Programowanie aplikacji klient-serwer

Projekt 6 Egzamin testowy

Projekt

(wersja 1.1)

AUTORZY PROJEKTU:

Uladzislau Harbuz

Maksym Yakymyshyn

Spis treści

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 został w językach C i C++. Dane o użytkownikach, testach i wynikach są przechowywane w bazie danych SQLite. Dodatkowo do testowania prawidłowości działania serwera zostali utworzone testy protokołu (tablica 3.1.1.2.1) w języku Haskell.

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 spróby komunikacji z serwerem poza aktywną sesją są odrzucane przez serwer.

Test – pytanie z odpowiedziami do niego.

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: {„text”:”tekst_pytania”, „variants”:[{„1”: „odp1”},{„2”:”odp2”}, …]}, 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: {„answewrs”:[{„test_n”:”ans_n”}, …]}, 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 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]. Klient dostaje w odpowiedź komunikat zwrotny z kodem błędu.
5	CLOSE_SESSION	Zamyka aktualną sesję. Kliend dostaje komunikat zwrotny z kodem błędu.
6	GET_RESULTS	Klient wysyła pustą ramkę. Dostaje od serwera ramke z polem „Dane” zawierającą wyniki testów w formacie JSON: {„pass”:”liczba_testów_zdanych”, „all”:”liczba_wszystkich_testów”}, gdzie ***pass*** – liczba prawidłowych odpowiedzi w tescie, ***all*** – liczba wszystkich testów.
7	CHANGE_CREDENTIALS	Klient wpisuje w pole „Dane” w pierwsze LOGIN_BYTE_SIZE bajt nowy login, a w następne PASSWORD_BYTE_SIZE nowe hasło. Klient dostaje komunikat zwrotny z kodem błędu.
8	ADD_GROUP	Klient wpisuje w pole „Dane” nazwę nowej grupy. Dostaje od serwera komunikat zwrotny z kodem błędu.
9	ADD_USER	Klient wpisuje w pole danych : napis w formacie JSON: "{\"login\":\"User\",\"password\":\"12345sd\",\"name\":\"Test new\",\"surname\":\"SurnameXXX\"}",gdzie ***login*** – nazwa użytkownika, ***password*** – hasło użytkownika, ***name*** – imię użytkownika, ***surname*** – nazwisko użytkownika.

		Żeby dodać nowego użytkownika trzeba mieć uprawniwnia administratora. Klient dostaje komunikat zwrotny z kodem błędu.
10	ERROR_DATAGRAM	Ten komunikat świadczy o tym, że w polu „Dane” w pierwszym bajcie znajduje się kod błędu , a w drugim bajcie - kod operacji której 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	Klient wysyła ramkę z nazwą użytkownika w polu „dane”. W odpowiedzi od serwera dostaje ramkę z polem polecenia „GET_USER_INFO”, i polem „dane” wypełnionym w formacie json zdefiniowanym następująco: {„groups”:[„grupa 1”. „grupa 2”, ...], „login”:”nazwa użytkownika”, „name”:”Imię”, „surname”:”Nazwisko”}, gdzie ***groups*** – lista grup do których należy użytkownik, ***login*** – nazwa użytkownika, ***name*** – imię użytkownika, ***surname*** – nazwisko użytkownika. W przypadku błędu lub nie istnienia użytkownika klient dostaje zwykły komunikat typu ERROR_DATAGRAM.
14	SET_USER_INFO	Ustawia dane podanego użytkownika na nowe. Klient wysyła ramkę z polem „dane” wypełnionym w formacie json zdefiniowanym w następujący sposób: {„login”:”nazwa użytkownika”, „name”:”Imię”, „surname”:”Nazwisko”}, gdzie ***login*** – nazwa użytkownika dla którego dane zostaną zmienione,

		name – nowe imię użytkownika, ***surname*** – nowe nazwisko użytkownika. Klient dostaje w odpowiedi ramkę z kodem błędu.
15	ADD_TO_GROUP	Dodaje podanego użytkownika do podanej grupy. Dla wykonania tej operacji są potrzebne uprawnienia administratora. Klient wysyła ramkę z polem „dane” wypełnionym w formacie json zdefiniowanym następująco: {„login”:”login użytkownika”, „group”:”nazwa grupy”}, gdzie ***login*** – nazwa użytkownika którego trzeba dodać do grupy, ***group*** – nazwa grupy do której zostanie dodany użytkownik. Klient dostaje w odpowiezi ramkę z kodem błędu.
16	REMOVE_FROM_GROUP	Usuwa podanego użytkownika z podanej grupy. Klient wysyła ramkę z polem danych wypełnionym w formacie json zdefiniowanym następująco: {„login”:”nazwa użytkownika”, „group”:”nazwa grupy”}, gdzie ***login*** – nazwa użytkownika który zostanie usunięty z grupy, ***group*** – nazwa grupy z której użytkownik zostanie usunięty. Użytkownik musi posiadać uprawnienia administratora.
17	ADD_TEST	Dodaje nowy test z odpowiedziami. Klient wysyła ramkę w fomacie JSON: {„text”:”tekst pytania”, „answers”:[„odp1”, „odp2”], „right_answer”:”id”}, gdzie ***text*** – tekst pytania, ***answers*** – lista możliwych odpowiedzi na pytanie oprucz prawidłowej odpowiedzi. ***right_answer*** – prawidłowa odpowiedź na pytanie. Klient musi posiadać uprawnienia egzaminatora.

		Jeżeli odpowiedź prawidłowa jest pusta klient dostaje błąd BAD_FORMAT.
18	REMOVE_TEST	Usuwa podany test z listy testów. Klient podaje w danych ramki numer testu do usunięcia. Użytkownik musi posiadać uprawnienia egzaminatora.

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.

3.1.1.3 – Grupy użytkowników

W systemie istnieje dwie domyślych grupy: Admins i Examinators. Te grupy nie mogą zostać usunięte nawet przez administratora. Grupa Admins nadaje uprawnienia administratora, czyli dodawać/usuwać użytkowników i.t.d. Grupa Examinators nadaje uprawnienia egzaminatora, czyli dodawać testy, i.t.d.

Jeden użytkownik moży być w wielu grupach jednocześnie.

3.1.2 Spis komunikatów błędów

Nr błędu	Komunikat	Znaczenie komunikatu
0	SUCCESS	Operacja skończyła się pomuślnie
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

xxxxxxxxxXXXXXXXXXXXXXXXXXX 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 XXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXX

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

Aplikacja serwera *hotest-server* służy do obsługi systemu testów opisanego w zadaniu i zapewnienia możliwości pracy aplikacji klienckich zgodnie ze specyfikacją protokołu (Tablica 3.1.1.2.1).

Dla działania serwera jest potrzebna obecność pliku bazy danych users.db, który jest dostarczany razem z kodem źródłowym aplikacji, w katalogu roboczym aplikacji. W przypadku uruchomienia aplikacji lokalnie ten plik zostanie zainstalowany przez cmake w prawidłowym miejscu, ale dla uruchomienia aplikacji jako demona należy umieścić ten plik w nowym katalogie roboczym.

Dla zbudowania aplikacji ze źródeł jest używany cmake, więc należy użyć polecenia

cmake .

make

żeby skompolować i zainstalować serwer w katalogu lokalnym.

6.2 Moduł Database

Służy do wykonania zapytań SQL do bazy danych i łączy bazę danych z logiką aplikacji.

6.3 Moduł Session

Implementuje większą część logiki aplikacji, każdy obiekt klasy Sessio przedstawie jedno połączenie zautorizowanego użytkownika.

6.4 Moduł posix_server

Odpowiada za komunikację sieciową z klientami i logowanie danych (stdout/stderr lub syslog).

6.5 Moduł hotest_protocol

Implementuje protokół aplikacji i przetwarzanie danych aplikacji na ramki protokołu w obie strony.

7 Przechowywane dane

7.1 Wstęp

7.2 Tabela/ plik1

Users.db – jest to baza danych SQLite i zawiera wszystkie dane serwera.

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