

Programowanie aplikacji klient-serwer

Projekt 6 Egzamin testowy

Projekt

(wersja 1.1)

AUTORZY PROJEKTU:

Uladzislau Harbuz

Maksym Yakymyshyn

Spis treści

1	Wstęp.....	3
1.1	Treść zadania.....	3
1.2	Dodatkowe założenia.....	3
1.2.1	Serwer.....	3
1.2.2	Klient.....	3
2	Słownik pojęć.....	3
3	Użyte protokoły.....	3
3.1	Protokół komunikacji klienta z serwerem.....	3
3.1.1	Wstęp.....	3
3.1.1.1	Technologia.....	3
3.1.1.2	Funkcjonalność:.....	3
3.1.2	Dokładny opis funkcjonalny poszczególnych metod/komend.....	4
3.1.2.1	XXXXXXXXXXXXXXX (.....	4
3.1.3	Spis komunikatów błędów.....	4
3.2	Protokół przesyłania plików pomiędzy klientami(o ile występuje).....	4
3.2.1	Wstęp.....	4
3.2.2	Żądanie.....	5
3.2.3	Odpowiedź.....	5
3.2.4	Spis komunikatów błędów.....	5
4	Przypadki użycia.....	5
4.1.1	Klient tworzy nowego użytkownika w bazie serwera.....	5
4.1.2	Klient loguje się do serwera.....	5
4.1.3	Klient wylogowuje się z serwera.....	5
4.1.4	XXXXXXXXXXXXXXXXXXXX.....	5
5	Aplikacja klienta.....	5
5.1	Wstęp.....	5
5.2	Moduł nazwa1.....	5
5.3	Moduł nazwa2.....	5
6	Aplikacja serwera.....	5
6.1	Wstęp.....	5
6.2	Moduł nazwa1.....	6
6.3	Moduł nazwa2.....	6
7	Przechowywane dane	6
7.1	Wstęp.....	6
7.2	Tabela/ plik1.....	6
7.3	Tabela/ plik2.....	6
8	Interfejs danych.....	6
8.1	Funkcja 1.....	6
8.2	Funkcja 2.....	6
8.3	6
8.4	Spis komunikatów błędów.....	6
9	Planowany podział prac.....	6

1 Wstęp

1.1 Treść zadania

Aplikacja umożliwia tworzenie i przeprowadzanie egzaminu testowego. 3 rodzaje użytkowników: administrator, egzaminator i student. Administrator tworzy grupy studenckie i przypisuje do nich studentów. Egzaminator wprowadza egzamin testowy (wszystkie pytania zamknięte) podając dla wszystkich pytań:

- Treść pytania,
- Propozycje odpowiedzi (co najwyżej 5),
- Poprawną odpowiedź (tylko 1).

Egzaminator przeprowadza egzamin udostępniając go wybranej grupie studentów na określony czas oraz uruchamia automatyczne sprawdzanie wyników testu dla poszczególnych studentów.

Po sprawdzeniu wynik testu jest przypisywany poszczególnym studentom.

Student loguje się, wprowadza odpowiedzi na poszczególne pytania i ogląda wyniki.

1.2 Dodatkowe założenia

1.2.1 Serwer

Serwer wykonany zostanie w językach C i C++. Dane o klientach, plikach i zamówieniach będą przechowywane w bazie danych SQLite.

1.2.2 Klient

Klient będzie aplikacją napisaną w języku C#

2 Słownik pojęć

Sesja – abstrakcja połączenia zautentyfikowanego klienta z serwerem. Wszystkie komunikaty są wysyłane w ramach sesji. Wszystkie próby komunikacji z serwerem poza aktywną sesją są odrzucane przez serwer.

3 Użyte protokoły

3.1 Protokół komunikacji klienta z serwerem

3.1.1 Wstęp

Został opracowany własny protokół komunikacji pomiędzy klientem a serwerem (Tablica 3.1.1.2.1).

3.1.1.1 Technologia

Zaprojektowana ramka danych składa się z:

Polecenie	Rozmiar danych	Dane
-----------	----------------	------

Polecenie – 1 bajt

Rozmiar danych – 1 bajt, który pokazuje rozmiar danych w sekcji „Dane”(dom. w bajtach).

W przypadku błędu po stronie serwera serwer musi zwracać komunikat ERROR_DATAGRAM (Tablica 3.1.1.2.1).

3.1.1.2 Funkcjonalność:

Protokół ten realizuje następującą funkcjonalność:

Tablica 3.1.1.2.1 – Opis protokołu komunikacji klient-serwer

Lp	Treść komunikatu	Spodziewana reakcja
0	INVALID_COMMAND	Serwer dostał nie istniejący komunikat.
1	GET_TEST_LIST_SIZE	Klient wysyła do serwera ramkę z polem „Rozmiar danych” = 0 i bez pola „Dane”. Klient otrzymuje liczbę testów w polu „Dane”.
2	GET_TEST	Klient w polu „Dane” wpisuje numer testu. Otrzymuje od serwera wypełnioną ramkę danych w formacie JSON zdefiniowanym następująco: <code>{„text”:"tekst_pytania", „variants”:[{„1”: „odp1”},{„2”:"odp2”, ...]},</code> gdzie text – tekst pytania, variants – lista możliwych odpowiedzi na pytanie.
3	SEND_TESTS_ANSWERS	Klient wysyła w polu „Dane” wybrane odpowiedzi dla testów w formacie JSON zdefiniowanym następująco: <code>{„answewrs”:[„test_n”:"ans_n", ...]},</code> gdzie answers – lista par „numer testu”:"numer wybranej odpowiedzi”. Klient dostaje w odpowiedź ramkę z kodem błędu.
4	OPEN_SESSION	Komunikat mówi serwerowi otworzyć nową sesję. Serwer czeka na nowe połączenie od klientów. Po uzyskaniu nowego połączenia serwer czeka

		<p>na ten komunikat. Pole „Rozmiar danych” jest równe LOGIN_BYTE_SIZE + PASSWORD_BYTE_SIZE z tablicy 3.1.1.2.2, i w polu „Dane” do serwera są przesyłane login w pierwszych LOGIN_BYTE_SIZE i hasło w bajtach [LOGIN_BYTE_SIZE, PASSWORD_BYTE_SIZE].</p> <p>Klient dostaje w odpowiedź komunikat zwrotny z kodem błędu.</p>
5	CLOSE_SESSION	<p>Zamyka aktualną sesję.</p> <p>Klient dostaje komunikat zwrotny z kodem błędu.</p>
6	GET_RESULTS	<p>Klient wysyła pustą ramkę.</p> <p>Dostaje od serwera ramkę z polem „Dane” zawierającą wyniki testów w formacie JSON:</p> <pre>{„pass”:"liczba_testów_zdanych", „all”:"liczba_wszystkich_testów"},</pre> <p>gdzie pass – liczba prawidłowych odpowiedzi w teście,</p> <p>all – liczba wszystkich testów.</p>
7	CHANGE_CREDENTIALS	<p>Klient wpisuje w pole „Dane” w pierwsze LOGIN_BYTE_SIZE bajt nowy login, a w następne PASSWORD_BYTE_SIZE nowe hasło.</p> <p>Klient dostaje komunikat zwrotny z kodem błędu.</p>
8	ADD_GROUP	<p>Klient wpisuje w pole „Dane” nazwę nowej grupy.</p> <p>Dostaje od serwera komunikat zwrotny z kodem błędu.</p>
9	ADD_USER	<p>Klient wpisuje w pole danych : pierwsze LOGIN_BYTE_SIZE bajtów – login nowego użytkownika, w następne PASSWORD_BYTE_SIZE bajtów – nowe hasło nowego użytkownika, i w ostatnim bajcie wysyła kod grupy nowego użytkownika z tablicy 3.1.1.2.3.</p> <p>Klient dostaje komunikat zwrotny z kodem błędu.</p>
10	ERROR_DATAGRAM	<p>Ten komunikat świadczy o tym, że w polu „Dane” w pierwszym bajcie znajduje się kod błędu , a w drugim bajcie - kod operacji której</p>

		ten błąd dotyczy. Rozmiar pola „Dane” Wynosi 2 bajty.
11	DELETE_USER	Użytkownik z uprawnieniami administratora podaje w polu „dane” nazwę użytkownika do usunięcia. W przypadku sukcesu podany użytkownik zostanie usunięty z listy użytkowników.
12	DELETE_GROUP	Użytkownik z uprawnieniami administratora podaje w polu „dane” nazwę grupy do usunięcia. W przypadku sukcesu podana grupa zostanie usunięta z listy istniejących grup.
13	GET_USER_INFO	<p>Klient wysyła ramkę z nazwą użytkownika w polu „dane”.</p> <p>W odpowiedzi od serwera dostaje ramkę z polem polecenia „GET_USER_INFO”, i polem „dane” wypełnionym w formacie json zdefiniowanym następująco:</p> <pre>{„groups”:[„grupa 1”, „grupa 2”, ...], „login”:”nazwa użytkownika”, „name”:”Imię”, „surname”:”Nazwisko”},</pre> <p>gdzie groups – lista grup do których należy użytkownik,</p> <p>login – nazwa użytkownika,</p> <p>name – imię użytkownika,</p> <p>surname – nazwisko użytkownika.</p> <p>W przypadku błędu lub nie istnienia użytkownika klient dostaje zwykły komunikat typu ERROR_DATAGRAM.</p>
14	SET_USER_INFO	<p>Ustawia dane podanego użytkownika na nowe.</p> <p>Klient wysyła ramkę z polem „dane” wypełnionym w formacie json zdefiniowanym w następujący sposób:</p> <pre>{„login”:”nazwa użytkownika”, „name”:”Imię”, „surname”:”Nazwisko”},</pre> <p>gdzie login – nazwa użytkownika dla którego dane zostaną zmienione,</p> <p>name – nowe imię użytkownika,</p> <p>surname – nowe nazwisko użytkownika.</p> <p>Klient dostaje w odpowiedzi ramkę z kodem błędu.</p>
15	ADD_TO_GROUP	Dodaje podanego użytkownika do podanej grupy. Dla wykonania tej operacji są potrzebne

		<p>uprawnienia administratora.</p> <p>Klient wysyła ramkę z polem „dane” wypełnionym w formacie json zdefiniowanym następująco:</p> <pre>{„login”:"login użytkownika", „group”:"nazwa grupy"},</pre> <p>gdzie login – nazwa użytkownika którego trzeba dodać do grupy,</p> <p>group – nazwa grupy do której zostanie dodany użytkownik.</p> <p>Klient dostaje w odpowiedzi ramkę z kodem błędu.</p>
16	REMOVE_FROM_GROUP	<p>Usuwa podanego użytkownika z podanej grupy.</p> <p>Klient wysyła ramkę z polem danych wypełnionym w formacie json zdefiniowanym następująco:</p> <pre>{„login”:"nazwa użytkownika", „group”:"nazwa grupy"},</pre> <p>gdzie login – nazwa użytkownika który zostanie usunięty z grupy,</p> <p>group – nazwa grupy z której użytkownik zostanie usunięty.</p>
17	ADD_TEST	<p>Dodaje nowy test z odpowiedziami.</p> <p>Klient wysyła ramkę w formacie JSON:</p> <pre>{„text”:"tekst pytania", „answers”:[„odp1”, „odp2”], „right_answer”:"id"},</pre> <p>gdzie text – tekst pytania,</p> <p>answers – lista możliwych odpowiedzi na pytanie oprócz prawidłowej odpowiedzi.</p> <p>right_answer – prawidłowa odpowiedź na pytanie.</p> <p>Jeżeli odpowiedź prawidłowa jest pusta klient dostaje błąd BAD_FORMAT.</p>
18	REMOVE_TEST	<p>Usuwa podany test z listy testów.</p> <p>Klient podaje w danych ramki numer testu do usunięcia.</p>

Tablica 3.1.1.2.2 – Wartości stałe protokołu

Wartość	Nazwa	Opis
20	LOGIN_BYTE_SIZE	Rozmiar loginu w bajtach.

30	PASSWORD_BYTE_SIZE	Rozmiar hasła w bajtach.

3.1.2 Dokładny opis funkcjonalny poszczególnych metod/komend

3.1.2.1 XXXXXXXXXXXXXXX (

Sygnatura:

Deklaracja (wartość zwracana, nazwa, parametry i ich znaczenie

Semantyka:

Po co jest / co robi

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Kody błędów:

3.1.3 Spis komunikatów błędów

Nr błędu	Komunikat	Znaczenie komunikatu
0	SUCCESS	Operacja skończyła się pomyślnie
1	ACCESS_DENIED	Klient nie ma uprawnień na wykonanie operacji
2	BAD_COMMAND	Zły format ramki lub polecenia lub danych w ramce.
3	ALREADY_EXISTS	Obiekt zapytania do serwera już istnieje (n.p. przy spróbie dodać już istniejącego użytkownika)
4	GENERIC_ERROR	Na serwerze wystąpił błąd.
5	DOES_NOT_EXISTS	Obiekt zapytania do serwera nie istnieje (n.p. przy spróbie uzyskania informacji o nieistniejącym użytkowniku).

4 Przypadki użycia

4.1.1 Klient tworzy nowego użytkownika w bazie serwera

XXXXXXXXXXXXXXXXXXXXXXXXXXXX Na przykład

4.1.2 Klient loguje się do serwera

Klient wywołuje login. Na przykład

4.1.3 Klient wylogowuje się z serwera

Klient wywołuje logout

4.1.4 XXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

5 Aplikacja klienta

5.1 Wstęp

Po co jest co robi.

5.2 Moduł ***nazwa1***

Bardzo ogólnie funkcjonalność

5.3 Moduł ***nazwa2***

Bardzo ogólnie funkcjonalność

6 Aplikacja serwera

6.1 Wstęp

Po co jest co robi.

6.2 Moduł ***nazwa1***

Bardzo ogólnie funkcjonalność

6.3 Moduł ***nazwa2***

Bardzo ogólnie funkcjonalność

7 Przechowywane dane

7.1 Wstęp

7.2 Tabela/ plik1

Po co jest, rodzaj przechowywanych danych

7.3 Tabela/ plik2

8 Interfejs danych

8.1 Funkcja 1

Krótki opis (po co, zwracana wartość, parametry)

8.2 Funkcja 2

Krótki opis (po co, zwracana wartość, parametry)

8.3 ...

8.4 Spis komunikatów błędów

9 Planowany podział prac

W formie tabeli kto realizuje którą funkcję / moduł