## Implementacja CQRS

W tym rozdziale zostanie omówiona wdrożenie i implementacja wzorca architektonicznego CQRS (Command Query Responsibility Segragation). Skupimy się tu bardziej na części praktycznej. Zagadnienia bardziej teoretyczne zostały poruszone w rozdziałach dotyczących ogólnemu opisowi zastosowanej architektury. Opis ten będzie dość ściśle powiązany z wybraną technologią, jednak budowa modułów jest na tyle reużywalna, że bez większego problemu można go zastosować do stworzenia implementacji na bazie innych języków i platform.

W ramach krótkiego przypomnienia, CQRS w swoim najbardziej podstawowym zadaniu służy do separowania operacji pobierania danych, od operacji ich zmiany bądź dodawania. Aby mieć łatwiejszą kontrolę nad tą separacją, zdefiniowaliśmy sobie odpowiednie handlery, w których zamknięta została logika danej operacji. Wyróżniamy dwa podstawowe typy handlerów których implantacja jest ściśle powiązana z jego rolą w systemie. Pierwszym typem jest QueryHandler. Ten typ handlera służy do pobierania danych z wybranego źródła bądź źródeł. Drugim typem jest CommandHandler. Jego istotą jest obsługą poleceń, tak zwanych Command, których wynikiem powinna być jakaś zmiana w systemie lub strukturze danych bądź wykonanie jakiejś akcji. Z obsługom CommandHandlerów wiąże się również pojęcie HandlerInvokera. Jego znaczenie będzie przybliżone w dalszej części rozdziału.

Praktycznie cała implementacja struktury i logiki stojącej za działaniem CQRS znajduje się na warstwie

Jak już wcześniej zostało napisane, CommandHandler służy nam w do wykonania jakiejś akcji w systemie. Według niektórych publikacji nie powinien on zwracać żadnych danych, jednak w moim mniemaniu jest to zbyt duża restrykcja. Istnieje spora grupa operacji, których wykonanie powinno zwrócić pewien rezultat do wywołującego. Wykorzystywanie do tego osobnych zapytań przy pomocy QueryHandlerów niepotrzebnie komplikowałoby przepływ danych. Pierwszym z brzegu przykładem jest chociażby wstawienie nowego rekordu do relacyjnej bazy danych, w wyniku tej operacji użytkownik powinien otrzymać id, które zostanie wygenerowane w czasie tworzenia wpisu.

Każdy CommandHandler powiązany jest z odpowiadającą mu komendą, która dostarcza mu odpowiednich danych, niezbędnych do wykonania dedykowanego mu zadania. Wszystkie Commands implementują ten sam interfejs – ICommand. Natomiast każdy CommandHandler dziedziczy po generycznym interfejsie, którego parametrami są właśnie typy implementujące ICommand. W ten sposób uzyskaliśmy połączenie między poleceniem wykonania danej czynności, oraz komponentem, który jest wstanie to polecenie wykonać. Zostanie to zaprezentowane na poniższym listingu:

….

Powyższy listing uchwycił istotę połączenia pomiędzy komendą oraz handlerem który jest do niej przypisany. Istnieją co prawda założenia, w których jeden handler może obsługiwać kilka komend, jednak w moim mniemaniu bardziej przejrzyste jest podejście, jedna komenda jedna obsługująca klasa.