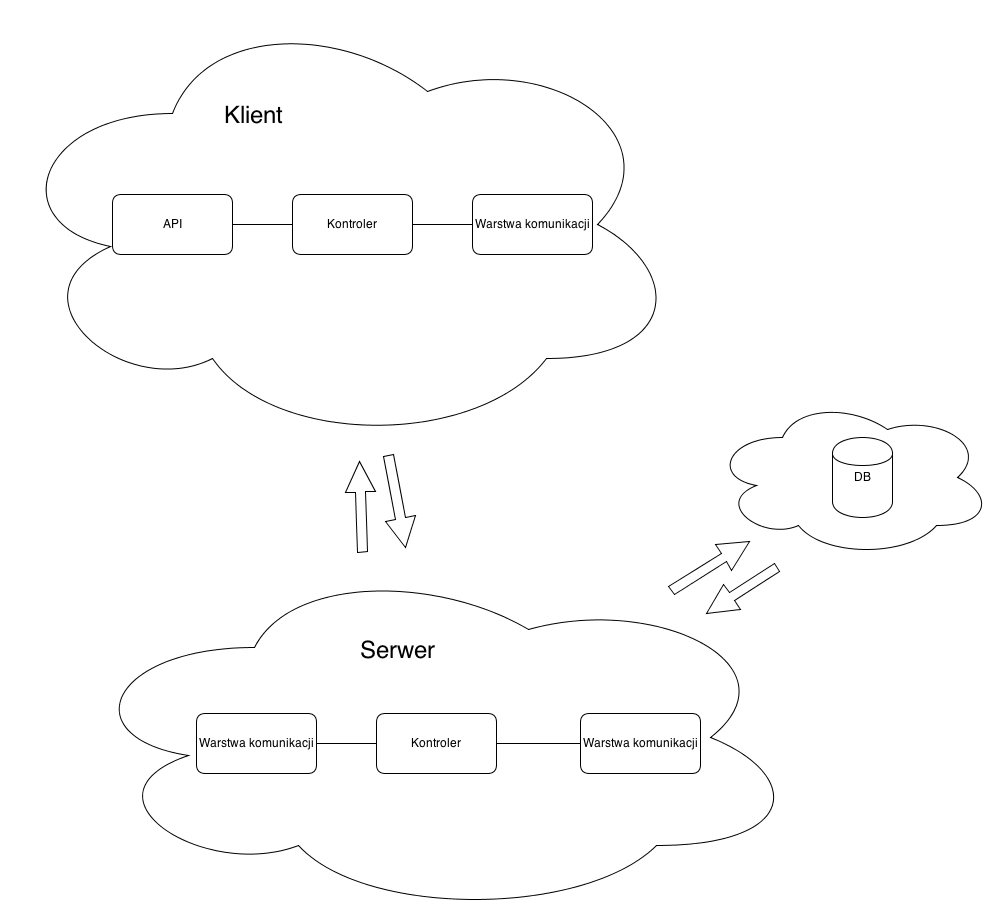
Biblioteka dostępowa do bazy danych

## 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:



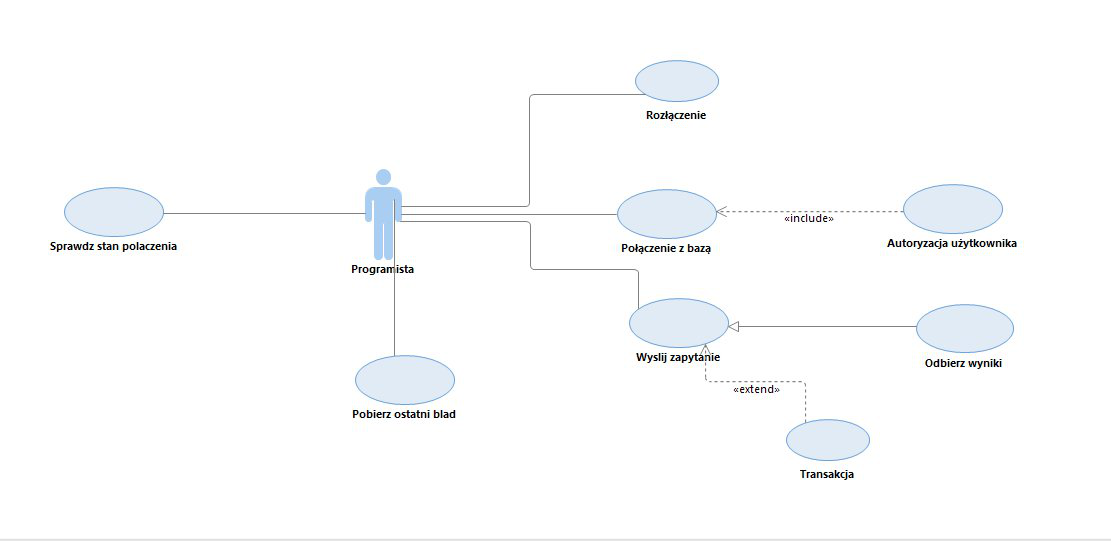
## Wymagania funkcjonalne:

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Nazwa** | **Opis** | **Priorytet** |
| REQ01.0 | Połączenie z bazą danych | System powinien umożliwiać połączenie się z bazą danych każdemu użytkownikowi, który przeszedł autoryzację. | Wysoki |
| REQ02.0 | Autoryzacja użytkownika | Warunkiem koniecznym nawiązania połączenia z bazą danych jest poprawna autoryzacja użytkownika. Podawany jest tutaj login, hasło, nazwa bazy. | Wysoki |
| REQ03.0 | Wykonywanie zapytań | Zautoryzowany użytkownik powinien mieć możliwość wykonywania zapytań na bazie danych. | Wysoki |
| REQ04.0 | Odbieranie wyników zapytań | System powinien obsługiwać możliwość odbioru wyników z bazy po wykonaniu zapytania. | Wysoki |
| REQ05.0 | Zapytania transakcyjne | Każda z operacji na bazie danych może być wykonana jako transakcyjna. Konieczne będą także transakcje wielopoziomowe. | Średni |
| REQ06.0 | Obsługa sytuacji krytycznych | System musi obsługiwać sytuacje krytyczne jak długa nieobecność użytkownika w systemie, utrata połączenia. | Wysoki |
| REQ07.0 | Sprawdzenie stanu połączenia | Użytkownik musi mieć możliwość sprawdzenia stanu nawiązanego połączenia poprzez zapytanie do serwera i/lub bazy danych po jego stronie. | Niski |
| REQ08.0 | Serializacja danych | Wymagana jest serializacja danych logowania oraz zwracanych wyników. | Średni |
| REQ09.0 | Rozłączenie z bazą | System powinien pozwalać użytkownikowi na rozłączenie z bazą danych i zakończenie połączenia. | Wysoki |
| REQ10.0 | Pobranie ostatniego błędu | Użytkownik powinien mieć możliwość pobrania ostatniego błędu, powstałego na nieudanej operacji na bazie danych. | 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 –** Udostępnia podstawowe metody pozwalające na ustanawianie połączenia z bazą danych, identyfikacje użytkownika, wykonywanie zapytań, transakcji, obsługę błędów oraz bezpieczne zakończenie polaczenia. API komunikuje sie z kontrolerem.

**Kontroler –** Stanowi łącznik pomiędzy API a warstwą komunikacji z serwerem. Dokonuje serializacji danych. Szyfruje oraz deszyfruje przesyłane informacje. Przyjmuje polecenia, analizuje je oraz przekazuje dalej w formie łatwej do przesłania.

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

1. **SERWER**

**Warstwa komunikacji (klient ) –** Odpowiada za odbieranie oraz wysyłanie poleceń do kontrolera. Kontroluje połączenie pomiędzy klientem a serwerem. Wznawia połączenie w wypadku awarii. Pozwala na komunikację z wieloma użytkownikami jednocześnie.

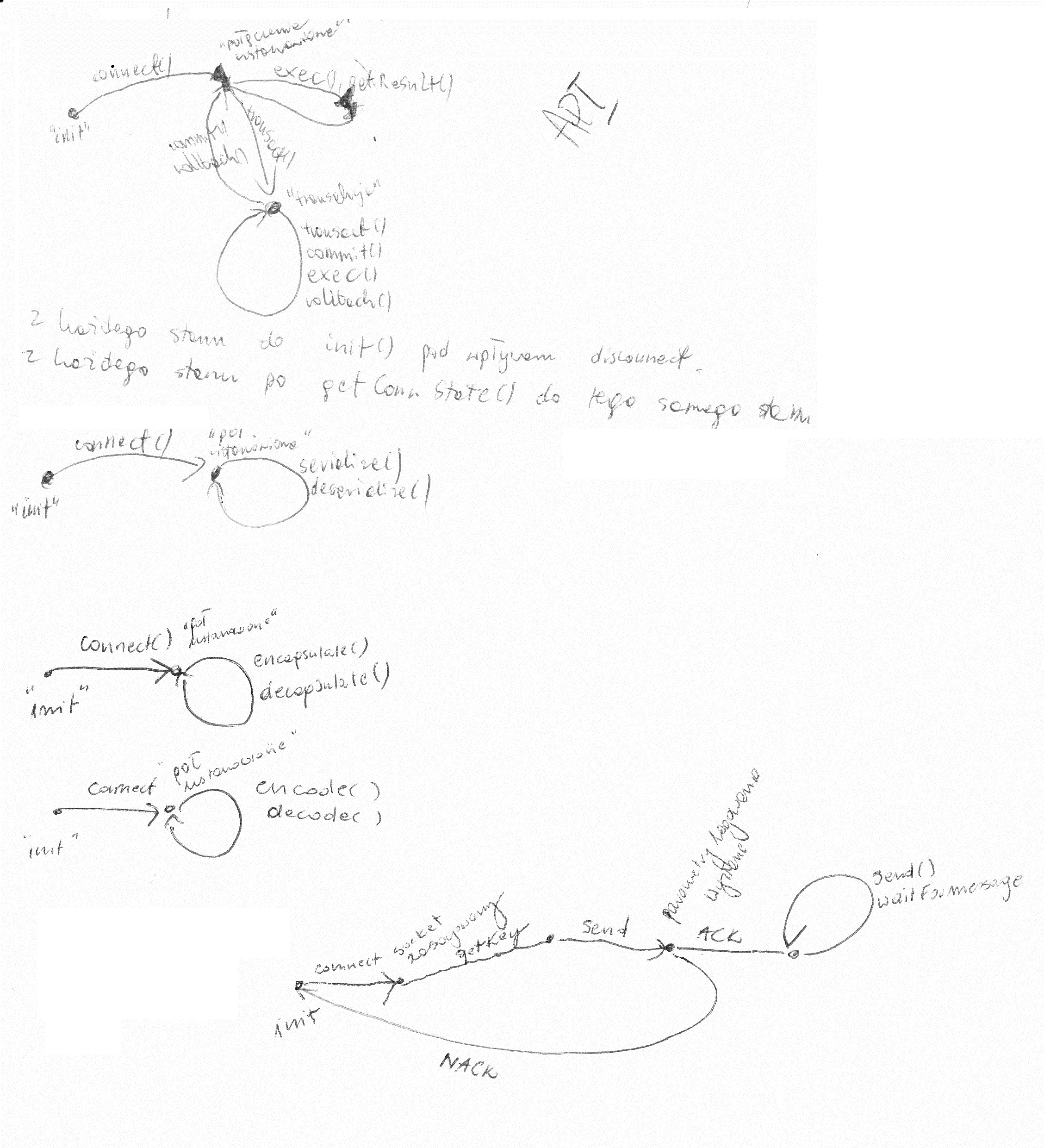
**Kontroler –** Stanowi łącznik pomiędzy warstwą komunikacji z klientem a warstwą komunikacji z bazą danych. Dokonuje serializacji danych. Szyfruje oraz deszyfruje przesyłane informacje. Przyjmuje polecenia, analizuje je oraz przesyła dalej w formie zrozumiałej dla bazy danych. Buforuje zapytania. Nadzoruje dostęp do bazy.

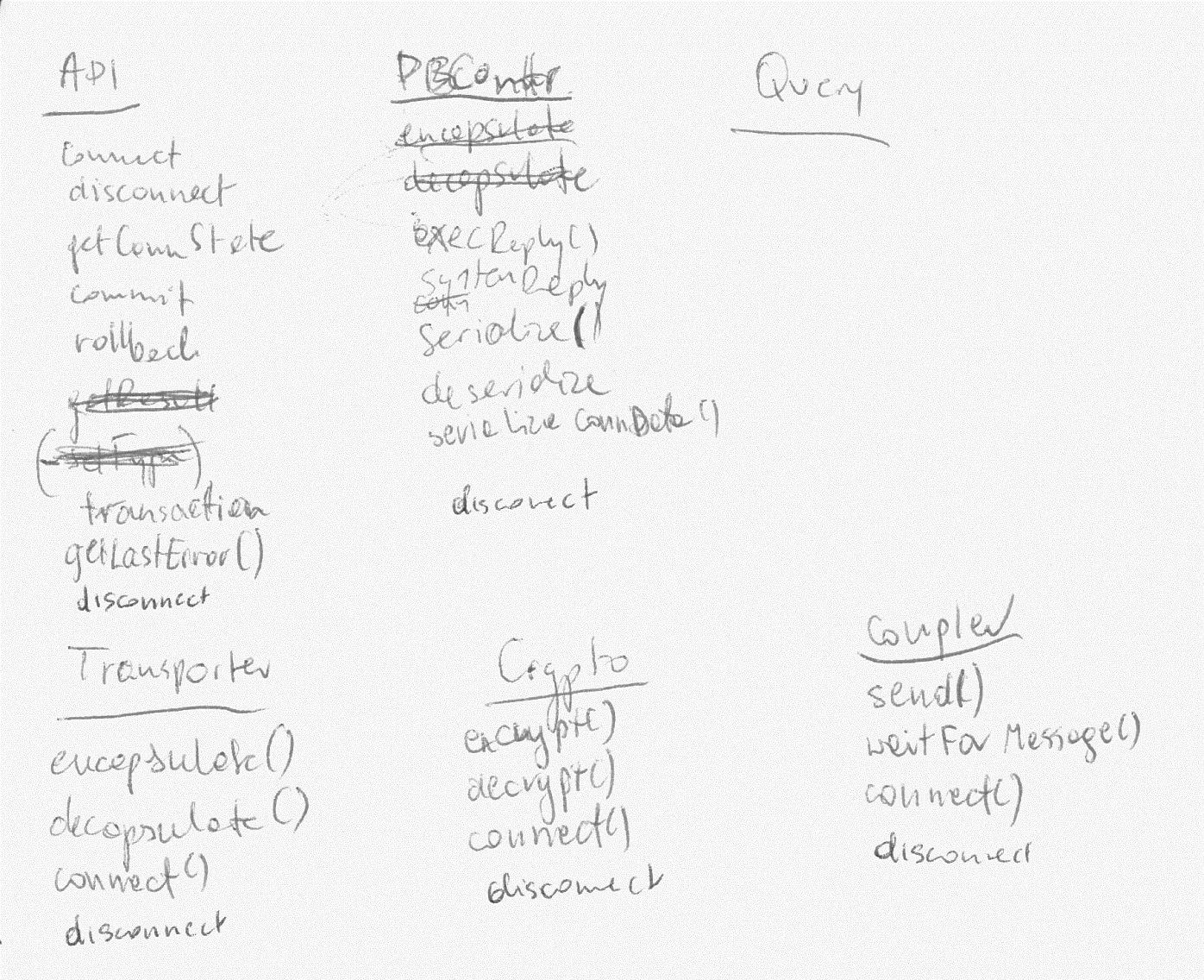
**Warstwa komunikacji ( baza ) –** Pośredniczy w wykonywaniu zapytań do bazy danych. Odbiera i przekazuje polecenia z modułu kontroler. Przyjmuje od bazy wyniki zapytań i przekazuje je z powrotem do kontrolera.

## Diagramy

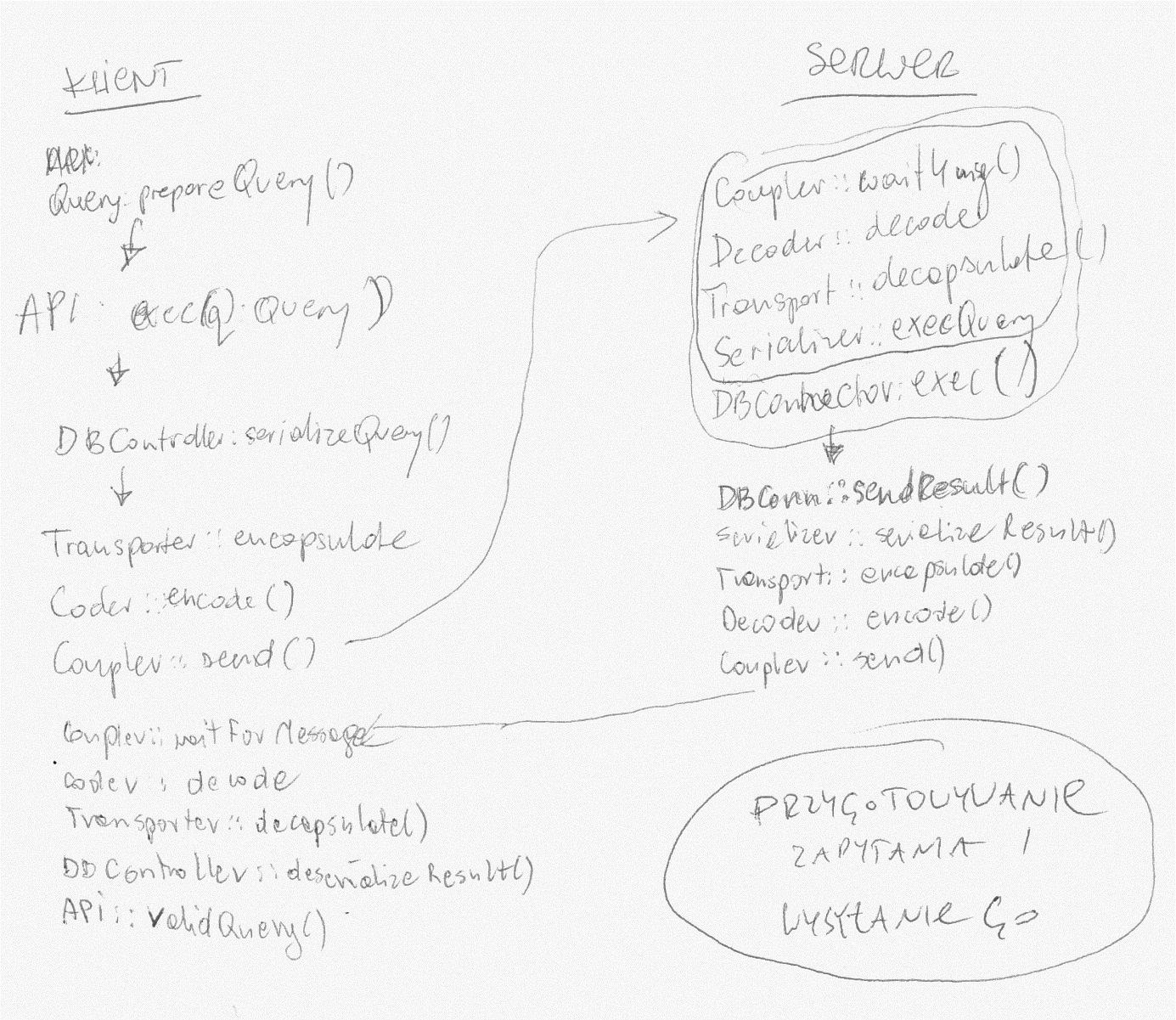
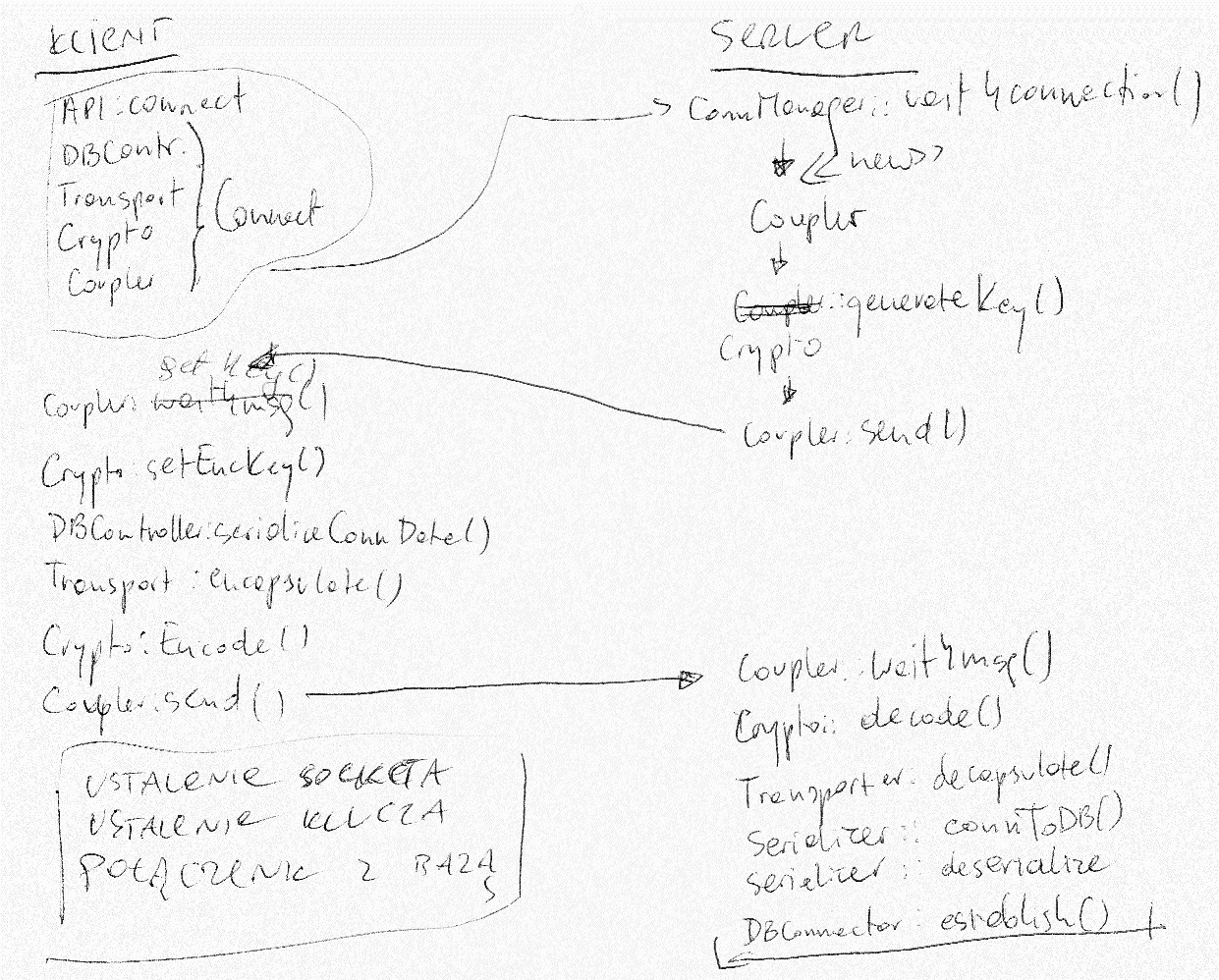
Poniżej znajdują się diagramy wymiany zapytań pomiędzy klientem a serwerem, co służy zrozumieniu, jaki był powód utworzenia danych funkcji. Rozważony jest przypadek

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

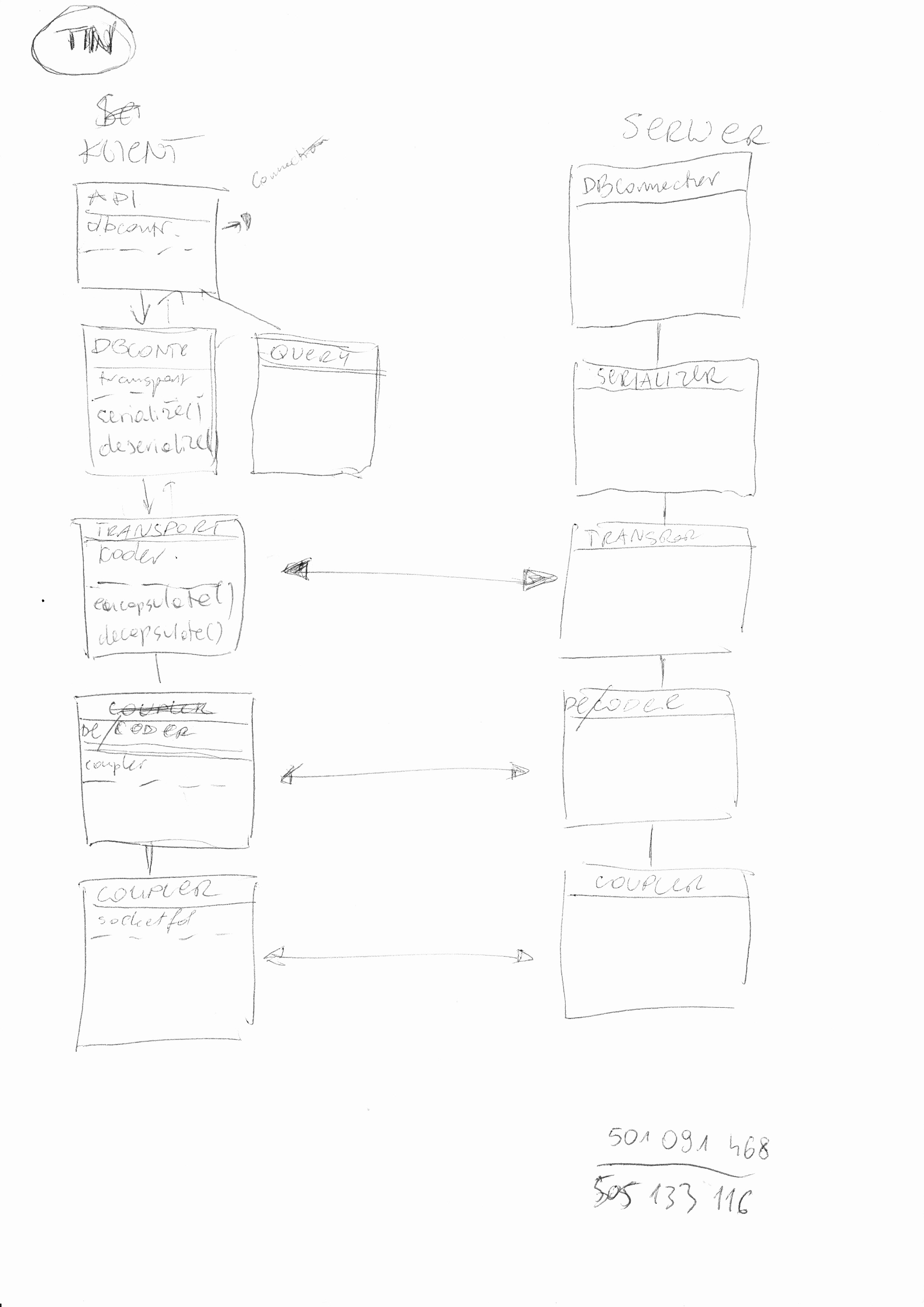
****

**KLIENT**

**SERWER**

****

Controller::serialize()



## Opisy najważniejszych pakietów

**Nagłówek ogólny**

|  |  |
| --- | --- |
| Int size |  |
| Int type |  |
| 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::exec – nieblokująca  
DBC::rollback – blokująca  
DBC::transtaction – blokująca  
DBC::connState – blokująca