

2023-05-05

Dokumentacja projektu

Przetwarzanie dokumentów XML w
bazie danych MS SQL Server



Bartłomiej Wypart

AKADEMIA GÓRNICZO-HUTNICZA W KRAKOWIE

1 SPIS TREŚCI

1.	Opis problemu.....	1
2.	Struktura bazy danych	2
3.	Metody oraz typy danych udostępnione w ramach API	3
4.	Przeprowadzone testy jednostkowe	11
5.	Opis aplikacji	14
5.1	XML documents list	14
5.2	Create document	17
5.3	XML content	20
6.	Podsumowanie	22
7.	Literatura	22

1. OPIS PROBLEMU

Typ danych XML jest bardzo popularnym sposobem przechowywania informacji. Używa się go często do zapisu pewnych stanów aplikacji, danych itp. Tak jak jego rówieśnik JSON, zyskał dużą popularność ze względu na jego prostotę oraz funkcjonalność. Mimo, że jest bardziej wymagający jeżeli chodzi o sposób przetwarzania, tak w dalszym ciągu jest używany i wspierany.

W bazie danych MS SQL Server została dostarczona funkcjonalność przetwarzania dokumentów XML. Takie funkcje zostały dodane do natywnego typu danych XML dostarczonego w ramach usługi.

Przetwarzanie dokumentu polega na wykonywaniu poleceń typu SELECT oraz UPDATE na kolumnie z typem danych XML. Jednak nie wszystkie konstrukcje są przyjemne z punktu widzenia użytkownika. W ramach projektu zostało stworzone API, które swoją funkcjonalnością oraz prostotą użytkowania rozwiązuje ten problem.

Do podstawowych funkcjonalności API należą:

- sprawdzenie poprawności połączenia z bazą danych,
- dodawanie dokumentów do bazy z zadaniem tytułem, opisem oraz ciałem dokumentu XML,
- dodawanie dokumentów z wczytanego pliku XML,
- modyfikacja parametrów, takich jak tytuł, opis, całość dokumentu XML,
- czytanie dokumentów XML z bazy (wszystkich, o zadanych ID lub o zadanych tytułach),
- usuwanie dokumentów XML korzystając z jego ID lub jego tytułu,
- usuwanie wszystkich dokumentów,
- dodawanie węzłów wraz z tekstem,
- dodawanie atrybutów do węzłów,
- edycja poszczególnych atrybutów wraz z tekstem,
- sprawdzanie, czy dany węzeł istnieje,

- zwrócenie informacji o wszystkich atrybutach danego węzła,
- konwersja węzłów XML na przyjemną dla programisty listę słowników C#,
- usuwanie atrybutów z węzłów,
- usuwanie węzłów.

Wszystkie metody oraz zwracane typy danych zostały opisane w punkcie 2. niniejszej dokumentacji.

Dodatkowo została stworzona klasa, która będzie wspomagała pisanie zapytań xQuery dla osób, które niekoniecznie miały styczność. Nazywa się **QueryBuilder**. Funkcjonalności także zostały opisane w punkcie 2. niniejszego dokumentu.

Zostały napisane testy jednostkowe z użyciem *xUnit*, które sprawdzają poprawność działania wszystkich metod na oddzielnej bazie danych, służącej tylko do testowania (a więc wykonanie testów nie wpływa na faktyczny stan bazy).

Dodatkowo została napisana aplikacja, która realizuje funkcjonalności dostarczone przez API.

Cały projekt został napisany z użyciem języka C#, wykorzystując przy tym wersję .NET 6.0. Aplikacja internetowa, oparta o model MVC 5, została napisana z użyciem języków: HTML5, CSS3 (z użyciem *Bootstrap5*), JS oraz hepler-ów dostarczonych w ramach frameworka *ASP.NET*. Do bazy danych został wykorzystany MSSQL Server w wersji 2016.

2. STRUKTURA BAZY DANYCH

Do realizacji projektu została wykorzystana baza danych MSSQL Server 2016.

Do realizacji projektu została stworzone dwie bazy danych. Jedna do właściwej implementacji funkcjonalności API, a druga do przeprowadzania testów jednostkowych.

Do projektu została stworzona jedna tabela, która przechowuje informację, o ID dokumentu (jest to klucz główny z automatycznym numerowaniem), tytule, krótkim opisie oraz właściwej treści dokumentu XML (*XDocument*). Taka struktura tabeli pozwala na optymalne przetwarzanie dokumentów XML.

Aby projekt mógł działać w pełni poprawnie, wymagane jest odtworzenie struktury bazy danych. Sprowadza się to do wykonania poleceń:

```
CREATE DATABASE BD2_Project;
USE BD2_Project;
GO
```

```
CREATE TABLE XMLDocument ( Id INT PRIMARY KEY IDENTITY(1, 1), [Title] VARCHAR(80),
Description VARCHAR(1000), XDocument XML );
```

Oczywiście nazwę bazy danych można dowolnie edytować. Jednak warto pamiętać, aby ustawić odpowiednią wartość **ConnectionString** w pliku *appsettings.json*.

3. METODY ORAZ TYPY DANYCH UDOSTĘPNIONE W RAMACH API

W ramach API został stworzony typ danych **XMLDoc**, który zawiera informacje o dokumencie XML. Zawiera on publiczne właściwości:

- `public int Id` – id dokumentu,
- `public string? Title` – tytuł dokumentu,
- `public string? Description` – opis dokumentu,
- `public string? XMLDocument` – Dokument XML przechowywany za pomocą zmiennej tekstowej.

API wymaga podania ścieżki do elementu poprzez proste użycie xQuery. Część osób jednak niekoniecznie może mieć wiedzę na ten temat. Dlatego została dołączona funkcjonalność tworzenia xQuery poprzez obiekt klasy **QueryBuilder**. Przy czym warto zaznaczyć, że obiekt ten nie jest spokrewniony z rzeczywistymi obiektami w bazie danych, a więc walidacja poprawności nazw i struktury dokumentu zależy od użytkownika korzystającego z tego obiektu.

Dostarcza ona następujących funkcjonalności:

- Konstruktor bezargumentowy.
- Konstruktor argumentowy przyjmujący jako pierwszy argument węzeł inicjalizacyjny, a jako drugi argument zmienną typu `bool`, czy węzeł ten jest korzeniem, czy pewnym elementem w dokumencie.
- `public QueryBuilder GoTo(string node)` – metoda, która pozwala na przejście do węzła wewnątrz obecnie wybranego. Jako argument przyjmuje nazwę węzła, do którego ma przejść. Zwraca obiekt typu **QueryBuilder**.
- `public QueryBuilder WithAttribute(string attributeName, string? attributeValue = null)` – metoda dodająca do obecnie wybranej ścieżki atrybut z opcjonalną jego wartością. Parametry te przekazane są jako argument wywołania metody. Zwracany jest obiekt typu **QueryBuilder**.
- `public QueryBuilder At(int position)` – metoda pozwalająca na wybranie indeksu elementu w przypadku, gdy węzłów jest więcej niż jeden. Warto pamiętać, że węzły te indeksowane są do 1. Zwraca obiekt typu **QueryBuilder**.
- `public override string ToString()` – metoda generująca xQuery z operacji wcześniej wykonanych na obiekcie. Zwraca `string` zawierający xQuery ze ścieżką do danego elementu.

Prosty przykład użycia:

```
QueryString query = new QueryString("catalog", true);
query.GoTo("book").WithAttribute("id", "123").GoTo("title");
Console.WriteLine(query.ToString());
```

W rezultacie powinno ukazać się na ekranie: **/catalog/book[@id="123"]/title**.

Dodatkowo zarządzanie dokumentami XML odbywa się przy pomocy obiektu klasy **XMLService**, która jest głównym API.

Metody klasy **XMLService** wraz z opisem ich funkcjonalności:

- Konstruktor główny, przyjmujący jako parametr *connectionString*, służący do połączenia z bazą danych MSSQL Server.
Utworzenie obiektu typu XMLService może wyglądać następująco:

```
string conenctionString = @"DATA
SOURCE=MSSQLServer;INITIAL CATALOG=BD2_Project;Server=(localdb)\mssqllocaldb";
XMLService xService = new XMLService(conenctionString);
```
- `public bool CheckConnection()` – metoda sprawdzająca połączenie z bazą danych. Jeżeli zostanie zwrócone **true**, takie połączenie jest ustanowione. W przeciwnym przypadku zwracana jest wartość **false**.
- `public int CountDocuments()` – metoda zwracająca ilość przechowywanych dokumentów w bazie danych. Zwraca liczbę rekordów.
- `public int CreateDocument(string title, string description, string xmlString)` – metoda tworząca nowy dokument w bazie.

Argumenty metody:

- **title** – tytuł dokumentu
- **description** – opis dokumentu
- **xmlString** – zawartość dokumentu

Metoda waliduje, czy łańcuch znaków przekazany jako xmlString jest poprawny pod względem poprawności składni XML. Jeżeli walidacja nie przebiegnie poprawnie, rzucany jest wyjątek typu **XMLException**.

Jeżeli podany tytuł istnieje już w bazie danych, zostaje zgłoszony wyjątek klasy **Exception** ze stosowną informacją: **Document with this title is now in the database**.

Uzupełniony obiekt zostaje zapisany w bazie danych.

- `public int CreateDocumentFromFile(string title, string description, string filepath)` – metoda tworząca dokument w bazie z parametrami przekazanymi jako argumenty funkcji.

Argumenty metody:

- **title** – tytuł dokumentu
- **description** – opis dokumentu
- **filepath** – ścieżka do pliku, w którym zapisany jest XML.

Metoda podobnie jak **CreateDocument()** na samym początku waliduje poprawność dokumentu XML. Jeżeli dokument jest niepoprawny, rzucany jest wyjątek **XMLException**. Jeżeli podany tytuł istnieje już w bazie danych, zostaje zgłoszony wyjątek klasy **Exception** ze stosowną informacją. Jeżeli jednak wszystko przebiegnie pomyślnie, dokument jest dodawany do bazy danych.

- `public List<XMLDoc>? GetAllDocuments()` – metoda zwracająca wszystkie (o ile istnieje co najmniej jeden) dokumenty z bazy danych. W przypadku, gdy nie ma jeszcze takich dokumentów, zwracana jest wartość **null**.

Typem zwracanym jest lista obiektów typu **XMLDoc**. Typ ten został wprowadzony na potrzeby funkcjonowania biblioteki i opisany został dokładniej na początku niniejszego punktu.

- `public XMLDoc GetDocumentById(int id)` – metoda zwracająca pojedynczy dokument od zadanym numerze identyfikacyjnym.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu

W przypadku braku dokumentu o podanym id, zostaje rzucony wyjątek klasy **Exception** z informacją: **Document with this id does not exist**.

Gdy jednak dokument taki istnieje, zostaje zwrócony obiekt typu **XMLDoc** z odpowiednimi właściwościami.

- `public XMLDoc? GetDocumentByTitle(string title)` – metoda zwracająca pojedynczy dokument od zadanym numerze identyfikacyjnym.

Argumenty metody:

- **title** – tytuł dokumentu

W przypadku braku dokumentu o podanym tytule, zostaje zwrócony wyjątek klasy **Exception** z informacją: **Document with this id does not exist**.

Gdy jednak dokument taki istnieje, zostaje zwrócony obiekt typu **XMLDoc** z odpowiednimi właściwościami.

- `public bool ModifyDocument(int id, string? newTitle = null, string? newDescription = null, string? newXMLDocument = null)` – metoda modyfikująca właściwości dokumentu.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu,
- **newTitle** – nowy tytuł dla dokumentu,
- **newDescription** – nowy opis dla dokumentu,
- **newXMLDocument** – nowy dokument XML.

Jeżeli któraś z własności nie zostanie wprowadzona, zostaje niezmieniona. Dodatkowo została dodana walidacja, która nie pozwala na zmianę tytułu na taki, który już istnieje w bazie. W takiej sytuacji zostanie zgłoszony wyjątek klasy **Exception** z adekwatną informacją, że dany dokument z zadaniem id nie istnieje w bazie.

Jeżeli wszystko przebiegnie pomyślnie, zwracana jest wartość **true**.

- `public bool DeleteDocumentById(int id)` – metoda usuwająca dokument o zadany ID.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu.

Jeżeli dokument o zadany numerze identyfikacyjnym nie istnieje w bazie, zostaje zwrócona wartość **false**.

Jeżeli wszystko przebiegnie pomyślnie, zwracana jest wartość **true**.

- `public int DeleteAllDocuments()` – metoda usuwająca wszystkie dokumenty z bazy. Metoda zwraca liczbę usuniętych dokumentów.

- `public string? GetNodes(int id, string xQuery)` – metoda zwracająca xmlString, który zawiera się w ścieżce podanej jako drugi argument w dokumencie o zadany id.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath

Jeżeli wszystko przebiegnie pomyślnie, zwracany jest string z elementami znajdującymi się pod podaną ścieżką XQuery. Jeżeli taka ścieżka nie istnieje lub dokument z podany numerem id nie istnieje, zwracany jest **null**.

- `public string? GetNodeText(int id, string xQuery)` – metoda zwracająca wartość elementu, którego ścieżka podana jest jako argument xQuery.

Argumenty metody:

- **id** – numer identyfikacyjny dokumenty
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath

Jeżeli dokument z zadaniem id istnieje oraz ścieżka jest poprawna, zwracany jest tekst znajdujący się w węźle. W przeciwnym przypadku zwracany jest **null**.

- `public List<string>? GetAllDocumentNodesQueries(int id, string xQuery)` – metoda zwracająca listę xmlString, w której każdy element to znaleziony wzorec według ścieżki podanej w argumencie metody.

Argumenty metody:

- **id** – numer identyfikacyjny dokumenty
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath

Jeżeli dokument z zadaniem id istnieje oraz ścieżka jest poprawna, zwracana jest lista z łańcuchami tekstowymi. W przeciwnym wypadku, zwracana jest wartość **null**.

- `public List<string>? GetAllDocumentNodesValues(int id, string xQuery)` – metoda zwracająca listę wartości stringów, w której każdy element to znaleziony wzorec według ścieżki podanej w argumencie metody.

Argumenty metody:

- **id** – numer identyfikacyjny dokumenty
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath

Jeżeli dokument z zadaniem id istnieje oraz ścieżka jest poprawna, zwracana jest lista z łańcuchami wartości. W przeciwnym wypadku, zwracana jest wartość **null**.

W odróżnieniu od poprzedniej metody, ta zwraca tylko wartości, nie cały xmlString.

- `public bool CheckNodeIfExists(int id, string xQuery)` – metoda sprawdzająca, czy węzeł określony przez ścieżkę istnieje, czy nie.

Argumenty metody:

- **id** – numer identyfikacyjny dokumenty
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath

Jeżeli węzeł istnieje, metoda zwraca wartość **true**, w przeciwnym przypadku **false**.

- `public Dictionary<string, string>? GetAllAttributes(int id, string xQuery)` – metoda zwracająca słownik, w której pierwszym elementem jest nazwa atrybutu węzła wskazanego przez ścieżkę, a drugą wartością jest wartość tego atrybutu.

Argumenty metody:

- **id** – numer identyfikacyjny dokumenty
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath

Jeżeli węzeł nie posiada żadnych atrybutów, zwracana jest wartość **null**. Jeżeli natomiast ścieżka xQuery/XPath jest niepoprawna, rzucony jest wyjątek klasy **Exception** ze stosownym komunikatem: **Node with this path does not exist**. W przeciwnym przypadku zwracany jest słownik, w którym pierwszy element odpowiada za nazwę atrybutu w węźle XML, natomiast drugi za jego wartość.

- `public List<Dictionary<string, string>>? GetStructuredNodes(int id, string xQuery, string[] values)` – metoda zwracająca ustrukturyzowane węzły z postaci słowników. Do każdego dopasowania do ścieżki jest zwracana informacja o węzłach, które następnie w języku C# są łatwo dostępne jako typ słownikowy.

Argumenty metody:

- **id** – numer identyfikacyjny dokumenty
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath
- **values[]** – tablica stringów określająca, jakie wartości węzłów należy pobrać w celu dalszego ich przetwarzania

Metoda zwraca listę słowników, gdzie pierwszy element każdego ze słowników to jest nazwa węzła, natomiast drugi element to wartość tekstowa węzła.

W przypadku wystąpienia błędu (z racji tego, że węzły mogą mieć wartość null) zwracany jest wyjątek **Exception**.

- `public List<string>? GetNodesWithAttribute(int id, string xQuery, string nameOfAttribute, string? valueOfAttribute = null)` – metoda zwracająca listę stringów, które reprezentują XML.

Argumenty metody:

- **id** – numer identyfikacyjny dokumenty
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath
- **nameOfAttribute** – nazwa atrybutu
- **valueOfAttribute** – wartość atrybutu. Argument opcjonalny, jeśli nie podany, szukane są węzły, które posiadają ten atrybut nie patrząc na wartości.

Gdy ścieżka XPath/xQuery jest niepoprawna, zwracana jest wartość **null**. W pozostałych przypadkach, gdy jako argument zostanie przekazana tylko nazwa atrybutu, zostaną zwrócone węzły, które posiadają zadany atrybut w postaci listy napisów znakowych. Jeżeli zostanie dodany dodatkowo argument `valueOfAttribute`, zostaną zwrócone tylko te węzły, u których atrybut z daną nazwą, posiada zadaną wartość.

- `public string GetValueOfAttribute(int id, string xQuery, string nameOfAttribute)` – metoda zwracająca wartość danego atrybutu, zadanego przez ścieżkę w argumencie metody.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath
- **nameOfAttribute** – nazwa atrybutu, którego wartość ma zostać pobrana

Z uwagi na to, że dany atrybut może być równy null, w przypadku, gdy takiego atrybutu nie ma, lub ścieżka niepoprawnie prowadzi do określonego węzła, zostaje zgłoszony wyjątek klasy `Exception` ze stosownym komunikatem.

Gdy natomiast ścieżka do węzła jest poprawna oraz nazwa atrybutu jest zgodna, zostaje zwrócony napis, określający wartość tego atrybutu w węźle.

- `public bool AddNewNode(int id, string xQuery, string newNodeString)` – metoda dodająca nowy węzeł do węzła już istniejącego, znajdującego się w określonej ścieżce poprzez XPath lub xQuery.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu
- **xQuery** – ścieżka do nowego węzła
- **newNodeString** – nowa wartość tekstowa, która może zawierać `XMLstring` oraz ma zostać zapisana w węźle

Jeżeli węzeł zostanie stworzony wraz z odpowiednim tekstem, zostanie zwrócona wartość **true**. W przeciwnym przypadku wartość **false**. Dodatkowo jeżeli zostanie wprowadzony zły format XML, zostanie rzucony wyjątek `XMLException` ze stosowną informacją.

- `public bool EditNodeName(int id, string xQuery, string newName)` – metoda zmieniająca wybrany węzeł na inną nazwę, nie tracąc przy tym zawartości poprzedniego węzła.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu
- **xQuery** – ścieżka do węzła, z nazwą teraźniejszą,
- **newName** – nowa nazwa dla węzła

Jeżeli wszystko przebiegnie pomyślnie, zwracana jest wartość **true**, w przeciwnym przypadku – **false**.

- `public bool EditNodeText(int id, string xQuery, string newValue)` – metoda usuwająca atrybut z węzła podanego jako ścieżkę.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu

- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath
- **newValue** – nowa wartość tekstowa dla węzła

Zwracana jest wartość **true**, jeżeli zmiana tekstu w węźle powiodła się. W przeciwnym przypadku zwracany jest **false**.

- `public bool AddAttributeToNode(int id, string xQuery, string nameOfAttribute, string valueOfAttribute)` – metoda usuwająca atrybut z węzła podanego jako ścieżkę.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath
- **nameOfAttribute** – nazwa atrybutu do dodania
- **valueOfAttribute** – wartość atrybutu określonego jako *nameOfAttribute*

Zwracana jest wartość **true**, jeżeli atrybut został dodany do węzła. Jeżeli nie, zwracana jest wartość **false**.

- `public bool RemoveAttributeFromNode(int id, string xQuery, string nameOfAttribute)` – metoda usuwająca atrybut z węzła podanego jako ścieżkę.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath
- **nameOfAttribute** – nazwa atrybutu do usunięcia

Zwracana jest wartość **true**, jeżeli atrybut został usunięty z węzła.

- `public bool DeleteNodeFromDocument(int id, string xQuery)` – metoda usuwająca węzeł z dokumentu XML o zadanej ścieżce.

Argumenty metody:

- **id** – numer identyfikacyjny dokumentu
- **xQuery** – ścieżka do węzła w postaci xQuery lub XPath

Metoda zwraca **true**, jeżeli węzeł już nie istnieje we wskazanej ścieżce.

- `private static bool validateXML(string xmlString)` – prywatna metoda służąca do walidacji dokumentów XML.

Argumenty metody:

- **xmlString** – łańcuch znaków z dokumentem XML

Jeżeli ciąg znaków spełnia wymagania narzucone przez dokument XML, zwracana jest wartość **true**, w przeciwnym przypadku **false**.

- `private int execNonQuery(string command, List<string, string>? parameters = null)` – prywatna metoda wykonująca zapytanie, które nie wymaga pobrania wartości zwracanej przez bazę danych.

Argumenty metody:

- **command** – polecenie SQL. W przypadku zmiennej należy zastosować zapis z „@”, jak np. *WHERE Id = @id*;
- **parameters** – lista krotek złożonych z dwóch łańcuchów znaków – parametrów do wykonania polecenia SQL. Argument opcjonalny, domyślnie ustawiony na wartość **null**

Jeżeli polecenie zostanie wykonane poprawnie, zwrócona zostanie liczba określająca ilość zmodyfikowanych wierszy w bazie danych. W przeciwnym wypadku zostanie zwrócona wartość 0 lub -1, w zależności od tego, czy błąd jest składniowy polecenia SQL, czy polecenie jest napisane poprawnie, jednak żadne rekordy w bazie nie spełniają warunków.

- `private int execScalarValue(string command)` – prywatna metoda zwracająca pojedynczą wartość skalarną z bazy danych w wyniku zrealizowania polecenia.

Argumenty metody:

- **command** – polecenie SQL

Jeżeli polecenie zostanie obsłużone poprawnie, zostanie zwrócona pojedyncza, skalarna wartość z bazy danych. W przeciwnym wypadku zostanie zwrócona wartość -1.

4. PRZEPROWADZONE TESTY JEDNOSTKOWE

Do każdej z funkcji napisanych zostało po około 2 testy jednostkowe w celu wytestowania, gdy do metody zostaną podane złe argumenty oraz gdy wszystko przebiegnie zgodnie z założeniami. Testy znajdują się w dwóch plikach. W jednym z nich, testowana jest funkcjonalność **QueryBuildera**, a w drugim właściwa część aplikacji – API **XMLService**.

W przypadku **QueryBuildera** zostały wykonane testy, które pokazują jego funkcjonalność:

- `Should_ReturnXQueryString_When_MethodApplicationIsCorrect`
- `Should_ReturnXQueryString_When_XQueryStartsWithRootPath`
- `Should_ReturnXQueryString_When_XQueryHasIndexedValues`
- `Should_ReturnXQueryString_When_XQueryHasAttributWithValue`

Poniżej znajduje się lista testów zaimplementowanych, które sprawdzają poprawność działania API serwisu XML.

- Should_ReturnTrue_When_ConnectionStringIsGood
- Should_ThrowException_When_XMLFormatIsInvalid
- Should_ThrowException_When_ThereIsDocumentWithTheSameTitle
- Should_InsertRowToDatabase_When_XMLFormatIsValid
- Should_ThrowException_When_DocumentWithTheTitleDoesNotExist
- Should_ThrowException_When_DocumentWithIdDoesNotExist
- Should_ReturnXMLDocObject_When_DocumentWithTheTitleExists
- Should_ThrowException_When_ModifiedDocumentHasInvalidXMLFormat
- Should_ThrowException_When_ModifiedDocumentDoesNotExist
- Should_CorrectlyModifyRow_When_DocumentExists
- Should_ReturnFalse_When_DeletedDocumentWithIdDoesNotExist
- Should_DeleteRowFromDatabase_When_DocumentWithIdExistsInDatabase
- Should_ReturnZero_When_DeletedAllDocuments
- Should_ReturnXMLString_When_NodeExists
- Should_ReturnNull_When_NodeDoesNotExist
- Should_ReturnNull_When_NodeDoesNotExistDuringReadingNodeText
- Should_ReturnNodeText_When_NodeExists
- Should_ReturnNull_When_DuringExecGetAllDocumentNodesQueriesNodeWithTheNameDoesNotExist
- Should_ReturnListOfXMLString_When_NodeWithTheNameExists
- Should_ReturnNull_When_NodeHasNoAttribute
- Should_ThrowException_When_NodeDoesNotExistDuringGettingAllAttributes
- Should_ReturnDictionaryWithAttributeNamesAndValues_When_NodeHasAtLeastOneAttribute
- Should_ReturnFalse_When_NodeDoesNotExist
- Should_ReturnTrue_When_NodeExists
- Should_ReturnNull_When_DuringExecGetAllDocumentNodesValuesNodeWithTheNameDoesNotExist
- Should_ThrowException_When_GettingStructuredNodesButXQueryIsInvalid
- Should_ThrowKeyNotFoundException_When_GettingStructuredNodesButTypedNodeNameIsInvalid
- Should_ReturnNull_When_AttributeDoesNotExist
- Should_ReturnListOfStrings_When_AttributeExistsAndTheValueIsNotGiven
- Should_ReturnListOfStrings_When_SearchingAttributeWithValueExists
- Should_ReturnDictionaryWithValues_When_GettingStructuredNodes
- Should_ReturnNull_When_NodeWithTheNameDoesNotExistDuringGettingAllDocumentsWithTheName
- Should_ReturnListOfString_When_NodeWithTheNameExists
- Should_ReturnFalse_When_NodeDoesNotExistDuringEditingTextNode
- Should_ChangeTextNodeAndReturnTrue_When_NodeExists
- Should_ThrowException_When_XMLIsNotValid
- Should_ReturnFalse_When_ChangingNameOfTheNodeButNodeDoesNotExist
- Should_ModifyNodeNameAndReturnTrue_When_ChangingNameOfTheNode
- Should_ReturnFalse_When_NodeDoesNotExist
- Should_AddNewNode_When_NodeExists
- Should_ThrowException_When_AttributeDoesNotExist
- Should_ReturnFalse_When_NodeExistsDuringAddingAttributeToNode
- Should_AddAttributeToNodeAndReturnTrue_When_NodeExistsDuringAddingAttributeToNode
- Should_RemoveAttributeFromNode_When_NodeExists

- Should_ReturnFalse_When_NodeDoesNotExistDuringRemovingTheAttribute
- Should_ReturnFalse_When_NodeStructureIsInCorrectDuringDeletingNode
- Should_DeleteNodeFromDocumentAndReturnTrue_When_NodeStructureIsCorrectDuringDeleteNode

Wszystkie z wymienionych testów poprawnie przeszły proces testowania jednostkowego.

Test run finished: 51 Tests (51 Passed, 0 Failed, 0 Skipped) run in 3,2 sec			
Test	Duration	Traits	Error Message
▲ ✓ XMLDocumentApp.Tests (51)	1,3 sec		
▲ ✓ XMLDocumentApp.Tests (51)	1,3 sec		
▸ ✓ QueryBuilderTests (4)	3 ms		
▸ ✓ XMLServiceTests (47)	1,3 sec		

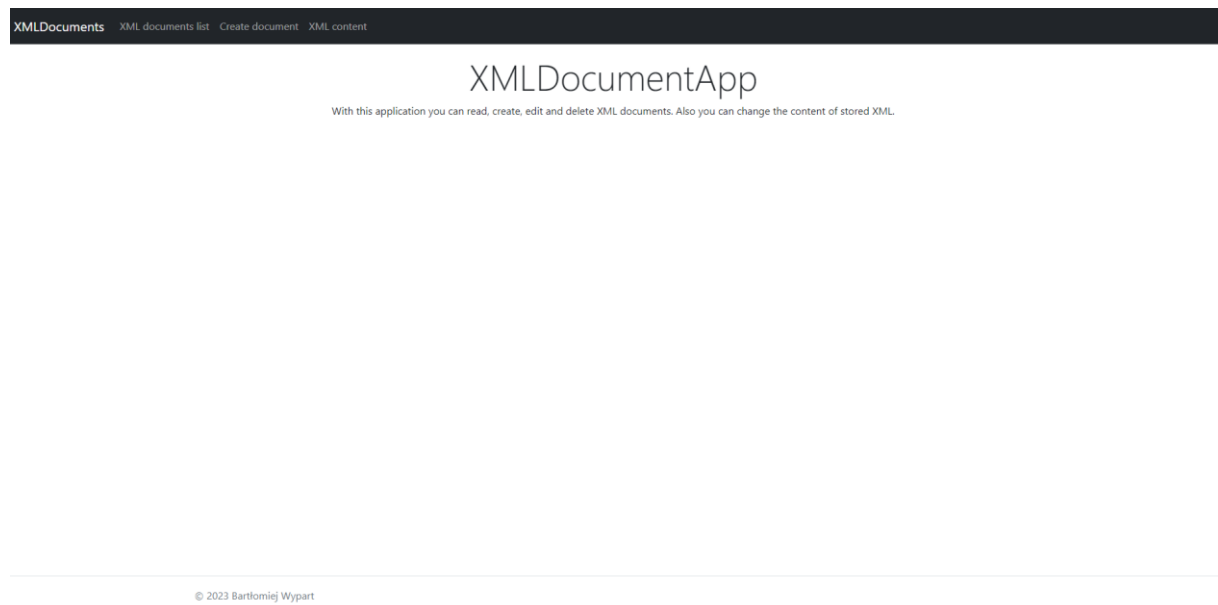
Rysunek 1. Wynik wykonania wszystkich testów jednostkowych.

5. OPIS APLIKACJI

Do API została także stworzona aplikacja, realizująca funkcjonalność dostarczoną w ramach tego API. Jest to aplikacja internetowa typu MPA, zrealizowana za pomocą technologii MVC użyciem platformy .NET 6. Sam serwis został dodany z wykorzystaniem kontenera zależności - *Dependency Injection*. Serwis ten został dodany przez wstrzyknięcie zależności do konstruktora.

5.1 XML DOCUMENTS LIST

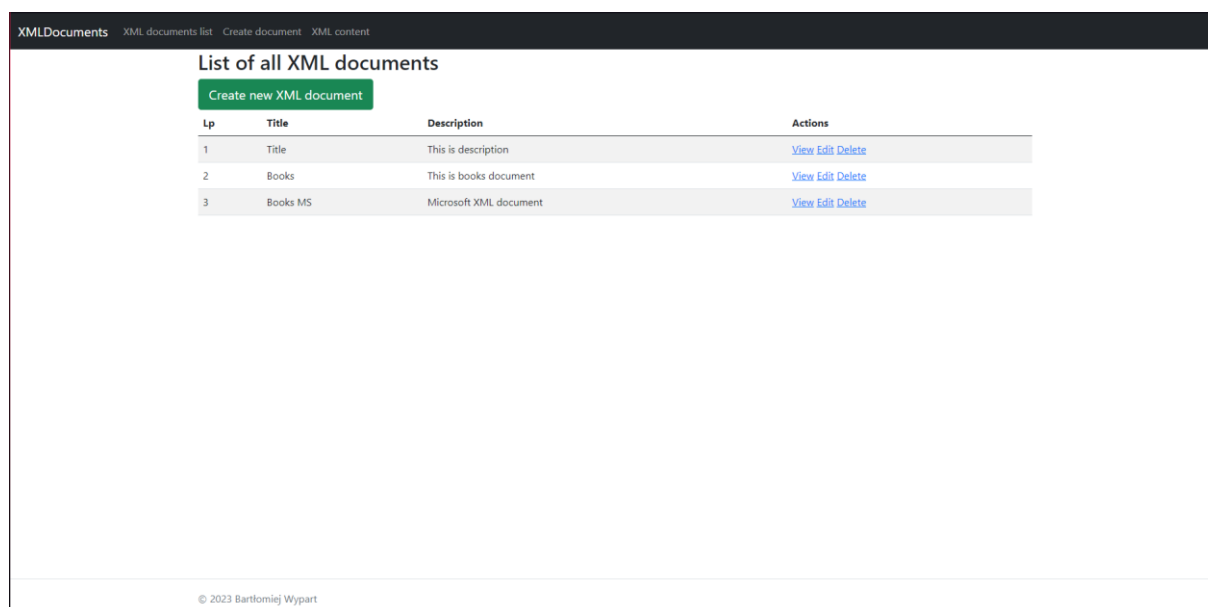
Interfejs użytkownika jest bardzo łatwy w obsłudze. Strona główna aplikacji została przedstawiona na rysunku 2.



Rysunek 2. Strona główna aplikacji internetowej.

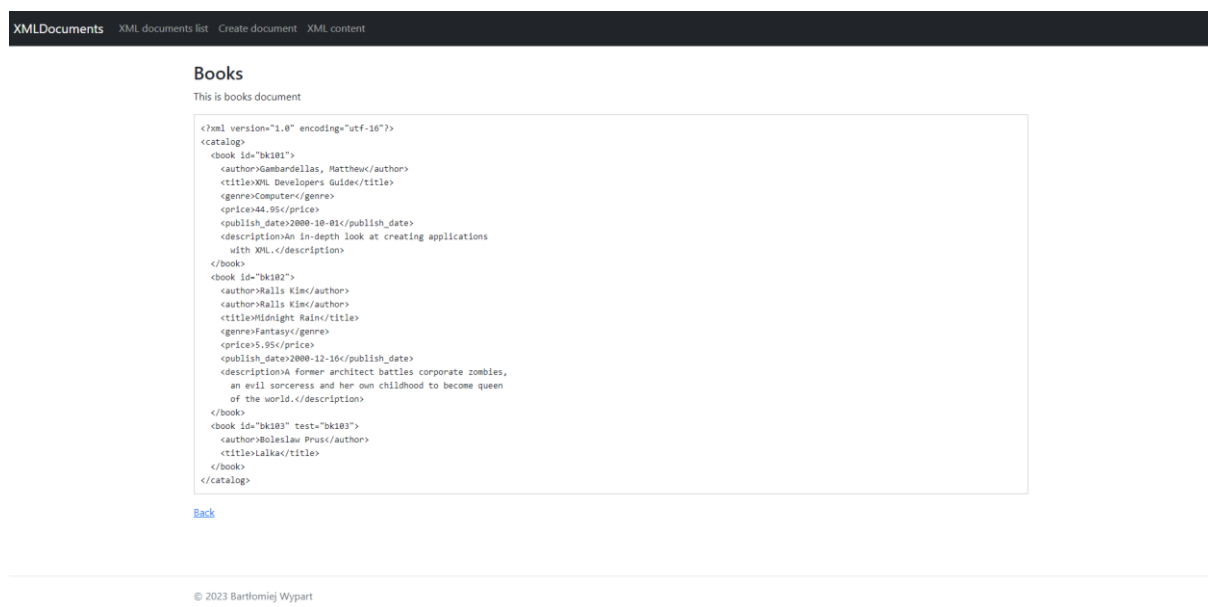
W górnej części znajduje się menu poziome, za pomocą którego użytkownik może wybrać różne opcje.

Pierwszą zakładką jest „XML documents list”, w której to użytkownik jest w stanie dostać listę dostępnych dokumentów XML zapisanych w bazie danych. Również za pomocą tego interfejsu, użytkownik może dodać nowy dokument, podejrzeć, zmodyfikować, a także usunąć wybrany dokument.



Rysunek 3. Widok listy dokumentów XML.

Po kliknięciu przy wybranej pozycji opcji „View”, użytkownik przenoszony jest na stronę podglądu danych:



Rysunek 4. Podgląd parametrów wybranego dokumentu, wraz z zawartością XML.

W przypadku wyboru opcji „Edit”, użytkownik przenoszony jest do formularza, gdzie może zmodyfikować dane parametry. Dane XML są zapisane jako zmienna tekstowa, stąd brak formatowania. Jednak większa swoboda modyfikowania dokumentu znajduje się w innej zakładce, o czym w dalszej części dokumentacji.

XMLDocumentsXML documents listCreate documentXML content

TitleBooks

DescriptionThis is books document

XML document<<catalog><book id="bk101"><author>Gambardellas, Matthew</author><title>XML Developers Guide</title><genre>Computer</genre><price>44.95</price><publish_date>2000-10-01</publish_date><description>An in-depth look at creating applications with XML.</description></book><book id="bk102"><author>Ralls Kim</author><author>Ralls Kim</author><title>Midnight Rain</title><genre>Fantasy</genre><price>5.95</price><publish_date>2000-12-16</publish_date><description>A former architect battles corporate zombies, an evil sorceress and her own childhood to become queen of the world.</description></book><book id="bk103" test="bk103"><author>Boleslaw Prus</author><title>Lalka</title></book></catalog>

SaveBack

© 2023 Bartłomiej Wypart

Rysunek 5. Widok edycji wybranego dokumentu.

Wybierając opcję „Delete”, użytkownik proszony jest o potwierdzenie, że na pewno chce usunąć danych dokument, a następnie w sytuacji, gdy użytkownik zaznaczył opcję potwierdzającą, dokument jest usuwany z bazy.

5.2 CREATE DOCUMENT

Za pomocą tej zakładki, użytkownik może stworzyć nowy dokument. Jest w stanie nadać tytuł (który będzie unikatowy w obrębie bazy, czyli nie będzie się powtarzał), opis, a także wybrać jedną z dwóch opcji: wpisania dokumentu XML ręcznie (przydatne w sytuacji wykorzystania schowka systemowego), oraz wgrania pliku na serwer (przydatne, gdy dokument został zapisany na dysku).

W przypadku, gdy istnieje dokument z tym samym tytułem, zwracany jest błąd. Również to ma miejsce w przypadku, gdy format XML jest niepoprawny.

The screenshot shows a web form titled 'CREATE DOCUMENT'. It contains the following elements:

- Title:** A text input field containing the word 'Books'.
- Description:** A text area containing the text 'Some books'.
- Upload by file?:** A toggle switch that is currently turned off.
- XML document:** A text area containing the XML code '<root />'.
- Choose XML file:** A button labeled 'Wybierz plik' (Choose file) and a disabled button labeled 'Nie wybrano pliku' (No file selected).
- Save:** A blue button with the text 'Save'.
- Back:** A blue link with the text 'Back'.
- Message:** A red text message at the bottom stating 'Document with this title is now in the database'.

Rysunek 6. Komunikat w sytuacji, gdy istnieje już w bazie dokument o podanej nazwie.

Również istnieje walidacja składni XML. W przypadku gdy ta jest niepoprawna, zwracany jest komunikat błędu.

Title

New books

Description

Some books

☐ Upload by file?

XML document

<root> </rooooooot>

Choose XML file

Wybierz plik Nie wybrano pliku

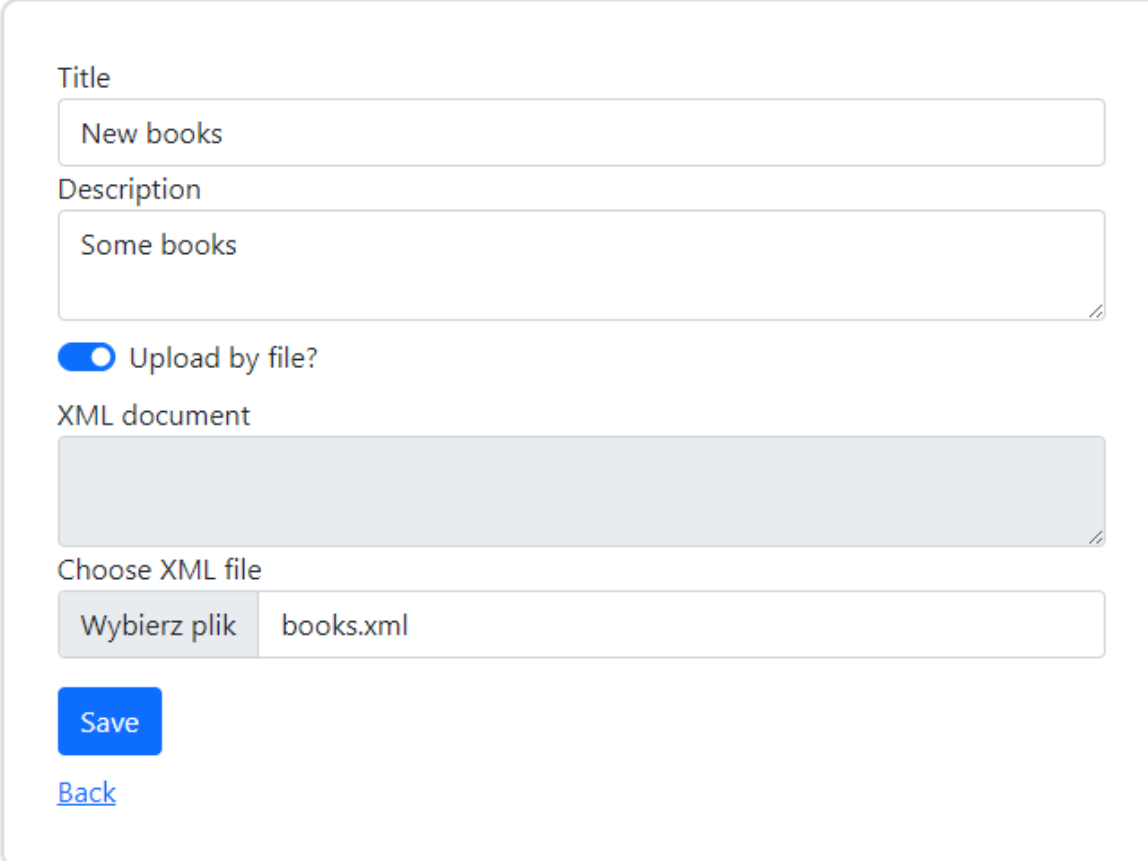
Save

[Back](#)

The 'root' start tag on line 1 position 2 does not match the end tag of 'rooooooot'. Line 1, position 9.

Rysunek 7. Komunikat w sytuacji, gdy format XML jest niepoprawny.

Oczywiście użytkownik może skorzystać z opcji wgrania pliku na serwer, za pomocą wybrania opcji „Upload by file” i wybrania ścieżki do pliku.



The screenshot shows a web form with the following elements:

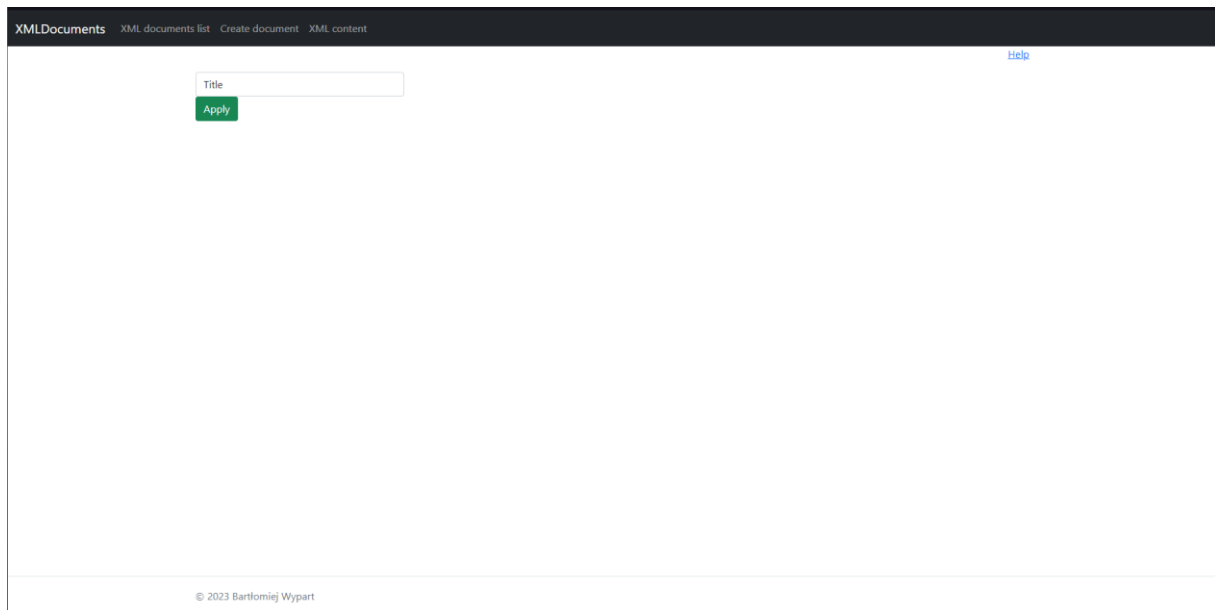
- Title:** A text input field containing "New books".
- Description:** A text input field containing "Some books".
- Upload by file?:** A toggle switch that is turned on (blue).
- XML document:** A large, empty text area for pasting XML content.
- Choose XML file:** A button labeled "Wybierz plik" (Choose file) next to a text input field containing "books.xml".
- Save:** A blue button.
- Back:** A blue link.

Rysunek 8. Widok formularza, w sytuacji, gdy użytkownik wybrał opcję wgrania pliku z dysku lokalnego.

Po poprawnej walidacji oraz dodaniu dokumentu, użytkownik przenoszony jest do listy dokumentów XML.

5.3 XML CONTENT

Właściwa zakładka aplikacji, w której użytkownik może przetwarzać plik XML. Po wejściu w wybraną zakładkę, użytkownik proszony jest o wybór dokumentu.



Rysunek 9. Formularz wyboru dokumentu z bazy.

Po dokonaniu wyboru, widoczny staje się pogląd dokumentu, który zajmuje lewą część strony internetowej. Po środku mamy możliwość wyboru odpowiedniej funkcjonalności, za pomocą której użytkownik będzie chciał przetwarzać zawartość dokumentu XML.

W sytuacji, gdy użytkownik nie jest jeszcze do końca zaznajomiony ze sposobem pisania ścieżki, może użyć okna pomocy, znajdującego się pod przyciskiem „Help”.

Help

×

1. Select title of the book and confirm that.
2. Select a function which you want to apply. (Some of them require additional informations, such as attribut name, value etc.).
3. Write Path to the element that you want to process in convection: `root/child/child`.
If you want to specify the a particular item, use `[]` like: `root/child[1]/child`. Remember that indexing is from 1, not from 0.
4. If you want to start searching not from root element, use `//` at the beginning of path. For example:
`//node/child`
5. You can mix all of this functions and create complex paths!
6. At the end click the "Execute" button, to process XML document.

Got it!

Rysunek 10. Okno modalne, zawierające pomoc, w jaki sposób przetwarzać dokument XML.

Poniżej znajduje się przykład pobrania zawartości węzła, z podanej ścieżki.

Books

Apply

```
<?xml version="1.0" encoding="utf-16"?>
<catalog>
  <book id="bk101">
    <author>Gambardellas, Matthew</author>
    <title>XML Developers Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls Kim</author>
    <author>Ralls Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
    an evil sorceress and her own childhood to become queen
    of the world.</description>
  </book>
  <book id="bk103" test="bk103">
    <author>Adam Mickiewicz</author>
    <title>Lalka</title>
  </book>
</catalog>
```

GetNodeText

catalog/book[3]/title

Execute

GetNodeText ():

catalog/book[3]/title

Lalka

Rysunek 11. Pobrana zawartość tekstowa węzła z podanej ścieżki.

Zwolennicy literatury mogą zauważyć pewną niespójność 😊 W celu edycji danego węzła, można użyć funkcji *EditNodeText()*. Prezentacja funkcjonalności została przedstawiona na rysunku (Rysunek 12).

Books

Apply

```
<?xml version="1.0" encoding="utf-16"?>
<catalog>
  <book id="bk101">
    <author>Gambardellas, Matthew</author>
    <title>XML Developers Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls Kim</author>
    <author>Ralls Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
    an evil sorceress and her own childhood to become queen
    of the world.</description>
  </book>
  <book id="bk103" test="bk103">
    <author>Adam Mickiewicz</author>
    <title>Pan Tadeusz</title>
  </book>
</catalog>
```

EditNodeText

catalog/book[3]/title

Pan Tadeusz

Execute

EditNodeText ():

catalog/book[3]/title

Node text edited successfully

Rysunek 12. Wynik działania funkcji modyfikującej tekst danego węzła.

Funkcjonalność dostarczona w ramach działania aplikacji (wykorzystując przy tym API) jest bardzo bogata w funkcje. Powyżej zostały przedstawione przykłady, jednak oprócz nich dostępne są jeszcze:

- **CheckNodeIfExists** – sprawdza, czy węzeł istnieje o podanej ścieżce.
- **GetNodeText** – wyświetla wartość z węzła.
- **GetNodes** – wyświetla całą zawartość węzła, wraz ze znacznikami XML.
- **GetAllDocumentNodesQueries** – wyświetla listę wszystkich pasujących do ścieżki węzłów, włącznie ze znacznikami XML.
- **GetAllDocumentNodesValues** – wyświetla listę wszystkich pasujących do ścieżki węzłów, ale bez znaczników XML, tylko same wartości.
- **GetAllAttributes** – wyświetla wszystkie nazwy oraz wartości atrybutów z danego węzła.
- **GetValueOfAttribute** – wyświetla wartość dla danego węzła i zadanej nazwy atrybutu.
- **GetNodesWithAttribute** – wyświetla wszystkie węzły, które mają dany atrybut (opcjonalnie z zadaną wartością).
- **GetStructuredNodes** – wyświetla ustrukturyzowane wartości węzłów. Należy podać nazwy węzłów, oddzielając je „,” (przecinkiem).
- **AddNewNode** – dodaje węzeł do ścieżki podanej przez użytkownika.
- **DeleteNode** – usuwa dany węzeł.
- **AddAttributeToNode** – dodaje atrybut wraz z wartością do danego węzła.
- **DeleteAttributeFromNode** – usuwa dany atrybut z węzła.
- **EditNodeText** – modyfikuje wartość tekstową węzła.
- **EditNodeName** – modyfikuje nazwę węzła.

Wszystkie z wyżej wymienionych funkcjonalności korzystają z funkcji dostarczonych w ramach API w postaci obiektu klasy **XMLService**.

6. PODSUMOWANIE

Wszystkie założone funkcjonalności zostały zaimplementowane w API. API oprócz podstawowych metod dostarcza także inne funkcjonalności, jak np. wyszukiwanie dokumentu XML po tytule (jest to bliźniacza metoda do wyszukiwania dokumentu po numerze ID, jednak w niektórych sytuacjach może być to wygodniejsze rozwiązanie).

Funkcjonalność API została zaprezentowana na przykładzie aplikacji internetowej typu MPA. Oczywiście stopień zaawansowania aplikacji jest już zależny od umiejętności programisty i przy użyciu języka JavaScript można stworzyć bardzo interaktywne API (np. w stylu *Drag&Drop*).

7. LITERATURA

- <https://learn.microsoft.com/en-us/sql/relational-databases/xml/xml-data-sql-server?view=sql-server-ver16>
- <https://learn.microsoft.com/en-us/sql/t-sql/xml/xml-data-type-methods?view=sql-server-ver16>
- <https://learn.microsoft.com/pl-pl/aspnet/mvc/overview/getting-started/introduction/getting-started>