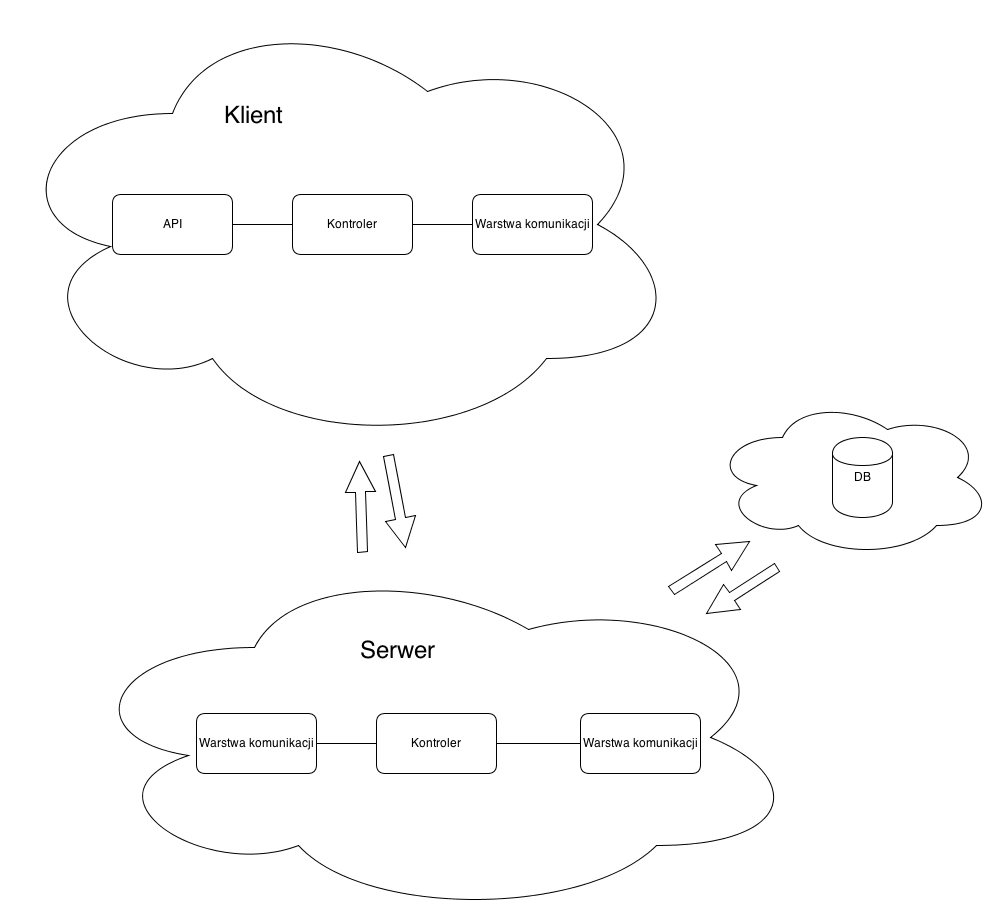
Biblioteka dostępowa do bazy danych

## Cel

Implementacja klienta oraz serwera udostępniające połączenie i wykonywanie podstawowych operacji na zdalnej bazie danych. Serwer będzie aplikacją umieszczoną na maszynie, na której znajduje się baza danych, natomiast klient to biblioteka udostępniająca operacje dostępowe po stronie użytkownika. Projekt wymaga zaprojektowanie protokołu umożliwiającego wymianę informacji pomiędzy stroną kliencką, a serwerem.

## Podział na moduły:



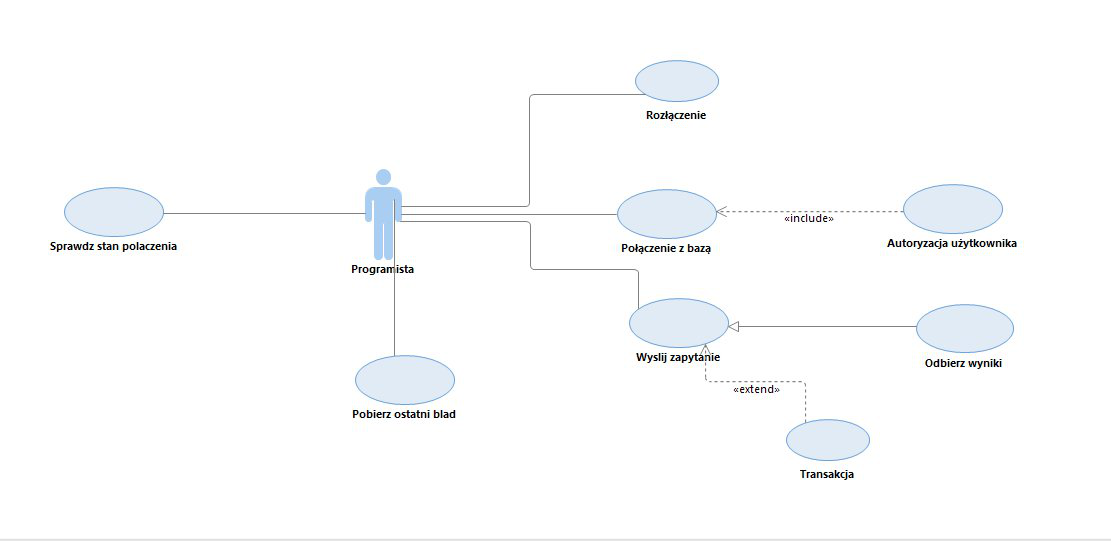
## Wymagania funkcjonalne:

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Nazwa | Opis | Priorytet |
|  | Połączenie z bazą danych | szatan | Wysoki |
|  | Autoryzacja użytkownika |  | Wysoki |
|  | Wykonywanie zapytań |  | Wysoki |
|  | Odbieranie wyników zapytań |  | Wysoki |
|  | Zapytania tranzakcyjne |  | Średni |
|  | Obsługa utraty połączenia |  | Wysoki |
|  | Sprawdzenie stanu połączenia |  | Niski |
|  | Serializacja danych |  | Średni |
|  | Rozłączenie z bazą |  | Wysoki |
|  | Pobranie ostatniego błędu |  | Niski |
|  |  |  |  |

## Wymaganie niefunkcjonalne:

* Szyfrowanie (XOR, MD5)
* Zabezpieczenie przed utratą danych przy błędzie połączenia
* Obsługiwane platformy: mysql, sqlite
* Obsługiwane systemy operacyjne: Linux
* Obsługa wielu klientów

## Przypadki użycia:



## Opis funkcjonalności poszczególnych modułów

1. **KLIENT**

**API –** Udostpnia podstawowe metody pozwalające na ustanawianie połączenia z bazą danych, identyfikacje użytkownika, wykonywanie zapytań, transakcji, obsługe bledów oraz bezpieczne zakonczenie polaczenia. API komunikuje sie z kontrolerem.

**Kontroler –** Stanowi łącznik pomidzy API a warstwą komunikacji z serwerem. Dokonuje serializacji danych. Szyfruje oraz deszefruje przesylane informacje. Przyjmuje polecenia, analizuje je oraz przesyla dalej w formie latwej do przesłania ☺

**Warstwa komunikacji ( serwer ) –** Odpowiada za odbieranie oraz wysylanie polecen do warstwy komunikacyjnej serwera. Kontroluje połączenie pomiedzy klientem a serwerem. Wznawia połączenie w wypadku awarii..

1. **SERWER**

**Warstwa komunikacji (klient ) –** Odpowiada za odbieranie oraz wysylanie polecen do kontrolera. Kontroluje połączenie pomiedzy klientem a serwerem. Wznawia połączenie w wypadku awarii. Pozwala na komunikacjie z wieloma użytkownikami jednocześnie.

**Kontroler –** Stanowi łącznik pomidzy warstwą komunikacji z klientem a warstwą komunikacji z bazą danych. Dokonuje serializacji danych. Szyfruje oraz deszefruje przesylane informacje. Przyjmuje polecenia, analizuje je oraz przesyla dalej w formie zrozumialej dla bazy danych. Buforuje zapytania. Nadzoruje dostep do bazy.

**Warstwa komunikacji ( baza ) –** Pośredniczy w wykonywaniu zapytan do bazy danych. Odbiera i przekazuje polecenia z modułu kontroler. Przyjmuje od bazy wyniki zapytan i przekazuje je z powrotem do kontrolera.

## Opisy najważniejszych pakietów

Nagłówek ogólny

|  |  |
| --- | --- |
| Size | 0 |
| Suma kontrolna MD5 | 128 bitów |

**Klient -> Serwer**

|  |  |
| --- | --- |
| **Cconnect – nawiązanie połączenia z bazą danych** | |
| Int type | DB\_CONNECT (0) |
| Int size |  |
| Char\* key |  |
| Char\* login |  |
| Char\* password |  |
| Char\* db\_name |  |

|  |  |
| --- | --- |
| **Cdisconnect - rozłączenie** | |
| Int type | DB\_DISCONNECT (1) |

|  |  |
| --- | --- |
| **CgetConnState – sprawdzenie połączenia** | |
| Int type | DB\_CON\_STATE (2) |

|  |  |
| --- | --- |
| **Ctransaction – rozpoczęcie transakcji** | |
| Int type | DB\_TRANSACTION (3) |

|  |  |
| --- | --- |
| **Ccommit – potwierdzenie transakcji** | |
| Int type | DB\_COMMIT (4) |

|  |  |
| --- | --- |
| **Crollback – cofnięcie transkacji** | |
| Int type | DB\_ROLLBACK (5) |

|  |  |
| --- | --- |
| **Cexec – wysłanie zapytania** | |
| Int type | DB\_EXEC (6) |
| Int size |  |
| Int id | Query id |
| Char\* query |  |

**Serwer -> Klient:**

|  |  |
| --- | --- |
| Odp na Cconnect | |
| Int type | DB\_CONNECT (0) |
| Int size | 4 + strlen(error\_msg) |
| Int ack | DB\_ACK (0) | DB\_NACK (1) |
| Char\* error\_msg | Komunikat od bazy |

|  |  |  |
| --- | --- | --- |
| Odp na CgetConnState | | |
| Int type | DB\_CON\_STATE (1) | |
| Int size | 4 + strlen(error\_msg) | |
| Int ack | DB\_OK (0) | DB\_NOK (1) | |
| Char\* error\_msg |  | |
| Odp na Ctransaction | | |
| Int type | | DB\_TRANSACTION (3) |
| Int size | | 4 + strlen(error\_msg) |
| Int ack | | DB\_OK (0) | DB\_NOK (1) |
| Char\* error\_msg | |  |

|  |  |
| --- | --- |
| Odp na Ccommit | |
| Int type | DB\_COMMIT (4) |
| Int size | 4 + strlen(error\_msg) |
| Int ack | DB\_OK (0) | DB\_NOK (1) |
| Char\* error\_msg |  |

|  |  |
| --- | --- |
| Odp na Crollback | |
| Int type | DB\_ROLLBACK (5) |
| Int size | 4 + strlen(error\_msg) |
| Int ack | DB\_OK (0) | DB\_NOK (1) |
| Char\* error\_msg |  |

|  |  |
| --- | --- |
| Odp na Cexec | |
| Int type | DB\_EXEC(6) |
| Int size | 4 + strlen(response) |
| Int ack | DB\_OK (0) | DB\_NOK (1) |
| Char\* response |  |

**Funkcje:**

**DBC::Commit – blokująca  
DBC::Cexec – blokująca  
DBC::Crollback – blokująca  
DBC::Ctranstaction – blokująca  
DBC::CconnState – blokująca**

## Diagramy

Poniżej znajdują się diagramy wymiany zapytań pomiędzy klientem a serwerem, co służy zrozumieniu, jaki był powód utworzenia danych funkcji. Rozważony jest przypadek

1. Utworzenia połączenia, wraz z zalogowaniem do bazy danych
2. Wysyłanie przygotowanego zapytania, oraz odpowiedź na nie.

Server

Client

Controller::connect()

Controller::serialize()

Coupler::connect()

ConnManager::listen()

ConnManager::associateSocket()

Coupler::acceptConnection()

Controller:serialize()

Controller::encode()

Coupler::waitforMessage()

C

Controller::received()

Controller::decode()

Controller::deserialize()

DBConnector::authenticate()

Controller::send()

Controller::serialize()

Controller::encode()

Coupler::send()

Coupler::send()

Coupler::waitForMessage()

Coupler::receive()

Controller::decode()

Controller::deserialize()

Zestawienie połączenia klient-serwer,  
Zalogowanie użytkownika do bazy

Controller::received()

Controller::decode()

Controller::deserialize

DBConnector::execQuery()

DBConnector::exec()

Controller::send()

Controller::serialize()

Contrller::encode()

Coupler::send(message)

Coupler::received()

Controller::received()

Controller::decode()

Controller::deserialize()

DBC::validQuery()

Query::setLostError()

Query::setValid()

Query::setResult

Coupler::waitForMessage()

Serwer

Controller::send()

Controller::serialize()

Contrller::encode()

DBC::exec(query)

Query::prepareQuery()

Klient

Coupler::send(message)

Coupler::waitForMessage()