Politechnika Warszawska

Podstawy projektowania podsystemów

dr hab. inż. Michał Śmiałek, prof. uczelni
dr inż. Kamil Rybiński

Wydział Elektryczny
POLITECHNIKA WARSZAWSKA

Inżynieria oprogramowania

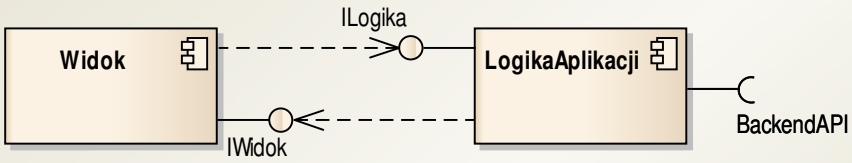
1

Kod „czysty” i „brudny”

- Kod „brudny”
  - Podatny na zmiany technologii
  - Zależy od konkretnych szkieletów technologicznych (framework)
  - Klasy zazwyczaj specjalizują klasy technologiczne
- Kod „czysty”
  - Zależy jedynie od logiki aplikacji lub dziedziny problemu
  - Stosuje klasy niezależne od stosu technologicznego
  - Stosuje klasy POJO/POCO
  - Relacje specjalizacji nie dotyczą klas technologicznych

2

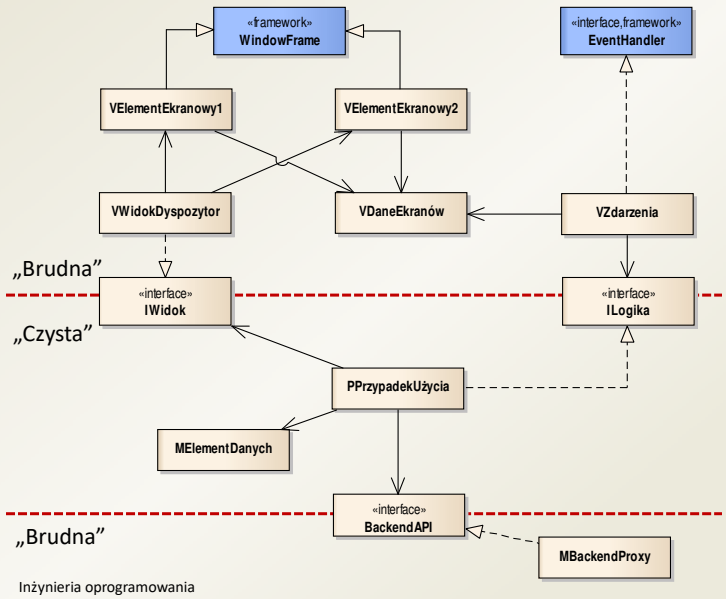
Ogólna struktura warstwy frontend



- Projektujemy na bazie projektu architektonicznego
- Warstwa widoku wsparta warstwą logiki aplikacji
- Warstwa logiki aplikacji korzysta z API do logiki dziedzinowej (backend)

3

Wzorzec dla warstwy frontend



4

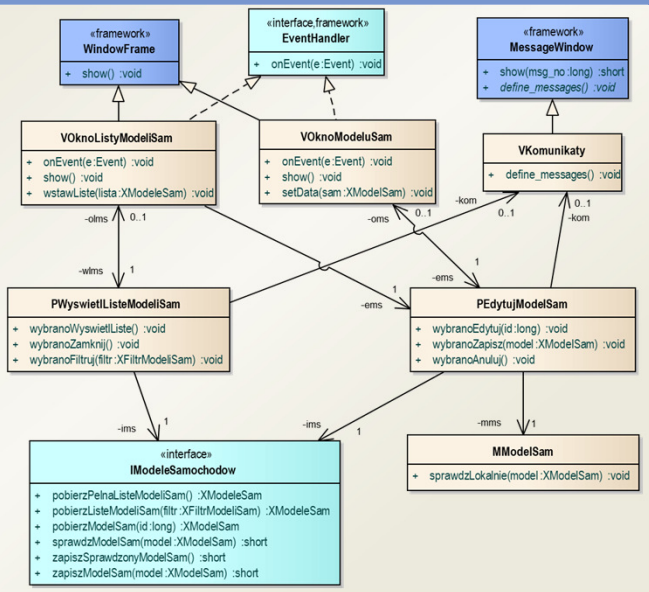
Składniki warstwy frontend

- Wyświetlanie elementów ekranowych
  - Klasa technologiczna (tu: WindowFrame)
  - Klasy specjalizujące
- Obsługa zdarzeń
  - Interfejs technologiczny (tu: EventHandler)
  - Klasy realizujące interfejs
- Komunikacja warstw widoku i logiki aplikacji
  - Możliwe zastosowanie interfejsów (dobra separacja)
  - Sterowanie wyświetlaniem widoków (tu: IWidok)
  - Przekazywanie zdarzeń do realizacji (tu: ILogika)
- Logika aplikacji
  - Korzysta z interfejsów udostępnianych przez widok i logikę dziedzinową
- Dostęp do warstwy backend
  - Zastosowanie klasy „proxy” (zastępnik) dla zdalnych wywołań

5

Przykładowy projekt warstwy frontend

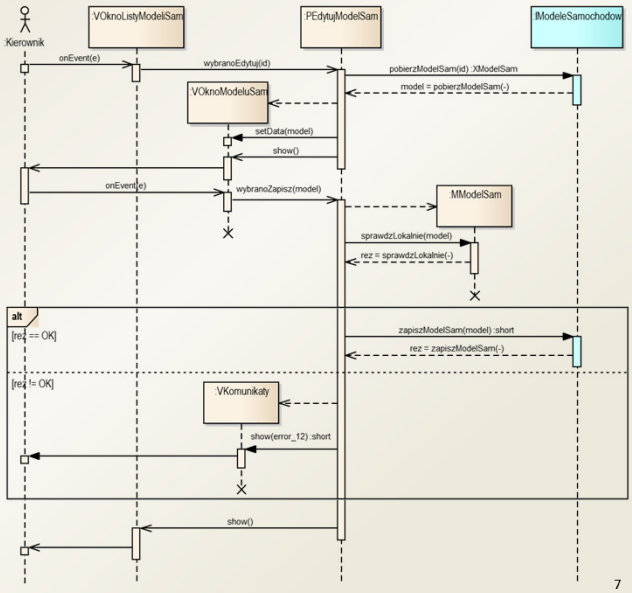
- Pominięte interfejsy między warstwami frontendu
- Klasy logiki aplikacji odpowiadają przypadkom użycia
- Dostęp do backendu poprzez interfejs



6

Działanie warstwy frontend

- Rozpoczyna komunikat od aktora
- Działaniem steruje obiekt w warstwie logiki aplikacji
  - Odwołania do logiki dziedzinowej
  - Lokalna walidacja danych
  - Odwołania do widoku



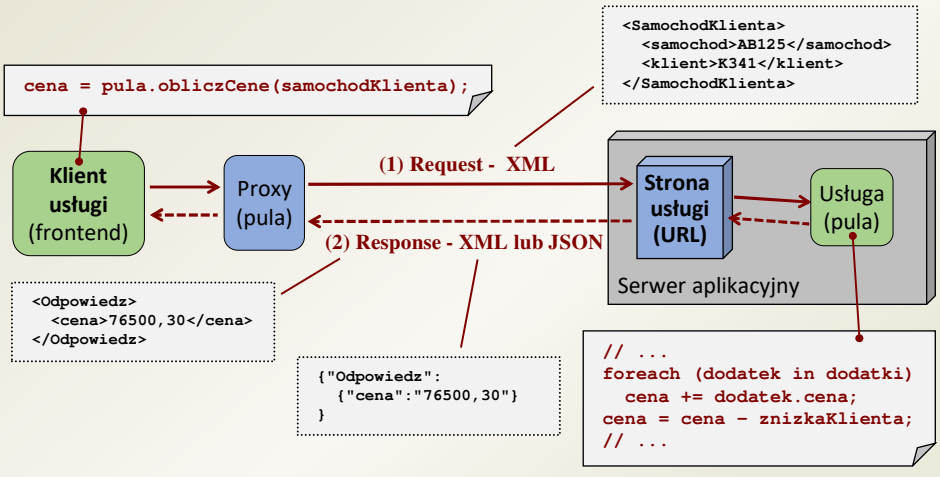
Wzorzec MVP

- Model – View – Presenter
  - Realizuje architekturę warstwową (patrz poprzedni wykład)
  - Klasy „V” – widok
  - Klasy „P” – prezenter (logika aplikacji)
  - Klasy „M” – model (logika dziedziny)
- Model MVP jest oparty na wzorcu MVC
  - Model – View – Controller

Technologie warstwy frontend

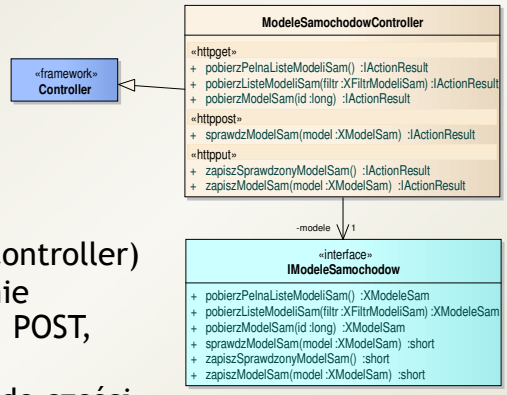
- Technologie oparte na języku JavaScript
  - Wariant obiektowy – język TypeScript
  - Obecnie najbardziej popularna grupa technologii
  - Najpopularniejsze: React, Vue, Angular (przetwarzanie po stronie klienta), Node (przetwarzanie po stronie serwera)
- Technologie oparte na języku Java
  - GWT, Vaadin – kod wykonywany po stronie serwera
  - Tłumaczenie na język JavaScript
  - JavaFX, Swing – kod wykonywany po stronie klienta
- Inne technologie
  - Blazor – środowisko .NET (język C#)
  - Dart – środowisko oparta a języku o tej samej nazwie

Zdalne wywołania procedur



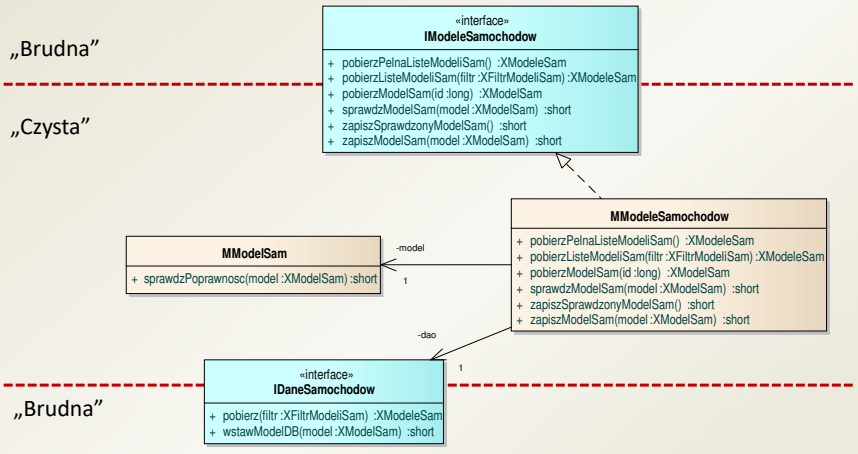
- Zastosowanie wzorca „proxy” do realizacji wywołań zdalnych
  - Można stosować w różnych technologiach (RPC, REST, SOAP)

Obsługa wywołań zdalnych



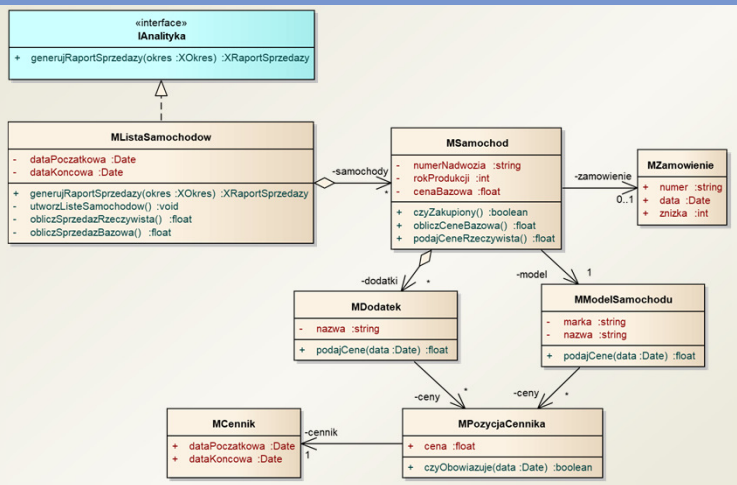
- Klasa kontrolera API
  - Specjalizuje klasę technologiczną (tu: Controller)
  - Obsługuje odpowiednie zapytania HTTP (GET, POST, PUT)
  - Korzysta z interfejsu do części „czystej”
  - Zgodna z projektem architektonicznym

Realizacja usług backend



- Implementacja interfejsu
  - Tu: wzorzec „skrypt transakcyjny”
  - Wyraźne oddzielenie kodu „czystego” (logiki) od kodu technologicznego

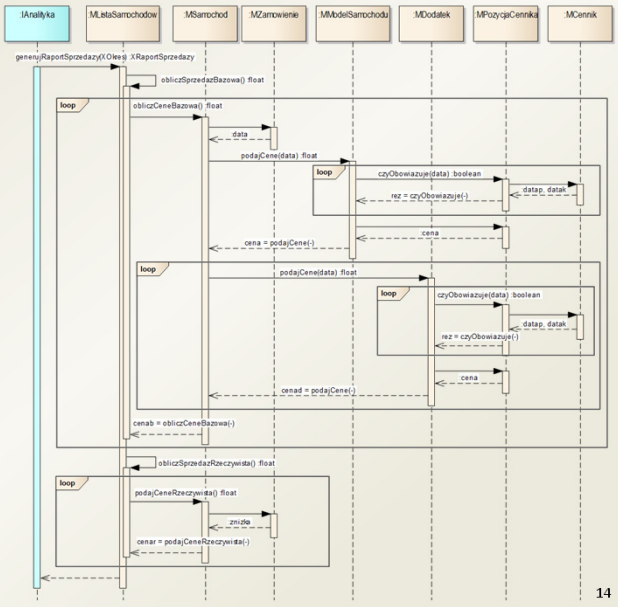
Wzorzec „model dziedziny”



- Logika rozproszona między klasy zgodne z modelem opisującym dziedzinę problemu

Działanie komponentów backend

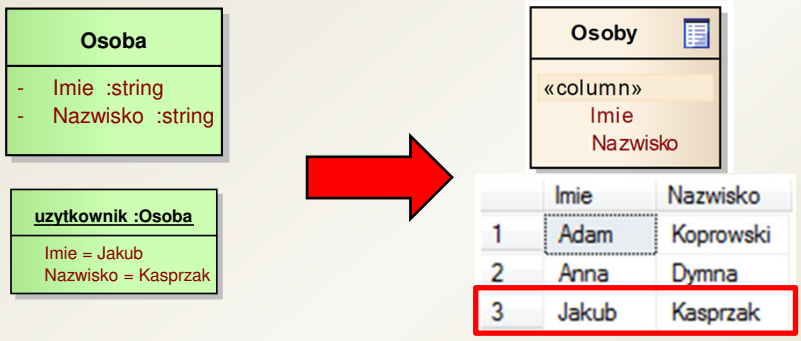
- Realizacja konkretnej operacji usługi
  - Podział zadań na operacje prywatne
  - Implementacja algorytmów przez kilka obiektów klas składowych



Projektowanie baz danych

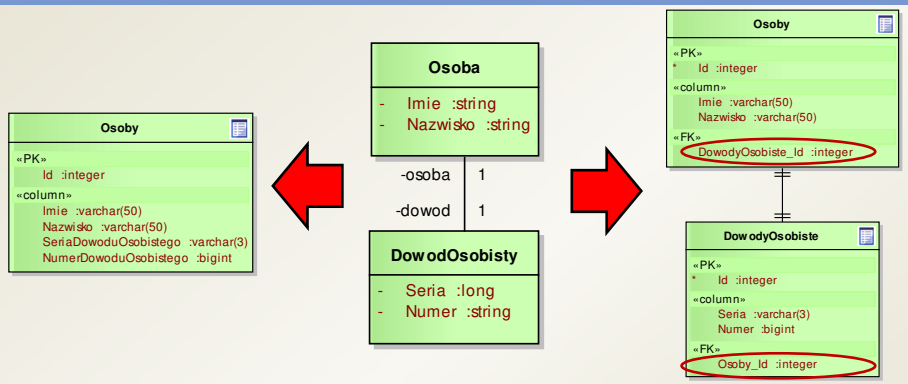
- Bazy danych relacyjne
  - Teoria relacyjna opracowana w latach 70.
  - Podstawowy język zapytań: SQL
  - Struktura bazy danych: tabele, kolumny, wiersze, powiązania
- Bazy danych nierelacyjne (NoSQL)
  - „Not only SQL” – czasami używają podzbioru języka SQL
  - Dane często przechowywane w postaci dokumentów
  - Struktura bazy danych często identyczna jak model danych (UML)
- Problem: przejście z modelu kod na strukturę bazy danych
  - Głownie występuje dla baz relacyjnych

Tabele relacyjne



- Klasy obiektów trwałych → tabele bazy danych
  - Obiekty odpowiadają wierszom w tabeli
  - Kolumny tabeli odpowiadają atrybutom klas

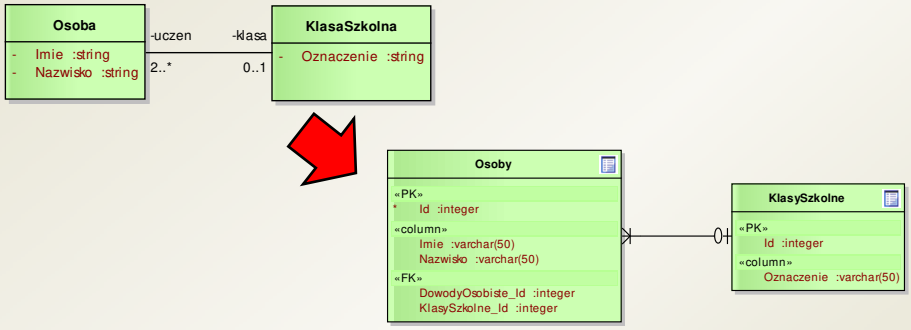
Powiązania między tabelami 1-1



- Asocjacje 1-1 w modelu obiektowym
  - Tłumaczymy w jedną połączoną tabelę
  - Tłumaczymy w dwie tabele z powiązaniem 1-1
  - Dodatkowo tworzymy klucze główne i klucze obce

17

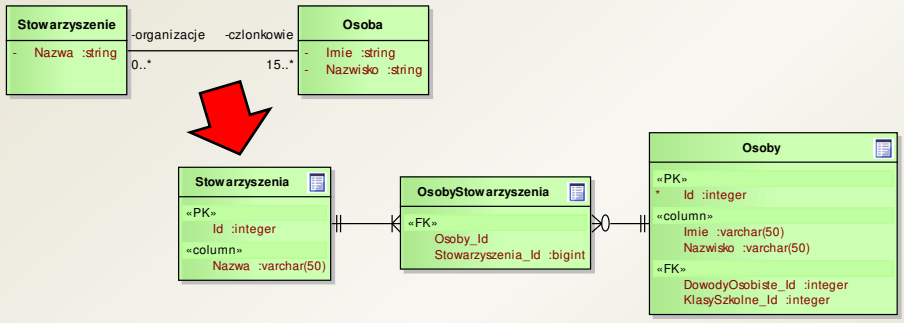
Powiązania między tabelami 1-N



- Asocjacje 1-N w modelu obiektowym
  - Tłumaczymy w dwie tabele
  - Klucz obcy umieszczamy w tabeli z kronością N

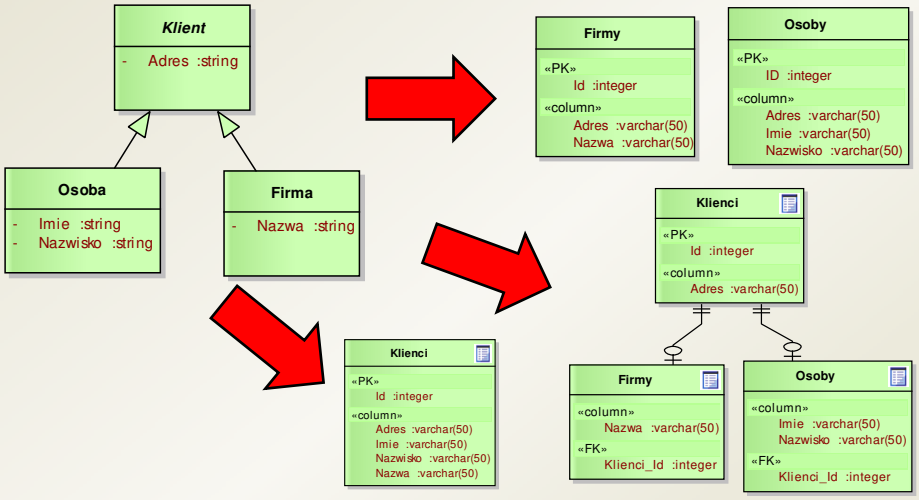
18

Powiązania między tabelami N-N



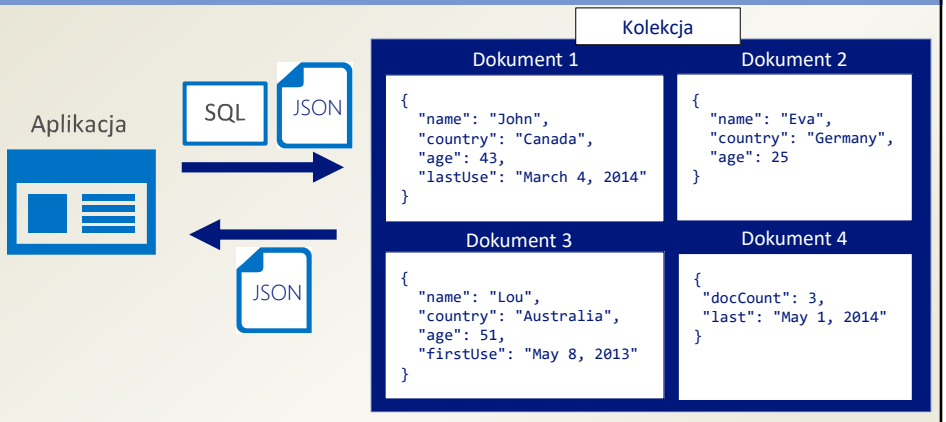
- Asocjacje N-N w modelu obiektowym
  - Konieczne utworzenie tabeli dodatkowej
  - Dodatkowa tabela zawiera klucze obce do dwóch pozostałych tabel

Realizacja relacji generalizacji



- Trzy warianty realizacji generalizacji

Struktura bazy nierelacynej



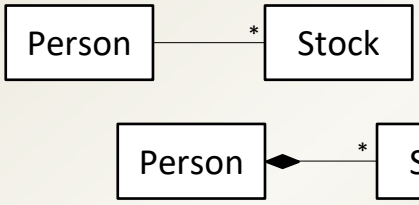
- Dane przechowywane w formie dokumentów tekstowych
  - Pliki JSON (JavaScript Object Notation)
  - Dokumenty zawarte w kolekcjach (pliki tego samego typu)
  - Zapytania formułowane np. w języku podobnym do SQL

21

Realizacja relacji 1-N

```
Person document:
{
    "id": "1",
    "firstName": "Thomas",
    "lastName": "Andersen",
    "holdings": [
        { "numberHeld":  100, "stockId": 1},
        { "numberHeld":  50, "stockId": 2}
    ]
}

Stock documents:
{
    "id": 1,
    "symbol": "zaza",
    "open": 1,
    "high": 2,
    "low": 0.5,
    "vol": 11970000,
    "mkt-cap": 42000000,
    "pe": 5.89
},
{
    "id": 2,
    "symbol": "xcxc",
    "open": 89,
    "high": 93.24,
    "low": 88.87,
    "vol": 2970200,
    "mkt-cap": 1005000,
    "pe": 75.82
}
```




```
{
    "id": "1",
    "firstName": "Thomas",
    "lastName": "Andersen",
    "holdings": [
        {
            "numberHeld": 100,
            "stock": { "symbol": "zaza", "open": 1, "high": 2, "low": 0.5 }
        },
        {
            "numberHeld": 50,
            "stock": { "symbol": "xcxc", "open": 89, "high": 93.24, "low": 88.87 }
        }
    ]
}
```

- Różne rozwiązania w zależności od rodzaju relacji (np. agregacja)

22

Realizacja relacji N-N


```
Author documents:
{"id": 1, "name": "Thomas Andersen", "books": [1, 2, 3]}
{"id": 2, "name": "William Wakefield", "books": [1, 4]}

Book documents:
{"id": 1, "name": "DocumentDB 101", "authors": [1, 2]}
{"id": 2, "name": "DocumentDB for RDBMS Users", "authors": [1]}
{"id": 3, "name": "Learn about Azure DocumentDB", "authors": [1]}
{"id": 4, "name": "Deep Dive in to DocumentDB", "authors": [2]}
```

- Prosta realizacja podobna do relacji 1-N
  - Umieszczenie referencji w obydwu dokumentach
  - Podobna do realizacji w kodzie obiektowym