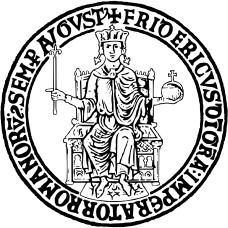
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL’INFORMAZIONE



CORSO DI LAUREA IN INFORMATICA

INSEGNAMENTO DI LABORATORIO DI SISTEMI OPERATIVI - ANNO ACCADEMICO 2022/2023

**Sistema Informativo per la Gestione di un DrinkShop**

|  |  |
| --- | --- |
| Autori | Docente |
|  |  |
| Alessandro Bonomo  N86003852 | Prof.ssa Alessandra Rossi |
| [al.bonomo@studenti.unina.it](mailto:al.bonomo@studenti.unina.it) |  |
| Mario De Luca |  |
| N86003911  [mario.deluca10@studenti.unina.it](mailto:mario.deluca10@studenti.unina.it)  Nome gruppo: JuicyLemons |  |
|  |  |

Il sistema adotta un’architettura di tipo 3-Tier (database - server API -client). Inoltre, stato scelto di utilizzare docker per facilitare il deployment del server in cloud. Il server è articolato in 2 containers:

* PostgresDB: database relazionale
* Http Socket Server in C: API di interfacciamento al database.

Il client è un’applicazione android scritta in Java, ha il ruolo di front-end e comunica direttamente con le API del server in cloud.

**Server**

Il server, come già detto in precedenza, è articolato in due container docker che comunicano tra loro. Il database gira nel container *postgres-db* e risponde su porta 42069, mentre il server in C gira nel container *server-c* e risponde su porta 80.

Il server in C accetta nuove connessioni TCP/IP per i client che si connettono alla porta 80. Per ogni client viene generato un thread in modo da servire più client contemporaneamente. Vengono gestiti i casi in cui viene raggiunto il max pool di connessioni, socket timeout e interruzione improvvisa del server. Il server è a tutti gli effetti una http socket server in quanto la comunicazione avviene secondo il protocollo http ed è possibile collegarsi ad esso mediante un classico browser oppure un software come postman che permette di personalizzare payload e headers della http request/response.

Inoltre, Il server presenta numerose caratteristiche:

* Elabora richieste Http GET, POST, PUT, DELETE
* Parsing di richieste Http con content application/json
* Risposte Http con content application/json
* Smart path routing con URL parameter (esempio: /user, /user/:id)
* Può includere files nella risposta Http
* Jwt bearer token authentication
* Jwt token scade dopo un periodo di tempo
* Password encryption
* Models per interfacciamento con tabelle del database
* Possibilità di eseguire custom middleware per ogni route
* Handler per ogni route
* Custom routes prefix
* Multithreading
* Pool di connessioni con custom max clients
* Custom socket timeout
* Gestione interrupt sigterm di terminazione (esempio CTRL+C)
* Messaggi di stato del server sulle connessioni attive e eventi

Le librerie utilizzate sono le seguenti:

* **libpq-dev**: libreria per interrogare la socket postgres
* **libjson-c-dev**: libreria per encoding/decoding formato json
* **libjwt-dev**: libreria per usare il json web token
* **libcrypt-dev**: libreria per criptare password utente

Il server predispone una serie di routes a cui il frontend fa richiesta. Il server risponde al client filtrando i dati per gli utenti autorizzati da un jwt token. In questo modo, il container del database non è esposto e comunica solo mediante il server.

Routes del server:

{ "POST", "/login", NO\_MIDDLEWARE, loginHandler },

{ "POST", "/register", NO\_MIDDLEWARE, registerHandler },

{ "GET", "/", requiresAuth, homeHandler },

{ "GET", "/user/:email", NO\_MIDDLEWARE, getUserHandler },

{ "GET", "/drink/image/:id", NO\_MIDDLEWARE, getDrinkImageHandler },

{ "GET", "/drink/:id", NO\_MIDDLEWARE, getDrinkHandler },

{ "GET", "/drinks", NO\_MIDDLEWARE, getDrinksHandler },

{ "POST", "/order/drink", requiresAuth, orderDrinkHandler },

{ "PUT", "/order/drink", requiresAuth, updateDrinkQuantityInOrderHandler },

{ "DELETE", "/order", requiresAuth, deleteOrderHandler },

{ "DELETE", "/order/drink/:id", requiresAuth, deleteDrinkFromOrderHandler},

{ "GET", "/order/last", requiresAuth, getLastOrderMadeByUserHandler },

{ "GET", "/orders", requiresAuth, getOrdersMadeByUserHandler },

{ "GET", "/order/:id", requiresAuth, getOrderHandler },

{ "POST", "/pay", requiresAuth, payOrderHandler }

**Client**

Per quanto riguarda la comunicazione con il server, abbiamo individuato tre tipologie di query differenti per ognuna della quale abbiamo creato un’apposita classe NetworkTask (sottoclasse di AsyncTask).

Abbiamo utilizzato:

- NetworkTaskLogin per effettuare tutte le query che hanno un solo valore di ritorno (per esempio per verificare le credenziali di accesso di un utente);

- NetworkTaskRegister per effettuare tutte le query che non hanno valori di ritorno (per esempio per effetuare l’inserimento o la modifica all’interno del database);

- NetworkTaskBevande per effettuare tutte le query il cui valore di ritorno è una lista di bevande (per esempio per mostrare la lista di bevande consigliate).

Abbiamo utilizzato un adapter personalizzato per popolare la ListView con le bevande presenti nel database. L'utilizzo di un adapter personalizzato ha permesso di personalizzare l'aspetto delle singole righe della ListView e di gestire l'aggiunta delle bevande al carrello.

**Test effettuati e risultati**

Durante lo sviluppo del sistema sono stati effettuati diversi test per verificare la sua funzionalità. In particolare, sono stati effettuati i seguenti test:

Test di connessione tra client e server

Test di invio di query SQL dal client al server

Test di ricezione di risultati delle query SQL da parte del client

Tutti i test sono stati superati con successo, dimostrando la funzionalità del sistema. In particolare, è stata verificata la corretta comunicazione tra client e server e la corretta elaborazione delle query SQL da parte del database PostgreSQL. Inoltre, è stato verificato che i risultati delle query SQL fossero correttamente ricevuti dal client e visualizzati all'utente.

**Caso d’uso utente già registrato**

La prima schermata visualizzata è quella di login.

L’utente inserisce le credenziali d’accesso.

Immagine che contiene diagramma

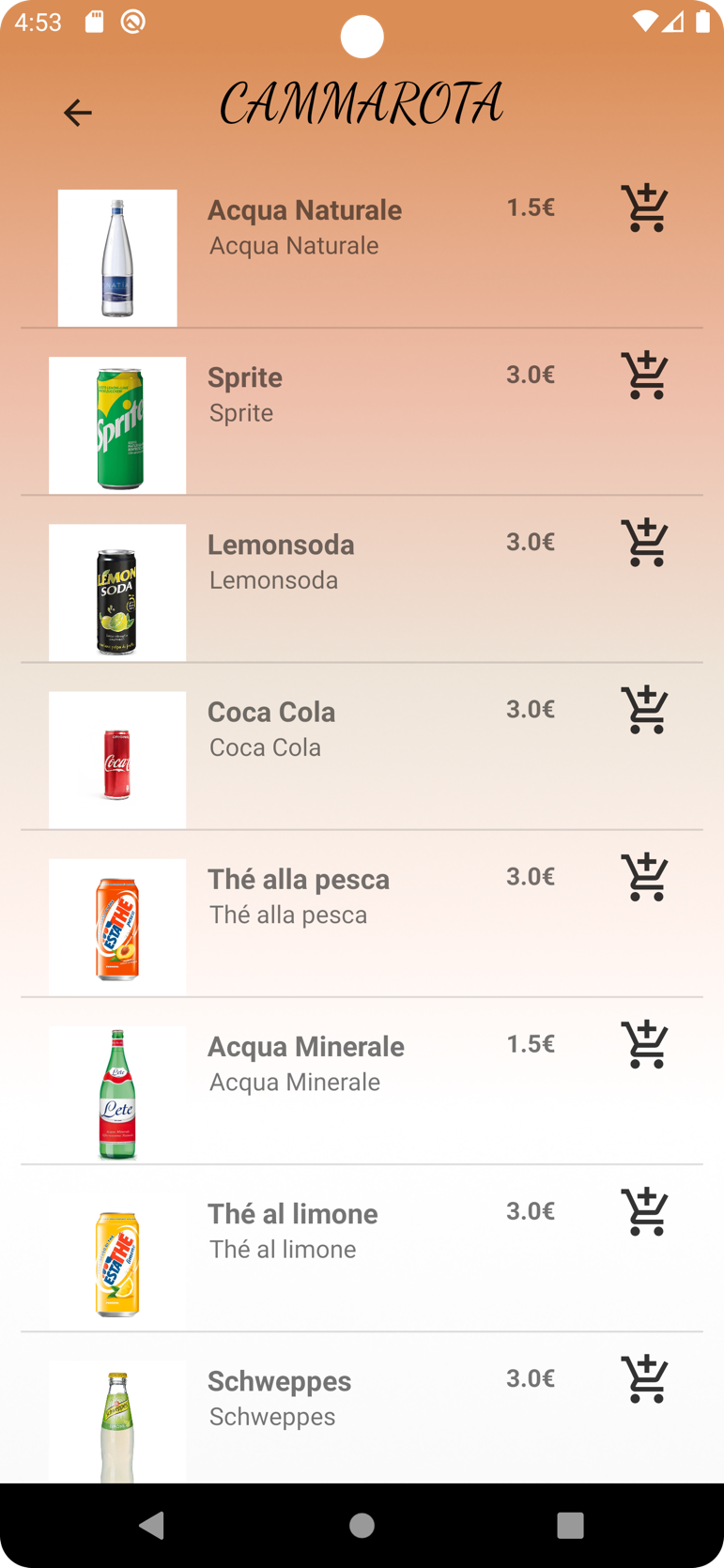
Descrizione generata automaticamente

Una volta cliccato il bottone “accedi”, se le credenziali sono corrette l’utente sarà reindirizzato al menù principale.

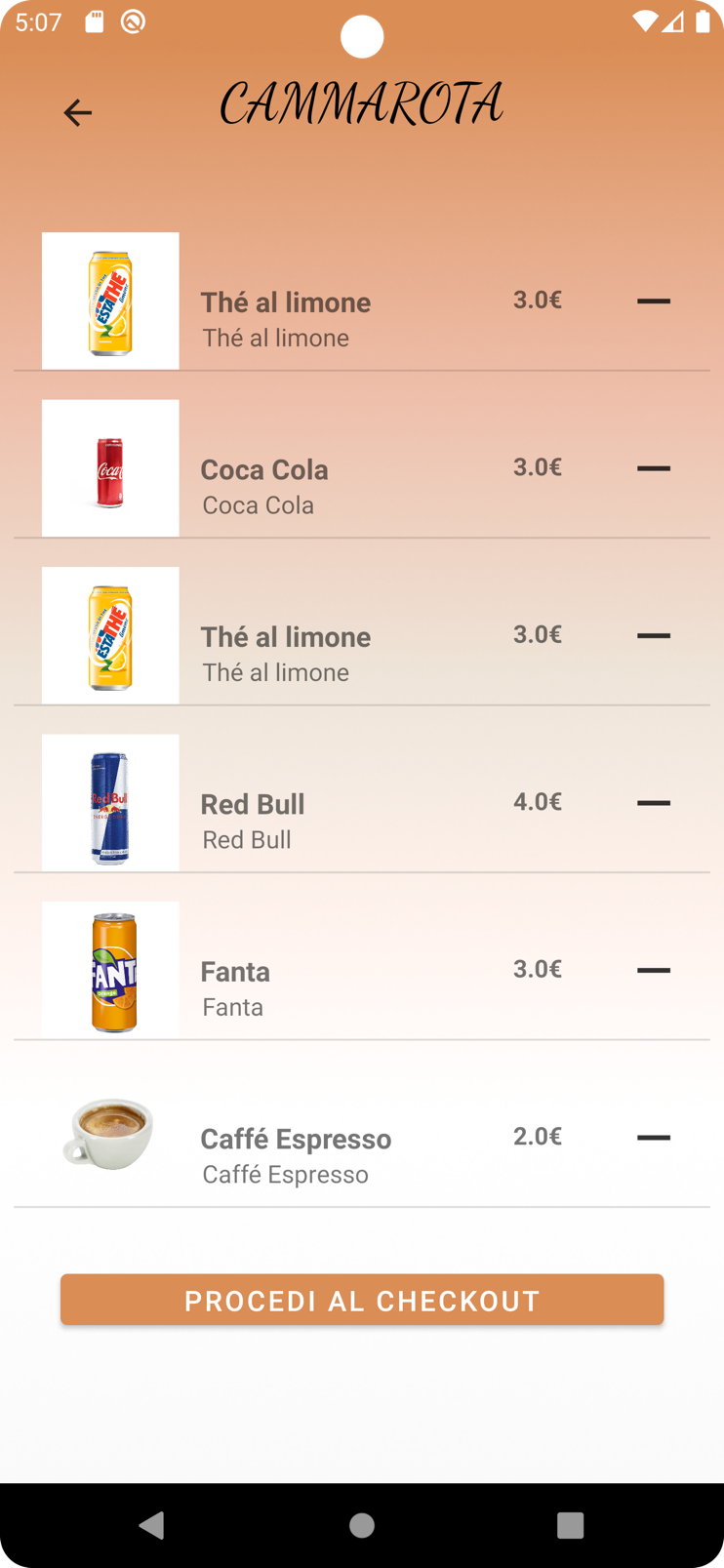


L’utente ha la possibilità di scegliere tra le bevande consigliate oppure accedere alle specifiche categorie.

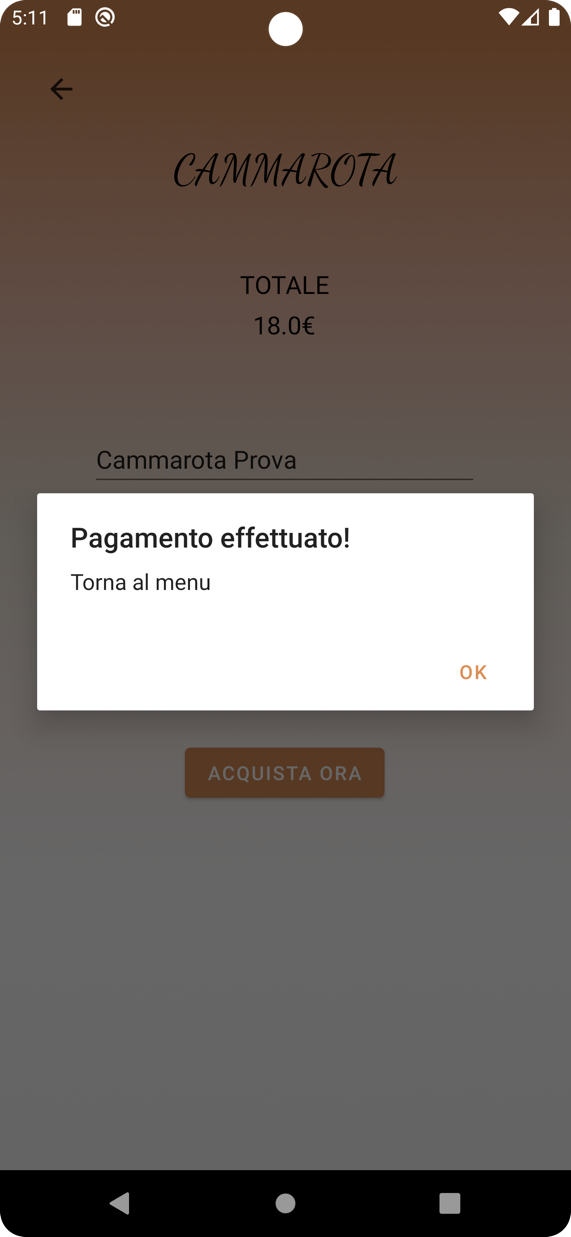
Ad esempio se accedesse alla categoria “soft drink” comparirà questa schermata:



Dopo aver aggiunto le bevande al carrello, l’utente può visualizzare quest’ultimo ed eventualmente rimuovere bevande.



Infine può procedere al checkout ed effettuare il pagamento.

Una volta inseriti i dati ed effettuato il pagamento, l’utente ritornerà al menù principale.

**Documentazione APIs e Source Code su git**

E’ stato utilizzato git come strumento di versioning. Git ha semplificato il versionamento e collaborazione nel team. Inoltre, è stata realizzata una documentazione dettagliata delle APIs del server in C in modo aumentare la manutenibilità del software e facilitarne la comprensione.

Link repository: https://github.com/AlessandroBonomo28/DrinkShop