

1. Opis problemu i opis funkcjonalności udostępnionej przez API

Typ danych XML jest powszechnie stosowany do przechowywania różnych rodzajów informacji. Przykładem może być plik projektowy w języku C#. W takim pliku łatwo także trzymać na przykład menu restauracji. Dlatego łatwy dostęp do tego typu danych oraz jego modyfikacja okazały się bardzo przydatne, gdy plik jest naprawdę duży.

API, które stworzyłem posiada następujące funkcjonalności:

- Metoda, która sprawdza czy baza danych oraz jej tabele istnieją. Jeśli tak nie jest to tworzy ona bazę i/lub tabele
- Metoda zapisująca w bazie danych dokument XML
- Metoda zwracająca z bazy dokument XML jako napis (string)
- Metoda usuwająca dokument XML z bazy danych
- Metoda aktualizująca nazwę węzła lub tekst w węźle
- Metoda aktualizująca wartość atrybutu w węźle
- Metoda zwracająca odpowiednio sformatowany typ danych (dzięki któremu będzie można modyfikować dane węzły/atributy), który przechowuje dokument XML
- Metoda zwracająca listę wszystkich dokumentów z bazy danych (Id oraz nazwa)
- Metoda szukająca węzłów o podanej nazwie w dokumencie XML
- Metoda szukająca atrybutów o podanej nazwie i wartości w dokumencie XML

Zostało ono przygotowane z pomocą frameworka .NET 7 i języka C# 10.

2. Opis typów danych oraz metod (funkcji) udostępnionych w ramach API

W ramach API udostępniono następujące typy danych:

- **Projekt.XmlService** – klasa implementująca interfejs **IXmlService**. Jest to główna klasa całej biblioteki i zawiera wszystkie funkcjonalności, które możemy wykonać na danych XML. Aby utworzyć obiekt tej klasy, należy przekazać w konstruktorze connection string do bazy danych (wystarczy nazwa serwera bez nazwy bazy. Serwis zawiera metodę do tworzenia bazy, o czym dalej).
- **Projekt.Models.Result** – klasa przechowująca informację o tym czy dana metoda z **Projekt.XmlService** zakończyła się sukcesem (we właściwości **IsSuccess**). Jeśli nie to we właściwości **Error** będzie informacja o błędzie jaki nadarzył się podczas wykonywania metody
- **Projekt.Models.Result<T>** : **Projekt.Models.Result** – generyczna klasa dziedzicząca po typie **Projekt.Models.Result**. Dodatkową właściwością tej klasy jest pole **Content** typu generycznego **T**. Przechowuje ono wartość zwracaną przez metodę z **Projekt.XmlService** jeśli zakończy się ona sukcesem.

- **Projekt.Models.XmlAttributeModel** – klasa przechowująca informację o pojedynczym atrybucie dokumentu XML. Zawiera Id atrybutu, jego nazwę, wartość oraz kolejność (jeśli jest więcej niż jeden atrybut w jednym węźle to trzeba wypisać w odpowiedniej kolejności)
- **Projekt.Models.XmlDocumentModel** – klasa przechowująca informację o pojedynczym dokumencie XML. Zawiera ona Id dokumentu oraz jego nazwę.
- **Projekt.Models.XmlElementModel** – klasa przechowująca informację o pojedynczym węźle lub tekście dokumentu XML. Zawiera ona Id elementu, kolejność jeśli na jednym poziomie jest kilka węzłów, wartość (nazwa węzła lub tekst), typ (specyfikuje czy dany element jest węzłem czy tekstem). Ten typ danych przechowuje także informację o dzieciach danego elementu we właściwości **Children** oraz informację o atrybutach we właściwości **Attributes**.
- **Projekt.Models.XmlEditDocumentModel** – typ danych, który przechowuje dokument XML w rekurencyjny sposób. Zawiera ona Id dokumentu, jego nazwę oraz obiekt klasy **Projekt.Models.XmlElementModel**.

Opis metod typu udostępnionych w klasie **Projekt.XmlService**:

- **Projekt.XmlService.CreateDatabase** – asynchroniczna metoda nieprzyjmująca żadnych argumentów oraz zwracająca obiekt klasy **Projekt.Models.Result**. Korzysta z bloku try-catch w celu przechwycenia wyjątków. Jeśli takowy wystąpi to zwrócony obiekt typu **Projekt.Models.Result** będzie zawierał informację o błędzie. Metoda ta tworzy połączenie z bazą danych dla podanego w konstruktorze klasy connection stringa i wywołuje skrypt SQL przechowywany w stałej **SqlStatements.CreateDatabaseQuery** oraz **SqlStatements.CreateTablesQuery**. Następnie połączenie jest zamykane i metoda zwraca obiekt typu Result z właściwością **IsSuccess** ustawioną na wartość true.
- **Projekt.XmlService.SaveXmlDocument** – asynchroniczna metoda przyjmująca dwa argumenty:
 - **string xml** – dokument XML przechowywany jako jeden napis
 - **string documentName** – nazwa dokumentu

Metoda zwraca obiekt klasy **Result<string>**. We właściwości **Content** będzie przechowywany Id dodanego dokumentu.

Jeśli podczas zapisu wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda najpierw tworzy dokument XML z nazwą podaną jako argument metody a następnie rekurencyjnie dodaje do bazy danych węzły zaczynając od najbardziej zewnętrznego elementu (korzeń – root) oraz tekst jeden po drugim. Dodatkowo w bazie dodawane są atrybuty węzłów.

- **Projekt.XmlService.ReadXmlDocument** – asynchroniczna metoda przyjmująca jeden argument:
 - **Guid documentId** – id dokumentu XML

Metoda zwraca obiekt klasy **Result<string>**. We właściwości **Content** będzie przechowywany dokument XML jako jeden napis.

Jeśli podczas odczytu wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda najpierw sprawdza czy dokument o podanym Id istnieje. Jeśli tak to zaczynając od najbardziej zewnętrznego elementu (korzeń – root) tworzy cały dokument pobierając z bazy rekurencyjnie dzieci danych elementów, potem ich dzieci itd.. Po całej procedurze tworzenia, dokument jest konwertowany na typ string i zwracany w obiekcie klasy **Result<string>**.

- **Projekt.XmlService.GetXmlDocumentForEditing** – asynchroniczna metoda przyjmująca jeden argument:
 - **Guid documentId** – id dokumentu XML

Metoda zwraca obiekt klasy **Result<XmlEditDocumentModel>**. We właściwości **Content** będzie przechowywany obiekt wcześniej opisanego typu **XmlEditDocumentModel**.

Jeśli podczas tworzenia obiektu wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda najpierw sprawdza czy dokument o podanym Id istnieje. Jeśli tak to rekurencyjnie tworzy obiekt typu **XmlEditDocumentModel** dodając do niego coraz bardziej zagnieżdżone węzły (rekurencyjnie).

- **Projekt.XmlService.FindElementsByNodeName** – asynchroniczna metoda przyjmujące dwa argumenty:
 - **Guid documentId** – id dokumentu XML
 - **String nodeName** – nazwa węzła, który użytkownik chce znaleźć w dokumencie

Metoda zwraca obiekt typu **Result<IEnumerable<XmlElementModel>>**. We właściwości **Content** będą przechowywane wszystkie znalezione węzły o podanej przez drugi argument nazwie.

Jeśli podczas tworzenia obiektu wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda najpierw sprawdza czy dokument o podanym Id istnieje. Jeśli tak to najpierw tworzy listę obiektów typu **XmlElementModel**, które mają nazwę węzła podaną przez drugi argument metody. Następnie rekurencyjnie dobudowywane są wewnętrzne węzły (dzieci) tych węzłów.

- **Projekt.XmlService.FindElementsByAttributeNameAndValue** – asynchroniczna metoda przyjmująca 3 argumenty:
 - **Guid documentId** – id dokumentu XML
 - **String attributeName** – nazwa szukanego atrybutu
 - **String attributeValue** – wartość szukanego atrybutu

Metoda zwraca obiekt typu **Result<IEnumerable<XmlElementModel>>**. We właściwości **Content** będą przechowywane wszystkie znalezione węzły, które zawierają atrybut o podanej nazwie i wartości.

Jeśli podczas tworzenia obiektu wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda najpierw sprawdza czy dokument o podanym Id istnieje. Jeśli tak to najpierw tworzy listę obiektów typu **XmlElementModel**, które mają atrybut o podanej nazwie i wartości. Następnie rekurencyjnie dobudowywane są wewnętrzne węzły (dzieci) tych węzłów.

- **Projekt.XmlService.DeleteXmlDocument** – asynchroniczna metoda przyjmująca jeden argument:
 - **Guid documentId** – id dokumentu XML

Metoda zwraca obiekt typu **Result**.

Jeśli podczas usuwania wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda najpierw sprawdza czy dokument o podanym Id istnieje. Jeśli tak to usuwa go z bazy danych razem z jego węzłami oraz atrybutami. Jeśli wszystko się powiedzie, to zwrócony obiekt typu **Result** będzie miał ustawioną właściwość **IsSuccess** na true.

- **Projekt.XmlService.UpdateXmlElement** – asynchroniczna metoda przyjmująca 2 argumenty:
 - **Guid elementId** – id elementu XML
 - **String value** – nazwa węzła lub wartość węzła (jeśli jego wartość to tekst)

Metoda zwraca obiekt typu **Result**.

Jeśli podczas aktualizacji elementu wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda wykonuje parametryzowany skrypt SQL do modyfikacji pojedynczego elementu w bazie na nową wartość. Jeśli zwrócona liczba zmodyfikowanych rekordów wyniesie 0, to metoda zwraca błąd z informacją, że element o podanym id nie istnieje. W przeciwnym razie obiekt **Result** będzie zawierał właściwość **IsSuccess** ustawioną na true.

- **Projekt.XmlService.UpdateXmlAttribute** – asynchroniczna metoda przyjmująca dwa argumenty:
 - **Guid attributeId** – id atrybutu XML
 - **String value** – nowa wartość atrybutu

Metoda zwraca obiekt typu **Result**.

Jeśli podczas aktualizacji atrybutu wystąpi błąd, to we właściwości **Error** będzie przechowywany błąd.

Metoda wykonuje parametryzowany skrypt SQL do modyfikacji wartości pojedynczego atrybutu w bazie. Jeśli zwrócona liczba zmodyfikowanych rekordów wyniesie 0, to metoda zwraca błąd z informacją, że atrybut o podanym id nie istnieje. W przeciwnym razie obiekt **Result** będzie zawierał właściwość **IsSuccess** ustawioną na true.

- **Projekt.XmlService.GetDocuments** – asynchroniczna metoda nieprzyjmująca żadnych argumentów i zwracająca obiekt typu **Result<IEnumerable<XmlDocumentModel>>**. Jeśli wystąpi błąd podczas pobierania dokumentów to we właściwości **Error** będzie przechowywany błąd.

Metoda pobiera wszystkie dokumenty XML z bazy i zwraca dla każdego obiekt typu **XmlDocumentModel**.

3. Prezentacja przeprowadzonych testów jednostkowych

Do testowania biblioteki stworzono 12 testów jednostkowych z wykorzystaniem **xUnit** oraz paczki NuGet **FluentAssertions**:

- Obiekt typu **Result** zawiera błąd przy próbie zapisu dokumentu z pustą nazwą:

```

public async Task SaveDocumentWithInvalidNameReturnsResultWithErrorMessage()
{
    // arrange
    var xml =
        @"<?xml version=""1.0"" encoding=""UTF-8""?>
        <words id='5'></words>";

    // act
    var result = await _xmlService.SaveXmlDocument(xml, documentName: "");

    // assert
    result.IsSuccess.Should().BeFalse();
    result.Error.Should().Be("Document name must not be empty!");
}

```

- Obiekt typu Result zawiera błąd przy próbie zapisu dokumentu XML bez deklaracji wersji i encoding:

```
public async Task SaveDocumentWithInvalidXmlDeclarationReturnsResultWithErrorMessage()
{
    // arrange
    var xml =
        @"
        <words id='5'></words>";

    // act
    var result = await _xmlService.SaveXmlDocument(xml, documentName: "test");

    // assert
    result.IsSuccess.Should().BeFalse();
    result.Error.Should().Be("Check document declaration (Version, Encoding)");
}
```

- Obiekt typu Result zawiera błąd przy próbie zapisu dokumentu XML bez prawidłowej zawartości dokumentu:

```
public async Task SaveDocumentWithInvalidXmlReturnsResultWithErrorMessage()
{
    // arrange
    var xml = "abcdefghijklndf";

    // act
    var result = await _xmlService.SaveXmlDocument(xml, documentName: "test");

    // assert
    result.IsSuccess.Should().BeFalse();
    result.Error.Should().Be("XML string is invalid!");
}
```

- Zapisanie prawidłowego dokumentu XML, a następnie jego pobranie kończy się sukcesem:

```
public async Task SaveDocumentWithValidXmlReturnsResultWithCorrectId()
{
    // arrange
    var xml =
        @"<?xml version='1.0' encoding='UTF-8'>
        <words id='5'></words>";

    // act
    var saveResult = await _xmlService.SaveXmlDocument(xml, documentName: "test");
    var getResult = await _xmlService.GetDocuments();

    // assert
    saveResult.IsSuccess.Should().BeTrue();
    getResult.IsSuccess.Should().BeTrue();
    saveResult.Content.Should().BeOneOf(validValues: getResult.Content!.Select(a => a.Id.ToString()));
}
```

- Zapis dokumentu a następnie jego odczyt zwraca taką samą zawartość XML:

```
public async Task ReadXmlDocumentWithCorrectIdReturnsSameXmlAsPassed()
{
    // arrange
    var xml =
        @"<?xml version=""1.0"" encoding=""UTF-8""><words id=""5"">123</words>";

    // act
    var saveResult = await _xmlService.SaveXmlDocument(xml, documentName: "test");
    var readResult = await _xmlService.ReadXmlDocument(Guid.Parse(saveResult.Content!));

    // assert
    readResult.IsSuccess.Should().BeTrue();
    readResult.Content!.Replace(oldValue: "\n", newValue: "").Replace(oldValue: "\r", newValue: "").Should().BeEquivalentTo(xml);
}
```

- Próba odczytu dokumentu o niepoprawnym id kończy się błędem:

```
public async Task ReadXmlDocumentWithInvalidIdReturnsResultWithError()
{
    // act
    var readResult = await _xmlService.ReadXmlDocument(Guid.Empty);

    // assert
    readResult.IsSuccess.Should().BeFalse();
    readResult.Error.Should().Be("Document with given Id does not exist!");
}
```

- Usunięcie dokumentu o poprawnym id kończy się sukcesem:

```
public async Task DeleteXmlDocumentWithValidIdReturnSuccess()
{
    // arrange
    var xml =
        @"<?xml version=""1.0"" encoding=""UTF-8""><words id=""5"">123</words>";

    // act
    var saveResult = await _xmlService.SaveXmlDocument(xml, documentName: "test");
    var deleteResult = await _xmlService.DeleteXmlDocument(Guid.Parse(saveResult.Content!));

    // assert
    delete.IsSuccess.Should().BeTrue();
}
```

- Usunięcie dokumentu o niepoprawnym id kończy się błędem:

```
public async Task DeleteXmlDocumentWithNotExistingIdReturnsFailure()
{
    // act
    var deleteResult = await _xmlService.DeleteXmlDocument(Guid.Empty);

    // assert
    delete.IsSuccess.Should().BeFalse();
    delete.Error.Should().Be("Document with this Id does not exist!");
}
```

- Edycja istniejącego atrybutu modyfikuje atrybut poprawnie i kończy się sukcesem:

```
public async Task EditXmlAttributeWithCorrectIdReturnsSuccess()
{
    // arrange
    var xml =
        @"<?xml version=""1.0"" encoding=""UTF-8""?><words id=""5"">123</words>";

    // act
    var saveResult = await _xmlService.SaveXmlDocument(xml, documentName: "test");
    var xmlEditDocumentModel = (await _xmlService.GetXmlDocumentModelForEditing(Guid.Parse(saveResult.Content!))).Content;
    var xmlElements :IEnumerable<XmlElementModel> = GetFlattenedElements(xmlEditDocumentModel!);
    var attribute = xmlElements // IEnumerable<XmlElementModel>
        .First(e :XmlElementModel => e.Attributes.Any(a :XmlAttributeModel => a.Value == "5")) // XmlElementModel
        .Attributes // List<XmlAttributeModel>
        .First(a :XmlAttributeModel => a.Value == "5");
    var editResult = await _xmlService.UpdateXmlAttribute(attribute.Id, value: "10");
    var readResult = await _xmlService.ReadXmlDocument(Guid.Parse(saveResult.Content!));

    // assert
    editResult.IsSuccess.Should().BeTrue();
    readResult.IsSuccess.Should().BeTrue();
    readResult.Content.Should().Contain(@"id=""10""");
}
```

- Edycja wartości węzła modyfikuje wartość poprawnie i zwraca sukces:

```
public async Task EditXmlElementWithCorrectIdReturnsSuccess()
{
    // arrange
    var xml =
        @"<?xml version=""1.0"" encoding=""UTF-8""?><words id=""5"">123</words>";

    // act
    var saveResult = await _xmlService.SaveXmlDocument(xml, documentName: "test");
    var xmlEditDocumentModel = (await _xmlService.GetXmlDocumentModelForEditing(Guid.Parse(saveResult.Content!))).Content;
    var xmlElements :IEnumerable<XmlElementModel> = GetFlattenedElements(xmlEditDocumentModel!);
    var element = xmlElements.First(e :XmlElementModel => e.Value == "123");
    var editResult = await _xmlService.UpdateXmlElement(element.Id, value: "10");
    var readResult = await _xmlService.ReadXmlDocument(Guid.Parse(saveResult.Content!));

    // assert
    editResult.IsSuccess.Should().BeTrue();
    readResult.IsSuccess.Should().BeTrue();
    readResult.Content.Should().Contain("10");
    readResult.Content.Should().NotContain(unexpected: "123");
}
```

- Znalezienie węzłów z podanym atrybutem zwraca odpowiednią ilość węzłów i kończy się sukcesem:

```
public async Task FindElementsByAttributeNameAndValueReturnCorrectResult()
{
    // arrange
    var xml =
        @"<?xml version=""1.0"" encoding=""UTF-8""?><words id=""5""><word d=""4"">dsc</word><word d=""4"">fsdfs</word></words>";

    // act
    var saveResult = await _xmlService.SaveXmlDocument(xml, documentName: "test");
    var findResult = await _xmlService.FindElementsByAttributeNameAndValue(documentId: Guid.Parse(saveResult.Content!), attributeValue: "4");

    // assert
    findResult.IsSuccess.Should().BeTrue();
    findResult.Content!.ToList().Count.Should().Be(2);
}
```

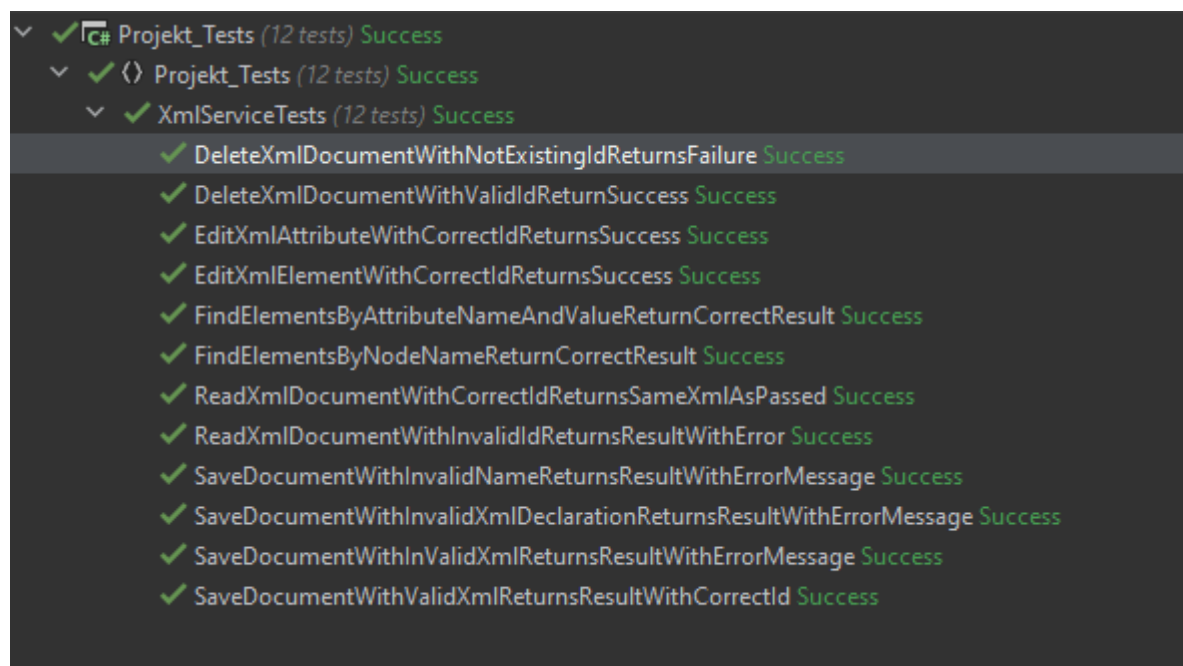

- Znalezienie węzłów o podanej nazwie węzła zwraca odpowiednią ilość węzłów i kończy się sukcesem:

```
public async Task FindElementsByNodeNameReturnCorrectResult()
{
    // arrange
    var xml =
        @"<?xml version=""1.0"" encoding=""UTF-8""?><words id=""5""><word d=""4"">dsc</word><word d=""4"">fsdfs</word></words>";

    // act
    var saveResult = await _xmlService.SaveXmlDocument(xml, documentName: "test");
    var findResult = await _xmlService.FindElementsByNodeName( documentId: Guid.Parse(saveResult.Content!), nodeName: "word");

    // assert
    findResult.IsSuccess.Should().BeTrue();
    findResult.Content!.ToList().Count.Should().Be(2);
}
```

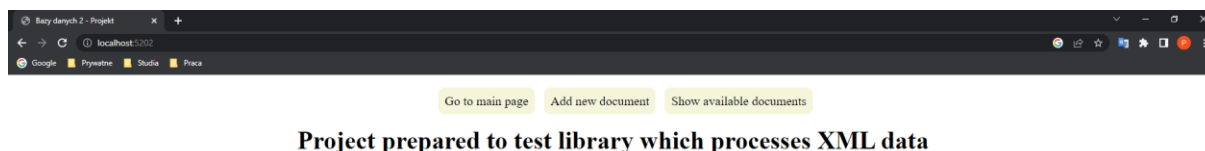
Jak widać na poniższym obrazku, wszystkie testy zakończyły się powodzeniem:



4. Prezentacja przykładowej aplikacji

Aplikację testującą przygotowaną bibliotekę stworzono przy użyciu ASP.NET Core MVC w wersji .NET 7. Jeśli użytkownik nie posiada .NET 7 na swoim urządzeniu to musi się udać na stronę <https://dotnet.microsoft.com/en-us/download/visual-studio-sdks> i ją zainstalować i następnie sprawdzić w konsoli poleceniem `dotnet --version` czy się zainstalowało. Aby uruchomić aplikację należy wejść do folderu `Projekt_Application` i zmienić connection string na swój serwer w pliku `appsettings.json`. Następnie w konsoli wpisać polecenie `dotnet run`.

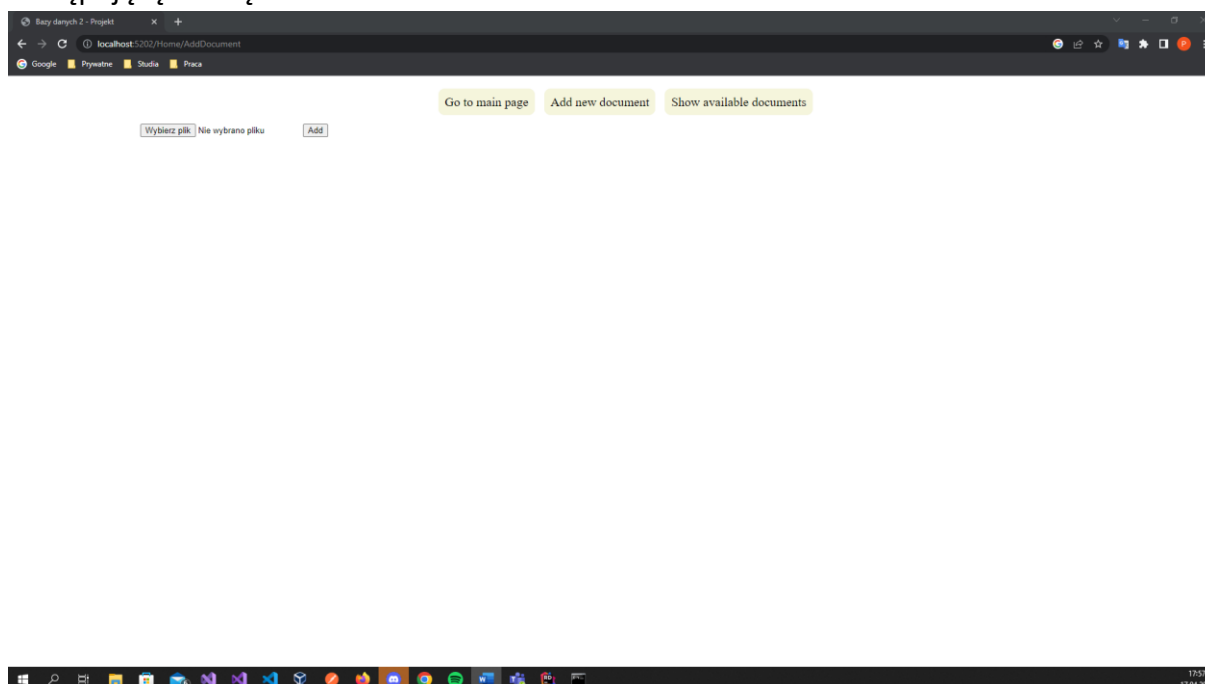
Po uruchomieniu aplikacji wejść pod URL: <http://localhost:5202/> :



Obraz 1: Strona startowa aplikacji testującej

Kliknięcie w guzik „Go to main page” powoduje przeniesienie na właśnie tę stronę.

Po kliknięciu w guzik „Add new document” użytkownik zostanie przeniesiony na następującą stronę:



Obraz 2: Strona do dodania dokumentu XML

Po wybraniu pliku i kliknięciu guzika Add, dokument powinien się dodać do bazy danych jeśli był poprawny. Użytkownik zostanie wtedy automatycznie przeniesiony na stronę znajdującą się pod guzikiem „Show available documents”. Dla celów testowych utworzono następujący dokument XML o nazwie file.xml:

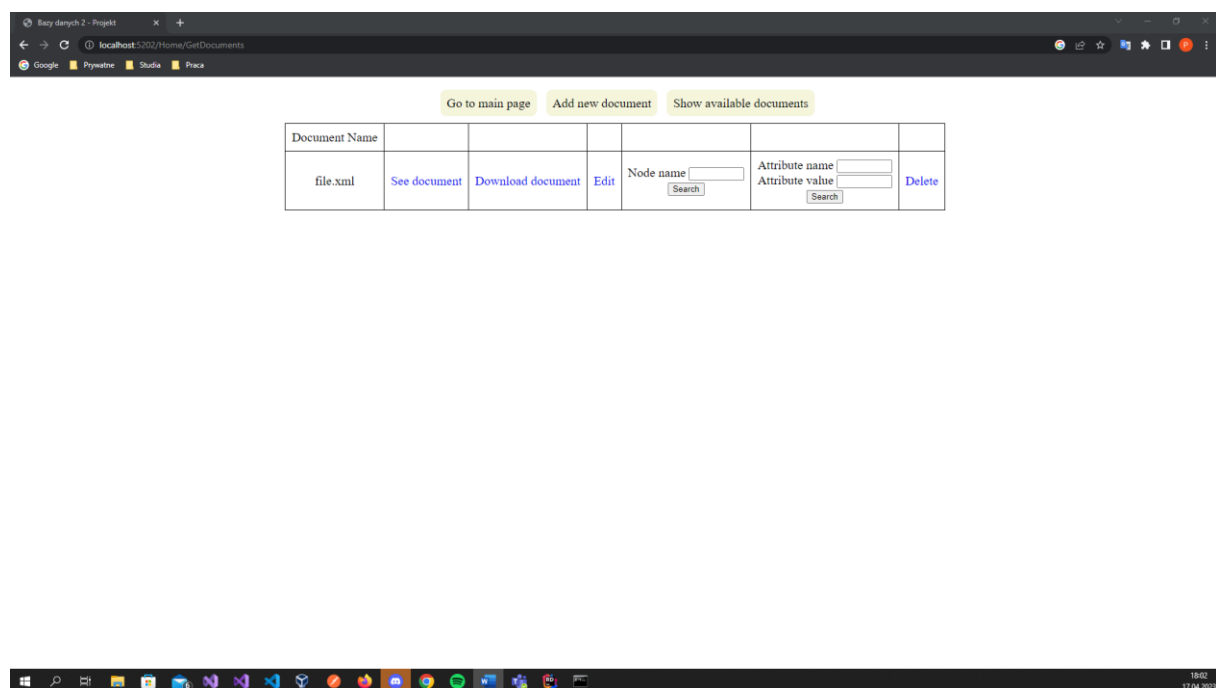
```

<?xml version="1.0" encoding="utf-8"?>
<words id="1">
  <word word_id="1" id="1">
    <l>aa</l>
    <l>bb</l>
  </word>
  <word key="test">sky</word>
  <word word_id="3">bottom</word>
  <word>cup</word>
  <word>book</word>
  <word>rock</word>
  <word>sand</word>
  <word>river</word>
</words>

```

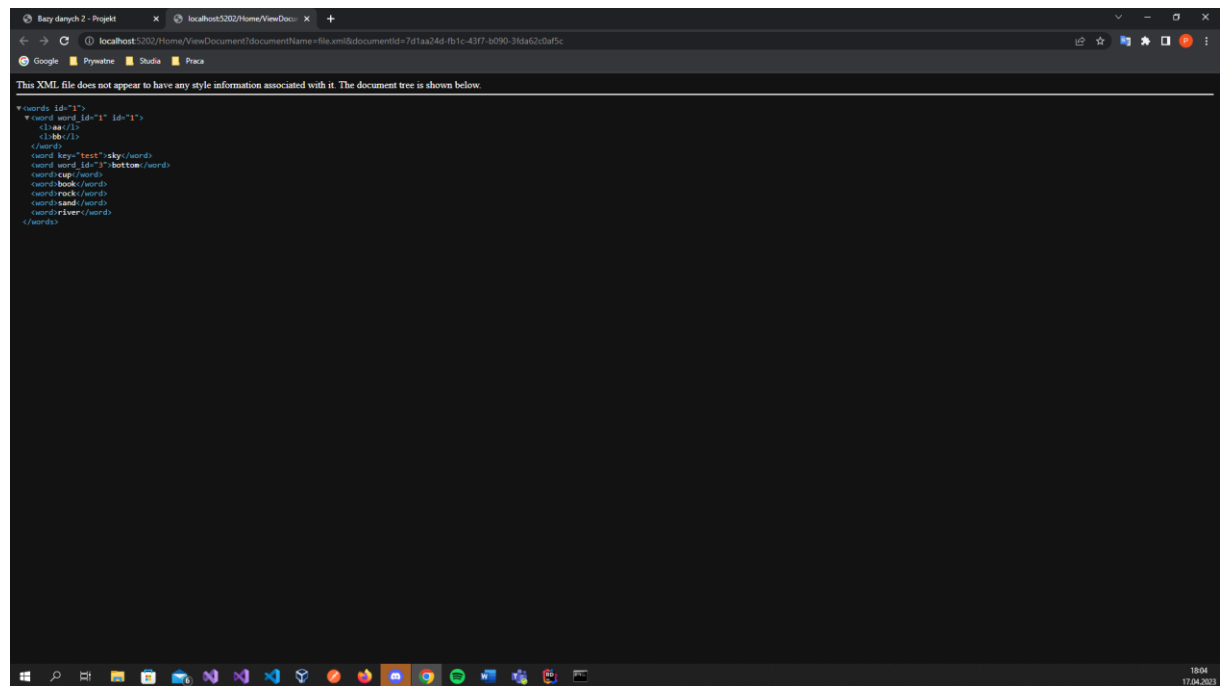
Obraz 3: Przykładowy dokument XML dodany do bazy danych

Po dodaniu dokumentu i przeniesieniu na stronę z listą dokumentów widok jest następujący:




Obraz 4: Widok strony po pomyślnym dodaniu dokumentu XML

W tym miejscu użytkownik dla każdego dokumentu ma do wybrania kilka opcji. Pierwszą z nich jest kliknięcie w przycisk „See document”. Po kliknięciu w niego w osobnej karcie przeglądarki otworzy się zawartość dokumentu XML:



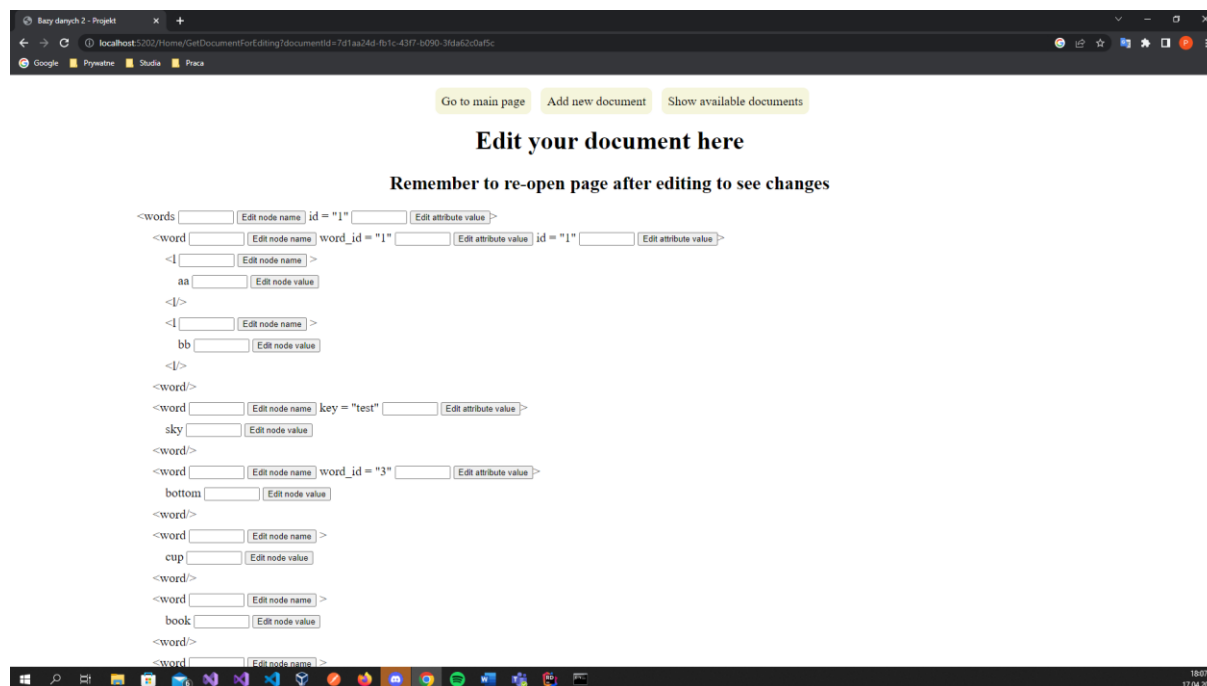
Obraz 5: Widok po kliknięciu w przycisk „See document”

Drugą opcją jest przycisk „Download document”, który pobiera plik na dysk użytkownika. Po kliknięciu w folderze pobrane pojawił się dokument:

Nazwa	Data modyfikacji	Typ	Rozmiar
▼ Dzisiaj (2)			
 file.xml	17.04.2023 18:05	Dokument XML	1 KB

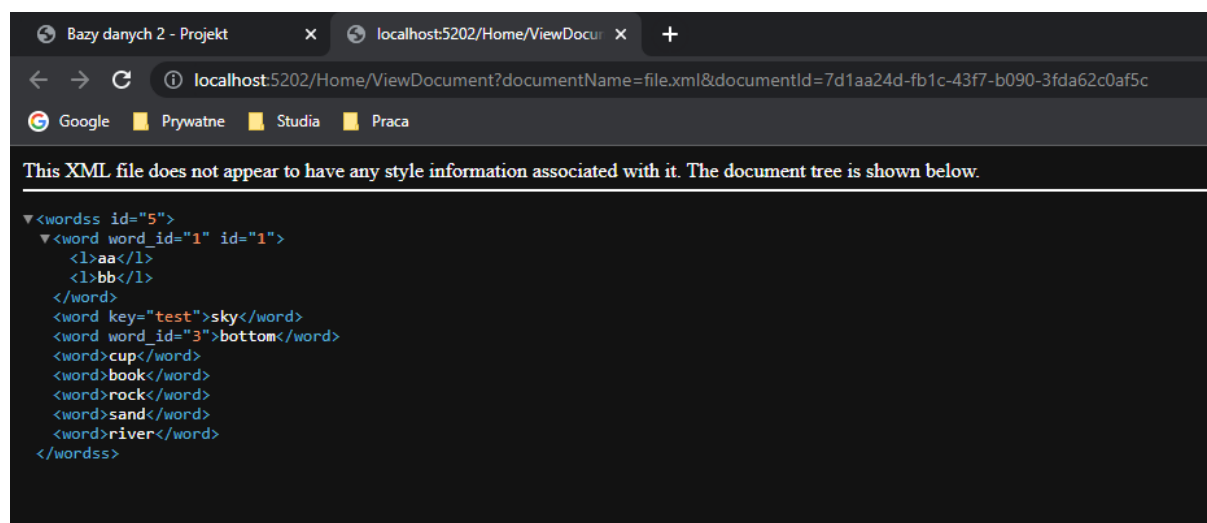
Obraz 6: Widok folderu Pobrane po pobraniu dokumentu ze strony

Trzecia opcja „Edit” to możliwość modyfikacji dokumentu XML. Po kliknięciu użytkownik zostanie przeniesiony na stronę edycji:



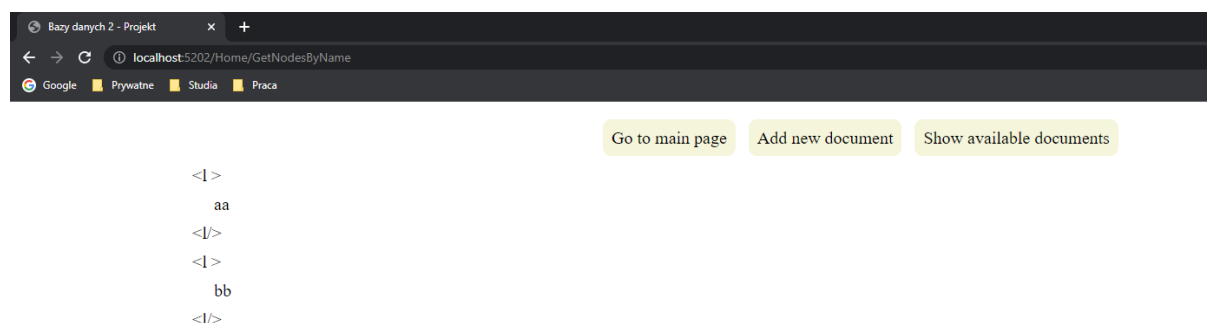
Obraz 7: Widok strony do edycji dokumentu XML

Tutaj użytkownik może modyfikować poszczególne węzły (ich nazwa lub wartość) oraz wartości atrybutów. Po modyfikacji każdej wartości należy kliknąć przycisk „Edit node name” lub „Edit attribute value” lub „Edit node value” (znajdujący się obok modyfikowanej wartości) w zależności od modyfikowanej wartości. W ramach przykładu zmodyfikowano nazwę węzła korzenia (root’a) dokumentu na „wordss” oraz jego atrybut „id” na wartość 5 i zapisano zmiany. Po zapisaniu zmian i wyświetleniu dokumentu widok jest następujący:



Obraz 8: Widok dokumentu po modyfikacji nazwy węzła oraz jego atrybutu

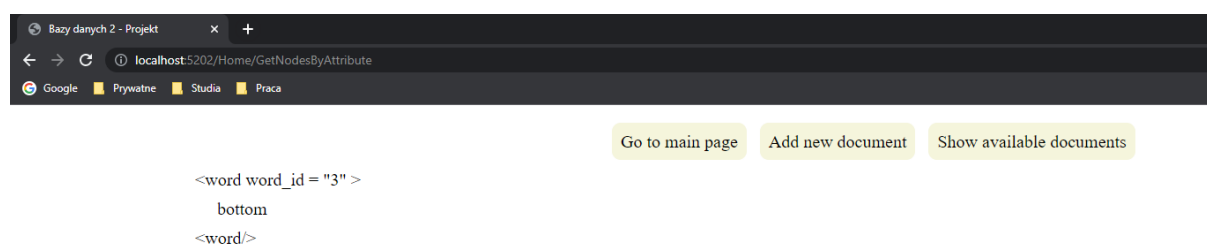
Czwartą opcją jest wyszukanie w dokumencie węzłów o podanej nazwie. W aplikacji przykładowo wyszukano węzeł o nazwie „l” i kliknięto przycisk „Search” znajdujące się pod miejscem na wpisanie nazwy węzła. Wynik jest następujący:



Obraz 9: Widok po wyszukaniu węzła po nazwie

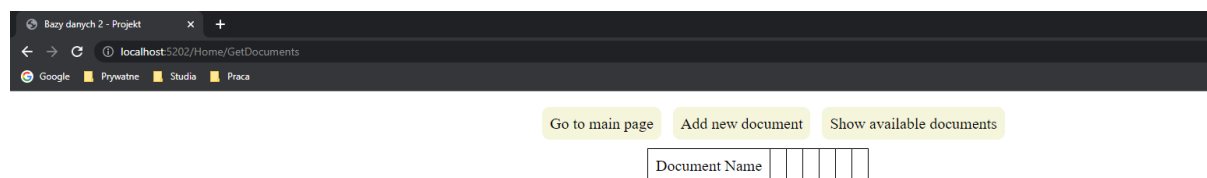
Jak widać, wynik jest poprawny.

Piąta opcja to wyszukanie węzłów po nazwie i wartości atrybutu. W aplikacji przykładowo wpisano w miejsce „Attribute name” wartość word_id, a w miejsce „Attribute value” wartość 3. Wynik po kliknięciu w przycisk Search jest następujący:



Obraz 10: Widok po wyszukaniu węzła po nazwie i wartości atrybutu

Ostatnią opcją jest usunięcie dokumentu. Po kliknięciu tego guzika wynik jest następujący:



Obraz 11: Widok po usunięciu dokumentu

Jak widać dokument poprawnie został usunięty z bazy danych.

5. Podsumowanie

W utworzonej bibliotece udało się osiągnąć wszystkie wyznaczone cele. Aplikacja webowa pomyślnie przetestowała wszystkie metody dostępne w bibliotece. Testy jednostkowe także okazały się pomyślne. W projekcie wykorzystano bazę danych SQL Server 2019, a łączenie z bazą zostało wykonane z pomocą ADO.NET. W dodatku skorzystano z paczki NuGet o nazwie Dapper do łatwiejszego wykonywania zapytań do bazy (mapowanie na obiekty przy wykonywaniu zapytań SELECT oraz łatwo parametryzowane zapytania). W bazie danych znajdują się 3 tabele:

- XmlDocument
- XmlElement
- XmlAttribute

Odpowiadają one klasom zadeklarowanym w języku C#, które wykorzystuje biblioteka.

Największą trudnością w projekcie było rekurencyjne przetworzenie całego dokumentu w obie strony, aby dobrze go zwrócić do użytkownika biblioteki ale ostatecznie wszystko działa poprawnie.

6. Literatura

W bibliotece wykorzystano przykłady z następujących stron:

- <https://learn.microsoft.com/en-us/dotnet/csharp/>
- <https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16>

7. Kod źródłowy

Kod źródłowy całej biblioteki, skryptów SQL, projektu testów jednostkowych i aplikacji webowej znajduje się na stronie GitHub pod linkiem: <https://github.com/Piotrreek/BD2>