HAPI Dokumentacja

Łukasz Gut

27.05.2020

Contents

1 Wprowadzenie

1.1 HAPI

HAPI to API umożliwiające użytkownikowi zarządzanie hierarchiczną strukturą firmy.

Stos technologiczny:

- C#
- Microsoft SQL Server

HAPI serwowane jest jako biblioteka. Udostępnia ona użytkownikowi zestaw klas, które ułatwiają współpracę z bazą danych i w elegancki sposób abstrachują detale implementacyjne. Dzięki temu kod klienta może być czysty i zwięzły.

1.2 Problem

Celem tego projektu było zaprojektowanie API w języku C#, umożliwiającego zarządzanie hierarchiczną strukturą firmy.

Głównym założeniem projektu było wykorzystanie typu XML do przechowywania danych hierarhicznych.

1.3 Funkcjonalność

Poprzez zestaw klas HAPI pozwala użytkownikowi na dodawanie/usuwanie drzew XML w tabeli (dodawanie/usuwanie firm), dodawanie/usuwanie węzłów (dodawanie/usuwanie pracowników) oraz tworzenie wybranych raportów.

2 API

2.1 ConnectionContext

Klasa reprezentująca kontekst połączeniowy z bazą danych.

connectionString
Typ

- <<string>>

Ciąg znaków, służący do ustanowienia połączenia z bazą danych.

ConnectionContext()
Parametry

- connectionString <<string>> Ciąg znaków, służący do ustanowienia połączenia z bazą danych.

2.2 CompanyDAO

Klasa udostępniająca interfejs do zarządzania firmami.

CompanyDAO()
Parametry

- connectionString <<ConnectionContext>> Kontekst połączania.

GetCompanyById()
Parametry

- id <<int>> Id firmy.

Typ zwracany

- <<Company>> instancja typu Company.

Metoda służąca do pobierania firmy z tabeli.

GetCompanyByName()
Parametry

- name <<string>> Nazwa firmy.

Typ zwracany

- <<Company>> instancja typu Company.

Metoda służąca do pobierania firmy z tabeli.

GetAllCompanies()
Typ zwracany

- <<List<Company>>> Lista instancji typu Company.

Metoda służąca do pobierania firmy wszystkich firm w tabeli.

AddCompany()
Parametry

- company <<Company>> Instancja typu Company.

Typ zwracany

- <<void>>

Metoda służąca do dodawania firmy do tabeli.

RemoveCompanyByName()
Parametry

- name <<string>> Nazwa firmy.

Typ zwracany

- <<void>>

Metoda służąca do usuwania firmy do tabeli.

2.3 EmployeeDAO

Klasa udostępniająca interfejs do zarządzania pracownikami w firmie.

EmployeeDAO()

Parametry

- connectionContext << *ConnectionContext* >> Kontekst połączenia.
- companyName <<string>> Nazwa firmy.

GetEmployeeById()

Parametry

- id <<string>> Id pracownika.

Typ zwracany

- <<Employee>> Instancja typu Employee.

Metoda służąca do pobierania pracownika z tabeli.

GetEmployeesByFullName()

Parametry

- firstName <<string>> Imię pracownika.
- lastName <<string>> Nazwisko pracownika.

Typ zwracany

- <<List<Employee>>> Lista instancji typu Employee.

Metoda służąca do pobierania pracowników z tabeli.

GetEmployeesByFirstName()

Parametry

- firstName <<string>> Imię pracownika.

Typ zwracany

- <<List<Employee>>> Lista instancji typu Employee.

Metoda służąca do pobierania pracowników z tabeli.

GetEmployeesByLastName()
Parametry

- lastName <<string>> Nazwisko pracownika.

Typ zwracany

- <<List<Employee>>> Lista instancji typu Employee.

Metoda służąca do pobierania pracowników z tabeli.

GetAllEmployees()
Typ zwracany

- <<List<Employee>>> Lista instancji typu Employee.

Metoda służąca do wszystkich pracowników z tabeli.

GetManagerByEmployeeId()
Parametry

- id <<string>> Id pracownika.

Typ zwracany

- <<Employee>> Instancja typu Employee.

Metoda służąca do pobierania menadżera podanego pracownika.

GetStaffByEmployeeId()
Parametry

- id <<string>> Id pracownika.

Typ zwracany

- <<Employee>> Lista instancji typu Employee.

Metoda służąca do pobierania wszystkich osób będących pod danym menadżerem.

AddEmployee()
Parametry

- employee <<Employee>> Instancja typu Employee.

Typ zwracany

- <<void>>

Metoda służąca do dodawania pracownika do tabeli.

RemoveEmployeeById()
Parametry

- id <<string>> Id pracownika.

Typ zwracany

- <<void>>

Metoda służąca do usuwania pracownika z tabeli.

2.4 Company

Klasa reprezentująca firmę.

Null
Typ

- <<Company>> Nowa, pusta instancja.

Id
Typ

- <<int>> Id firmy.

Name
Typ

- <<int>> Nazwa firmy.

Data
Typ

- <<XDocument>> Dane o firmie.

Company()
Parametry

id <<string>> Id firmy. Parametr opcjonalny, jeśli się go nie poda firmie zostanie nadane unikalne GUID.

- name <<string>> Nazwa firmy.
- data <<XDocument>> Kompletne dane o firmie. Dokument musi się poprawnie walidować z XSD Schema zdefiniowaną w bazie danych.

ToString()
Typ zwracany

- <<string>>

2.5 Employee

Klasa reprezentująca pracownika.

Null
Typ

- <<Company>> Nowa, pusta instancja.

Id
Typ

- <<string>> Id pracownika.

ManagerId
Typ

- <<string>> Id menadżera.

FirstName
Typ

- <<string>> Imię pracownika.

LastName
Typ

- <<string>> Nazwisko pracownika.

ContactNo
Typ

- <<string>> Telefon pracownika.

Email
Typ

- <<string>> Email pracownika.

Address
Typ

- <<Address>> Adres pracownika.

Employee()
Parametry

- document <<XDocument>> Dane o pracowniku.

Employee()
Parametry

id <<string>> Id pracownika. Parametr opcjonalny, jeśli się go nie poda firmie zostanie nadane unikalne GUID.

- managerId <<string>> Id menadżera.
- firstName <<string>> Imię pracownika.
- lastName <<string>> Nazwisko pracownika.
- contactNo <<string>> Telefon pracownika.
- email <<string>> Email pracownika.
- address <<Address>> Adres pracownika.

ToXDocument()
Typ zwracany

- <<XDocument>>

ToString()
Typ zwracany

- <<string>>

2.6 Address

Klasa reprezentująca adres pracownika.

Null
Typ

- <<Address>> Nowa, pusta instancja.

City
Typ

- <<string>> Miasto.

State
Typ

- <<string>> Województwo.

Zip
Typ

- <<string>> Kod pocztowy.

Address()
Parametry

- city <<string>> Miasto.
- state <<string>> Województwo.
- zip <<string>> Kod pocztowy.

ToString()
Typ zwracany

- <<string>>

---

3 Implementacja

Jako, że celem projektu było wykorzystanie typu XML w Microsoft SQL Server, zdecydowałem się na zaimplementowanie wszelkiej logiki służącej do pobierania informacji z bazy danych po stronie SQL Server. W związku z tym HAPI jest tylko cienką warstwą, która wystawia użytkownikowi przyjazny interfejs do obsługi bazy danych, zaś wszelka logika została napisana w języku T-SQL.

Szczegóły implementacyjne można zobaczyć w kodzie źródłowym.

4 Testy jednostkowe

Testy jednostkowe znajdują się w folderze **hapi-tests**/. Są one oddzielnym projektem,
który otwiera się razem z głównym Solution. Przetestowane zostało **100%** kodu.

Do stworzenia testów jednostkowych wykorzystany został moduł UnitTesting dostępny natywnie w języky C#.

Testy jednostkowe wymagają lokalnie działającej instancji SQL Server o nazwie **MSSQLLocalDB** oraz bazy danych o nazwie **[Hierarchy DB]** i wstępnie spreparowanej tabeli o nazwie **[Company]**. Skrypty do utworzenia takiej bazy znajdują się w folderze **scripts**/.

5 Podsumowanie

Wykorzystanie typu XML do reprezentacji hierarchii z zależnościami jest bardzo trudne i nieporęczne. Pomimo tego, że da się to zrobić, nie jest to optymalne, a sam kod w języku T-SQL bardzo trudny do napisania.

SQL Server nie wspiera również w pełni specyfikacji XML. Dla przykładu: element **unique**, który wykorzystany w XML schema potrafi zapewnić unikalne wartości w węzłach. Jest to niesamowicie uciążliwe, ponieważ praktycznie uniemożliwia to zapewnienie unikalnych wartości w węzłach (jest to duży problem np. w przypadku id pracownika).

Bardzo trudna i nieporęczna jest również rekurencja.

Podsumowując, do reprezentacji hierarchicznych danych z zależnościami, dużo prościej jest wykorzystać w SQL Server typ HierarchyID, który natywnie wspiera wszystkie operacje potrzebne do operacji na tego typu strukturze drzewiastej.

6 Literatura

- https://docs.microsoft.com/en-us/dotnet/csharp/