UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA IN INFORMATICA
LABORATORIO DI SISTEMI OPERATIVI
ANNO ACCADEMICO 2022/2023

APPLICAZIONE PER LA VENDITA E DISTRIBUZIONE DI BEVANDE

Lorenzo Sepe N86003622
Raimondo Morisini N86003839
Giulio Boriello N86003381
25/05/2023

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

Capitolo 1

Requisiti del sistema

Il cliente ha richiesto una applicazione per dispositivi Android per gestire le ordinazioni di drink e alcolici di un locale.
Il sistema deve poter essere utilizzato da multipli utenti simultaneamente senza perdita di prestazioni, permettendo a tali utenti di eseguire una registrazione per tener conto degli ordini eseguiti ed eseguire pagamenti (simulati) attraverso l'applicazione. L'utente può accedere anche tramite i dati biometrici che il dispositivo può raccogliere.
L'applicazione deve essere capace di filtrare il menú attraverso gli attributi dei drink e, sfruttanto le informazioni raccolte, poter dare suggerimenti su cosa ordinare.
L'applicazione deve usare una base di dati per memorizzare le informazioni necessarie.

Capitolo 2

Progettazione e struttura

L'applicativo é strutturato in due macro blocchi, il Client per dispositivi android scritto in linguaggio Java e il Server scritto per macchine Linux in linguaggio C.
Le due componenti comunicano tramite socket TCP usando l'indirizzo IP del Server, hostato in una macchina fisica di nostra proprietá.

2.1 Backend

2.1.1 Server

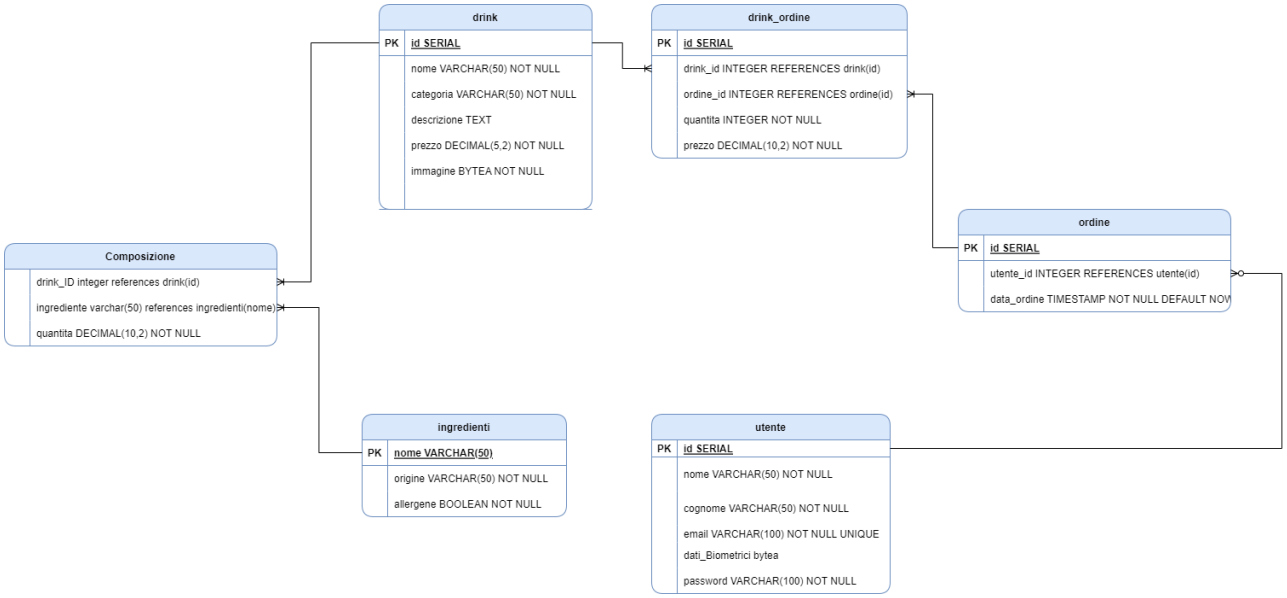
Per poter soddisfare il requisito della multiplicitá simultanea di utenti e devicees, il server é stato costruito per essere multithreaded, creando all'avvio una quantitá prefissata di thread che vengono inizializzati con la funzione `void* thread_function(void* args)`. Questa metodo ci permette di non avere uno spreco immotivato di risorse e le funzioni `pthread_cond_signal(condition-variable)` e `pthread_cond_wait(condition-variable, mutex)` sono responsabili di mettere in pausa e riavviare i thread quando é necessario.
Il motivo che ci ha spinto ad usare un numero di thread predefinito (20) é che il servizio di hosting utilizzato (ElephantSQL) permette l'accesso a un numero limitato di utenti contemporaneamente.
Data la limitazione degli accessi, abbiamo creato una coda in cui aggiungiamo gli utenti che non sono stati serviti, contenente gli indirizzi IP degli utenti che vorrebbero fare una richiesta, quindi é divenuto necessario un mutex che impedisce a due thread di accedere alla zona critica sottostante contemporaneamente.

```
1  /// @brief funzione che accoda le connessioni per poi essere processate
2  /// @param args argomenti della funzione
3  void* thread_function(void* args){
4      while (true)
5      {
6          int *pclient;
7          pthread_mutex_lock(&mutex);
8          if( (pclient = decoda()) == NULL){
9              pthread_cond_wait(&condition_var, &mutex);
10             pclient = decoda();
11         }
12         pthread_mutex_unlock(&mutex);
13
14         if(pclient != NULL){
15             handle_connection(pclient);
16         }
17     }
18 }
```

I messaggi scambiati tra applicativo e server sono stati conformati con un semplice protocollo di comunicazione di entrata e uscita: ogni messaggio che arriva al server é una stringa divisa in due parti, il codice della transazione e la query SQL da eseguire.
I codici sono univocamente corrispondenti all'operazione da effettuare, dando a server un modo per smistare questi messaggi al database.
Il Server poi restituisce le informazioni mandando il risultato come stringhe che il client mette nei relativi oggetti.
Per evitare che il server si comporti in modo dannoso per colpa di errori imprevisti, é stata scritta una funzione che viene invocata quando vengono usate funzioni che potrebbero far interrompere il funzionamento.

2.1.2 Database

Per memorizzare le informazioni, é stato utilizzato un database SQL scritto usando PostgresSQL.
La struttura di tale database é descritta dal seguente diagramma UML:



Questo ci permette di confrontare i dati ricevuti e fare query complesse, cosí soddisfando un'altra richiesta della traccia: usando una funzione interna al database, possiamo organizzare i drink attraverso dei parametri prestabiliti e ottenere due liste generate dinamicamente con i dati precedentemente inseriti.
Vi sono due metodi per suggerire un drink ad un utente:

- Il database controlla quali drink il bar ha venduto con piú frequenza e suggerisce un prodotto che l'attuale utente non ha mai comprato. Questo viene chiamato metodo delle Tendenze.
- Il database controlla quali prodotti l'utente ha ordinato e consiglia un drink che condivide ingredienti con elementi precedentemente ordinati da esso. Questo viene chiamato metodo delle Preferenze.

2.2 Frontend

2.2.1 ModelView-Controller

Si é utilizzato un paradigma ModelView-Controller per separare la parte grafica dalla comunicazione con il database e la logica.
Ogni azione che l'utente effettua sull'interfaccia (ad esempio la registrazione o il pagamento), viene spedita al server mediante una funzione presente in questo sistema basato sulla classe Controller.
La gestione e le specifiche della connessione con il server śono fornite dalla classe Connessione.
Per evitare conflitti o problemi a runtime, il pattern Singleton viene applicato sia a Controller che a Connessione
L'applicazione scritta nella sua interezza per in sistema operativo Android, utilizza un istanza della classe Controller per mandare le richieste, mentre le classi posizionate sotto il package Model servono principalmente per memorizzare i dati ottenuti dal Server.

2.2.2 Thread

Tutte le funzionalitá che si interfacciano con il server sono gestite tramite i thread sincroni e asincroni a seconda della necessitá o meno di far aspettare l'utente.
La maggior parte delle operazioni avvengono in maniera sincrona, ad eccezione del caricamento delle immagini che avviene in modalitá asincrona.
La sincronia é implementata tramire la funzione join(), mentre l'asincronia é una caratteristica di default dei thread.
Il motivo per cui abbiamo bisogno di generare thread diversi da quello principale anche per le operazioni sincrone é dovuto da android stesso, che lancia un eccezione in caso si cerchi di effettuare delle operazioni di rete dal MainThread.

2.2.3 Funzionalitá implementate

Login

L'utente all'avvio dell'applicazione si troverá di fronte alla schermata di login, implementata in MainActivity dove sará possibile accedere sia tramite email e password sia tramite dati biometrici.
La richiesta di login sará demandata alla classe controller.
L'applicazione ricorda le credenziali dell'ultimo utente che ha fatto l'accesso tramite la classe SharedPreferences .
L'autenticazione dei dati invece é stata implementata tramite la classe BiometricManager che definisce il comportamento in caso di successo o insuccesso dell'autenticazione.
In caso di insuccesso, in questo come in molti altri casi, comparirá un Toast all'utente che lo avvertirá dell'errore.

Registrazione

Vi é un activity dedicata solamente alla registrazione che é accessibile dalla MainActivity mediante un apposito pulsante. L'utente per accedere ha bisogno di inserire nome utente, nome, cognome, e password e, in caso di successo l'utente viene portato alla pagina di selezione dei drink.
L'utente non é in grado di inserire 2 username identici.

RecyclerView

RecyclerView é una delle componenti piú importanti dell'applicazione: é la view che modifica la visualizzazione degli elementi mostrati e quindi fa in modo di poter implementare le logiche di filtraggio, permettendo al contempo all'utente di scorrere con il dito la lista dei drink.
La ragione di voler utilizzare una RecyclerView piuttosto che una ListView é che quest'ultima permette di "riciclare" le risorse occupate da gli elementi finiti fuori dallo schermo, per visualizzare quelli che invece sono entrati nella visuale dell'utente, rendendo piú efficiente il sistema.
Gli elementi fondamentali della RecyclerView sono l'holder e l'adapter: il primo é un model per i componenti mostrati sullo schermo, mentre il secondo invece defisce il comportamento della RecyclerView e dei suoi elementi.

Quando viene creato, l'adapter richiama il metodo `getDrinks()` di controller per ottenere una List<Drink>, grazie alla quale all'interno del metodo `onBindViewHolder(@NonNull DrinksHolder holder, int position)` riempirá correttamente gli attributi dell'holder.
In questo metodo viene anche definito come si devono comportare le varie View quando l'utente interagisce con esse.
Le immagini usate per i vari drink sono state create usando Bing Create

Filtri

Si puó filtrare i drink da scegliere o tramite opportune categorie o tramite la barra di ricerca.
Per gestire la barra di ricerca si utilizza una SearchView, il componente che permette la ricerca tramite il suo metodo `setOnQueryTextListener`.
Il filtraggio tramite categorie é invece implentato da uno Spinner, una view che consente di mostrare con un menú a tendina le categorie ottenute tramite una funzione del Controller.
Mentre questi ultimi elementi consentono all'utente di specificare il proprio metodo di filtraggio, il vero e proprio lavoro di filtraggio é fatto da la SuperClasse Filterable.
Essa infatti consente di specificare dei metodi grazie ai quali ottenere un oggetto di tipo Filter che poi verrá passato all'adapter che lo userá per il filtraggio degli elementi.
L'oggetto di tipo Filter avrá nei suoi metodi la logica per implementare il filtraggio.

Carrello

Il carrello é implementato tramite l'activity `CarrelloActivity`, accedibile tramite `DrinkActivity` che la mostra all'utente chiamandola tramite il metodo `startActivityForResult(Intent intent, int requestCode)`.
Consente all'utente di aumentare o diminuire la quantitá dei drink precedentemente selezionati e eventualmente di eliminare il drink.
Inoltre tramite il carrello é possibile visualizzare il prezzo totale o il prezzo di un singolo drink moltiplicato per la sua quantitá.
Un elemento viene eliminato dal carrello quando la sua quantitá viene impostata a zero o quando viene effettuato lo swipe verso destra o sinistra.
Il controller gestisce l'eliminazione all'interno del database, mentre al livello grafico, essa é gestita dall'adapter mediante il metodo removeItem(int position), che gestisce anche evenuali problemi creati dal cambiamento di posizione degli oggetti.
L'eliminazione tramite la quantitá é gestita banalmente da un listner sul pulsante "meno" mentre l'eliminazione tramite swipe ha bisogno di due classi per essere gestita ovvero: SwipeToDeleteCallback (presente sotto il package classiDiSupporto e che estende la classe ItemTouchHelper.Callback) e ItemTouchHelper.
La classe SwiperToDeleteCallBack gestisce le animazioni e la logica di eliminazione, mentre ItemTouchHelper é una classe utility per le interazioni dell'utente con la RecyclerView che ha bisogno di un oggetto di tipo ItemTouchHelper.Callback per funzionare nel modo desiderato. Quando si modifica la quantità dei prodotti e si torna indietro cin il pulsante back a `DrinkActivity` tramite il metodo `OnBackPressed()` si imposta il risultato a `RESULT_OK` cosí facendo `DrinkActivity` notificherá l'adapter della sua `RecyclerView` che è avvenuto un cambiamento nei dati, tramite il metodo `onActivityResult(int requestCode, int resultCode, @Nullable Intent data)` che definisce il comportamento dell'activity quando si ritorna da un activity lanciata con `startActivityForResult(Intent intent, int requestCode)`.

Pagamento

Il pagamento é simulato tramite dati di carta di credito. La classe CrediCardTextWatcher sotto il package classiDiSupporto estende la classe TextWatcher.
Essa viene passata al campo numero carta tramite il metodo `addTextChangedListener(TextWatcher watcher)` e consente di aggiungere uno spazio ogni 4 cifre della carta di credito in modo da far avere all'utente una migliore esperienza di interazione. Mentre il controllo che gli altri campi siano inseriti in un formato corretto é implementato tramite un insieme di attributi delle view definiti nei file XML e controlli effettuati nel metodo `effettuaPagamento(String nome, String cognome, String numerocarta, String dataScadenza, String cvv)`, presente nella classe Controller.
Qualunque sia l'esito dell'operazione di pagamento l'attributo statico di Controller pagamentoEffettuatoConSuccesso verrá impostato a true a false.
A questo punto o in caso di errore verrá mostrato un Toast all'utente con con un messaggio di errore, oppure il carrello verrá svuotato, verrá mostrata DrinkActivity e verrá visualizzato un Toast per indicare che il pagamento é andato a buon fine.

2.3 Funzionalità aggiuntive

2.3.1 Modalitá ad alto contrasto

Per aggiungere un grado di accessibilità all'applicazione, anche se non era richiesta dalla traccia, abbiamo introdotto una modalità ad alto contrasto.
Essa é implementata tramite `ImpostazioniActivity`, accessibile tramite un pulsante delle impostazioni tramite qualunque altra activity e avviata tramite il metodo `startActivityForResult(Intent intent, int requestCode)`.
in `ImpostazioniActivity` é presente uno switch che serve a disabilitare o attivare la modalità ad alto contrasto, impostando un valore condiviso in tutte le classi e presente in memoria anche quando l'applicazione é chiusa tramite la classe SharedPreferences.
Le activity tramite il metodo `onResume()` controllano se la modalitá é attivata o meno e modificano i colori dei propri elementi.
Tramite lo stesso principio che consente a `DrinkActivity` di aggiornare i propri valori al ritorno da `CarrelloActivity` le activity che contengono una RecyclerView notificano l'adapter di cambiare i colori degli elementi.

2.3.2 loop.sh

Questo è uno script che avvia il server in caso esso abbia smesso di funzionare per qualche circostanza inaspettata.

2.4 Testing

Il testing dell'applicazione é stato eseguito tramite lo strumento Espresso.
Tale strumento consente di creare delle batterie di test in modo semplice e veloce.
Le batterie sono state divise in delle cartelle che rappresentano ognuna una funzionalitá del sistema.
Uno strano bug che non siamo putroppo riusciti a risolvere riguarda l' attivazione contemporanea di tutte le batterie, l' ultimo test ad essere eseguito causa sempre un errore, anche se singolarmente la batteria funziona correttamente. Avendo provato a eseguire manualmente in successione in passaggi che i test fanno non ci é presentato nessun errore particolare.
Questo ci fa pensare che il bug é dovuto al tool di generazione dei test.

Capitolo 3

README

Questa sezione è necessaria per garantire un corretto set-up del sistema e delle sue funzionalità.
Nella cartella inviata al docente vi sono tutti i codici sorgenti, permettendo così di testare le componenti in modo indipendente l'uno dall'altro.
Verranno incluse le versioni dell'applicazione usanti il server in localhost per una configurazione e avvio semplificati.

3.1 Database

Il databse PostegreSQL è stato hostato usando un servizio gratuito di ElephantSQL per uso privato. Il Server in questo modo può gestire fino ad un massimo di 5 connessioni concorrenti, quindi è stata allegato il file per creare la base di dati in privato: i file createDbms.sql e dump.sql
Per evitare problemi legati alla configurazione, bisgna per prima cosa creare il database con con il nome "catalogo_drink", quindi usare gli statements definiti dal file createDbms.sql per costruire le varie tabelle, view e funzioni.
Una volta finito il file dump.sql contiene tutti i dati dei vari drink inseriti, necessari per far funzionare il sistema nella sua interezza.

3.2 Server

Il server deve essere attivo prima che l'applicativo Android, quindi per garantire che il server sia sempre in esecuzione, è stato scritto un semplice script bash chiamato `loop.sh` per eseguire il server in un ciclo infinito:

```
1 #!/bin/bash
2
3 while true
4 do
5     ./Server &
6     sleep 1
7 done
```

Per avviare il server, quindi è necessario eseguire questo script dopo aver dato permessi di esecuzione al file con:

```
1 chmod 777 loop.sh
```

3.3 Client

L'applicativo va eseguito dopo che il server è stato messo in funzione si può accedere senza effettuare una registrazione tramite l'utente amministratore che ha username= admin password= admin, oppure si può registrare un nuovo utente e utilizzare quello per accedere successivamente.
Per procedere al carrello bisogna inserire almeno un prodotto, dopo il pagamento il carrello viene svuotato e si viene riportati alla pagina di scelta dei drink.