

Load Balancer

Rafał Łukosz
Maciej Adamus
Dawid Małecki

1. Opis projektu	2
2. Funkcjonalności systemu	2
3. Stos Technologiczny	2
4. Architektura	3
4.1. Architektura Logiczna	3
4.2. Architektura Fizyczna	4
5. Wykorzystane wzorce projektowe	5
5.1. Unit of Work	5
5.2. Strategy	5
5.3. State	6
5.4. Factory	6

1. Opis projektu

Celem projektu było stworzenie systemu równoważenia obciążenia ruchu w bazie danych, umożliwiającego operacje CRUD (Create, Read, Update, Delete) w wielu zsynchronizowanych bazach danych jednocześnie. Zapytania typu Select wykonywane są tylko na jednej bazie, wskazanej przez algorytm Load Balancingu. Ma to na celu zmniejszenie obciążenia i zmniejszenie średniego czasu wykonywania zapytań.

W przypadku utraty połączenia z bazą danych, zapytania są buforowane i wysyłane po ponownym ustanowieniu połączenia.

System zakłada, że w trakcie inicjalizacji każda baza danych jest albo pusta, albo ma identyczny stan. Nie jest w stanie zsynchronizować baz danych, które różnią się podczas fazy uruchamiania systemu

2. Funkcjonalności systemu

System pozwala na

- Utworzenie LoadBalancera, który posiada zdefiniowane połączenia do różnych baz danych, stworzonych przy użyciu tej samej technologii
- Synchronizację danych pomiędzy różnymi bazami danych
- Wykonywanie zapytań typu Select na jednej z dostępnych baz
- Zmianę parametrów działania Load Balancera, takich jak rodzaj wykorzystywanego algorytmu równoważenia obciążenia

3. Stos Technologiczny

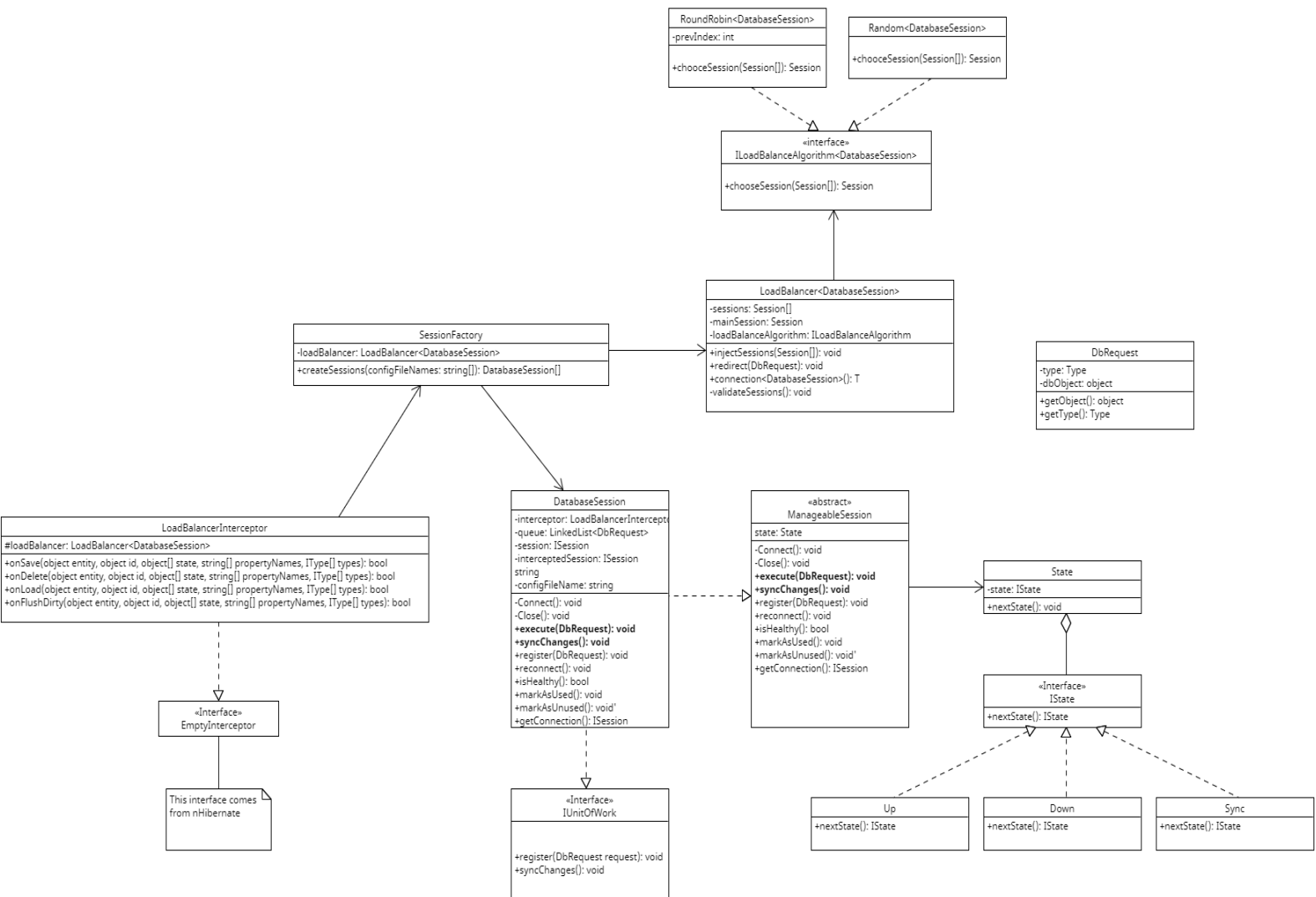
W projekcie skorzystaliśmy z:

- C#
- .NET 8.0
- Docker
- PostgreSQL

4. Architektura Systemu

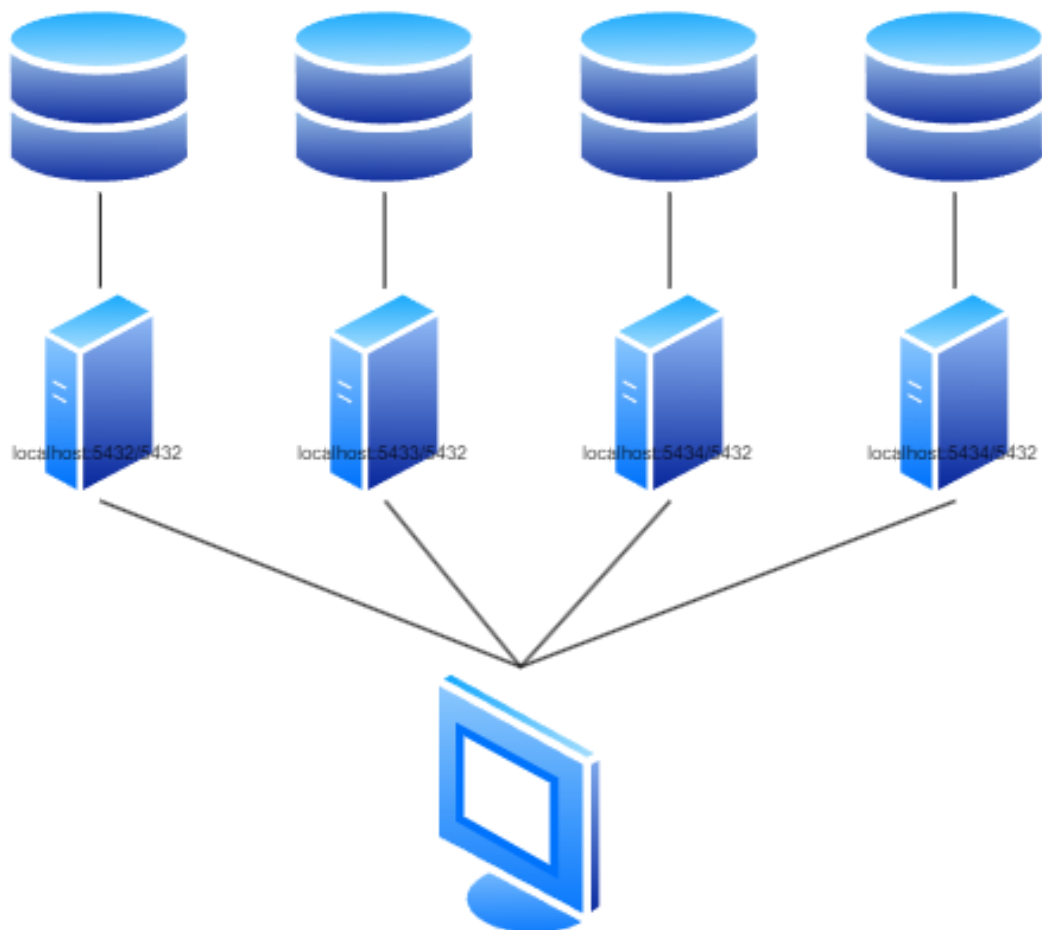
4.1. Architektura Logiczna

Na poniższym diagramie UML zaprezentowane są klasy znajdujące się w projekcie wraz z ich powiązaniami. Na diagramie uwzględniono również metody i atrybuty poszczególnych klas.



4.2. Architektura Fizyczna

Architekturę fizyczną można opisać jako system składający się z 4 serwerów bazodanowych. Do tych serwerów łączy się urządzenie klienta z odpowiednio skonfigurowanym Load Balancerem, który zarządza dostępem do każdej z baz.



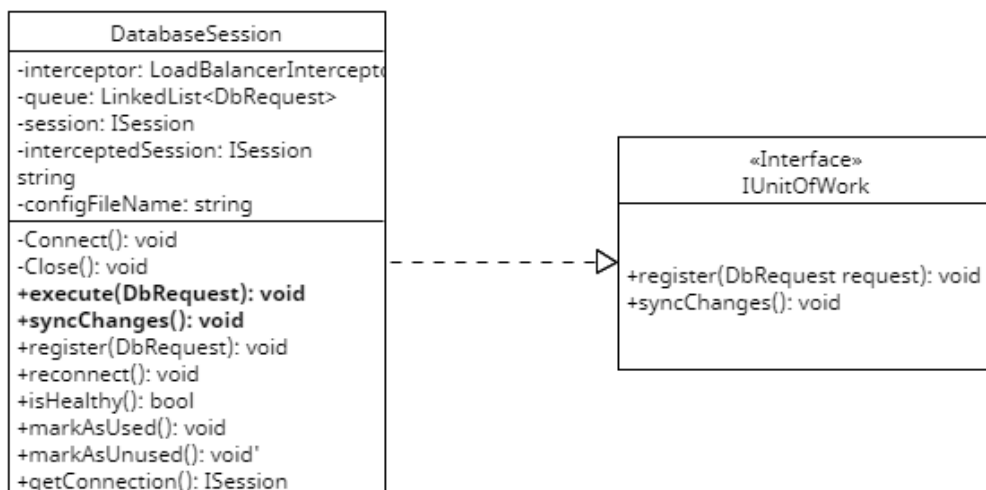
5. Wykorzystane wzorce projektowe

5.1. Unit of Work

Wzorzec odpowiada za utrzymanie spójności danych pomiędzy różnymi serwerami bazodanowymi. Dzięki temu można wykonywać zapytania typu Insert, Update i Delete na serwerach, które nie są obecnie aktywne. Dane są przechowywane w pamięci podręcznej i wysyłane do serwera, kiedy staje się on ponownie aktywny.

Interfejs deklaruje dwie metody:

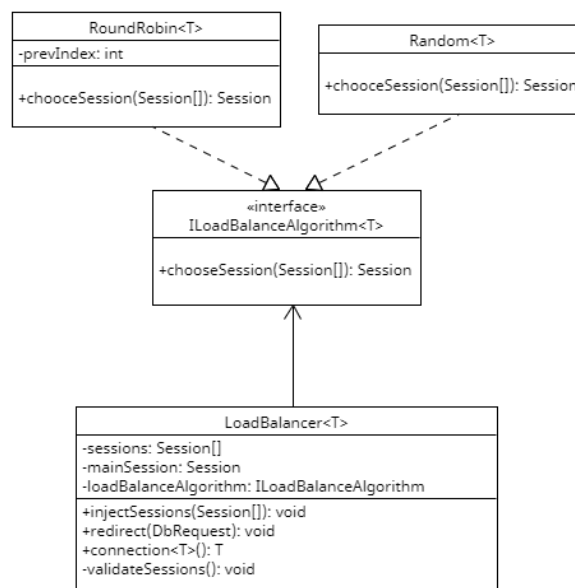
- **register(DbRequest)** : metoda rejestrująca zapytanie w pamięci podręcznej
- **syncChanges()** : metoda wysyłająca dane zapytań z bufora do serwera



5.2. Strategy

Wzorzec umożliwia łatwą podmianę algorytmu odpowiedzialnego za wybór serwera bazodanowego, do którego zostanie wysłane zapytanie typu Select.

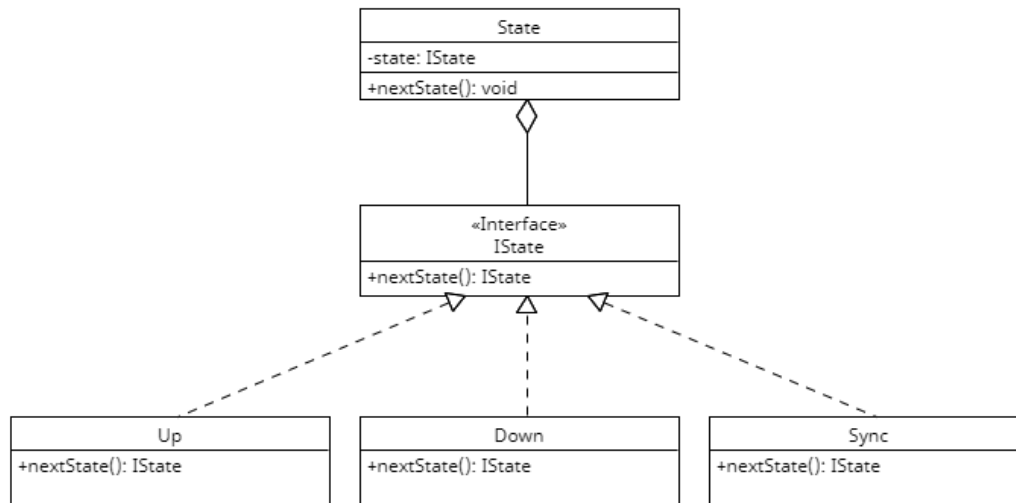
Interfejs deklaruje metodę **chooseSession(Session[])** odpowiedzialną za wybór bazy danych i zwrócenie referencji do niego.



5.3. State

Wzorzec umożliwia zmianę stanu bazy danych, co wiąże się ze zmianą trybu działania systemu. Gdy baza danych jest w stanie Down/Sync to zapytania przechowywane są w buforze i egzekwowane przy zmianie stanu na Up. Stany są zapętlone w następujący sposób: Down -> Sync -> Up -> Down

Interfejs deklaruje metodę **nextState()** zwracającą kolejny stan.



5.4. Factory

Wzorzec umożliwia elastyczne tworzenie sesji bazy danych, przy jednoczesnym odseparowaniu klienta od szczegółów implementacyjnych klas sesji. Klasa **SessionsFactory** jest odpowiedzialna za abstrakcję procesu tworzenia sesji i zarządzanie zależnościami między obiektami. Metoda **createSessions(configFileNames: string[])** jest odpowiedzialna za dynamiczne tworzenie sesji baz danych na podstawie dostarczonych plików konfiguracyjnych.

