Relazione progetto e-commerce

Edoardo Bouquillon (2014320)
Simone Camagna (1997716)
Federica Sardo (1760539)

Contenuti

1. 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 e-mail, nome, cognome, un indirizzo di residenza e altri eventuali indirizzi secondari, dei quali interessa sapere via, numero civico, città 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 necessità, 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

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 città
    - 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. Fornitore
    - 2.1 nome
    - 2.2 partita iva
    - 2.3 indirizzo sede (vedere 1.5)
    - 2.4 e-mail
    - 2.5 numero telefono

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

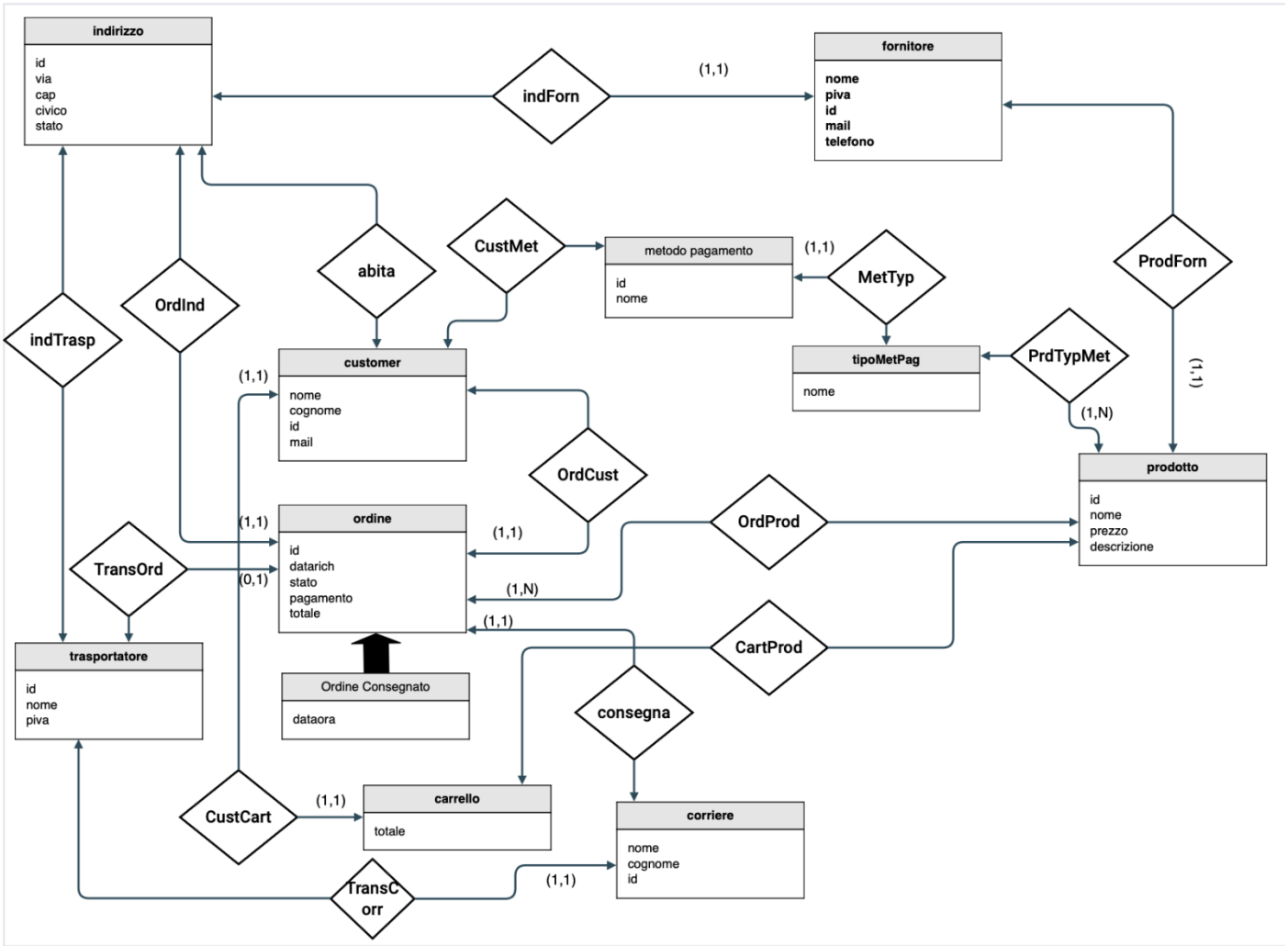
4. Ordine
    - 4.1 customer
    - 4.2 totale ordine
    - 4.3 indirizzo di consegna
    - 4.4 stato (pendente, annullato, accettato)
    - 4.5 data richiesta
    - 4.6 metodo pagamento usato

4.7 se consegnato richiedere anche
4.7.1 data consegna

5. Consegna
   5.1 ordine
   5.2 corriere

6. Corriere
   6.1 azienda per cui lavora
   6.2 nome
   6.3 cognome

Entity-Relationship diagram



Use-case diagram

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

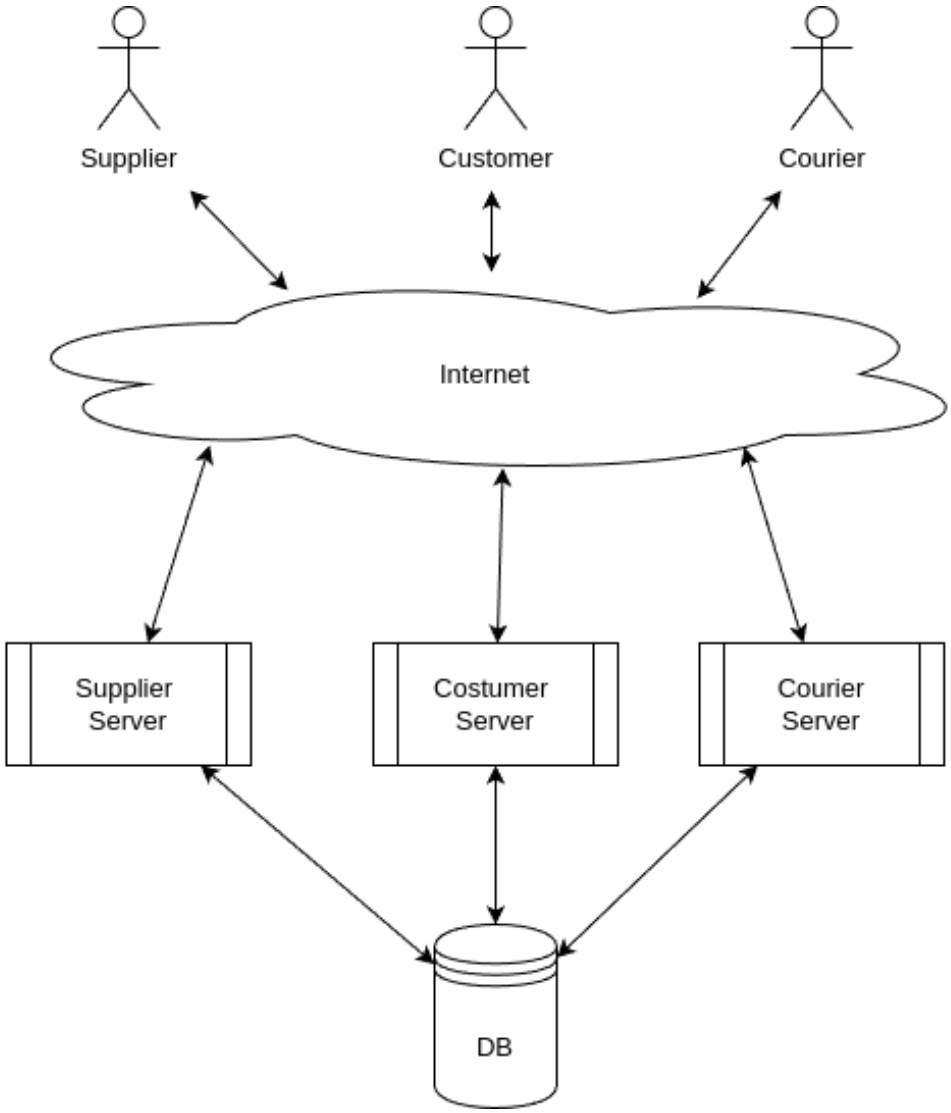
2. Il fornitore deve poter
   2.1 gestire prodotti

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

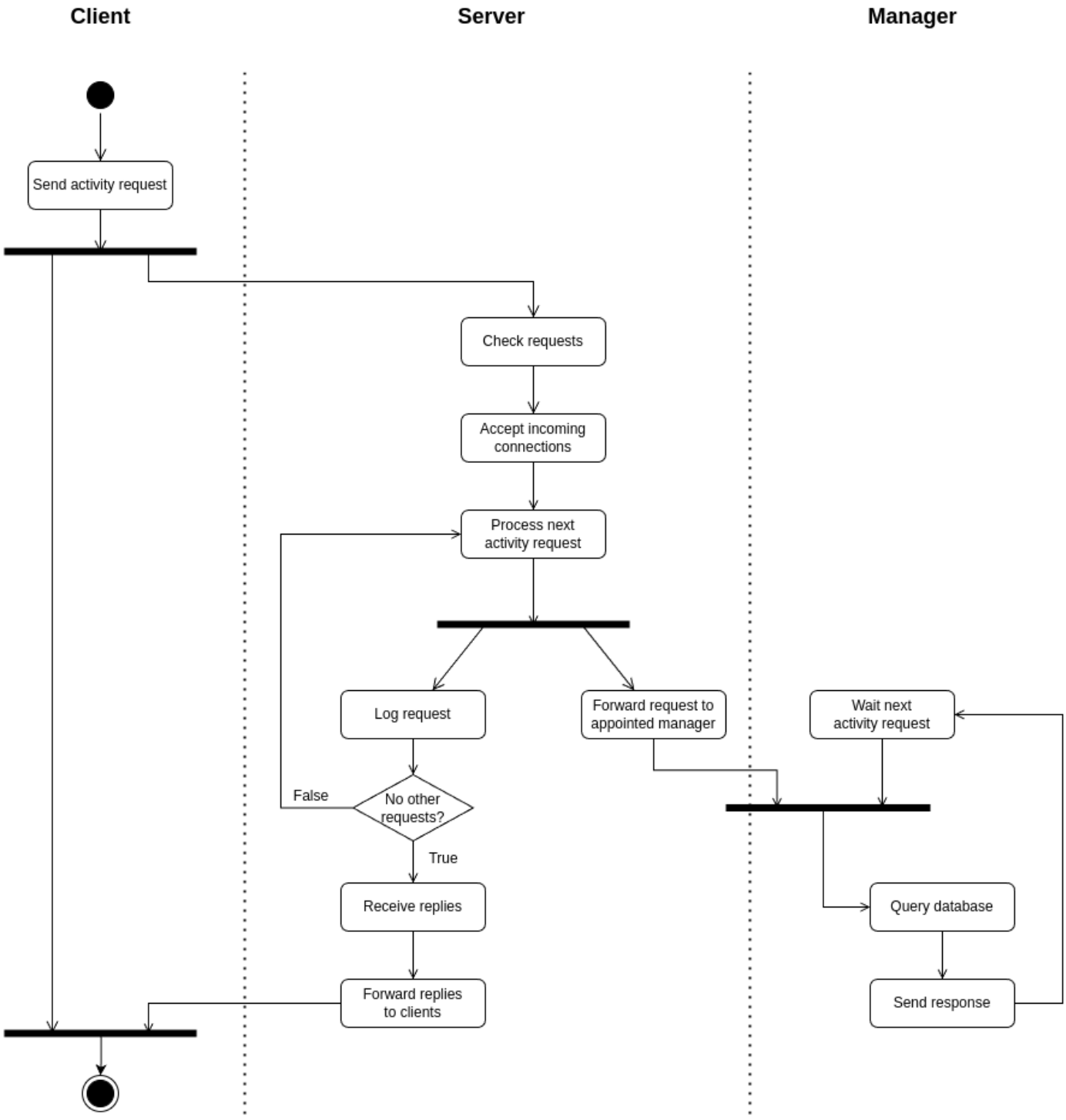
Architettura del sistema

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 fornitori e un ultimo accessibile ai trasportatori. Ogni server è gestito parallelamente agli altri e si occupa di ricevere e processare le varie richieste per cui è pensato.



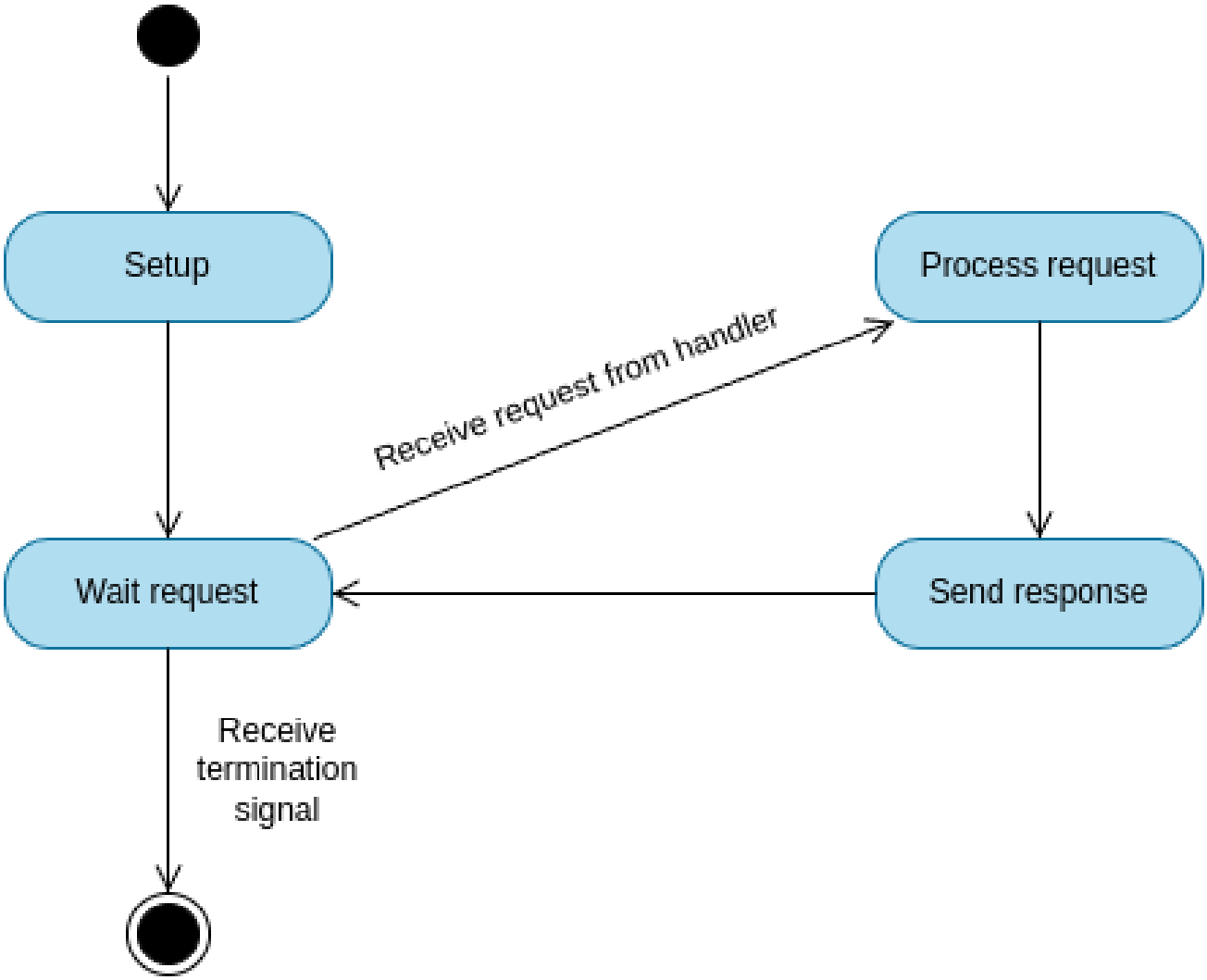
Activity diagram

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 tutti e tre i 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.



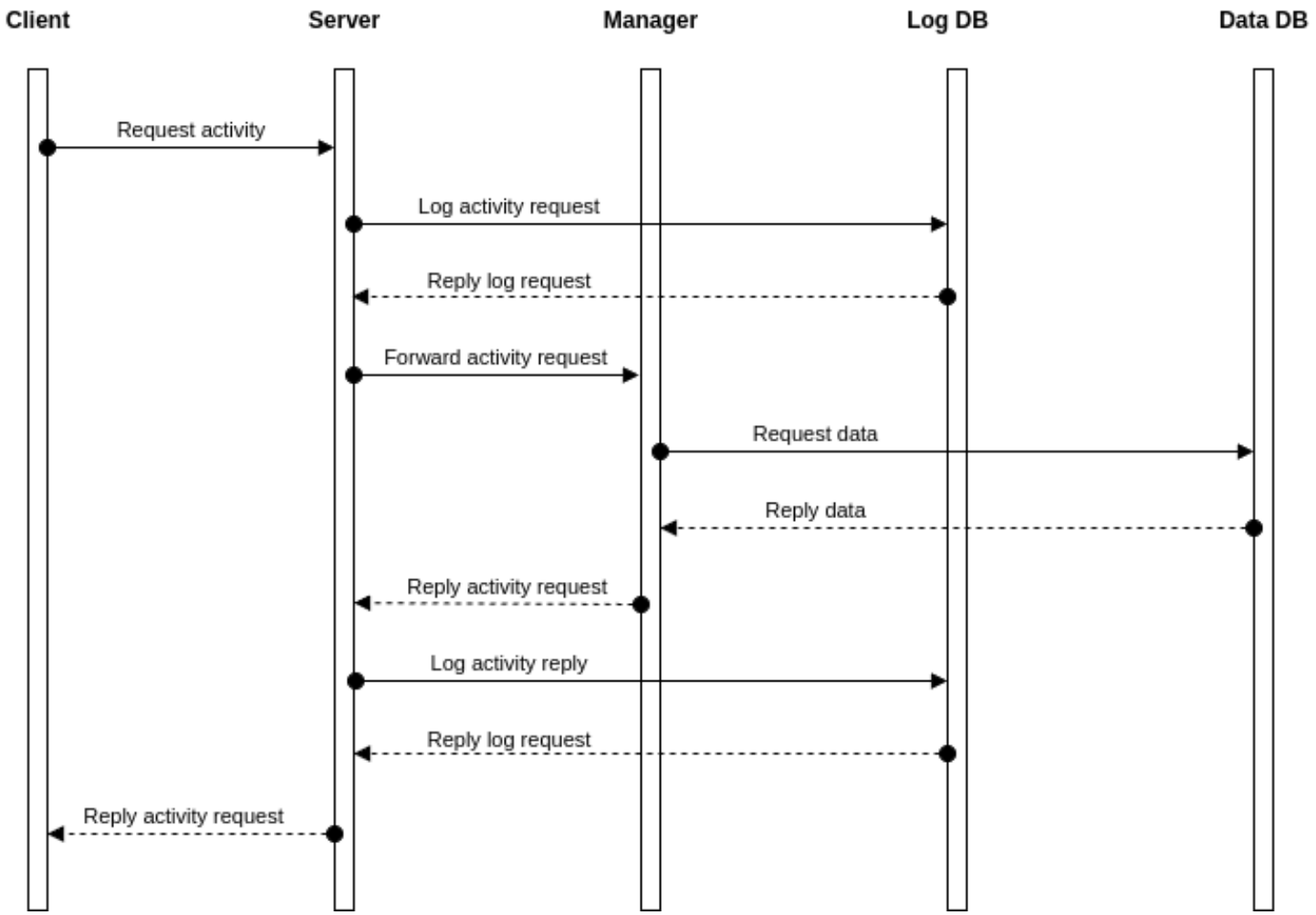
State diagram

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.



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 viene chiusa e nessuna altra richiesta viene più analizzata fino a riapertura della comunicazione.

Message sequence chart



3. Implementazione del software

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));
```

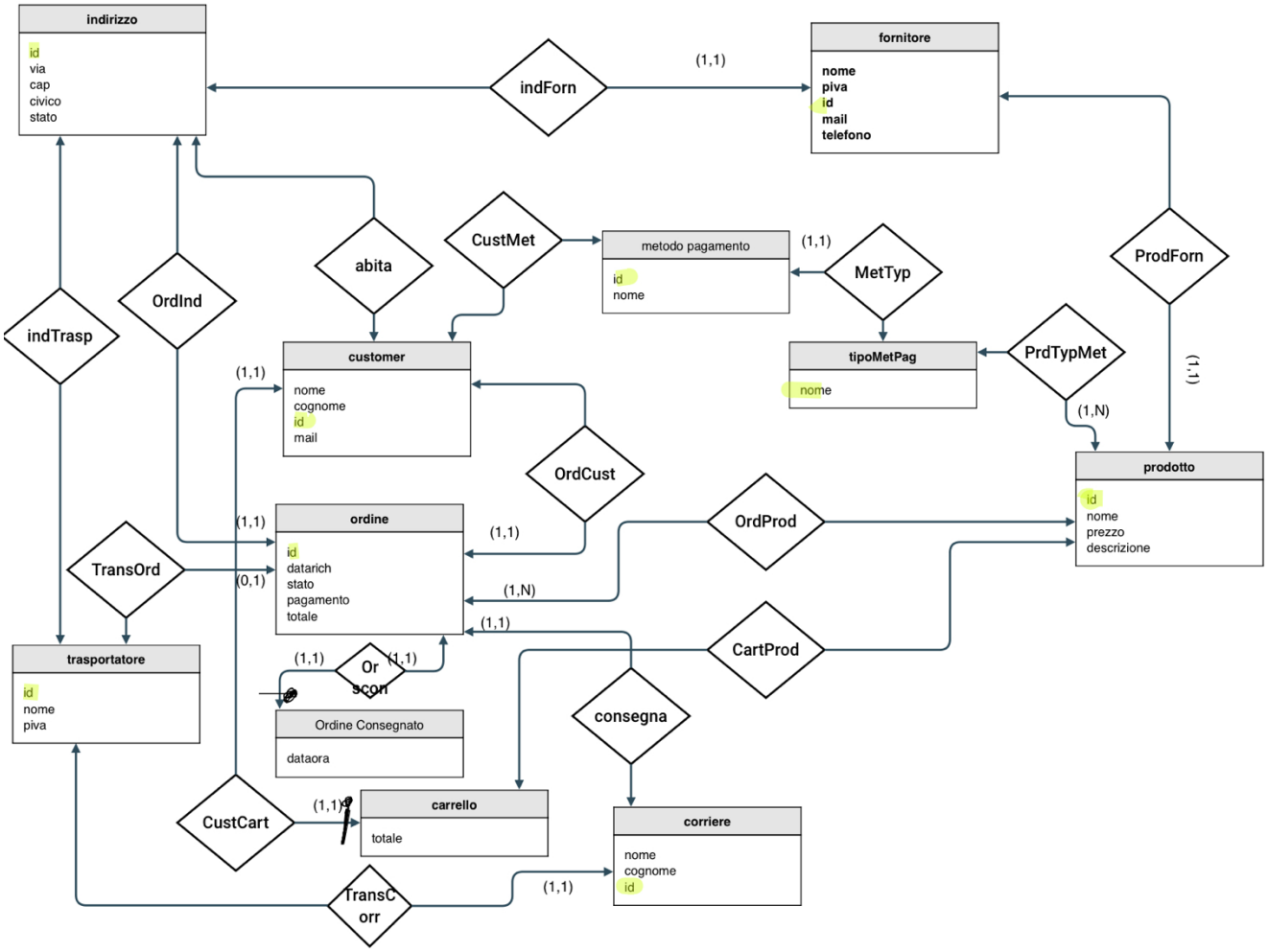
La funzione chiamata si occupa di reperire tutti i dati utili presenti all'interno della richiesta, per poi salvarli nello stream Redis.
Sarà compito delle classi che interagiscono direttamente con il database leggere i dati da Redis, eseguire la query e scrivere sullo stream il risultato (se necessario), 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 e-mail, che otteniamo dall'header. Tramite e-mail, 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 precedentemente scritti e formattarli in modo da essere stampabili su schermo del terminale che ha fatto la richiesta.

Schema del database



Monitor 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:

1. *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();
```

2. *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 1
        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();
```

3. 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 1
        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();
```

Risultati sperimentali

Il sistema rappresenta il backend di un sito di e-commerce che può essere effettivamente installato e compilato tramite il file install.sh e seguendo i comandi disponibili nel file README presente nella cartella progetto.
I test effettuati garantiscono il funzionamento del sistema anche in caso di sovraccarichi di richieste, input errati e inconsistenti. È da notare però che il database non riesce a gestire più di 50 richieste contemporaneamente sul sistema operativo utilizzato per lo sviluppo (Ubuntu 24). Questa soglia può essere aumentata cambiando le impostazioni di sistema tramite terminale.
L'altra nota da fare è che in caso di errori fatali nelle query del database, il server termina e non è più in grado di ricevere richieste.
In generale le operazioni vengono eseguite sempre in meno di un secondo e l'uso sia di REST API con Pistache, sia di Redis permette una buona gestione delle operazioni concorrenti. Riportiamo in seguito l'elenco degli endpoint con le rispettive operazioni implementate nel progetto.

1. Server customer

Metodo	*Endpoint*	*Descrizione*
GET	/autentica/:email	Autenticazione del customer tramite e-mail
GET	/:email/indirizzi/	Recupera gli indirizzi di un customer
GET	/:email/ordini/	Recupera gli ordini di un customer
GET	/:email/carrello/	Recupera il carrello di un customer
GET	/:email/prodotti/	Recupera i prodotti
PUT	/:email/indirizzi/	Aggiunge un indirizzo per un customer
PUT	/:email/ordini/	Inserisce un nuovo ordine
PUT	/:email/carrello/	Aggiunge un prodotto al carrello
PUT	/autentica/	Crea un nuovo customer
POST	/:email/	Modifica le informazioni di un customer
DELETE	/:email/carrello/:productID	Rimuove un prodotto dal carrello
DELETE	/:email/ordini/:ordineID	Annulla un ordine
DELETE	/:email/indirizzi/:addressID	Rimuove un indirizzo

2. Server fornitore

Metodo	*Endpoint*	*Descrizione*
GET	/:email/prodotti/	Recupera i prodotti di un fornitore
GET	/autentica/:email`	Autenticazione del fornitore tramite e-mail
PUT	/autentica/	Crea un nuovo fornitore
PUT	/:email/prodotti/	Aggiunge un nuovo prodotto
POST	/:email/	Modifica le informazioni di un fornitore
POST	/:email/prodotti/:idProdotto	Modifica un prodotto esistente
DELETE	/:email/prodotti/:idProdotto	Elimina un prodotto

3. Server trasportatore

Metodo	*Endpoint*	*Descrizione*
GET	/autentica/:piva	Autenticazione del trasportatore tramite PIVA
GET	/:piva/ricerca/	Recupera gli ordini del trasportatore
GET	/:piva/corrieri/	Recupera i corrieri associati al trasportatore
GET	/:piva/correnti/	Recupera le correnti del trasportatore
GET	/:piva/registrati/	Recupera i registrati del trasportatore
PUT	/:piva/corrieri/	Aggiunge o modifica un corriere
PUT	/:piva/ordini/	Accetta un ordine
PUT	/autentica/	Crea un nuovo trasportatore
POST	/:piva/ordini/:orderID	Consegna un ordine
DELETE	/:piva/corrieri/:courierID	Elimina un corriere