Akademia Górniczo – Hutnicza im. St. Staszica w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej
Informatyka, Rok akademicki 2017-2018
Wzorce Projektowe

Row-Level Authorization

Skład osobowy:
Aleksander Lisiecki
Marcin Jakubowski
Natalia Materek
Minh Nhật Trịnh

1. Cel projektu

Celem projektu było stworzenie biblioteki implementującej moduł bezpieczeństwa dostępu do danych (na poziomie encji) w zależności od przypisanej roli użytkownika. Moduł miał wykorzystywać programowanie aspektowe, pozwalające na zastosowanie w dowolnym systemie, gdzie wykorzystano mapowanie O-R w technologii .Net. Role systemu jako struktura drzewiasta.

2. Język implementacji

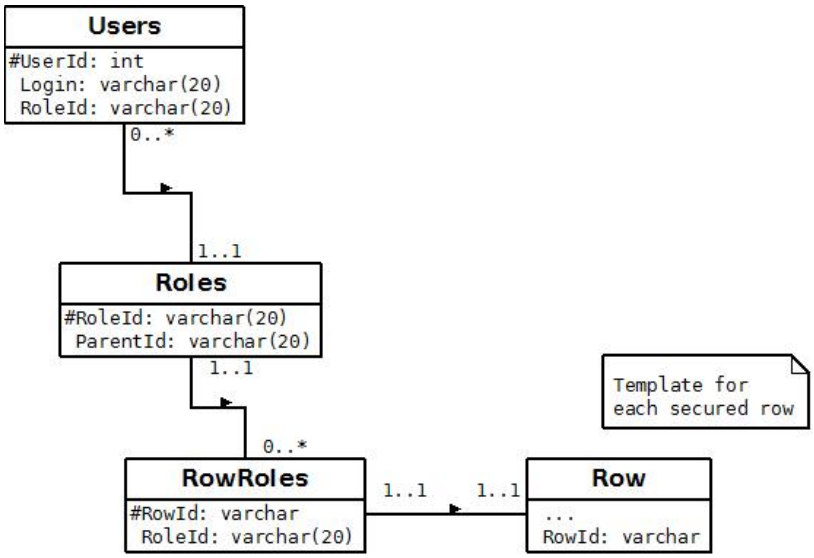
Projekt zrealizowany w języku C# 5.0 w technologii .Net.

3. Użyte biblioteki zewnętrzne

- ➔ Entity Framework 6.0 – system mapowanie O-R
- ➔ EntityFramework.DynamicFilters – biblioteka dynamicznie dodająca filter modyfikujący każde zapytanie do bazy danych
- ➔ PostSharp – biblioteka ułatwiająca programowanie aspektowe

4. Szczegóły implementacji

Głównym elementem biblioteki jest klasa abstrakcyjna ModelContext dziedzicząca po DbContext. Dodaje ona do istniejącej (lub nowej) bazy tablice Users, Roles, RowRoles, a dla każdego wiersza pozostałych tablic w bazie przypisuje unikalny w skali bazy RowId, służący do zidentyfikowania encji w tabeli RowRoles. Tabela RowRoles składa się z dwóch atrybutów: RowId i RoleId, gdzie RoleId jest id roli w systemie, której udostępniony jest wiersz o danym RowId. Role w systemie mają strukturę drzewiastą, tzn. jeżeli rola „Manager” jest rodzicem roli „Accountant” i roli „Accountant” jest udostępniony wiersz „x”, to automatycznie wiersz „x” jest udostępniony roli „Manager”.



Klasa ModelContext wykorzystuje DynamicFilter, aby zmodyfikować każde zapytanie do bazy, tak aby zwróciło tylko te rekordy, które aktualnie zalogowanemu użytkownikowi są udostępnione.

Aby dodać autoryzacje do własnego systemu mapowania O-R, należy stworzyć klasę, z definicją obiektową własnych tabel, dziedziczącą po ModelContext. Przed wykonaniem pierwszego zapytania do bazy danych, należy za pomocą metody Authorize(string username) ustawić login aktualnie zalogowanego użytkownika.

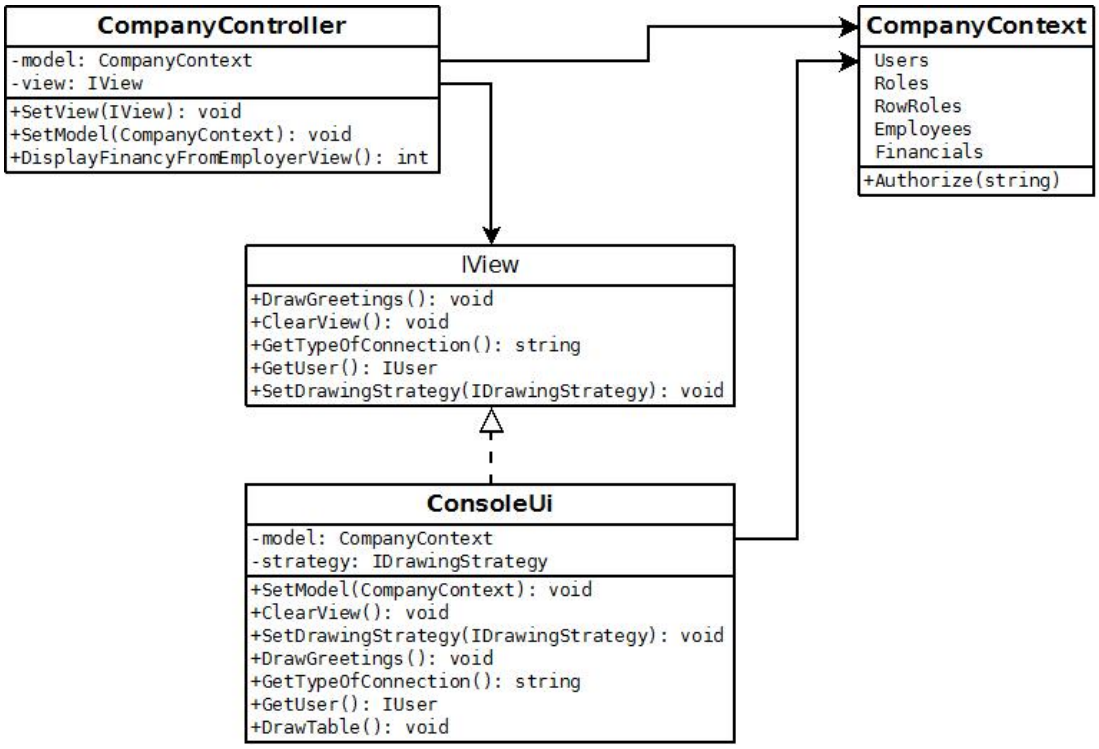
5. Przykładowe wykorzystanie biblioteki

Przykład firmy z bazą zawierającą pracowników i przypisane im wypłaty. Użytkownicy przypisani do konkretnych ról. Symulacja w oparciu o wzorzec MVC.

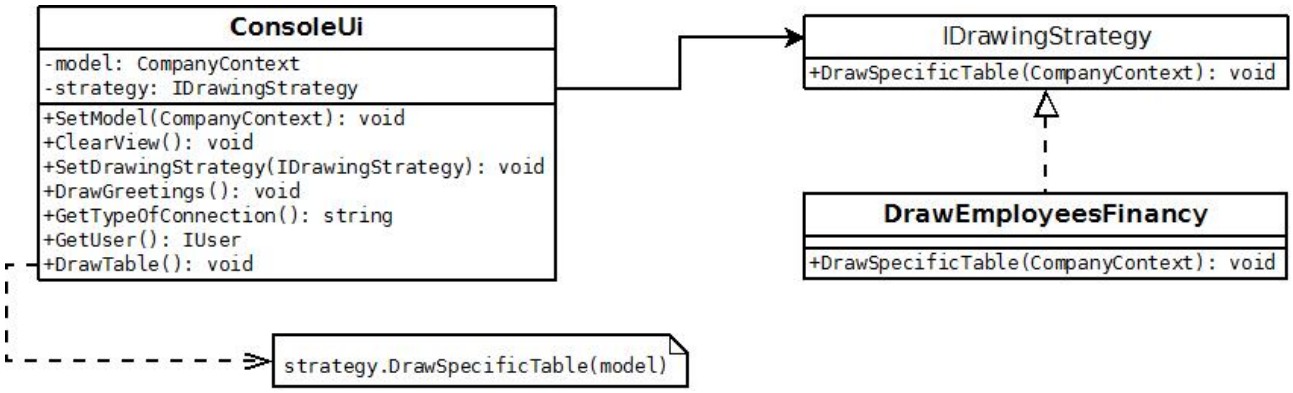
6. Wykorzystane wzorce projektowe

A) **Model-View-Controller** – opiera się na nim cała struktura aplikacji przykładowej. Składa się z trzech głównych klas:

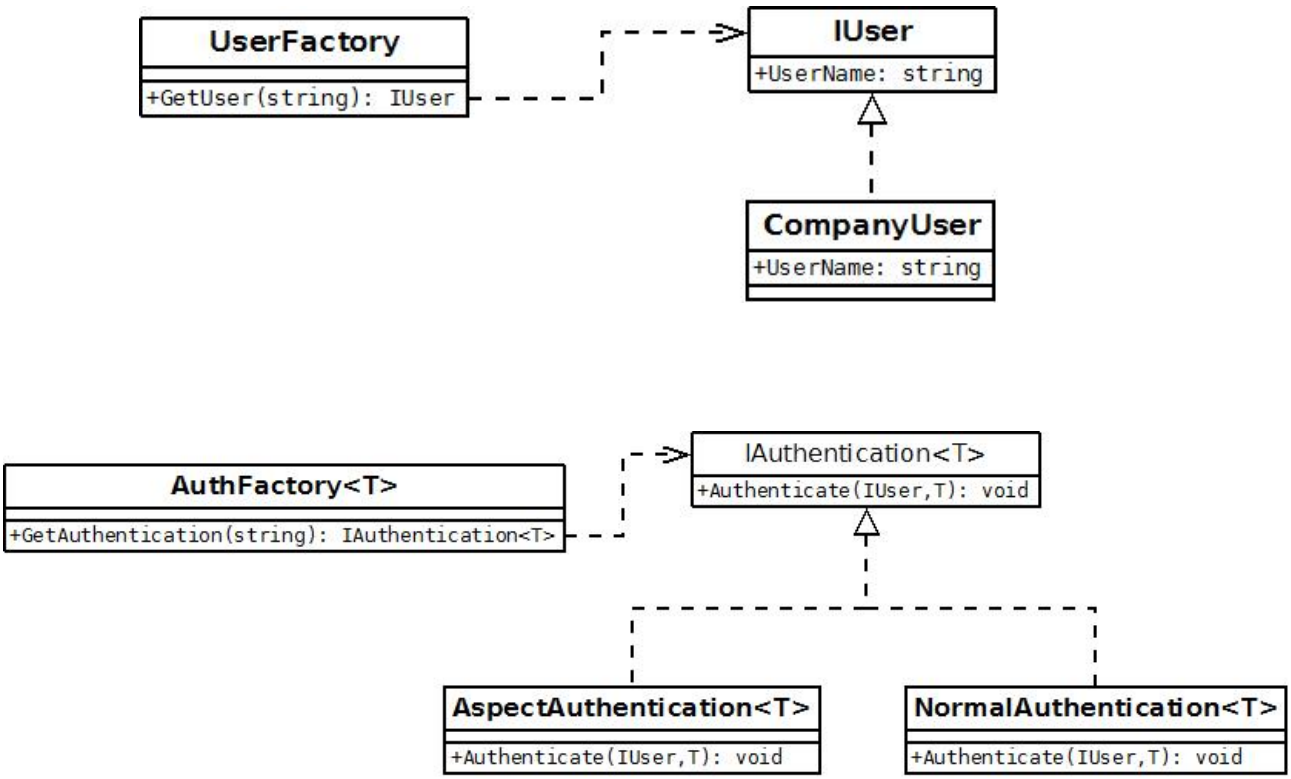
- ➔ **CompanyContext**, klasy dziedziczącej po ModelContext, czyli bazy danych reprezentującej „model”;
- ➔ **ConsoleUi** implementującej interfejs IView, czyli „view” służące do interakcji z użytkownikiem;
- ➔ **CompanyController** jako „controller” obsługujący żądania użytkownika. Obsługuje autoryzacje użytkownika.



B) **Strategy** – wykorzystana, aby umożliwić dynamiczne podstawienie różnych strategii reprezentujących algorytm pobrania konkretnych danych z modelu i ich wyświetlenie. Przydatne przy testowaniu aplikacji.



C) **Factory** – prosta fabryka parametryzowana, służąca do szybkiego, wygodnego dla klienta, tworzenia obiektów CompanyUser (implementujących IUser) oraz obiektów implementujących interfejs Authentication<T>



D) **Adapter** – wykorzystany do konwersji starego interfejsu OldAuthentication<T> (wykorzystywanego w początkowej fazie projektu) do nowego Iauthentication<T>