# Progetto e-commerce

A. Bandiera, I. Radu Barbalata, S. Bianco, S. Lazzaroni

March 16, 2024

# Contents

# Chapter 1

# Descrizione generale

## 1.1 Specifica dei requisiti

Si vuole progettare una piattaforma di e-commerce, nella quale grandi fornitori possono mettere in vendita prodotti per acquirenti privati.

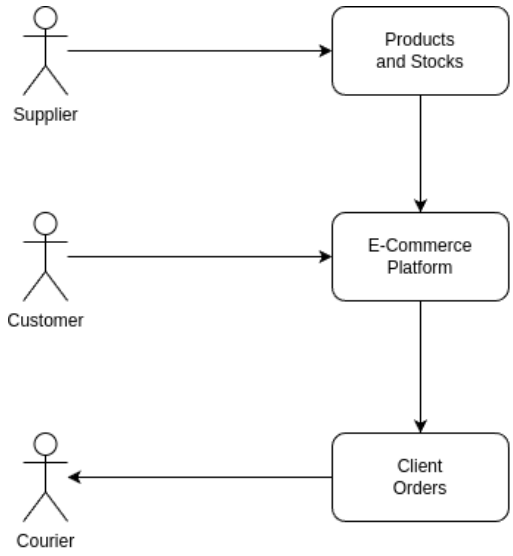
Un acquirente deve potersi registrare sulla piattaforma attraverso una e-mail, il proprio nome, il proprio cognome ed un recapito telefonico; inoltre, per effettuare ordini ogni acquirente deve indicare almeno un indirizzo per la consegna dei propri acquisti, ed ogni indirizzo è contrassengato da CAP, via, numero civico, città e nazione di appartenenza. Ogni utente deve anche registrare almeno una carta per effettuare gli acquisti, la quale è definita dal proprio codice identificativo. Infine, il sistema deve poter permettere agli acquirenti di cercare i prodotti della piattaforma per nome, ed ogni acquirente deve poter lasciare un feedback ai prodotti da lui acquistati, rappresentato da un punteggio da 1 a 5 stelle.

Un prodotto è costituito da un codice identificativo, un nome, una descrizione ed il prezzo. Ogni acquirente può ordinare i vari prodotti presenti all'interno della piattaforma, resi disponibili da fornitori; all'interno di un ordine l'acquirente può scegliere la quantità di ogni prodotto che desidera acquistare, anche di vari fornitori. Il costo dell'acquisto è pari al totale dei prezzi dei prodotti ordinati, sommato al costo di spedizione, il quale è costituito da una tariffa fissa imposta dal servizio di e-commerce, indipendente dai prodotti e dai fornitori. Inoltre, è possibile annullare gli ordini effettuati

prima che questi vengano spediti all'acquirente. Se l'acquirente è insoddisfatto di alcuni prodotti ricevuti all'interno di un acquisto, il sistema permette di effettuarne il reso gratuitamente.

Un fornitore, ovvero un'azienda privata, deve potersi registrare inserendo nel sistema la propria ragione sociale. Inoltre, ogni fornitore deve poter registrare, modificare e rifornire i prodotti che desidera mettere in vendita, e può verificare quale siano i prodotti maggiormente venduti in un lasso temporale specificato.

I trasportatori sono costituiti da aziende private di trasporto merci, che possono registrarsi sulla piattaforma attraverso la propria ragione sociale, ed hanno accesso alla gestione delle spedizioni da loro effettuate; in particolare, si occuperanno di prendere in carico gli ordini a cui non è ancora stato assegnato uno spedizioniere, notificare l'avvenuta consegna di un ordine, l'avvenuto reso o l'eventuale perdita del pacco da consegnare.



# Chapter 2

# Analisi del software

## 2.1 User requirements

### 2.1.1 Lista dei requisiti

1. Customer

    1.1 Registrazione alla piattaforma

        1.1.1 E-mail

        1.1.2 Nome

        1.1.3 Cognome

        1.1.4 Recapito telefonico

        1.1.5 Indirizzo

            1.1.5.1 CAP

            1.1.5.2 Via

            1.1.5.3 Numero civico

            1.1.5.4 Città

            1.1.5.5 Nazione

    1.2 Registrazione carte

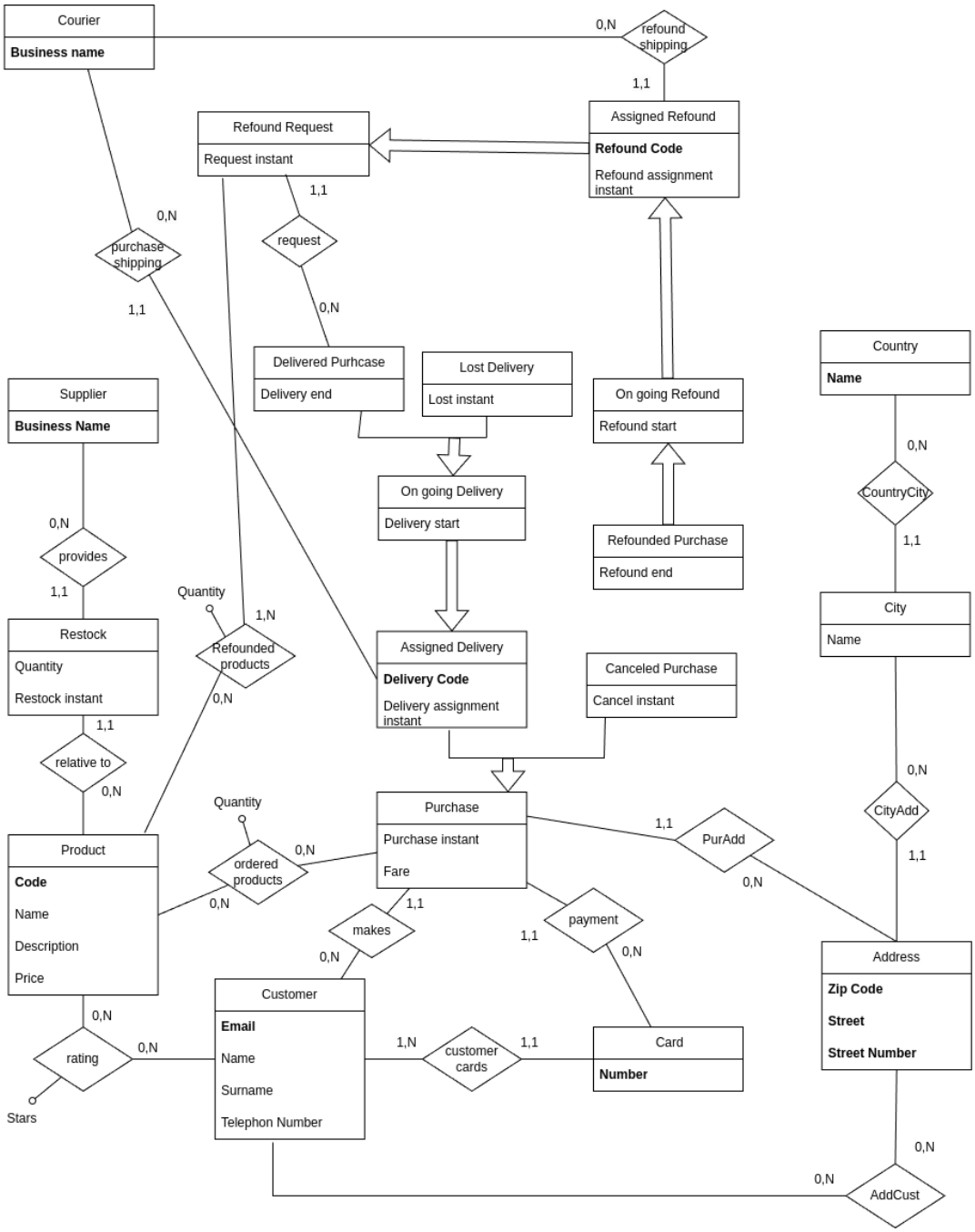
        1.2.1 Codice identificativo

    1.3 Effettuare ordini di vari prodotti attualmente disponibili

        1.3.1 Quantità per ogni prodotto

        1.3.2 Carta del customer
        1.3.3 Indirizzo di spedizione del customer
    1.4 Cancellare un ordine effettuato
    1.5 Effettuare resi gratuiti dei propri ordini
        1.5.1 Prodotti e quantità di prodotto da restituire
    1.6 Ricerca prodotti per nome
    1.7 Recensire prodotti da loro acquistati attraverso punteggio in [1, 5]
2. Fornitore
    2.1 Registrazione alla piattaforma
        2.1.1 Ragione sociale
    2.2 Gestione prodotti (registrazione, modifica e rifornimento)
        2.2.1 Codice identificativo
        2.2.2 Nome
        2.2.3 Descrizione
        2.2.4 Prezzo
    2.3 Statistiche prodotti
        2.3.1 Prodotti maggiormente venduti in un dato arco temporale
3. Trasportatore
    3.1 Registrazione alla piattaforma
        3.1.1 Ragione sociale
    3.2 Gestione spedizioni
        3.2.1 Prendere in carico spedizioni
        3.2.2 Notificare l'avvenuta consegna di ordini
        3.2.3 Notificare l'avvenuto reso di ordini
        3.2.4 Notificare lo smarrimento di un ordine

### 2.1.2 Entity-Relationship diagram



### 2.1.3 Use-case diagram

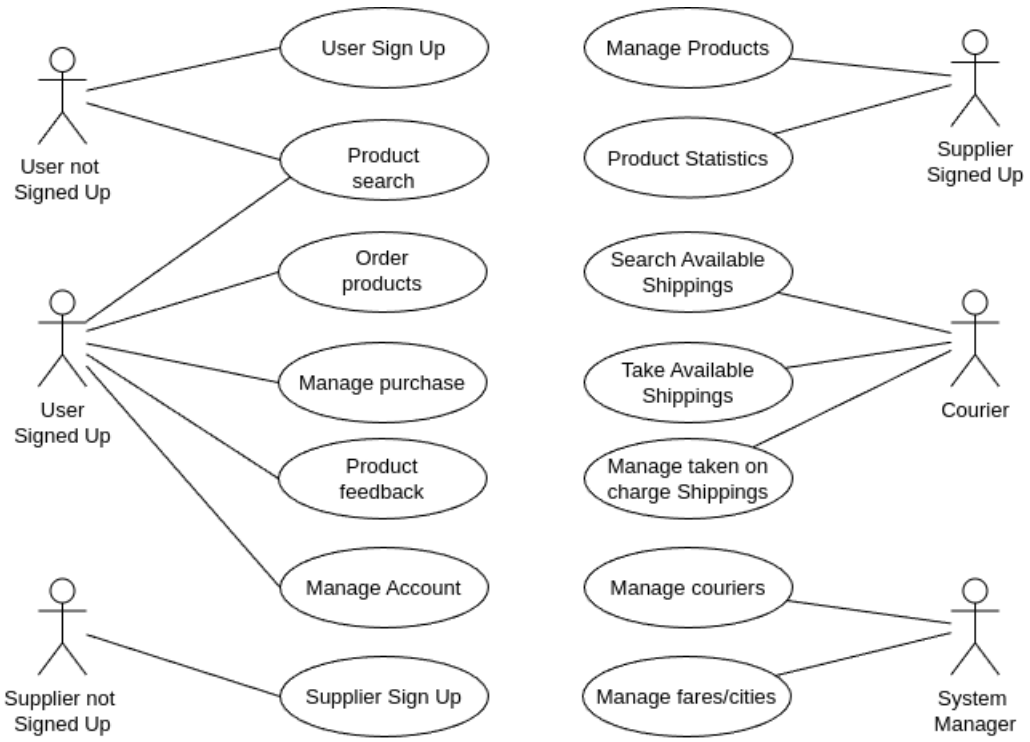



Figure 2.1: Diagramma degli use-case

## 2.2 System requirements

### 2.2.1 Requisiti funzionali e non-funzionali

1. Customer
    - 1.1 Effettuare acquisti
        - 1.1.1 Ogni acquisto deve essere effettuato da un customer, attraverso una delle carte di credito da lui registrate
        - 1.1.2 Ogni acquisto deve essere spedito ad uno degli indirizzi registrati da parte del customer che ha effettuato l'acquisto
        - 1.1.3 Il numero di prodotti che vengono acquistati all'interno di un ordine non deve eccedere le disponibilità dei fornitori registrate all'interno del sistema
        - 1.1.4 I prodotti acquistati all'interno di un ordine possono essere di diversi fornitori
    - 1.2 Cancellare ordini
        - 1.2.1 Un ordine può essere cancellato solamente se non è stato ancora preso in carico da nessun corriere
    - 1.3 Effettuare resi
        - 1.3.1 Il reso di un ordine può essere effettuato solamente dopo aver ricevuto l'ordine all'indirizzo indicato
        - 1.3.2 Il reso di un ordine può essere effettuato solamente da parte del customer che aveva fatto l'acquisto
        - 1.3.3 La quantità di prodotto della quale un customer vuole effettuarne il reso non deve eccedere la quantità di prodotto presente all'interno dell'ordine originale
    - 1.4 Recensire prodotti
        - 1.4.1 Un customer può recensire un prodotto solamente se ha precedentemente effettuato un acquisto che lo conteneva
2. Fornitori
    - 2.1 Gestione prodotti
        - 2.1.1 L'unica caratteristica che è possibile modificare dei prodtotti è la loro descrizione

2.2 Gestione restock

2.2.1 Se un fornitore desidera mettere a disposizione nuove scorte di un dato prodotto deve necessariamente registrare una nuova istanza di restock, e non può modificare le istanze già esistenti

3. Trasportatori

3.1 Gestione spedizioni

3.1.1 Le spedizioni che possono essere prese in carico sono quelle di cui nessun altro trasportatore si è ancora fatto carico

3.2 Notificare l'avvenuta consegna di ordini

3.2.1 È possibile notificare l'avvenuta consegna di un ordine solamente se il pacco non è andato perduto

3.3 Notificare la perdita di ordini

3.3.1 È possibile notificare la perdita di un pacco solamente se questo non è stato ancora consegnato

4. Requisiti non-funzionali

4.1 Il tempo medio di sessione di un client deve essere inferiore a 30 secondi

4.2 Il tempo medio di risposta del server ad una richiesta da parte di un client deve essere inferiore a 30 secondi

### 2.2.2 Architettura del sistema

Il sistema viene progettato seguendo un approccio basato sulla separazione delle richieste ricevute dal sistema. In particolare, l'intero sistema viene decentralizzato in tre server: un server accessibile dai **clienti**, un server accessibile dai **fornitori** ed un server accessibile dai **trasportatori**. Ognuno dei tre server è eseguito parallelamente agli altri due e si occupa di gestire e soddisfare solo ed esclusivamente le tipologie di richieste ad esso designate.

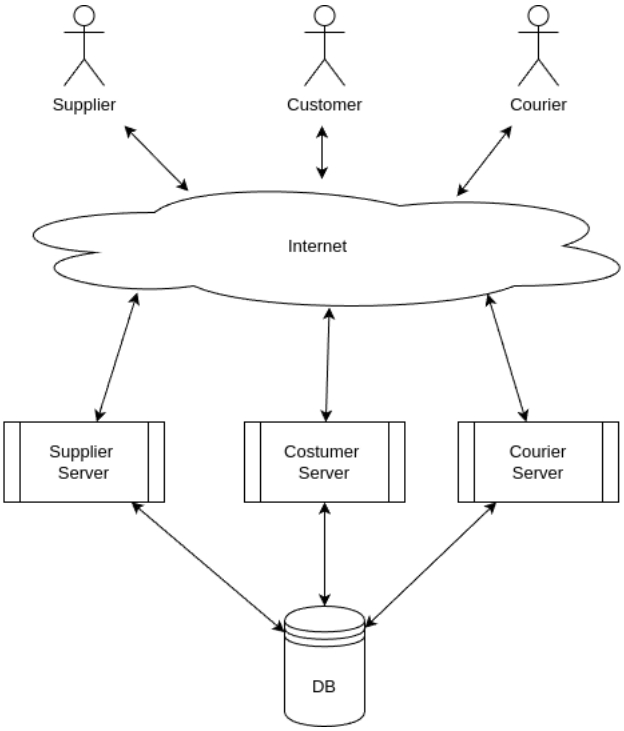
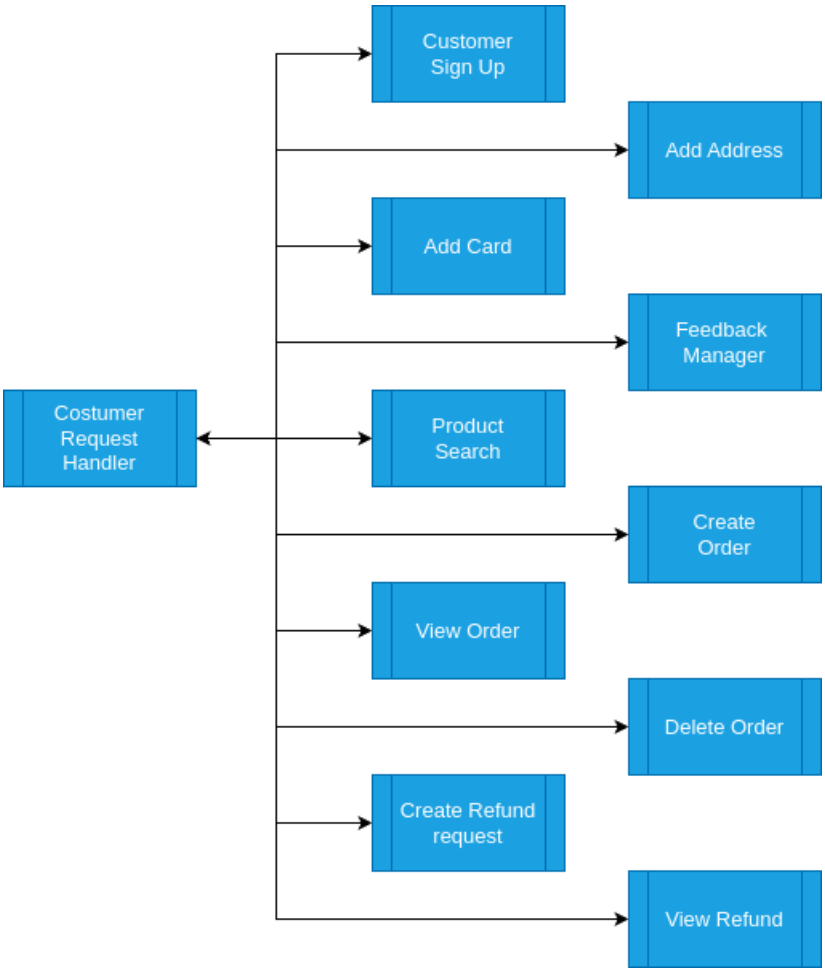
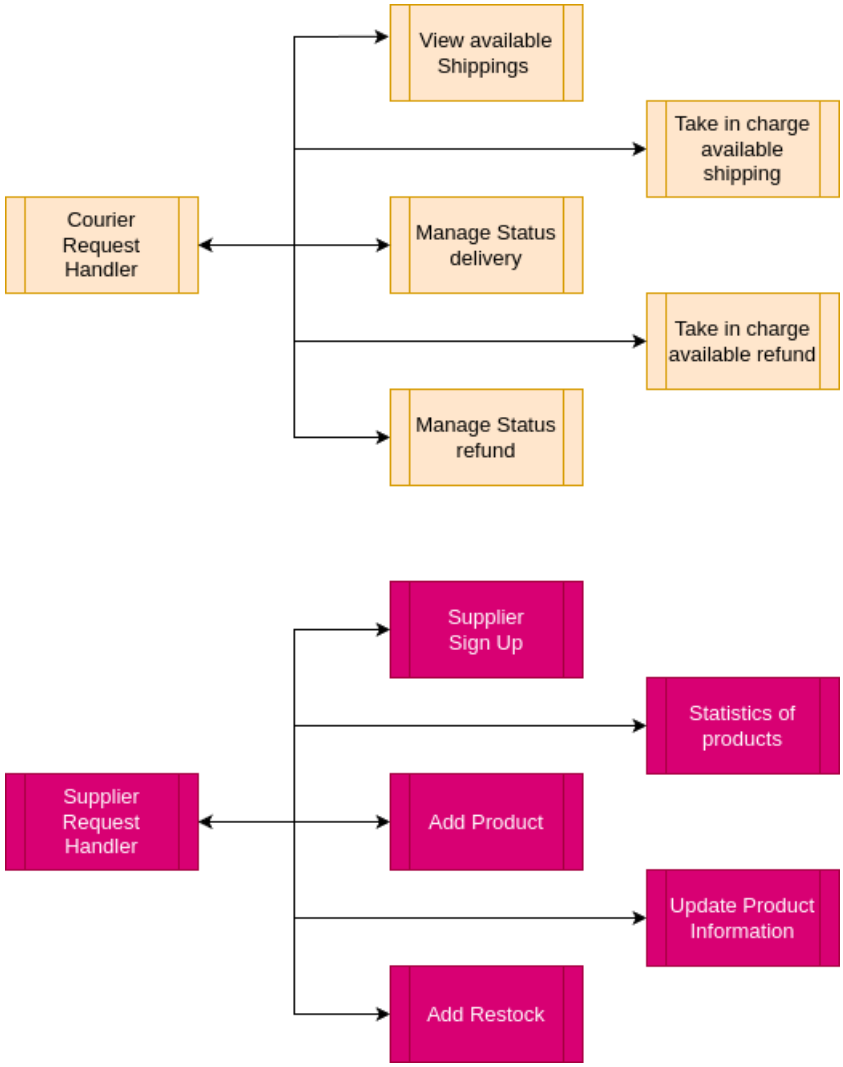



Figure 2.2: Architettura fisica del sistema

Ognuno dei tre server viene a sua volta suddiviso in due componenti, un primo modulo denominato **Server**, addetto alla gestione delle richieste ricevute da e inviate a ognuno degli utenti connessi al sistema, e un secondo modulo denominato **Handler**, addetto alla gestione dello smistamento delle richieste tra vari processi in esecuzione sul server stesso, inviando ogni richiesta tramite uno stream Redis al processo incaricato.

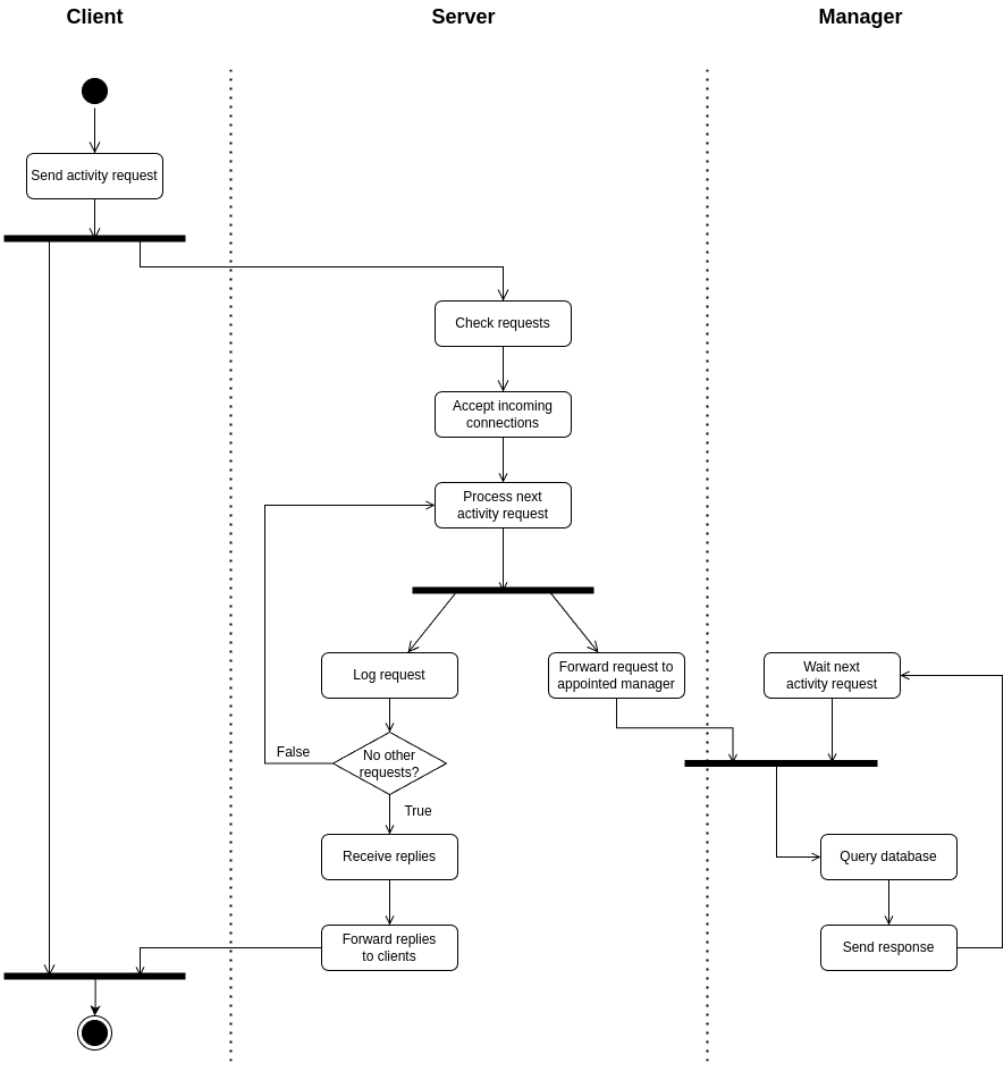
Ciascuno di tali processi viene denominato **Manager**. Ogni Manager è una macchina a stati finiti (vedere sezioni successive) che rimane in attesa di ricevere una richiesta da parte dell'Handler associato, per poi processare tale richiesta interagendo con il database ed infine restituire l'esito all'Handler.




View available Shippings
Take in charge available shipping
Courier Request Handler
Manage Status delivery
Take in charge available refund
Manage Status refund
Supplier Sign Up
Statistics of products
Supplier Request Handler
Add Product
Update Product Information
Add Restock

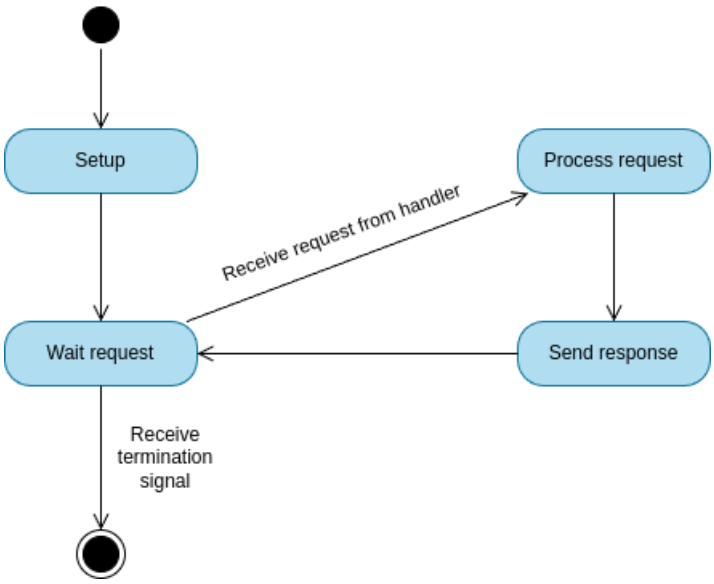
### 2.2.3 Activity diagram

Di seguito viene riportato l'activity diagram relativo all'invio di una richiesta da parte di un client, l'elaborazione di essa e dell'invio della risposta da parte del server. Tale processo risulta comune per tutti i server e i loro Manager, indipendentemente dal tipo di richiesta effettuata dal client. Inoltre, è necessario specificare che, nello scenario descritto, il client abbia già precedentemente stabilito la connessione con il server e che il Manager incaricato sia già attivo e pronto a ricevere e processare una richiesta.



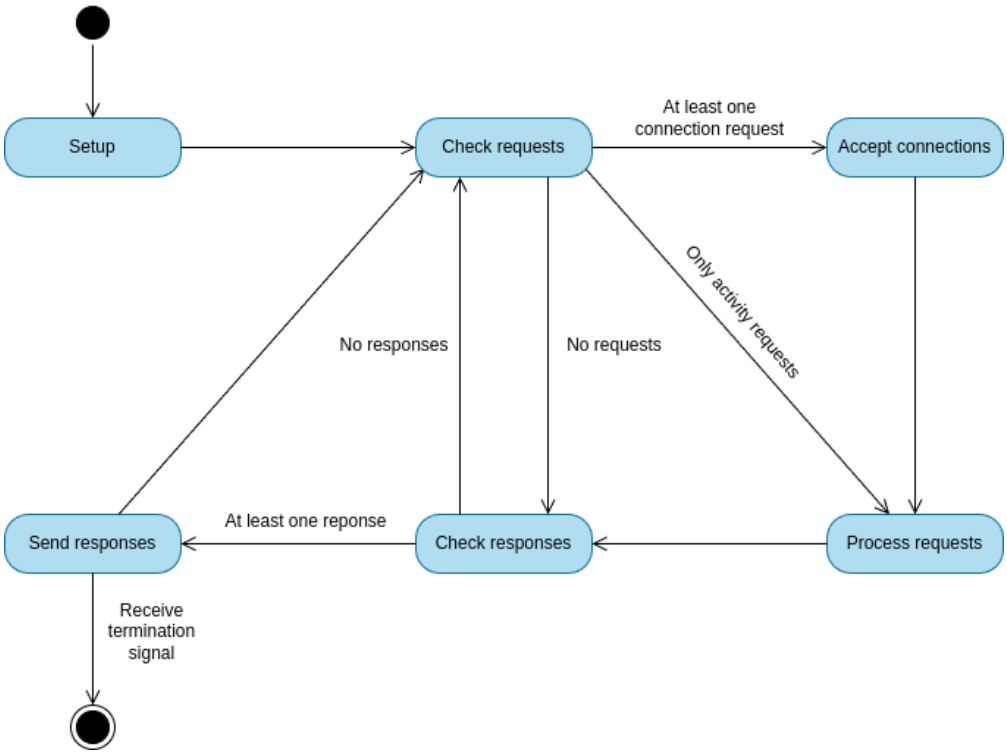
### 2.2.4 State diagram

Come precedentemente accennato, ogni Manager in esecuzione su uno dei tre server corrisponde ad una macchina a stati finiti, la quale alterna tra uno stato di ricezione, stato di elaborazione e stato di invio della risposta. Tale comportamento risulta comune per tutti i Manager, senza alcuna azione aggiuntiva rispetto agli altri.



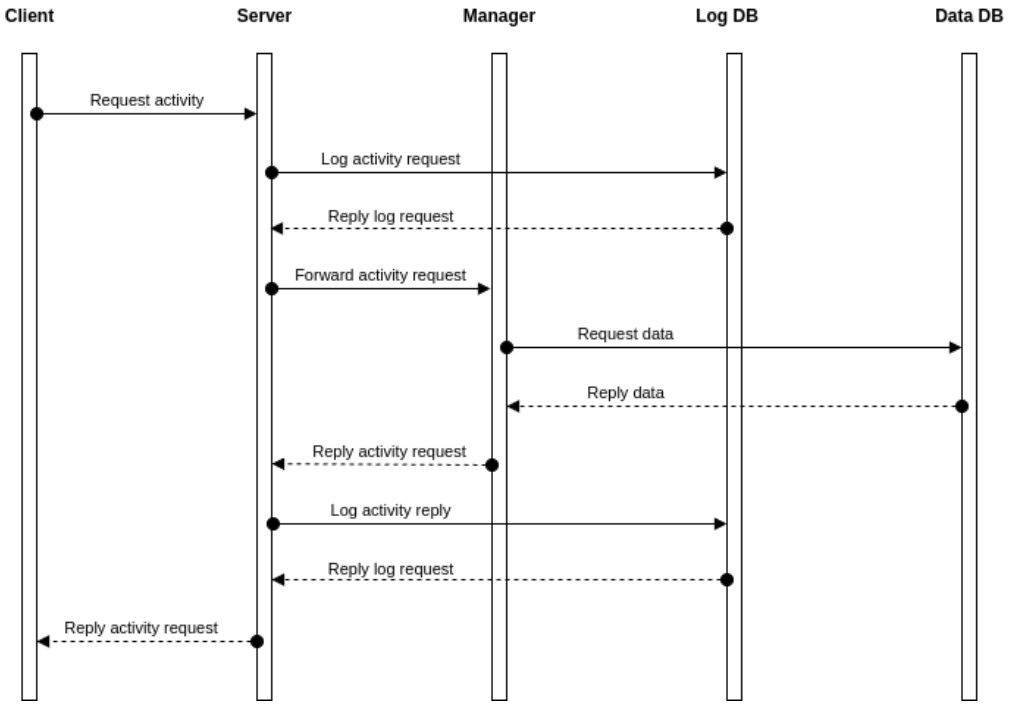

In particolare, è opportuno notare che ogni Manager rimanga in esecuzione permanente fino al momento in cui venga ricevuto un segnale di terminazione da parte del server. Una volta ricevuto tale segnale, ogni richiesta attualmente in elaborazione verrà scartata, senza comunicare alcun riscontro all'Handler associato.

Analogamente, ognuno dei tre server può essere descritto come una macchina a stati finiti. Tuttavia, a differenza dei Manager, ogni server non rimarrà in attesa di una richiesta da parte di un client (dunque non si tratta di un'attesa bloccante). Bensì, nel caso in cui non vi sia alcuna richiesta entrante, il server procederà direttamente con la fase di controllo per eventuali risposte ricevute da parte dei Manager ad esso associati.



### 2.2.5 Message sequence chart

Di seguito viene riportato il message sequence chart relativo all'invio di una richiesta da parte di un client, l'elaborazione di essa e dell'invio della risposta da parte del server. Tale processo risulta comune per tutti i server e i loro Manager, indipendentemente dal tipo di richiesta effettuata dal client. Inoltre, è necessario specificare che, nello scenario descritto, il client abbia già precedentemente stabilito la connessione con il server e che il Manager incaricato sia già attivo e pronto a ricevere e processare una richiesta.



# Chapter 3

# Implementazione del software

## 3.1 Struttura del codice

Trattandosi di tre server distinti, il codice è stato suddiviso in tre sezioni: una sezione `customer`, una sezione `courier` ed una sezione `supplier`. Ciascuna delle tre sezioni contiene al suo interno il codice relativo al proprio handler e i propri manager.

Tuttavia, grazie alla suddivisione modulare tra server e handler, ognuno dei tre server è stato implementato tramite un'unica classe `Server`. Tale classe si interfaccia con i client esterni e con il sotto-modulo handler ad esso associato.

Ciascuno degli handler possiede al suo interno una lista di richieste gestibili, ciascuna corrispondente ad un manager associato al server. Se ad un server viene inviata una richiesta non prevista dal proprio handler viene restituito un messaggio di "Bad Request". Inoltre, tutte le richieste e risposte passanti per un handler vengono loggate all'interno del DB dei log.

A differenza dei server e gli handler, ciascun manager è stato implementato tramite un proprio file. Tale scelta deriva dalla differente conformazione dei manager all'interno delle attività da loro svolti: sebbene ciascuno di essi rispetti il modello di macchina a stati mostrato nelle sezioni precedenti, le operazioni previste da ogni stato cambiano leggermente tra un manager e l'altro, impedendo una generalizzazione completa tramite una singola classe.

Di seguito, vengono riportati gli pseudo-codici dei processi server e manager, incluso il sotto-modulo handler.

```
Process Manager():
    connectToDB();
    connectToRedis();

    repeat untill sigterm is received:

        request = waitForRequest();

        If request is not valid:
            send("bad request");
        else:
            query = convertRequest();
            result = queryDB(query);

            If result is not valid:
                send("bad request");
            else:
                response = formatResponse(result)
                send(response);

Process Server():
    setupHandler();

    repeat until sigterm is received:

        requests = checkRequests();
        acceptIncomingConnections();

        if there are requests:
            foreach activity request:
                logRequest(request);
                isValidRequest = sendRequestToHandler(request);

                if not isValidRequest:
                    sendToClient("bad request");
```

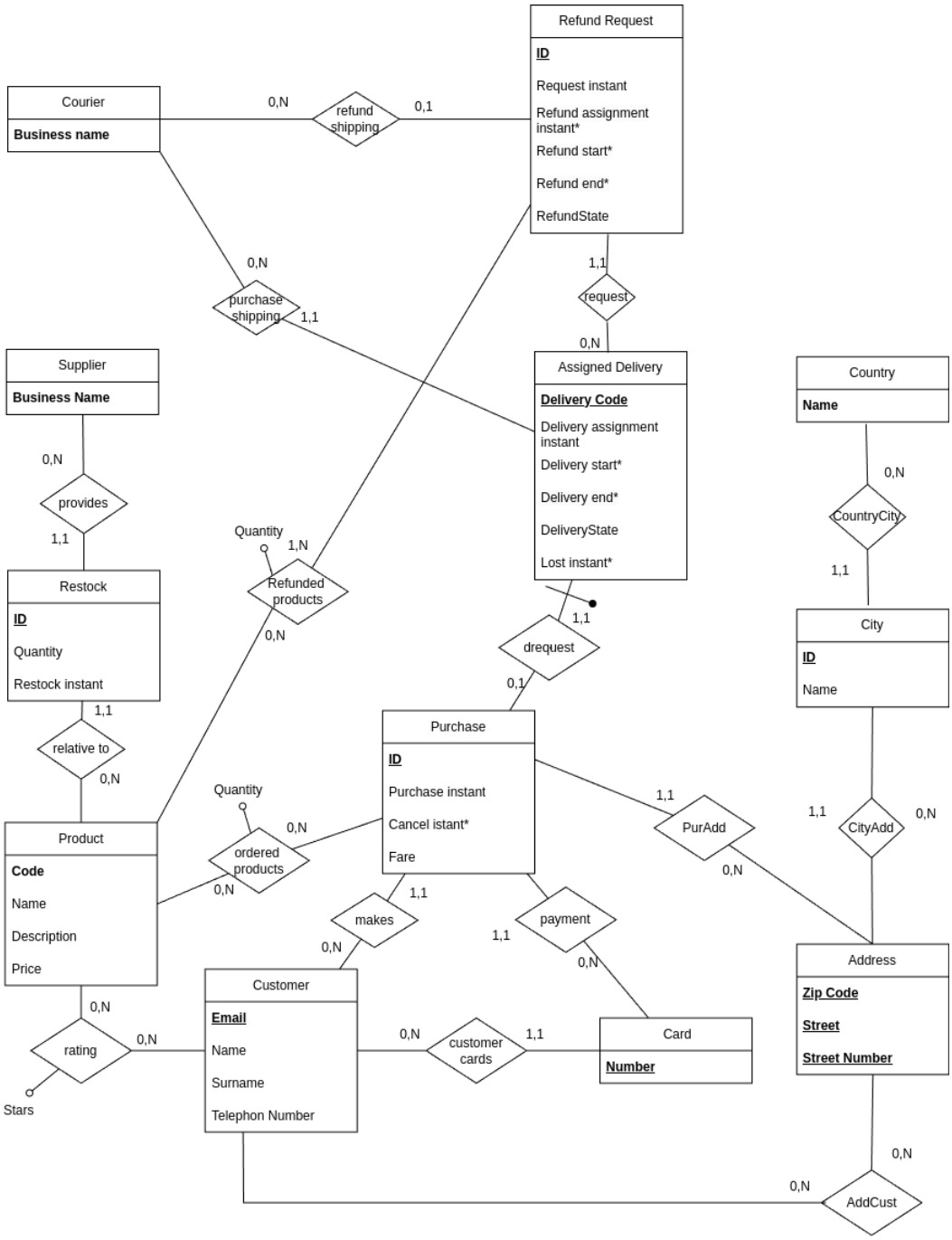
```
        responses = checkResponses();

        if there are responses:
            foreach response:
                sendToClient(response);

Component Handler():
    request = receiveRequestFromServer();
    request_type getRequestType();

    if request_type is not valid:
        return false
    else:
        sendToAppointedManager(request);
        return true
```

## 3.2 Struttura delle connessioni Redis

Ogni Handler istanzia 2 stream Redis per ogni Manager che controlla; ad esempio, per il Manager `Customer Sign Up` – il quale prende il nome di `add-customer` all'interno del programma – il `Customer Request Handler` istanzia uno stream per la comunicazione *dall'Handler verso il manger*, ed un altro *dal Manager verso l'Handler*.

Al fine di tenere traccia dei vari customer che si interfacciano con il sistema, al momento della connessione il server assegna un ID ad ogni client, e tali ID vengono come prima chiave di ogni messaggio scambiato tra Handlers e Managers, permettendo agli Handlers di sapere a chi inoltrare la risposta.

## 3.3 Schema del database



## 3.4 Monitor funzionali

All'interno del software sono stati inseriti dei **monitor funzionali**, implementati tramite dei **trigger** all'interno del database dei dati. Tali trigger fungono da monitor attivi, impenendo che i dati all'interno del database siano inconsistenti tra loro o che non rispettino i vincoli esterni definiti nelle sezioni precedenti.

Ad esempio, sono stati implementati i seguenti tre trigger:

- Trigger per la consistenza tra prodotti recensiti dall'utente e prodotti acquistati da esso

```
-- [V.Customer.ratingProdottiAcquistati]
CREATE OR REPLACE FUNCTION consistent_rating()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM Customer, Product
        WHERE NEW.product = Product.code
        AND Customer.email = NEW.customer
        AND EXISTS (
            SELECT 1 FROM OrderedProducts, Purchase
            WHERE NEW.product = OrderedProducts.product
            AND Purchase.customer = Customer.email
            AND Purchase.id = OrderedProducts.purchase
        )
    ) IS FALSE THEN
        RAISE EXCEPTION 'invalid rating';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER consistent_rating_trg
AFTER INSERT ON rating FOR EACH ROW
EXECUTE FUNCTION consistent_rating();
```

- Trigger per la consistenza tra prodotti totali ordinati e prodotti presenti nello stock del fornitore

```
-- [V.Purchase.restockDisponibile]
CREATE OR REPLACE FUNCTION ordered_product_quantity_consistency()
RETURNS TRIGGER AS $$
BEGIN
    IF (
        WITH restock_quantity AS (
            SELECT SUM(quantity) AS quantity
            FROM Restock
            WHERE product = NEW.product
        )
        SELECT SUM(DISTINCT rq.quantity) >= SUM(op.quantity)
        FROM OrderedProducts AS op, restock_quantity as rq
        WHERE op.product = NEW.product
    ) IS FALSE THEN
        RAISE EXCEPTION 'invalid quantity for order request';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ordered_product_quantity_consistency_trg
AFTER INSERT ON OrderedProducts FOR EACH ROW
EXECUTE FUNCTION ordered_product_quantity_consistency();
```

- Trigger per la consistenza tra prodotti rimborsati e prodotti acquistati dall'utente

```
-- [V.RefoundRequest.prodottiConsistenti]
CREATE OR REPLACE FUNCTION refunded_products_consistency()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM RefundRequest AS rr, AssignedDelivery AS ad,
        Purchase AS p, OrderedProducts as op
        WHERE NEW.refund_request = rr.id
        AND rr.assigned_delivery = ad.delivery_code
        AND ad.purchase = p.id
        AND op.purchase = p.id
        AND op.product = NEW.product
    ) IS FALSE THEN
        RAISE EXCEPTION 'invalid refund request';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER refunded_products_consistency_trg
AFTER INSERT ON RefundedProduct FOR EACH ROW
EXECUTE FUNCTION refunded_products_consistency();
```

**Nota**: ulteriori trigger funzionali possono essere trovati all'interno del file `db-scripts/triggers.sql`.

## 3.5 Monitor non-funzionali

```
while(true) {
    query = "
        SELECT EXTRACT(
            EPOCH FROM AVG(disconnection_instant - connection_instant)
        ) * 1000 as avg
        FROM Client
        WHERE disconnection_instant IS NOT NULL
    ";

    average = log_db.execQuery(query);

    if average <= MAX_CONNECTION_TIME_AVG:
        responseStatus = "SUCCESS";
    else:
        responseStatus = "ERROR";

    query = "
        INSERT INTO
        SessionStatistic(type, end_instant, value, response_status)
        VALUES ('Session', CURRENT_TIMESTAMP, average, responseStatus)
    ";

    log_db.execQuery(query);

    query = "
        SELECT EXTRACT(
            EPOCH FROM AVG(response_instant - request_instant)
        ) * 1000 as avg
        FROM Communication
        WHERE response_instant IS NOT NULL
    ";

    average = log_db.checkQueryResult(query);
```

```
    if average <= MAX_RESPONSE_TIME_AVG:
        responseStatus = "SUCCESS";
    else:
        responseStatus = "ERROR";

    query = "
        INSERT INTO
        SessionStatistic(type, end_instant, value, response_status)
        VALUES ('Response', CURRENT_TIMESTAMP, average, responseStatus)
    ";

    log_db.execQuery(query);

    sleep(60sec);
}
```