${\bf Dokumentacja~projektu}-{\tt Przetwarzanie~danych}$

hierarchicznych w oparciu o typ danych XML

Maciej Leśniak

15.06.2025

https://github.com/maciekgangus/projectXML

Spis treści

1	Opis problemu i funkcjonalności API	2
2	Typy danych i metody API	2
3	Implementacja API	3
4	Testy jednostkowe	4
5	Przykładowa aplikacja	4
6	Podsumowanie i wnioski	7
7	Literatura	8

1 Opis problemu i funkcjonalności API

Celem projektu było stworzenie systemu zarządzania strukturami hierarchicznymi zapisanymi w formacie XML w bazie danych SQL Server. System może być wykorzystany m.in. do reprezentowania drzew geologicznych lub struktur organizacyjnych firm. Dzięki wykorzystaniu typu danych XML możliwe jest przechowywanie złożonych, wielopoziomowych struktur w jednej kolumnie bazy danych, co upraszcza ich odczyt, modyfikację i archiwizację.

Udostępnione API pozwala na:

- tworzenie nowych drzew wraz z nazwą i zawartością XML,
- pobieranie wszystkich drzew lub konkretnego drzewa po ID,
- dodawanie węzłów do wskazanego miejsca w drzewie przy użyciu pełnej ścieżki (XPath lub uproszczonej ścieżki elementów),
- usuwanie węzłów z drzewa na podstawie pełnej ścieżki,
- generowanie raportów z wybranego fragmentu drzewa,
- usuwanie całego drzewa z bazy danych.

2 Typy danych i metody API

Do przechowywania danych wykorzystywany jest typ XML dostępny w Microsoft SQL Server. W tabeli OrganizationTrees każda struktura przechowywana jest jako tekstowy zapis XML w kolumnie TreeData. Modyfikacja danych XML odbywa się z wykorzystaniem klasy XElement i metody XPathSelectElement, która umożliwia wyszukiwanie węzłów po ścieżce — jednak nie zawsze w pełni wspiera składnię XPath 1.0 (brak obsługi funkcji agregujących czy namespace'ów).

Struktura danych:

- Id klucz główny (liczba całkowita),
- TreeName nazwa drzewa,
- TreeData dane XML w postaci tekstu (typ kolumny: xml).

Ścieżki do węzłów:

Scieżki wprowadzane przez użytkownika są przekazywane jako parametr do XPathSelectElement. Oznacza to, że muszą one być dokładne i jednoznaczne – brak dopasowania skutkuje błędem. Przykład:

/Root/Department[@id='3']/Employee

Główne metody API:

- GET /api/OrganizationTree pobiera wszystkie dostępne drzewa,
- GET /api/OrganizationTree/id pobiera jedno drzewo po jego identyfikatorze,
- POST /api/OrganizationTree tworzy nowe drzewo na podstawie przesłanego XML,
- PATCH /api/OrganizationTree/id/addnode dodaje nowy węzeł do drzewa,
- DELETE /api/OrganizationTree/id/removenode usuwa wskazany węzeł z drzewa,
- GET /api/OrganizationTree/id/report?path=... zwraca wybrany fragment XML (raport),
- DELETE /api/OrganizationTree/id usuwa całe drzewo.

3 Implementacja API

Projekt został zrealizowany jako aplikacja webowa oparta na platformie ASP.NET Core (backend) i Vue 3 (frontend). Komunikacja odbywa się w oparciu o żądania HTTP oraz dane w formacie XML przesyłane w treści zapytań i odpowiedzi.

Backend (ASP.NET Core):

- Wykorzystano Entity Framework Core do mapowania danych do SQL Server,
- Baza danych i struktura tabeli są generowane automatycznie na podstawie klasy kontekstu OrganizationTreeContext i klasy modelu OrganizationTree,
- Klasa OrganizationTree zawiera właściwości: Id, TreeName oraz TreeData, przy czym ta ostatnia jest oznaczona jako typ xml poprzez atrybut [Column(TypeName = "xml")] oraz dodatkowo w metodzie OnModelCreating,
- XML jest przechowywany jako typ xml w bazie danych i obsługiwany jako tekst w aplikacji przy użyciu XElement,
- Kontroler OrganizationTreeController obsługuje wszystkie operacje CRUD oraz manipulacje XML,
- Do pracy z XML wykorzystano klasę XElement oraz przestrzeń nazw System.Xml.XPath.

Frontend (Vue 3):

- Aplikacja kliencka umożliwia tworzenie, edytowanie, podgląd i usuwanie drzew XML,
- Komponenty umożliwiają dynamiczne wprowadzanie danych XML i ścieżek,
- Każda akcja (create, delete, patch) wywołuje odpowiednie endpointy API,
- XML jest formatowany i prezentowany w formacie tekstowym oraz uproszczonym drzewie,
- Obsługa błędów po stronie klienta pokazuje komunikaty przy nieudanych akcjach.

Walidacja i obsługa błędów:

- Weryfikacja poprawności składni XML po stronie klienta,
- Obsługa wyjątków po stronie serwera: XmlException, brak węzła, nieprawidłowa ścieżka,
- Zwroty statusów HTTP (400, 404, 500) z komunikatami opisującymi problem.

4 Testy jednostkowe

Testy jednostkowe zostały przygotowane z użyciem frameworka pytest oraz biblioteki requests. Każdy test symuluje konkretne wywołania API z poziomu użytkownika i sprawdza poprawność odpowiedzi serwera.

Zakres testów:

- tworzenie drzewa i pobieranie danych (test_create_and_get_tree),
- aktualizacja danych w drzewie (test_update_tree),
- dodanie nowego węzła (test_add_node),
- usuwanie węzła (test remove node),
- generowanie raportu całego drzewa (test_generate_report_full_tree),
- generowanie raportu poddrzewa (test_generate_report_subtree),
- walidacja błędnego XML przy tworzeniu i aktualizacji (test_invalid_xml_create, test_invalid_xml_update),
- usuwanie drzewa (test_delete_tree).

Środowisko testowe:

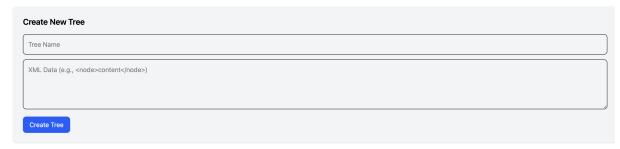
- Backend uruchomiony lokalnie pod adresem: http://localhost:5000,
- Testy uruchamiane lokalnie za pomocą polecenia pytest,
- Dane testowe identyfikowane po nazwie TestTree i czyszczone po każdym teście,
- Komunikacja odbywa się przez żądania HTTP z nagłówkiem Content-Type: application/xml.

5 Przykładowa aplikacja

Aplikacja frontendowa została stworzona w Vue 3 i udostępnia graficzny interfejs do komunikacji z API. Umożliwia interakcję z hierarchicznymi danymi XML bez potrzeby ręcznego wysyłania zapytań HTTP.

Dostępne funkcjonalności:

• Tworzenie nowego drzewa: użytkownik podaje nazwę oraz strukturę XML.



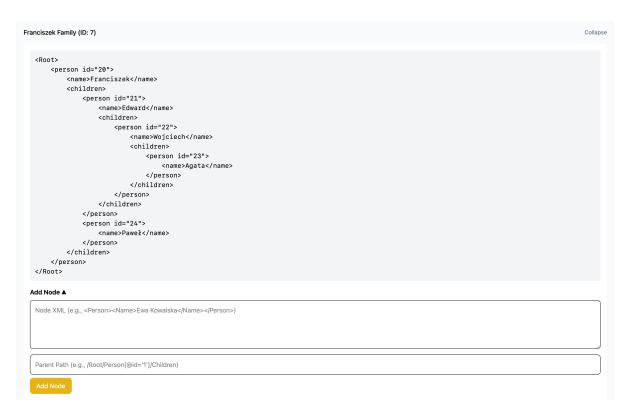
• Wyświetlanie listy utworzonych drzew: z ID i możliwością rozwinięcia szczegółów.



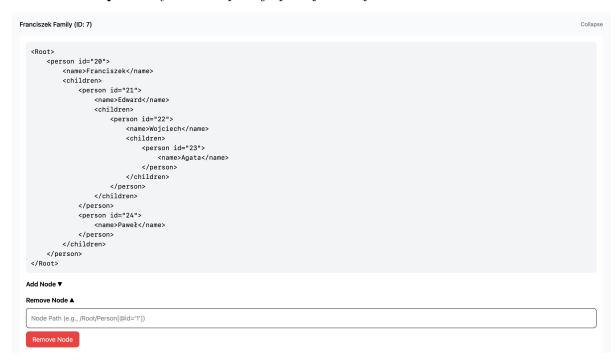
• Przeglądanie zawartości XML drzewa: prezentacja danych w czytelnym formacie.

```
Franciszek Family (ID: 7)
       <person id="20">
           <name>Franciszek</name>
           <children>
               <person id="21">
                   <name>Edward</name>
                   <children>
                       <person id="22">
                           <name>Wojciech</name>
                                <person id="23">
                                    <name>Agata</name>
                                </person>
                            </children>
                   </children>
               </person>
               <person id="24">
                   <name>Paweł</name>
               </person>
          </children>
  </Root>
 Remove Node ▼
 Generate Report ▼
```

• Dodawanie węzła: na podstawie pełnej ścieżki oraz podanego XML.



• Usuwanie węzła: użytkownik podaje pełną ścieżkę do elementu XML.



• Generowanie raportu: wybrany fragment XML wyświetlany jako wynik.

```
<cniiaren>
                <person id="21">
                     <name>Edward</name>
                     <children>
                          <person id="22">
                               <name>Woiciech</name>
                                   <person id="23">
                                        <name>Agata</name>
                                   </person>
                              </children>
                          </person>
                     </children>
                </person>
                <nerson id="24">
                    <name>Paweł</name>
                </person>
           </children>
      </person>
 </Root>
Remove Node A
 Node Path (e.g., /Root/Person[@id='1'])
Generate Report ▲
 /person[@id='20']/children/person[@id='21']
 Generate Report
 • person:

- (name)Edward

• children:
     • person:
       - (name)Wojciech• children:
         • person:
- (name)Agata
```

- Usuwanie całego drzewa: operacja z poziomu interfejsu.
- Obsługa błędów: komunikaty walidacyjne i błędy ścieżki/XML.

6 Podsumowanie i wnioski

Zrealizowany projekt stanowi funkcjonalne i rozszerzalne rozwiązanie do zarządzania danymi hierarchicznymi w formacie XML, przechowywanymi w bazie danych SQL Server. Dzięki połączeniu technologii ASP.NET Core, Vue 3 oraz typu danych XML możliwe było stworzenie aplikacji, która w sposób przejrzysty i interaktywny umożliwia zarządzanie strukturami drzewiastymi.

Opracowane API pozwala na pełen cykl życia danych: tworzenie nowych drzew, edycję zawartości, modyfikację konkretnych węzłów oraz generowanie raportów z fragmentów danych. Frontend w technologii Vue 3 zapewnia użytkownikowi wygodny i intuicyjny interfejs do korzystania z tych funkcji bez konieczności znajomości protokołu HTTP ani bezpośredniego wysyłania zapytań.

W trakcie realizacji projektu zauważono następujące wnioski:

- Typ danych xml w SQL Server umożliwia elastyczne przechowywanie złożonych struktur, jednak jego manipulacja wymaga ostrożności, zwłaszcza w zakresie ścieżek XPath.
- Wykorzystanie metody XPathSelectElement zapewnia prostą integrację wyszukiwania, ale wymaga dokładnego podania ścieżki co może stanowić ograniczenie przy bardziej złożonych zapytaniach.

- Oddzielenie warstwy frontend od backendu pozwoliło na lepszą modularność i łatwiejsze testowanie oraz rozwój poszczególnych komponentów.
- Testy jednostkowe API potwierdziły poprawność działania oraz odporność systemu na typowe błędy użytkownika (np. błędny XML, nieistniejąca ścieżka, brak danych).

Projekt może być w przyszłości rozszerzony o:

- obsługę pełniejszej składni XPath (np. poprzez integrację z bibliotekami zewnętrznymi),
- możliwość przeszukiwania wielu węzłów jednocześnie,
- wersjonowanie drzew i zapisywanie historii zmian,
- rozbudowany mechanizm autoryzacji i ograniczeń dostępu do danych.

Stworzony system potwierdził, że przechowywanie i przetwarzanie danych XML w SQL Server jest realną i efektywną alternatywą dla klasycznych struktur relacyjnych w przypadkach, gdy dane mają charakter hierarchiczny.

7 Literatura

- Microsoft Docs XML Data (SQL Server): https://learn.microsoft.com/en-us/sql/relational-databases
- A. Freeman, D. Sanderson, *Pro ASP.NET Core MVC 2*, Apress, 2017 wzorce tworzenia kontrolerów API.
- Dokumentacja biblioteki Vue.js –
 https://vuejs.org/guide/introduction.html
- Stack Overflow społeczność i dyskusje dotyczące obsługi XML w .NET i błędów XPath.