

# Quiz Game

Onesim Robert Ioan, IIA4

January 18, 2016

## Abstract

Proiectul descrie o aplicatie server-client, creata cu scopul de a implementa jocul "Quiz Game". In cadrul acestui joc, utilizatorii vor avea posibilitate sa-si testeze cunostiintele din mai multe domenii, raspunzand la o serie de intrebari de cultura generala.

## 1 Introducere

### 1.1 Despre

Quiz Game este un joc al mintii in care jucatorii trebuie sa raspunda la o serie de intrebari de cultura generala. Acesta este destinat atat jucatorilor carora le place competitia cat si celor care doresc sa-si dezvolte cunostiintele in diverse domenii. Exista multiple jocuri de acest tip, unele destinate televiziunii, dar cele mai populare sunt cele din mediul online care au milioane de utilizatori. Jocul poate fi dedicat unui numar restrans de domenii (sport, medicina, etc), in acest caz intrebarile fiind extrem de dificile si specializate, sau pot acoperi o plaja larga de discipline, caz in care, intrebarile vor testa cultura generala a concurentului.

### 1.2 Utilizare

Aplicatia "Quiz Game" va consta intr-un joc la pot participa un numar nelimitat de utilizatori, fiecare, inregistrandu-se cu un **nickname unic**. Cele patru optiuni (**Start Game**, **Leaderboard**, **Instructions** si **Quit**) ii vor fi afisate clientului, in meniul de start al jocului. In momentul in care un client doreste sa inceapa un joc nou, acestuia ii vor fi afisate 4 domenii din care va putea primi intrebari (**General**, **Biology**, **Geography** si **Math**). In momentul in care incepe jocul, utilizatorul va primi o serie de intrebari cu 4 variante de raspuns si va fi nevoit sa raspunda intr-un interval de 10 secunde. Fiecare intrebare este notata cu 10 puncte, iar la final, se va afisa punctajul final. In sectiunea **Leaderboard**, utilizatorii vor putea vizualiza clasamentul, primii 10 jucatori cu cele mai bune punctaje fiind afisati in ordine descrescatoare. De asemenea, sectiunea **Instructions** va prezenta informatii despre joc, iar prin alegerea optiunii **Quit**, utilizatorii vor putea parasii aplicatia.

## 2 Tehnologii utilizate

Proiectul va folosi diverse tehnologii, atat pentru partea de server, cat si pentru cea de client, aplicatia fiind realizata pe modelul server-client, **TCP/IP** concurent.

### 2.1 Protocolul TCP/IP

- protocol orientat conexiune-serverul va sti carui utilizator ii va transmite informatiile
- asigura transportul fara pierdere a informatiilor
- asigura sosirea pachetelor la destinatie in ordinea trimiterii lor
- controleaza fluxul de date

### 2.2 Socketuri

Comunicarea server-client se va realiza folosind socketuri. Un socket este un punct terminal de comunicare intre doua sisteme din retea, fiind o combinatie de adresa **IP** si **port**.

### 2.3 Unix - C/C++

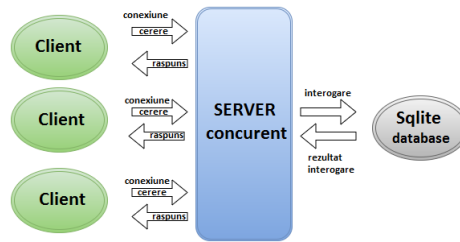
Aplicatia a fost dezvoltata si testata pe sistemul de operare Ubuntu Desktop, limbajul de programare fiind **C/C++** iar IDE-ul utilizat a fost **Code::Blocks**.

### 2.4 Sqlite

Serverul va realiza o conexiune cu o baza de date **SQLite** (sistem open-source de baze de date relationale) in care vor fi stocate intrebarile si variantele de raspuns.

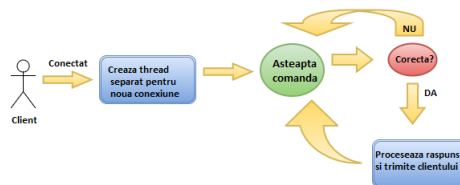
## 3 Arhitectura aplicatiei

Aplicatia este formata din doua module: serverul si clientul. Serverul, va fi implementat concurent (**multithreading server**) si va suporta oricati clienti, conexiunea la acesta realizandu-se prin intermediul IP-ului si al portului. Aplicatia client va putea fi rulata de catre utilizatori, independent, avand rolul de a realiza conexiunea cu serverul si de a permite clientilor accesul la jocul Quiz Game.



**Figura 3.1** Diagrama generala a aplicatiei

Aplicatia server are rolul de a gestiona clientii si de a stabili logica jocului. Pentru fiecare conexiune, acesta va crea un **thread** (fir de executie) diferit, astfel avand posibilitatea de a gestiona mai multi utilizatori in paralel. Serverul va raspunde la cerintele clientilor si va coordona desfasurarea jocului pentru fiecare utilizator in parte. De asemenea, validarea nickname-ului unic se va realiza in server. Acesta va fi conectat la o baza de date din care va extrage intrebari in mod random si le va trimite clientilor, de la care va astepta un raspuns timp de 10 secunde. Punctajele participantilor vor fi calculate si stocate tot in server, acestea fiind accesate prin intermediul comenzii de Leaderboard.



**Figura 3.2** Diagrama aplicatiei server

Aplicatia client are rolul de a realiza conexiunea la server si de a primi si transmite informatii catre acesta. Clientul va avea un proces de baza care se va ocupa de preluarea cerintelor si a raspunsurilor solicitate de catre utilizator si de trimiterea lor catre server. Dupa conectarea la server, utilizatorul va fi nevoit sa se inregistreze cu un nickname unic, care va fi trimis serverului. In cazul in care utilizatorul alege sa se autentifice, acesta va putea trimite comenzi serverului si va primi raspunsuri de la acesta.

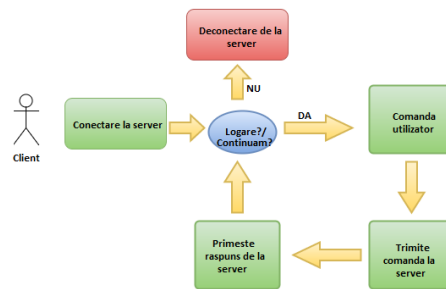


Figura 3.3 Diagrama aplicatiei client

## 4 Detalii de implementare

Partea de comunicare intre client si server se va realiza in limbajul C folosind socketuri

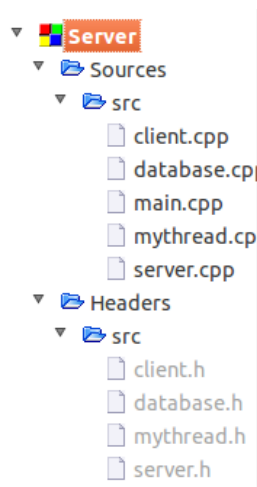
```

//Init serverSock and start listen()
serverSock = socket(AF_INET, SOCK_STREAM, 0);
memset(&serverAddr, 0, sizeof(sockaddr_in));
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = INADDR_ANY;
serverAddr.sin_port = htons(PORT);
  
```

Figura 4.1 Implementare socket

### 4.1 Server

Serverul are rolul de a gestiona clientii in mod concurent, folosind threaduri.



Aplicatia server este compusa din 4 clase: Server,Client,Thread,Database. Clasa server gestioneaza clientii (clasa Client) si foloseste Clasa Thread pentru crearea separata a threadurilor si clasa Database pentru accesarea si interogarea bazei da date.

```

38 void Server::AcceptAndDispatch() {
39
40     Client *client;
41     MyThread *thread;
42
43     socklen_t cliSize = sizeof(sockaddr_in);
44
45     while(1) {
46
47         client = new Client();
48         thread = new MyThread();
49
50         client->SetSock(accept(serverSock, (struct sockaddr *) &clientAddr, &cliSize)
51         if(client->getSock() < 0) {
52             cerr << "Error on accept";
53         }
54         else {
55             thread->Create((void *) Server::HandleClient, client);
56         }
57     }
58 }
59

```

Figura 4.3 Bucla principala in clasa Server

```

61 void *Server::HandleClient(void *args) {
62
63     int option=0;
64     //Pointer to accept()'ed Client
65     Client *client = (Client *) args;
66
67     //Add client in Static clients <vector>
68     int finish=Server::AddClient(client);
69
70     while(!finish){
71
72         int n = read(client->getSock(),&option, 4);
73         if(n == 0){
74             cout<<"[SERVER] Client "<<client->getName()<<" has been disconnected..
75             Server::RemoveClient(client);
76             finish=1;
77             break;
78         }
79         else if(n<=0) cerr<<"Error while reading..";
80         else{
81             switch(option){
82                 case 1: finish=Server::StartGame(client); break;
83                 case 2: finish=Server::ViewLeaderboard(client); break;
84                 case 3: finish=Server::SendInstructions(client);break;
85                 case 4: Server::RemoveClient(client); finish=1; break;
86                 default: cout<<"[SERVER] Comanda incorecta";
87             }
88         }
89     }
90     return NULL;
91 }

```

Figura 4.4 Functia de gestionare a clientilor

Fiecare client poate trimite una dintre cele 4 optiuni la server: Start Game, Leaderboard, Instructions si Quit. Pentru prima optiune, serverul va primi de la client domeniul din care doreste sa primeasca intrebari, urmand ca mai apoi sa-i trimita clientului o serie de intrebari si sa calculeze scorul in urma raspunsurilor primite. Pentru optiunea a doua(Leaderboard), serverul va trimite clientului clasamentul primilor 10 concurenti in ordinea punctajelor. Optiunea Instructions permite serverului sa trimita un set de instructiuni in timp ce, optiunea Quit, deconecteaza clientul si il sterge.

## 4.2 Client

Conectarea clientilor la server se va realiza folosind primitiva connect. Aplicatia client este formata din clasa Client are rolul de a realiza conexiunea cu serverul si de a gestiona cerintele utilizatorului.

```

6  Client::Client(){
7
8
9      serverSock = socket(AF_INET, SOCK_STREAM, 0);
10     server.sin_family = AF_INET;
11     server.sin_addr.s_addr = inet_addr(ADDRESS);
12     server.sin_port = htons(PORT);
13
14     if(connect(serverSock, (struct sockaddr *) &server, sizeof(struct sockaddr)) =
15         cerr<<"Error at connect()\n";
16     }
17     else{
18         Client::ClientFlow();
19     }
20 }

52     Client::ShowMenu();
53
54     while(!finish){
55
56         cin>>option;
57         if(write(serverSock, &option,4) <= 0 ){
58             cerr<<"Error while writing...\n";
59         }
60         switch(option){
61             case 1:finish=Client::StartGame(); break;
62             case 2:finish=Client::Leaderboard(); break;
63             case 3:finish=Client::ViewInstructions(); break;
64             case 4:finish=Client::Quit(); break;
65             default: cout<<"Comanda invalida, reincerati!\n";
66         }
67     }
68 }

```

**Figura 4.5 Clasa client si functia de gestionare a utilizatorilor**

Dupa inregistrare utilizatorul ii va fi prezentat un meniu ce contine cele 4 optiuni:Start Game, Leaderboard, Instructions si Quit.

## 4.3 Sqlite Database

Baza de date contine tabele pentru fiecare domeniu, avand ca attribute: ID,intrebare, raspuns1, raspuns2, raspuns3, raspuns4 si raspunsCorect.

```

quiz.sql x
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE general(
ID int primary key not null,
Intrebare char(2000) not null,
raspuns1 char(2000) not null,
raspuns2 char(2000) not null,
raspuns3 char(2000) not null,
raspuns4 char(2000) not null,
corect int not null);
INSERT INTO "general" VALUES(1,'Care este capitala Romaniei?', 'Paris', 'Londra', 'Bucuresti', 'Iasi',3);
INSERT INTO "general" VALUES(2,'Cati cromozomi are un om?', '49', '46', '44', '48',2);
INSERT INTO "general" VALUES(3,'Din cati jucatori este formata o echipa de
handbal?', 'cinci', 'sase', 'sapte', 'opt',3);
INSERT INTO "general" VALUES(4,'Unde au fost filmate peisajele din Memuritorul', 'Irlanda', 'Grecia', 'Marea
Britanie', 'Scottia',4);
INSERT INTO "general" VALUES(5,'Care este cel mai lung fluviu din
Europa?', 'Dunarea', 'Volga', 'Nil', 'Ural',2);
INSERT INTO "general" VALUES(6,'Care este capitala statului
Muntenegru?', 'Podgorita', 'Bruxelles', 'Lisabona', 'Dublin',1);
INSERT INTO "general" VALUES(7,'Care este capitala statului
Letonia?', 'Riga', 'Vilnius', 'Talin', 'Oslo',2);
INSERT INTO "general" VALUES(8,'Pe ce continent se afla cea mai mare cascada din
lume?', 'Africa', 'Europa', 'America de Nord', 'Australia',3);
INSERT INTO "general" VALUES(9,'Care este al doilea varf de munte ca inaltime din lume?', 'Manga
Parbat', 'Makalu', 'Lhotse', 'K2',4);
INSERT INTO "general" VALUES(10,'In ce tara se afla cei mai activi vulcani?', 'Indonezi', 'Noua
Zeelanda', 'Japonia', 'Italia',1);
INSERT INTO "general" VALUES(11,'Tegucigalpa este capitala?', 'Guatemala', 'Honduras', 'Panama', 'El
Salvador',2);

```

Figura 4.6 SQL baza de date

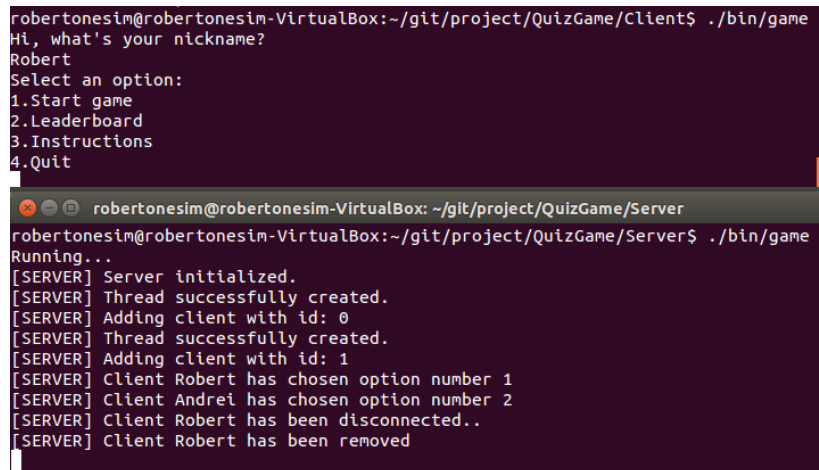
```

16 vector<vector<string> > Database::query(char *query){
17
18     sqlite3_stmt *statement;
19     vector<vector<string> > results;
20
21     if(sqlite3_prepare_v2(database, query, -1, &statement, 0) == SQLITE_OK){
22         int cols=sqlite3_column_count(statement);
23         int result=0;
24         while(true){
25
26             result = sqlite3_step(statement);
27
28             if(result == SQLITE_ROW){
29                 vector<string> values;
30                 for(int col=0; col<cols; col++){
31                     values.push_back((char*)sqlite3_column_text(statement, col));
32                 }
33                 results.push_back(values);
34             }
35             else break;
36         }
37         sqlite3_finalize(statement);
38     }
39     string error = sqlite3_errmsg(database);
40     if(error!="not an error") cout<<query<<" "<<error<<"\n";
41
42     return results;
43 }
44

```

Figura 4.7 Implementare query

## 4.4 Aplicatia finala



```
robertonesim@robertonesim-VirtualBox: ~/git/project/QuizGame/Client$ ./bin/game
Hi, what's your nickname?
Robert
Select an option:
1.Start game
2.Leaderboard
3.Instructions
4.Quit

robertonesim@robertonesim-VirtualBox: ~/git/project/QuizGame/Server
robertonesim@robertonesim-VirtualBox:~/git/project/QuizGame/Server$ ./bin/game
Running...
[SERVER] Server initialized.
[SERVER] Thread successfully created.
[SERVER] Adding client with id: 0
[SERVER] Thread successfully created.
[SERVER] Adding client with id: 1
[SERVER] Client Robert has chosen option number 1
[SERVER] Client Andrei has chosen option number 2
[SERVER] Client Robert has been disconnected..
[SERVER] Client Robert has been removed
```

Figura 4.8 Aplicatia finala

## 5 Concluzii

Proiectul "Quiz Game" este o aplicatie interactiva prin care utilizatorii isi vor putea testa cunostiintele de cultura generala. Aplicatia ar putea fi imbunatatita prin construirea unui chat in care utilizatorii sa poata sa discute pe baza intrebarilor aparute dar si o functie prin care sa fie permisa adaugarea noilor intrebarilor de catre utilizatori

## 6 Bibliografie

1. Cursurile de *Rețele și calculatoare* 2015-2016, Facultatea de informatica, Universitatea Alexandru Ioan Cuza Iasi  
(<https://profs.info.uaic.ro/adria/teach/courses/net/>)
2. SQLite c/c++ Tutorial ([www.tutorialspoint.com/sqlite](http://www.tutorialspoint.com/sqlite))
3. C/C++ reference ([cppreference.com](http://cppreference.com))