DesignPatterns

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Autorzy

Lukasz Łabuz	luklabuz@student.agh.edu.pl
Jakub Głowacki	jglowacki@student.agh.edu.pl
Hubert Asztabski	asztabski@student.agh.edu.pl
Michał Koczkodaj	mkoczkodaj@student.agh.edu.pl

24 stycznia 2024

Spis treści

1 Opis projektu **2**

2 Funkcjonalności systemu **3**

3 Architektura **4**
3.1 Architektura logiczna aplikacji 4
3.1.1 Pojedyncze diagramy klas 4
3.1.2 Całościowy diagram klas 7
3.2 Architektura fizyczna aplikacji 9
3.3 Wzorce Projektowe 10
3.3.1 Strategy 10
3.3.2 Unit of Work 11
3.3.3 Bridge (most) 12

4 Stos technologiczny **13**

Rozdział 1

Opis projektu

Celem naszego projektu było zaimplementowanie Load Balancera na warstwie ORM który miał służyć jako warstwa pośrednia pomiędzy aplikacją kliencką a bazami danych.
Projekt udało nam się zrealizować dzięki zastosowaniu:

- Interceptora Hibernate,
- Abstrakcji na obiekcie Session z bibloteki Hibernate,
- Użyciu wzorców projektowych takich jak:
    1. Most,
    2. Unit of Work,
    3. Strategy (Load balancing strategy),

Wynikiem naszej pracy jest system składający się z:

- Aplikacji klienckiej zintegrowanej z Load Balancerem,
- Baz danych postawionych na kontenerach dockerowych.

Rozdział 2

Funkcjonalności systemu

Nasz system pozwala na:

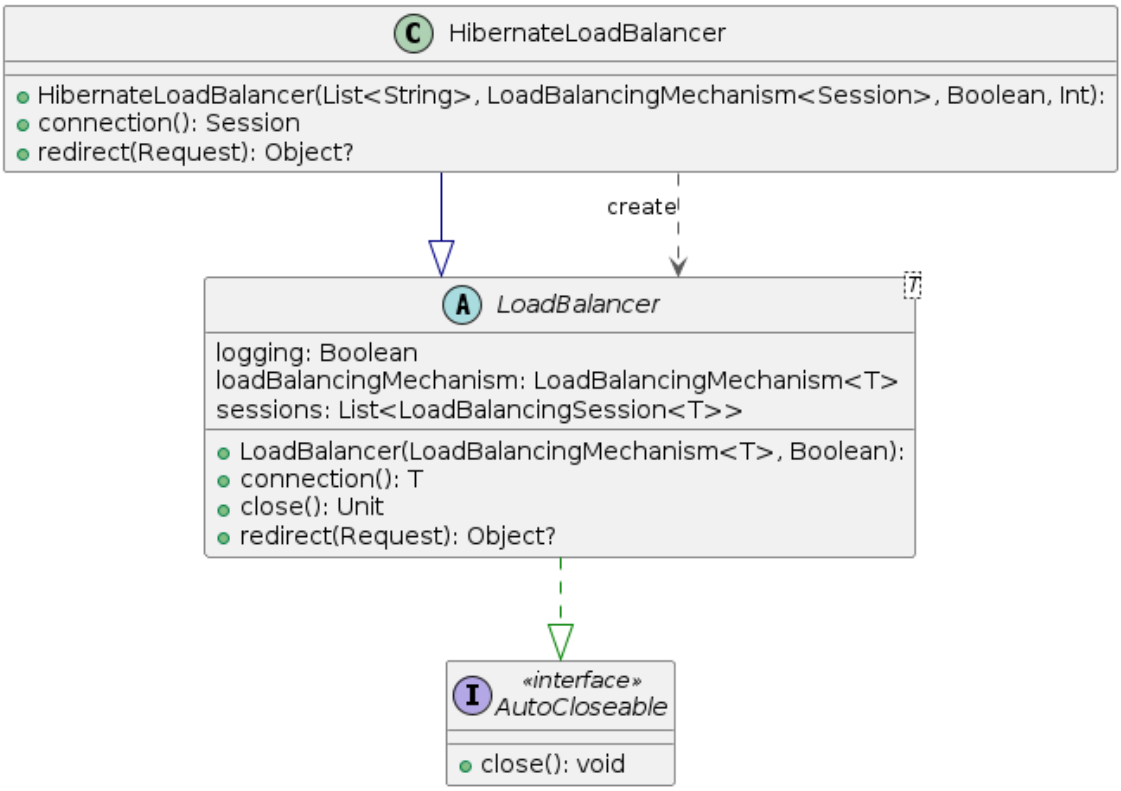
1. Wykonanie operacji CRUD,
    - Operacje które nie zmieniają stanu bazy danych wykonają się zgonie z zdefiniowaną strategią na jednej z baz,
    - Operacje zmieniające stan bazy danych wykonają się na wszystkich podłączonych do aplikacji baz danych.
2. Synchronizacje baz danych pomiędzy sobą,

Rozdział 3

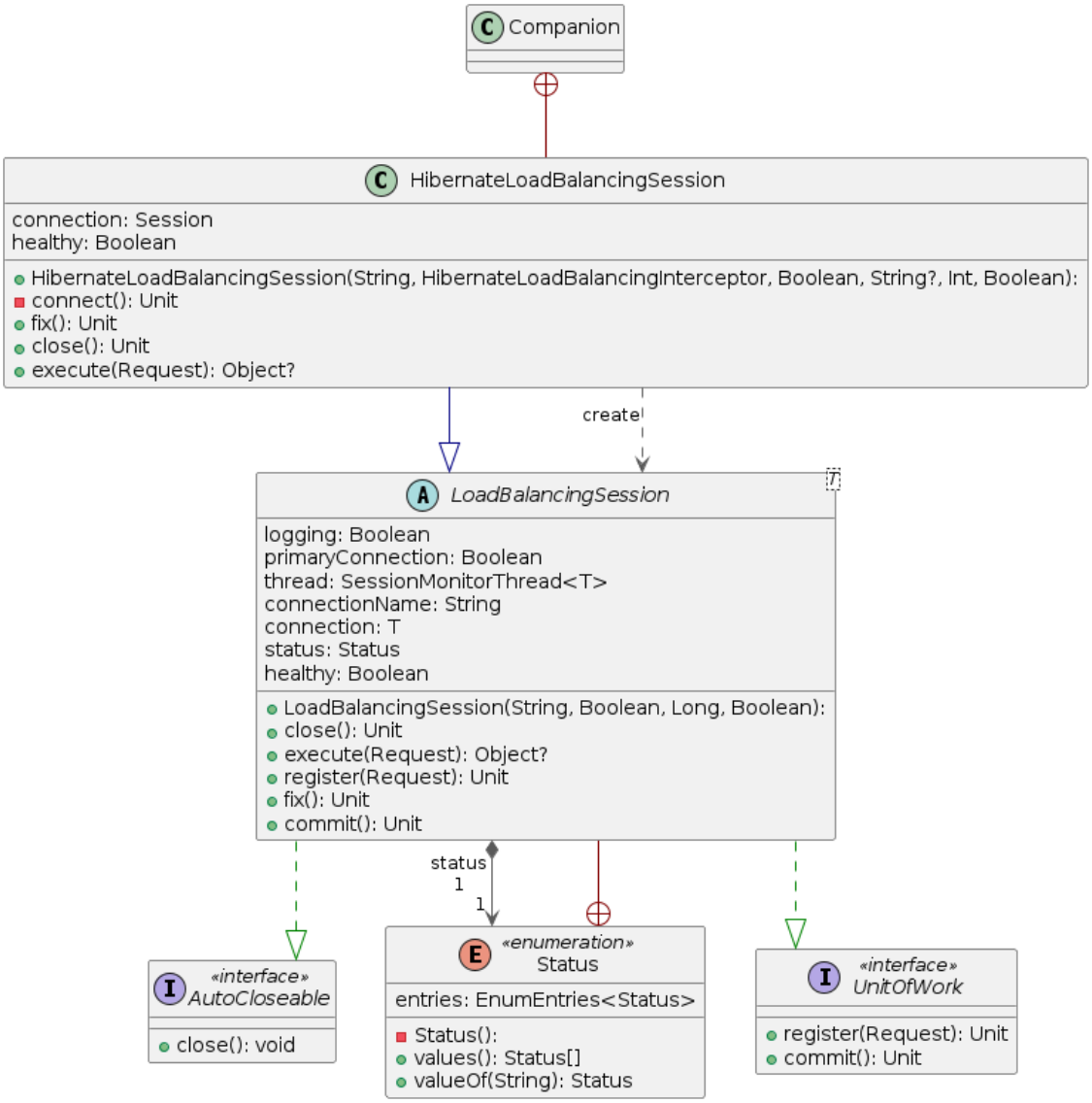
Architektura

3.1 Architektura logiczna aplikacji

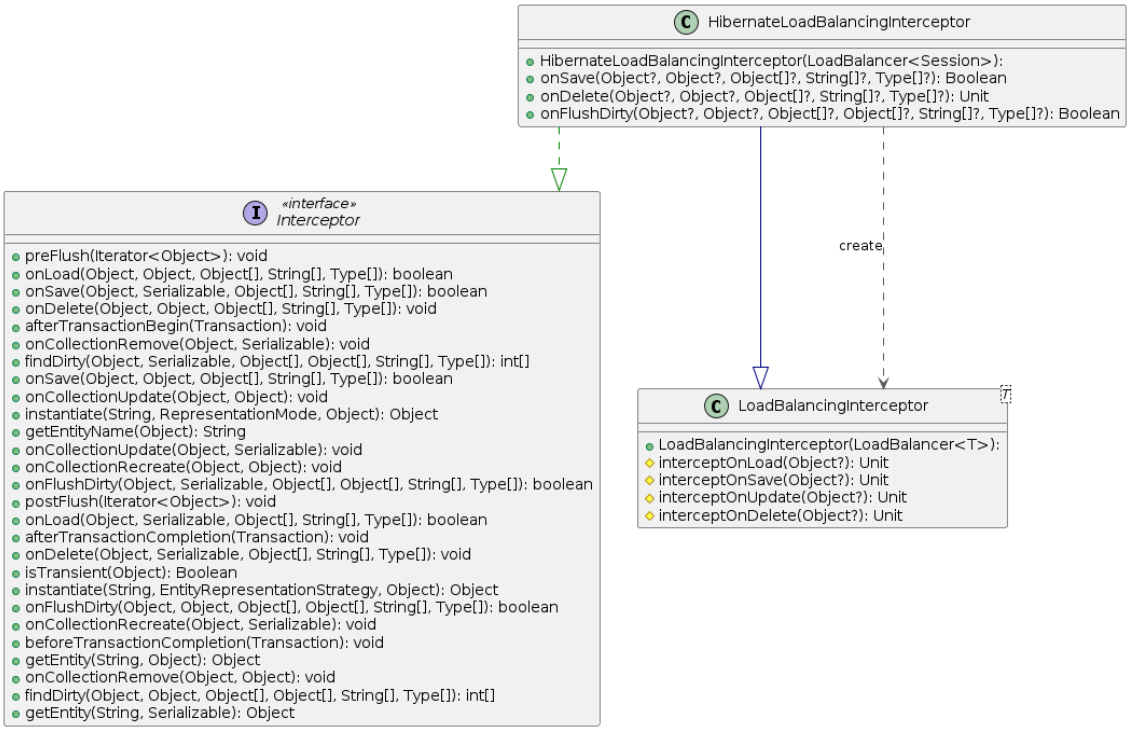
3.1.1 Pojedyncze diagramy klas



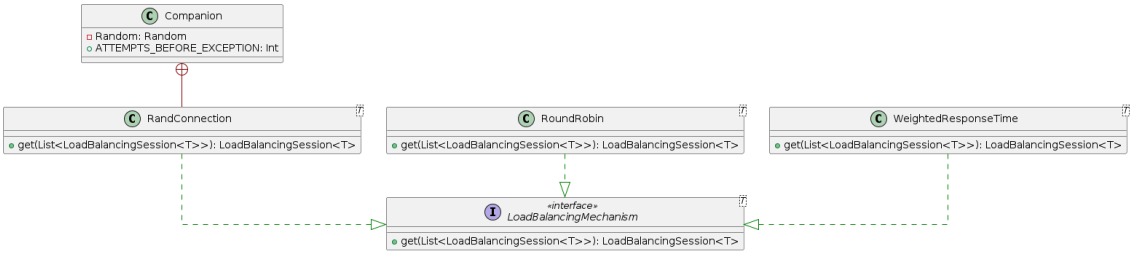
Rysunek 3.1: Diagram klasy HibernateLoadBalancer



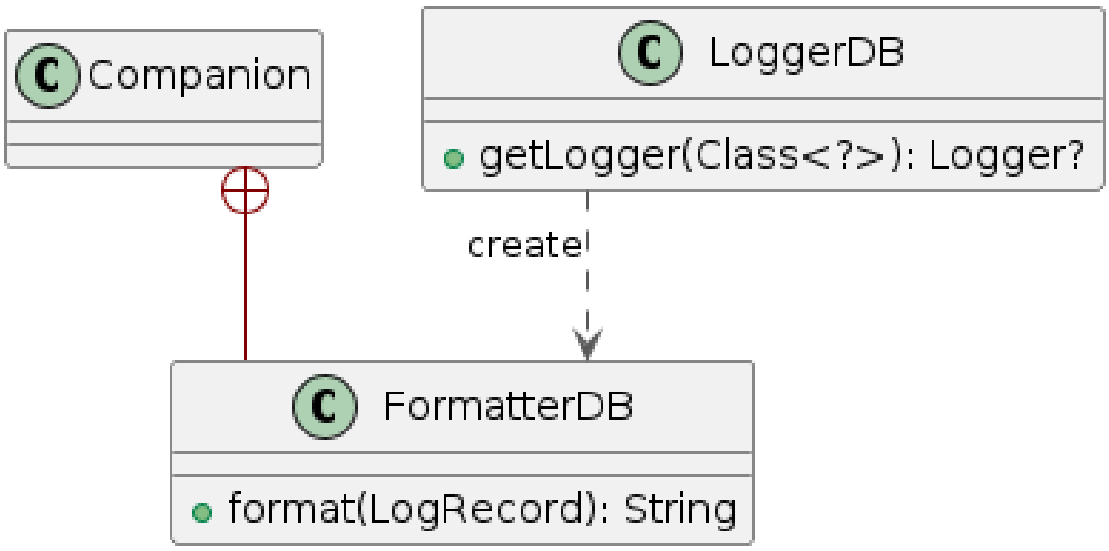
Rysunek 3.2: Diagram klasy HibernateLoadBalancingSession



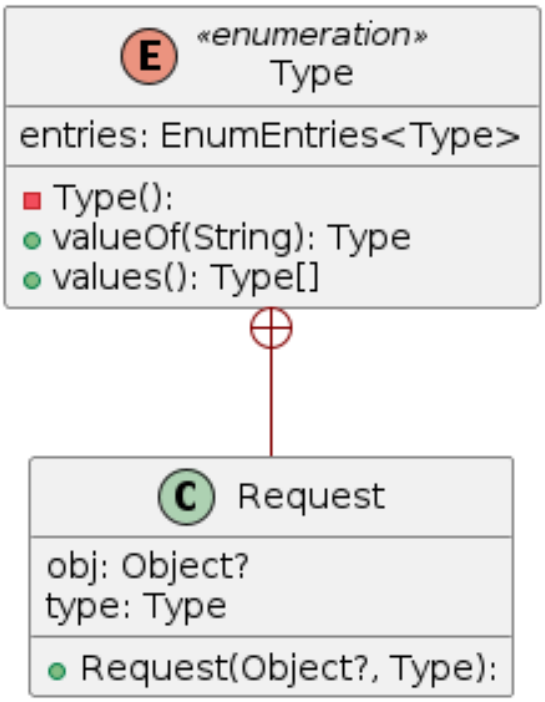
Rysunek 3.3: Diagram klasy HibernateLoadBalancingInterceptor



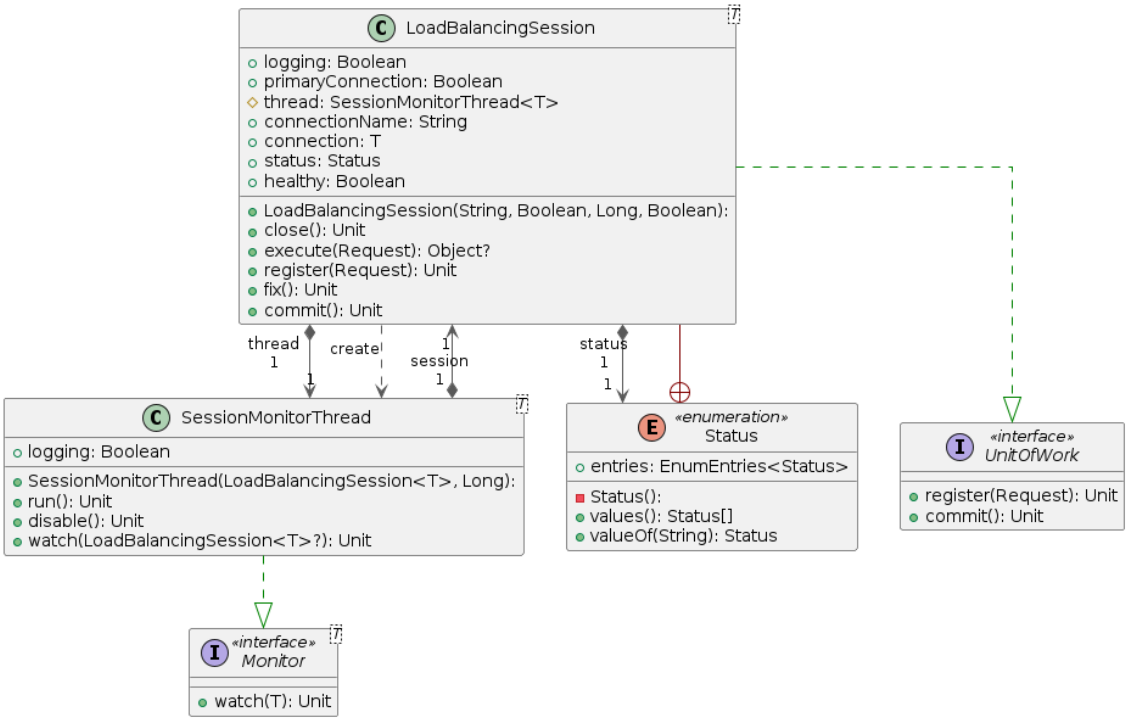
Rysunek 3.4: Diagram klas LoadBalancingMechanism



Rysunek 3.5: Diagram Loggera

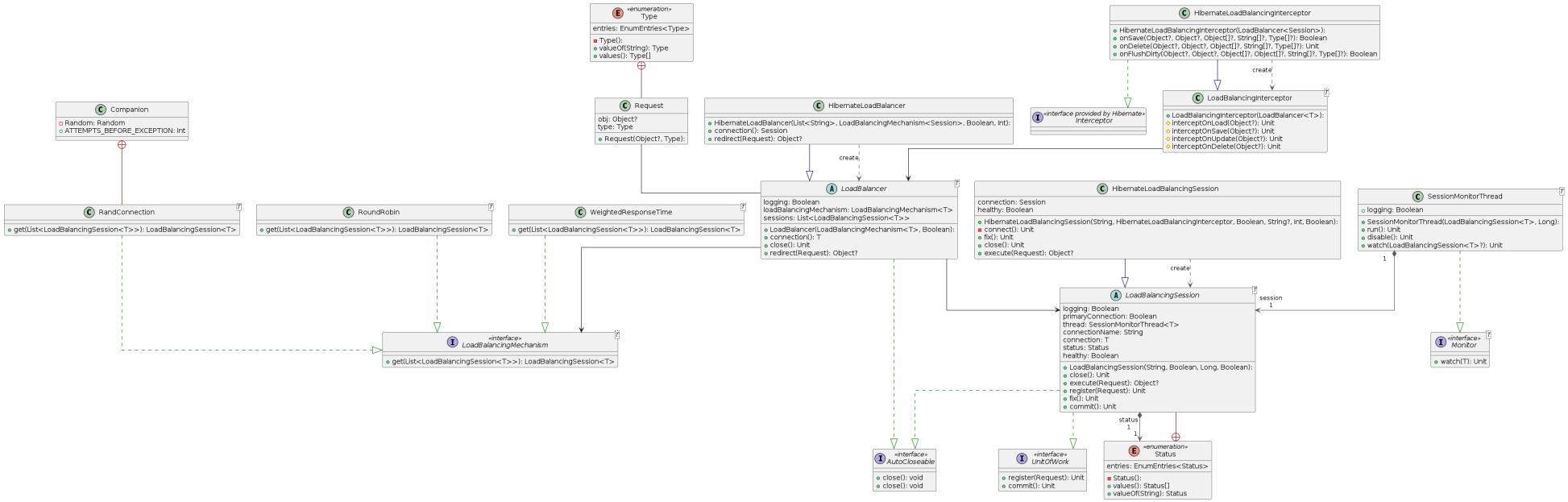


Rysunek 3.6: Diagram typu Request



Rysunek 3.7: Diagram klasy abstrakcyjnej Session

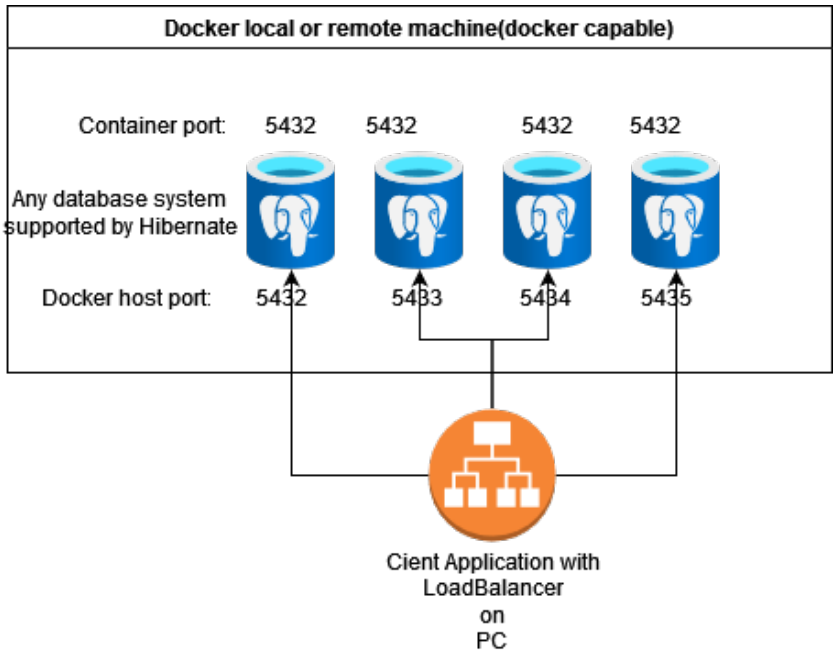
3.1.2 Całościowy diagram klas



Rysunek 3.8: Diagram klasy abstrakcyjnej Session

3.2 Architektura fizyczna aplikacji

Architektura fizyczna jest to system składający się z co najmniej 4 serwerów bazodanowych i aplikacji klienckiej która służy jako loadbalancer.

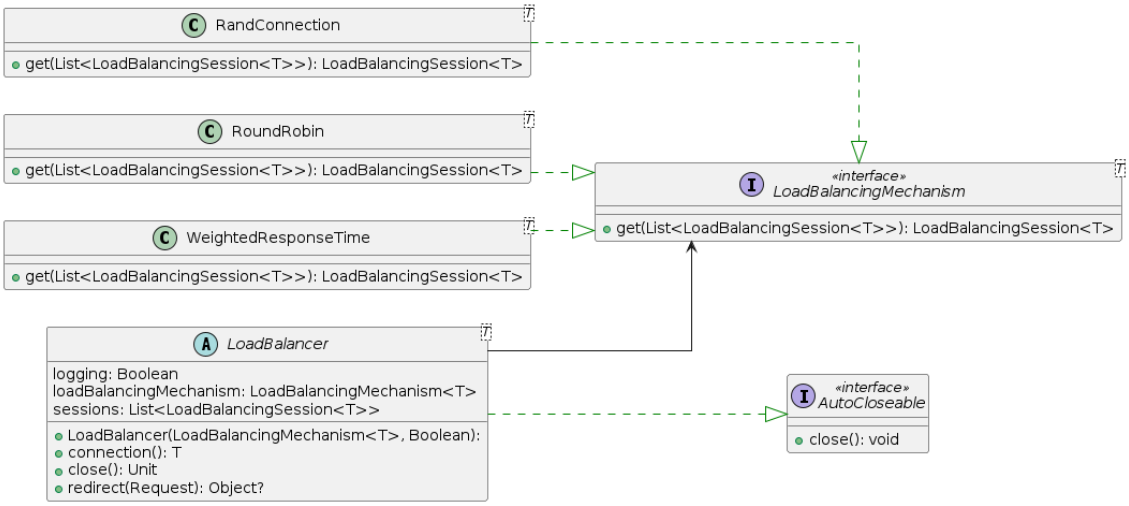


Rysunek 3.9: Przykładowa Architektura Fizyczna

3.3 Wzorce Projektowe

3.3.1 Strategy

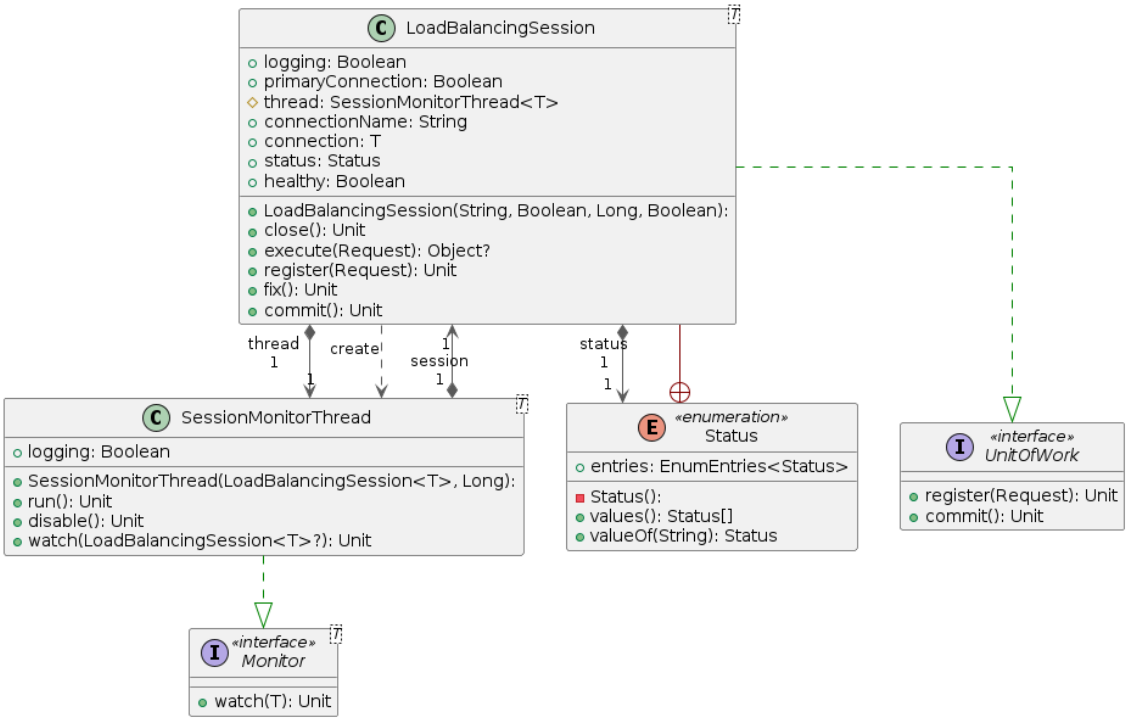
Strategia jest wzorcem projektowym umożliwiającym łatwą zamianę algorytmu odpowiedzialnego za wybieranie serwera, do którego zostanie wysłane zapytanie typu select.
W praktyce oznacza to, że można elastycznie zmieniać sposób, w jaki system decyduje, któremu serwerowi przekazać konkretne zapytanie select, bez konieczności modyfikowania samego zapytania czy jego obsługi.
Wzorzec ten pozwala na uniezależnienie wyboru serwera od reszty systemu, co sprawia, że możemy z łatwością dostosować strategię wyboru serwera do zmieniających się warunków lub wymagań systemu.
W naszym projekcie występuje poniższa implementacja.



Rysunek 3.10: Strategy

3.3.2 Unit of Work

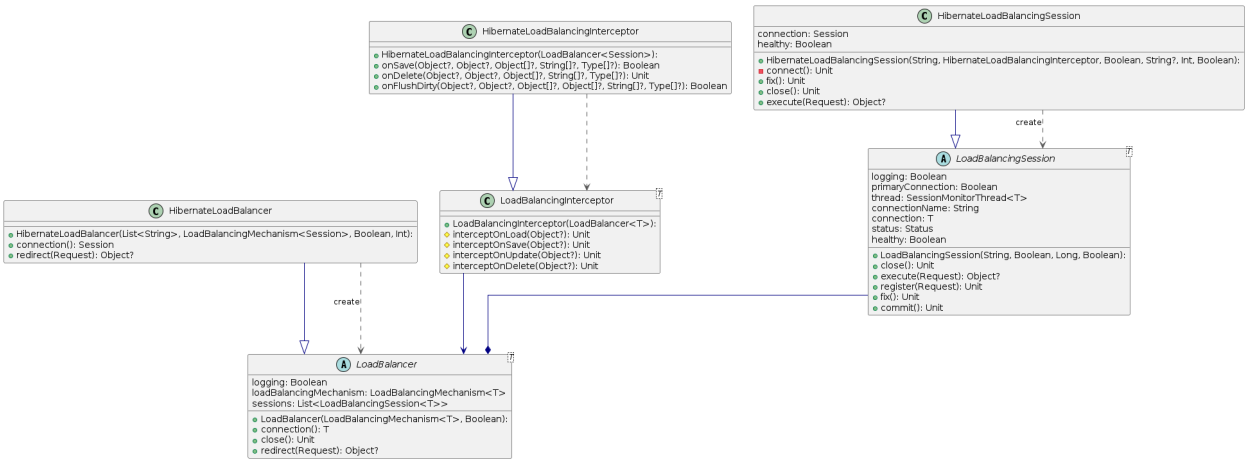
Unit of Work to wzorzec projektowy, który zarządza spójnością danych na różnych serwerach.
Jego głównym celem jest umożliwienie wykonywania operacji typu insert, update i delete na serwerach, które w danym momencie nie są aktywne.
Działa to poprzez tymczasowe przechowywanie operacji na danych w pamięci podręcznej, a następnie ich wysyłanie do serwera, gdy ten stanie się ponownie aktywny.
W naszym projekcie występuje poniższa implementacja.



Rysunek 3.11: Unit of Work

3.3.3 Bridge (most)

Bridge (most) to wzorzec projektowy, który umożliwia rozdzielenie jednej skomplikowanej hierarchii dziedziczenia na dwie osobne hierarchie: jedną dla abstrakcji i drugą dla implementacji.
Wzorzec ten wykorzystuje kompozycję zamiast dziedziczenia, co prowadzi do elastyczniejszej struktury kodu.
W skomplikowanych hierarchiach dziedziczenia często trudno jest utrzymać klarowność i elastyczność kodu. Most rozwiązuje ten problem, pozwalając na oddzielenie abstrakcji od implementacji.
W naszym projekcie występuje poniższa implementacja.



Rysunek 3.12: Bridge

Rozdział 4

Stos technologiczny

Projekt został stworzony z pomocą poniższych technologi.

- Kotlin jako język implementacji,
- Gradle jako narzędzie do tworzenia i budowania projektów,
- Hibernate core (6.4.1) final (obiekty Interceptor, Session)
- PostgreSQL jako system bazodanowy,
- Docker jako narzędzie do uruchomienia niezależnych od siebie baz danych.