



## **API Firmy (w oparciu o HierarchyID)**

# **Przetwarzanie danych hierarchicznych**

Tomasz Gajda

## **SPIS TREŚCI**

<b>1. Problem i założenia projektu</b>	<b>1</b>
<b>2. Krótki wstęp teoretyczny</b>	<b>2</b>
<b>3. Zakres funkcjonalności</b>	<b>2</b>
3.1 Tabela Employee	2
3.2 Metody klasy Company	3
<b>4. Opis implementacji</b>	<b>5</b>
<b>5. Przykładowe dane</b>	<b>6</b>
<b>6. Prezentacja testów jednostkowych</b>	<b>7</b>
<b>7. Podsumowanie</b>	<b>8</b>
<b>8. Literatura</b>	<b>8</b>
<b>9. Kod źródłowy</b>	<b>9</b>

# 1. Problem i założenia projektu

Celem **CMAPI** (Company Management API) jest umożliwienie obsługi struktury firmy będącej strukturą **hierarchiczną**. Elementami w hierarchii będą **pracownicy** wraz z ich danymi personalnymi. Głównym założeniem projektu jest wykorzystanie typu **HierarchyID** do przechowywania wspomnianych danych hierarchicznych.

## Stos technologiczny:

- C#
- Microsoft SQL Server
- Typ HierarchyID
- Visual Studio Testing Tools

**CMAPI** jest biblioteką udostępniającą zestaw **dwóch klas** głównych (Employee i Company), z których jedna jest placeholderem do przechowywania danych o pracowniku, a druga daje dostęp do grupy metod działających na tych danych.

# 2. Krótki wstęp teoretyczny

Do stworzenia API wykorzystałem typ **HierarchyID** - używamy go, aby przedstawić pozycję w hierarchii. Kolumna typu *hierarchyid* nie reprezentuje automatycznie drzewa. Do aplikacji należy generowanie i przypisywanie wartości *hierarchyid* w taki sposób, aby pożądana relacja między wierszami była odzwierciedlona w wartościach.

# 3. Zakres funkcjonalności

Poprzez zestaw metod API pozwala użytkownikowi na dodawanie/usuwanie **pracowników** oraz tworzenie wybranych raportów.

Oprócz samego **API**, w projekcie zostaną również uwzględnione testy jednostkowe, skrypty służące do utworzenia przykładowej bazy oraz aplikacja konsolowa, służąca do przedstawienia przykładu działania stworzonego **API**.

## 3.1 Tabela Employee

Przechowuje dane na temat pracowników

Employee TABLE	
id	int UNQ ID
level	hierarchyid
firstName	nvarchar(30)
lastName	nvarchar(30)
position	nvarchar(30)
salary	int

Rys. 1 - Tabela **Employee** przechowująca dane o pracownikach

## 3.2 Metody klasy Company

Udostępniają manipulację danymi pracowników

### 1. Dodawanie pracownika - AddEmployee()

Parametry

- id <<int>>
- level <<string>>
- firstName <<string>>
- lastName <<string>>
- position <<string>>
- salary <<int>>

Typ zwracany

- <<void>>

### 2. Usuwanie pracownika po id - RemoveEmployeeById()

Parametry

- id <<int>>

Typ zwracany

- <<void>>

**3. Usuwanie pracownika po imieniu - RemoveEmployeeByFirstName()**

Parametry

- **firstName** <<string>>

Typ zwracany

- <<void>>

**4. Usuwanie pracownika po nazwisku - RemoveEmployeeByLastName()**

Parametry

- **lastName** <<string>>

Typ zwracany

- <<void>>

**5. Usuwanie pracownika po hierarchii - RemoveEmployeeByLevel()**

Parametry

- **level** <<string>>

Typ zwracany

- <<void>>

**6. Usuwanie wszystkich pracowników - RemoveAllEmployees()**

Parametry

- <<void>>

Typ zwracany

- <<void>>

**7. Zwróć pracownika po id - GetEmployeeById()**

Parametry

- **id** <<int>>

Typ zwracany

- <<Employee>>

**8. Zwróć pracownika ze specyficznym HierarchyID - GetEmployeeByLevel()**

Parametry

- **level** <<string>>

Typ zwracany

- <<Employee>>

**9. Zwróć pracownika po imieniu - GetEmployeeByFirstName()**

Parametry

- **firstName** <<string>>

Typ zwracany

- **<<Employee>>**

**10. Zwróć pracownika po nazwisku - GetEmployeeByLastName()**

Parametry

- **lastName** <<string>>

Typ zwracany

- **<<Employee>>**

**11. Zwróć pracownika ze specyficznym HierarchylD wraz z jego podwładnymi -**

**GetEmployeeWithSubordinates()**

Parametry

- **level** <<string>>

Typ zwracany

- **<<List<Employee>>>**

**12. Zwróć wszystkich pracowników - GetAllEmployees()**

Parametry

- **<<void>>**

Typ zwracany

- **<<List<Employee>>>**

**13. Zwróć najwyższe wynagrodzenie - GetMaxSalary()**

Parametry

- **<<void>>**

Typ zwracany

- **salary** <<int>>

**14. Zwróć średnie wynagrodzenie - GetAverageSalary()**

Parametry

- **<<void>>**

Typ zwracany

- **salary** <<int>>

## 4. Opis implementacji

**CMAPI** udostępnia procedury wykonujące operacje na rekordach tabeli pracowników (Employee) w bazie danych. Do zaimplementowania wykorzystałem klasy w języku **C#**, które wykonują **zapytania SQL** do bazy danych, które uruchamiają żądane **procedury**. W procedurach wykonywane są instrukcje SQL, które korzystają m. in. z typu **hierarchiid**.

Wszystkie funkcje są dostępne z poziomu interaktywnej konsolowej aplikacji (**demo**), przyjmującej input od użytkownika, który decyduje co chce zrobić. Jest to przykład implementacji API w działającej aplikacji - funkcje można wdrożyć również do istniejącego wcześniej systemu.

```
===== Konsolowa aplikacja testująca CMAPI =====
Dostępne komendy:

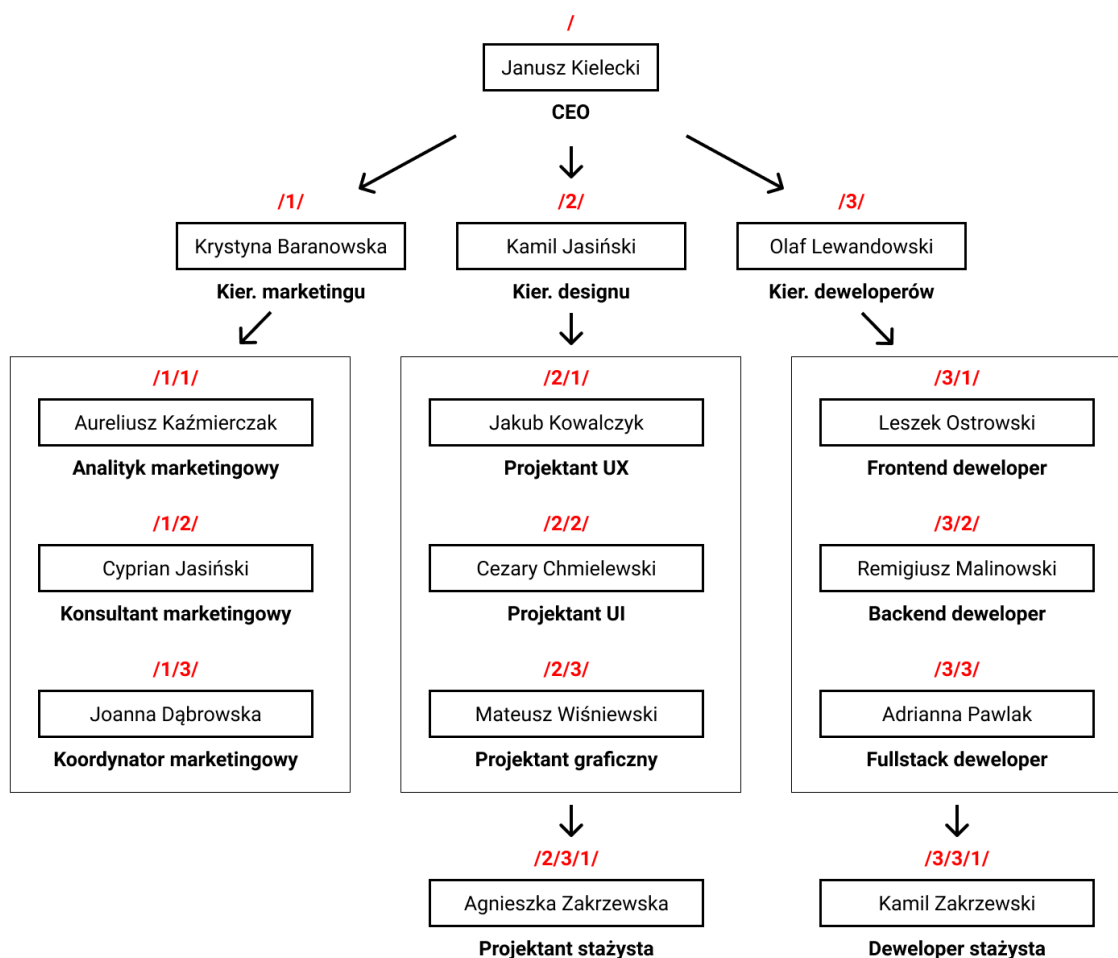
1. Dodaj nowego pracownika
2. Usuń pracownika po ID
3. Usuń pracownika po imieniu
4. Usuń pracownika po nazwisku
5. Usuń pracownika po hierarchii
6. Usuń wszystkich pracowników
7. Zwróć pracownika po ID
8. Zwróć pracownika po hierarchii
9. Zwróć pracownika po imieniu
10. Zwróć pracownika po nazwisku
11. Zwróć pracownika z podwładnymi (po hierarchii)
12. Zwróć wszystkich pracowników
13. Zwróć największą wypłatę
14. Zwróć średnią wypłatę
15. Dodaj przykładowe dane
16. Zakończ działanie aplikacji

Wybierz numer komendy:
-
```

Rys. 2 - Obraz przedstawia **startowy ekran** konsolowej aplikacji demonstracyjnej

## 5. Przykładowe dane

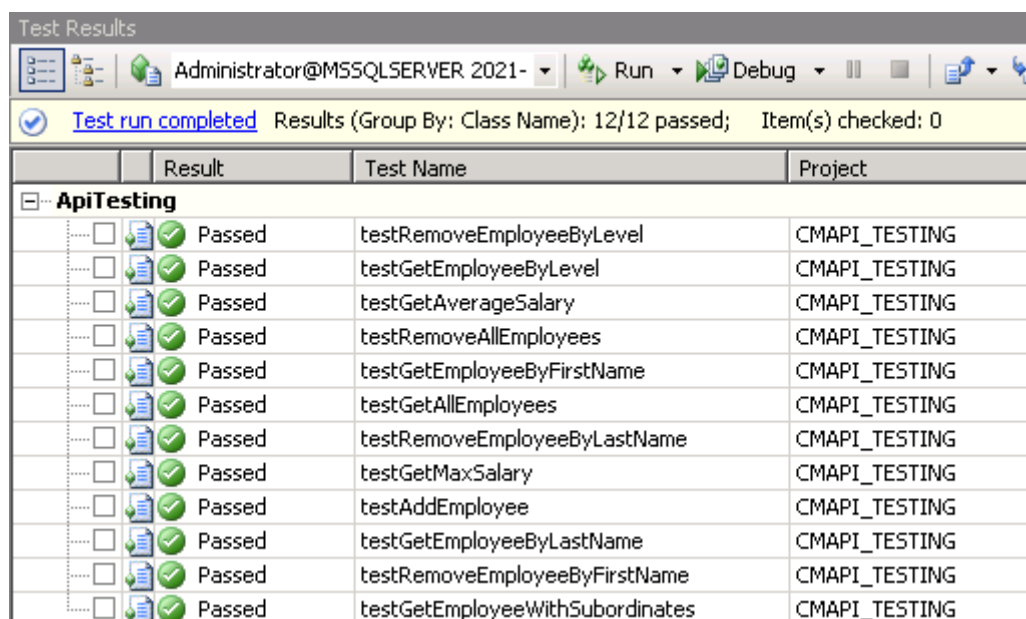
W ramach aplikacji demonstrującej działanie **CMAPI**, możemy do bazy dodać gotowe przykładowe dane - są to testowe dane firmy, która korzysta z **hierarchicznej** struktury danych. **Zbieżność imion i nazwisk jest przypadkowa.**



Rys. 3 - Drzewo przedstawiające strukturę danych w przykładowej firmie

## 6. Prezentacja testów jednostkowych

Do testowania **CMAPI** wykorzystałem narzędzia do testowania udostępnione przez **Visual Studio**. W projekcie testowym, na przykładowych danych przetestowane zostały wszystkie\* metody wykorzystane w API, poniżej przedstawiam wyniki przeprowadzonych testów.



The screenshot shows the 'Test Results' window in Visual Studio. The status bar indicates 'Test run completed' and 'Results (Group By: Class Name): 12/12 passed; Item(s) checked: 0'. The table below lists the test results for the 'ApiTesting' group.

	Result	Test Name	Project
[-] ApiTesting			
<input type="checkbox"/>	Passed	testRemoveEmployeeByLevel	CMAPI_TESTING
<input type="checkbox"/>	Passed	testGetEmployeeByLevel	CMAPI_TESTING
<input type="checkbox"/>	Passed	testGetAverageSalary	CMAPI_TESTING
<input type="checkbox"/>	Passed	testRemoveAllEmployees	CMAPI_TESTING
<input type="checkbox"/>	Passed	testGetEmployeeByFirstName	CMAPI_TESTING
<input type="checkbox"/>	Passed	testGetAllEmployees	CMAPI_TESTING
<input type="checkbox"/>	Passed	testRemoveEmployeeByLastName	CMAPI_TESTING
<input type="checkbox"/>	Passed	testGetMaxSalary	CMAPI_TESTING
<input type="checkbox"/>	Passed	testAddEmployee	CMAPI_TESTING
<input type="checkbox"/>	Passed	testGetEmployeeByLastName	CMAPI_TESTING
<input type="checkbox"/>	Passed	testRemoveEmployeeByFirstName	CMAPI_TESTING
<input type="checkbox"/>	Passed	testGetEmployeeWithSubordinates	CMAPI_TESTING

Rys. 4 - Wyniki testów przeprowadzonych na **Company Management API**

## 7. Podsumowanie

Typ **hierarchyid** zdaje się być bardzo dobrą opcją do przetwarzania danych hierarchicznych - jest szybki i łatwy w użyciu. Oprócz tego, udostępnia również wiele pomocniczych funkcji (takich jak *IsDescendantOf*), które są bardzo przydatne, a musiałyby być dodatkowo zaimplementowane w przypadku własnoręcznego rozwiązania problemów tego typu.

Z drugiej strony, wsparcie C# dla *hierarchyid* nie jest pełne - przykładem może być konieczność ciągłego parsowania typu danych z **SqlHierarchyId** do **string**, by mogły one zostać przetworzone przez bazę danych. Słyszałem również że wydajność **hierarchyid** w bazach większego kalibru potrafi być nieakceptowalna - w przypadku małych baz, może nadać się idealnie, ale z czasem i z rośnięciem ilości rekordów, wydajność potężnie spada.

\* metody bazujące na ID nie zostały przetestowane, jako że ID może się zmieniać



## 8. Literatura

Do stworzenia projektu wykorzystałem głównie **oficjalną dokumentację Microsoftu**, oraz pomniejsze strony służące jako poradniki do obsługi typu **hierarchyid**:

1. <https://codingsight.com/how-to-use-sql-server-hierarchyid-through-easy-examples/>
2. <https://docs.microsoft.com/en-us/sql/t-sql/data-types/hierarchyid-data-type-method-reference?view=sql-server-ver15>
3. <https://docs.microsoft.com/en-us/dotnet/csharp/>
4. <https://docs.microsoft.com/en-us/sql/relational-databases/tables/tutorial-using-the-hierarchyid-data-type?view=sql-server-ver15>
5. <https://www.sqlshack.com/use-hierarchyid-sql-server/>

## 9. Kod źródłowy

Kod źródłowy wszystkich skryptów T-SQL wykorzystanych w projekcie znajduje się w folderze o nazwie **"dbsetup"**. Znajdują się tam skrypty dotyczące **tworzenia bazy, tabeli Employee, dodawania i usuwania potrzebnych procedur** oraz pomniejszych skryptów służących do testowania aktualnego stanu bazy danych.