projektu. System pozwala na wykonywanie operacji CRUD na głównych encjach, takich jak Kategorie, Produkty i Dostawcy, a także zapewnia przejrzysty interfejs użytkownika oparty na formularzach graficznych.

Założenia projektu

1. Projekt o nazwie **CarParts** to aplikacja do zarządzania częściami samochodowymi, kategoriami oraz dostawcami, zbudowana w technologii **C#** i **ASP.NET Core**.
2. Aplikacja umożliwia realizację operacji CRUD (Create, Read, Update, Delete) na encjach: **Category**, **Product**, i **Supplier**.
3. Interfejs użytkownika jest zbudowany z wykorzystaniem formularzy graficznych **GUI Forms**, co zapewnia intuicyjne zarządzanie danymi przez użytkownika.
4. Struktura projektu jest warstwowa, co obejmuje warstwy: **Controllers**, **Services**, **Models**, **Helpers**, i **Middleware**.
5. Dane aplikacji są przechowywane w plikach JSON, zarządzanych przy użyciu klasy **FileStorageHelper**.
6. Globalne przetwarzanie wyjątków zostało zaimplementowane w klasie **ExceptionHandlingMiddleware** dla stabilności systemu.

Cele projektu

1. Stworzenie skalowalnej i łatwej w utrzymaniu aplikacji do zarządzania częściami samochodowymi.
2. Zapewnienie użytkownikom możliwości zarządzania kategoriami, produktami oraz dostawcami w sposób intuicyjny i wydajny.
3. Ułatwienie zarządzania danymi za pomocą graficznego interfejsu użytkownika z obsługą operacji CRUD.
4. Wdrożenie dobrej praktyki projektowania, opartej na podejściu warstwowym, aby oddzielić logikę biznesową od prezentacji i danych.
5. Zastosowanie narzędzi takich jak **Swagger** dla automatycznej dokumentacji API.

Rozdział 1

Wymagania funkcjonalne i niefunkcjonalne

1.1 Wymagania funkcjonalne

- System umożliwia operacje CRUD na encjach:
  - Kategorie (Category) – zarządzanie nazwą i opisem.
  - Produkty (Product) – zarządzanie nazwą, ceną, kodem oraz powiązaniem z kategorią i dostawcą.
  - Dostawcy (Supplier) – zarządzanie nazwą, informacjami kontaktowymi oraz adresem.
- Intuicyjny interfejs użytkownika pozwala na przeglądanie, dodawanie, edytowanie oraz usuwanie danych.
- Walidacja danych wejściowych, np. ograniczenie zakresu cen czy maksymalna długość pól tekstowych.
- Obsługa API, umożliwiająca komunikację z backendem za pomocą klasy HttpClient.
- Przechowywanie danych w plikach JSON.

1.2 Wymagania niefunkcjonalne:

- **Wydajność:** Operacje CRUD muszą działać z minimalnym opóźnieniem.
- **Skalowalność:** Struktura projektu umożliwia dodawanie nowych funkcjonalności.
- **Stabilność:** Obsługa wyjątków chroni system przed błędami krytycznymi.
- **Przenośność:** Kompatybilność z każdym systemem obsługującym środowisko .NET Core.
- **Bezpieczeństwo:** Walidacja danych chroni przed błędami i nieprawidłowymi danymi.
- **Dostępność:** Intuicyjny GUI ułatwia obsługę aplikacji przez użytkowników bez zaawansowanej wiedzy technicznej.

Rozdział 2

Opis struktury projektu

2.1 Struktura i Opis Techniczny

System został zaprojektowany w oparciu o architekturę warstwową, składającą się z następujących komponentów:

- Warstwa Kontrolerów: Obsługuje zapytania HTTP i deleguje logikę do odpowiednich usług.
- Warstwa Usług: Implementuje logikę biznesową, operując na danych i koordynując interakcje z bazą danych.
- Warstwa Modeli: Definiuje struktury danych używane w systemie.
- Baza Danych: Przechowuje dane systemowe w formacie JSON, korzystając z plików jako persystencji.

2.2 Język, Narzędzia i Wymagania Sprzętowe

2.2.1 Język Programowania

System został zaimplementowany w języku **C#** z wykorzystaniem frameworka **ASP.NET Core**.

2.2.2 Narzędzia

- IDE: Microsoft Visual Studio
- Biblioteki: Newtonsoft.Json (do operacji na danych JSON)
- Serwer: Kestrel wbudowany w ASP.NET Core
- Testowanie: Swagger UI dla testowania API

2.2.3 Minimalne Wymagania Sprzętowe

- Procesor: Dual-core 2 GHz lub lepszy
- RAM: 4 GB
- Dysk: 100 MB wolnego miejsca
- System Operacyjny: Windows 10 lub nowszy

2.3 Zarządzanie Danymi i Baza Danych

Dane w systemie przechowywane są w formacie JSON w plikach:

- `categories.txt` - kategorie produktów
- `products.txt` - produkty
- `suppliers.txt` - dostawcy

Operacje na danych realizowane są za pomocą klasy `FileStorageHelper`, która umożliwia odczyt i zapis danych JSON.

2.4 Hierarchia Klas i Kluczowe Metody

Hierarchia klas została zaprojektowana w sposób modularny i rozszerzalny. Kluczowe klasy to:

2.4.1 Klasa `Category`

Definicja kategorii produktów.

- **Właściwości:**
  - `Id` - identyfikator kategorii
  - `Name` - nazwa kategorii
  - `Description` - opis kategorii

2.4.2 Klasa `CategoryService`

Implementacja logiki biznesowej dla kategorii.

- `GetAllCategories()` - zwraca wszystkie kategorie.
- `GetCategoryById(int id)` - zwraca kategorię na podstawie `Id`.
- `AddCategory(Category category)` - dodaje nową kategorię.
- `UpdateCategory(int id, Category category)` - aktualizuje istniejącą kategorię.
- `DeleteCategory(int id)` - usuwa kategorię.

2.4.3 Klasa `Product`

Definicja produktu.

- **Właściwości:**
  - `Id` - identyfikator produktu
  - `Name` - nazwa produktu
  - `Price` - cena produktu
  - `Code` - kod produktu

2.4.4 Klasa `ProductService`

Logika biznesowa dla produktów.

- `GetAllProducts()` - zwraca wszystkie produkty.
- `AddProduct(Product product)` - dodaje nowy produkt.
- `UpdateProduct(int id, Product product)` - aktualizuje istniejący produkt.

2.4.5 Klasa `Supplier`

Definicja dostawcy.

- **Właściwości:**
  - `Id` - identyfikator dostawcy
  - `Name` - nazwa dostawcy
  - `ContactInfo` - informacje kontaktowe
  - `Address` - adres

2.4.6 Klasa `SupplierService`

Logika biznesowa dla dostawców.

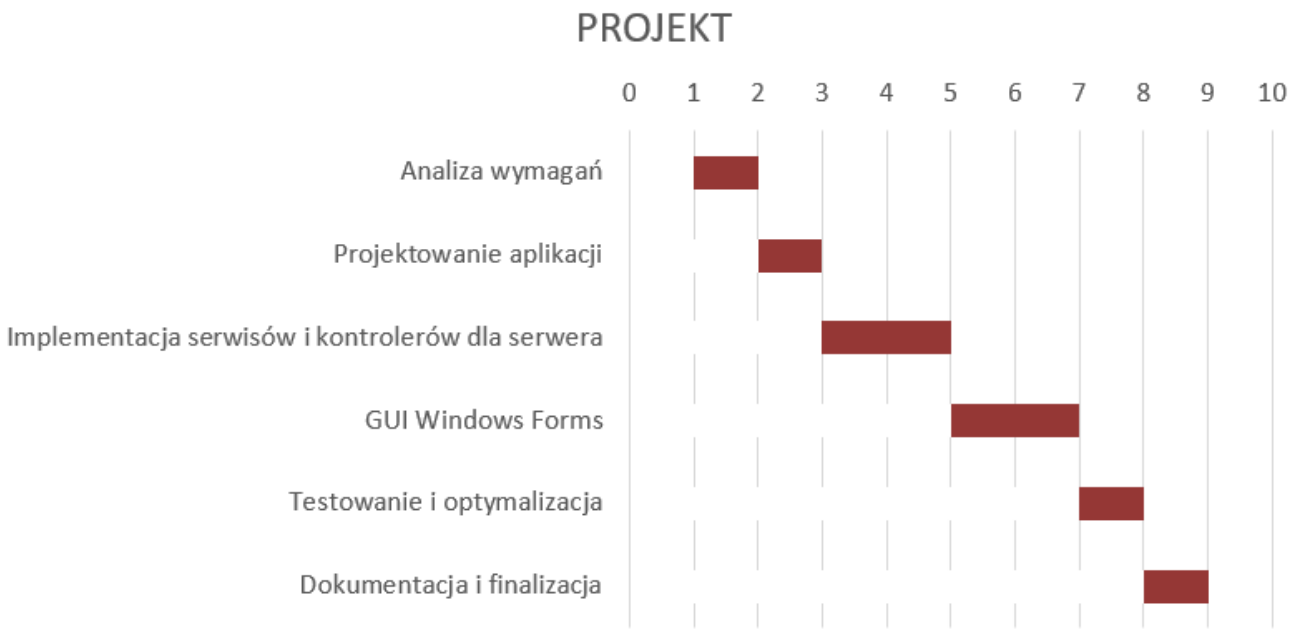
- `GetAllSuppliers()` - zwraca wszystkich dostawców.
- `AddSupplier(Supplier supplier)` - dodaje nowego dostawcę.
- `UpdateSupplier(int id, Supplier supplier)` - aktualizuje dane dostawcy.

Rozdział 3

Harmonogram realizacji projektu

3.1 Harmonogram realizacji

W celu zaplanowania i monitorowania postępów w realizacji projektu został stworzony diagram Gantta, który przedstawia podział projektu na poszczególne etapy i terminy ich realizacji. Diagram ten jest przedstawiony na rysunku 3.1.



Rysunek 3.1: Diagram Gantta przedstawiający harmonogram realizacji projektu.

Diagram obejmuje następujące etapy:

1. **Analiza wymagań (tygodnie 1-2):** Na tym etapie zebrano wymagania i przygotowano ogólną koncepcję projektu.

2. **Projektowanie aplikacji (tygodnie 2-3):** Opracowano architekturę systemu, w tym schematy baz danych i strukturę kodu.

3. **Implementacja serwisów i kontrolerów dla serwera (tygodnie 3-5):** Zaimplementowano backend aplikacji z wykorzystaniem API.

4. **GUI Windows Forms (tygodnie 5-7):** Stworzono graficzny interfejs użytkownika, umożliwiający interakcję z systemem.

5. **Testowanie i optymalizacja (tygodnie 7-8):** Przeprowadzono testy jednostkowe i integracyjne, wprowadzono poprawki oraz zoptymalizowano kod.

6. **Dokumentacja i finalizacja (tygodnie 8-9):** Przygotowano dokumentację użytkownika i kodu oraz sfinalizowano projekt.

3.2 Repozytorium i system kontroli wersji

Kod źródłowy projektu został umieszczony w publicznie dostępnym repozytorium na platformie GitHub. Repozytorium dostępne jest pod adresem w załączniku.

W projekcie wykorzystano system kontroli wersji **Git**, który umożliwia:

- Śledzenie historii zmian w kodzie.
- Współpracę zespołową poprzez rozgałęzienia (*branches*) i scalanie (*merge*).
- Szybkie cofanie wprowadzonych zmian w przypadku wykrycia błędów.

Struktura repozytorium: Struktura repozytorium odpowiada strukturze projektu opisanej w poprzednich rozdziałach

Rozdział 4

Prezentacja warstwy użytkowej projektu

4.1 Zrzuty ekranu aplikacji i ich opisy

Poniżej znajduje się zestaw zrzutów ekranu przedstawiających różne funkcje i operacje w Panelu Kontrolnym Dostawców. W panelu wyróżniono następujące sekcje:

- Pierwszy textbox i zielony przycisk: **Załaduj listę dostawców**.
- Drugi textbox i niebieski przycisk: **Dodaj nowego dostawcę**.
- Trzeci textbox i niebieski przycisk: **Aktualizuj dane istniejącego dostawcy**.
- Czwarty textbox i czerwony przycisk: **Usuń dostawcę według ID**.

4.1.1 Funkcjonalności panelu

W aplikacji główny widok umożliwia wykonywanie podstawowych operacji CRUD (Create, Read, Update, Delete) na danych dostawców. Na przykład, na rysunku 4.1 widzimy ekran startowy panelu, gdzie możemy załadować listę dostawców przy użyciu zielonego przycisku. Po wczytaniu danych, jak na rysunku 4.2, użytkownik może przystąpić do dodawania, modyfikacji lub usuwania danych.

Na rysunku 4.3 zaprezentowano przykład dodawania nowego dostawcy, gdzie w drugim textboxie należy wprowadzić szczegóły, a następnie kliknąć **Dodaj**. System wyświetla potwierdzenie sukcesu operacji, jak pokazano na rysunku 4.4.

Z kolei na rysunku 4.6 pokazano możliwość aktualizacji danych istniejącego dostawcy, za pomocą trzeciego textboxa i przycisku **Aktualizuj**. Po dokonaniu zmian lista dostawców zostaje odświeżona, co widać na rysunku 4.7. Dodatkowo użytkownik może usuwać dostawców, podając ich ID w czwartym textboxie i klikając czerwony przycisk, jak pokazano na rysunku 4.8.

W przypadku błędów, takich jak nieprawidłowe dane wejściowe, aplikacja wyświetla odpowiednie komunikaty, co przedstawiono na rysunku 4.10.

4.1.2 Rozszerzenia aplikacji

Poza obsługą dostawców aplikacja zawiera panele umożliwiające zarządzanie produktami i kategoriami. Jak widać na rysunku 4.11, panel produktów działa na podobnych zasadach, umożliwiając dodawanie, modyfikację i usuwanie danych produktów. Panel kategorii, przedstawiony na rysunku 4.12, pozwala na podobne operacje w kontekście zarządzania kategoriami.

4.1.3 Dokumentacja API

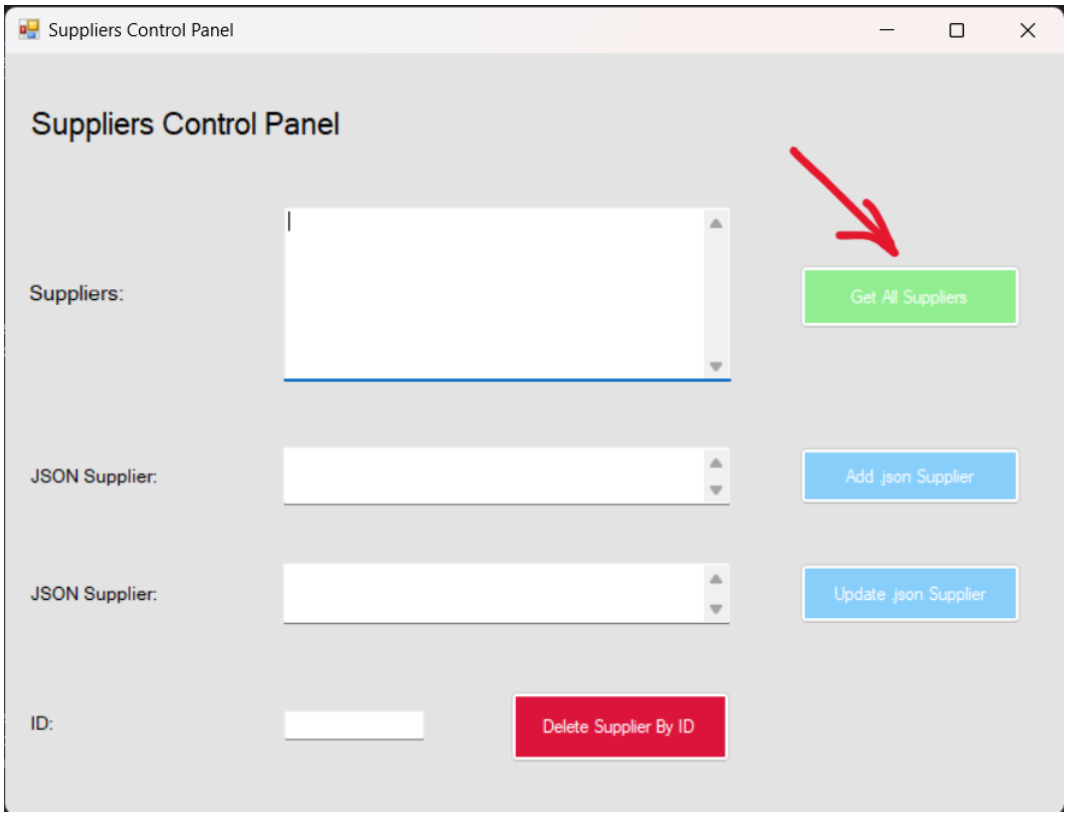
Rysunek 4.13 przedstawia dokumentację API wygenerowaną za pomocą narzędzia Swagger. Dzięki tej dokumentacji możliwe jest łatwe zrozumienie struktury zapytań i odpowiedzi w aplikacji.

API zostało zaprojektowane w sposób uniwersalny, umożliwiając interakcję z backendem nie tylko za pomocą aplikacji napisanej w Windows Forms, ale również przy użyciu dowolnej platformy czy frameworka, takich jak:

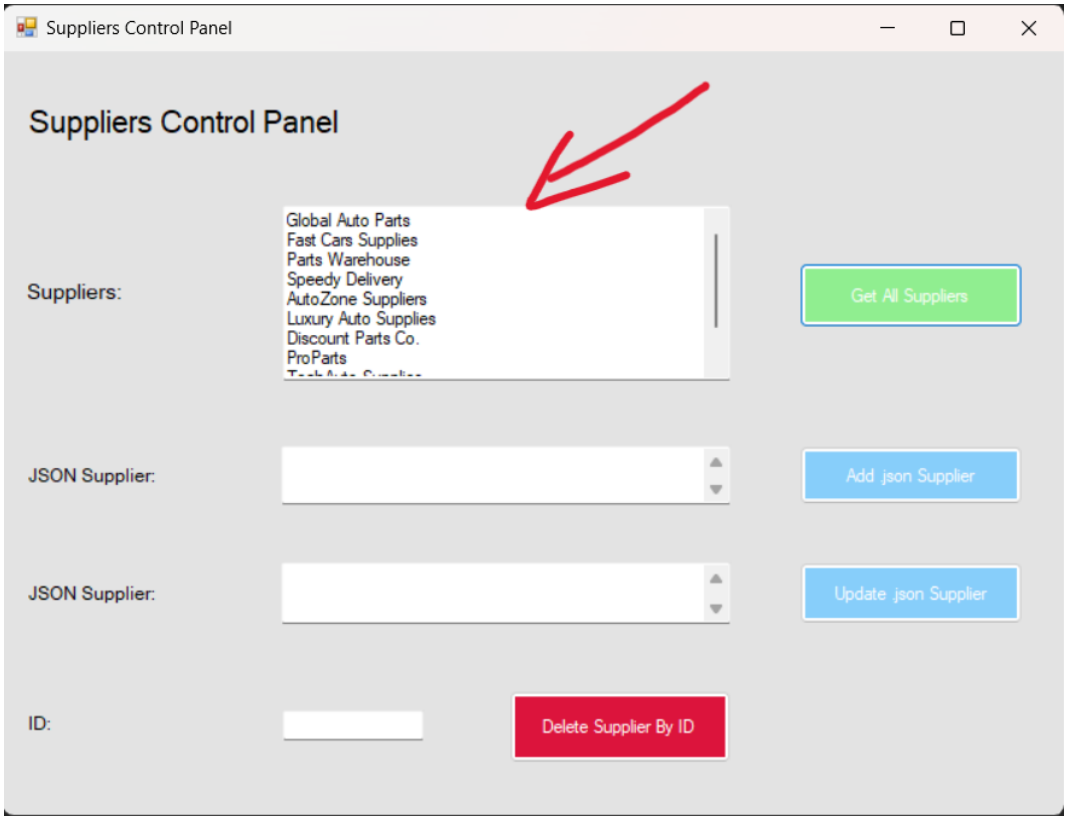
- aplikacje webowe oparte na frameworkach takich jak **React**, **Angular** czy **Vue.js**,
- aplikacje mobilne tworzone w **Flutter**, **React Native** lub **Swift**,
- aplikacje desktopowe budowane z użyciem innych frameworków.

Dzięki temu podejściu frontend może być dostosowany do różnych potrzeb użytkowników, co znacząco zwiększa wszechstronność i skalowalność rozwiązania. Przykładowo, firma może zaimplementować aplikację mobilną do szybkiego zarządzania dostawcami, podczas gdy inny zespół pracuje nad rozbudowaną aplikacją webową dla większych operacji.

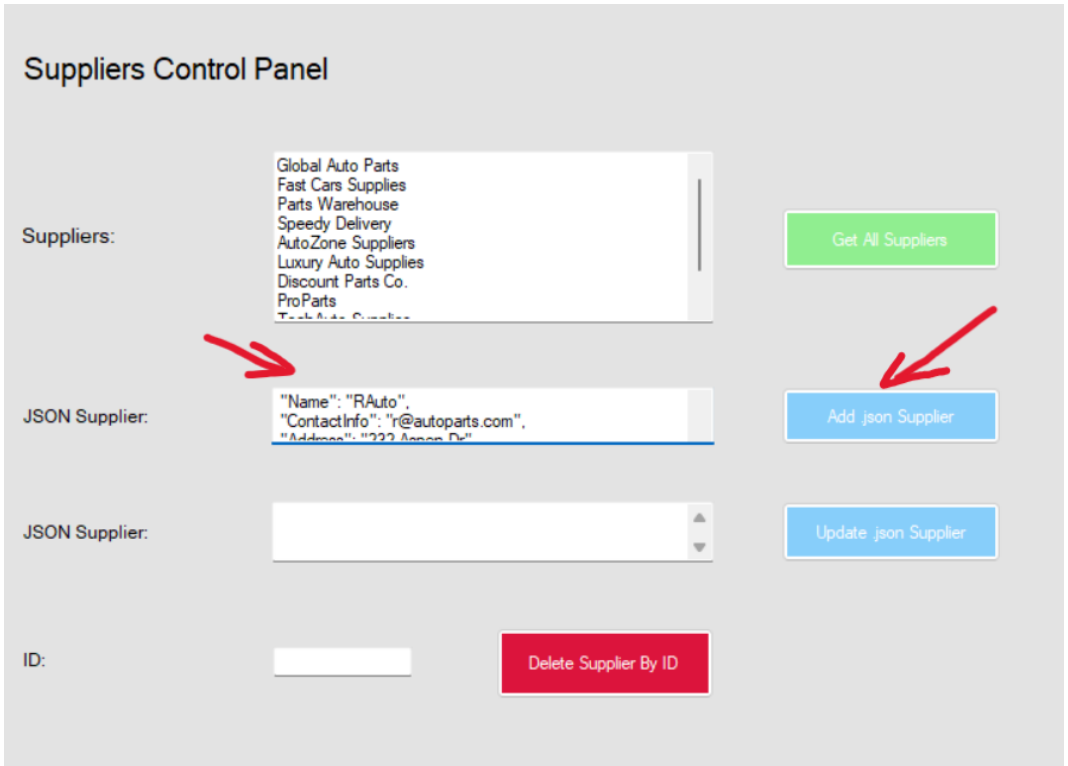
4.1.4 Zrzuty ekranu



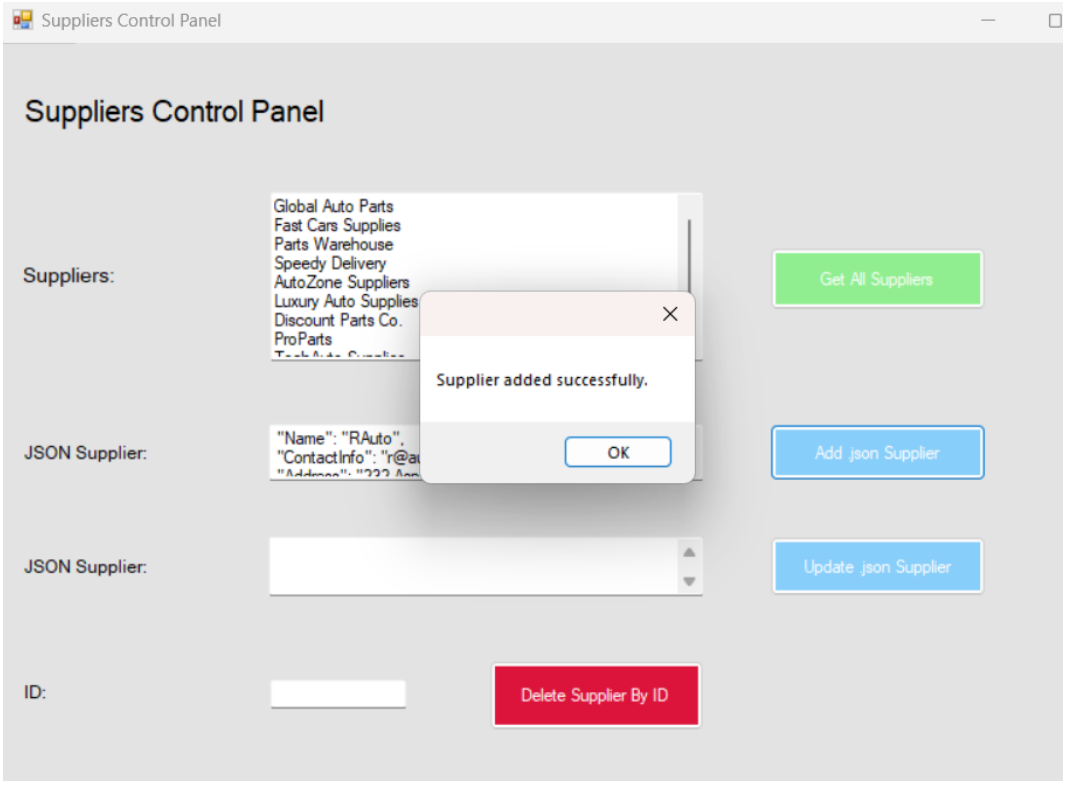
Rysunek 4.1: Ekran startowy panelu. Funkcja załaduj listę dostawców za pomocą zielonego przycisku.



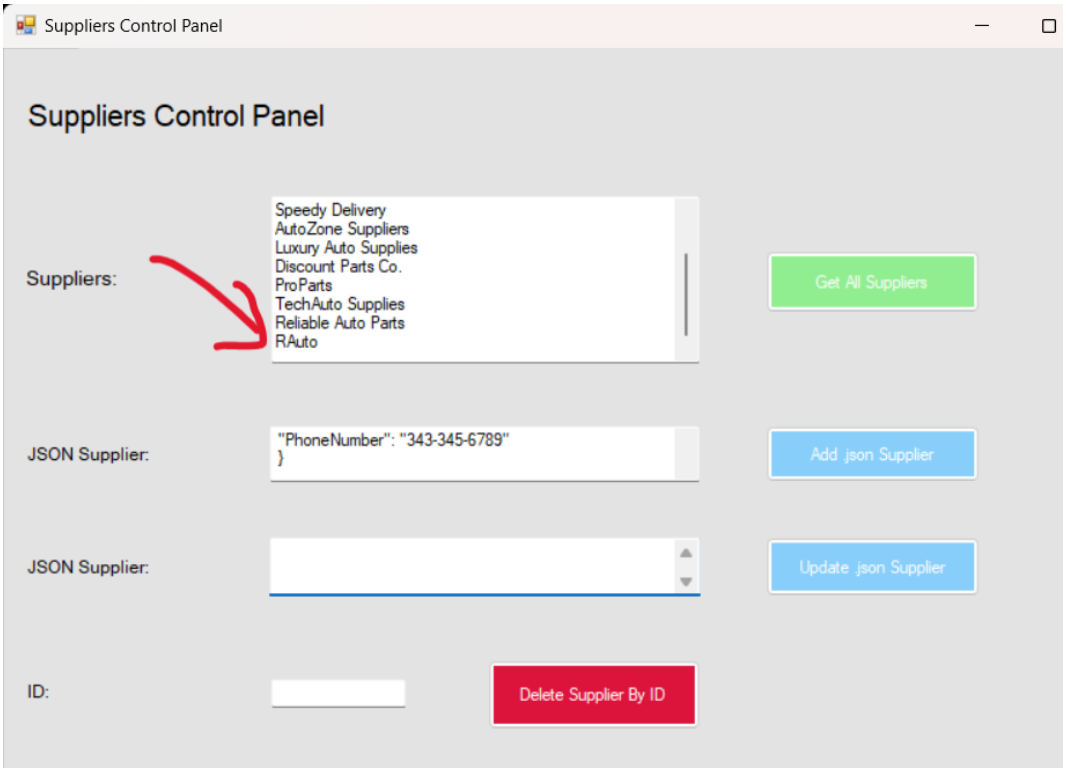
Rysunek 4.2: Lista dostawców załadowana w pierwszym polu tekstowym po kliknięciu przycisku **Załaduj**.



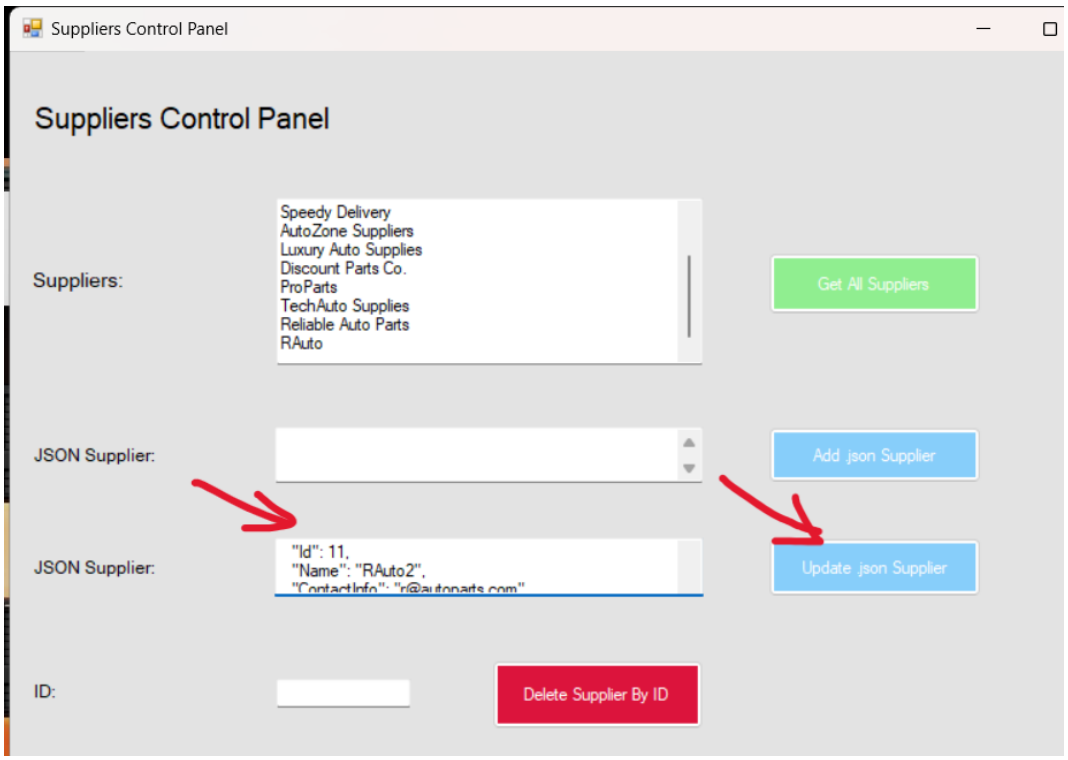
Rysunek 4.3: Dodawanie nowego dostawcy. Drugi tekstboks służy do wprowadzania danych nowego dostawcy. Kliknij **Dodaj**, aby zatwierdzić.



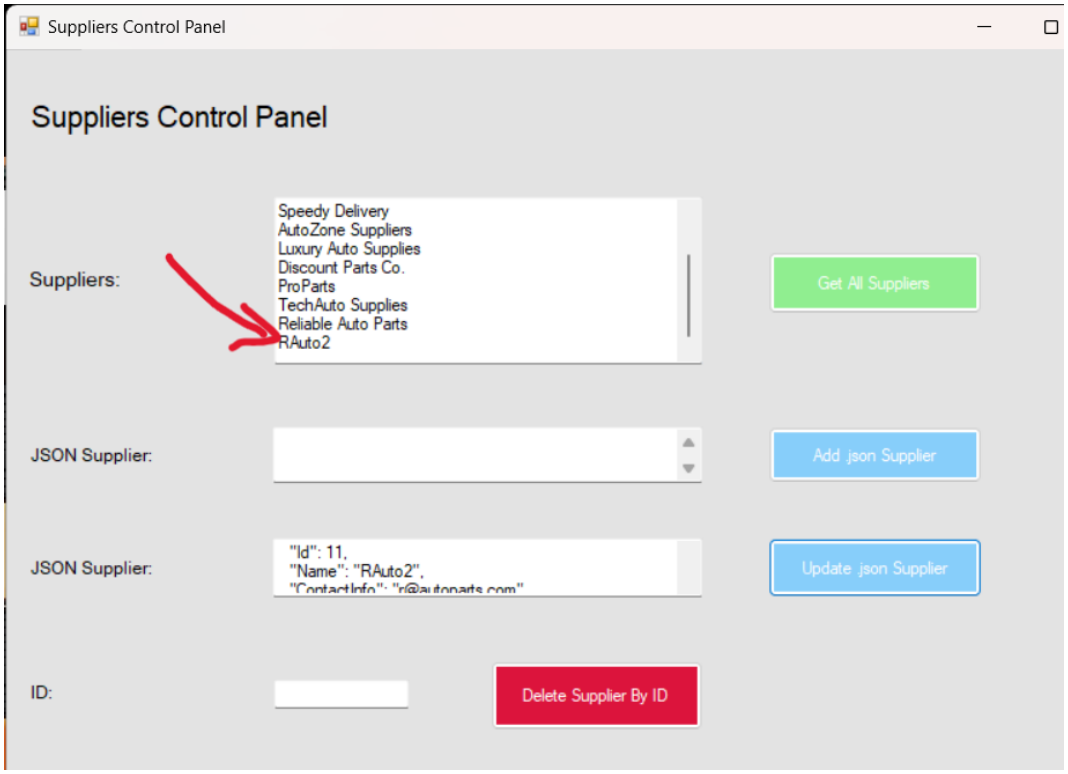
Rysunek 4.4: Potwierdzenie pomyślnego dodania nowego dostawcy.



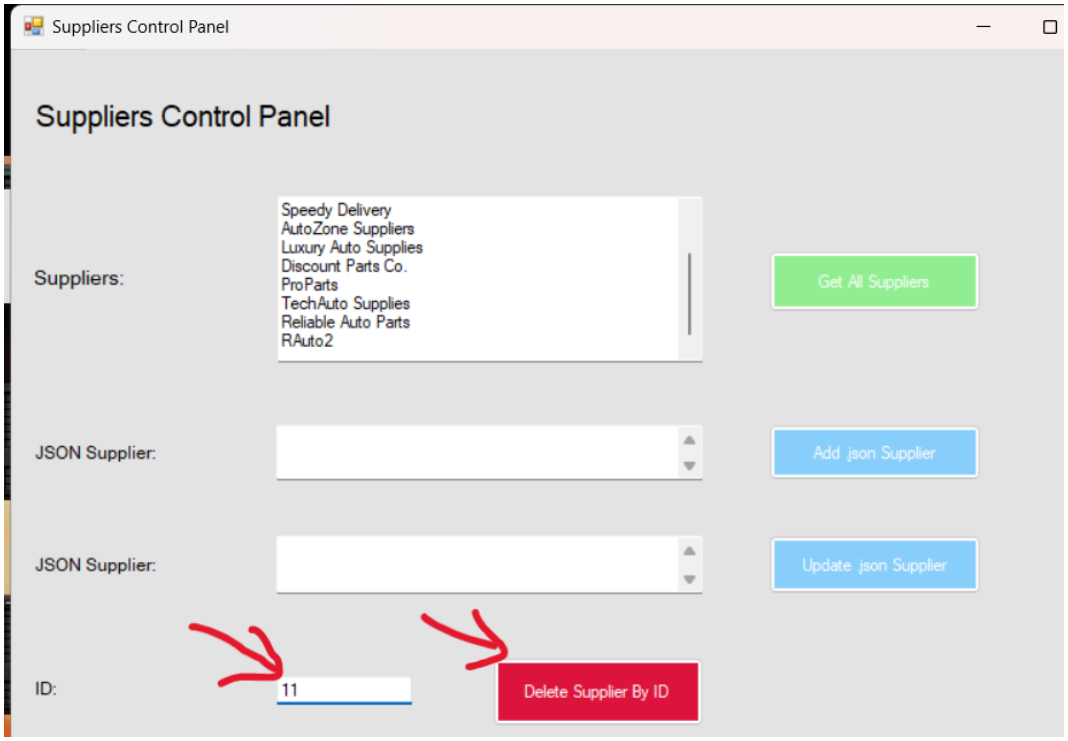
Rysunek 4.5: Potwierdzenie pomyślnego dodania nowego dostawcy.



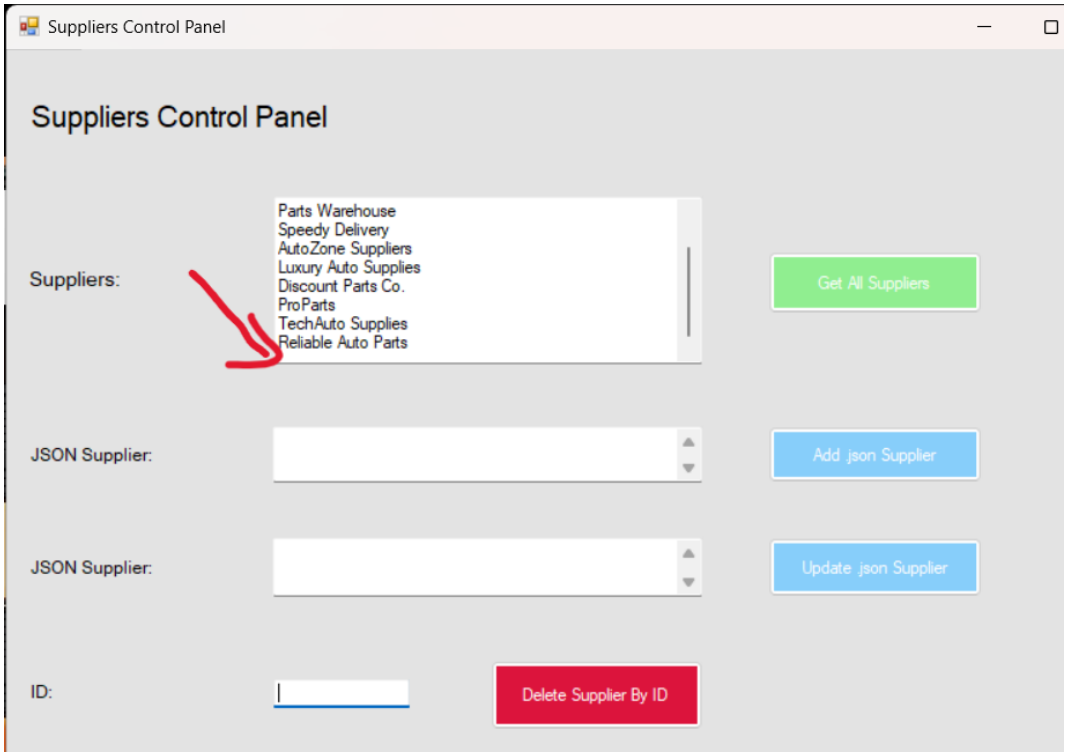
Rysunek 4.6: Wprowadzanie danych dostawcy do aktualizacji w trzecim tekście i zatwierdzanie przyciskiem **Aktualizuj**.



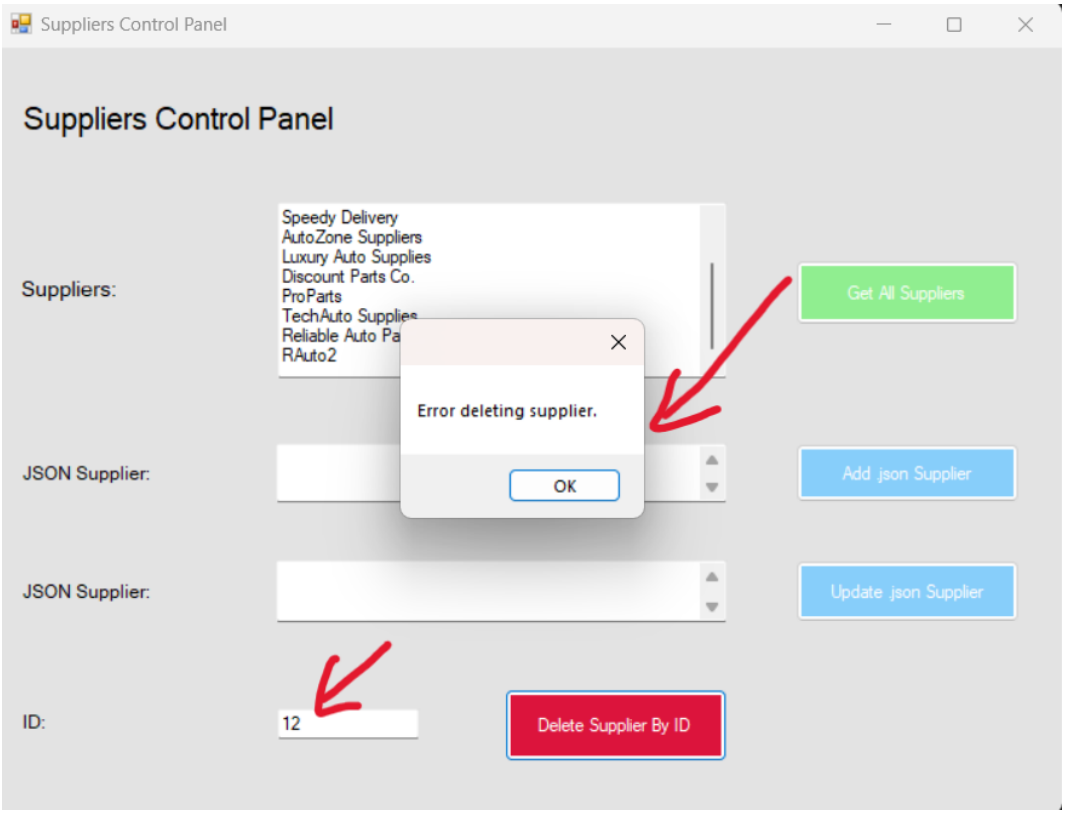
Rysunek 4.7: Lista dostawców wczytana po modyfikacji z możliwością usunięcia rekordu po ID.



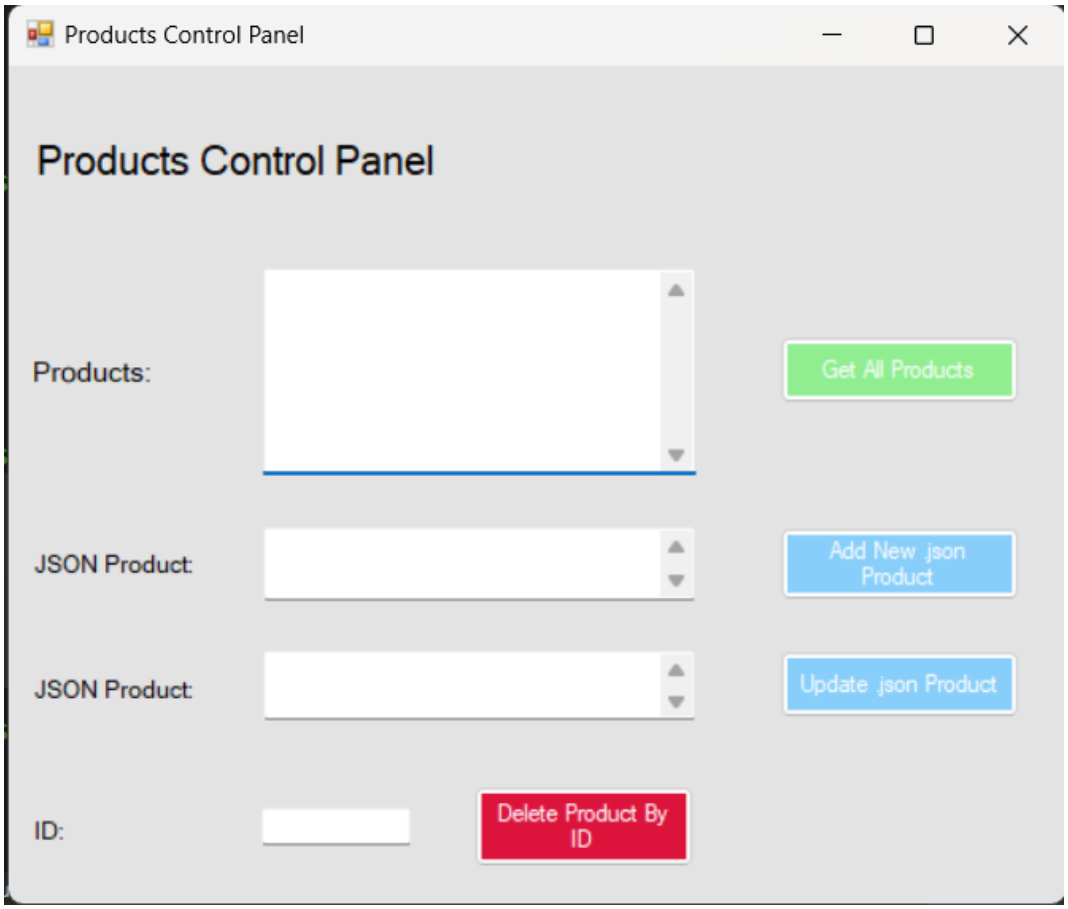
Rysunek 4.8: Usunięcie dostawcy po ID za pomocą czwartego tekstowego pola i czerwonego przycisku **Usuń**.



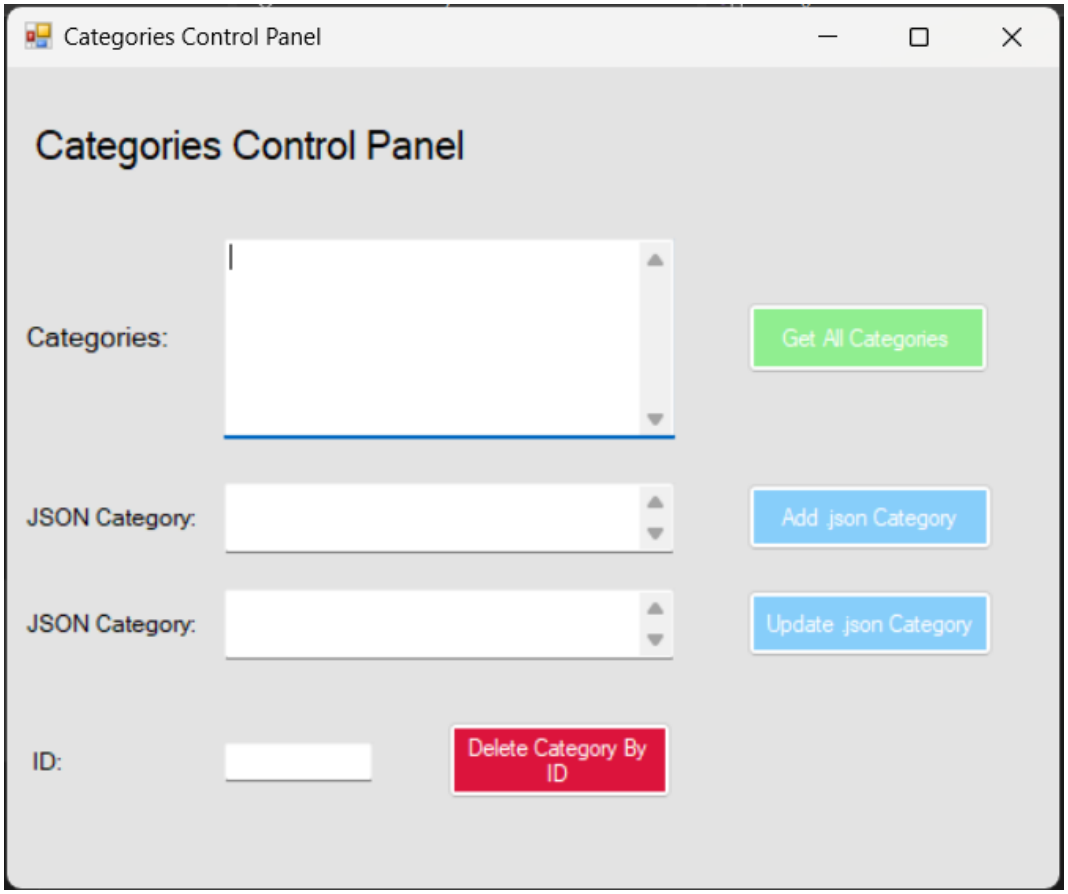
Rysunek 4.9: Wyświetlenie danych po usunięciu.



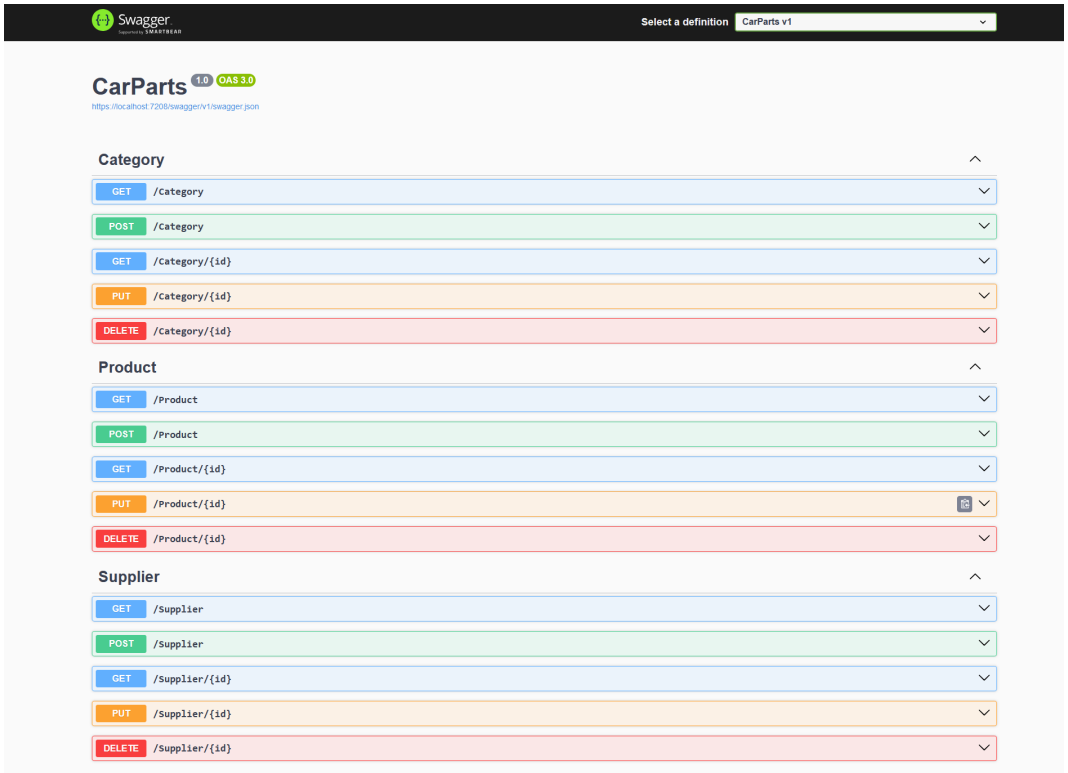
Rysunek 4.10: Wyświetlenie błędu lub informacji o nieprawidłowych danych wprowadzonego dostawcy.



Rysunek 4.11: Panel zarządzania produktami z funkcjami podobnymi do Panelu Dostawców.



Rysunek 4.12: Panel zarządzania kategoriami z możliwością dodawania, aktualizowania i usuwania kategorii.



Rysunek 4.13: Widok dokumentacji API (Swagger) dla interakcji z backendem aplikacji.

Rozdział 5

Podsumowanie

W ramach projektu zrealizowano wszystkie zaplanowane etapy, obejmujące analizę, projektowanie oraz implementację aplikacji. Ostatecznym rezultatem jest funkcjonalna aplikacja desktopowa do zarządzania sklepem z częściami samochodowymi, wykorzystująca technologię Windows Forms i backend oparty na REST API. Aplikacja została zintegrowana z systemem kontroli wersji Git, a jej kod źródłowy umieszczono w publicznym repozytorium.

5.1 Zrealizowane prace

1. **Analiza wymagań**: Zidentyfikowano kluczowe funkcjonalności aplikacji, takie jak zarządzanie dostawcami, produktami i kategoriami.

2. **Projektowanie aplikacji**: Przygotowano strukturę aplikacji, diagramy klas i przepływu danych.

3. **Implementacja serwisów i kontrolerów**: Zrealizowano backend z wykorzystaniem .NET.

4. **Interfejs użytkownika**: Stworzono GUI w technologii Windows Forms, zapewniające intuicyjność i wygodę użytkowania.

5. **Testowanie i optymalizacja**: Przeprowadzono testy jednostkowe i integracyjne oraz optymalizację wydajności.

6. **Dokumentacja**: Przygotowano dokumentację projektu, w tym diagramy Gantta i szczegółowy opis funkcjonalności.

5.2 Możliwości dalszego rozwoju

Aplikacja jest modułowa i pozwala na łatwy rozwój. Możliwe kierunki rozwoju to:

- Rozbudowa bazy danych o nowe modele i specyficzne funkcje w endpointach API.
- Migracja na aplikację webową z użyciem nowoczesnych frameworków.
- Rozszerzenie funkcjonalności o generowanie raportów, analizy danych czy powiadomienia email.
- Stworzenie wersji mobilnej z wykorzystaniem Xamarin lub Flutter.
- Ulepszenie UX/UI z wykorzystaniem nowoczesnych bibliotek graficznych.

5.3 Podsumowanie końcowe

Projekt został zrealizowany zgodnie z założeniami i harmonogramem. Aplikacja spełnia wymagania i jest gotowa do dalszego rozwoju. Modularność i użyte technologie umożliwiają łatwą adaptację do zmieniających się potrzeb i rozbudowy funkcjonalności, co stanowi solidną podstawę do wdrożenia w środowisku produkcyjnym.

Bibliografia

[1] Swagger Documentation, *https://swagger.io/docs/*, dostęp: 20.01.2025.

[2] Microsoft, *ASP.NET Core Documentation*, Microsoft, dostęp: 20.01.2025.

[3] Fielding, R. T., *Architectural Styles and the Design of Network-based Software Architectures*, University of California, 2000.

[4] Microsoft, *Windows Forms Documentation*, Microsoft, dostęp: 20.01.2025.

[5] Git Documentation, *https://git-scm.com/doc*, dostęp: 20.01.2025.

Spis rysunków