Relazione progetto e-commerce

F.Sardo(1760539) S.Camagna(1997716)
E.Bouquillon(2014320)

Contenuti

DESCRIZIONE GENERALE

Si progetti una piattaforma di e-commerce in cui i fornitori possono mettere in vendita i propri prodotti e acquirenti privati possono comprarli.

Un acquirente deve potersi registrare fornendo la propria mail, nome cognome, un indirizzo di residenza e altri eventuali indirizzi secondari, dei quali interessa sapere via, numero civico, citta e cap. Per effettuare un acquisto, oltre a un indirizzo di consegna l'utente deve inserire almeno un metodo di pagamento di cui interessa sapere il tipo (carta di credito, bancomat o carta prepagata). L'utente deve poter cercare i prodotti da inserire nel carrello, dei prodotti serve sapere nome, fornitore, prezzo e una breve descrizione. Il fornitore è il venditore del prodotto, di cui bisogna sapere ragione sociale, partita iva, mail, numero di telefono e indirizzo della sede. Il fornitore deve poter mettere in vendita i suoi prodotti o, in caso di necessita, di toglierli dalla piattaforma. Una volta inseriti i prodotti nel carrello l'utente deve poter effettuare un ordine con i prodotti presenti, un ordine è caratterizzato da almeno un prodotto, la data di richiesta dell'ordine, il metodo di pagamento utilizzato, l'indirizzo a cui effettuare la consegna(che deve essere uno di quelli registrati per quell'utente), il totale e lo stato dell'ordine(se annullato, pendente o accettato). È possibile annullare gli ordini se sono ancora in stato pendente.
Il sistema deve permettere a delle compagnie di trasporto di prendere in carico l'ordine per una consegna. Delle compagnie di trasporto interessa la ragione sociale, la partita iva, e l'indirizzo della sede. Delle consegne invece interessa sapere l'ordine in consegna e il corriere che effettua la consegna, che deve essere della compagnia che ha preso in carico. L'utente deve poter risalire agli ordini consegnati.

2. Analisi software

2.1 Requisiti utente

2.1.1 Lista dei requisiti

1.customer
- 1.1 e-mail
- 1.2 nome
- 1.3 cognome
- 1.4 numero telefonico(0,1)
- 1.5 indirizzo
  - 1.5.1 CAP
  - 1.5.2 via
  - 1.5.3 numero civico
  - 1.5.4 citta
- 1.6 metodi pagamento
  - 1.6.1 tipo(prepagata, bancomat, carta di credito)
  - 1.6.2 nome
- 1.7 carrello
  - 1.7.1 insieme di prodotti nel carrello

2.Produttore
- 2.1 nome
- 2.2 partita iva
- 2.3 indirizzo sede(vedere 1.5)
- 2.4 mail
- 2.5numero telefono

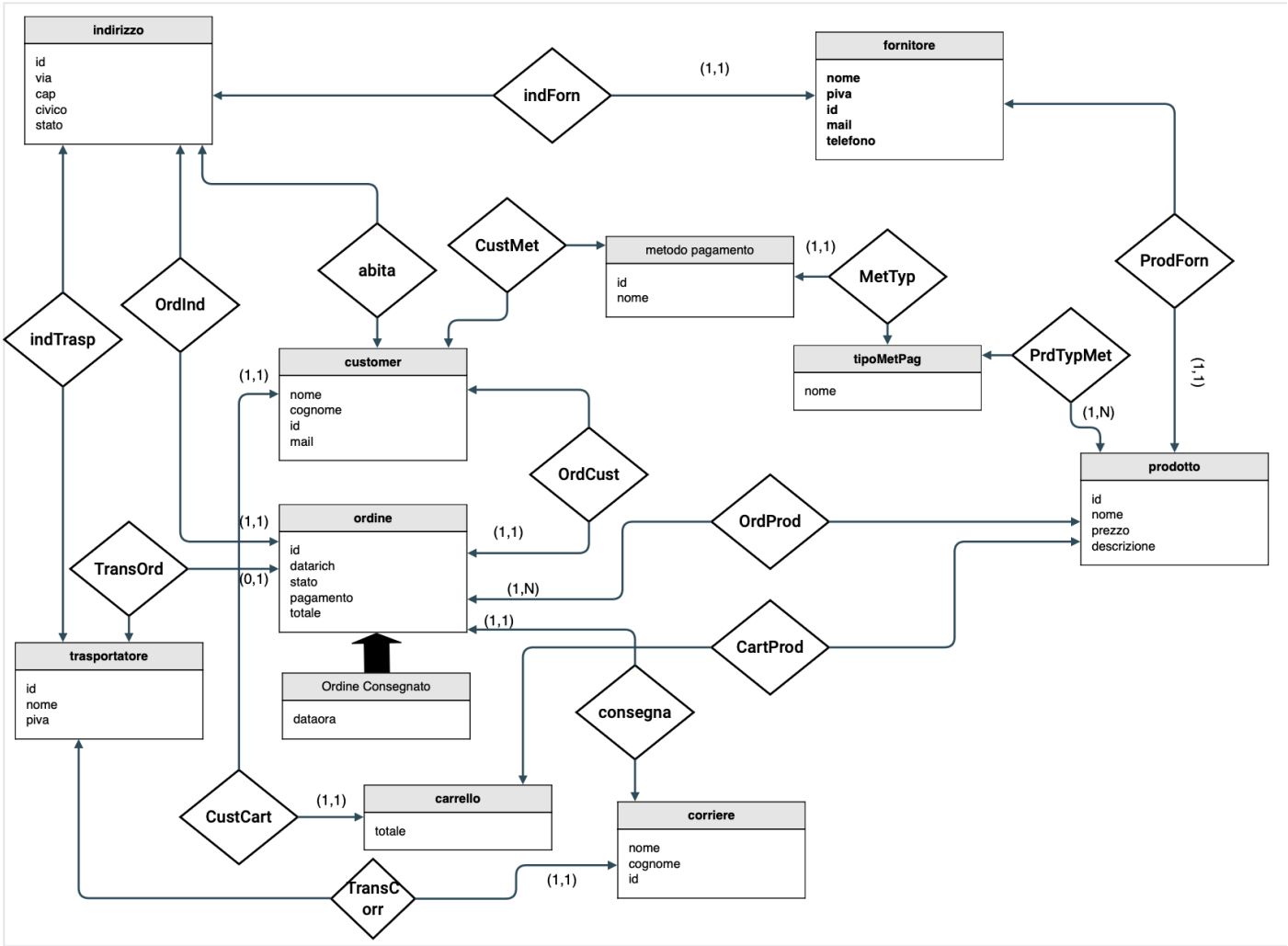
3.prodotti
- 3.1 nome
- 3.2 produttore(vedi 2.)
- 3.3 descrizione
- 3.4 prezzo
- 3.5 metodi pagamento accettati

4.ordini
- 4.1 customer
- 4.2 totale ordine
- 4.3 indirizzo di consegna
- 4.4 stato(pendente, annullato, accettato)
- 4.5 data richiesta
- 4.6 metodo pagametno usato
- 4.7 se consegnato richiedere anche
  - 4.7.1 data consegna

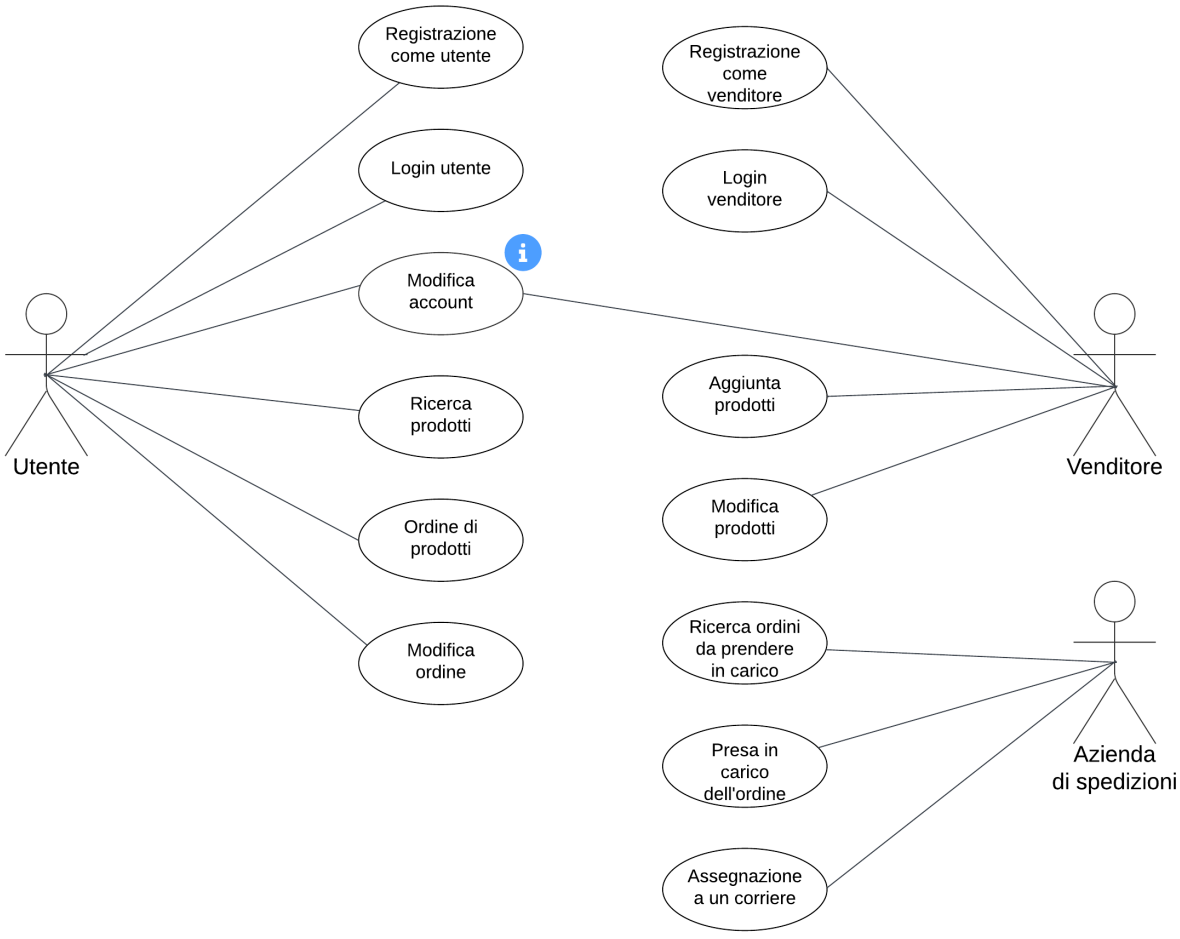
5.consegna
- 5.1 ordine
- 5.2 corriere

6.corriere
- 6.1azienda per cui lavora
- 6.2 nome
- 6.3 cognome

2.1.2 Entity-Relationship diagram



2.1.3 Use-case diagram



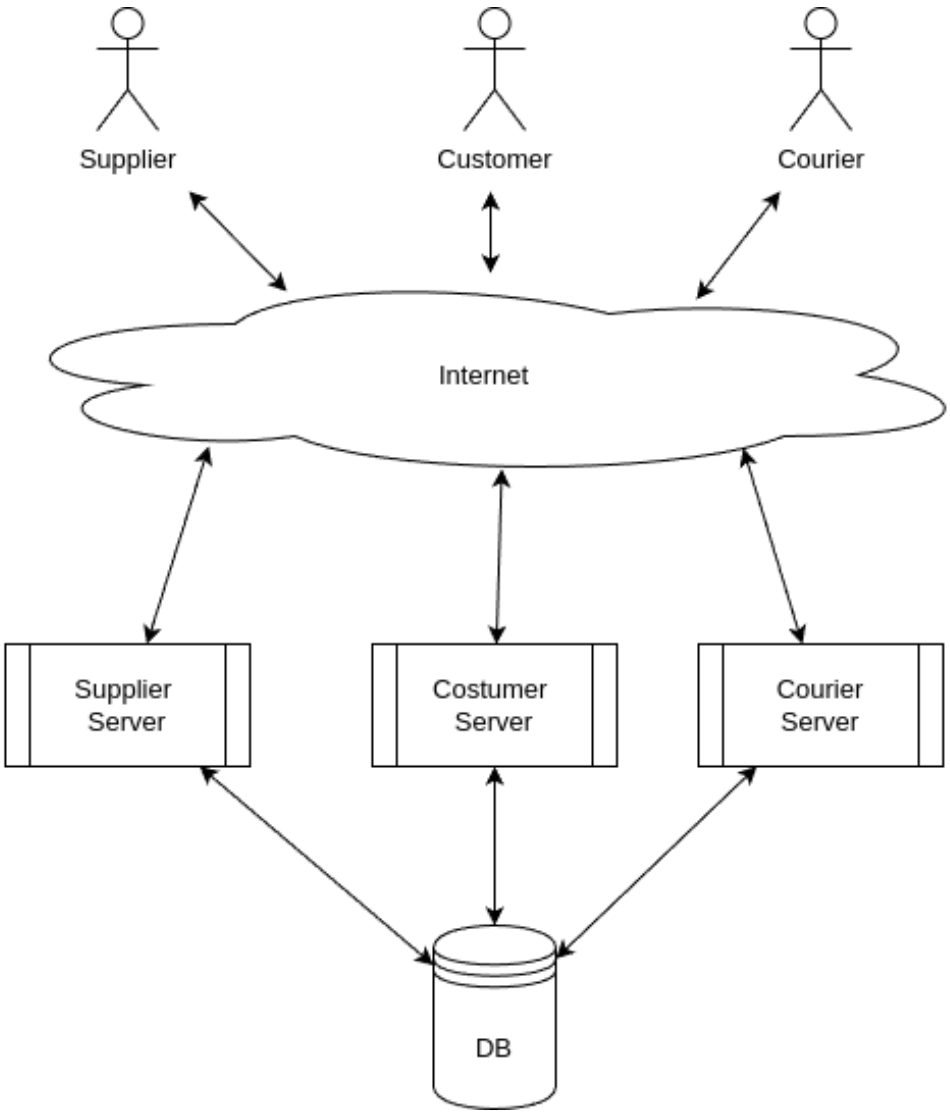
2.2 System Requirement
2.2.1 requisiti funzionali e non funzionali

1. il customer deve poter
   1.1 cercare prodotti
   1.2 inserire e gestire i prodotti nel carrello
   1.3 effettuare un ordine
   1.4 inserire indirizzi

2. il fornitore deve poter
   2.1 gestire prodotti
3. il trasportatore deve poter
   3.1 scegliere ordini da consegnare
   3.2 confermare consegna ordine
      3.2.1 fornendo data e ora
4. requisiti non funzionali
   4.1 il sistema deve eseguire ogni operazione in un tempo massimo di 1 secondo

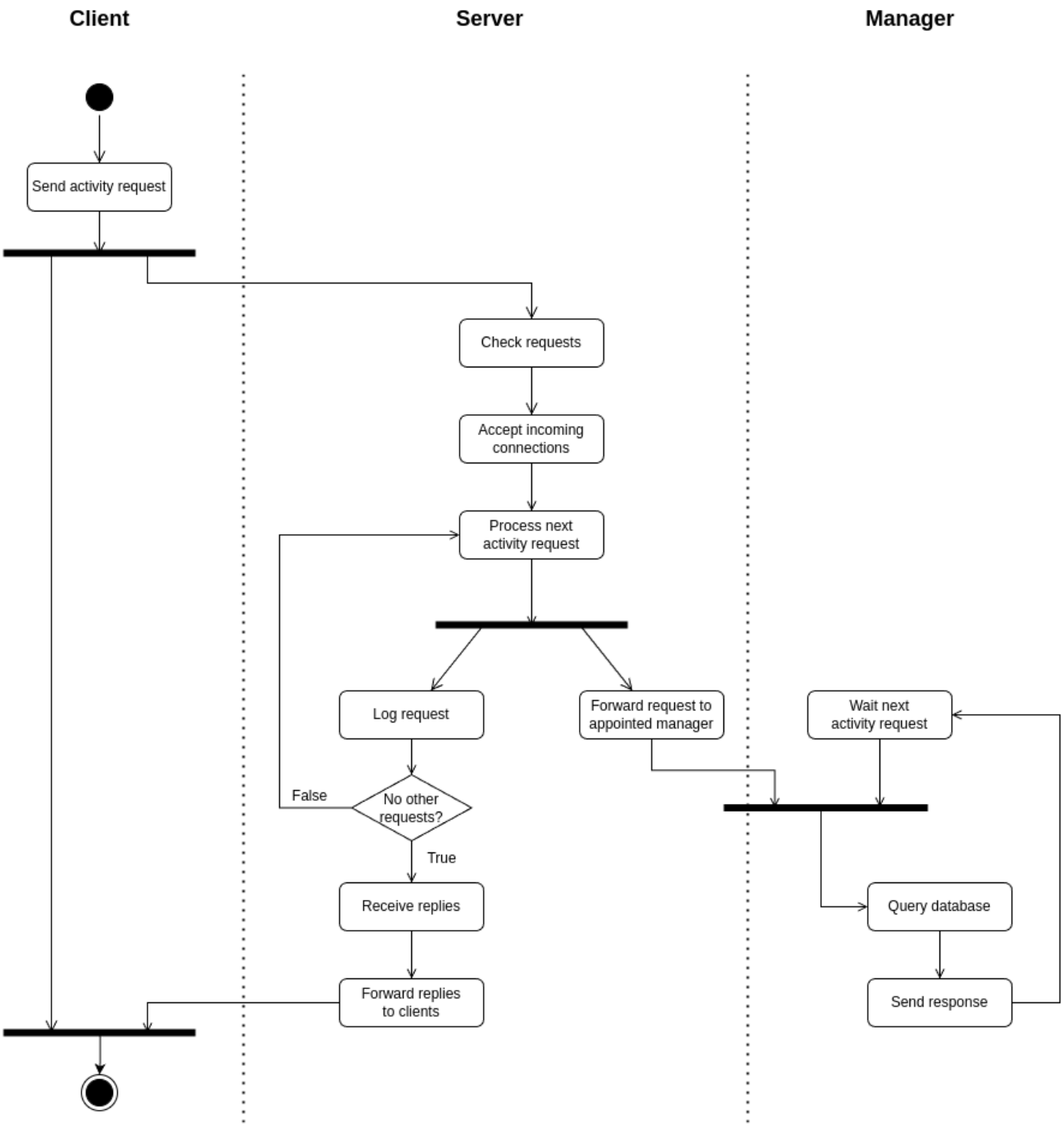
2.2.2 architettura del sistema

Il sistema viene gestito separando le richieste ricevute in base al ruolo di chi effettua la richiesta. Nello specifico il sistema viene diviso in tre server, uno accessibile ai customer, uno accessibile ai produttori e un ultimo accessibile ai trasportatori. I server sono gestiti parallelamente Agli altri due e si occupa di gestire e processare le varie richieste per cui è pensato.



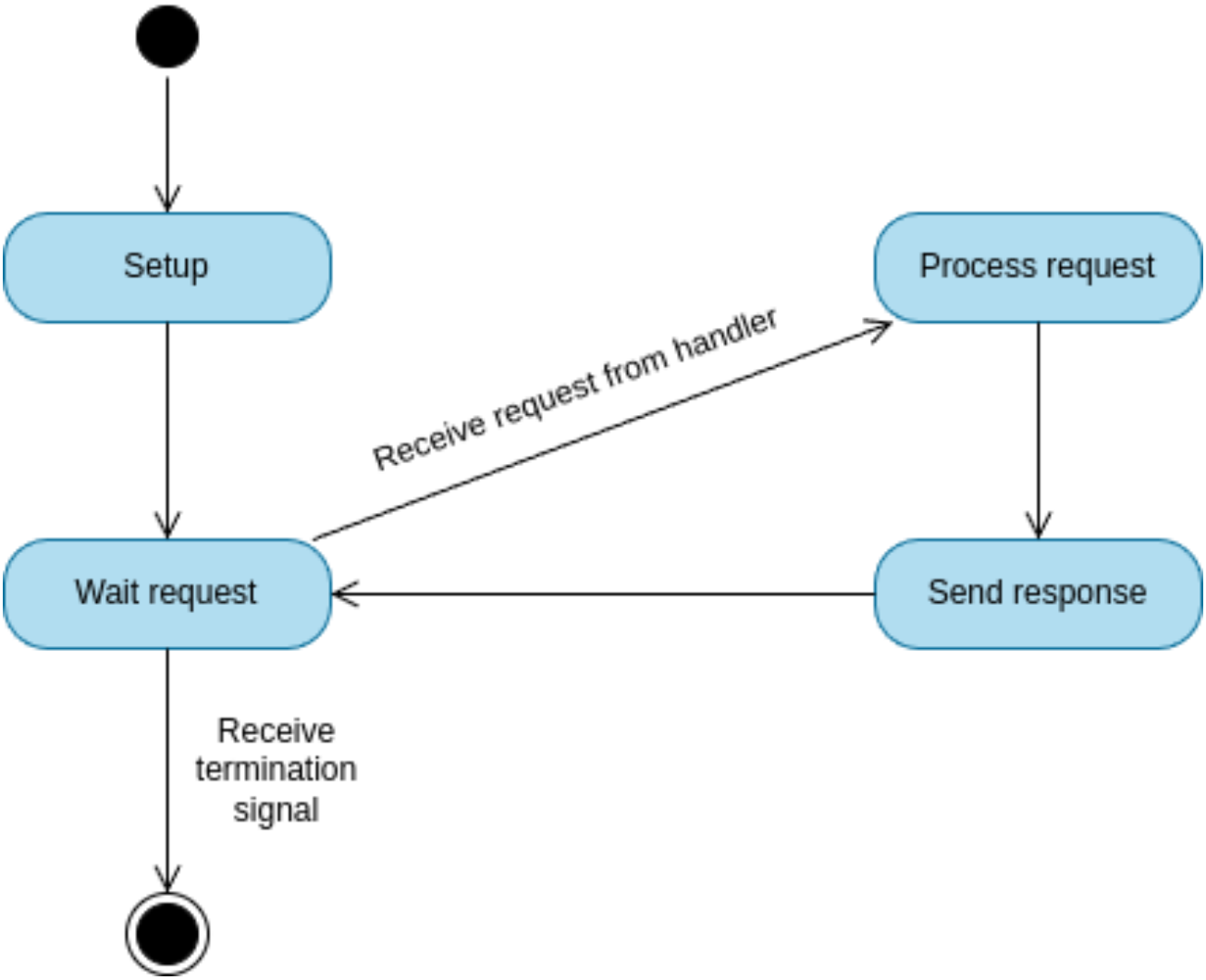
2.2.3 activity diagram

Riportiamo in seguito l'activity diagram relativo a una richiesta da parte di un customer, la sua elaborazione e la gestione della risposta da parte del server. Questo processo è analogo per gli altri due server. Nel diagramma è considerata la situazione in cui il customer ha già effettuato la connessione con il server e che il suo handler sia pronto a processare una richiesta.



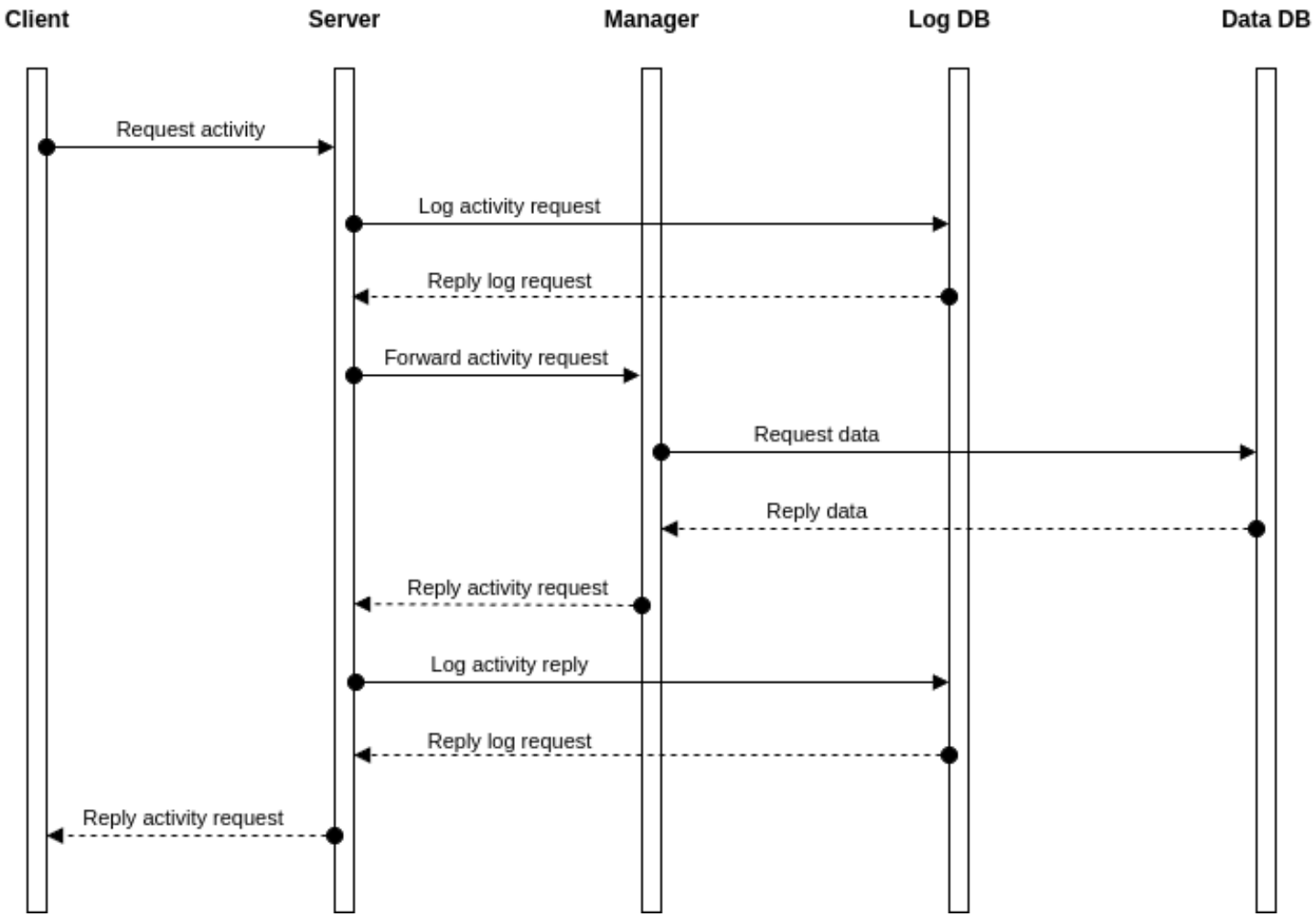
2.2.4 state diagram

Ogni manager corrisponde a una macchina a stati finiti che altera tra uno stato di ricezione, uno stato di elaborazione e infine uno stato di invio della risposta. Il comportamento ‘ ovviamente analogo per gli altri manager.



È opportuno notare che ogni manager rimane in funzione finché non riceve un segnale di terminazione dal server. Una volta ricevuto tale segnale, se il manager sta aspettando una richiesta, la comunicazione

2.2.5 message sequence chart



3.implementazione software
3.1 struttura del codice e delle connessioni redis
All'interno del progetto sono presenti tre server distinti, Customer, Trasportatore e Fornitore, che si interfacciano allo stesso database utilizzando lo stream Redis loro dedicato. Il client invia richieste al server tramite protocollo http (implementato grazie alla libreria Pistache). A tale scopo sono stati definiti degli endpoint connessi alle rispettive funzioni. Prendiamo come esempio l'autenticazione del Customer:

quando il client invia una richiesta http al server di questo tipo:
"curl -X GET http://localhost:5001/autentica/abc@abc.it", il server identifica la funzione corrispondente alla rotta. In questo caso si tratta di autenticaCustomer.

"Pistache::Rest::Routes::Get(router, "/autentica/:email",
Pistache::Rest::Routes::bind(&autenticaCustomer));"

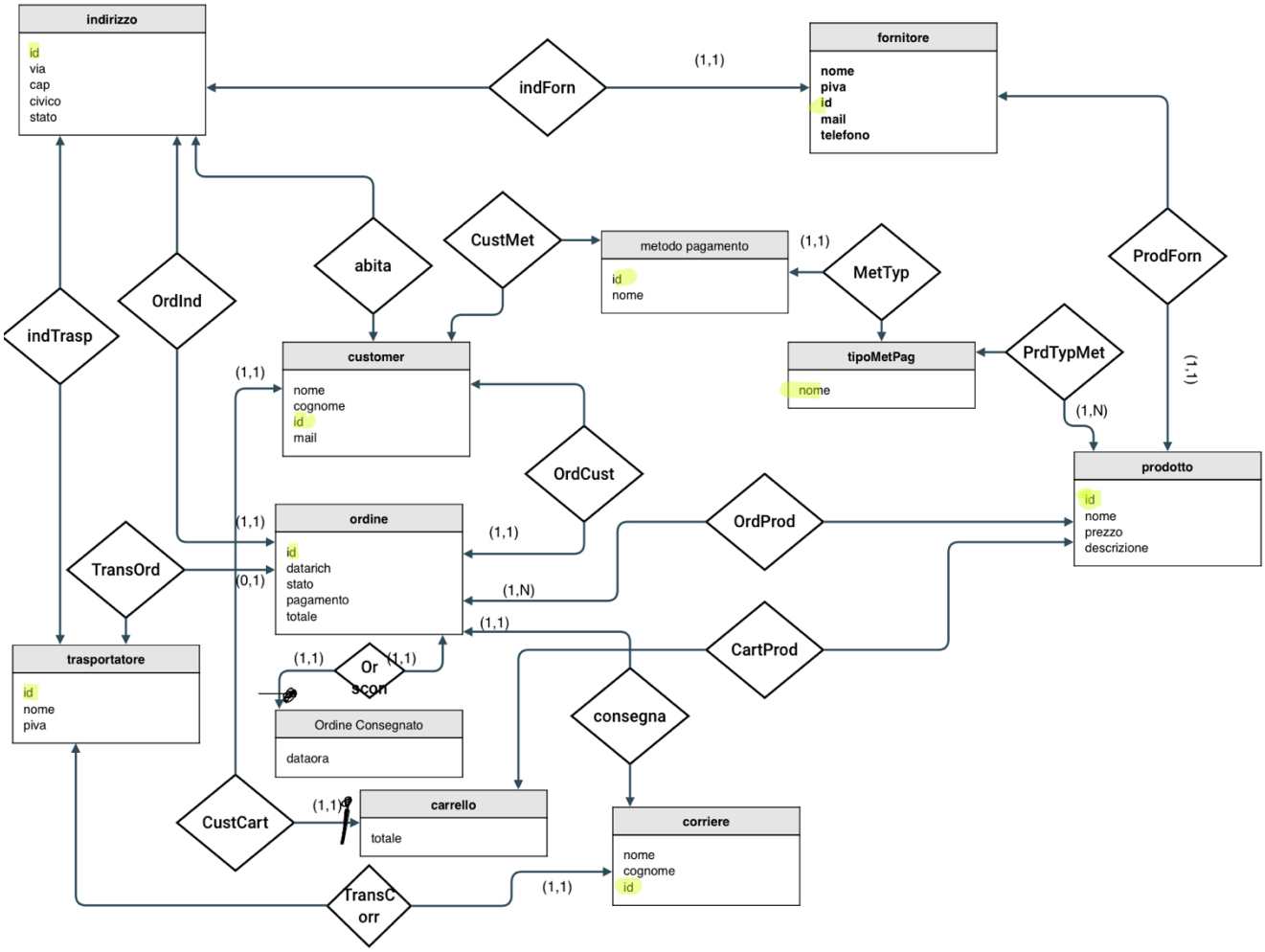
La funzione chiamata si occupa di reperire tutti i dati utili presenti all'interno della richiesta, per poi salvarli nello stream Redis.
Sarà compito delle classi che interagiscono direttamente con il database leggere i dati da Redis, eseguire la query e scrivere sullo stream il risultato (se necessario), di modo che gli endpoint possano ritornarlo al client.
Prendiamo per esempio il caso in cui dal client arriva una richiesta HTTP GET per il carrello. Per poter reperire il carrello dell'utente abbiamo bisogno di avere la sua email, che otteniamo dall'header. Tramite email, viene chiamata la funzione del server che interagisce direttamente con il database e una volta eseguita la query, i prodotti presenti nel carrello vengono salvati in Redis:

```
        // Memorizza il prodotto in Redis come hash
        redisCommand(c2r, "HMSET prodottoCarr:%d id %d descrizione %s prezzo %f
nome %s fornitore %s quantità %d",
                ID, ID, descrizione, prezzo, nome, fornitore, quantita);

        // Aggiungi l'ID del prodotto alla lista associata all'email
        redisCommand(c2r, "RPUSH carrello:%s %d", mail, ID);
```

Sarà quindi compito della parte di server che si occupa di interagire con il client, leggere i dati precedentemenți scritti e formattarli in modo da essere stampabili su schermo del terminale che ha fatto la richiesta.

3.2 schema del database



3.3monitor funzionali

per garantire il corretto funzionamento logico del sistema nel database sono stati inseriti dei monitor funzionali implementati tramite trigger, in modo che i dati rimangano consistenti tra loro e che rispettino tutti i vincoli rappresentati in precedenza.

Eccone tre di esempio:

Trigger che gestisce il prezzo del carrello in base ai prodotti inseriti:

```
create or replace function tot_cart() returns trigger as $buy_price$
   BEGIN
   update carrello
   set totale=(select SUM((p.prezzo * NEW.quantita))
         from  carrello c, prodotto p
         where c.customer=NEW.carrello
         and NEW.prodotto=p.id)
    where carrello.customer=NEW.carrello;


      return NEW;
   END
   $buy_price$
   language plpgsql;


   create or replace TRIGGER buy_price after insert or update on prodincart
                        for each row execute procedure tot_cart();
```

Trigger che gestisce l'inserimento multiplo di prodotti uguali nel carrello:

```
create or replace function no_double_cart() RETURNS TRIGGER AS
$check_cart$
   BEGIN
        if exists(select *
                     from prodincart pc
                     where pc.carrello=NEW.carrello and
pc.prodotto=NEW.prodotto
```

```
                    ) then update prodincart set quantita=(select (pc.quantita +
NEW.quantita)
                                                    from prodincart pc
                                                    where pc.prodotto=NEW.prodotto
                                                    and pc.carrello=NEW.carrello)
                            where prodincart.prodotto=NEW.prodotto and
prodincart.carrello=NEW.carrello;
                            RETURN NULL;

            else RETURN NEW;
            END IF;
   END
  $check_cart$
  language plpgsql;
  create or replace TRIGGER check_cart before insert on prodincart
for each row execute procedure no_double_cart();
```

Trigger che controlla che tutti i prodotti inseriti nell'ordine utilizzino un metodo comune:

```
CREATE or REPLACE FUNCTION check_meth() RETURNS trigger as
$same_meth$
   BEGIN
      if EXISTS(
               select *
                  From ordine o, prodmet pm
                           where pm.prodotto=NEW.prodotto
                           and o.id=NEW.ordine and o.pagamento=pm.metodo
            ) THEN return NEW;

      ELSE  return null;

      end if;
   END
   $same_meth$
   LANGUAGE plpgsql;



create or replace TRIGGER same_meth before insert on prodinord
for each row execute Procedure check_meth();
```

3.4 risultati sperimentali

Il sistema rappresenta un sistema di e-commerce che puo essere effettivamente utilizzata tramite il file install.sh e seguendo i comandi disponibili nel file README presente nel progetto.
I test effettuati garantiscono il funzionamento del sistema anche in caso di sovraccarichi di richieste, input errati e inconsistenti.