

Backend TecnoMarket

Progetto di Ingegneria del Software

Mattia Di Marco, Simone La Bella, Simone Zaccagnino

Marzo 2024



Indice

1	Introduzione	3
2	Descrizione generale	3
2.1	Interazione con gli utenti	3
2.2	Accesso alla piattaforma	3
2.3	Acquisto, vendita e trasporto	4
3	Requisiti	4
3.1	Requisiti sui dati	4
3.2	User Requirements	6
3.2.1	Use Case UML	8
3.3	System Requirements	8
3.3.1	Architettura del Sistema	10
3.3.2	Activity Diagram UML	10
3.3.3	State Diagram UML	11
3.3.4	Message Sequence Chart UML	11
3.4	Requisiti non-funzionali	12
4	Implementazione	13
4.1	Componenti del Sistema	13
4.2	Database	15
4.3	Connessioni Redis	17
5	Risultati sperimentali	20

1 Introduzione

TecnoMarket è una piattaforma per l'e-commerce, in cui gli utenti, divisi in customers, fornitori e trasportatori, possono comprare, vendere e gestire le spedizioni dei vari prodotti.

2 Descrizione generale

In questa sezione andremo a fare una descrizione dettagliata del sistema e delle sue funzionalità principali.

2.1 Interazione con gli utenti

TecnoMarket è pensata per essere una web app, indirizzata alla compravendita di prodotti. La piattaforma dà la possibilità di mettere in vendita oggetti, di ricercarli e acquistarli, ed inoltre permette ai trasportatori di gestire al meglio la spedizione. È possibile per i:

- Customer:
 - Ricercare e acquistare oggetti, e ricevere aggiornamenti sullo stato della spedizione;
- Fornitori:
 - Mettere in vendita prodotti in inventari appositi ed eventualmente aumentarne la quantità;
- Trasportatori:
 - Visualizzare le proprie spedizioni e aggiornare lo stato delle stesse.

2.2 Accesso alla piattaforma

Al primo accesso alla piattaforma viene richiesta la registrazione, con informazioni base quali:

- Nome;
- Cognome;
- Email;
- Password;
- Metodi di pagamento (vedere dopo per specifiche informazioni);
- Il proprio ruolo fra:
 - Customer;

- Fornitore;
- Trasportatore.

Effettuata la registrazione sarà possibile accedere con un semplice login inserendo email e password del proprio account.

2.3 Acquisto, vendita e trasporto

La piattaforma permette ad un utente con account fornitore di inserire nel proprio catalogo una quantità N di oggetti, specificando il nome, e il prezzo per unità e lo stock che si ha disponibile. Questi oggetti potrebbero essere presenti o meno nel database del sito, se non dovesse esserci una scheda tecnica per l'oggetto in questione, la piattaforma la crea automaticamente chiedendo al fornitore di inserire il codice a barre dell'oggetto.

Permette agli utenti con account di tipo customer, di acquistare una quantità N di diversi oggetti, che possono essere anche di diversi fornitori, pagando con il metodo che preferiscono, e sarà poi il sito a gestire la divisione dell'importo.

Agli utenti con account trasportatore è possibile visualizzare la lista degli ordini a proprio carico, ed aggiornare lo stato degli stessi.

3 Requisiti

3.1 Requisiti sui dati

1. Degli **utenti registrati**, che siano fornitori, trasportatori o customer, si vuole mantenere:
 - 1.1 Nome
 - 1.2 Cognome
 - 1.3 Email
 - 1.4 Password (HASH)
 - 1.5 ID
 - 1.6 I metodi di pagamento della quale dispone. Dei **metodi di pagamento** ci interessa:
 - 1.6.1 ID
 - 1.6.2 Nome

- 1.7 Un utente può essere:
 - 1.7.1 **Customer**, ovvero gli utenti del sito che hanno la capacità di acquistare
 - 1.7.2 **Fornitore**, ovvero i venditori, di questi ci interessa sapere:
 - 1.7.2.1 Partita Iva
 - 1.7.3 **Trasportatore**, ovvero i corrieri che si occupano di gestire la spedizione, di questi interessa sapere:
 - 1.7.3.1 Azienda (intesa come ragione sociale)
2. Come **Oggetto** intendiamo la scheda tecnica di un prodotto. Di un Oggetto ci interessa:
 - 2.1 ID
 - 2.2 Nome
 - 2.3 Descrizione
 - 2.4 Codice a Barre
 - 2.5 Ad ogni oggetto sarà assegnata una **categoria**. Di ogni categoria ci interessa:
 - 2.5.1 ID
 - 2.5.2 Nome
3. Degli **Acquisti** ci interessa mantenere:
 - 3.1 ID
 - 3.2 Istante di Acquisto
 - 3.3 Stato dell'ordine, che può essere:
 - In preparazione
 - Spedito
 - Consegnato
 - 3.4 Il trasportatore che la gestisce
 - 3.5 Un insieme di Oggetti, della quale ci interessa sapere:
 - 3.5.1 Quantità di ogni singolo Oggetto nell'ordine
 - 3.6 Customer
4. La **Inventario** di un oggetto da parte di un fornitore:
 - 4.1 ID
 - 4.2 Stock iniziale
 - 4.3 Stock attuale
 - 4.4 Prezzo per unità
 - 4.5 Istante di inserimento
 - 4.6 Oggetto di vendita
 - 4.7 Fornitore

3.2 User Requirements

1. U-Req-GestisciAccount:

1.1 U-Req-Registrazione:

- Priorità: primaria;
- Descrizione: permette ad un nuovo utente di registrarsi, inserendo le proprie informazioni;

1.2 U-Req-Login:

- Priorità: primaria;
- Descrizione: permette ad un utente registrato di accedere al suo account;

1.3 U-Req-Logout:

- Priorità: primaria;
- Descrizione: permette ad un utente che ha effettuato il login di uscire dal suo account;

2. U-Req-EffettuaPagamento:

- Priorità: primaria;
- Descrizione: permette di effettuare un pagamento tra utenti;

3. U-Req-GestisciCompravendite

3.1 U-Req-AggiungiVendita:

- Priorità: primaria;
- Descrizione: permette ad un fornitore di aggiungere una vendita di N oggetti;

3.2 U-Req-ModificaDatiVendita:

- Priorità: primaria;
- Descrizione: permette ad un fornitore di modificare i dati di una vendita da lui inserita;

3.3 U-Req-Acquisto:

- Priorità: primaria;
- Descrizione: permette ad un costumer di acquistare N oggetti da un fornitore;

3.4 U-Req-AggiornaSpedizione:

- Priorità: primaria;
- Descrizione: permette ad un trasportatore di comunicare lo stato di una spedizione in corso;

4. U-Req-CreaOggetto:

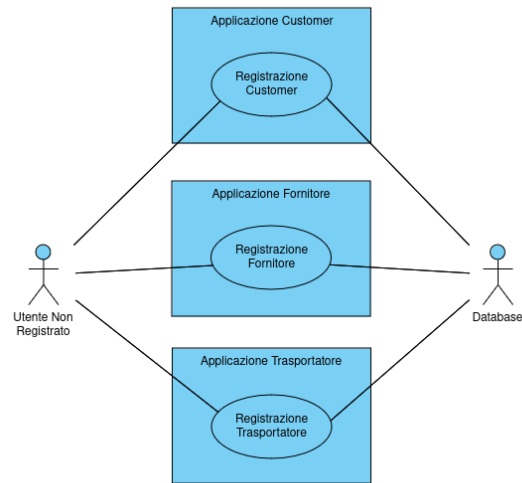
- Priorità: primaria;
- Descrizione: permette al sistema di creare una nuova scheda tecnica per l'oggetto chiedendo al fornitore le varie informazioni (codice a barre, nome, ecc).

5. U-Req-GestisciRicerche:

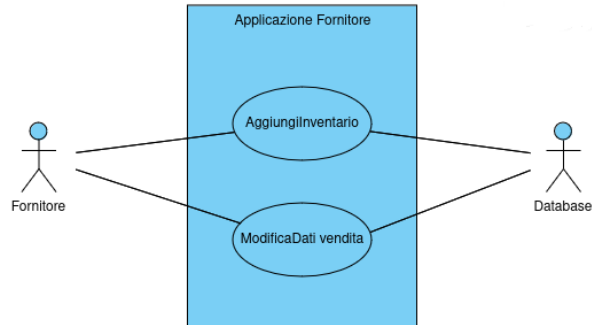
5.1 U-Req-RicercaOggetto:

- Priorità: primaria
- Descrizione: permette ad un costumer di cercare un oggetto;

3.2.1 Use Case UML



(a) *UseCase Registrazione*



(b) *UseCase Fornitori*

3.3 System Requirements

1. S-Req-Registrazione:

- 1.1 Alla fine della registrazione il sistema salva le informazioni nel database per creare l'account;
- 1.2 Se l'email è già in uso rifiuta la registrazione.

- 2. S-Req-Login:
 - 2.1 Quando viene effettuato un login il sistema controlla che le informazioni siano già presenti nel database.
 - 2.2 Se le informazioni non sono presenti chiede di effettuare la registrazione.
- 3. S-Req-AggiungiInventario:
 - 3.1 Quando un fornitore aggiunge una quantità N di prodotti salva le informazioni nel database;
 - 3.2 Se l'oggetto messo in vendita non ha una scheda tecnica chiede il codice a barre per aggiungerla al database.
- 4. S-Req-ModificaDatiInventario:
 - 4.1 Quando i dati di una vendita vengono modificati salva le nuove informazioni nel database.
- 5. S-Req-Acquisto:
 - 5.1 Quando un customer effettua un acquisto salva le informazioni nel database;
 - 5.2 Aggiunge l'acquisto all'insieme degli ordini effettuati dal customer;
 - 5.3 Aggiunge l'acquisto all'insieme delle vendite del fornitore;
 - 5.4 Aggiunge l'acquisto all'insieme delle spedizioni del trasportatore.
- 6. S-Req-AggiornaSpedizione:
 - 6.1 Quando viene assegnato ad un trasportatore un ordine, si occupa di gestire la spedizione, aggiornando lo stato della stessa;
 - 6.2 Quando si verifica un cambio di stato inerente ad una spedizione, questa viene salvata nel database;
- 7. S-Req-RicercaOggetto:
 - 7.1 Quando un customer cerca un oggetto tramite il nome, il sistema gli mostra tutti gli oggetti relativi a quel nome con venditore ed informazioni annesse.

3.3.1 Architettura del Sistema

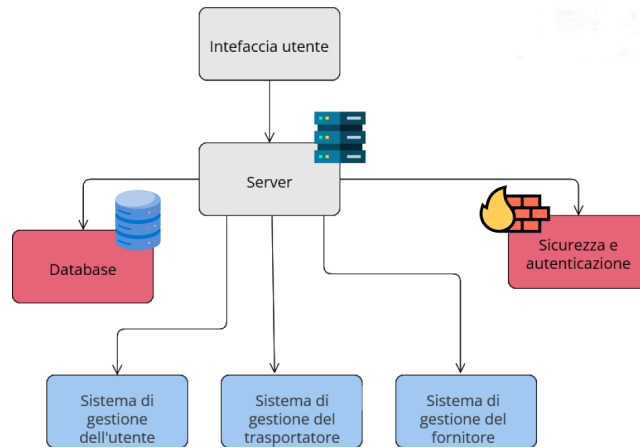


Figura 1: Architettura del Sistema

3.3.2 Activity Diagram UML

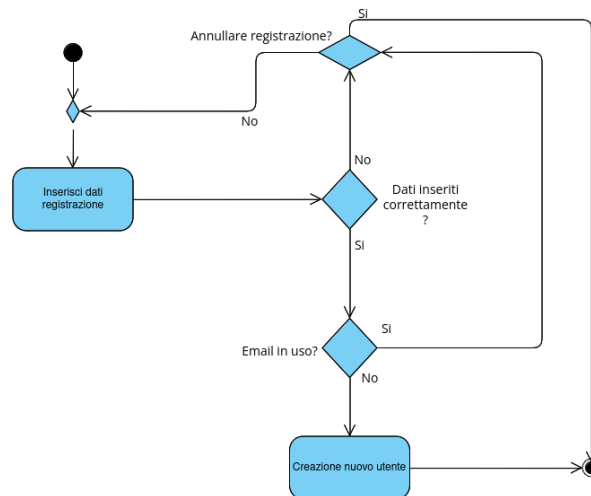


Figura 2: AD-UML dell'operazione di Registrazione

3.3.3 State Diagram UML

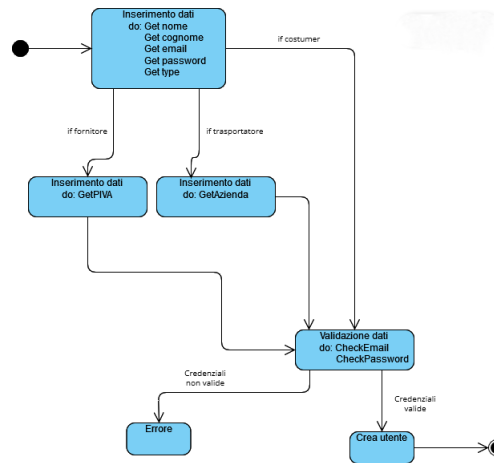


Figura 3: SD-UML dell'operazione di Registrazione

3.3.4 Message Sequence Chart UML

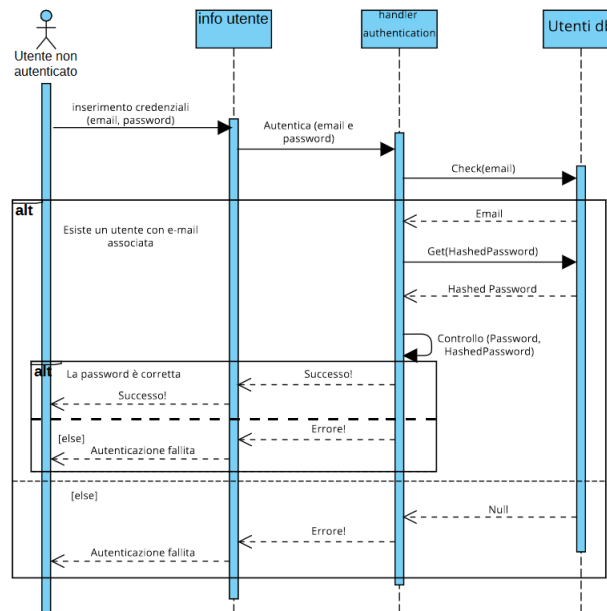


Figura 4: MSC-UML dell'operazione di Login

3.4 Requisiti non-funzionali

1. Req-NF-TempoDiRisposta:

- Priorità: primario;
- Descrizione: la piattaforma deve effettuare le richieste degli utenti in tempi ragionevoli (pochi secondi al massimo), nel caso in cui questo non fosse momentaneamente possibile, è necessario comunicarlo all'utente precisando l'errore;

2. Req-NF-Privacy:

- Priorità: primario;
- Descrizione: la piattaforma dovrà garantire la riservatezza di dati sensibili degli utenti;

3. Req-NF-Backup:

- Priorità: Primario;
- Descrizione: la piattaforma deve effettuare backup dei contenuti caricati dagli utenti (insieme ai loro dati), in modo da garantire la disponibilità e la persistenza dei dati;

4. Req-NF-Compatibilità:

- Priorità: Secondaria;
- Descrizione: la piattaforma dovrà essere compatibile su dispositivi Android e iOS (per quanto riguarda l'app), e sui maggiori browser: Firefox, Chrome, Safari (per quanto riguarda la web app);

4 Implementazione

4.1 Componenti del Sistema

È stato sviluppato e testato un backend per un e-commerce, analizzando nello specifico, troviamo 3 componenti, ovvero gli attori del nostro sistema:

1. **Utente:** Rappresenta un utente ancora non autenticato, a prescindere dal ruolo che andrà a ricoprire. È rappresentata da una classe, il cui costruttore comprende tutti i dati relativi alla creazione di un utente, sono presenti le funzioni Get relative ad ogni dato ed inoltre sono presenti le funzioni possibili per un utente:
 - 1.1 *Registrazione*, che varia in base al tipo di ruolo che andrà a ricoprire. Si occupa di inserire tramite una query nel db i dati relativi, lavorando la password con un hash in modo da garantire sicurezza;
 - 1.2 *Login nel server di riferimento*, effettuando i relativi controlli sui dati, ed in particolare si effettua hash tramite il salt inserito nel database in modo da effettuare un controllo più sicuro ed efficiente possibile.
2. **Customer:** Rappresenta uno user reale del sistema. È rappresentata da una sottoclasse di utente, ereditando quindi i relativi metodi. Inoltre può effettuare 2 azioni principali:
 - 2.1 *Ricerca*, che tramite una query nel database, che prende in ingresso una stringa rappresentante il nome di oggetto e un intero rappresentante la quantità che si vuole acquistare, restituendo una lista del prodotto cercato con tutti i relativi fornitori che si occupano della vendita del prodotto nella quantità desiderata;
 - 2.2 *Acquisto*, una volta cercato il prodotto di interesse per un utente, la funzione rende possibile l'acquisto del prodotto nella quantità desiderata, effettuando una query di inserimento dei valori nella tabella che si occupa della gestione degli acquisti e portando così il sistema a diminuire la quantità disponibile per quel prodotto.
3. **Trasportatore:** Rappresenta uno trasportatore reale del sistema. È rappresentata da una sottoclasse di utente, ereditando quindi i relativi metodi. Inoltre può effettuare 3 azioni principali:
 - 3.1 *getOrders*, tramite una query nel database, ritorna tutti gli ordini relativi ad un trasportatore;
 - 3.2 *getStatus*, tramite una query nel database, ritorna lo stato corrente di un ordine. Gli stati possibili sono:
 - in preparazione;
 - spedito;
 - consegnato.

- 3.3 *updateStatus*, tramite una query nel database, si occupa di aggiornare lo stato corrente in quello successivo. In caso in cui l'ordine si trovi già nello stato 'consegnato', la funzione non farà nulla.
4. **Fornitore:** Rappresenta un venditore reale nel sistema. È rappresentata da una sottoclasse di utente, ereditando quindi i relativi metodi. Inoltre può effettuare 2 azioni principali:
- 4.1 *addInventario*, si occupa di aggiungere un oggetto nell'inventario. Se l'oggetto non esiste nel database, si occupa prima di aggiungerlo, inserendo tutti i dati relativi allo stesso, creando la relativa scheda tecnica. Se invece esiste, si occupa di inserire l'oggetto nell'inventario del fornitore che la chiama, inserendo il prezzo e la quantità iniziale di vendita;
- 4.2 *addQuantity*, si occupa di aggiornare la quantità disponibile per un determinato prodotto, relativo al fornitore che la chiama. La quantità si aggiungerà a quella disponibile nel momento della chiamata.
5. **Oggetto:** È la classe che si occupa di rappresentare un oggetto, inteso come scheda tecnica, sono presenti i metodi get per dati relativi allo stesso come ad esempio il codice a barre, descrizione e categoria. Ha una funzione principale:
- 5.1 *addOggetto*, si occupa di aggiungere un oggetto nel database, prendendo tutti i dati utili alla creazione. Ovviamente non è possibile creare due volte lo stesso oggetto, essendo il codice a barre unico fra i vari oggetti.

4.2 Database

Per quanto riguarda il database l'abbiamo strutturato a partire dal seguente ER:

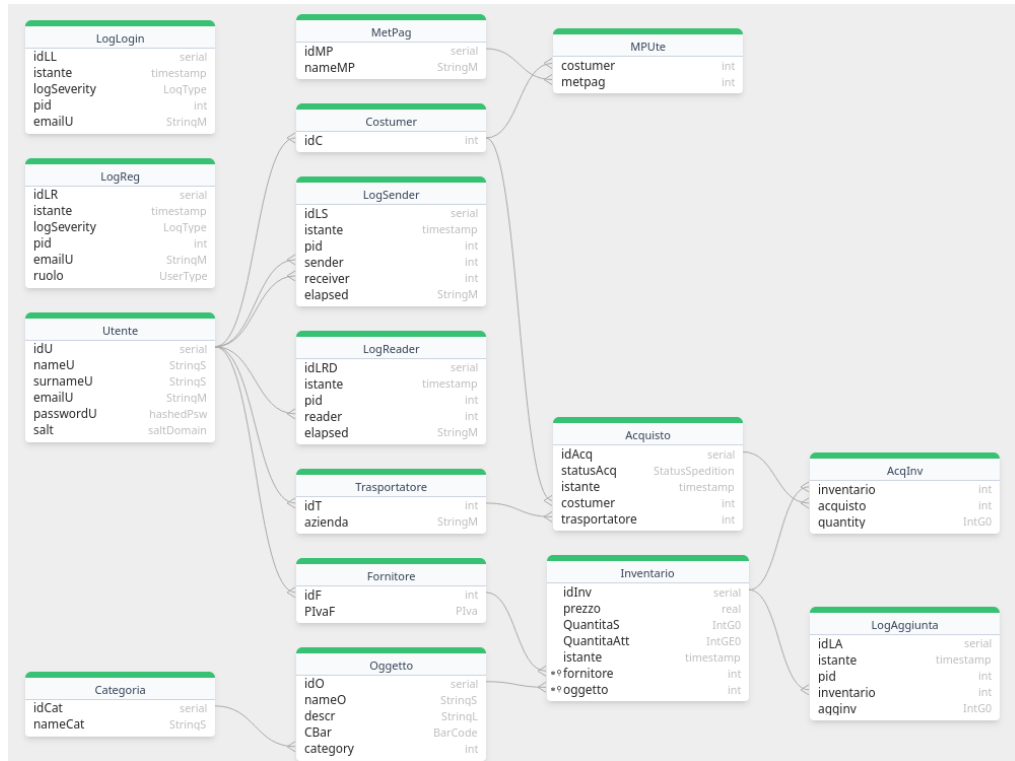


Figura 5: Diagramma ER del Database

1. **Tabella Utente**: rappresenta l'utente che usa il sito, in esso vengono conservate id, nome, cognome, email, password lavorata tramite hash e il salt che viene usato per l'hashing della password;
2. **Tabella Customer**: rappresenta il customer, ovvero l'utente che acquista sul sito e descrive una IsA di utente;
3. **Tabella Fornitore**: rappresenta il fornitore, ovvero l'utente che mette in vendita gli articoli, descrive una IsA di utente a cui aggiungiamo anche l'informazione relativa alla partita IVA;
4. **Tabella Trasportatore**: rappresenta il trasportatore, ovvero l'utente che effettua le spedizioni, descrive una IsA di utente a cui aggiungiamo anche l'informazione relativa all'azienda per cui lavora;

5. **Tabella MetPag:** rappresenta un metodo di pagamento, con id e nome del metodo in quale è univoco univoco;
6. **Tabella MPUte:** rappresenta la relazione che unisce un customer ad un metodo di pagamento, abbiamo infatti l'attributo customer che si riferisce l'id del customer e metpag che si riferisce all'id del metodo di pagamento. Questo permette ad un customer di avere più metodi di pagamento disponibili;
7. **Tabella Categoria:** rappresenta la categoria a cui appartiene un oggetto, con relativo id e nome univoco della categoria;
8. **Tabella Oggetto:** rappresenta la scheda tecnica di un oggetto, con id, nome, descrizione, codice a barre e categoria, quest'ultima un intero che si riferisce all'id della relativa categoria nella sua tabella;
9. **Tabella Inventario:** rappresenta l'inventario di un fornitore per un oggetto, ovvero la quantità di un oggetto messo in vendita da un fornitore ad un determinato prezzo. E' rappresentato da id, prezzo, quantità iniziale, quantità attuale, istante di inserimento, fornitore, un intero che si riferisce all'id del fornitore che lo vende, e oggetto, un intero che si riferisce all'id dell'oggetto messo in vendita;
10. **Tabella Acquisto:** rappresenta l'acquisto effettuato da un customer, al suo interno sono presenti id, stato della spedizione, istante di acquisto, customer, un intero che si riferisce all'id del customer, e trasportatore, un intero che si riferisce all'id del trasportatore;
11. **Tabella AcqInv:** rappresenta la relazione che lega un inventario al relativo acquisto da parte di un customer, si compone di inventario, un intero che si riferisce all'id dell'inventario, acquisto, un intero che si riferisce all'id dell'acquisto e quantità che rappresenta la quantità acquistata.

Sono inoltre presenti 5 tabelle necessarie a conservare i dati relativi ai log per varie funzionalità:

1. **Tabella LogReg:** mantiene i log relativi alle registrazioni. Troviamo id, istante in cui è avvenuta la registrazione, l'esito della registrazione ('*info*' se la registrazione è andata a buon fine, '*error*' se è stata inserita una mail già in uso), il pid del processo e l'email usata per la registrazione;
2. **Tabella LogLogin:** mantiene i log relativi ai login. Troviamo id, istante in cui è avvenuto il login, l'esito del login ('*info*' se il login è andato a buon fine, '*warn*' se la password inserita è errata, '*error*' se l'email inserita non è presente nel database), il pid del processo e l'email usata nel login;
3. **Tabella LogAggiunta:** mantiene i log relativi all'aggiunta di una quantità in un inventario da parte di un fornitore. Troviamo id, istante di aggiunta, pid del processo, un intero che si riferisce all'id dell'inventario in cui è avvenuta l'aggiunta e la quantità aggiunta dal fornitore;

4. **Tabella LogSender:** mantiene i log relativi al tempo necessario a inviare un messaggio tramite Redis. Troviamo l'id, istante di registrazione del log, pid del processo, due interi che si riferiscono rispettivamente all'id dell'utente che manda il messaggio e l'id dell'utente che lo riceve, e il tempo necessario all'invio;
5. **Tabella LogReader:** mantiene i log relativi al tempo necessario a leggere un messaggio tramite Redis. Troviamo l'id, istante di registrazione del log, pid del processo, un intero che si riferisce all'id dell'utente che legge il messaggio, e il tempo necessario alla lettura.

4.3 Connessioni Redis

Quando vengono avviati i 3 modelli disponibili (fornitore, customer, trasportatore), vengono in automatico create 3 connessioni

1. **customer2fornitore**, utilizzata dal customer per comunicare al fornitore la creazione di un nuovo ordine. Il trasportatore è in ascolto sullo stream per la ricezione dell'ordine.

```

simolb@fedora:~/Documents/GitHub/e-commerce/models/fornitore/bin -- ./main

Scheda tecnica creata correttamente per il prodotto con barcode: 0014067757058
Prodotto aggiunto correttamente nell'inventario per l'oggetto con barcode: 00140677
57058
    Aggiunta di 86 prodotti all'inventario

Scheda tecnica creata correttamente per il prodotto con barcode: 5176693102386
Prodotto aggiunto correttamente nell'inventario per l'oggetto con barcode: 51766931
02386

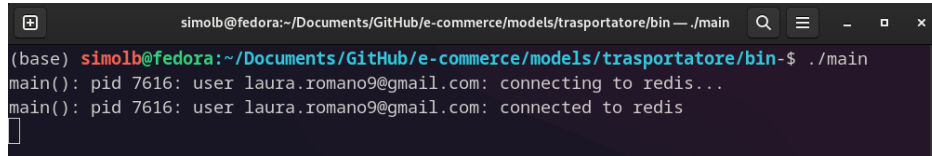
Scheda tecnica creata correttamente per il prodotto con barcode: 5813897882482
Prodotto aggiunto correttamente nell'inventario per l'oggetto con barcode: 58138978
82482

-----
main(): pid 7552: user giulia.rossi22@gmail.com: connecting to redis...
main(): pid 7552: user giulia.rossi22@gmail.com: connected to redis
    (string) customer2fornitore
        (string) 1710108182512-0
        (string) key Costumer
        (string) Nuovo ordine
result fval : Nuovo ordine

```

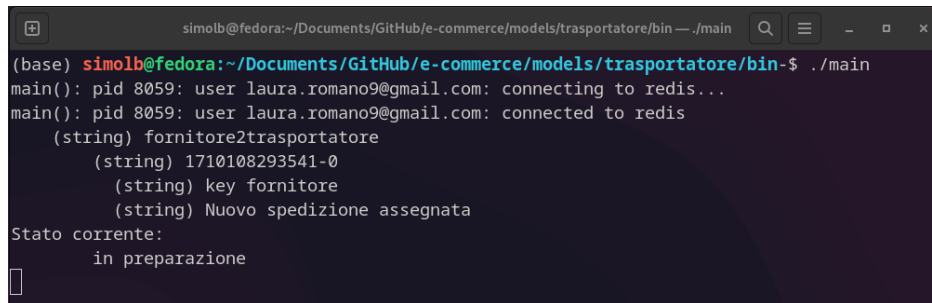
Figura 6: Processo del fornitore, nel momento in cui riceve un ordine proveniente dallo stream

2. **fornitore2trasportatore**, utilizzata dal fornitore per inoltrare, una volta ricevuto un ordine dalla stream 'customer2fornitore', l'ordine al trasportatore assegnato. Il trasportatore sarà in ascolto sullo stream.



```
simolb@fedora:~/Documents/GitHub/e-commerce/models/trasportatore/bin$ ./main
(base) simolb@fedora:~/Documents/GitHub/e-commerce/models/trasportatore/bin$ ./main
main(): pid 7616: user laura.romano9@gmail.com: connecting to redis...
main(): pid 7616: user laura.romano9@gmail.com: connected to redis
```

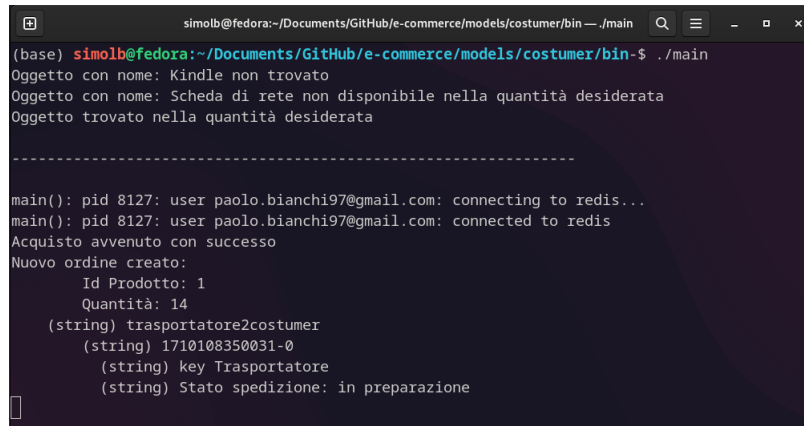
Figura 7: Processo del trasportatore, in ascolto sullo stream, in attesa di ordini assegnati dal fornitore



```
simolb@fedora:~/Documents/GitHub/e-commerce/models/trasportatore/bin$ ./main
(base) simolb@fedora:~/Documents/GitHub/e-commerce/models/trasportatore/bin$ ./main
main(): pid 8059: user laura.romano9@gmail.com: connecting to redis...
main(): pid 8059: user laura.romano9@gmail.com: connected to redis
  (string) fornitore2trasportatore
    (string) 1710108293541-0
      (string) key fornitore
        (string) Nuovo spedizione assegnata
Stato corrente:
  in preparazione
```

Figura 8: Processo del trasportatore, nel momento in cui viene assegnata una spedizione

3. **trasportatore2costumer**, utilizzata dal trasportatore per comunicare lo stato dell'ordine al customer e aggiornare lo stato dello stesso. Il customer riceverà aggiornamento sullo stream in tempo reale sullo stato dell'ordine.

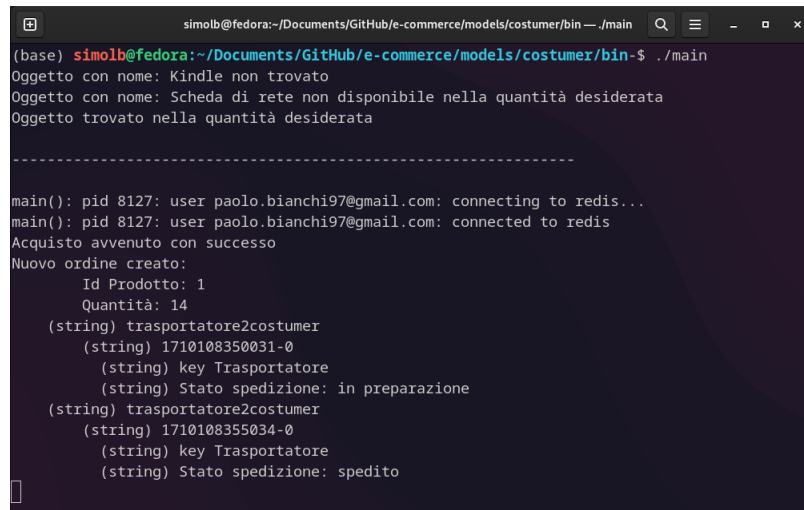


```
simolb@fedora:~/Documents/GitHub/e-commerce/models/costumer/bin$ ./main
(base) simolb@fedora:~/Documents/GitHub/e-commerce/models/costumer/bin$ ./main
Oggetto con nome: Kindle non trovato
Oggetto con nome: Scheda di rete non disponibile nella quantità desiderata
Oggetto trovato nella quantità desiderata

-----

main(): pid 8127: user paolo.bianchi97@gmail.com: connecting to redis...
main(): pid 8127: user paolo.bianchi97@gmail.com: connected to redis
Acquisto avvenuto con successo
Nuovo ordine creato:
  Id Prodotto: 1
  Quantità: 14
  (string) trasportatore2costumer
  (string) 1710108350031-0
  (string) key Trasportatore
  (string) Stato spedizione: in preparazione
```

Figura 9: Processo del customer, quando viene avviato invia in automatico un ordine come test, e riceve aggiornamenti sullo stato della spedizione



```
simolb@fedora:~/Documents/GitHub/e-commerce/models/costumer/bin$ ./main
(base) simolb@fedora:~/Documents/GitHub/e-commerce/models/costumer/bin$ ./main
Oggetto con nome: Kindle non trovato
Oggetto con nome: Scheda di rete non disponibile nella quantità desiderata
Oggetto trovato nella quantità desiderata

-----

main(): pid 8127: user paolo.bianchi97@gmail.com: connecting to redis...
main(): pid 8127: user paolo.bianchi97@gmail.com: connected to redis
Acquisto avvenuto con successo
Nuovo ordine creato:
  Id Prodotto: 1
  Quantità: 14
  (string) trasportatore2costumer
  (string) 1710108350031-0
  (string) key Trasportatore
  (string) Stato spedizione: in preparazione
  (string) trasportatore2costumer
  (string) 1710108350034-0
  (string) key Trasportatore
  (string) Stato spedizione: spedito
```

Figura 10: Processo del customer, quando lo stato dell'ordine diventa "spedito"

```
simolb@fedora:~/Documents/GitHub/e-commerce/models/costumer/bin — ./main
Oggetto con nome: Scheda di rete non disponibile nella quantità desiderata
Oggetto trovato nella quantità desiderata

-----

main(): pid 8127: user paolo.bianchi97@gmail.com: connecting to redis...
main(): pid 8127: user paolo.bianchi97@gmail.com: connected to redis
Acquisto avvenuto con successo
Nuovo ordine creato:
    Id Prodotto: 1
    Quantità: 14
    (string) trasportatore2costumer
    (string) 1710108350031-0
    (string) key Trasportatore
    (string) Stato spedizione: in preparazione
    (string) trasportatore2costumer
    (string) 1710108355034-0
    (string) key Trasportatore
    (string) Stato spedizione: spedito
    (string) trasportatore2costumer
    (string) 1710108360036-0
    (string) key Trasportatore
    (string) Stato spedizione: consegnato
```

Figura 11: Processo del customer, quando lo stato dell'ordine diventa "consegnato"

È stato creato un file *connRedis.cpp*, nella quale abbiamo inserito le funzioni per effettuare azioni sulle connessioni redis. Nello specifico sono state inserite all'interno:

- *readMsg*, per leggere contenuti da uno stream redis;
- *sendMsg*, per scrivere contenuti su uno stream redis.

5 Risultati sperimentali

Sono stati effettuati test generati automaticamente per tutte le funzionalità del sistema, tenendo in considerazione anche i casi di errore. In particolare sono stati generati oggetti, utenti e ruoli casualmente, utilizzando dei cluster di dati. I test affrontano varie casistiche:

- Registrazione e Login;
- Inserimento di oggetti e creazione di inventari;
- Ricerca ed acquisto;
- Aggiornamento stato spedizione.

I test hanno riportato risultati conformi a quelli attesi. Non sono state rilevate problematiche o situazioni non correttamente gestite. Il sistema garantisce conformità alle specifiche richieste.