Trivia

Iusan Flaviu

Cuprins

1. Introducere

1.1 Context

Participarea la o competitie de trivia este o activitate interesanta, distractiva si un motiv pentru a te intalnii cu prietenii. Multe localuri si puburi organizeaza astfel de competitii pentru a ajuta clientii sa se destinda si sa uite de problemele cotidiene prin creearea unei atmosfere linistitoare si placute si prin incurajarea la discutii interesante despre subiectele abordate in timpul competitiei. La astfel de competitii participa de obicei multi oamenii astfel ca organizatorul unei astfel de competitii are nevoie de o modalitate de a trimite intrebarile la participanti, de a primii raspunsurile de la acestia si de a comunica scorurile participantiilor cat mai eficient posibil pentru a nu creea confuzie cu privire la, de exemplu: intrebarea actuala la care trebuie gasit raspunusl, cine a raspuns corect primul si punctajele participantiilor, lucru care ar putea sa cauzeze scaderea interesului in participarea la competitie, tulburarea atmosferei sau certuri intre participantii. La majoritatea competitiilor exista un “ albitru” care sa fie responsabil de aceste lucruri. Acesta transmite in scris si oral intrebarile, scorurile si alte informatii participantiilor iar, participantii transmit cu voce tare raspunsul sau il trimit in scris, pe o foaie, arbitrului. In acest caz este foarte probabil sa apara problemele descrise mai sus deoarece arbitrul ar fi nevoit sa faca mai multe lucruri deodata in acelasi timp, de exemplu: Arbitrul verifica raspunsurile primite si de fiecare data ar trebuii sa anunte fiecare participant in cazul in care a raspuns gresit pentru a continua sa caute alt raspuns, insa daca albitrul primeste multe raspunsuri deodata atunci acest proces va creea un timp mare de asteptare pentru validarea raspunsului iar participantii nu vor stii daca sa caute sau nu un alt raspuns la intrebare.

Aceste competitii ar putea devenii mult mai organizate si usor de urmarit daca participantii ar avea la dispozitie un canal de comunicare usor de folosit pentru primii intrebari si pentru a trimite raspunsuri si daca ar avea acces rapid la toate informatiile de care au nevoie in timpul acesteia, de exemplu daca raspunsul dat este corect sau nu. De asemenea ar fi de folos un istoric al fiecarei competitii cu intrebarile si raspunsurile date in timpul acesteia ca dovada a corectitudinii scorurilor si o baza de date cu intrebari organizate pe subiecte pentru ca organizatorul sa poata generea seturi de intrebari si sa adauge sau stearga intrebari.

2. Obiectivele proiectului

2.1Obiective generale

Obiectivul proiectului este de a creea un mod eficient de organizarea a competitiilor trivia pentru a facilita o desfasurare corecta a competitiei si o atmosfera placuta si linistita pentru participanti.

2.2Obiective specifice

Proiectul trebuie sa ofere o intrefata usoara pentru gestionarea seturilor de intrebari, creearea unei comeptitii trivia, salvarea istoricului competitiei si autentificarea participantiilor pentru organizatori si pentru participanti un canal de comunicatie pe care sa trimita raspunsuri si sa poata primii informatii.

2.2.1 Aplicatia Android pentru participanti

In aplicatia Android pentru participanti, acestia trebuie mai intai sa se autentifice/inregistreze si optional, daca doresc, sa isi schimbe numele contului, la inregistrare acesta este setat automat sa fie id-ul unic al contului respectiv. Dupa ce s-au autentificat cu succes, acestia vor putea sa se conecteze la canalul de comunicatie prin internet.

2.2.2 Aplicatia Android pentru organizatori

In aplicatia Android pentru organizatori, acestia pot sa faca modificari in baza de date cu intrebari, sa sa salveze istoricul unei competitii si sa creeze o competitie noua. La creearea unei competitii noi, aplicatia va creeea un server la care participanti se vor putea conecta iar, acest server va trimite intrebari si informatii la participanti, va primii raspunsuri de la participanti pe care le va verifica si va atribuii punctaje participantiilor.

3. Aspecte de proiectare

3.1Arhitectura generala

Arhitectura proiectului Trivia este de timp client-server. Aplicatia client contine doua componente:

- Meniul aplicatiei, unde clientii se pot inregistra si autentifica in cont, pot schimba numele contului si se pot conecta la server. Inregistrarea si autentificare trimit catre un site web unde clientii isi vor introduce datele si daca inregistrarea sau autentificarea se indeplinesc cu succes atunci clientii vor fi retrimisi in meniul aplicatiei iar interfrata meniului se v-a incarca cu datele contului clientului. Conectarea la server trimite datele contului la server si deschide interfata jocului.
- Interfata de joc, unde clientii pot trimite si pot primii informatii de la server. Aceasta este actualizată prin returnarea răspunsurilor de la server.

Aplicatia server contine trei componente:

- Interfata aplicatiei de unde se poate porni serverul, adauga intrebari si salva istoricul competitiei.
- Un server tcp care accepta conectiuni, primeste mesaje si trimite informatii si implementeaza regulile competitiei trivia. Acesta ruleaza pe un thread unde accepta conectiuniile de la clienti iar dupa accea creaza un thread pentru fiecare client unde asculta si trimite inforamtii si actualizeaza scorurile intr-o baza de date.
- O baza de date unde sunt stocate datele clientiilor si seturile de intrebari.

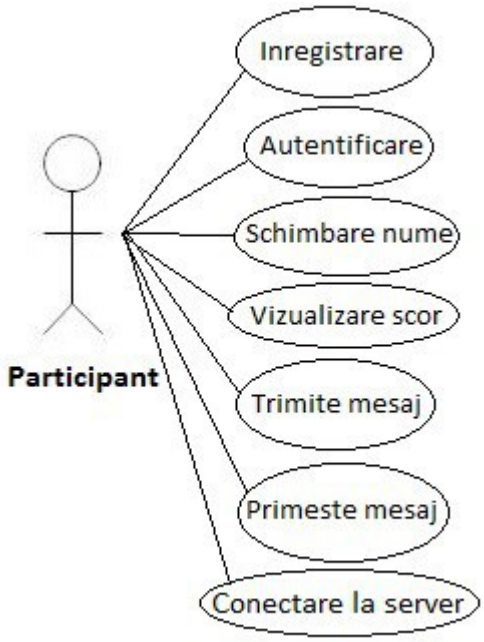
3.2Cerinte functionale

Participantul poate sa acceseze site-ul pentru inregistrare si autentificare din interfata de meniu. Acesta isi poate vedea numele contului, pe interfata, si il poate schimba inainte de a se conecta la server. Cand se conecteaza la server, interfata aplicatiei se actualizeaza cu interfata de joc unde acesta va primii intrebari si informatii de la server si poate trimite raspunsuri la server. Participantul poate sa isi vada scorul actualizat in orice moment pe interfata de joc.

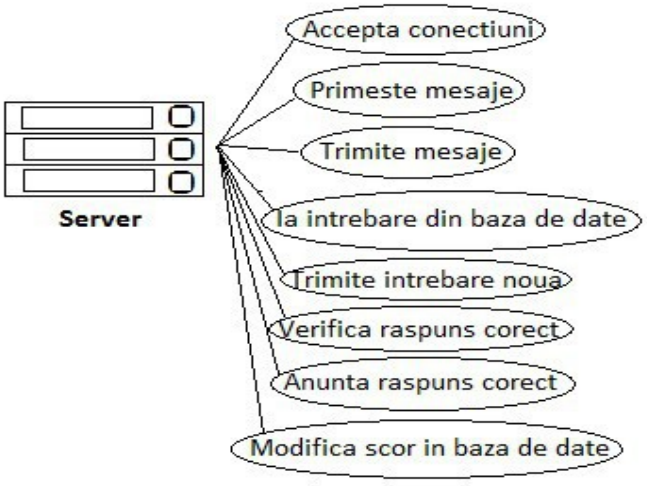
Organizatorul foloseste aplicatia server unde poate sa porneasca serverul, sa adauge intrebari si sa salveze istoricul unei competitii pe memoria locala a dispozitivului mobil.

3.3Cazuri de utilizare

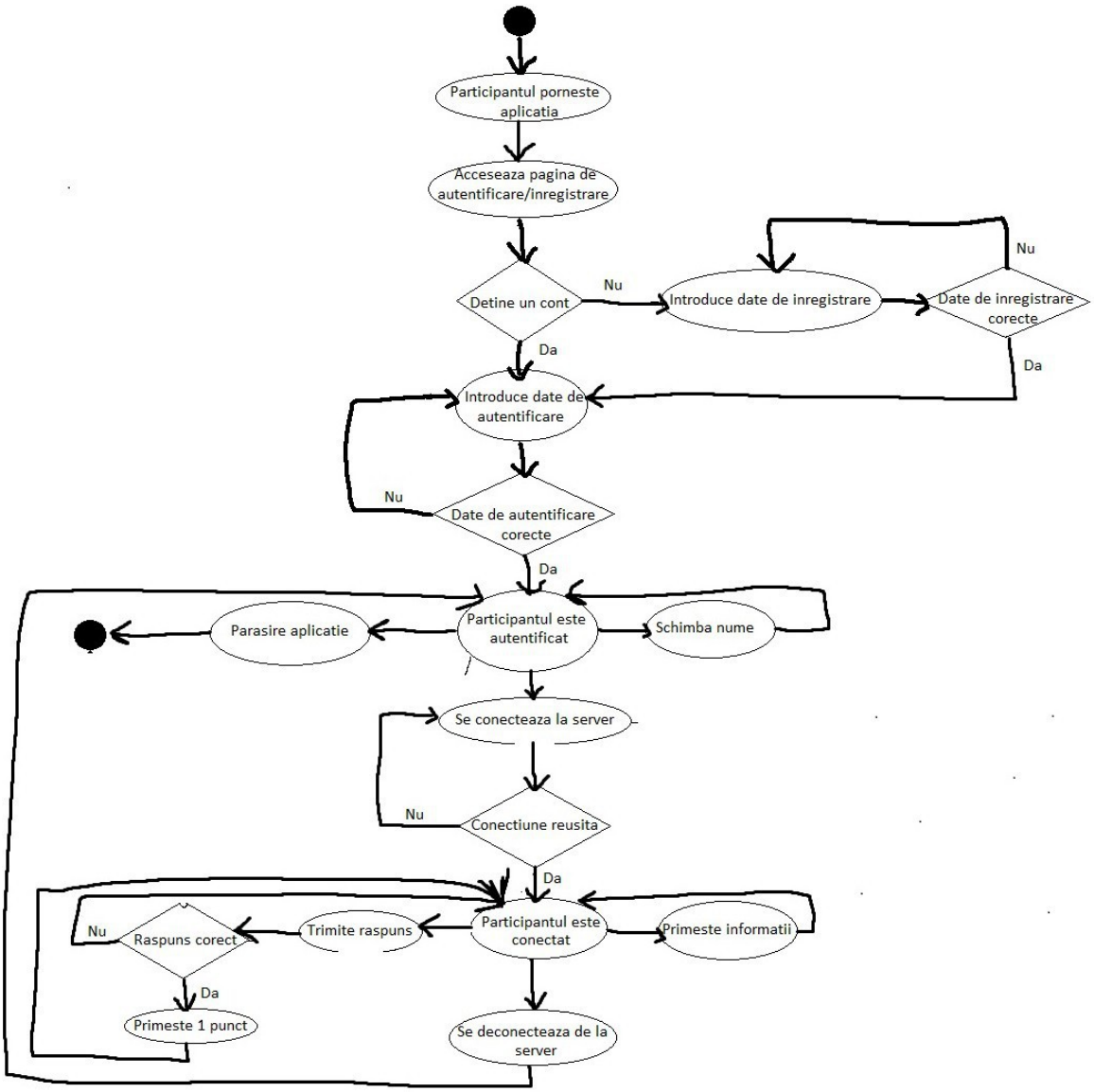
3.3.1 Cazuri de utilizare Participant



3.3.2 Cazuri de utilizare Organizator



3.4Diagrama de stare



4. Detalii de implementare

4.1Implementarea API-ului Auth0

```
final Auth0 auth0 = new Auth0( context: MainActivity.this);
final AuthenticationAPIClient authentication = new AuthenticationAPIClient(auth0);
auth0.setOIDCConformant(true);
auth0.setLoggingEnabled(true);

WebAuthProvider.login(auth0) //open login page
        .withScheme("demo")
        .withAudience(String.format("https://%s/userinfo", "dev-rt4c3v5j.eu.auth0.com"))
        .start( activity: MainActivity.this, new AuthCallback() {
            @Override
            public void onFailure(@NonNull Dialog dialog) {
                // Show error Dialog to user
            }

            @Override
            public void onFailure(AuthenticationException exception) {
                // Show error to user
            }

            @Override
            public void onSuccess(@NonNull final Credentials credentials) {

                userCredentials = credentials;
                Log.e( tag: "Login", userCredentials.getAccessToken());

                authentication
                        .userInfo(userCredentials.getAccessToken())
                        .start(new BaseCallback<UserProfile, AuthenticationException>() {
                            @Override
                            public void onSuccess(UserProfile information) {
                                user.userId = information.getId();
                                Log.e( tag: "User Id", user.userId);

                                databaseReference.addListenerForSingleValueEvent(new ValueEventListener() {
                                    @Override
                                    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                                        if (dataSnapshot.child("users").hasChild(user.userId)) {
                                            user.score = (long) dataSnapshot.child("users").child(user.userId).child("score").getValue();
                                            user.username = (String) dataSnapshot.child("users").child(user.userId).child("username").getValue();
                                            fragment.changeTextInTextViewUsername(user.username);
                                            fragment.changeTextInTextViewScore(Integer.valueOf((int)user.score).toString());

                                        } else {
                                            Log.e( tag: "else", msg: "else else");
                                            databaseReference.child("users").child(user.userId).setValue(new User(user.userId,  score: 0,  userId: null));
                                            user.username = user.userId;
                                            user.score = 0;
                                            fragment.changeTextInTextViewUsername(user.username);
                                            fragment.changeTextInTextViewScore(Integer.valueOf((int)user.score).toString());
                                        }
                                    }

                                    @Override
                                    public void onCancelled(@NonNull DatabaseError databaseError) {

                                    }
                                });
                            }

                            @Override
                            public void onFailure(AuthenticationException error) {
                                //user information request failed
                            }
                        });
            }
        });
```

4.2Conectarea la server

```
@SuppressLint("StaticFieldLeak")
public void connectToServer(){
    (AsyncTask) (voids) → {
            try {
                socket = new Socket( host: "192.168.0.110",  port: 8888);
                send = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())),  autoFlush: true
                get = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            }catch (Exception ex){
                Log.e( tag: "connection", ex.toString());
            }
            return null;
    }.execute();
}
```

4.3Ascultarea de informatii de la server

```
private final class listenRun implements Runnable{

    @Override
    public void run() {
        while(true) {
            try {
                String line;
                if(get != null) {
                    if ((line = get.readLine()) != null) {
                        if(line.toString().compareTo("Server: " + user.userId + " a raspuns corect !!!")==0){
                            Intent intent = new Intent();
                            intent.setAction("UpdateUserScore");
                            sendBroadcast(intent);
                        }
                        else {
                            Log.e( tag: "listenRunFunc",  msg: "Mesajul primit este " + line);
                            listaMesaje.add(line);

                            Intent intent = new Intent();
                            intent.setAction("RefreshMesajeView");
                            sendBroadcast(intent);
                        }
                    }
                        Log.e( tag: "listenRunFunc",  msg: "Mesajul a fos null");
                }


            } catch (Exception e) {
                Log.e( tag: "Exceptie primit", e.toString());
            }
            try {
                Thread.sleep( millis: 500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

4.4Trimitere de mesaje la server

```
private final class sendRun implements Runnable{

    @Override
    public void run() {
        try {
            if(firstMessage){
                send.println(user.userId + "- " + user.username + ": " + mesaj);
                Log.e( tag: "sendRunFunc",  msg: "S-a trimis" + user.username);
                firstMessage = false;
            }
            else {
                send.println(user.username + ": " + mesaj);
                Log.e( tag: "sendRunFunc",  msg: "S-a trimis" + user.username);
            }

        } catch (Exception e) {
            Log.e( tag: "erroare trimis",  msg: "Nu s-a trimis");
        }
    }
}
```

4.5Reimprospatarea interfetei de joc cu ultimele informatii primite

```
@Override
public void onReceive(Context context, Intent intent) {

    String actiunea = intent.getAction();
    if(actiunea.compareTo("RefreshMesajeView")==0){
        mesajeAdapter.notifyItemInserted(mesajeAdapter.getItemCount());
        recyclerView.smoothScrollToPosition(mesajeAdapter.getItemCount());
    }
    if(actiunea.compareTo("UpdateUserScore")==0){
        user.score = user.score + 1;
        fragment.changeTextInTextViewScore(Integer.valueOf((int)user.score).toString());
    }
}
```

4.6Schimbare nume cont

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(data != null) {
        Bundle results = data.getExtras();
        user.username = results.getString( key: "Username");
        Log.e( tag: "Result", user.username);
        databaseReference.child("users").child(user.userId).child("username").setValue(user.username);
        fragment.changeTextInTextViewUsername(user.username);
    }

}
```

```java
public class AccountSettingsActivity extends AppCompatActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.accountsettings_activity);

        Intent intent = getIntent();
        Bundle extras = intent.getExtras();
        final AppCompatEditText EditTextUsername = findViewById(R.id.EditTextUsername);
        Log.e( tag: "Username",  msg: extras.getString( key: "Username") + "jkl");
        EditTextUsername.setText(extras.getString( key: "Username"));
        findViewById(R.id.buttonSaveSettings).setOnClickListener((v) -> {
                if(EditTextUsername.getText().toString().compareTo("")!=0) {
                    Intent intent = new Intent();
                    intent.putExtra( name: "Username", EditTextUsername.getText().toString());
                    setResult(Activity.RESULT_OK, intent);
                    finish();
                }
                else{
                    Toast.makeText( context: AccountSettingsActivity.this,  text: "Introduceti un username !!!", Toast.LENGTH_SHORT).show();
                }
        });
    }
}
```

4.7Implementarea serverului

```
lateinit var database: FirebaseDatabase
lateinit var databaseReference: DatabaseReference

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val listaSocketClienti: ArrayList<Socket> = ArrayList()
    var question = ""
    var answer = ""

    findViewById<Button>(R.id.buttonStartServer).setOnClickListener { it: View!
        Log.e( tag: "Button",  msg: "Apasat")
        database = FirebaseDatabase.getInstance()
        databaseReference = database.getReference()
        val server = ServerSocket( port: 8888, backlog: 10, InetAddress.getByName( host: "0.0.0.0"))
        thread { //Thread acceptare conectiune clienti
            run { this: MainActivity
                Log.e( tag: "first run check server", server.inetAddress.hostAddress)
                Log.e( tag: "first run",  msg: "waiting for clients")
                while (true) {
                    val client = server.accept()
                    Log.e( tag: "first run",  msg: "got a client")
                    listaSocketClienti.add(client)

                    Log.e( tag: "first run",  msg: "added client")
                    thread { //Thread ascultare client, trimitere mesaje la client, game logic
                        run { this: MainActivity
                            Log.e( tag: "second run",  msg: "waiting for messages")
                            val reader = BufferedReader(InputStreamReader(client.getInputStream()))
                            var userId = ""
                            var userName = ""
                            var score : Long = -1
                            var firstMessage = true
                            var indexSfarsitUserId : Int = -1
                            var text : String
```

4.8Ascultarea si trimiterea mesajelor

```
while(true) {
    try { //aici primesc mesajele de la clienti
        val citeste = reader.readLine()
        text = citeste
        if(userId.compareTo("")==0 &&userName.compareTo("")==0 && text != null){
            databaseReference.addListenerForSingleValueEvent(object : ValueEventListener {
                override fun onDataChange(dataSnapshot: DataSnapshot) {
                    indexSfarsitUserId = text.toString().indexOf( string: "-")
                    userId = text.toString().substring(0, indexSfarsitUserId)
                    userName = dataSnapshot.child( path: "users").child(userId).child( path: "username").getValue() as String
                    score = dataSnapshot.child( path: "users").child(userId).child( path: "score").getValue() as Long
                }

                override fun onCancelled(databaseError: DatabaseError) {}
            })
            while(userName.compareTo("")==0 && score.compareTo(-1)==0){
                Log.e( tag: "Waiting for data user",  msg: "Still waiting")
            }
            Log.e( tag: "initializat user",  msg: userName + " " + score.toString()) //score se aloca corect dar afiseaza -1 in consola / functioneaza

        }
        if(text != null){
            if(firstMessage){
                val firstText = text.toString().substring(indexSfarsitUserId, text.toString().length)
                text = firstText
                firstMessage = false
            }
            for(clientSocket in listaSocketClienti){
                val writer = PrintWriter(BufferedWriter(OutputStreamWriter(clientSocket.getOutputStream())),  autoFlush: true)
                writer.println(text)
            }
        }
```

4.9Verificarea raspunsurilor si adaugarea punctelor

```
if(text.toString().contains(answer) && answer!=""){
    //adauga punctaj in baza de date
    score=score+1
    databaseReference.child( pathString: "users").child(userId).child( pathString: "score").setValue(score)

    for(clientSocket in listaSocketClienti){
        val writer = PrintWriter(BufferedWriter(OutputStreamWriter(clientSocket.getOutputStream())),  autoFlush: true)
        Log.e( tag: "send announcement answear correct", question)
        writer.println("Server: " + userName + " a raspuns corect !!!")
        writer.println("Server: " + userId + " a raspuns corect !!!")

    }
    question = ""
    answer = ""
    val randomQuestion = Random.nextInt( from: 1,  until: 3)
    databaseReference.addListenerForSingleValueEvent(object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            question = dataSnapshot.child( path: "questions").child(randomQuestion.toString()).child( path: "question").getValue().toString()
            answer = dataSnapshot.child( path: "questions").child(randomQuestion.toString()).child( path: "answer").getValue().toString()
            Log.e( tag: "for database question", question)
            Log.e( tag: "for database answer", answer)


        }

        override fun onCancelled(databaseError: DatabaseError) {}
    })
    while(question.compareTo("")==0 && answer.compareTo("")==0){
        Log.e( tag: "Waiting for data question",  msg: "Still waiting");
    }

    for(clientSocket in listaSocketClienti){
        val writer = PrintWriter(BufferedWriter(OutputStreamWriter(clientSocket.getOutputStream())),  autoFlush: true)
        Log.e( tag: "send question", question)
        writer.println("Server: " + question)
    }
}
```

4.10 Diagrama de clase si activitati