Przetwarzanie danych hierarchicznych w XML - Dokumentacja projektu.

Bartosz Konopka

May 2024

1 Opis problemu i opis funkcjonalności udostępnianej przez API

Celem projektu jest opracowanie biblioteki umożliwiającej zarządzanie danymi hierarchicznymi, a dokładniej drzewami genealogicznymi. Dane takie charakteryzują się hierarchicznym układem, gdzie każdy węzeł może mieć wiele podwęzłów, tworząc drzewo. Dostęp do danych powinien być łatwy, w celu zapewnienia wiuzalizacji danych, oraz manipulacji nimi.

API przedstawione w projekcie udostępnia funkcjonalności obejmujące: Tworzenie drzewa genealogicznego, dodawanie węzłów (osób) do drzewa, usuwanie ich oraz wizualizację odpowiednich raportów.

API wykorzystuje typ *XML* dostępny w *SQL Server* do przechowywania i manipulacji danymi hierarchicznymi. Interfejs *ADO.NET* jest używany do komunikacji z bazą danych. Aplikacja pokazująca działanie *API* napisana została w języku *JAVA*.

2 Opis typów danych oraz metod udostępnionych w ramach API

Dane przetrzymywane są w bazie danych o nazwie *HXML*, w tabeli o nazwie *DrzewoGenealogiczne*. Sktruktura tabeli prezentuje się następująco:
W pierwszym elemencie tabeli znajduje się liczba całkowita odpowiadająca *ID* drzewa genalogicznego. Drugim elementem tabeli jest dokument *XML* przetrzymujący informację o *NODach*, czyli osobach, znajdujących się w drzewie genealogicznym. Dokument *XML* zawiera osoby należące do drzewa, a każda osoba przetrzymuje następujące informacje:

- **Id osoby** - unikalna wartość reprezentująca konkretną osobę.
- **Imię** osoby.
- **Nazwisko** osoby.
- **Płeć** osoby.
- **ID Ojca** - Id innej osoby z drzewa genealogicznego, osoba ta reprezentuje ojca opisywanej osoby. Wartość ta może być *nullem*, w takim wypadku osoba ta jest pierwszym wierzchołkiem drzewa.
- **ID Matki** - Id innej osoby z drzewa genealogicznego, osoba ta reprezentuje matkę opisywanej osoby. Wartość ta może być *nullem*, w takim wypadku osoba ta jest pierwszym wierzchołkiem drzewa.
- **Data urodzin**.
- **Data śmierci**. W przypadku gdy wartość ta jest *nullem*, opisywana osoba dalej żyje.

API udostępnia następujące metody:

- **Integer addTree(Connection connection);** - metoda tworząca nowe drzewo genealogiczne. Przyjmuje ona połączenie do bazy danych, a zwraca ID utworzonego drzewa genealogicznego.
- **ResultSet report(Connection connection, Integer tree_id, Integer node_id);** - metoda generująca raport o konkretnym drzewie genealogicznym. Przyjmuje ona połączenie z bazą danych oraz ID drzewa genealogicznego z którego generujemy raport, oraz ID osoby; jeżeli wartość ta jest równa 0, generowany jest raport nr 1: wypisanie wszystkich osób z drzewa. Gdy wartość *node_id* jest inna niż 0, interpretowana jest jako ID osoby i generowany jest raport nr 2: wypisanie rodziców konkretnej osoby. Metoda ta zwraca tabelę otrzymaną z $SELECTa$.
- **boolean addNode(Connection connection, Integer tree_id, String person_id, String name, String surname, String gender, String dads_id, String moms_id, String birth, String death);** - metoda dodająca wierzchołek (osobę) do drzewa genealogicznego. Metoda ta przyjmuje połączenie z bazą danych oraz wszystkie potrzebne informacje do utworzenia nowej osoby w drzewie genealogicznym. Metoda ta zwraca *true*, jeżeli dodanie osoby się powiodło, lub $false$, gdy dodanie osoby nie powiodło się.
- **boolean deleteNode(Connection connection, Integer tree_id, Integer node_id);** - metoda usuwająca wierzchołek (osobę) z drzewa genealogicznego. Przyjmuje ona połączenie z bazą danych, ID drzewa z którego usuwamy osobę, oraz ID osoby którą usuwamy. Metoda ta zwraca wartość *true* gdy usunięcie osoby powiodło się, lub $false$, gdy nie udało się usunąć osoby.

3 Opis implementacji udostępnionego API przez bibliotekę

Biblioteka napisana została w jeżyku $JAVA$ i wymaga otrzymania połączenia z bazą danych, przechowującą tabelę o struturze opisanej wcześniej.

Utworzenie bazy skryptem SQL prezentuje się następująco:

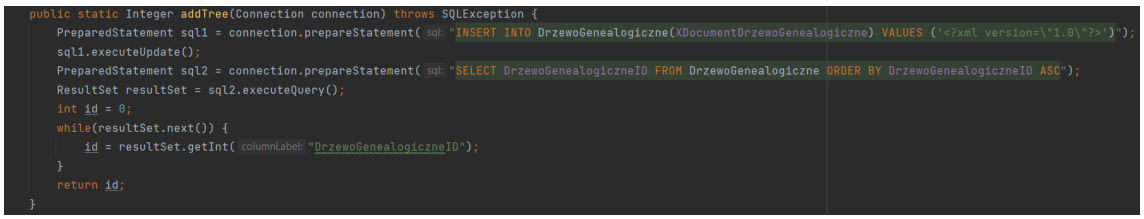
```
CREATE DATABASE [HXML]
ON
(NAME = N'HXML',
    FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\HXML.mdf',
    SIZE = 10,
    MAXSIZE = 20,
    FILEGROWTH = 2)
LOG ON
(NAME = N'HXML_Log',
    FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\HXML_Log.ldf',
    SIZE = 5,
    MAXSIZE = 20,
    FILEGROWTH = 2);
```

Utworzenie tabeli skryptem SQL prezentuje się następująco:

```
CREATE TABLE DrzewoGenealogiczne(
    DrzewoGenealogiczneID int IDENTITY(1,1),
    XDocumentDrzewoGenealogiczne xml,
    CONSTRAINT PK_DrzewoGenealogiczne PRIMARY KEY (DrzewoGenealogiczneID)
);
```

Opis implementacji udostępnionego *API*:

- **Dodawanie drzewa genealogicznego**



Metoda używa polecenia *INSERT* do dodania nowego elementu tabeli *DrzewoGenealogiczne*. Nastepnie poleceniem *SELECT* wybierane zostają *ID* drzew genealogicznych i zwracana jest wartość ostanio dodanego drzewa.

- **Generowanie raportów**

```
public static ResultSet report(Connection connection, Integer tree_id, Integer node_id) throws SQLException {
    PreparedStatement sql;
    if(node_id == 0) {
        sql = connection.prepareStatement( sql: "SELECT \n" +
                "\tTbl.Col.value('(Osoba_id)[1]','varchar(20)') AS ID, \n" +
                "\tTbl.Col.value('(Imie)[1]', 'varchar(20)') AS Imie, \n" +
                "\tTbl.Col.value('(Nazwisko)[1]', 'varchar(20)') AS Nazwisko,\n" +
                "\tTbl.Col.value('(Plec)[1]', 'varchar(20)') AS Plec,\n" +
                "\tTbl.Col.value('(Ojciec_id)[1]', 'varchar(20)') AS Ojciec_id,\n" +
                "\tTbl.Col.value('(Matka_id)[1]', 'varchar(20)') AS Matka_id,\n" +
                "\tTbl.Col.value('(Urodziny)[1]', 'varchar(20)') AS Urodziny,\n" +
                "\tTbl.Col.value('(Smierc)[1]', 'varchar(20)') AS Smierc\n" +
                "FROM \n" +
                "    DrzewoGenealogiczne\n" +
                "CROSS APPLY \n" +
                "    XDocumentDrzewoGenealogiczne.nodes('Osoba') AS Tbl(Col)\n" +
                "WHERE \n" +
                "    DrzewoGenealogiczneID = " + tree_id);
    }
```

```
    else {
        sql = connection.prepareStatement( sql: "WITH ParentIDs AS (\n" +
                "    SELECT\n" +
                "        T.O.value('(Ojciec_id/text())[1]', 'NVARCHAR(50)') AS Ojciec_id,\n" +
                "        T.O.value('(Matka_id/text())[1]', 'NVARCHAR(50)') AS Matka_id\n" +
                "    FROM DrzewoGenealogiczne\n" +
                "    CROSS APPLY XDocumentDrzewoGenealogiczne.nodes('Osoba[Osoba_id=" + node_id + "]') T(O)\n" +
                "\tWHERE DrzewoGenealogiczneID = " + tree_id + "\n" +
                ")\n" +
                "SELECT \n" +
                "    P.value('(Osoba_id/text())[1]', 'NVARCHAR(50)') AS Osoba_id,\n" +
                "    P.value('(Imie/text())[1]', 'NVARCHAR(50)') AS Imie,\n" +
                "    P.value('(Nazwisko/text())[1]', 'NVARCHAR(50)') AS Nazwisko,\n" +
                "    P.value('(Plec/text())[1]', 'NVARCHAR(20)') AS Plec,\n" +
                "    P.value('(Ojciec_id/text())[1]', 'NVARCHAR(50)') AS Ojciec_id,\n" +
                "    P.value('(Matka_id/text())[1]', 'NVARCHAR(50)') AS Matka_id,\n" +
                "    P.value('(Urodziny/text())[1]', 'NVARCHAR(50)') AS Urodziny,\n" +
                "    P.value('(Smierc/text())[1]', 'NVARCHAR(50)') AS Smierc\n" +
                "FROM DrzewoGenealogiczne\n" +
                "CROSS APPLY XDocumentDrzewoGenealogiczne.nodes('Osoba') T(P)\n" +
                "JOIN ParentIDs PI ON T.P.value('(Osoba_id/text())[1]', 'NVARCHAR(50)') = PI.Ojciec_id\n" +
                "UNION ALL\n" +
                "SELECT \n" +
```

```
                "SELECT \n" +
                "    P.value('(Osoba_id/text())[1]', 'NVARCHAR(50)') AS Osoba_id,\n" +
                "    P.value('(Imie/text())[1]', 'NVARCHAR(50)') AS Imie,\n" +
                "    P.value('(Nazwisko/text())[1]', 'NVARCHAR(50)') AS Nazwisko,\n" +
                "    P.value('(Plec/text())[1]', 'NVARCHAR(20)') AS Plec,\n" +
                "    P.value('(Ojciec_id/text())[1]', 'NVARCHAR(50)') AS Ojciec_id,\n" +
                "    P.value('(Matka_id/text())[1]', 'NVARCHAR(50)') AS Matka_id,\n" +
                "    P.value('(Urodziny/text())[1]', 'NVARCHAR(50)') AS Urodziny,\n" +
                "    P.value('(Smierc/text())[1]', 'NVARCHAR(50)') AS Smierc\n" +
                "FROM DrzewoGenealogiczne\n" +
                "CROSS APPLY XDocumentDrzewoGenealogiczne.nodes('Osoba') T(P)\n" +
                "JOIN ParentIDs PI ON T.P.value('(Osoba_id/text())[1]', 'NVARCHAR(50)') = PI.Matka_id;");
    }
    return sql.executeQuery();
}
```

W przypadku gdy wartość pola *node_id* równa jest zero generowany jest pierwszy raport. Przy pomocy funkcji *nodes* wyowałnej na $XQuery$ wybranego drzewa genealogicznego, wartości $XMLa$ rodzielane są na $NODy$ czyli poszczególne osoby. Następnie przy pomocy funkcji *value*, informacje o osobach zostają parsowane i przetrzymywane w tabeli.
Gdy wartość *node_id* jest różna od 0, interpretowana jest ona jako ID osoby. Na podstawie tego ID wybierani są ojciec i matka osoby o podanym ID oraz wyświetlane są informacje o nich w taki sam sposób jak w raporcie pierwszym.
W obu przypadkach zwrócona wartość przetrzymywana jest w *ResultSecie*, a następnie zostaje zwórcona.

- **Dodawanie osoby do drzewa genealogicznego**

```
public static boolean addNode(Connection connection, Integer tree_id, String person_id, String name, String surname, String gender, String dads_id, String moms_id, String
    if(dads_id.equals("0")){
        dads_id = " null ";
    }
    if(moms_id.equals("0")){
        moms_id = " null ";
    }
    if(death.isEmpty()){
        death = " null ";
    }
    String xml_to_insert = "<Osoba>\n" +
            "<Osoba_id>" + person_id + "</Osoba_id>\n" +
            "<Imie>" + name + "</Imie>\n" +
            "<Nazwisko>" + surname + "</Nazwisko>\n" +
            "<Plec>" + gender + "</Plec>\n" +
            "<Ojciec_id>" + dads_id + "</Ojciec_id>\n" +
            "<Matka_id>" + moms_id + "</Matka_id>\n" +
            "<Urodziny>" + birth + "</Urodziny>\n" +
            "<Smierc>" + death + "</Smierc>\n" +
            "</Osoba>";
    PreparedStatement sql = connection.prepareStatement( sql: "UPDATE DrzewoGenealogiczne\n" +
            "SET XDocumentDrzewoGenealogiczne.modify('\n" +
            "    insert " + xml_to_insert + " as last into (/)[1]\n" +
            "')\n" +
            "WHERE DrzewoGenealogiczneID = " + tree_id + ";");
    sql.executeUpdate();
```

```
    PreparedStatement sql_check = connection.prepareStatement( sql: "SELECT Tbl.Col.value('(Osoba_id)[1]', 'varchar(20)') AS Osoba_id\n" +
            "FROM\n" +
            "DrzewoGenealogiczne\n" +
            "CROSS APPLY \n" +
            "XDocumentDrzewoGenealogiczne.nodes('Osoba') AS Tbl(Col) \n" +
            "WHERE \n" +
            " DrzewoGenealogiczneID = " + tree_id + ";");
    List<Integer> check = new ArrayList<>();
    ResultSet result_check = sql_check.executeQuery();
    while(result_check.next()){
        check.add(result_check.getInt( columnLabel: "Osoba_id"));
    }

    return check.contains(Integer.valueOf(person_id));
}
```

Metoda przyjmuje wszystkie potrzebne do utworzenia osoby wartości które zostają dodane do XML. Następnie za pomocą metody $modify$ utworzony dokument XML zostaje dodany do konkretnego drzewa genealogicznego. Metoda sprawdza czy dodana osoba znajduje się w drzewie genealogicznym, jeżeli tak zwracane jest $true$, jeżeli nie - $false$.

- **Usuwanie osoby z drzewa genealogicznego**

```
public static boolean deleteNode(Connection connection, Integer tree_id, Integer node_id) throws SQLException {
    PreparedStatement sql = connection.prepareStatement( sql: "UPDATE DrzewoGenealogiczne\n" +
            "SET XDocumentDrzewoGenealogiczne.modify('delete Osoba[Osoba_id=" + node_id + "]')\n" +
            "WHERE DrzewoGenealogiczneID = " + tree_id + ";");
    sql.executeUpdate();

    PreparedStatement delete_mother = connection.prepareStatement( sql: "UPDATE DrzewoGenealogiczne\n" +
            "SET XDocumentDrzewoGenealogiczne.modify('\n" +
            "    replace value of (Osoba[Matka_id="+ node_id+ "]/Matka_id/text())[1]\n" +
            "    with \" null \"\n" +
            "')\n" +
            "WHERE DrzewoGenealogiczneID = " + tree_id + ";");

    PreparedStatement delete_father = connection.prepareStatement( sql: "UPDATE DrzewoGenealogiczne\n" +
            "SET XDocumentDrzewoGenealogiczne.modify('\n" +
            "    replace value of (Osoba[Ojciec_id="+ node_id + "]/Ojciec_id/text())[1]\n" +
            "    with \" null \"\n" +
            "')\n" +
            "WHERE DrzewoGenealogiczneID = "+ tree_id +"; ");

    delete_mother.executeUpdate();
    delete_mother.executeUpdate();
    delete_father.executeUpdate();
    delete_father.executeUpdate();
```

```
    PreparedStatement sql_check1 = connection.prepareStatement( sql: "SELECT Tbl.Col.value('(Osoba_id)[1]', 'varchar(20)') AS Osoba_id\n" +
            "FROM \n" +
            "    DrzewoGenealogiczne\n" +
            "CROSS APPLY \n" +
            "    XDocumentDrzewoGenealogiczne.nodes('Osoba') AS Tbl(Col)\n" +
            "WHERE \n" +
            "    DrzewoGenealogiczneID = " + tree_id + ";");
    List<Integer> check = new ArrayList<>();
    ResultSet result_check = sql_check1.executeQuery();
    while(result_check.next()){
        check.add(result_check.getInt( columnLabel: "Osoba_id"));
    }

    return !check.contains(node_id);
}
```

Metoda przyjmuje ID osoby którą chcemy usunąć, i z jakiego drzewa genealogicznego chcemy ją usunąć. Za pomocą funkcji $modify$ usuwamy konkretną osobę. Następnie poleceniem $SELECT$ sprawdzamy czy osboa o usuniętym ID znajduje się dalej w drzewie, jeśli nie - zwracany jest $true$, jeśli tak - zwracany jest $false$.

4 Prezentacja przeprowadzonych testów jednostkowych

W aplikacji znajdują się trzy testy jednostkowe odpowiadające za testowanie metod zaimplementowanych w API, poza metodą generującą raporty:

- **Before All**

```
@BeforeAll
static void setUp() throws SQLException, ClassNotFoundException {
    database = new UtilsDatabase();
    database.connectToDatabase();
    connection = database.getConnection();
    database.initialization_sql();
}
```

Metoda $BeforeAll$ odpowiada za utworzenie połączenia z bazą danych, oraz za inicjalizację jej, w celu testowania metod zaimplementowanych w API.

- **Test dodawania drzewa**

```
@Test
void addTreeTest() throws SQLException {
    Integer id = APILibrary.addTree(connection);
    PreparedStatement sql2 = connection.prepareStatement( sql: "SELECT DrzewoGenealogiczneID FROM DrzewoGenealogiczne ORDER BY DrzewoGenealogiczneID ASC");
    ResultSet resultSet = sql2.executeQuery();
    Integer id2 = 0;
    while(resultSet.next()) {
        id2 = resultSet.getInt( columnLabel: "DrzewoGenealogiczneID");
    }
    assertEquals(id, id2);
}
```

Test dodający drzewo wywołuje odpowiednią metodę w API, a następnie za pomocą $ASSERT$ sprawdza czy drzewo zostało dodane.

- **Test dodawania osoby**

```
@Test
void addNode() throws SQLException {
    Integer treeId = 1;
    Integer nodeId = 9;
    String name = "Bartosz";
    String surname = "Konopka";
    String gender = "Mezczyzna";
    Integer dadId = 0;
    Integer momId = 0;
    String birth = "2002-03-4";
    String death ="";
    boolean isAdded = APILibrary.addNode(connection, treeId, String.valueOf(nodeId), name, surname, gender, String.valueOf(dadId), String.valueOf(momId), birth, death);
    assertTrue(isAdded);
}
```

Test dodawania osoby wywołuje odpowiednią metodę w API, a następnie za pomocą $ASSERT$ sprawdza czy metoda zwróciła wartość *true* - czyli czy osoba została dodana.

- **Test usuwania osoby**

```
@Test
void deleteNode() throws SQLException {
    boolean isDeleted = APILibrary.deleteNode(connection, tree_id: 1, node_id: 1);
    assertTrue(isDeleted);
}
```

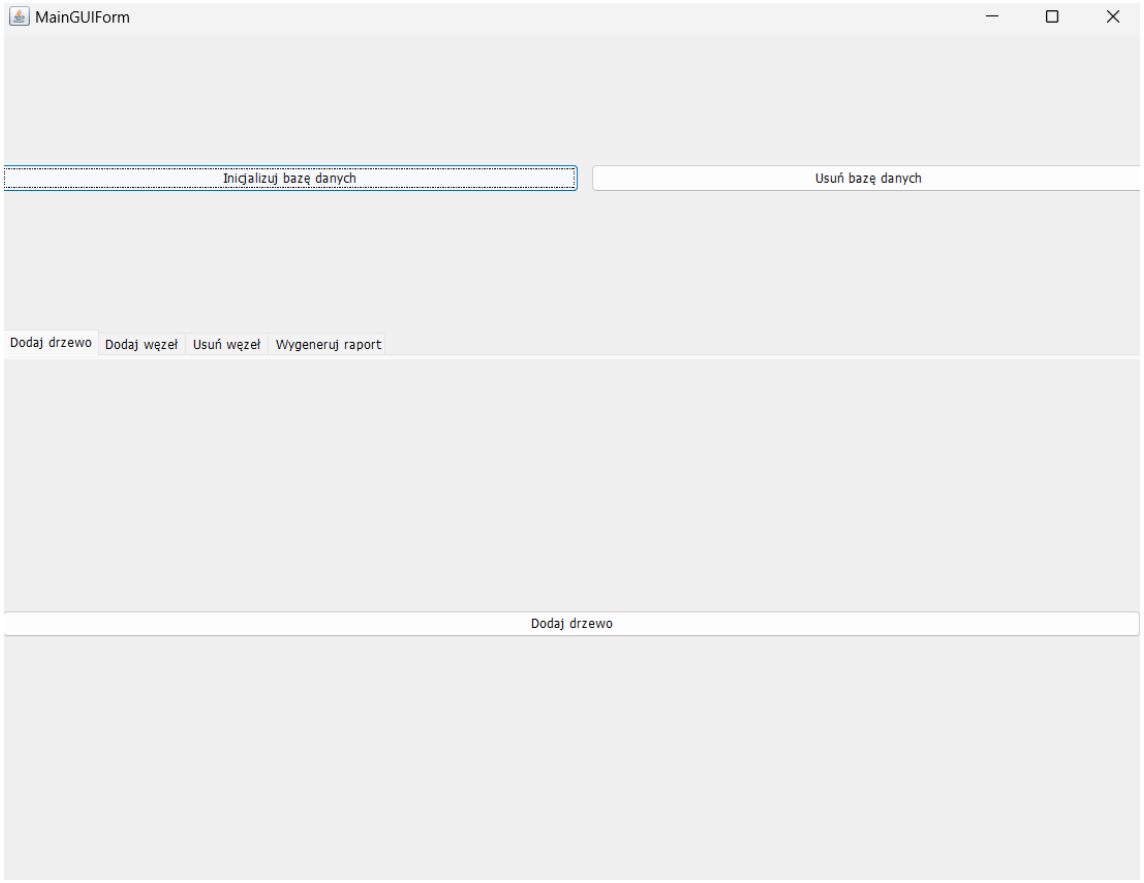
Test usuwania osoby wywołuje odpowiednią metodę w API, a następnie za pomocą $ASSERT$ sprawdza czy metoda zwróciła wartość *true* - czyli czy osoba została usunięta.

Wywołanie powyższych testów prezentuje się następująco:

Zauważyć można że wszystkie testy przechodzą prawidłowo.

5 Prezentacja przykładowej aplikacji

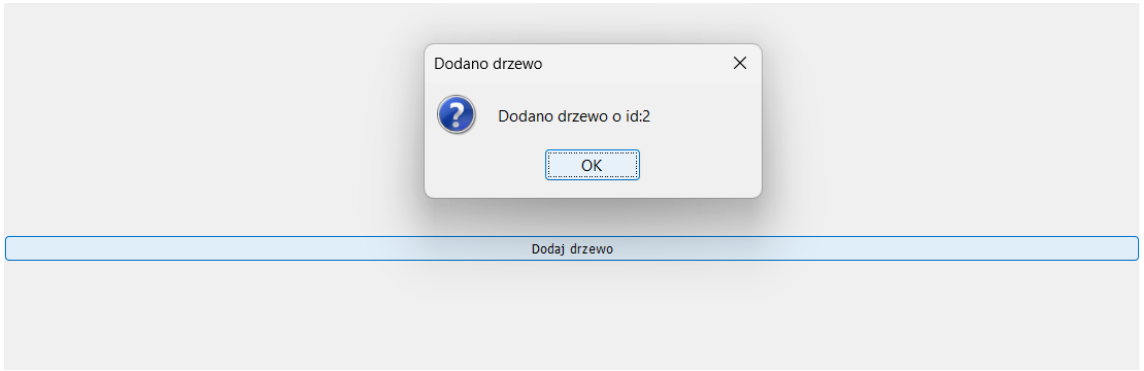
Aplikacja prezentująca możliwości API napisana została w języku $JAVA$. Stworzona ona została metodą MVC z wykorzystaniem singletonów. GUI aplikacji stworzone zostało za pomocą $JSwinga$. Poniżej zaprezentowany został wygląd aplikacji.



Przycisk "Inicjalizuj bazę danych" tworzy bazę, tabelę, oraz wypełnia ją wartościami. Przycisk "Usuń bazę danych", usuwa tabelę i bazę.
Dolna część aplikacji dzieli się na cztery części - pierwsza odpowiada za dodawanie drzewa, druga za dodawanie węzła, trzecia za usunięcie węzła, a ostatnia za generowanie raportów.

- **Dodawanie drzewa**

  Pierwszy panel odpowiada za dodawanie nowego drzewa genealogicznego. W przypadku powodzenia operacji, na ekranie wyskakuje okno z informacją o dodaniu drzewa.

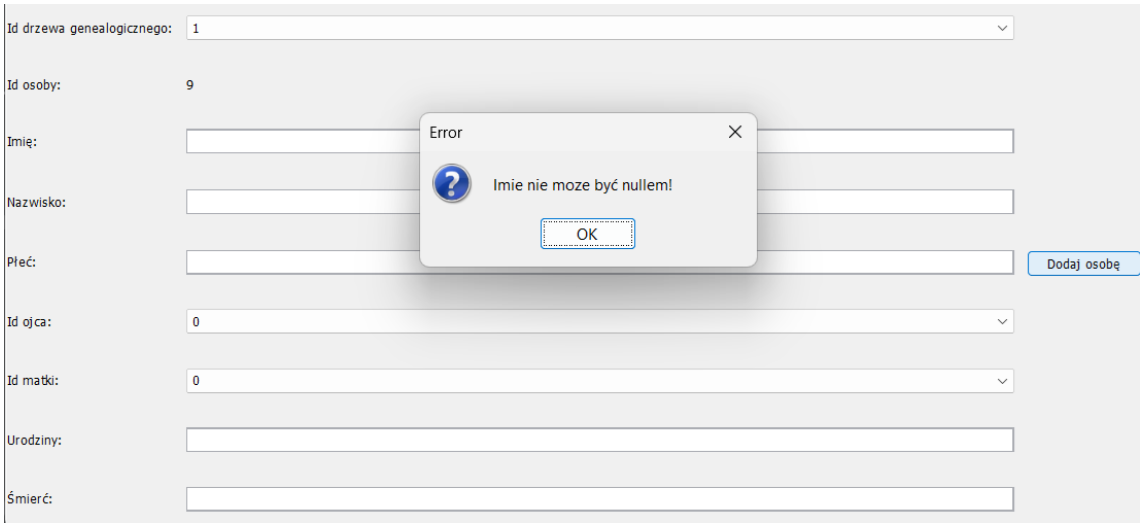


- **Dodawanie osoby**

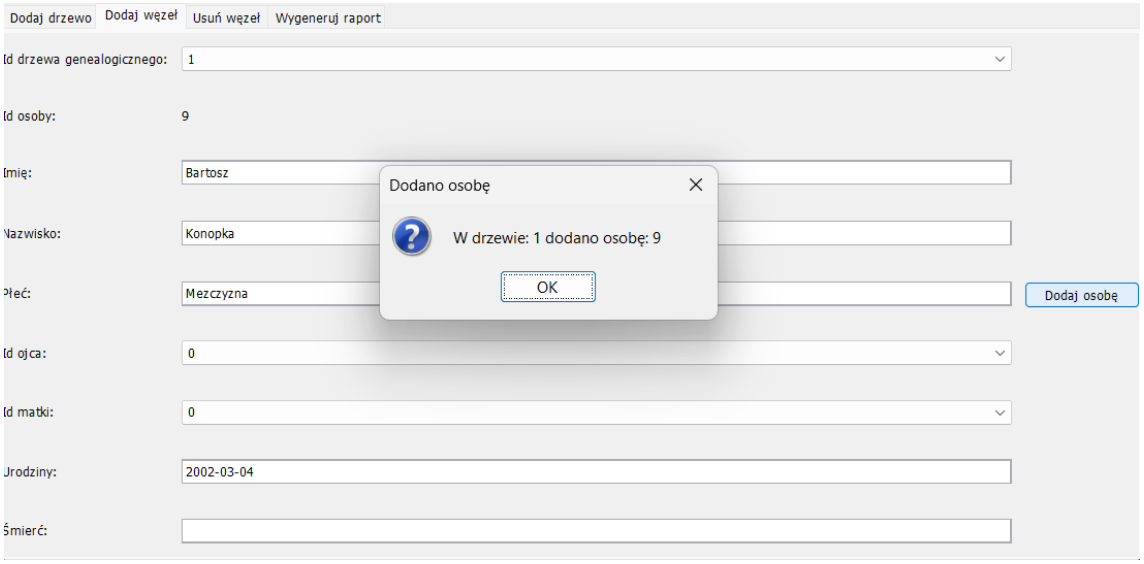
  Drugi panel odpowiada za dodawanie osoby do drzewa genealogicznego. Jest to formularz do którego można wpisać wartości opisujące dodawaną osobę. Gdy nie

chcemy dodawać ojca lub matki, w odpowiednich polach należy wybrać wartość 0. ID osoby jest wyliczane automatycznie na podstawie wartości znajdujących się w drzewie genealogicznym do którego dodajemy osobę.

W przypadku gdy wpisano nieprawidłową wartość w któreś z pól, wyskakujące okienko informuje o błędzie.

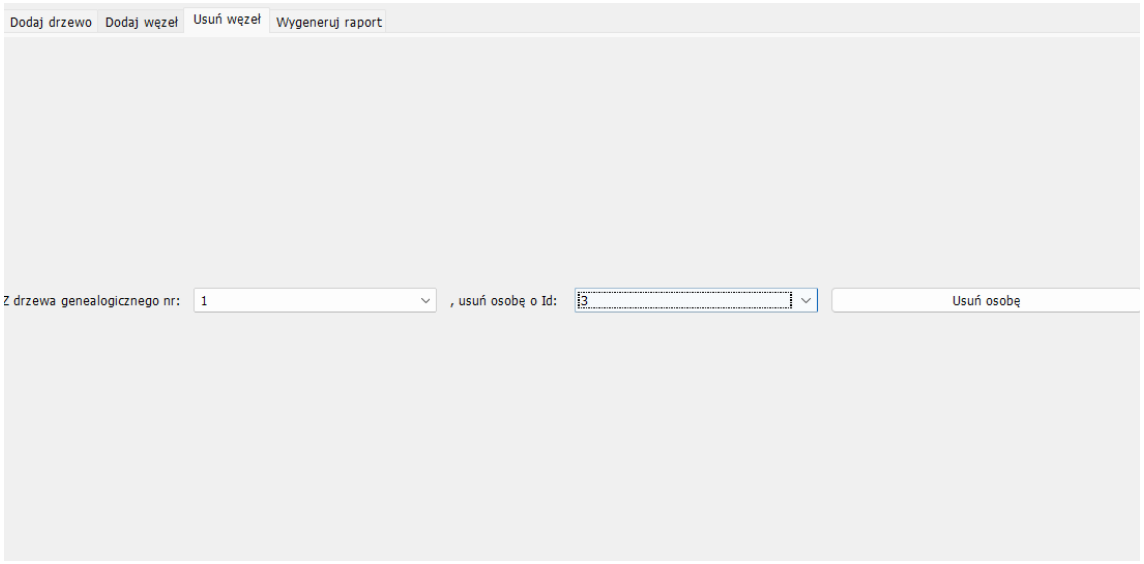


Gdy wszystkie wartości zgadzają się z wartościami oczekiwanymi, oraz wszystko przebiegnie pomyślnie, na ekranie wyświetla się informacja do którego drzewa dodano osobę i jakie ID zostało jej przypisane.

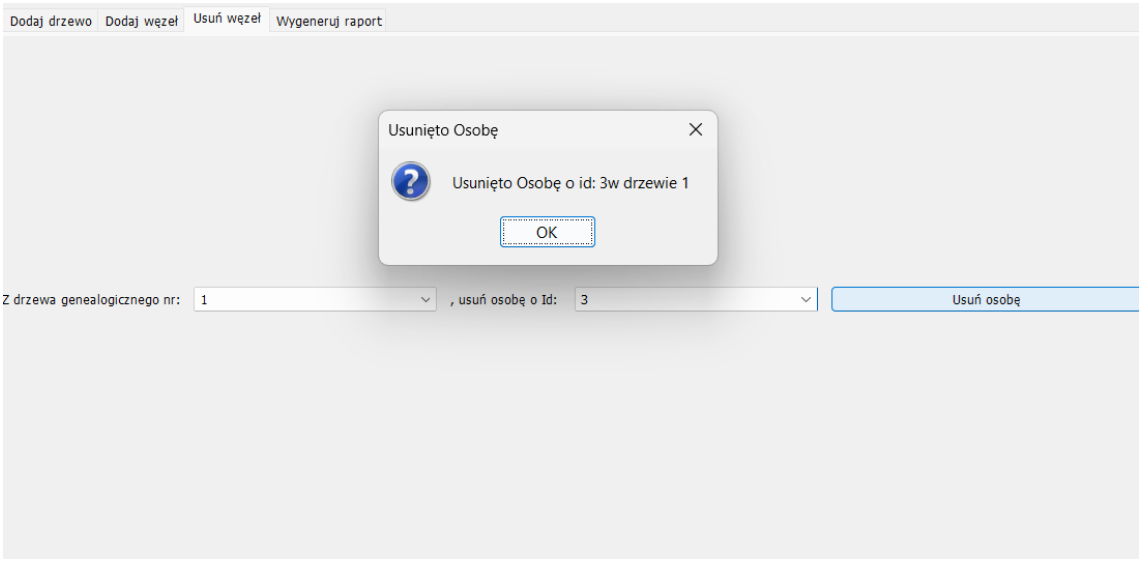


- **Usuwanie osoby**

  Trzeci panel odpowiada za usuwanie osoby z drzewa genealogicznego. Najpierw można wybrać drzewo z którego usuwamy osobę, a następnie jaką osobę chcemy usunąć.

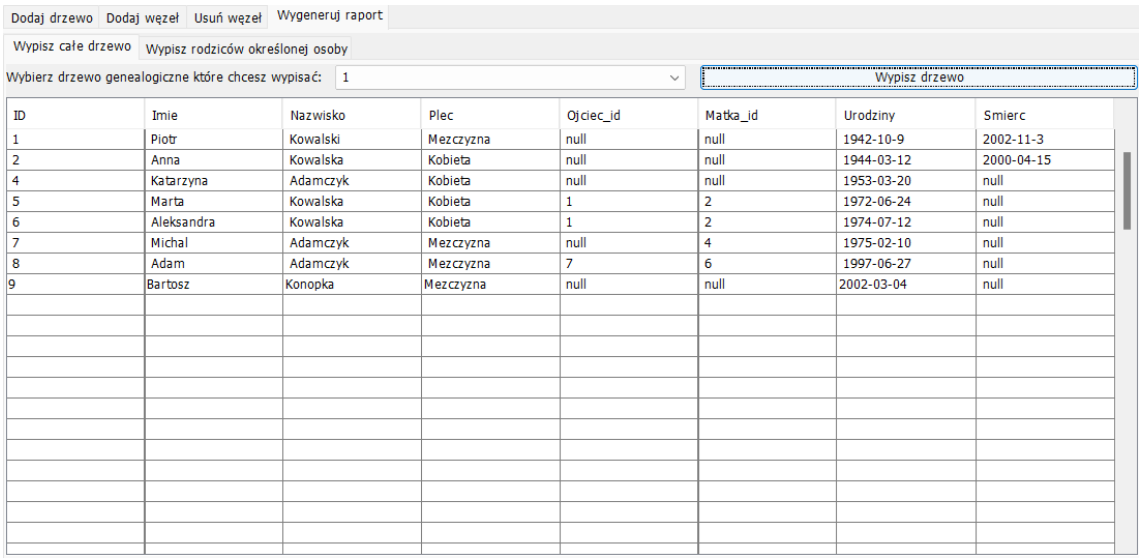


  Jeśli wszystko przebiegło pomyślnie, zobaczymy okienko informujące o powodzeniu operacji.



- **Generowanie raportu**

  Ostatni panel odpowiada za wyświetlanie raportów.
  Pierwszy raport oczekuje podania drzewa genealogicznego na którym aplikacja będzie wykonywała *SELECT*, a następnie wypisuje z niego wszystkie osoby.



ID	Imie	Nazwisko	Plec	Ojciec_id	Matka_id	Urodziny	Smierc
1	Piotr	Kowalski	Mezczyzna	null	null	1942-10-9	2002-11-3
2	Anna	Kowalska	Kobieta	null	null	1944-03-12	2000-04-15
4	Katarzyna	Adamczyk	Kobieta	null	null	1953-03-20	null
5	Marta	Kowalska	Kobieta	1	2	1972-06-24	null
6	Aleksandra	Kowalska	Kobieta	1	2	1974-07-12	null
7	Michal	Adamczyk	Mezczyzna	null	4	1975-02-10	null
8	Adam	Adamczyk	Mezczyzna	7	6	1997-06-27	null
9	Bartosz	Konopka	Mezczyzna	null	null	2002-03-04	null

W drugim raporcie podajemy *ID* drzewa genealogicznego, a następnie *ID* osoby której rodziców chcemy wypisać. Gdy dana osoba nie ma wpisanych rodziców, nic nie wyświetli się w tabeli.

6 Podsumowanie i wnioski

Opracowane *API* i biblioteka skutecznie umożliwiają zarządzanie danymi hierarchicznymi - drzewami genealogicznymi z wykorzystaniem typu *XML* w *SQL Server*. Testy jednostkowe potwierdziły poprawność implementacji, a przykładowa aplikacja zademonstrowała praktyczne zastosowanie biblioteki.