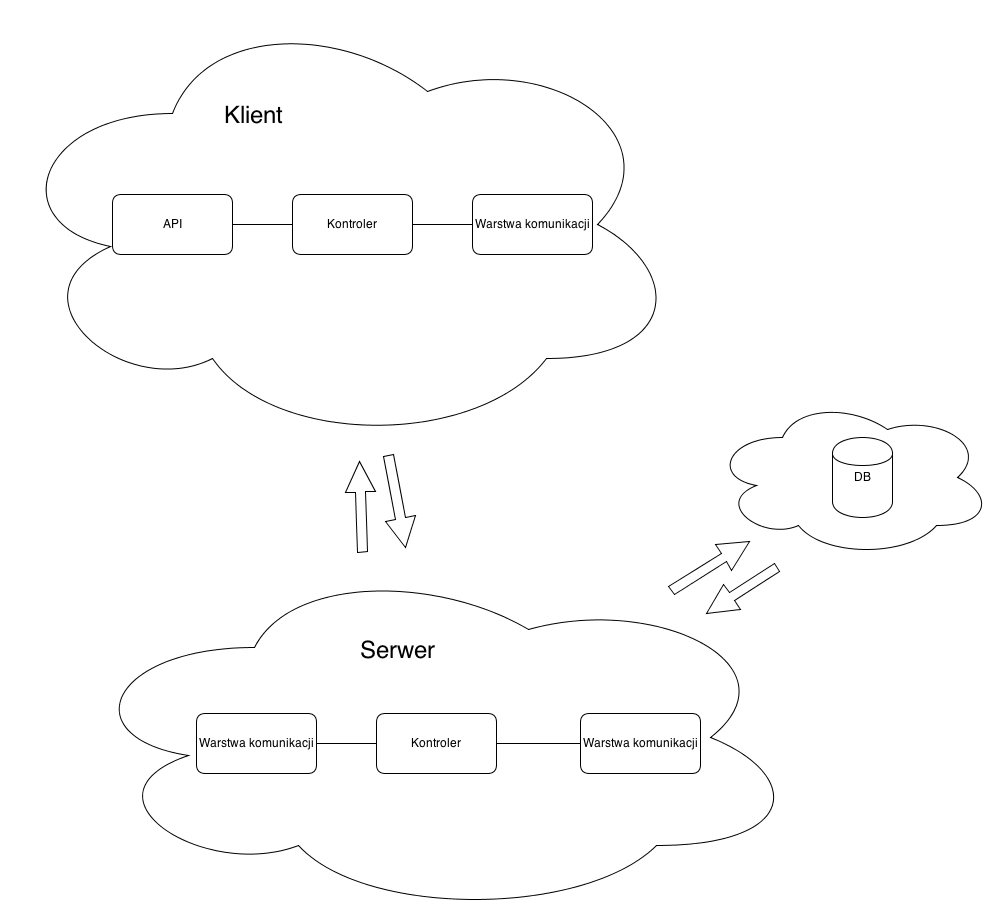
Biblioteka dostępowa do bazy danych

## 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:



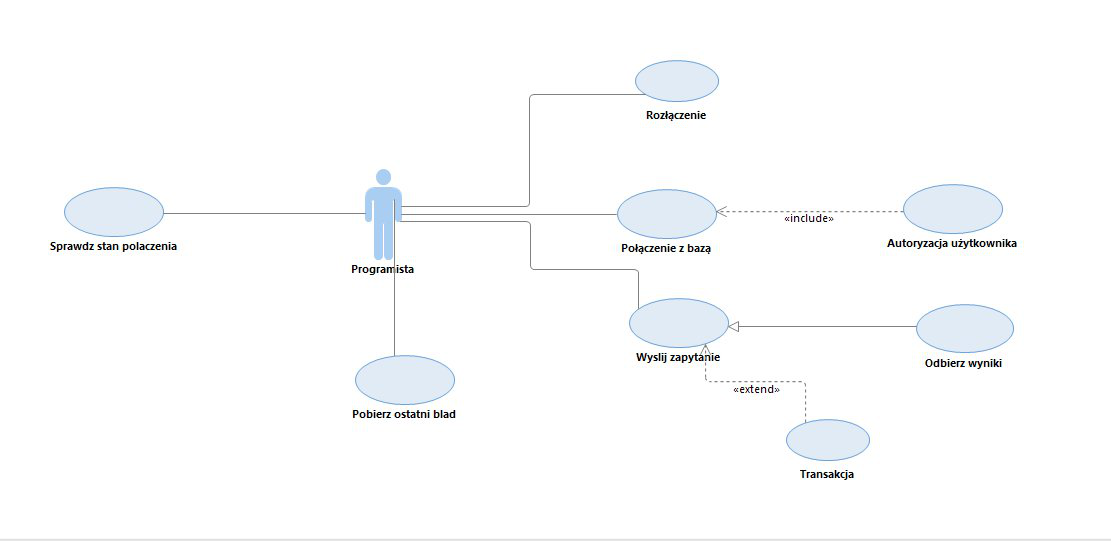
## Wymagania funkcjonalne:

|  |  |  |  |
| --- | --- | --- | --- |
| **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.

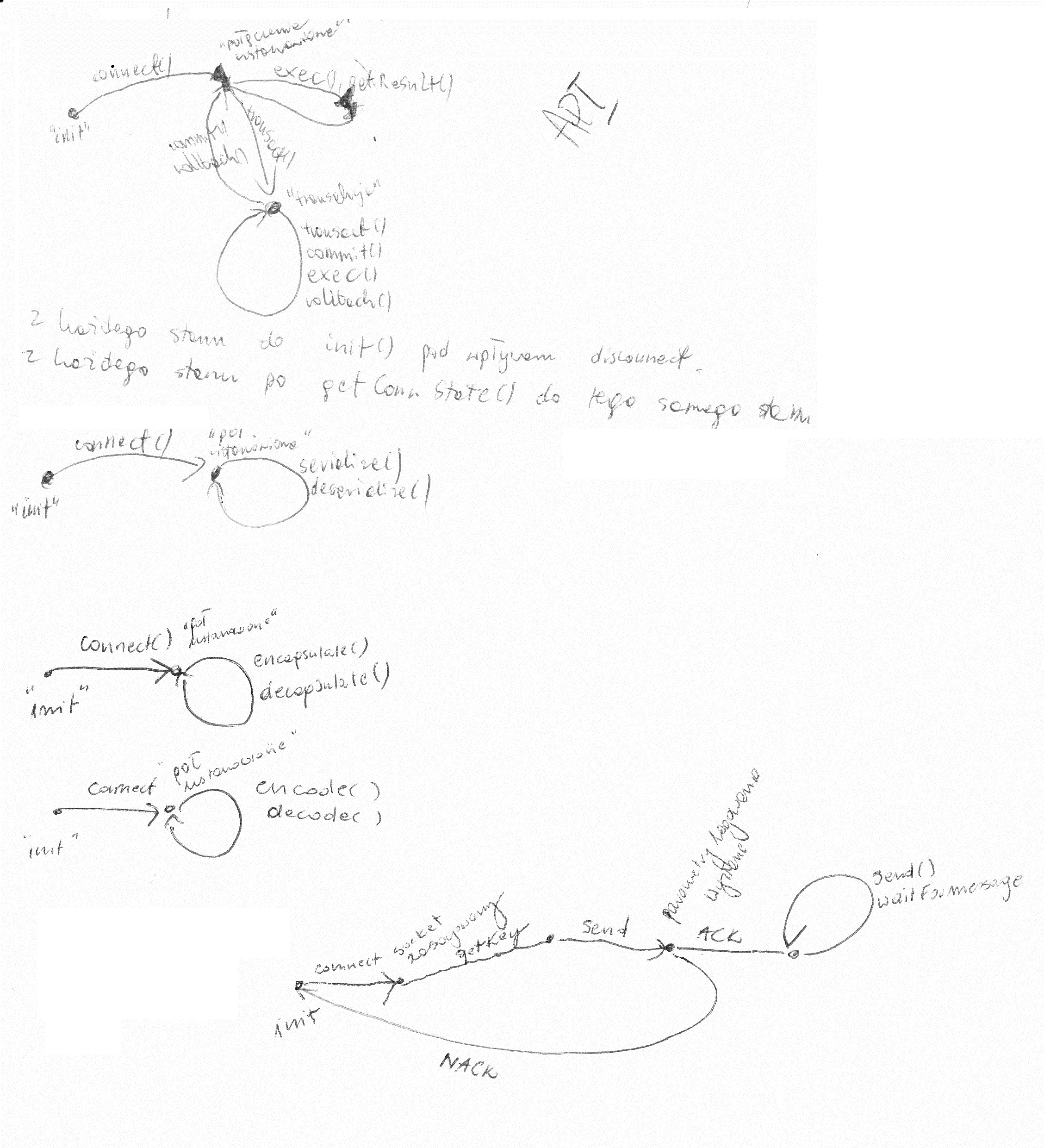
**Kontroler –** Stanowi łącznik pomiędzy warstwą komunikacji z klientem a warstwą komunikacji z bazą danych. Dokonuje serializacji danych. Szyfruje oraz deszyfruje przesyłane informacje. Przyjmuje polecenia, analizuje je oraz przesyła dalej w formie zrozumiałej dla bazy danych. Buforuje zapytania. Nadzoruje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

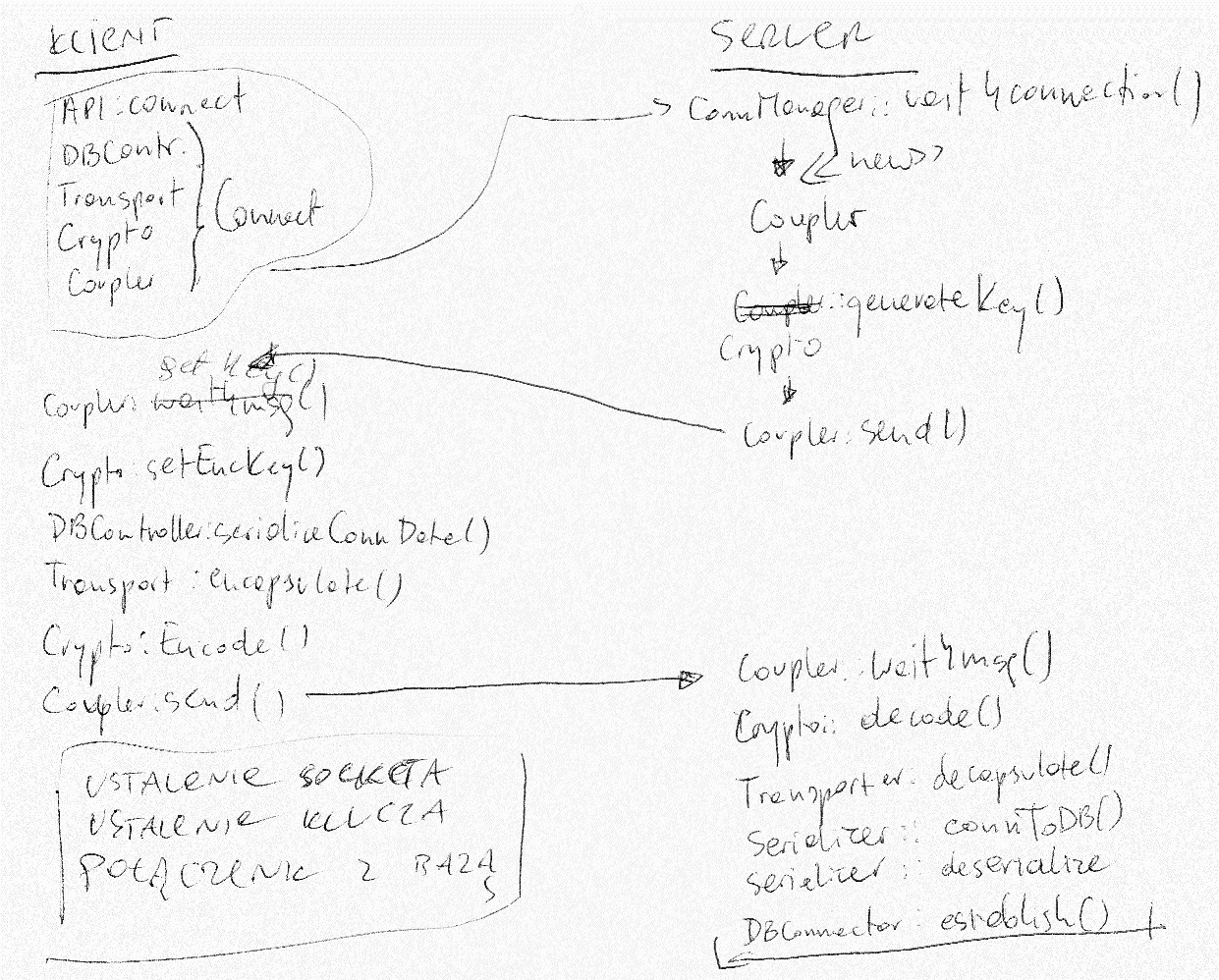
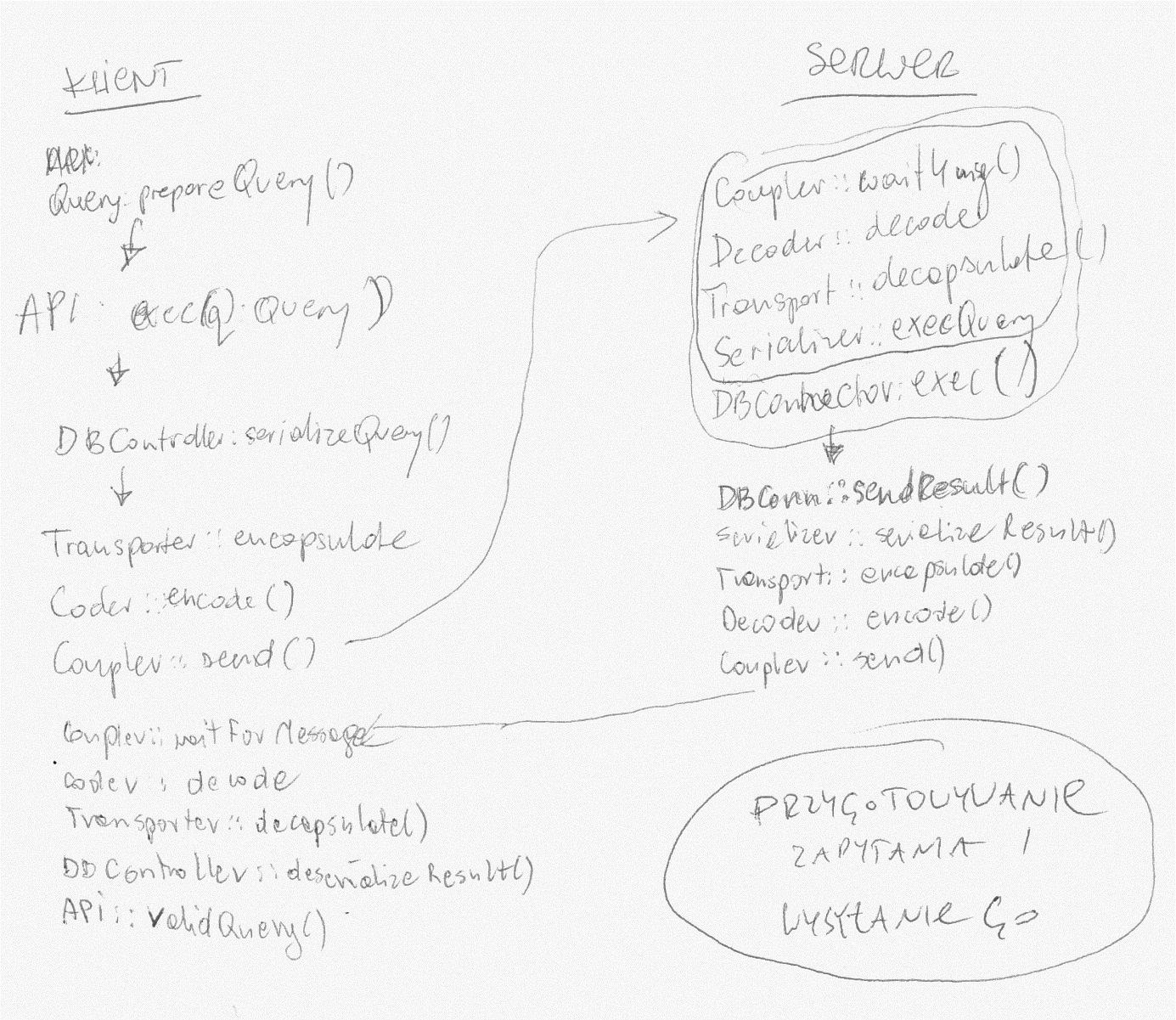
**Warstwa komunikacji ( baza ) –** Pośredniczy w wykonywaniu zapytań do bazy danych. Odbiera i przekazuje polecenia z modułu kontroler. Przyjmuje od bazy wyniki zapytań i przekazuje je z powrotem do kontrolera.

## Diagramy

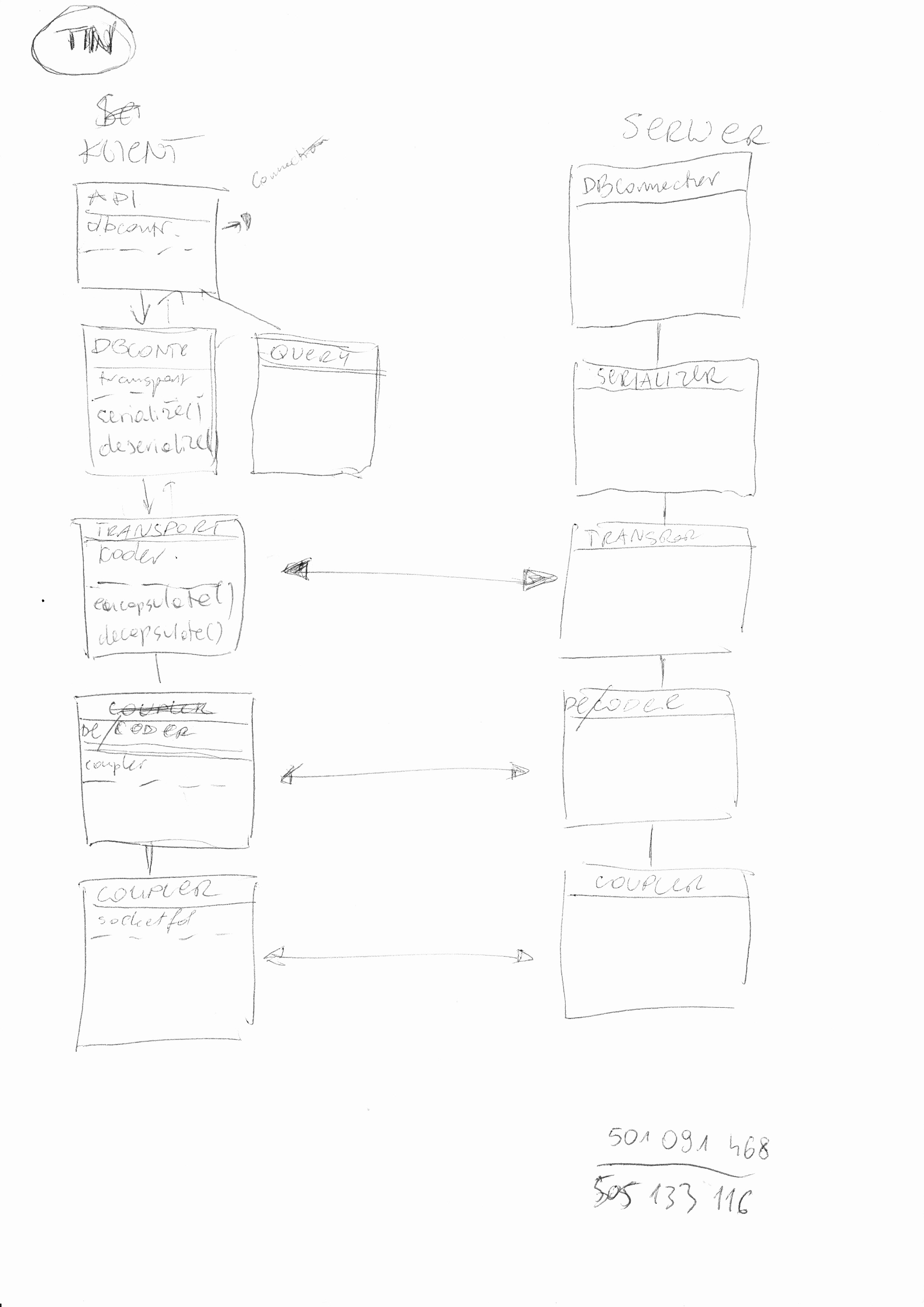
Poniżej znajdują się diagramy wymiany zapytań pomiędzy klientem a serwerem, co służy zrozumieniu, jaki był powód utworzenia danych funkcji. Rozważony jest przypadek

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

****

****

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

|  |
| --- |
| Coupler |
| SOCKET sock;  SOCKADDR\_IN SockAddr;  Crypter \*parent;  timeval tv; |
| + Coupler(Crypter \*parent);  + ~Coupler(void);  + int sendAndWait(const char\* msg, int length);  + int waitForMessage(void);  + int readBytes(char\* buf, int size);  + bool read(char\* buf, int size);  + int write(char\* buf, int size);  + int conn(const char\* addr);  + void disconn(); |
| Crypter |
| char klucz[33];  Coupler \*child;  Transporter \*parent; |
| + Crypter(Transporter \*parent);  + ~Crypter(void);  + int encrypt(const char\* msg, int length);  + int decrypt(const char\* msg, int length);  + int connect(const char\* addr, const char\* msg, int length);  + void disconnect(void);  + void setKey(const char\* key) ; |
| Transporter |
| Controller \*parent;  Crypter \*child; |
| + Transporter(Controller \*parent);  + ~Transporter(void);  + int encapsulate(Types type, const char\* msg, int length);  + int decapsulate(const char\* msg, int length);  + int connect(const char\* addr, const char\* msg, int length);  + void disconnect(void); |
| Controller |
| NDBC \*parent;  Transporter \*child; |
| + Controller(NDBC \*parent);  + ~Controller(void);  + int executionReply(const char\* msg);  + int systemReply(const char\* msg);  + int serialize(Types type, string query, unsigned int id = 0);  + void deserialize(void);  + int serializeConnectionData(string user, string pass, string db, string addr);  + void disconnect(void); |
| NDBC |
| friend class Controller;  vector<Query\*> queries;  Controller \*child;  string error;  bool connected; |
| + NDBC(void);  + ~NDBC(void);  + bool connect(string user, string pass, string db, string addr);  + void disconnect(void);  + void getConnectionState(void);  + bool commit(void);  + bool rollback(void);  + bool transaction(void);  string getLastError(void);  + bool exec(Query \*query);  - void setLastError(const string &error) { this->error = error; }  - void updateQuery(int id, unsigned int ack, string &msg); |

# Klient

# Serwer

|  |
| --- |
| Server |
| unsigned int clients;  unsigned int port;  int MAX\_CLIENTS;  timeval tv;  WSADATA wsaData;  SOCKADDR\_IN saddr;  SOCKET client;  int newId;  vector<HANDLE> threads;  SOCKET sock;  static void handler();  static Server \*instance; |
| + Server(unsigned int clients, unsigned int port);  + ~Server(void);  + int run(void); |
| SCoupler |
| Crypter\* parent;  SOCKET sock;  timeval tv;  bool read(char\* buf, int size);  int readBytes(char\* buf, int size);  int write(char \*buf, int size); |
| + SCoupler();  + ~SCoupler(void);  + void setSocket(SOCKET socket) { sock = socket; };  + int run();  + int waitForMessage();  + int sendMessage(const char\* msg, int length);  + static DWORD WINAPI init (LPVOID ctx);  + static bool close; |
| Crypter |
| char klucz[33];  SCoupler \*child;  Transporter \*parent; |
| + Crypter(SCoupler\* child);  + ~Crypter(void);  + int encrypt(const char\* msg, int length);  + int decrypt(const char\* msg, int length);  + int connect(const char\* addr, const char\* msg, int length);  + void disconnect(void);  + char\* genKey(); |
| Transporter |
| DBConnector \*parent;  Crypter \*child; |
| + Transporter(Crypter\* child);  + ~Transporter(void);  + int encapsulate(Types type, unsigned int ack, const char\* msg, int length, unsigned int id = 0);  + int decapsulate(const char\* msg, int length);  + int connect(const char\* addr, const char\* msg, int length);  + void disconnect(void); |
| DBConnector |
| Transporter\* child;  Driver \*driver;  Connection \*connection;  Statement \*statement;  ResultSet \*resultSet;  ResultSetMetaData \*rsmd;  string result; |
| + DBConnector(Transporter \*child);  + ~DBConnector(void);  + int connect(string host, string user, string password, string db);  + int executeQuery(string query, unsigned int type, unsigned int id);  + string getResult(); |

# Implementacja

Na potrzeby projektu zaimplementowano dwie aplikacje. Aplikacje kliencką oraz aplikacje serwer udostępniającą dostęp do bazy mySQL. W aplikacjach wykorzystano kolejno biblioteki: RapidXML oraz mySQL Connector/C++.

# Testowanie oprogramowania

## Testowanie jednostkowe

Przedmiotem testowania jednostkowego były podstawowe jednostki programu. Celem testowania na tym poziomie było sprawdzenie zgodności działania wszystkich opracowanych jednostek z ich specyfikacja, wynikającą z projektu, oraz wykrycie i usuniecie jak największej liczby bledów.

## Testowanie integracyjne

Przedmiotem działań podejmowanych podczas testowania integracyjnego było łączenie jednostek programu w coraz większe komponenty i sprawdzenie zgodności ich działania ze specyfikacją wynikającą z projektu architektury oprogramowania.

### Testy funkcjonalne

Podczas tych testów sprawdzono poprawność wykonania wszystkich funkcji systemu.

### Testy wydajnościowe

Podczas tych testów sprawdzono działanie oprogramowania w warunkach codziennego obciążenia systemu.

### Testy przeciążeniowe

Podczas tych testów sprawdzono odporność oprogramowania w warunkach przekroczenia założonego obciążenia systemu.

### Testy bezpieczeństwa

Podczas tych testów sprawdzono skuteczność mechanizmów szyfrowania.

### Testy odporności

Podczas tych testów sprawdzono działanie oprogramowania w wypadku awarii ( przerwania działania aplikacji )

### Testy dokumentacji

Podczas tych testów sprawdzono zgodność działania oprogramowania z opisem zawartym w dokumentacji.

## Testowanie akceptacyjne

Przedmiotem testowania akceptacyjnego był kompletny produkt. Przeprowadzono testy alfa oraz testy beta.

# Załącznik

W ramach testów funkcjonalnych sprawdzono poprawność wykonania niżej wymienionych funkcji systemu

1. Połączenie z bazą danych
2. Wykonywanie zapytań
3. Odbieranie wyników zapytań
4. Rozpoczęcie transakcji
5. Wykonanie transakcji
6. Zamkniecie transakcji
7. Pobranie ostatniego błędu
8. Sprawdzenie stanu połączenia
9. Rozłączenie połączenia z bazą danych

