Immagine che contiene nero, oscurità

Descrizione generata automaticamente

PROGETTO “GUESS THE WORD”

LABORATORIO DI SISTEMI OPERATIVI

FEDERICO II - NAPOLI

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Prof.ssa:** | Alessandra Rossi |  |
|  |  |  |
| **Autori:** | Federico Gargiulo  Cristian Carrella  Gianmarco Lembo | NA86002884  NA86003677  NA86003940 |
|  |  |  |

**INDICE**

[INTRODUZIONE 2](#_Toc627659583)

[IMPLEMENTAZIONE DEL SERVER 3](#_Toc2089413652)

[Funzionamento delle richieste 3](#_Toc1606897785)

[Accettazione dei nuovi client 3](#_Toc1314934499)

[Strutture utilizzate 4](#_Toc712071737)

[Generazione delle parole e delle definizioni 4](#_Toc803841344)

[IMPLEMENTAZIONE DEL GIOCO 5](#_Toc263578327)

[Lobby di gioco 6](#_Toc1123412567)

[Start di una partita 6](#_Toc1510237335)

[Svolgimento del gioco 6](#_Toc1217113103)

[Fine di una partita 8](#_Toc1743661618)

[README 8](#_Toc1143818903)

# INTRODUZIONE

“Guess the word” è un’applicazione sviluppata - secondo le specifiche della traccia:

* Client: Android - Java/XML
* Server: C

Il server, mediante l’ausilio della libreria libpq, ha un collegamento ad un database relazionale PosgreSQL dove sono conservate le informazioni degli utenti che si registrano all’applicazione.  
Per l’utilizzo dei JSON lato server abbiamo utilizzato la libreria json-c.

Gli utenti possono creare un account tramite l’inserimento di una e-mail, una password ed un username. Una volta loggati con il proprio account gli utenti si troveranno nella home page, da cui è possibile accedere a 3 funzionalità:

* **Creazione di una stanza di gioco:** l’utente potrà creare una propria stanza di gioco, specificandone il nome, il numero massimo di giocatori e il numero di parole da indovinare per vincere.
* **Partecipare ad una stanza di gioco esistente:** da qui sarà possibile visualizzare una lista delle stanze di gioco create dagli altri utenti ed unirsi ad una di esse.
* **Cambiare il proprio avatar:** l’utente potrà personalizzare il proprio avatar scegliendone uno tra quelli predefiniti presenti nell’app.

Il gioco inizierà solamente quando lo avvierà l’admin della partita (ovvero il creatore della stanza), a questo punto la stanza non accetterà ulteriori partecipanti. Si è scelto come aiuto l’inserimento a schermo di una breve definizione della parola. Il gioco termina quando un partecipante raggiunge il numero di parole indovinate prestabilito; gli verrà inoltre assegnata permanentemente una vittoria.

# IMPLEMENTAZIONE DEL SERVER

## **Funzionamento delle richieste**

Affinché il server possa elaborare le richieste del client, quest'ultimo dovrà inviare al server le richieste in formato JSON. Le richieste verranno analizzate per determinare il tipo di operazione da eseguire. Il server leggerà il campo "operation" del JSON e, in base al suo contenuto, invocherà la corrispondente funzione che eseguirà l'operazione e restituirà un JSON appositamente creato.

Es: Client - invia *{ operation: login, email: email@example.com, password: pwd }*

Server - capisce che l’operazione è di login allora chiama la funzione *login(email, password)* e successivamente costruisce il JSON di risposta *{ logged : true }*

# Immagine che contiene schermata, cartone animato Descrizione generata automaticamente

## **Accettazione dei nuovi client**

Quando un utente si connette al server, al momento dell’accept(), quest’ultimo crea un thread che andrà a gestire tutte le richieste corrispondenti ad esso. Il thread dell’utente terminerà la sua esecuzione nel momento in cui la socket relativa al client si chiude.

Immagine che contiene schermata, Policromia, linea

Descrizione generata automaticamente

## **Strutture utilizzate**

|  |  |  |
| --- | --- | --- |
| **Utente** |  | **Stanza** |
| char username[32] |  | int idStanza |
| char passw[16] |  | char nomeStanza[16] |
| char email[64] |  | int numeroMaxgiocatori |
| int partiteVinte |  | int rounds |
| int idStanza |  | bool started |
| int clientSocket |  | utente \*adminUser |
| int imgId |  | utente \*\*players |
| pthread\_t tid |  | pthread\_mutex\_t roomMutex |

La struttura utente si occupa di gestire gli utenti loggati, mentre la struttura stanza si occupa di tenere traccia e gestire tutti i dati relativi alle stanze di gioco in attesa dell’inizio.

Il server conserva un’array di puntatori a stanza contenente tutte le stanze create dagli utenti, per gestire gli accessi concorrenti a tali strutture si utilizza il sistema di sincronizzazione dei **pthread\_mutex**.

## **Generazione delle parole e delle definizioni**

Le parole presenti nel gioco sono generate completamente in modo casuale effettuando delle richieste a delle API fornite in modo gratuito da:

* random-word-api.herokuapp.com: per generare la parola in modo casuale
* api.dictionaryapi.dev: da cui prendere le definizioni utilizzate come aiuto per indovinare la parola

Immagine che contiene schermata, linea, cerchio, connettore

Descrizione generata automaticamente

# IMPLEMENTAZIONE DEL GIOCO

## **Lobby di gioco**

La lobby di gioco è quella fase dopo la creazione di una stanza in cui altri utenti possono aggiungersi alla partita fino a quando l’utente admin non decide di iniziare il gioco.  
In questa activity gli utenti hanno la possibilità di visualizzare l’username di tutti gli altri utenti presenti nella stanza. Per rimanere aggiornati sugli utenti presenti i client effettuano periodicamente una richiesta di update al server.

# 

## **Start di una partita**

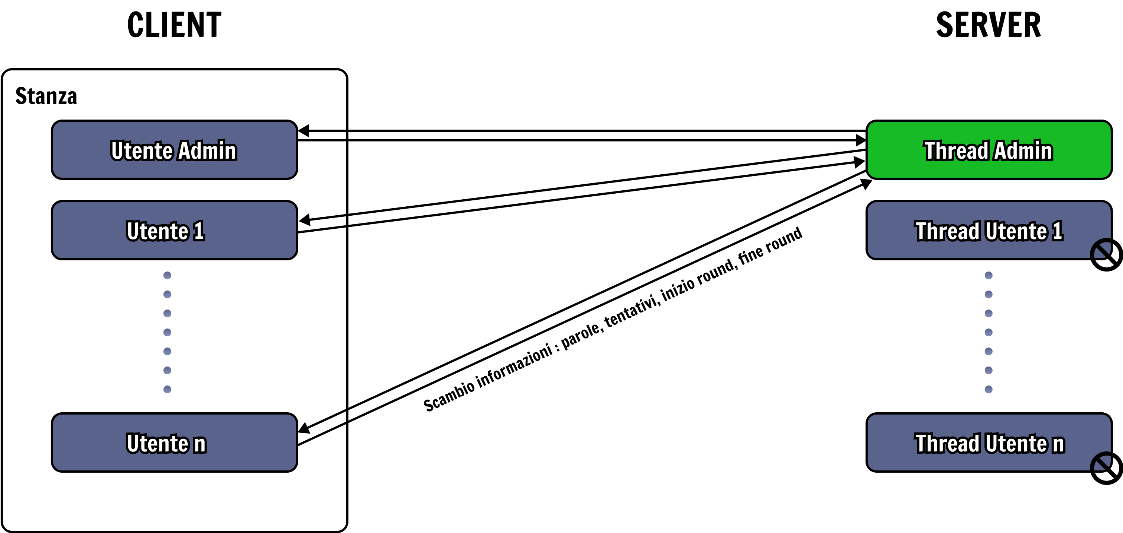
Il gioco comincia nel momento in cui l’admin invia una richiesta di “startGame” al server. Questa operazione imposta all’interno della struttura stanza la variabile booleana “started” a true, e non sarà quindi più permesso a ulteriori utenti di partecipare. Per evitare problemi di comunicazione relativi alla sincronizzazione, prima di iniziare il server aspetta che tutti i client abbiano fatto una richiesta di update.  
I client, tramite la richiesta di update, accedono alla struttura stanza e leggono il campo “started”, quando impostato a true i thread utente si bloccano con una pause() e la gestione della comunicazione passa completamente al thread dell’admin della partita, che accede alle socket tramite la struttura stanza.

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

## **Svolgimento del gioco**

Da questo momento la logica di gioco lato client viene affidata alla classe GameManager che estende la classe Thread e si occupa delle operazioni di rete e della modica dell’UI.



Fase pre-round

Il server designa il giocatore che dovrà scegliere la parola, dopodiché segnalerà ai client l’operazione da svolgere attraverso l’invio di una stringa:

* *WAIT:* il client si mette in attesa di ricevere la parola scelta dall’utente designato
* *CHOOSE:* il client si mette in attesa di ricevere un JSON contenente le parole tra cui l’utente potrà scegliere e le relative definizioni, nel seguente formato:  
   { “word0” : [“parola0”, “definizione0”] }  
   …  
   { “word4” : [“parola4”, “definizione4”] }
* *END:* la partita è finita e il client può procedere a rimuovere la stanza e tornare alla Home Activity. Il server provvederà a effettuare le opportune modifiche ai dati.

Fase di svolgimento del round

Una volta che tutti i client avranno ricevuto la parola scelta inizierà la fase di svolgimento del round. I client aspetteranno ciascuno il loro turno il cui inizio verrà segnalato dal server attraverso l’invio della stringa “*YOUR\_TURN*”. All’inizio del turno verrà attivato un timer di 15 secondi entro i quali l’utente potrà effettuare il proprio tentativo di indovinare la parola. Dopo che tutti gli utenti avranno tentato di indovinare la parola il server genererà in maniera randomica un aiuto che consiste nello sblocco di una lettera della parola. Informazioni aggiuntive potranno essere ricevute dal client durante l’attesa del turno:

* *NEW\_HINT:* il client si mette in attesa di ricevere un JSON contenente la lettera generata dal server e la posizione all’interno della parola:  
   *{“letter” : “lettera”, “position” : posizione}*
* *HINT\_END:* gli aiuti possibili sono terminati, il punto viene assegnato all’utente che ha scelto la parola e si ritorna alla fase pre-round.
* Una stringa in formato JSON contenente informazioni sui tentativi degli altri giocatori:
  + Guessed: valore booleano che indica se la parola è stata indovinata.
  + Word: stringa contenente la parola con cui ha provato ad indovinare un altro giocatore.
  + PlayerName: stringa contenente l’username del giocatore che ha provato ad indovinare.

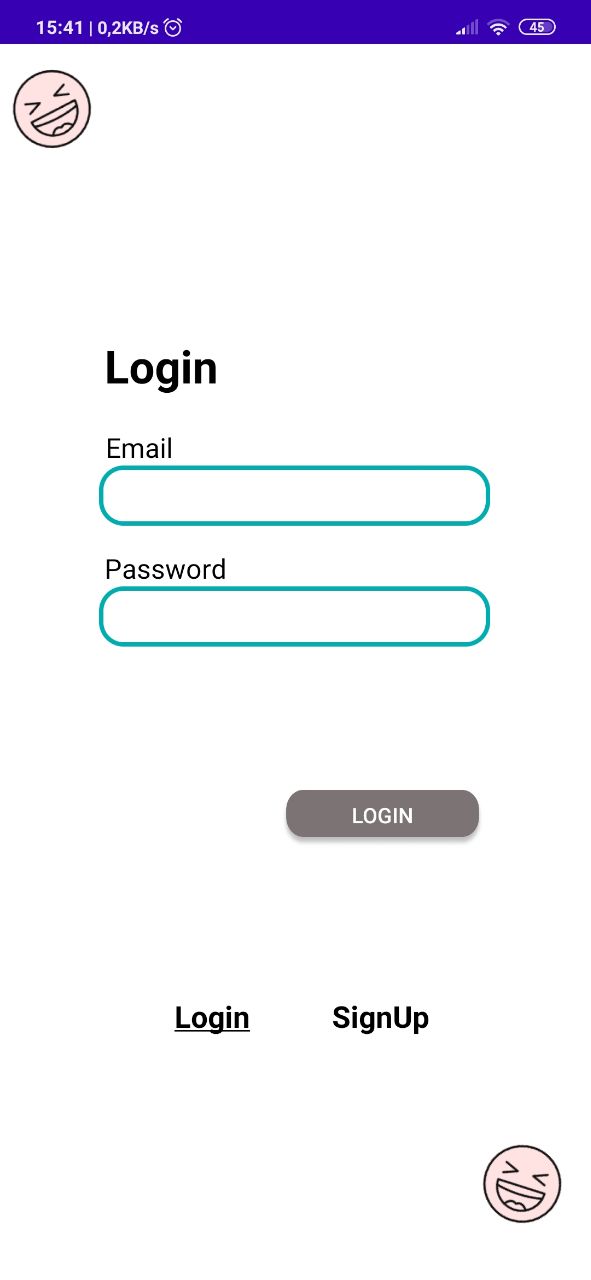
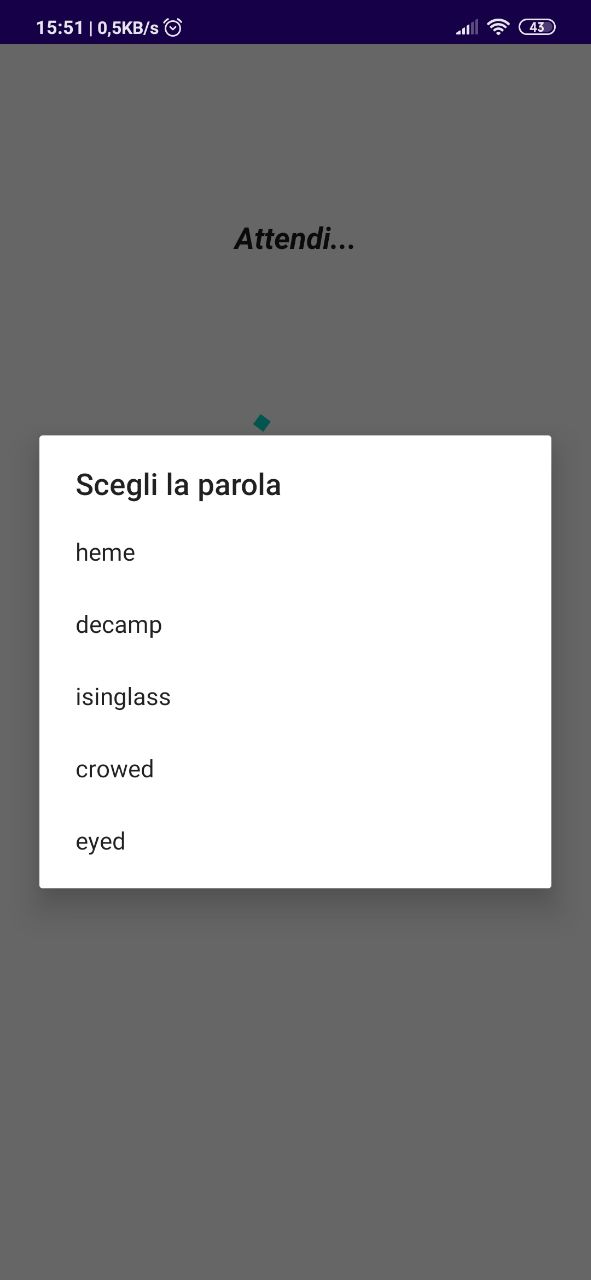
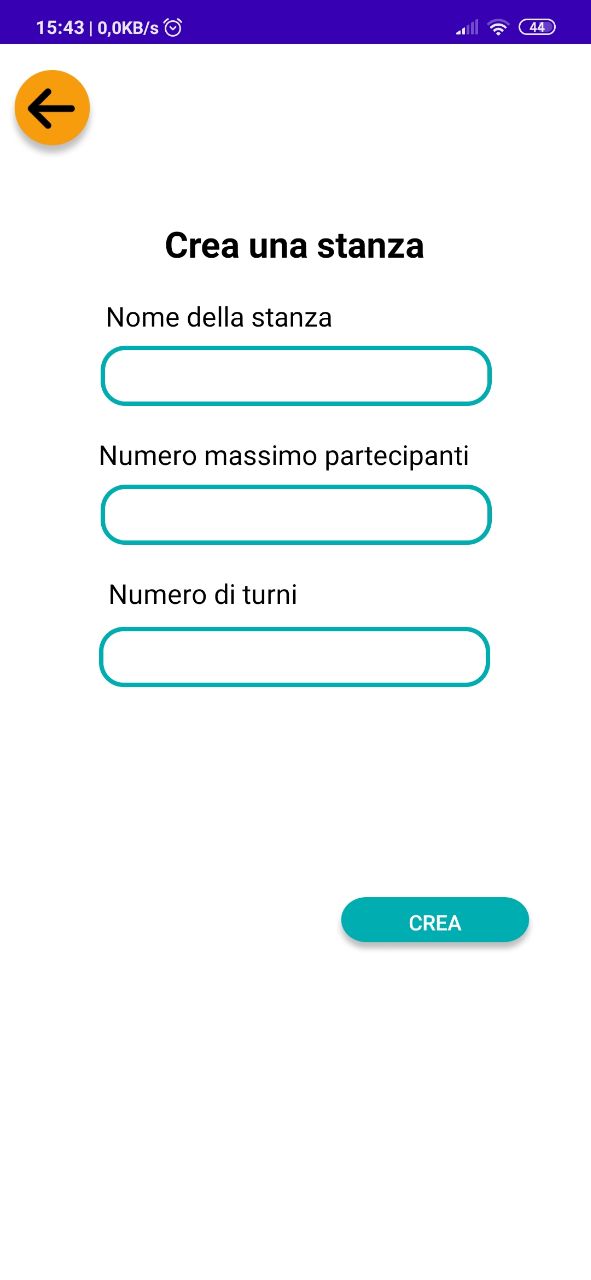
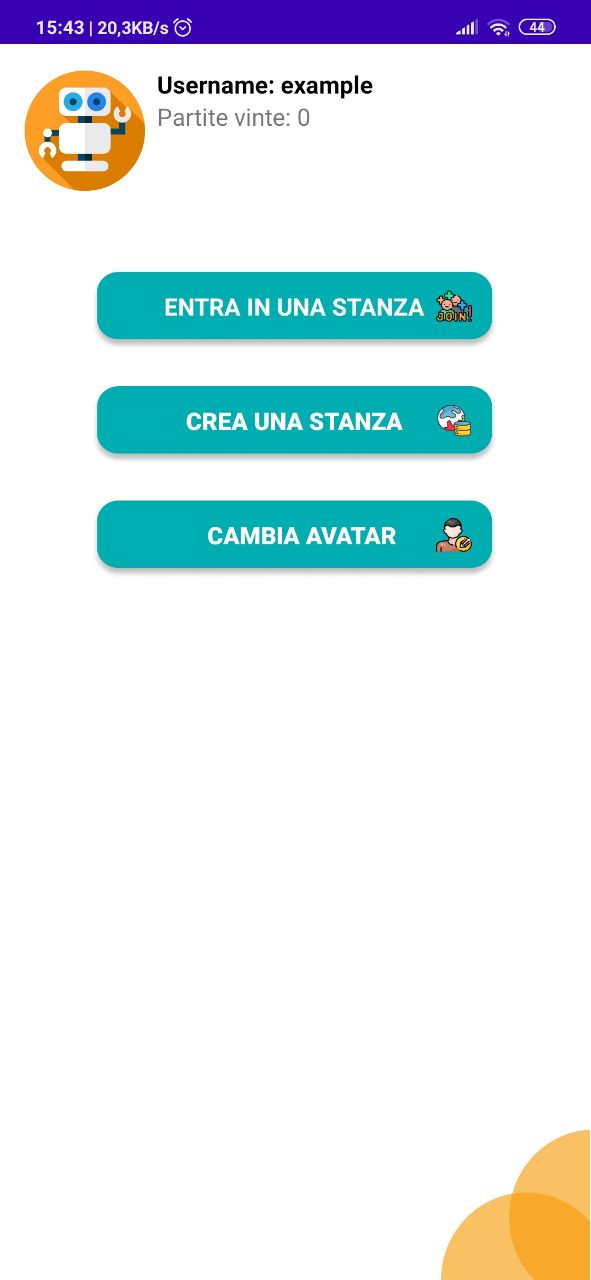
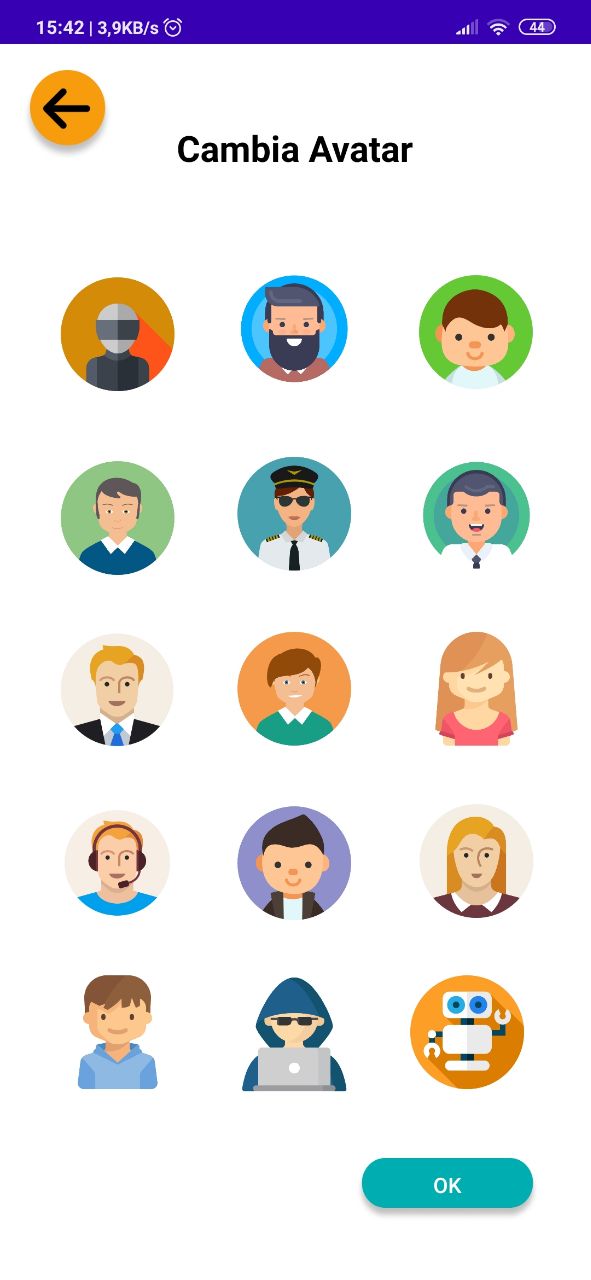
## **Fine di una partita**

Nel momento in cui un utente riesce ad indovinare un numero di parole pari a quello prestabilito dall’admin, il server lo segnala ai partecipanti che possono terminare il gioco, dopodiché manda un segnale ai thread degli utenti presenti in partita (che erano stati bloccati all’inizio). Il segnale viene opportunamente gestito e i thread utente possono nuovamente ricevere le richieste dai relativi client.   
Inoltre, il server accede al database e tramite l’e-mail del giocatore (chiave primaria) assegna una vittoria permanente a quell’utente.

# 

# Screenshoot

Immagine che contiene testo, schermata, logo, Carattere

Descrizione generata automaticamente  Immagine che contiene testo, schermata, Sistema operativo

Descrizione generata automaticamenteImmagine che contiene testo, elettronica, schermata, Carattere

Descrizione generata automaticamente

# README

• Ospitare il server sulla macchina locale:

1. Installare Docker
2. Eseguire da bash "docker compose up" nella directory /Guess30L/server.

Docker avvierà il server su porta 5001.

• Avviare il client:

1. Aprire al path /Guess30L/Android la directory con il progetto tramite Android Studio.
2. Avviare tramite Android studio l’applicazione che si collegerà al server locale avviato in precedenza.