Backend sito e-commerce

ferdinandi.1958589

April 2024

1 Descrizione generale del sistema da realizzare

Il progetto d'esame per il corso di Ingegneria del Software consiste nella creazione del backend di un sito e-commerce. Il sistema permette di gestire tre diverse tipologie di utenti:

- **Utenti Compratori**: coloro che navigano, cercano e acquistano prodotti nel sito e-commerce.
- **Utenti Fornitori**: responsabili della gestione dei prodotti del sito e-commerce.
- **Utenti Trasportatori**: incaricati della gestione delle spedizioni relative agli ordini effettuati dagli utenti Compratori.

Per gestire le diverse tipologie di utenti ed anche al fine di poter garantire le performance necessarie a certe tipologie di utenti , sono stati implementati **tre tipi di server**, ciascuno dedicato a una categoria di utenti.

Nel progetto:

- la *comunicazione tra i processi* - cioè il **middleware** avviene tramite il **database/framework NoSQL Redis**.
- per la *persistenza dei dati elaborati* dal sito e-commerce si fa uso del **database PostgreSQL**.

La struttura generale del sistema prevede che i **tre diversi tipi di utente**, attraverso un' opportuna *applicazione client (che simula il browser web del caso reale)*, **inviino richieste ai rispettivi server** di categoria tramite ***streams Redis dedicati***.
Il **server** coinvolto **elabora la richiesta e invia** al determinato utente (che possiamo chiamare genericamente Client) **il risultato** attraverso le ***streams Redis appropriate***.

Per assicurare l'**unovicità delle richieste dei vari utenti** che effettuano richieste, è fondamentale mantenere attivo il **sessionID**. Pertanto, ogni volta che un *utente si registra o effettua il login*, viene **generato un sessionID** che viene memorizzato e associato all'utente nel database. In *caso di logout*, il **sessionID viene eliminato**.
Questo approccio garantisce che, ogni volta che un utente tenta di eseguire un'azione, è sempre possibile verificare l'esistenza in vita e la validità dell'utente proprio attraverso il sessionID associato.

È importante utilizzare **streams separate per le richieste e le risposte tra Client e server** delle diverse categorie per favorire eventuali necessità di prioritizzare determinate richieste, per esempio per fare in modo che certe tipologie di utenti abbiano accessi più veloci per le loro esigenze.
Inoltre questo approccio garantisce che, in caso di aggiornamento di un determinato server tutti gli altri non risentiranno di eventuali periodi di off-line.

2 Requisiti utente

I **requisiti utente** (spesso chiamati *Funzionali*) sono esigenze, aspettative e funzionalità desiderate degli utenti finali del sistema.

Questi requisiti sono espressi in modo non tecnico e focalizzati sulle necessità dell'utente.

Descriviamo i requisiti:

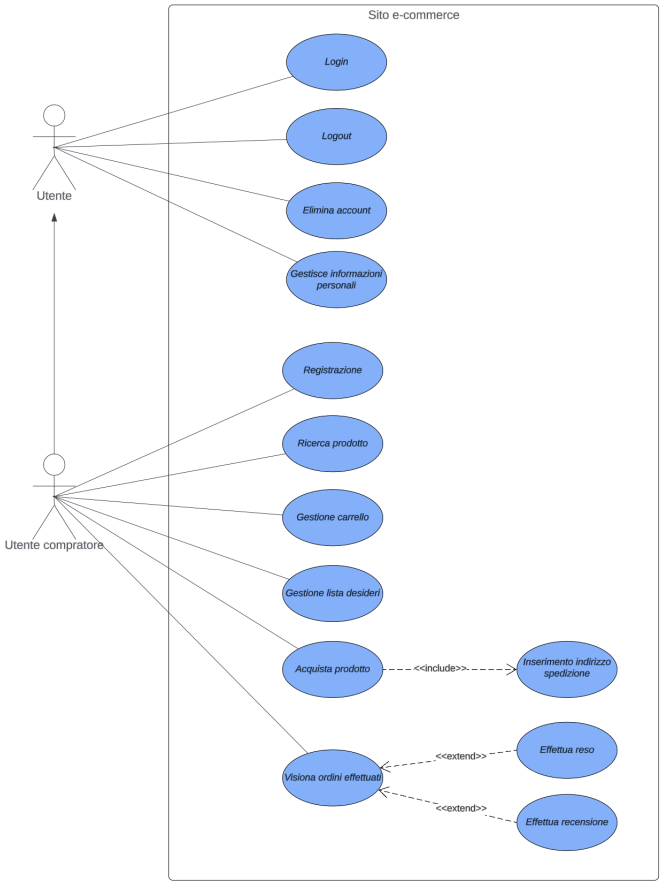
1. Gli *utenti* devono avere la possibilità di **registrarsi** al sito e-commerce.
2. Gli *utenti* devono avere la possibilità di **effettuare il login** al sito e-commerce.
3. Gli *utenti* devono avere la possibilità di **effettuare il logout** al sito e-commerce.
4. Gli *utenti* devono avere la possibilità di **eliminare il proprio account**.
5. Gli *utenti* devono avere la possibilità di poter **gestire il proprio profilo**.
6. Gli *utenti compratori* devono avere la possibilità di **trovare i prodotti** che desiderano acquistare.
7. Gli *utenti compratori* devono avere la possibilità di **gestire i prodotti nel carrello**.
8. Gli *utenti compratori* devono avere la possibilità di **gestire i prodotti nella lista desideri**.
9. Gli *utenti compratori* devono avere la possibilità di **acquistare un prodotto**.
10. Gli *utenti compratori* devono avere la possibilità di **visualizzare la cronologia dei propri ordini e lo stato dell'ordine effettuato**.
11. Gli *utenti compratori* devono avere la possibilità di **effettuare il reso di un prodotto**.
12. Gli *utenti compratori* devono avere la possibilità di **effettuare l'annullamento dell'acquisto di un prodotto**.
13. Gli *utenti compratori* devono avere la possibilità di **gestire le recensioni** di un prodotto acquistato.
14. I *fornitori* devono avere la possibilità di **gestire prodotti al sito e-commerce**.
15. I *trasportatori* devono avere la possibilità di **comunicare con il sistema**.

3 Use case UML.

Di seguito sono riportati i diagrammi dei casi d'uso per l'attore "Utente compratore".

Gli **utenti acquirenti**, come le altre tipologie di utenti (fornitori e trasportatori) interagiscono con il sistema grazie alla possibilità di ***effettuare l'accesso*** (*login*) e il ***logout***, oltre a poter ***eliminare il proprio profilo*** e ***gestire le informazioni personali***.

Più nello specifico, gli **utenti compratori** possono effettuare la ***registrazione*** con determinati campi, ***cercare prodotti***, ***gestire il carrello*** e la ***lista dei desideri***.
Inoltre, gli utenti acquirenti possono ***acquistare prodotti***, includendo ***l'inserimento dell'indirizzo di spedizione***.
In aggiunta, gli utenti acquirenti possono ***visualizzare lo storico degli ordini*** effettuati e, se lo desiderano, possono ***procedere con il reso*** di un prodotto oppure ***lasciare una recensione***.



4 Descrizione requisiti sistema

I **requisiti di sistema** trattano informazioni più specifiche sulle funzioni e i servizi. Descrivono come il sistema deve essere implementato per soddisfare i requisiti utente.

Descriviamo i requisiti:

1. Registrazione:
   (a) Il sistema deve consentire agli **utenti compratori di registrarsi al sito e-commerce** fornendo dei dati specifici :
   - nome-utente;
   - categoria;
   - nome;
   - cognome;
   - e-mail;
   - numero di telefono;
   - residenza (via, città, cap e numero civico);
   - data di nascita;
   - password (almeno 8 caratteri con una lettera maiuscola, un numero e un carattere speciale)
   - conferma password (ovviamente identica alla password scritta in precedenza).

(b) Il sistema deve consentire agli **utenti fornitori di registrarsi al sito e-commerce** fornendo dei dati specifici:

- nome-utente;
- nome;
- categoria;
- cognome;
- e-mail;
- numero di telefono;
- azienda di produzione che rappresenta (al fine di sapere quali prodotto saranno inseriti nel sito);
- password (almeno 8 caratteri con una lettera maiuscola, un numero e un carattere speciale)
- conferma password (ovviamente identica alla password scritta in precedenza).

(c) Il sistema deve consentire agli **utenti trasportatori di registrarsi al sito e-commerce** fornendo dei dati specifici :

- nome-utente;
- nome;
- cognome;
- e-mail;
- numero di telefono;
- ditta di spedizione che rappresenta;
- password (almeno 8 caratteri con una lettera maiuscola, un numero e un carattere speciale)
- conferma password (ovviamente identica alla password scritta in precedenza).

(d) Ogni utente deve avere un suo nome utente, una mail e un session ID **univoci**, diversi da tutti gli altri.

(e) L'e-mail deve contenere il carattere "@".

(f) La data di nascita deve essere conforme alle norme delle date.

(g) La password deve essere di almeno 8 caratteri, un carattere maiuscolo, un carattere speciale, un numero.

(h) La conferma della password deve essere uguale alla password scritta in precedenza.

2. Login:

(a) Per accedere al sistema ci sarà uno spazio dedicato al **login** tramite 2 campi: *nome-utente* e *password*.

(b) Per effettuare il **login** è necessario un **controllo delle credenziali** inserite.

(c) Viene garantito che il sessionID che verrà associato all'utente che si logga sia univoco.

3. Logout:

(a) Il sistema deve permettere il **logout dell'utente** per mantenere sicurezza informatica sui diversi dispositivi.

4. Eliminazione profilo:

(a) Il sistema deve garantire l'**eliminazione del proprio profilo nel sito**.

5. Gestione profilo:

   (a) Il sistema deve permettere di **aggiornare le informazioni relative alla registrazione di un account**.
   Solo alcune di queste possono essere modificate: sia per compratore, fornitore e trasportatore può essere modificato il numero di telefono e la password.
   In particolare per l'utente compratore può essere aggiornata la residenza. Per l'utente fornitore può essere aggiornata l'azienda di produzione e per l'utente trasportatore può essere aggiornata la ditta di spedizione per cui si lavora.

6. Ricerca prodotti:

   (a) Il sistema deve permettere all'utente compratore di **ricercare un prodotto** tramite un motore di ricerca.

7. Gestione prodotti carrello:

   (a) Il sistema deve permettere all'utente di avere un metodo per **inserire i prodotti nel carrello**.

   (b) Il sistema deve permettere all'utente di avere un metodo per **rimuovere i prodotti dal carrello**.

8. Gestione prodotti lista desideri:

   (a) Il sistema deve permettere all'utente di avere un metodo per **inserire i prodotti nella lista desideri**.

   (b) Il sistema deve permettere all'utente di avere un metodo per **rimuovere i prodotti dalla lista desideri**.

9. Acquisto prodotti:

   (a) Il sistema deve permettere agli utenti di **acquistare i prodotti dal carrello**.

   (b) Al momento dell'acquisto del prodotto o di più prodotti, il sistema richiede la conferma delle informazioni di spedizione, tra cui la via di spedizione, la città, il CAP e il numero civico.

   (c) Il sistema deve permettere all'utente di **aggiungere delle carte** per effettuare i pagamenti.

   (d) Una volta che un prodotto è stato acquistato e spedito sarà diminuito il numero di copie disponibili.

10. Visualizzazione ordini effettuati:

    (a) Il sistema permette all'utente che ha acquistato dei prodotti di **vedere l'ordine/i effettuati**.

    (b) Il sistema garantisce che una volta acquistato un prodotto verrà fornito **lo stato dell'ordine e della spedizione** tramite una comunicazione del trasportatore attraverso la piattaforma di e-commerce. (Quando il trasportatore inizia il viaggio per la spedizione dell'ordine lo comunica al sistema. Questo viene mostrato all'utente compratore quando visiona gli ordini effettuati.)

11. Reso prodotto:

    (a) Il sistema permette di **effettuare il reso di un prodotto** specificando il motivo.

(b) Il sistema deve garantire che il **reso** di un prodotto può essere effettuata da un cliente solamente nel caso in cui il cliente abbia comperato questo prodotto. Perciò se un cliente ha uno storico degli ordini deve essere possibile controllare se il prodotto in questione è presente e , se desiderato, il cliente potrà effettuare il reso.

12. Annullamento prodotto:

    (a) Il sistema permette all'utente di **annullare l'ordine effettuato**.

    (b) Il sistema permette di **annullare un ordine** effettuato solamente nel caso in cui l'ordine non è stato già spedito. Perciò se il trasportatore ancora non ha preso il pacco da spedire si può annullare l'ordine.

13. Recensione prodotto:

    (a) Il sistema deve permettere all'utente di **inserire una nuova recensione** di un prodotto che ha acquistato, inserendo testo.

    (b) Il sistema deve garantire che la **recensione** di un prodotto può essere effettuata da un cliente solamente nel caso in cui il cliente abbia acquistato il prodotto. Perciò se un cliente ha uno storico degli ordini deve essere possibile controllare il suddetto elenco e se il prodotto è presente allora il cliente può recensirlo.

    (c) Il sistema permette all'utente di **rimuovere una recensione** di un prodotto, gestendola nel database.

    (d) Il sistema deve permettere all'utente di **dare un voto da 1 a 5 per il prodotto** che ha acquistato.

14. Gestione prodotti nel sito:

    (a) Il sistema deve permettere solo agli ***utenti fornitori*** la disponibilità di una sezione dedicata per **inserire un prodotto specifico**, le informazioni da inserire per ogni prodotto dovranno essere le seguenti:

        - nome;
        - descrizione;
        - categoria;
        - azienda di produzione, fornitore da cui è prodotto;
        - numero di copie disponibili del prodotto;
        - prezzo: deve essere espresso in euro;

    (b) Il sistema deve permettere agli utenti fornitori di **rimuovere un prodotto inserito precedentemente**.

15. Comunicazione trasportatore con sistema:

    (a) Il sistema garantisce che il trasportatore, **appena prende in carico la spedizione lo comunica alla piattaforma** in una zona dedicata.

    (b) Il sistema garantisce che il trasportatore **quando ha consegnato il pacco lo comunica al sistema**, attraverso una zona dedicata.

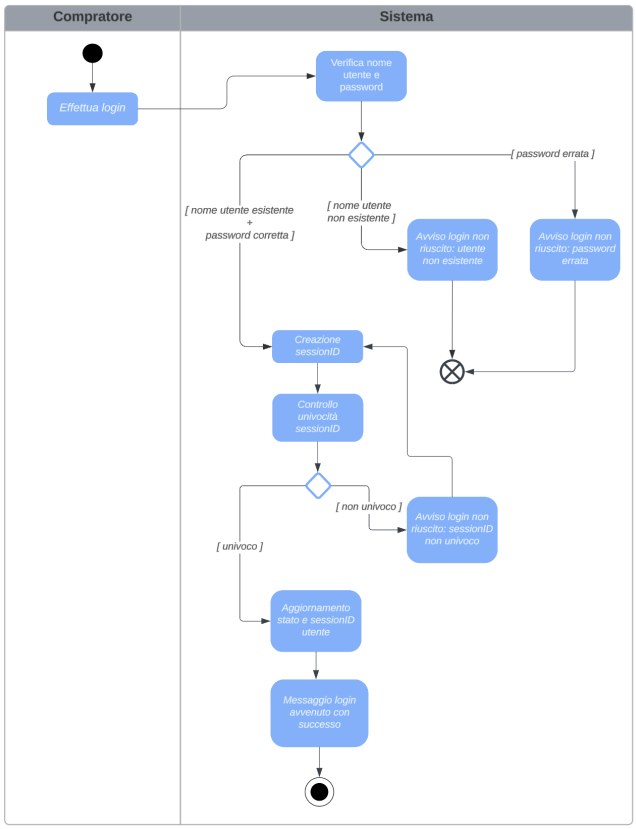
    (c) Per garantire semplicità al sistema ogni utente trasportatore si occupa della **spedizione di un solo ordine alla volta**.

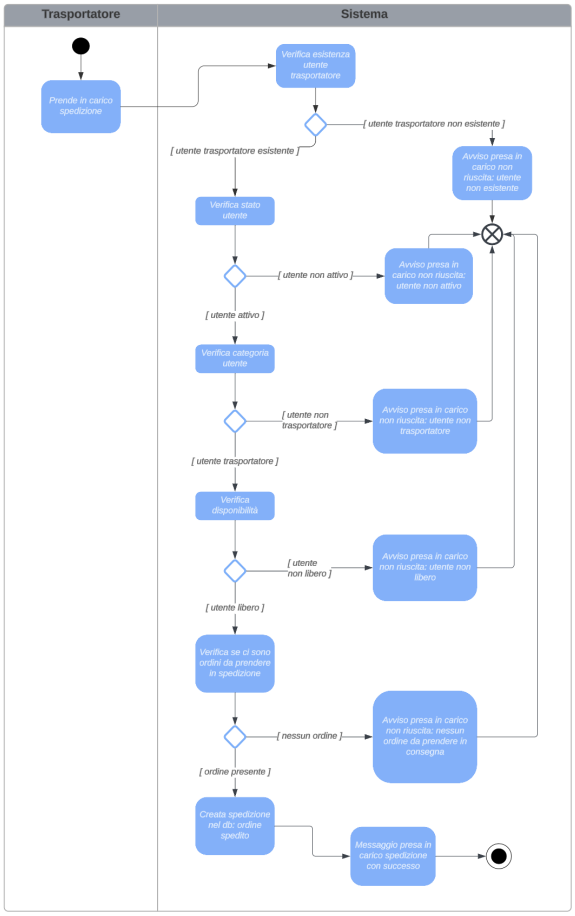
5 Activity Diagram UML.

Ho realizzato 2 diagrammi di attività UML al fine di mostrare il **flusso di lavoro** (*attività*) **dei diversi componenti**.

Il *diagramma di attività alla sinistra* mette in evidenza il componente **UtenteCompratore** e il **sistema** (server).
Questi 2 componenti comunicano tra di loro per soddisfare i requisiti utente.
In questo caso si può notare come l'utente compratore effettua la richiesta per il login.
Il sistema effettua determinati controlli e restituisce un avviso di successo o di errore relativo alla soddisfazione del requisito utente.

Il *diagramma sulla destra* mette invece in evidenza il componente **UtenteTrasportatore** e il **sistema** (server). In questo caso viene mostrato come, per soddisfare la richiesta dell'utente trasportatore di prendere in consegna un ordine richieda molti controlli.
Anche in questo caso viene restituito un avviso di successo o di errore relativo alla soddisfazione del requisito utente.

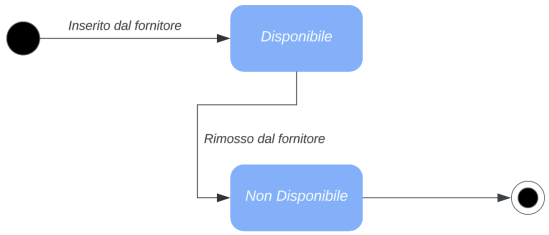




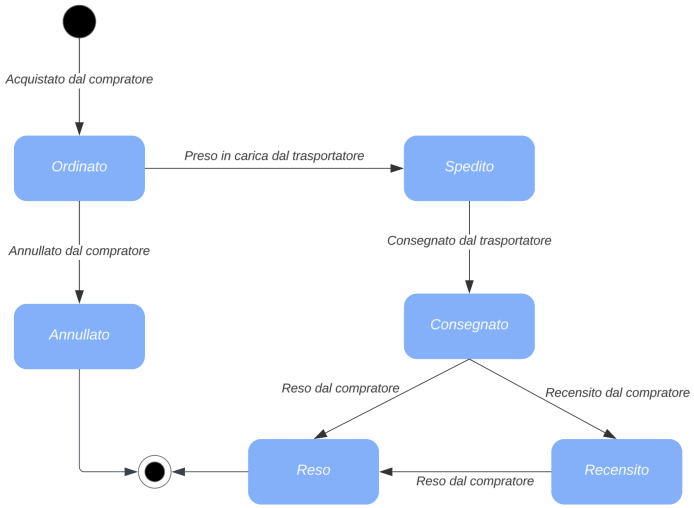
6 State Diagram UML.

Ho realizzato 3 diagrammi di stato. Questi sono utili al fine di rappresentare i **vari stati di una componente nel sistema in reazione a qualche evento**.

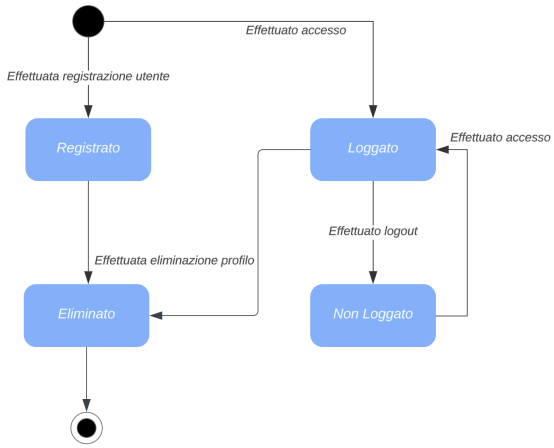
1. Il seguente diagramma mostra come un **prodotto** può avere 2 stati: *disponibile* e *non disponibile*.



2. Il seguente diagramma mostra come un **ordine** si può trovare in diversi stati in reazione al comportamento dell'utente trasportatore e compratore.



3. Nell'ultimo diagramma si evidenzia il passaggio dell'**utente** attraverso vari stati. Per facilitare l'implementazione del sistema, ho considerato che un utente, al momento della registrazione, assuma automaticamente lo stato attivo. Di conseguenza, può compiere azioni come se fosse già effettivamente loggato.

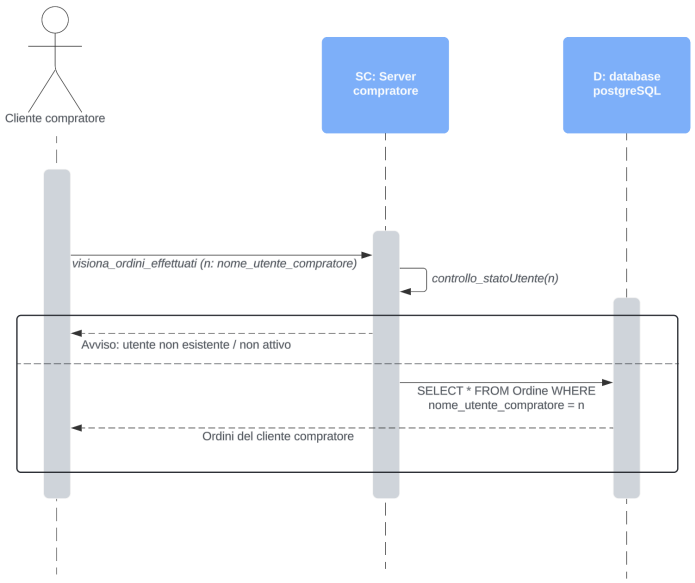


7 Message Sequence Chart UML.

Questo diagramma è utilizzato per modellare le **interazioni tra gli attori e gli oggetti in un sistema**, nonché le **interazioni tra gli stessi oggetti**.

Nel diagramma, si mostra come il **cliente acquirente** effettua una richiesta al server per *visualizzare gli ordini effettuati*.
Il **server**, a sua volta, *esegue dei controlli*, come verificare l'esistenza dell'utente e la validità della sessione attiva. Successivamente, viene *eseguita una query al database* e, in caso di successo, gli *ordini vengono mostrati all'utente acquirente*.

Nel caso in cui l'utente non esista o non abbia una sessione attiva, viene *inviato un avviso di errore*.



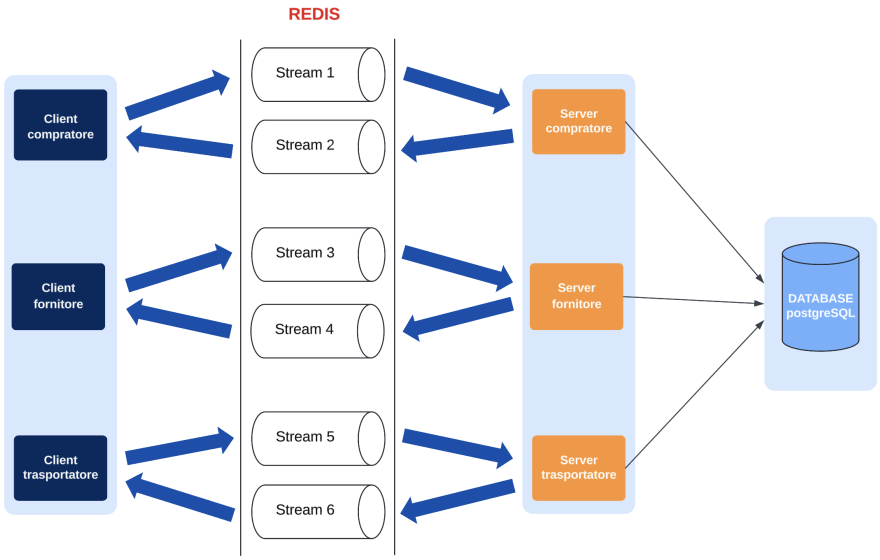
8 Requisiti non funzionali.

- **Prestazioni**: Il sistema deve gestire un elevato volume di traffico e transazioni simultanee senza subire degradazioni delle prestazioni.
- **Tempi di risposta:** Il sistema deve garantire tempi di risposta rapidi, assicurando che ogni richiesta del cliente venga soddisfatta immediatamente, senza attese prolungate.
- **Distribuzione su server**: Il sistema è distribuito su tre server distinti in base alle categorie di servizio. Questa architettura permette il mantenimento della disponibilità del sistema anche in caso di manutenzione di uno dei server, assicurando che gli altri due rimangano operativi e possano rispondere alle richieste dei clienti.
- **Manutenibilità**: Il sistema è progettato per essere facilmente comprensibile, modificabile e testabile, facilitando così l'aggiornamento e l'evoluzione del software nel tempo.
- **Timeout di lettura bloccante**: Il server rimane in attesa di lettura bloccante per un massimo di 16 minuti anche in assenza di richieste dai clienti, assicurando una gestione efficace del traffico e la prontezza nel rispondere alle richieste successive.
- **Feedback utente**: Durante l'esecuzione del programma, il sistema fornisce feedback chiaro e tempestivo nel terminale, consentendo agli utenti di comprendere le richieste e le risposte durante l'interazione con il sistema.

9 Diagramma architettura del sistema.

L'architettura del sistema che stiamo utilizzando è basata su un modello **client-server** dove la rete che consente ai client di accedere ai server avviene attraverso Redis. Nel diagramma, possiamo osservare chiaramente le interazioni tra le varie componenti.

- **Clienti**: ci sono diverse categorie di clienti che interagiscono con il sito di e-commerce. Questi clienti inviano le loro richieste ai server pertinenti attraverso Stream Redis.
- **Redis**: questo componente, che è un **database NoSQL**, funge da intermediario per instradare le richieste dai clienti ai server e per recapitare le risposte dei server ai clienti. In pratica, Redis si comporta come un ponte che facilita la comunicazione tra clienti e server.
- **Database PostgreSQL**: è utilizzato per archiviare i dati di sistema, come informazioni sugli utenti, sui prodotti, sulle spedizioni, sugli ordini e altro ancora.
- **Server**: Questi componenti gestiscono le richieste provenienti dai clienti. Interagiscono con il database PostgreSQL per soddisfare tali richieste e quindi trasmettono le risposte ai clienti appartenenti alle rispettive categorie. Questa comunicazione avviene nuovamente attraverso Stream Redis.



10 Descrizione generale per l'implementazione.

Per implementare il backend del sito e-commerce descritto in precedenza, sono state organizzate **diverse cartelle** per gestire separatamente le funzionalità dei server per i client compratore, fornitore e trasportatore.

Ogni categoria di client e server dispone di una propria cartella, suddivisa ulteriormente in tre sotto-cartelle per i file *sorgente*, *oggetto* ed *eseguibili*.

10.1 Interazione tra client e server.

Consideriamo l'**interazione generale tra un client e un server**, appartenenti a una delle tre categorie specificate.

All'interno della componente del **client**, viene stabilita la **connessione a Redis**, utilizzando l'indirizzo '*localhost*' e la porta predefinita *6379*.
Successivamente, vengono **eliminati gli Stream già esistenti** per garantire l'assenza di messaggi provenienti da esecuzioni precedenti.

10.1.1 Test pianificati.

Per assicurare una **valida verifica del sistema**, è stata creata una ***cartella di test*** all'interno di ogni categoria client contenente file con estensione ".txt" che fungono da **test**, poiché contengono le **richieste che giungono al server** della categoria corrispondente tramite le Streams Redis.
Perciò per condurre **test non casuali**, si apre uno dei file di test presenti nella cartella corrispondente alle richieste del client.

Per consentire una **verifica più veloce dei dati e un controllo dei risultati**, ad ogni esecuzione del programma si apre anche un file in modalità di scrittura (presente in una cartella *result*) in cui è presente **ogni richiesta del client e ogni risposta del server** , per ogni categoria, enumerati con un indice. Questo file è utile per controllare tutte le richieste e risposte dei client e server. Se il file non esiste viene creato.

Ritornando all'esecuzione della componente client *per ogni richiesta presente nel file di test*, il **client invia un messaggio al server tramite lo stream di scrittura** di Redis, annotando la richiesta sul file di risultato aperto precedentemente. Dopo ogni richiesta, il programma viene temporaneamente interrotto per *mezzo secondo* al fine di mantenere l'ordine delle richieste. (necessario per i test pianificati)

A questo punto interviene il **server**. Quest'ultimo effettua anch'esso la connessione a Redis con l'indirizzo '*localhost*' e la porta di default *6379*. In più verrà effettuata anche la **connessione al database postgreSQL** in '*localhost*' sulla porta di default *5432*.

Dopo aver preparato anche il server il file di scrittura per registrare le risposte del server e le richieste del client, il **processo entra in un ciclo in cui viene costantemente letta lo stream di lettura del server**, contenente le richieste del client.
Questa lettura è bloccante, comportando un'attesa di 1000000000 millisecondi (16 minuti), e viene ripetuta finché l'esecuzione del codice non viene interrotta.

Per ogni richiesta, il **server estrae l'azione da eseguire** *insieme ai relativi parametri necessari* e procede a soddisfare la richiesta del client **eseguendo l'azione**. La **risposta del server viene inviata al client** tramite lo stream di scrittura. Il **client**, a sua volta, **riceve la risposta** sullo stream di lettura.

Questo avviene finché ci sono richieste da parte del client. Successivamente, il server rimarrà in ascolto sullo stream di lettura di Redis per ricevere ulteriori messaggi e gestire eventuali nuove richieste.

10.1.2 Test randomici.

Per garantire la capacità del sistema di generare sequenze di **test casuali**, e simulare **scenari più realistici**, è possibile utilizzare valori casuali che consentono una variazione nei test e una maggiore diversificazione delle sequenze di prova.
A tal fine, anziché leggere le richieste da un file di test, è stato creato un array contenente tutte le possibili richieste di un client di una determinata categoria, insieme ad altri array per ottenere parametri casuali.

Una volta **inviato il messaggio corrispondente alla richiesta del client**, il **server elabora la richiesta** esattamente come fatto in precedenza, **inviando al client la risposta** corrispondente, sempre attraverso l'utilizzo di Stream Redis.

Dopo ogni richiesta del client, il **programma viene** temporaneamente **interrotto** per *5 secondi* al fine di mantenere l'ordine delle richieste e di rendere visibile sul terminale l'esecuzione.

Le **interazioni tra client e server** sono **visibili** tramite **output formattato sul terminale** e dalla **scrittura delle richieste del client e delle relative risposte del server su file che si trovano nella cartella 'result'** all'interno di ogni categoria di server e client.

10.2 Dettagli implementazione.

Per non creare interferenza tra le richieste e le risposte del client-server ho utilizzato 6 stream differenti.

Per il compratore è stata utilizzata la ***stream1*** per inviare le richieste e quindi a sua volta il server legge le richieste sulla medesima stream. Per inviare le risposte del server alle richieste è stata utilizzata la ***stream2***.

Per il fornitore è stata utilizzata la ***stream3*** per inviare le richieste e quindi a sua volta il server legge le richieste sulla medesima stream. Per inviare le risposte del server alle richieste è stata utilizzata la ***stream4***.

Per il trasportatore è stata utilizzata la ***stream5*** per inviare le richieste e quindi a sua volta il server legge le richieste sulla medesima stream. Per inviare le risposte del server alle richieste è stata utilizzata la ***stream6***.

Dato che alcune funzioni si sono rivelate utili per diverse categoria di utenti ho creato un ulteriore cartella chiamata "*shared-server*" al fine di contenere metodi e componenti (come l'utente generale) utili ai 3 diversi server. Ad esempio la classe ***utente*** (*super classe* delle 3 diverse categorie di utenti) è stata inserita in questa cartella e contiene variabili e metodi comuni per ogni categoria di utenti.

10.3 Utilizzo di PostgreSQL e Redis nel sistema.

Per interagire con **Redis** e con il database **PostgreSQL** nel sistema, sono presenti due cartelle aggiuntive contenenti file pertinenti.
Inoltre, per quanto riguarda l'**interazione specifica con il database PostgreSQL**, vi è un'altra cartella dedicata che include file per la *definizione dello schema delle tabelle* del database, i *parametri per il nome del database e l'username*, i *privilegi* assegnati all'utente specificato nei parametri, oltre a uno script in formato shell *(".sh")* che implementa le azioni delineate nei file sopra menzionati.

10.4 Tracciamento e registrazione errori e successi.

Per **tracciare gli errori e i successi** dei task o dei requisiti nel sistema è stato implementato un **sistema di logging** . Questo sistema opera in modo tale che, ad ogni metodo effettuato da un componente del sistema, che abbia successo o meno, viene registrato tramite una tabella del database chiamata *LogTable*. Questa tabella contiene le relative colonne:

- **timevalue**: utilizzata per indicare anno-mese-giorno-ora-minuto-secondo del log;
- **pid**: l'identificativo del processo in esecuzione;
- **statoLog**: un valore enumerativo che indica la categoria del log, che può essere *'INFO'*, *'WARNING'*, *'ERROR'*;
- **messaggio**: una stringa che fornisce una spiegazione di ciò che è accaduto;
- **sessionID**: l'identificativo della sessione del relativo utente che effettua la richiesta;
- **nomeRequisiti**: il nome del requisito specifico coinvolto;
- **statoRequisito**: un ulteriore valore enumerativo che indica se la richiesta/ il metodo effettuato è stato eseguito correttamente o ha presentato errori, e comprende i 3 valori *'SUCCESS'*, *'NOT SUCCESS'*, *'WAIT'*.

Questo sistema implementa il monitor richiesto e assolve efficacemente il compito di validare i requisiti del sistema.

11 Descrizione componenti del sistema.

11.1 Server per utente compratore.

Partiamo con la descrizione delle componenti del server relativo al compratore.
Quest'ultimo contiene diverse componenti.

11.1.1 Utente compratore.

La componente **Utente compratore** è una classe che rappresenta l'utente compratore, una categoria dell'utente del sistema. Questa componente eredita attributi e comportamenti della superclasse Utente. Oltre agli attributi della classe Utente , la classe Utente compratore ha i seguenti attributi : data di compleanno, via di residenza, numero civico, cap e città di residenza.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può effettuare la registrazione, effettuare il login e aggiornare la propria residenza.

Il metodo relativo alla **registrazione dell'utente** comprende come parametri di input *il nome dell'utente compratore, la categoria, il nome, il cognome, il sessionID* (generato dal server), *il numero di telefono, l'email, via di residenza, numero civico, cap, città di residenza, password, conferma password e data di compleanno.*
Prima di effettuare la registrazione vengono fatti dei controlli:

1. che il nome utente e l'email siano **univoci**.
2. che l'email **contenga** il carattere '@'.
3. che la passowrd **rispetti i criteri** decisi.
4. che il **valore della password sia uguale al valore della conferma della password**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo al **login dell'utente** comprende come parametri di input *il nome dell'utente compratore e la password.*
Il metodo login viene ereditato dalla super classe 'Utente'. In questo metodo viene aggiunto un semplice controllo:

1. che l'**utente si tratti della categoria propria**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'aggiornamento della residenza** comprende come parametri di input *il nome dell'utente compratore, la via di residenza, il numero civico, il cap e la città di residenza.*
Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.1.2 Carrello.

La componente **Carrello** è una classe che rappresenta il concetto di un carrello all'interno del sistema. Questo include attributi che identificano il nome dell'utente compratore che lo possiede, il codice del prodotto presente nel carrello e la relativa quantità.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può aggiungere e rimuovere prodotti dal carrello.

Il metodo relativo **all'operazione di aggiunta** è utilizzato appunto per aggiungere un prodotto al carrello dato *il nome dell'utente e il codice del prodotto*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che il **codice del prodotto esista** nel database.
4. che il **codice del prodotto esista o meno** già **nel carrello dell'utente**. Nel caso in cui è già presente, incrementiamo la quantità, altrimenti lo aggiungiamo.

Se i **controlli non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo alla **rimozione** è utilizzato per rimuovere un prodotto al carrello dato *il nome dell'utente e il codice del prodotto*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che il **codice del prodotto esista nel carrello dell'utente**. Nel caso in cui è presente viene rimosso, altrimente viene registrato un errore e il metodo viene concluso.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.1.3 Lista Desideri.

La componente **Lista Desideri** è una classe che rappresenta il concetto di una lista desideri all'interno del sistema. Questo include attributi che identificano il nome dell'utente compratore che la possiede, il codice del prodotto presente nel carrello e la relativa quantità.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può aggiungere e rimuovere prodotti dalla lista desideri.

Il metodo relativo **all'aggiunta** è utilizzato per aggiungere un prodotto alla lista desideri dato *il nome dell'utente e il codice del prodotto*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che il **codice del prodotto esista** nel database.
4. che il **codice del prodotto esista o meno** già **nella lista desideri dell'utente**. Nel caso in cui è già presente, incrementiamo la quantità, altrimenti lo aggiungiamo.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **alla rimozione** è utilizzato per rimuovere un prodotto dalla lista desideri dato *il nome dell'utente e il codice del prodotto*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che il **codice del prodotto esista nella lista desideri dell'utente**. Nel caso in cui è presente viene rimosso, altrimente viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.1.4 Carta di pagamento.

La componente **Carta** è una classe che rappresenta il concetto di una carta di pagamento all'interno del sistema. Questo include attributi che identificano il nome dell'utente compratore che la possiede, il numero della carta e il cvv: Card Security Card.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può aggiungere e rimuovere carte di pagamento.

Il metodo relativo **all'aggiunta** è utilizzato per aggiungere una carta di pagamento dato *il nome dell'utente, il numero della carta e il cvv*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.

Dopo questi controlli viene aggiunta la carta di pagamento e viene loggata la richiesta avvenuta con successo nella tabella LogTable del database.
Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **alla rimozione** è utilizzato per rimuovere una carta di pagamento dato *il nome dell'utente e l'id della carta*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che la **carta esista nel database**. Nel caso in cui sia presente , controlliamo se appartiene all'utente che vuole rimuoverla. Se è così possiamo rimuoverla, altrimenti viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.1.5 Ordine.

La componente **ordine** è una classe che rappresenta il concetto di un ordine, un acquisto di un prodotto da parte di un utente compratore. Questo include attributi che identificano il nome dell'utente compratore l'identificativo dell'ordine, il codice del prodotto , la data dell'ordine effettuato, lo stato dell'ordine (valore enumerativo che comprende i valori 'NonOrdinato', 'InElaborazione', 'Spedito', 'Annullato'), e altri attributi per l'indirizzo di spedizione.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può visionare gli ordin effettuati e annullare un ordine.

Il metodo relativo alla **visione degli ordini** effettuati comprende come parametri di input *il nome dell'utente compratore*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.

Dopo questi controlli vengono mostrati i vari ordini dell'utente compratore.
Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'annullamento di un ordine** comprende come parametri di input *il nome dell'utente e l'id dell'ordine*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che **l'utente che vuole annullare l'ordine sia uguale all'utente che ha effettuato l'ordine**. Se si tratta dello stesso utente si controlla anche che **l'ordine sia ancora in fase di elaborazione** e non sia stato spedito o consegnato. In quel caso l'ordine può essere annullato.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.1.6 Recensione.

La componente **recensione** è una classe che rappresenta il concetto di una recensione, un giudizio di un acquisto di un prodotto da parte di un utente compratore. Questo include attributi che identificano il nome dell'utente compratore, l'identificativo della recensione, l'identificativo dell'ordine , la descrizione della recensione, e il voto in stelle (valore enumerativo che comprende i valori da 1 a 5).
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può effettuare e rimuovere una recensione .

Il metodo relativo **all'effettuazione della recensione** comprende come parametri di input *il nome dell'utente compratore, l'id dell'ordine, la descrizione e il voto in stelle*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che l'**id dell'ordine esiste** nel database.
4. che **l'utente non abbia già effettuato una recensione per lo stesso ordine**.

5. che **lo stato della spedizione sia consegnato**. Solo in questo caso l'ordine può essere recensito dall'utente compratore.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **alla rimozione della recensione** comprende come parametri di input *il nome dell'utente e l'id della recensione*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che **l'utente che vuole rimuovere la recensione corrisponda all'utente che ha effettuato la recensione**.
4. che l'**id della recensione esista** nel database. Arrivati a questo punto, superati gli altri controlli, la recensione può essere rimossa e viene loggato l'evento nel database nella tabella LogTable.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.1.7 Reso.

La componente **reso** è una classe che rappresenta il concetto di un reso, una restituzione di un prodotto acquistato da parte di un utente compratore. Questo include attributi che identificano il nome dell'utente compratore, l'identificativo del reso, l'identificativo dell'ordine , la motivazione del reso (valore enumerativo che comprende i valori: 'NonReso', 'Difettoso', 'MisuraErrata', 'NonConformeAlleAspettative', 'CambioOpinione').
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può effettuare reso.

Il metodo relativo **all'effettuazione del reso** comprende come parametri di input *il nome dell'utente compratore, l'id dell'ordine e la motivazione del reso*. Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che **l'id dell'ordine esiste** nel database.
4. che **l'utente non abbia già effettuato un reso per lo stesso ordine**.
5. che **lo stato della spedizione sia consegnato**. Solo in questo caso l'ordine può essere reso dall'utente compratore.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.2 Server per utente fornitore.

Contiene diverse componenti.

11.2.1 Utente fornitore.

La componente **Utente fornitore** è una classe che rappresenta l'utente fornitore, una categoria dell'utente del sistema. Questa componente eredita attributi e comportamenti della superclasse Utente. Oltre agli attributi della classe Utente , la classe Utente fornitore ha anche il nome dell'azienda di produzione come attributo
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente fornitore può effettuare la registrazione, effettuare il login e aggiornare il nome dell'azienda di produzione.

Il metodo relativo **alla registrazione dell'utente** comprende come parametri di input *il nome dell'utente fornitore, la categoria, il nome, il cognome, il sessionID* (generato dal server), *il numero di telefono, l'email, password, conferma password e azienda di produzione.*
Prima di effettuare la registrazione vengono fatti dei controlli:

1. che il nome utente e l'email siano **univoci**.
2. che l'email **contenga** il carattere '@'.
3. che la passowrd **rispetti i criteri decisi**.
4. che il **valore della password sia uguale al valore della conferma della password**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **al login dell'utente** comprende come parametri di input *il nome dell'utente fornitore e la password.*
Il metodo login viene ereditato dalla super classe 'Utente'. In questo metodo viene aggiunto un semplice controllo:

1. che l'utente si tratti della **categoria propria**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'aggiornamento del nome dell'azienda di produzione** comprende come parametri di input *il nome dell'utente compratore e il nuovo nome dell'azienda di produzione.*
Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteFornitore**'.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.3 Server per utente trasportatore.

Contiene diverse componenti.

11.3.1 Utente trasportatore.

La componente **Utente trasportatore** è una classe che rappresenta l'utente fornitore, una categoria dell'utente del sistema. Questa componente eredita attributi e comportamenti della superclasse Utente. Oltre agli attributi della classe Utente , la classe Utente fornitore ha anche il nome della ditta di spedizione

come attributo.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente fornitore può effettuare la registrazione, effettuare il login e aggiornare il nome della ditta di spedizione.

Il metodo relativo **alla registrazione dell'utente** comprende come parametri di input *il nome dell'utente trasportatore, la categoria, il nome, il cognome, il sessionID* (generato dal server), *il numero di telefono, l'email, password, conferma password e la ditta di spedizione.*
Prima di effettuare la registrazione vengono fatti dei controlli:

1. che il nome utente e l'email siano **univoci**.
2. che l'email contenga il **carattere** '@'.
3. che la passowrd **rispetti i criteri decisi**.
4. che il **valore della password sia uguale al valore della conferma della password**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **al login dell'utente** comprende come parametri di input *il nome dell'utente trasportatore e la password.*
Il metodo login viene ereditato dalla super classe 'Utente'. In questo metodo viene aggiunto un semplice controllo:

1. che **l'utente si tratti della categoria propria**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'aggiornamento del nome della ditta di spedizione** comprende come parametri di input *il nome dell'utente compratore e il nuovo nome della ditta di spedizione.*
Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteTrasportatore'**.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.3.2 Spedizione.

La componente ***spedizione*** è una classe cha rappresenta il concetto di una spedizione, un invio di un prodotto acquistato trasportato da un utente di categoria trasportatore.
Questo include attributi che identificano il nome dell'utente trasportatore, l'id della spedizione, l'id dell'ordine effettuato dall'utente compratore, lo stato della spedizione e la ditta di spedizione dell'utente trasportatore che consegna il pacco.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente trasportatore può prendere in carico la spedizione e può avvisare il sistema che la spedizione è stata consegnata.

Il metodo relativo al **carico della spedizione** comprende come parametri di input *il nome dell'utente trasportatore.* Nel metodo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.

2. che l'utente sia della **categoria 'UtenteTrasportatore'**.

3. che l'utente abbia il **valore dell'attributo disponibilità uguale a 0**. Questo perchè per semplicità l'utente trasportatore può prendere in carico una spedizione alla volta. Se il valore di 'disponibilità' è 0 si controlla **se ci sono ordini in elaborazione**. In questo caso viene fatto scegliere randomicamente a un utente trasportatore un ordine da prendere in carico e viene avviata la spedizione.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'avviso della spedizione consegnata** comprende come parametri di input *il nome dell'utente trasportatore e l'id della spedizione*. Nel metodo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.

2. che l'utente sia della **categoria 'UtenteTrasportatore'**.

3. che **l'id della spedizione esista**.

4. che **l'utente trasportatore che vuole avvisare che la spedizione è consegnata corrisponde all'utente che ha preso in carico la spedizione**. Solo in questo caso si può avvisare che la spedizione è stata consegnata.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.4 Componenti comuni per i server.

Come già enunciato nel paragrafo 5, alcune componenti e funzioni si sono rivelate utili per tutti e 3 i server, perciò sono state inseriti in una cartella differente. Descriviamo questi componenti:

11.4.1 Utente.

La componente **Utente** è una classe che rappresenta l'utente nella sua generalità. Questa componente definisce degli attributi, costruttori e metodi che verranno ereditati dalle 3 categorie di utenti sottoclasse. Gli attributi definiti nella classe Utente sono il nome utente, il sessionID, la categoria, nome, cognome, numero di telefono, password ed email.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può effettuare il login, il logout, l'eliminazione del profilo, l'aggiornamento del numero di telefono e della password.

Il metodo relativo **al login** comprende come parametri di input *il nome dell'utente compratore e la password*. Nel metodo vengono fatti dei controlli:

1. che l'utente sia **già stato loggato o meno** tramite il valore della colonna stato del database. Se lo stato è già attivo l'utente sta effettuando un nuovo login.

2. che la **password inserita corrisponda alla password salvata nel database** durante la registrazione dell'utente. Se corrisponde, allora l'utente viene loggato tramite l'aggiornamento del sessionID e il cambiamento dello stato se l'utente non è già loggato.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **al logout dell'utente** comprende come parametri di input *il nome dell'utente compratore*.
Nel metodo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente **sia già disconnesso**. Se così non fosse l'utente viene disconnesso tramite un aggiornamento del sessionID e dello stato.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'eliminazione del profilo** comprende come parametri di input il nome dell'utente compratore.
Nel metodo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database. In questo caso può essere eliminato il profilo.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'aggiornamento del numero di telefono** comprende come parametri di input *il nome dell'utente compratore e il nuovo numero di telefono*. Nel metodo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database. In questo caso può essere aggiornato il numero di telefono.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'aggiornamento della password** comprende come parametri di input *il nome dell'utente compratore, la vecchia password e la nuova*. Nel metodo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che la **vecchia password inserita dall'utente corrisponda a quella salvata nel database** durante la registrazione o durante l'ultima modifica.
3. che **la nuova password da inserire rispetti i criteri definiti per il sistema**. Solo in questo caso la password può essere aggiornata.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.4.2 Prodotto.

La classe **Prodotto** incarna il concetto di un articolo in modo completo. Questa entità risulta essenziale sia per l'utente acquirente, che può procedere con l'acquisto o effettuare ricerche specifiche, sia per l'utente fornitore, che può aggiungere nuovi prodotti al sito o eliminarne alcuni.
Questa classe include attributi che identificano il codice del prodotto, il nome, la categoria, il prezzo in euro, la descrizione, il nome dell'azienda di produzione e il numero di copie disponibili.
Sono definiti 2 costruttori che inizializzano i membri classi con valori nulli e valori predefiniti.

L'utente compratore può ricercare un prodotto oppure acquistarlo.

Il metodo relativo **alla ricerca del prodotto** comprende come parametri di input *il nome dell'utente compratore e il nome del prodotto.* Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.

Dopo questi controlli vengono mostrati i vari ordini dell'utente compratore.
Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **all'aquisto di un prodotto** comprende come parametri di input *il nome dell'utente , il codice del prodotto e parametri per l'indirizzo di spedizione.* Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteCompratore'**.
3. che il **prodotto esista** nel database. Solo in questo caso il prodotto può essere acquistato.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.
Questa operazione implica il fatto che il numero di copie disponibili del prodotto acquistato venga diminuito di 1.

L'utente fornitore può inserire o rimuovere prodotti nel sito.

Il metodo relativo **all'aggiunta di un prodotto** comprende come parametri di input *il nome dell'utente fornitore , il nome , la categoria e il prezzo del prodotto.* Prima di fare questo vengono fatti dei controlli:

1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteFornitore'**.
3. che il **prodotto esista** già nel database. Se esiste incrementiamo il numero di copie disponibili, altrimenti lo inseriamo per la prima volta.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

Il metodo relativo **alla rimozione di un prodotto** comprende come parametri di input il nome dell'utente fornitore e il codice del prodotto. Prima di fare questo vengono fatti dei controlli:

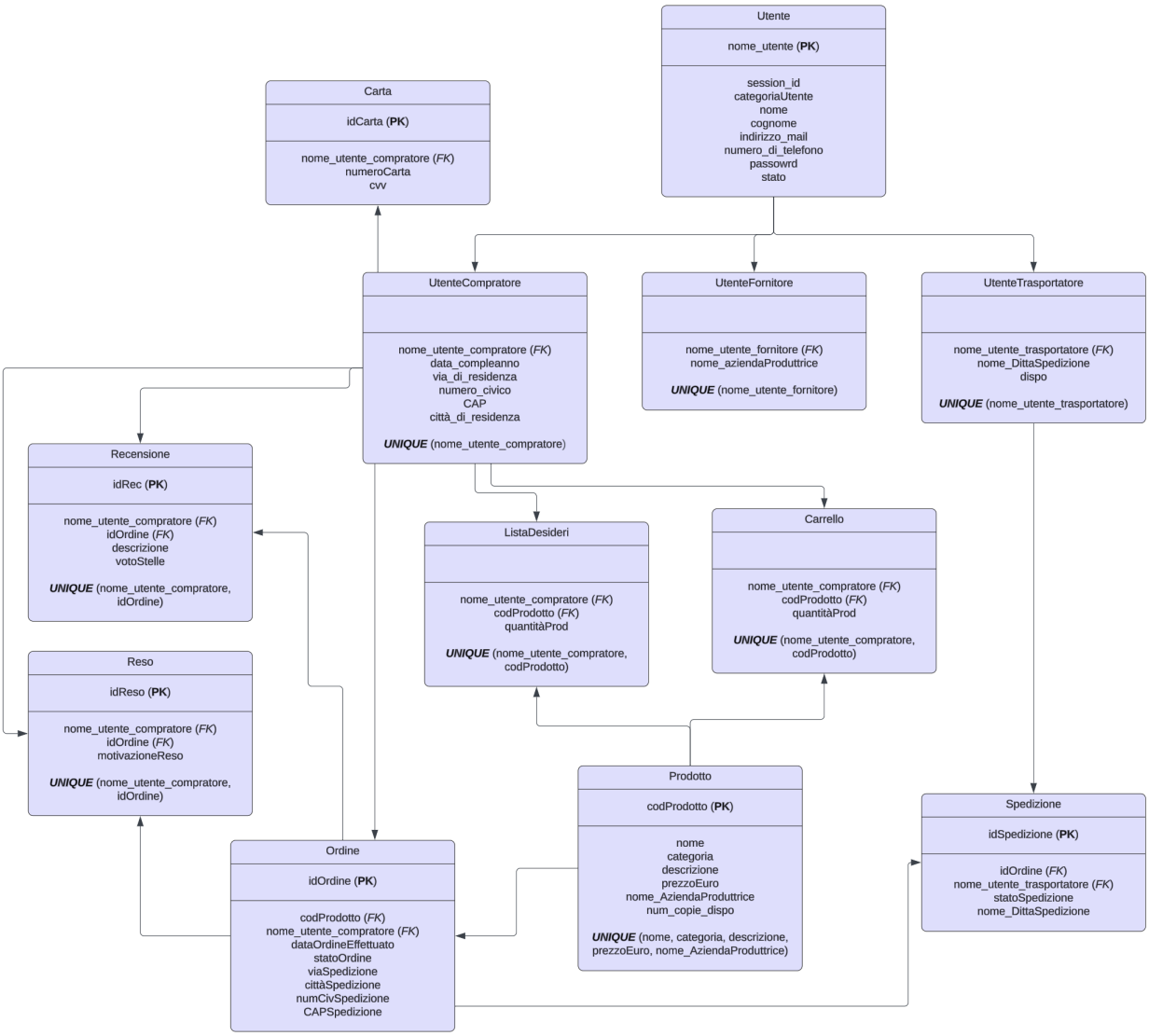
1. che l'utente **esista** e abbia una **sessione attiva** tramite una query del valore della sessionID presente nella tabella 'Utente' del database.
2. che l'utente sia della **categoria 'UtenteFornitore'**.
3. che il **prodotto da rimuovere esista.** Solo in questo caso il prodotto può essere rimosso.

Se i controlli **non vanno a buon fine** viene registrato un errore nella tabella LogTable del database e il metodo viene concluso.

11.5 Client per le categorie.

I client non contengono componenti specifici, ma vi è solo l'utilizzo di alcune funzioni e del main per inviare le richieste come descritto nella sezione 5.

12 Schema del DB.

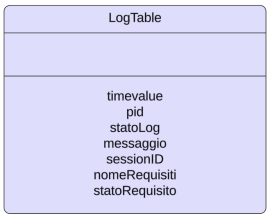


Sono stati definiti dei tipi di dati utili per specificare il tipo di informazione che può essere memorizzato in una determinata colonna di una tabella. Tra questi troviamo:

- **motivazioneReso** che può assumere i valori ('difettoso', 'misura errata', 'non conforme alle aspettative', 'cambio opinione').
- **votoStelle**, utilizzata per dare un voto in stelle a un prodotto acquistato. ('1', '2', '3', '4', '5').
- **statoOrdine** che può assumere i vari stati: ('in elaborazione', 'spedito', 'annullato').

- **statoSpedizione** che può assumere i vari stati: ('in transito', 'consegnato').
- **statoLog** delle 3 diverse categorie: ('INFO', 'WARNING', 'ERROR').
- **statoRequisito** che assume i 3 diversi stati :('SUCCESS', 'NOT SUCCESS', 'WAIT'). Utile per verificare se un requisito è stato soddisfatto o meno.

Inoltre, vi è una tabella a parte definita per il **log delle richieste del client**. Viene utilizzata per **registrare errori, warning e log di successo**.



13 Funzioni del sistema.

Sono presenti **funzioni** per interagire e usare Redis e postgreSQL nelle cartelle 'con2db' e 'con2redis'.
Oltre a queste ci sono altre funzioni:

- La '***micro-sleep***' che è stata utilizzata parecchio, interrompe l'esecuzione del programma per un numero di microsecondi definito dall'utente.
- La funzione '***isSpecialCharacter***' definita nel file '*isSpecialCharacter.h*' verifica se un carattere è uno speciale o no. Utilizza la funzione std::isalnum per determinare se il carattere è alfanumerico o meno. Se il carattere non è alfanumerico, la funzione restituirà true, altrimenti restituirà false.
- La funzione '***checkSessionID***' definita nel file '*checkSessionID.h*' verifica se un sessionID generato dal server sia univoco oppure no. È molto importante poiché, quando un utente si registra oppure effettua un login, il server genera un sessionID e lo assegna all'utente per tenere traccia di tutti i dati associati alla sessione corrente di un utente. Questo sessionID deve essere perciò univoco.
- La funzione '***generateSessionID***' definita nel file '*generateSessionID.h*' si occupa di generare un identificatore di sessione casuale utilizzando una serie di caratteri validi specificati e un generatore di numeri casuali.
- La funzione '***InsertToLogDB***' definita nel file '*log2db.h*' si occupa di inserire record di log nel database nella tabella LogTable.
- La funzione '***getCurrentDateAsString***' viene utilizzata per ottenere la data corrente come stringa nel formato "*GG-MM-AAAA*".

14 Risultati simulazione del sistema.

Come abbiamo già spiegato per consentire una **verifica più veloce dei test**, ad ogni esecuzione del programma, per ogni categoria di utente, viene creato per il server e per il client un file all'interno di una cartella "*result*" contenente **tutte le richieste relative al client e tutte le risposte del corrispettivo server**.

Grazie a questi file è possibile verificare, soprattutto per i test pianificati controllari se sono andati a buon fine oppure no.

Ad esempio , io ho definito alcuni file di test in maniera sistematica. (da mantenere l'ordine)

1. Il client compratore/fornitore/trasportatore effettua delle **richieste relative alla gestione del profilo** (registrazione , login, logout, eliminazione profilo ed aggiornamenti informazioni personali).
2. (a) Il client fornitore si occupa di **gestire i prodotti nel sito**. (aggiunge prodotti, li rimuove, ...)
   (b) Il client compratore **effettua delle richieste relative alla gestione degli acquisti** (ricerca, acquisti, ...)
   (c) Il client trasportatore si occupa di **gestire la spedizione di alcuni ordini** (presa in carico e avviso della spedizione effettuata)
3. Il client si occupa di alcune **richieste post acquisto**. (recensione, reso, annullamento).

In questa maniera mi sono occupato di generare dei *test piuttosto reali e verificabili.*

Oltre al controllo delle richieste e delle risposte del server, ho **formattato i vari output sul terminale**. (già possibile con la funzione dump definita nella cartella 'con2redis').