"*Stati di un Robot: RoboDrink-App*"

Corso di Lab. di Sistemi Operativi, Informatica

Prof.ssa Alessandra Rossi, Anno Accademico 2022/2023

DIETI - Università Degli Studi di Napoli Federico II

Erasmo Prosciutto, N86003546
Biagio Scotto di Covella N86003605
Antonio Lanuto, N86003762

September 15, 2023

Contents

1 Progetto

Il progetto "Stati di un Robot" ha l'obiettivo di sviluppare un sistema interattivo in cui un Robot agisce come Server per consentire l'interazione con n utenti, che fungono da Client. L'interazione è gestita attraverso una "state machine", che traccia e controlla l'intera sequenza di eventi dall'inizio alla fine. Il sistema prevede una serie di stati che rappresentano diverse fasi dell'interazione tra il robot e l'utente. L'obiettivo finale del progetto è realizzare un'applicazione mobile, adatta sia a dispositivi cellulari che tablet, che permetta agli utenti di interagire con il robot in base a questo sistema di stati.

Partendo da questi punti cardine, nei prossimi paragrafi spiegheremo nel dettaglio il funzionamento dell'applicativo e le varie scelte implementative.

2 Implementazione

Nell'elaborazione del progetto, diverse decisioni chiave hanno orientato la natura e le funzionalità dell'applicazione. Di seguito, approfondiremo le principali scelte implementative e i motivi alla base delle stesse.

RoboDrink si configura come un'efficace applicazione Client-Server che opera attraverso una rete basata sul protocollo TCP/IP. Questa infrastruttura si avvale dell'uso di socket per assicurare una comunicazione fluida tra Client e Server. Per evitare vincoli legati all'uso di una rete locale, abbiamo scelto una VPS Aruba con sistema operativo Ubuntu 20.04.6 LTS come piattaforma operativa per il Robot. Tale scelta ha reso possibile utilizzare l'applicazione ovunque, garantendo come unici requisiti una connessione internet e un dispositivo mobile Android.

In merito alla gestione dei dati, abbiamo selezionato PostgreSQL come sistema di Database, integrato all'interno del Server Ubuntu. La presenza di un Database è fondamentale per mantenere e gestire le informazioni relative agli utenti dell'applicazione. Ogni interazione del Client, infatti, sollecita una risposta del Server, calibrata sulla base delle informazioni contenute nel Database. Tale sistema non solo assicura risposte accurate, ma anche conferisce stabilità e coerenza all'esperienza dell'utente.

Di seguito descriviamo l'implementazione data ai vari stati dell'applicazione.

- **Out of Sight**: Questa funzionalità monitora l'inattività dell'utente. Se l'interazione è assente per più di un minuto, l'app entra nello stato "Out of Sight", mostrando un pulsante per ritornare alla schermata precedente. Se l'utente non interviene in 10 secondi, l'app entra nello stato "Gone".

- **New**: Qui, l'utente può scegliere tra accedere, registrarsi o utilizzare l'app come ospite. Se la connessione al server va a buon fine, si riceve un ID di sessione; in caso contrario, un messaggio informa l'utente dell'errore.

- **Welcome**: Prima di passare all'ordine, l'app verifica quanti altri utenti sono in attesa. Questa verifica assicura che l'esperienza dell'utente sia modellata in base alla situazione attuale.

- **Waiting**: In questa schermata, l'utente attende che sia il suo turno per ordinare. La durata dell'attesa dipende dal numero di persone in coda, il quale viene aggiornato periodicamente.

- **Ordering**: L'utente può scegliere un drink proposto o selezionare da una lista.

- **Serving**: Qui, l'utente decide se vuole o meno interagire con il robot.

- **ChatBot**: Viene proposto all'utente un quiz basato sui suoi argomenti preferiti scelti durante la registrazione.

- **Sala d'attesa**: Una breve pausa di 20 secondi viene inserita prima che il drink sia pronto.

- **Farewelling**: Una volta che il drink è pronto, l'utente può ritirarlo e viene fornita una descrizione del drink.

- **Gone**: Terminato l'utilizzo di RoboDrink, l'utente può dare un feedback sull'esperienza vissuta.

- **Signup**: Qui, l'utente fornisce dettagli per la registrazione, che saranno poi verificati dal server.

- **Interview**: L'utente viene guidato attraverso una serie di scelte, come la selezione del suo drink preferito e l'argomento di interesse.

3 Server

Il Server utilizza le socket per stabilire e gestire le connessioni. Contemporaneamente, al fine di servire più client contemporaneamente, quest'ultimo fa uso di un modello basato su thread: quando un client si connette, il server crea un nuovo thread per gestire la comunicazione con quel client, garantendo così un servizio non bloccante e scalabile. La comunicazione con il client si basa su un meccanismo di ascolto costante e risposta, tramite le funzionalità di *read* e *write*, con messaggi significativi che corrispondono al successo o fallimento di una operazione, oppure all'ingresso o uscita da uno stato del sistema. Al Server è inoltre dato il compito di interagire con un database, al fine di controllare gli stati degli utenti, verificare le loro credenziali e gestire le loro preferenze su drink e argomenti per l'intrattenimento.

3.1 Funzionalità

Dato il gran quantitativo di funzionalità da gestire, ogni stato è stato mappato con una funzione specifica, che a sua volta utilizza delle funzioni ausiliarie per completare la sua chiamata, rispettando così la modularità e rendendo il codice molto più leggibile. Di seguito diamo una sintesi di come sono state implementate alcune funzionalità precedentemente descritte, fornendo una suddivisione per compiti.

Gestione delle comunicazioni

- *int* ***read_from_socket****(int sockfd, char *buffer, int bufsize)*: lettura dei messaggi.
- *int* ***write_to_socket****(int sockfd, const char *message)*: scrittura dei messaggi.
- *PGconn** ***connect_to_db****()*: gestione della connessione con il database.
- *void** ***client_handler****(void *socket_desc)*: gestisce le richieste dei vari client che si collegano.

Gestione delle sessioni attive degli utenti

- *void* ***addUserToList****(const char* email)*: aggiunge un utente alla lista degli utenti attivi.
- *void* ***removeUserFromList****(int session_id)*: rimuove un utente dalla lista degli utenti attivi.
- *User** ***findUserByEmail****(const char* email)*: restituisce un utente in base alla sua email.

Gestione degli stati

- *void* ***handle_login****(int sockfd, PGconn *conn, char *buffer)*: gestisce il login degli utenti.
- *void* ***handle_signup****(int sockfd, PGconn *conn)*: gestisce la registrazione di un nuovo utente.
- *void* ***handle_welcoming****(int sockfd, PGconn *conn)*: gestisce la fase di benvenuto controllando se l'utente deve passare per la coda.
- *void* ***handle_add_user_ordering****(int sockfd,PGconn *conn)*: gestisce l'inserimento di un utente nello stato di ordering.

Invio di informazioni

- *char** ***suggest_drink****(PGconn *conn, const char *email)*: inoltra al client un drink randomico tra la lista dei drink preferiti dell'utente.
- *char** ***get_drinks_name****(PGconn *conn)*: inoltra al client la lista dei drink disponibili.
- *char** ***get_topics****(PGconn *conn, char *email)*: inoltra al client la lista degli argomenti preferiti dell'utente.
- *void* ***send_drink_description****(PGconn *conn, int sockfd)*: inoltra al client la descrizione del drink selezionato.

4 Client

L'architettura dell'applicazione RoboDrink è stata delineata tenendo conto dei vincoli imposti dalla traccia del progetto, con un focus particolare sulla gestione dei diversi scenari di utilizzo e sui flussi che l'utente può attraversare, in modo tale da consentire a quest'ultimo di navigare attraverso i diversi stati interagendo con elementi di feedback appositamente progettati.

Scenario 1: Funzionamento Completo con Connessione al Server

Nel primo scenario, in cui tutte le componenti sono funzionanti e l'utente può effettuare l'identificazione al Server, l'applicazione segue il seguente flusso:

1. Avvio dell'applicazione.
2. L'utente ha la scelta tra login, registrazione e accesso come ospite.
3. Nel caso di accesso come ospite, la connessione al Server viene esclusa, garantendo un utilizzo offline.
4. Nel caso di login o registrazione, l'utente passa allo stato di "welcome" dopo l'identificazione, dove il Server verifica le informazioni e determina se l'utente può procedere all'ordinazione o deve aspettare il proprio turno.
5. Nel caso di ordinazione, l'utente può scegliere un drink basato sulle sue preferenze comunicate in fase di registrazione o da una lista di drink disponibili.
6. L'utente passa allo stato di "serving", dove può decidere se interagire con il robot tramite la fase di "chatBot" o attendere il completamento dell'ordine nella sala d'attesa.
7. Nel caso di interazione con il "chatBot", vengono proposti quiz basati sulle preferenze dell'utente.
8. Una volta che l'ordine è pronto, l'utente passa allo stato di "farewelling" dove riceve una descrizione del drink e può ritirare l'ordine.
9. L'utente completa il processo passando allo stato di "gone" ed esce dall'applicazione.

Scenario 2: Uso come Ospite senza Connessione al Server

Nel secondo scenario, l'utente sceglie di utilizzare l'applicazione come ospite senza stabilire una connessione al Server. Il flusso di utilizzo rimane simile al primo scenario, ma senza l'interazione con il Server:

1. Avvio dell'applicazione.
2. Scelta tra login, registrazione e accesso come ospite (senza connessione al Server).
3. L'utente passa attraverso le fasi di "welcome", "waiting, "ordering", "serving", "chatBot" (se interagisce) o "sala d'attesa", "farewelling", e infine "gone".

È fondamentale notare che in tutte le fasi dell'applicazione, se l'utente si allontana, verrà messo in stato "out of sight" in attesa che ritorni entro un tempo prefissato. In caso contrario, verrà fatto uscire dall'applicazione.

L'architettura di RoboDrink è stata progettata per gestire entrambi gli scenari in modo fluido, garantendo un'esperienza coerente e coinvolgente per gli utenti, sia in presenza di connessione al Server che in modalità offline come ospite.

4.1 Funzionalità

Le funzionalità implementate sono esattamente le medesime, le differenze sono dovute solo al differente modo di operare.

5 Comunicazione Client-Server

Nella seguente sezione, verrà esposto nel dettaglio lo schema di scambio di messaggi del sistema. Il Server ha la capacità di ricevere diversi messaggi provenienti dal Client e, per ognuno di essi, è in grado di dipartire le varie operazioni necessarie per far progredire l'utente all'interno della struttura a stati finiti.

5.1 Messaggi Client-Server

- `LOG_IN`: Attraverso questa funzione, viene gestita l'operazione di accesso dell'utente. Il Client trasmette le credenziali dell'utente al server, ovvero l'indirizzo email e la password. Una volta verificata l'autenticità dell'utente tramite il Database, il Server risponde al Client comunicandone l'esito positivo o gli eventuali insuccessi nell'effettuare l'accesso.
- `SIGN_UP`: Si gestisce il procedimento di registrazione dell'utente mediante uno scambio di informazioni fondamentali tra il Client e il Server. Questi dati consentono l'inserimento dell'utente nel sistema. Anche al termine di questa fase, il Server risponderà al Client segnalando un eventuale successo o l'individuazione di possibili problematiche nel processo.
- `START_CHAT`: Si coordinano le operazioni necessarie al caricamento degli argomenti con cui l'utente potrà intrattenere un dialogo.
- `CHECK_USERS_ORDERING`: Mediante questa richiesta, il Client interpella il Server per ottenere il numero attuale di utenti in fase di *Ordering* e *Serving* e sapere in che stato andare.
- `ADD_USER_ORDERING`: Con questa funzione, l'utente comunica al Server e al database l'aggiunta dell'utente alla lista degli utenti in fase di *Ordering*.
- `ADD_USER_WAITING`: Attraverso questa operazione, l'utente notifica al Server e al database l'aggiunta dell'utente agli utenti in attesa (*Waiting*).
- `UPDATE_QUEUE`: Mediante questa richiesta, il Client interpella il Server per ottenere il numero attuale di utenti in fase di *Ordering* e *Serving* e sapere se può andare in *Ordering*.
- `DRINK_DESCRIPTION`: Il Server restituisce al Client la descrizione del drink selezionato dall'utente.
- `SUGG_DRINK`: Il sistema seleziona un drink dal database, in base a quelli scelti come preferiti dall'utente, e lo propone.
- `USER_STOP_ORDERING`: Con questa azione, il Client comunica al server che l'utente non è più nella fase di *Ordering*.
- `USER_STOP_SERVING`: Con questa azione, il Client comunica al server che l'utente non è più nella fase di *Serving*.
- `DRINK_LIST`: Il Server fornisce all'utente l'elenco completo dei drink disponibili.
- `USER_GONE`: Questa funzione gestisce il caso in cui l'utente lasci l'applicazione, modificando il suo stato nel database.

5.2 Messaggi Server-Client

- `LOG_IN_SUCCESS`: Conferma dell'accesso riuscito.
- `LOG_IN_ERROR`: Notifica di errore nell'accesso.
- `SIGNUP_SUCCESS`: Conferma della registrazione avvenuta con successo.
- `SIGNUP_ERROR`: Notifica di errore nella registrazione.
- `DRINK_DESCRIPTION_NOT_FOUND`: Notifica che la descrizione della bevanda non è stata trovata.

6 Istruzioni per l'uso

6.1 Compilazione del Server in C

Per compilare il Server in C, è necessario includere la libreria di Postgresql. Assicurarsi che la libreria di Postgresql sia installata nel sistema e specificare il flag di compilazione corretto durante la compilazione del file C. Di seguito è riportato un esempio di compilazione utilizzando GCC:

```
gcc -g -o server main.c user_management.c auxiliary.c connections_sockets.c manager.c

states_management.c -w -I/usr/include/postgresql -pthread -lpq
```

Nel caso si stesse utilizzando il Server proprietario dove è stata sviluppata l'applicazione si puo' anche usare lo script bash custom chiamato `compila.sh`.

7 Logs ed esempi

Nello sviluppo software, i log rappresentano una risorsa preziosa per la diagnostica dei problemi, il monitoraggio delle performance, la tracciabilità, la documentazione e lo storico delle operazioni svolte, ecc. Proprio per questo motivo, oltre ai log già presenti lato client tramite LogCat di Android Studio, abbiamo implementato un sistema di logs basilare lato server, al fine di tracciare il comportamento di quest'ultimo ed evidenziare eventuali problematiche o errori.

A titolo di esempio si mostra parte del contenuto dei Log lato server di una comunicazione con un nuovo client ed una serie di screenshots dell'applicativo negli stati più rilevanti:

```
###          FILELOG RESULT SERVER          ###
root@Robot-Server:~# ./new_server
Server in ascolto sulla porta 8080
[Connessione] per il client 4 avvenuta con successo
[Read] SIGN_UP
[SignUp] Il client ha avviato la fase di registrazione...
[Read] giusepe@gmail.com
[Read] Giuseppe
[Read] Romano
[Read] password
[Read] 4
[Read] Mojito
[Read] Martini
[Read] Pina Colada
[Read] Gin Lemon
[Read] 4
[Read] Geografia
[Read] Sport
[Read] Attualità
[Read] Scienza
[Send] SIGN_UP_SUCCESS
[SignUp] Registrazione per l'utente Giuseppe Romano
         (email: giusepe@gmail.com | password: password) andata a buon fine
[Read] LOG_IN giusepe@gmail.com password
[Login] L'utente 'giusepe@gmail.com' con password 'password' ha effettuato l'accesso
[Send] LOG_IN_SUCCESS 1 Giuseppe
[Read] CHECK_USERS_ORDERING
[Send] 0
[Welcoming] Nessun utente in attesa, procedere con la fase di ordering.
[Read] DRINK_LIST
[Drinks-List] Inoltro lista dei drink: Mojito,Martini,Midori,Manhattan,Negroni,
              Daiquiri,Pina Colada,Gin Lemon
[Send] Mojito,Martini,Midori,Manhattan,Negroni,Daiquiri,Pina Colada,Gin Lemon
[Read] SUGG_DRINK
[Read] 1
[Sugg-Drink] Drink preferiti dell'utente giusepe@gmail.com: Mojito,Martini,Pina Colada,Gin Lemon
[Sugg-Drink] Drink preferito randomico dell'utente giusepe@gmail.com: Martini
[Send] Martini
[Read] ADD_USER_ORDERING
[Read] 1
[Add-Ordering] L'utente giusepe@gmail.com e' ora nello stato di ordering
[Read] USER_STOP_ORDERING
[Read] 1
[Add-Serving] L'utente giusepe@gmail.com e' ora nello stato di serving
[Read] START_CHAT
[Read] 1
[Chat] I topics preferiti dell'utente giusepe@gmail.com sono: Geografia,Sport,Attualità,Scienza
[Send] Geografia,Sport,Attualità,Scienza
[Read] USER_STOP_SERVING
[Read] 1
[Stop-Serving] L'utente giusepe@gmail.com ha concluso la fase di serving ed e' ora in idle
[Read] DRINK_DESCRIPTION
[Read] Manhattan
```

```
[Drink-Description] Richiesta descrizione per il drink: Manhattan
[Send] Cocktail a base di whisky, servito come aperitivo
[Drink-Description] Descrizione inoltrata con successo
[Read] USER_GONE
[Read] 1
[Gone] L'utente giusepe@gmail.com è uscito dall'applicazione

###          FILELOG RESULT CLIENT          ###
Connessione al server...
Connesso al server.
Sending message: SIGN_UP
Sending message: giusepe@gmail.com
Sending message: Giuseppe
Sending message: Romano
Sending message: password
Sending message: 4
Sending message: Mojito
Sending message: Martini
Sending message: Pina Colada
Sending message: Gin Lemon
Sending message: 4
Sending message: Geografia
Sending message: Sport
Sending message: Attualità
Sending message: Scienza
Received message: SIGN_UP_SUCCESS
Sending message: LOG_IN giusepe@gmail.com password
Received message: LOG_IN_SUCCESS 1 Giuseppe
Sending message: CHECK_USERS_ORDERING
Received message: 0
Sending message: DRINK_LIST
Received message: Mojito,Martini,Midori,Manhattan,Negroni,Daiquiri,Pina Colada,Gin Lemon
Sending message: SUGG_DRINK
Sending message: 1
Received message: Martini
Sending message: ADD_USER_ORDERING
Sending message: 1
Sending message: USER_STOP_ORDERING
Sending message: 1
Sending message: START_CHAT
Sending message: 1
Received message: Geografia,Sport,Attualità,Scienza
Sending message: USER_STOP_SERVING
Sending message: 1
Sending message: DRINK_DESCRIPTION
Sending message: Manhattan
Received message: Cocktail a base di whisky, servito come aperitivo
Sending message: USER_GONE
Sending message: 1
```

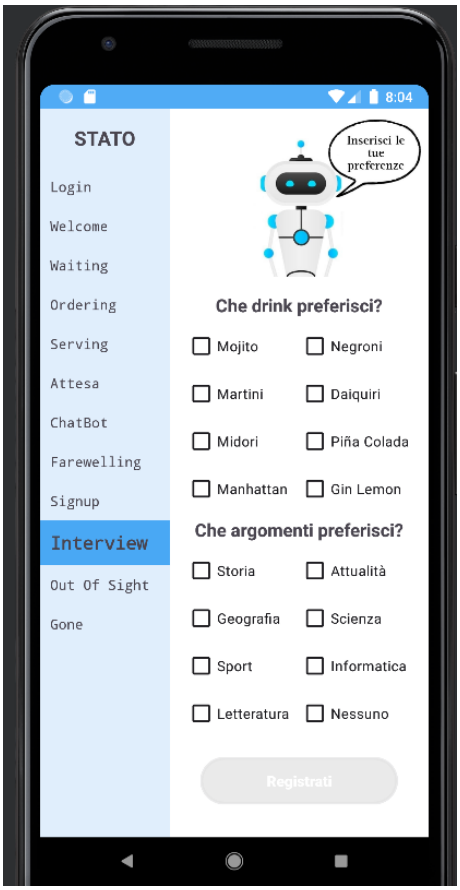

Figure 1: Interview

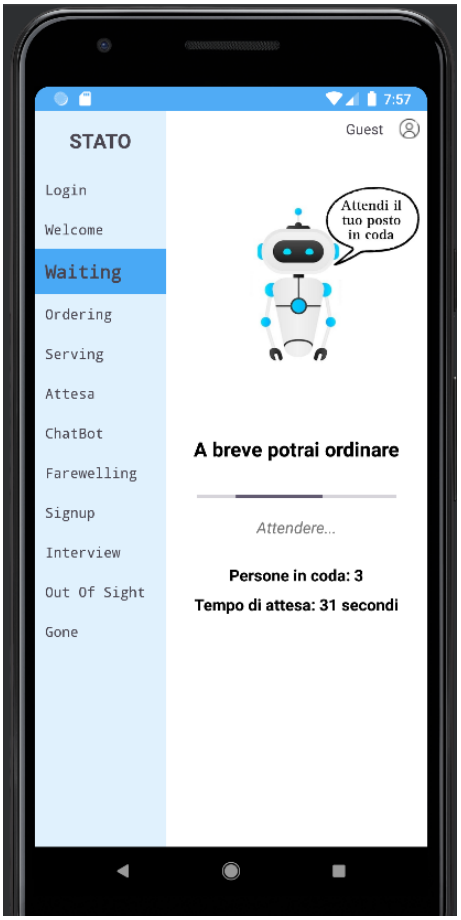

Figure 2: Waiting

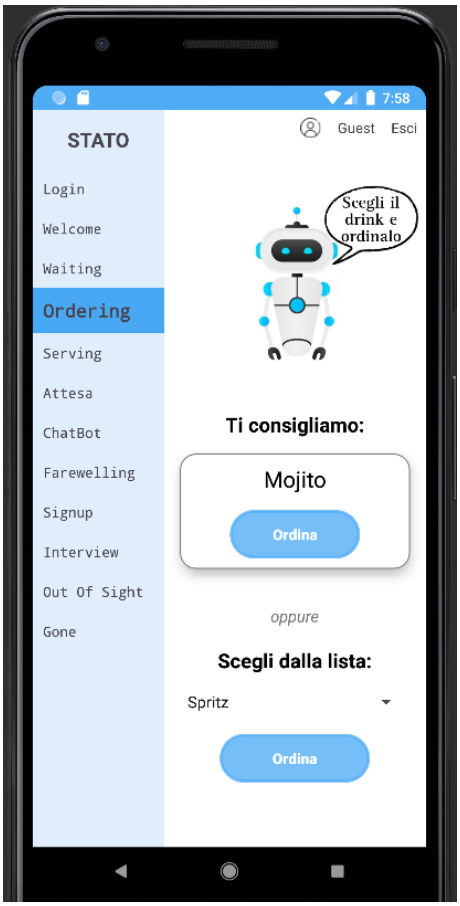

Figure 3: Ordering

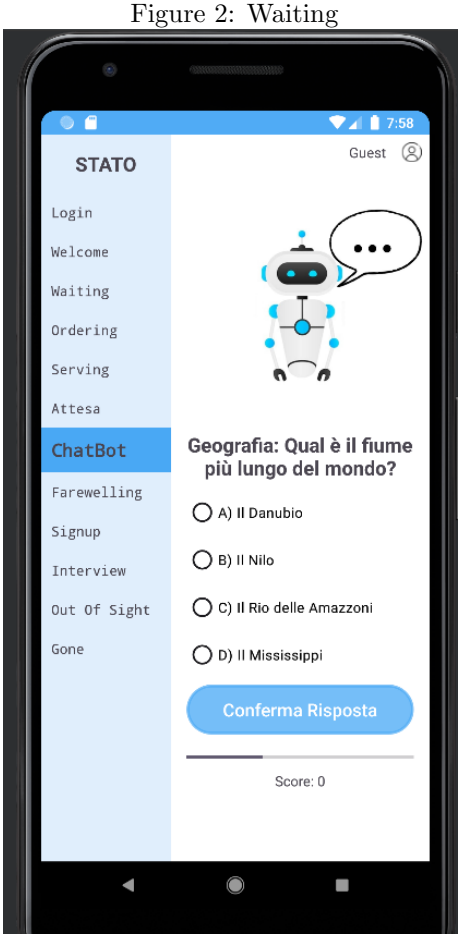

Figure 4: ChatBot

8 Testing

Durante l'iter di sviluppo del progetto, è stata condotta una serie di approfonditi test al fine di garantire il corretto funzionamento di tutte le funzionalità implementate all'interno del sistema. Nei paragrafi seguenti, verranno presentati in dettaglio i tipi di test eseguiti e i relativi risultati:

- **Test di Autenticazione e Accesso** Sono stati eseguiti una serie di test al fine di assicurare la corretta autenticazione degli utenti. Attraverso una serie di scenari di prova diversificati, abbiamo verificato che ogni utente sia in grado di registrarsi e accedere all'applicazione in modo affidabile. Inoltre, abbiamo condotto test sulla funzionalità di accesso anonimo, nota come *Modalità Guest*, per garantire che gli utenti possano fruire dell'applicazione anche senza autenticazione.

- **Test di Gestione della Coda** Sono stati eseguiti test approfonditi per valutare la gestione della coda degli utenti nell'applicazione. Abbiamo simulato l'accesso simultaneo di utenti diversi per verificare l'efficacia del sistema nel comunicare agli utenti se devono attendere nella fase "Waiting" o avanzare alla fase "Ordering" solo quando il numero corretto di partecipanti è presente.

- **Test di Ordinazione dei Drink** Sono stati effettuati test dedicati alla corretta ordinazione dei drink all'interno del sistema. Questi test hanno valutato sia la capacità del sistema di suggerire i drink appropriati in base alle preferenze degli utenti, sia la corretta gestione della disponibilità dei drink nel database.

- **Test di Quiz e Sala d'Attesa** In questa fase, una serie di test è stata dedicata alle funzionalità di sala d'attesa e ai quiz proposti agli utenti. Si è verificato che gli utenti possano ritirare i drink alla fine del tempo di preparazione nella sala d'attesa. Per quanto riguarda i quiz, è stata valutata l'accuratezza nella visualizzazione delle risposte corrette, evidenziando in verde le risposte esatte e in rosso le risposte errate fornite dagli utenti. Il punteggio ottenuto nel quiz è stato attribuito in modo appropriato e il flusso di utilizzo è proseguito con successo alla fase di ritiro del drink.

- **Test di Fase di "Serving" e "Gone"** I test sono stati condotti per verificare l'adeguatezza delle fasi di "Serving" e "Gone", garantendo che gli utenti possano disconnettersi dall'applicazione e ritirare il proprio drink in modo coerente e affidabile.

- **Test di "Out of Sight"** Attraverso una serie di test specifici, è stato valutato il comportamento dell'applicazione in seguito a un periodo di timeout noto come "Out of Sight". Si è verificato che, se l'utente interagisce entro il limite di tempo prestabilito, l'applicazione riprenda esattamente dal punto in cui era stata interrotta. In caso contrario, se l'utente non interagisce entro l'intervallo di tempo specificato, l'applicazione lo indirizza correttamente alla fase "Gone".

- **Test di registrazione e intervista** Sono stati eseguiti una serie di test al fine di assicurare la corretta registrazione degli utenti nel sistema. Attraverso una serie di scenari di prova diversificati, abbiamo verificato che ogni utente sia in grado di registrarsi rispettando tutti i vincoli imposti dal sistema (lunghezza password, email, ecc..).

I risultati dei test sono stati positivi, evidenziando una risposta accurata del sistema alle interazioni degli utenti e una gestione coerente degli stati in base alle regole predefinite.

9 Considerazioni finali

Lo sviluppo del progetto ci ha permesso di approfondire la conoscenza dell'ambiente di sviluppo di Android. Inoltre, è stato interessante capire come possano funzionare le normali applicazioni che usiamo tutti i giorni e come vengono sviluppate. Durante lo sviluppo, le principali problematiche incontrate sono state legate alla comunicazione tra client e server con le socket e alla gestione degli eventuali errori e ritardi di invio e ricezione dei messaggi. Per risolvere tali problemi, sono state fatte molteplici prove e sono state utilizzate diverse strategie.