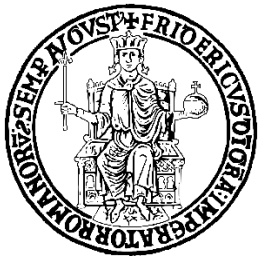
UNIVERSITÀ DEGLI STUDI DI NAPOLI “FEDERICO II”



Dipartimento di Ingegneria Elettrica e delle Tecnologie dell’Informazione

Corso di Laurea Triennale in Informatica

PROGETTO DI LABORATORIO DI SISTEMI OPERATIVI

IL SUPERMERCATO

Studenti

Gianfranco Duminuco
N86 004061

Fabrizio Formicola
N86 003487

Anno Accademico 2023/2024

Sommario

Traccia

Progetto 1: Il supermercato - Lo studente dovrà realizzare la simulazione di un sistema che modella un supermercato con K casse e frequentato da un certo numero di clienti. Il numero dei clienti nel supermercato è contingentato: non ci possono essere più di C clienti che fanno acquisti (o che sono in coda alle casse) in ogni istante. All'inizio, tutti i clienti entrano contemporaneamente nel supermercato, successivamente, non appena il numero dei clienti scende a C-E (0<E<C), ne vengono fatti entrare altri E. Ogni cliente spende un tempo variabile T all'interno del supermercato per fare acquisti, quindi si mette in fila in una delle casse che sono in quel momento aperte ed aspetta il suo turno per "pagare" la merce acquistata. Dopo aver pagato, il cliente esce dal supermercato. Ogni cassa attiva ha un cassiere che serve i clienti in ordine FIFO con un certo tempo di servizio. Il tempo di servizio del cassiere ha una parte costante (diversa per ogni cassiere) più una parte variabile che dipende linearmente dal numero di prodotti acquistati dal cliente che sta servendo. I clienti che non hanno acquistato prodotti (P=0), non si mettono in coda alle casse.

- Opzionale: i clienti senza acquisti non si mettono in coda, ma prima di uscire dal supermercato devono attendere il permesso di uscire. Il permesso può essere dato da un direttore del supermercato.
- Opzionale: la scelta degli acquisti di un cliente può essere fatta o in ordine casuale oppure scelta dal cliente su una semplice interfaccia grafica/testuale

Analisi e specifica dei requisiti

Il supermercato deve essere dotato di entità che possano gestire l'affluenza dei clienti al suo interno, in quanto il numero di clienti deve essere limitato.

Allo stesso modo, deve essere limitato il numero delle casse (e dei relativi cassieri) presenti, ed ogni cassa potrà gestire un solo cliente per volta, selezionato da una coda che dovrà essere servita in ordine FIFO.

I clienti devono poter essere liberi di visualizzare i prodotti che offre il supermercato (ossia, un catalogo coi prodotti ed i rispettivi prezzi), devono poterli aggiungere e rimuovere dal carrello (per tutto il tempo che vogliono), e nel caso siano soddisfatti, devono poter pagare dopo essersi messi in fila. Anche nel caso in cui i clienti non dovessero essere soddisfatti dell'offerta, questi dovranno aver la possibilità di lasciare il supermercato senza effettuare acquisti. Per questo compito è stata designata l'entità del Direttore.

Deve essere possibile, per gli utilizzatori dell'applicativo, scegliere gli acquisti tramite un'interfaccia grafica.

Requisiti

Visualizzazione dei prodotti	Ai clienti deve essere garantita la visualizzazione di un catalogo di prodotti da cui scegliere cosa acquistare.
Aggiunta di un prodotto al carrello	I clienti nel supermercato devono poter aggiungere dei prodotti al loro carrello.
Rimozione di un prodotto dal carrello	I clienti nel supermercato devono poter rimuovere dei prodotti dal loro carrello.
Visualizzazione dei prodotti nel carrello	I clienti nel supermercato devono poter visualizzare tutti i prodotti nel loro carrello.
Inserimento del cliente in coda per le casse	I clienti, dopo aver terminato l'aggiunta dei prodotti al carrello, devono potersi mettere in fila per le casse.
Pagamento alla cassa	I cassieri devono essere in grado di "processare" il contenuto dei carrelli, consentendo poi al cliente di pagare.
Uscita del cliente dopo l'acquisto	I clienti, dopo aver effettuato il pagamento, devono poter lasciare il supermercato.
Uscita del cliente anche senza acquisti	I clienti che non hanno aggiunto prodotti al carrello ma che vogliono uscire devono essere in grado di farlo.
Ordine di servizio	I clienti devono essere serviti in ordine FIFO, ossia First In First Out.

Vincoli

Massimo numero di clienti	Il numero massimo di clienti presenti all'interno del supermercato deve essere limitato superiormente da un valore C.
Entrata in base alla capacità attuale	Se sono già presenti C clienti all'interno del supermercato, non possono entrarne altri prima che ne escano un numero E.
Uscita di clienti che bloccano il sistema	Si deve garantire un meccanismo di controllo sui clienti all'interno del supermercato per far sì che eventuali utenti inattivi non blocchino il funzionamento del sistema.
Interfaccia grafica	Si deve implementare un'interfaccia grafica che consenta agli utenti di selezionare i prodotti da acquistare.

Architettura e scelte gestionali

Il progetto consiste nella produzione un applicativo **client-server**, usando **Docker-compose**.
Il server è stato sviluppato (come da requisiti) in **linguaggio C**, mentre il client è stato sviluppato in linguaggio **Dart**, in particolare con l'utilizzo del framework **Flutter**.
Client e server sono in **locale** e comunicano attraverso l'uso di **Socket** (come da requisiti).

La scelta di utilizzare Docker-compose è dovuta alla facilità di apprendimento del tool.
La scelta di utilizzare Flutter è dovuta ad:

- ottime capacità del framework nella gestione delle socket attraverso funzioni e widgets forniti dalla libreria 'dart:io';
- possibilità di creare, con un codice snello e facilmente intuitivo, degli applicativi funzionanti sia su dispositivi Android che iOS, nonché applicazioni desktop o web-based;
- esperienza pregressa degli sviluppatori nell'utilizzo di tale framework.

Il **server** è responsabile della gestione dell'intero supermercato, ossia della fila per l'ingresso, della possibilità da parte di un cliente di aggiungere e rimuovere prodotti dal carrello una volta entrati, della fila per le casse, delle operazioni effettuate dai cassieri (variabili da cassiere a cassiere, e da prodotto a prodotto), del pagamento, e del lavoro del Direttore che si occupa di "permettere" ai clienti di uscire.

Il server effettua le sue operazioni tramite numerosi **thread**:

- il main thread ha il compito di accettare le richieste per assegnarle ad un thread dedicato al processarle;
- un thread per l'interfaccia grafica per consentire agli utenti di interagire col sistema;
- N thread (in base al numero di cassieri che si vogliono utilizzare) per la gestione della coda di clienti alle casse;
- un thread per la rimozione di clienti che non interagiscono più da diverso tempo col sistema;
- un thread (quello del Direttore) che consente ai clienti che non hanno effettuato acquisti di uscire dal supermercato;
- un thread dedicato alla pulizia dei carrelli che sono stati "lasciati" dai clienti dopo la loro uscita dal supermercato.

Il **client** rappresenta un utente che vuole entrare (con lo scopo di effettuare acquisti) nel supermercato. Ogni cliente viene generato eseguendo l'applicazione client sviluppata in Flutter su un dispositivo. In aggiunta, è stato implementato un client in C con interfaccia testuale per effettuare test in maniera semplificata con molteplici client attivi contemporaneamente.

Per evitare **problemi di stallo** dovuti a clienti che per qualche motivo (connessione o chiusura dell'app) rimangono all'interno del supermercato, sono stati utilizzati dei thread appositi che si occupano di, dopo un tempo periodico, controllare che non ci siano clienti inattivi per troppo tempo (tempo variabile, impostabile manualmente). I thread relativi hanno una funzionalità nell'applicazione: quella dell'Employee (che si occupa di svuotare eventuali carrelli rimasti inutilizzati nel supermercato per troppo tempo) e quella del Bouncer (ossia, il buttafuori che controlla se dei clienti sono rimasti "fermi" nella coda di ingresso, causando un blocco, e che quindi devono essere rimossi).

Gestione della concorrenza

La corretta gestione della concorrenza è un aspetto cruciale per il funzionamento dell'applicativo: in molti punti del flusso di esecuzione, potrebbe capitare che dei clienti o dei cassieri provino ad accedere (ed eventualmente apportare modifiche) alla stessa risorsa, cosa che ovviamente non può essere consentita.
Per ovviare a problemi simili, si è fatto un ampio utilizzo dei **mutex**. In particolare sono stati utilizzanti i seguenti mutex:

- **mutex_entrance_queue**: per non consentire a due clienti in contemporanea di inserirsi nella stessa posizione all'interno della fila di ingresso al supermercato;
- **mutex_checkout_queue**: per non consentire a due clienti in contemporanea di inserirsi nella stessa posizione all'interno della fila per le casse;
- **mutex_clients_number**: per non consentire a diversi utenti (che siano Direttore, Buttafuori o Employee) di modificare il numero dei clienti totali presenti in quel momento nel supermercato;
- **mutex_ticket_snail**: per evitare che due clienti in fila all'ingresso possano prendere lo stesso bigliettino numerato dalla chiocciola prima di entrare nel supermercato;
- **mutex_carts**: per evitare che clienti diversi possano "prendersi" lo stesso carrello o che altri utenti possano effettuarvi operazioni sopra prima del tempo debito. Per gestire bene i carrelli, ognuno di questi è stato dotato di un "mutex personale" e di uno stato, e tramite la combinazione tra le due cose, è possibile distinguere i momenti in cui il carrello deve essere utilizzato dal cliente (per essere riempito), da un cassiere (per essere processato) o da un'altra figura come Employee o Direttore (per essere svuotato e "ceduto" al prossimo cliente).
- **mutex_cashiers**: per evitare che più cassieri provino ad assistere lo stesso cliente.

Protocollo per la comunicazione client-server

Per la **comunicazione client-server** si è optato per un protocollo personalizzato, avente la seguente sintassi:

tipo : id_richiedente : comando \n
payload

In cui:

- **tipo**: definisce la tipologia di richiesta, ossia la distinzione tra richieste effettuate da un cliente per una determinata operazione, o richieste per ottenere il catalogo dei prodotti.
- **id_richiedente**: identifica il cliente che effettua una richiesta;
- **comando**: specifica la funzionalità che deve essere eseguita sul server;
- **payload**: consente di specificare informazioni aggiuntive necessarie al server per adempiere alla richiesta.

Moduli e rispettive funzionalità

Main

Descrizione: gestisce la logica generale del server.
Funzionalità:

main	Entry point del server. Inizializza i carrelli, la coda alla cassa, la coda all'ingresso, i mutex e i cassieri.
process	Processa la richieste che arrivano dal client.
read_request	Legge le richieste dal client.
send_response	Invia le risposte al client.
sendCatalog	Restituisce il catalogo coi prodotti all'utente.
reorderCarts	Pulisce e "risistema" i carrelli se non vengono utilizzati per un certo periodo di tempo.
bouncerAtEntrance	Controlla che ci siano ancora clienti in fila all'ingresso, e nel caso in cui questi non entrino in un certo tempo, li butta fuori.
ui	Serve ad aggiornare costantemente l'interfaccia grafica

Cart

Descrizione: classe per la gestione del carrello e degli oggetti al suo interno.
Funzionalità:

add_product	Per aggiungere prodotti al carrello.
remove_product	Per rimuovere prodotti dal carrello.
print_cart	Per la stampa nella socket del contenuto del carrello.
calculate_total	Calcola il prezzo totale degli oggetti nel carrello.
initialize_carts	Svuota completamente un carrello e ne setta lo stato a "FREE", rendendolo disponibile per i prossimi clienti.
clear_cart	Svuota il carrello.

Cashier

Descrizione: classe per la gestione delle casse e delle funzionalità dei cassieri.
Funzionalità:

cashierEnters	Crea ed inizializza un cassiere, impostando anche i valori "variabili" sui suoi tempi per il processamento dei prodotti.
waitQueue	Funzione che consente al cassiere di restare in attesa di eventuali carrelli che si trovano nella coda per andare alla cassa.
processCart	Funzione che gestisce le operazioni del cassiere nel processare un carrello che arriva dalla coda, consentendogli di effettuare il "checkout" e di pagare.

Client

Descrizione: classe per la gestione di tutte le operazioni effettuabili da un client(e).
Funzionalità:

get_clients_number	Per ottenere il numero di clienti nel supermercato.
increase_clients_number	Per incrementare il numero di clienti nel supermercato.
decrease_clients_number	Per decrementare il numero di clienti nel supermercato.
clientParser	Per effettuare il parsing delle richieste ricevute dal client.
clientEnters	Gestisce l'entrata di un cliente nel supermercato: se il cliente è libero di poter entrare poiché ci sono carrelli disponibili, viene fatto entrare, altrimenti viene messo in coda.

canEnter	Per verificare se un utente può entrare nel supermercato.
clientEntersInEntranceQueue	Gestisce le operazioni del cliente alla coda di ingresso: se un cliente è appena arrivato, gli viene assegnato un ticket.
clientExits	Gestisce le operazioni da effettuare dopo l'uscita dal supermercato di un cliente, come la pulizia del database.
clientAddItem	Per aggiungere un prodotto scelto al carrello.
clientRemoveItem	Per rimuovere un prodotto scelto dal carrello.
clientPrintCartContent	Per visualizzare l'intero contenuto del carrello.
clientEntersCheckoutQueue	Gestisce le operazioni del cliente alla coda per le casse: se il cliente non ha prodotti nel carrello, viene mandato avanti senza entrare nella coda; se il cliente è nel supermercato ed è pronto per andare alla cassa, viene inserito alla coda.
clientPays	Per la gestione del pagamento da parte del cliente. Se il cliente paga, lo stato del carrello viene settato a "PAYED" ed il cliente è pronto per essere accompagnato fuori dal Direttore.

EntranceQueue

Descrizione: classe per la gestione della coda di ingresso al supermercato.
Funzionalità:

add_client_to_checkout_queue	Aggiunge un cliente alla coda.
remove_client_from_cash_queue	Rimuove un cliente dalla coda.
remove_client_from_cash_queue_by_id	Rimuove un cliente dalla coda dato il sui ID.
clients_number_checkout_queue	Per ottenere il numero di clienti nella coda.
position_client_checkout_queue	Per ottenere la posizione di un cliente all'interno della coda.

CheckoutQueue

Descrizione: classe per la gestione della coda per andare alle casse.
Funzionalità:

add_client_to_checkout_queue	Aggiunge un cliente alla coda.
remove_client_from_cash_queue	Rimuove un cliente dalla coda.
remove_client_from_cash_queue_by_id	Rimuove un cliente dalla coda dato il sui ID.
clients_number_checkout_queue	Per ottenere il numero di clienti nella coda.
position_client_checkout_queue	Per ottenere la posizione di un cliente all'interno della coda.

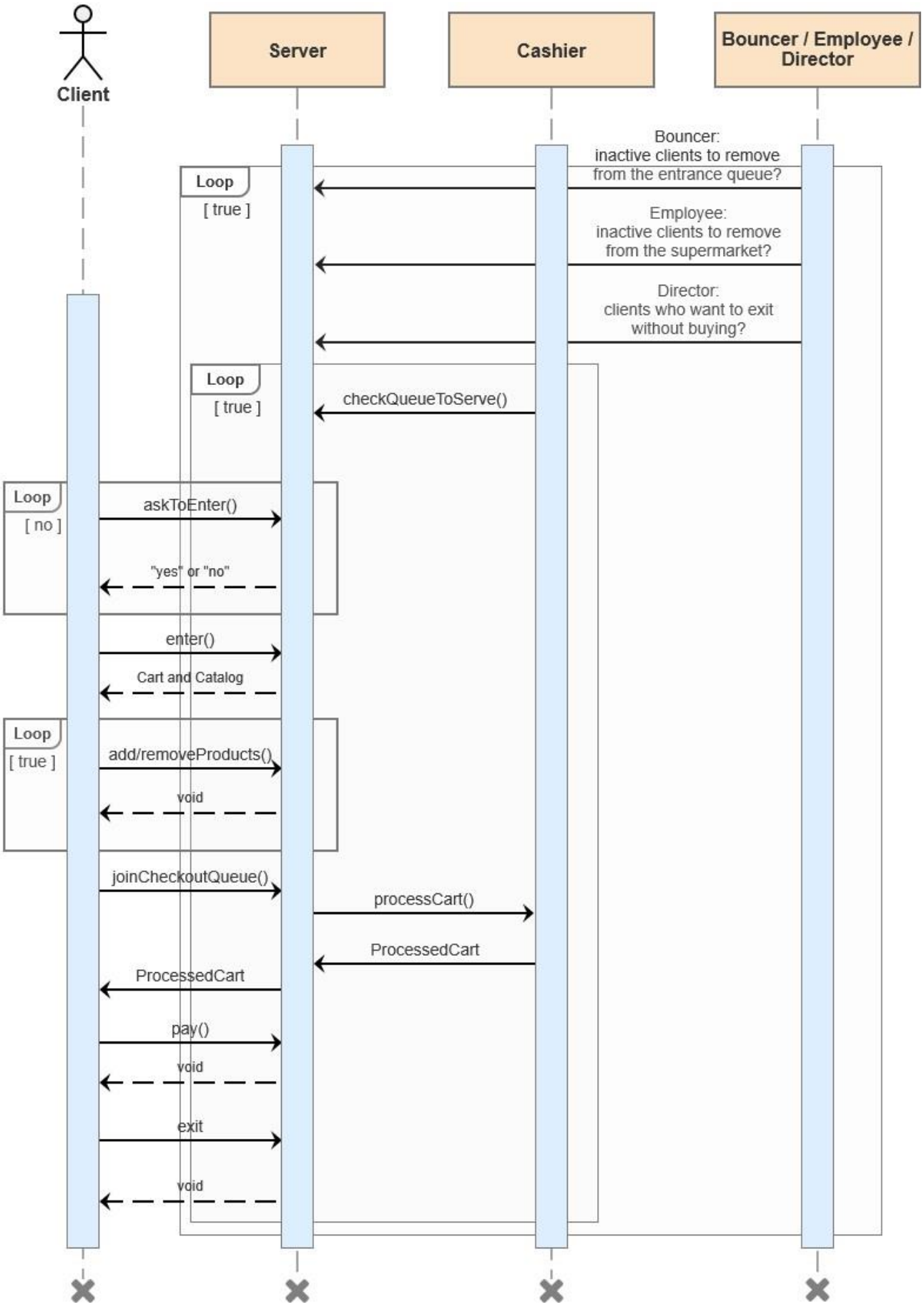
Director

Descrizione: classe per il controllo delle uscite dal supermercato gestite dal Direttore.
Funzionalità:

checkExit	Controlla periodicamente se c'è qualche carrello in stato di "CONFIRM", ossia lo stato in cui un cliente vuole uscire dal supermercato senza aver effettuato acquisti, in cui è necessaria l'approvazione del Direttore.

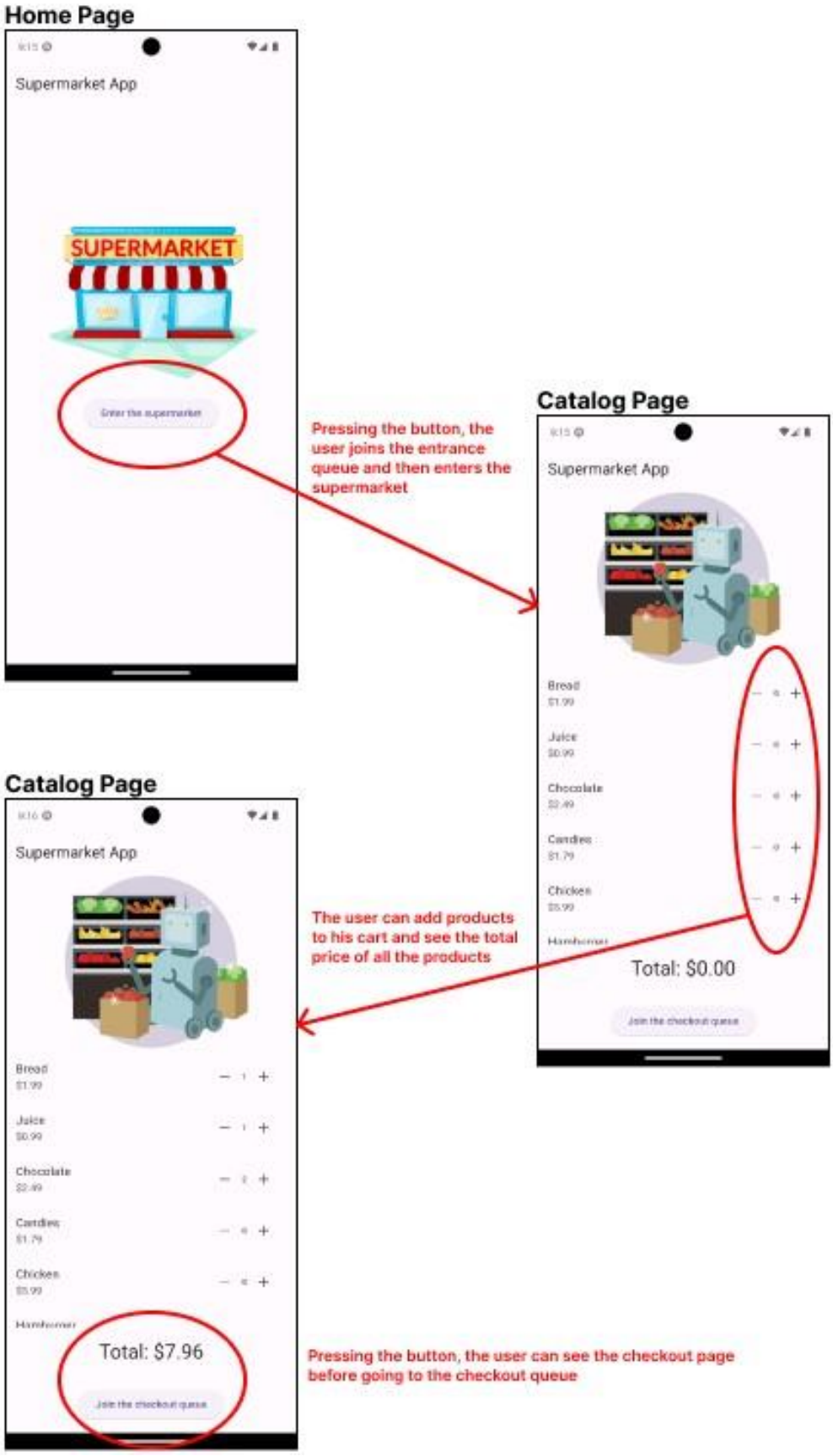
Sequence Diagram per il flusso di esecuzione

Esempio del flusso di esecuzione dell'applicazione in uno scenario lineare e senza errori.



Simulazione del flusso di esecuzione tramite interfaccia grafica

Flusso di esecuzione dell'applicativo utilizzando il client in Flutter tramite dispositivo Android.



Checkout Page

Pressing the button, the user joins the and then the cashier will be able to process the cart.

Receipt Page

After the payment, the client is able to leave the supermarket.

Contatti degli sviluppatori

Gianfranco Duminuco – N86 004061
email istituzionali: g.duminuco@studenti.unina.it

Fabrizio Formicola – N86 003487
email istituzionale: fabri.formicola@studenti.unina.it