UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Documentazione Progetto

Informatica III - Progettazione e Algoritmi

Prof.ssa Patrizia Scandurra

Barbieri Filippo 1054060
Bousaid Acram 1052948
Lorenzo Mazzoleni 1052949

Laurea Magistrale in Ingegneria Informatica
Giugno 2022

Indice

Elenco delle figure

Elenco delle tabelle

1 Iterazione 0

1.1 Introduzione

Lo scopo di questo progetto è quello di creare un sistema distribuito per la gestione della consegna a domicilio della spesa effettuata online da parte dei clienti del supermercato e-Market.
L'applicativo è sviluppato per far fronte alle esigenze di quattro attori:

- L'amministratore del supermercato,
- I corrieri, che hanno il compito di portare a casa la spesa dei clienti,
- I magazzinieri che devono caricare la merce sui camion,
- I clienti che effettuano la spesa online.

Il supermercato ha a disposizione un numero limitato di camion per effettuare le consegne per cui è in grado si effettuare solo un certo numero di viaggi al giorno. Per questo motivo, per migliorare l'efficienza sia in termini di costo che di tempo è stato sviluppato un apposito algoritmo grazie al quale è stato possibile ottimizzare il numero di viaggi effettuato dai corrieri, massimizzando il numero di consegne per ogni viaggio. Inoltre, a causa della pandemia e purtroppo anche della recente guerra, il supermercato ha deciso di adottare una politica di razionamento sugli alimenti tale per cui, per certi prodotti, i clienti non possono acquistare oltre una certa quantità massima. Il focus del progetto è quindi quello di gestire la spesa online effettuata da parte dei clienti ottimizzando le risorse necessarie per effettuare le consegne (camion, tempo).

1.2 Requisiti Funzionali

Il sistema consente all'amministratore del supermercato di sovraintendere agli aspetti principali della gestione dei corrieri e degli ordini effettuati dai clienti. In particolare, inserendo il nome utente e la password l'amministratore può:

- Accedere ad un'area riservata per visualizzare e modificare il razionamento dei vari generi alimentari;
- Accedere ad un'area riservata per visualizzare, aggiungere, modificare e rimuovere i dati di un corriere;
- Accedere ad un'area riservata per visualizzare l'assegnamento degli ordini ai vari corrieri e il loro stato di avanzamento, cioè l'output dell'algoritmo;
- Eseguire il logout.

I corrieri, dopo aver inserito nome utente e password, possono:

- Accedere ad un'area riservata per visualizzare gli ordini a loro assegnati;
- Accedere ad un'area riservata per aggiornare lo stato di consegna dei vari ordini;
- Logout.

I magazzinieri possono:

- Accedere ad un'area riservata per visualizzare gli ordini assegnati ai vari camion per poter caricare la merce;

I clienti, dopo essersi registrati, possono:

- Accedere ad un'area riservata per visualizzare e modificare i loro dati personali;
- Accedere ad un'area riservata per visualizzare i prodotti disponibili del supermercato e aggiungerli al carrello;
- Accedere ad un'area riservata per effettuare l'acquisto dei prodotti inseriti nel carrello;
- Accedere ad un'area riservata per visualizzare lo stato della consegna dell'ordine;
- Eseguire il logout

1.3 Analisi dei casi d'uso

Al fine di procedere ad uno sviluppo efficente, è stato deciso di dividere le specifiche funzionali in tre code di priorità: alta, media, bassa. Nella coda ad alta priorità si trovano i casi d'uso fondamentali al corretto funzionamento dell'applicazione, nella coda a media priorità sono inserti i casi d'uso riguardanti le funzionalità aggiuntive e nella coda a bassa priorità ci sono le funzionalità non strettamente necessarie.

CODICE	**DESCRIZIONE**
UC1	Gestione dei corrieri (visualizzazione, inserimento, cancellazione e modifica)
UC2	Gestione dei prodotti (visualizzazione, inserimento, cancellazione e modifica)
UC3	Razionamento dei prodotti
UC4	Gestione ordini (assegnamento e visualizzazione)

Tabella 1: Casi d'uso ad alta priorità

CODICE	**DESCRIZIONE**
UC5	Registrazione cliente
UC6	Login Cliente
UC7	Logout Cliente
UC8	Visualizzazione ordini per ogni corriere
UC9	Gestione del carrello utente (visualizzazione, inserimento, modifca, cancellazione)

Tabella 2: Casi d'uso a media priorità

CODICE	**DESCRIZIONE**
UC10	Modifica dati utente
UC11	Stato di avanzmanto dell'ordine per ogni cliente

Tabella 3: Casi d'uso a bassa priorità

1.4 Diagramma UML dei casi d'uso

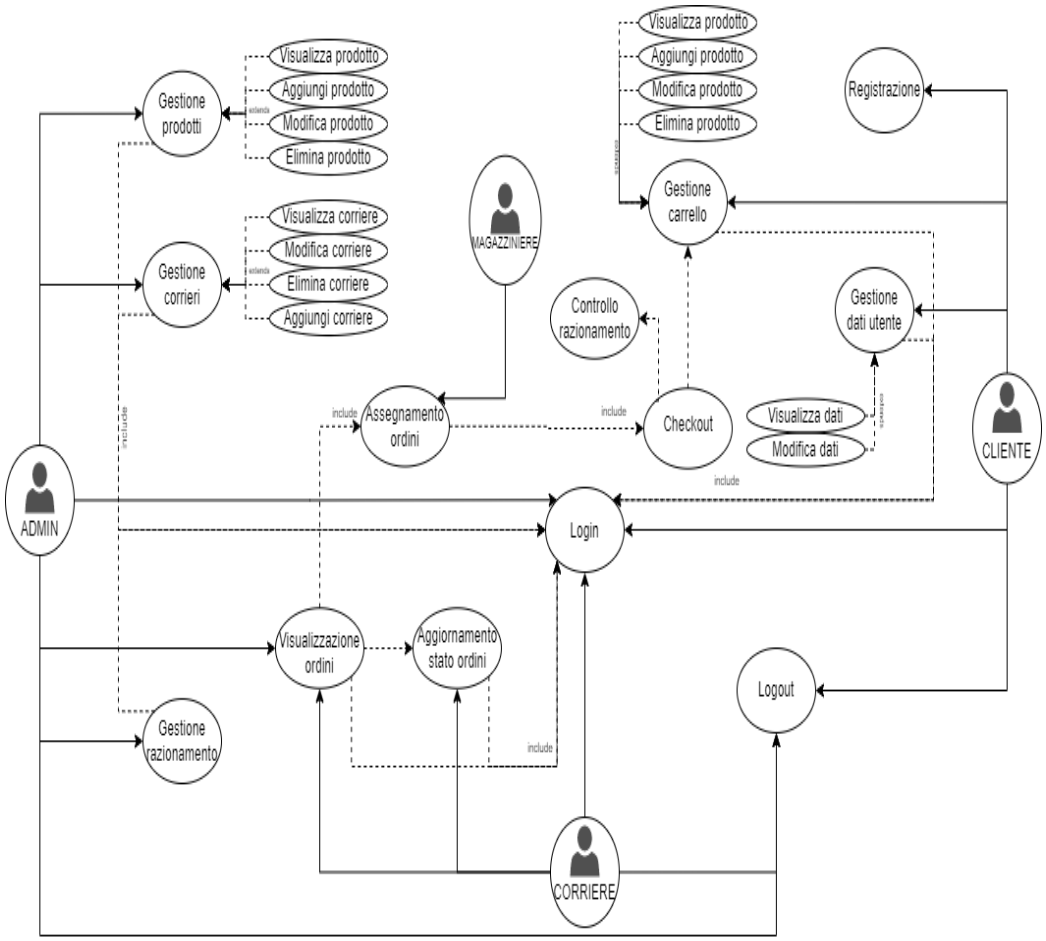

Figura 1: Diagramma UML dei casi d'uso

1.5 Requisiti non funzionali

Il progetto verrà sviluppato tenendo in considerazione anche alcuni requisiti non funzionali, quali la portabilità, l'efficienza e l'usabilità.

Efficienza

Il requisito dell'efficienza è anche l'obiettivo primario del progetto, ovvero la divisione ottimale dei viaggi sui corrieri in modo da ottimizzare i costi ed i tempi di consegna.

Usabilità

L'usabilità è garantita dalla decisione di sviluppare il programma attraverso un'applicazione Android, di facile utilizzo sia per gli utenti che per il proprietario. Inoltre, è possibile accedere al sistema tramite una pagina Web dotata di chiare e semplici interfacce.

Portabilità

Il sistema sarà sviluppato come una Web App il che renderà più semplice la portabilità su un'App Android.

1.6 Pattern di progettazione

Il design pattern scelto per lo svilupppo del progetto è il Model View Presenter (MVP), un pattern lineare composto dai seguenti layer:

- Model: incapsula i dati dell'applicazione, definendo le regole di accesso e modifica
- View: rappresenta l'aspetto grafico dei dati del modello, visualizza i dati e notifica le azioni dell'utente. Solitamente non presenta nessuna logica applicativa, ma ha una funzione solo visuale.
- Presenter: è il layer intermedio tra Model e View, prende i dati dal Model, li elabora e li restituisce alla View

In Figura 2 è riportata la comunicazione tra i tre componenti.

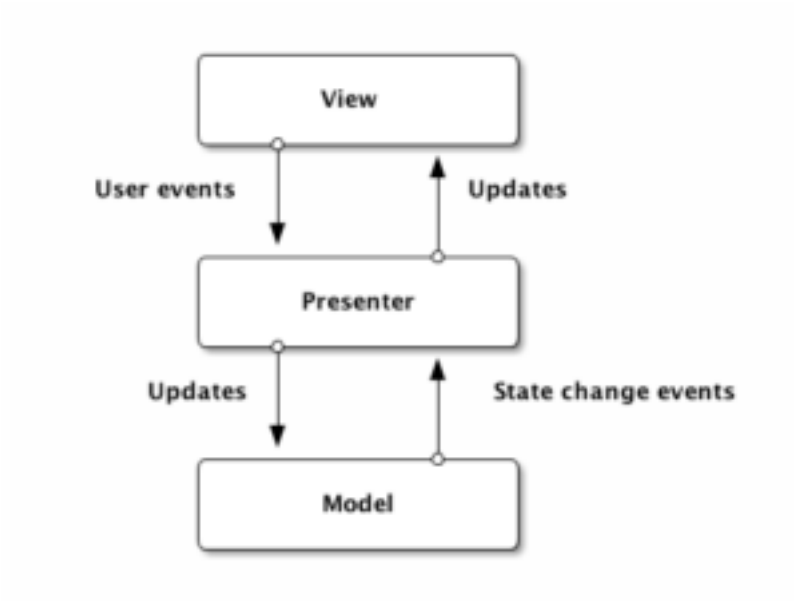

Figura 2: Model View Presenter

1.7 Topologia del sistema

La topologia del sistema, mostrata in Figura 3, evidenzia come il progetto sia stato sviluppato utilizzando un'architettura Three-Tier. L'applicazione è stata dunque suddivisa in tre livelli architetturali, ognuno specializzato in un aspetto del sistema. Questi sono:

- Data, dedicato alla gestione dei dati persistenti;
- Application, che racchiude la logica funzionale;
- Presentation, osssia l'interfaccia utente.

Nello specifico il modulo Data è costituito da un Database gestito con MySQL, all'interno del modulo Application si trova il Web Server usato per gestire la logica funzionale dell'applicazione sviluppata in Java attraverso il framework Spring, e infine, tramite le API esposte dal Server, il modulo Presentation recupera i dati e li rappresenta all'utente (che può essere l'Admin, il Corriere, il Magazziniere o il Cliente) tramite una WebApp realizzata in JavaScript, HTML/CSS o un app Android.

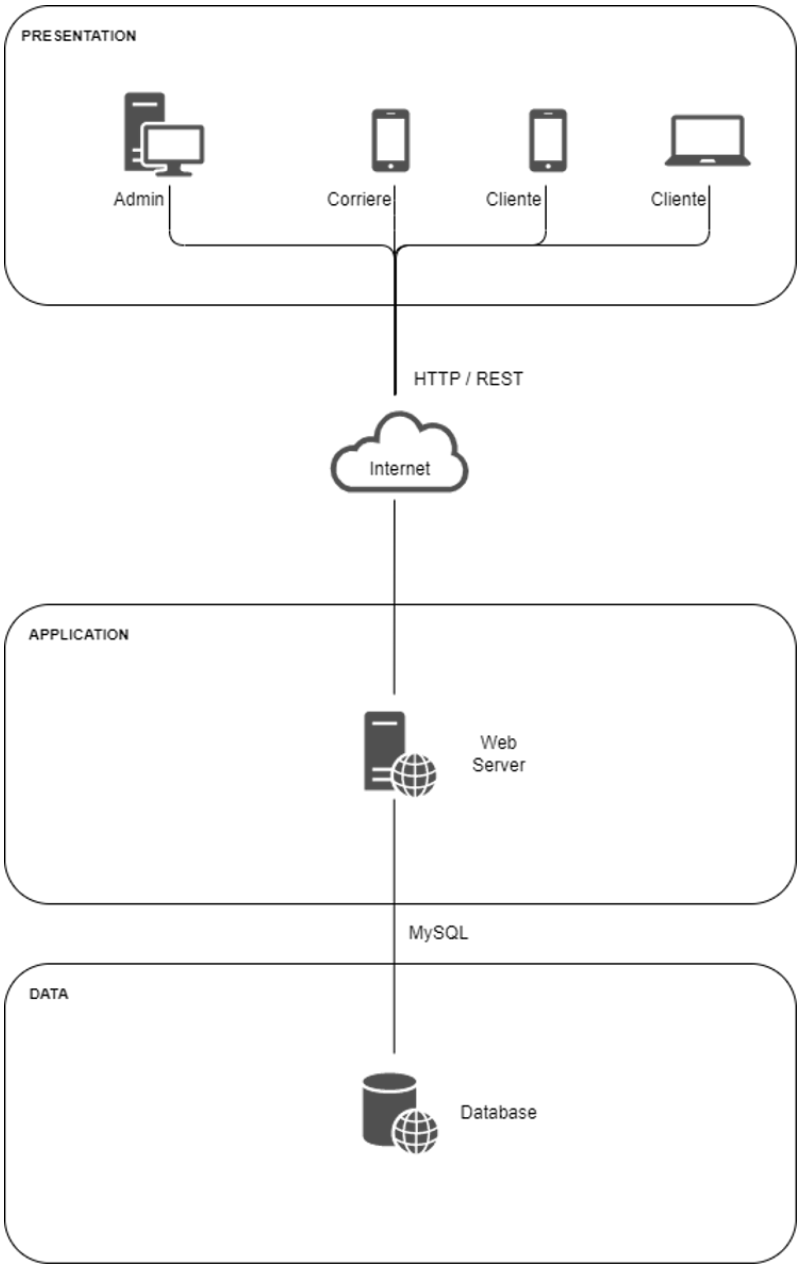

Figura 3: Topologia del sistema

1.8 ToolChain

- Modellazione
  - Use case diagram, class diagram, component diagram, Draw.io
- Implementazione software
  - Linguaggio di programmazione: Java e JavaScript
  - IDE: Eclipse
  - API Development: Java Spring e Postman
  - DBMS: MySQL
  - Interfaccia grafica: HTML e CSS
- Analisi del Software
  - Analisi statica: STAN4J
  - Analisi dinamica: JUnit
- Documentazione, Versioning e gestione del gruppo
  - Versioning: Git e Github, Github desktop e CodeTogether
  - Gestione Team: Google Drive, Google Meet e Trello
  - Documentazione: Latex

2 Iterazione 1

Nella prima iterazione si è deciso di implementare i primi casi d'uso ad alta priorità riportati nella tabella 1 di pagina 7:

- UC1: Gestione corriere
  - UC1.1: Inserisci corriere
  - UC1.2: Visualizza corriere
  - UC1.3: Modifica corriere
  - UC1.4: Elimina corriere
- UC2: Gestione prodotto
  - UC2.1: Inserisci prodotto
  - UC2.2: Visualizza prodotto
  - UC2.3: Modifica prodotto
  - UC2.4: Elimina prodotto
- UC3: Razionamento dei prodotti

Di seguito viene riportata una descrizione testuale per ogni caso d'uso implementato in questa iterazione.

2.1 UC1: Gestione corriere

Breve descrizione: l'amministratore deve avere una visione generale dei corrieri assunti dal supermercato. Perciò, oltre a visualizzarli tutti, deve poter inserire un nuovo corriere , modificare le relative informazioni qualora presentino degli errori o debbano essere aggiornate (es: cambio di stipendio) ed infine deve poterlo eliminare qualora decida di licenziarlo o liquidarlo. Per questo la gestione dei corrieri è suddivisa in quattro casi d'uso:

- UC1.1: Inserimento corriere
- UC1.2: Visualizzazione corrieri

- UC1.3: Modifica corriere
- UC1.4: Eliminazione corriere

Attori coinvolti: Proprietario, Sistema.
Procedimento: il Sistema mostra la pagina home dedicata all'amministratore (*index*) che comprende tre bottoni: *Gestione Corrieri, Gestione Prodotti* e *Assegnamento Ordini*. Cliccando sul primo bottone il proprietario può accedere alla sezione dedicata alla gestione dei corrieri in cui potrà inserire, visualizzare, modificare ed eliminare i vari corrieri.

2.1.1 UC1.1: Inserimento corriere

Breve descrizione: l'amministratore aggiunge un nuovo corriere nel sistema.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore
2. L' amministratore clicca sul bottone *Gestione Corrieri*
3. Il sistema mostra a schermo la relativa pagina (*corriere.html*) in cui è presente un apposito form
4. l'amministratore compila il form e clicca sul bottone *Inserisci*.
5. Il sistema aggiorna la pagina mostrando a schermo il nuovo corriere inserito.

2.1.2 UC1.2: Visualizzazione corrieri

Breve descrizione: l'amministratore visualizza i corrieri inseriti nel sistema e le relative informazioni.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore
2. L' amministratore clicca sul bottone *Gestione Corrieri*

3. Il sistema mostra a schermo la relativa pagina (*corriere.html*) in cui è presente una tabella (sotto il form per inserire un nuovo corriere) nella quale sono riportati tutti i corrieri nel database.

2.1.3 UC1.3: Modifica corriere

Breve descrizione: l'amministratore visualizza i corrieri inseriti nel sistema e può decidere di modificare le loro informazioni.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore
2. L' amministratore clicca sul bottone *Gestione Corrieri*
3. Il sistema mostra a schermo la relativa pagina (*corriere.html*) in cui è presente una tabella nella quale sono riportati tutti i corrieri nel database.
4. L'amministratore clicca sul pulsante *Modifica* del corriere che vuole modificare e compariranno a schermo tre nuovi pulsanti: *Conferma, Cancella* e *Annulla.*
5. L'amministratore aggiorna i campi e infine clicca il pulsante *Conferma.*

2.1.4 UC:1.4 Elimina corriere

Breve descrizione: l'amministratore elimina un corriere salvato nel database.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore
2. L' amministratore clicca sul bottone *Gestione Corrieri*
3. Il sistema mostra a schermo la relativa pagina (*corriere.html*) in cui è presente una tabella nella quale sono riportati tutti i corrieri nel database.

4. L'amministratore clicca sul pulsante *Modifica* del corriere che vuole eliminare e compariranno a schermo tre nuovi pulsanti: *Conferma, Cancella* e *Annulla*.

5. L'amministratore clicca sul pulsante *Cancella* e il sistema elimina il corriere selezionato ricaricando la pagina.

2.2 UC2: Gestione prodotto

Breve descrizione: l'amministratore deve avere una visione generale dei prodotti prensenti nel supermercato. Perciò, oltre a visualizzarli tutti, deve poter inserire un nuovo prodotto , modificare le relative informazioni qualora presentino degli errori o debbano essere aggiornate (es: cambio di prezzo) ed infine deve poterlo eliminare qualora decida di non vendere più quel prodotto. Per questo, la gestione dei prodotti è suddivisa in quattro casi d'uso:

- UC2.1: Inserimento prodotto
- UC2.2: Visualizzazione prodotti
- UC2.3: Modifica prodotto
- UC2.4: Eliminazione prodotto

Attori coinvolti: Proprietario, Sistema.
Procedimento: il Sistema mostra la pagina home dedicata all'amministratore (*index*) che comprende tre bottoni: *Gestione Corrieri, Gestione Prodotti* e *Assegnamento Ordini*. Cliccando sul secondo bottone il proprietario può accedere alla sezione dedicata alla gestione dei prodotti in cui potrà inserire, visualizzare, modificare ed eliminare i vari prodotti.

2.2.1 UC2.1: Inserimento prodotto

Breve descrizione: l'amministratore aggiunge un nuovo prodotto nel sistema.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore

2. L' amministratore clicca sul bottone *Gestione Prodotti*
3. Il sistema mostra a schermo la relativa pagina (*prodotto.html*) in cui è presente un apposito form
4. l'amministratore compila il form e clicca sul bottone *Inserisci*.
5. Il sistema aggiorna la pagina mostrando a schermo il nuovo prodotto inserito.

2.2.2 UC2.2: Visualizzazione prodotti

Breve descrizione: l'amministratore visualizza i prodotti inseriti nel sistema e le relative informazioni.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore
2. L' amministratore clicca sul bottone *Gestione Prodotti*
3. Il sistema mostra a schermo la relativa pagina (*prodotto.html*) in cui è presente una tabella (sotto il form per inserire un nuovo prodotto) nella quale sono riportati tutti i prodotti del database.

2.2.3 UC2.3: Modifica prodotto

Breve descrizione: l'amministratore visualizza i prodotti inseriti nel sistema e può decidere di modificare le loro informazioni.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore
2. L' amministratore clicca sul bottone *Gestione Prodotti*
3. Il sistema mostra a schermo la relativa pagina (*prodotto.html*) in cui è presente una tabella nella quale sono riportati tutti i prodotti del database.

4. L'amministratore clicca sul pulsante *Modifica* del prodotto che vuole modificare e compariranno a schermo tre nuovi pulsanti: *Conferma, Cancella* e *Annulla.*

5. L'amministratore aggiorna i campi e infine clicca il pulsante *Conferma.*

2.2.4 UC:2.4 Elimina prodotto

Breve descrizione: l'amministratore elimina un prodotto salvato nel database.
Attori coinvolti: Proprietario, Sistema.
Procedimento:

1. Il sistema mostra la pagina home dell'amministratore

2. L' amministratore clicca sul bottone *Gestione Prodotti*

3. Il sistema mostra a schermo la relativa pagina (*prodotto.html*) in cui è presente una tabella nella quale sono riportati tutti i prodotti nel database.

4. L'amministratore clicca sul pulsante *Modifica* del prodotto che vuole eliminare e compariranno a schermo tre nuovi pulsanti: *Conferma, Cancella* e *Annulla.*

5. L'amministratore clicca sul pulsante *Cancella* e il sistema elimina il prodotto selezionato ricaricando la pagina.

2.3 UC3: Razionamento dei prodotti

Breve descrizione: il supermercato adotta una politica di razionamento dei prodotti, questo significa che per ciascun prodotto venduto non è possibile comprarne oltre una certa quantità massima. L'amministratore deve quindi essere in grado di fissare e modificare la quantità massima di acquisto per ogni tipologia di prodotto.
Attori coinvolti: Proprietario, Sistema.
Procedimento: ci si riconduce al procedimento del caso d'uso UC2 Gestione

Prodotto al capitolo 2.2 in cui durante le fasi di inserimento e modifica del prodotto è stata aggiunta una nuova voce "*quantità massima*".

2.4 UML Component Diagram

Analizzando i casi d'uso implementati in questa iterazione è stato possibile progettare il Component Diagram mostrato in Figura 4. Il grafico evidenzia le interazioni tra i componenti dal punto di vista delle funzionalità

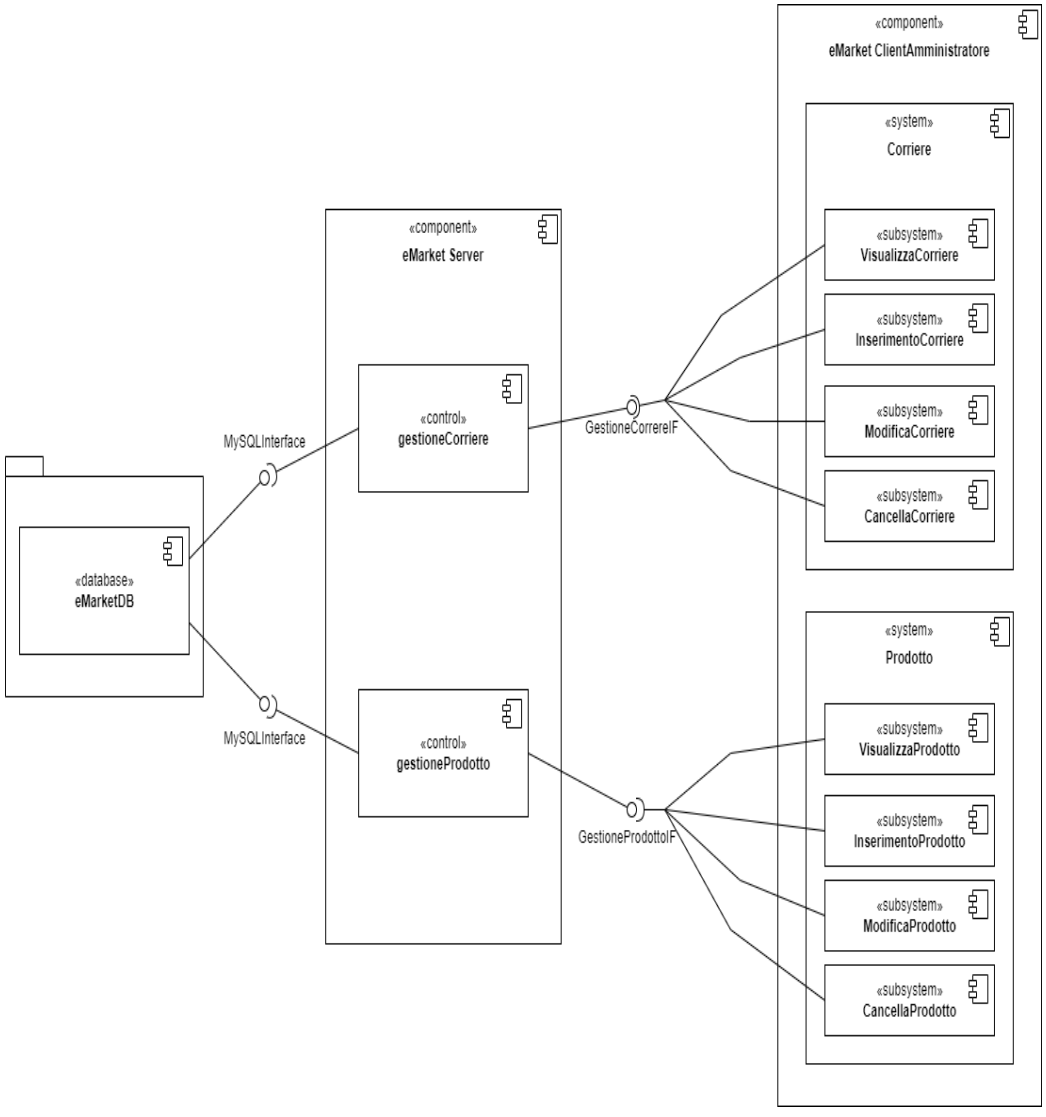

Figura 4: Component Diagram delle parti implementate

2.5 UML Class Diagram per le interfacce

Il class diagram mostrato in Figura 5 riferito all'interfaccia *GestioneCorriereIF* implementata dalla classe *CorriereController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

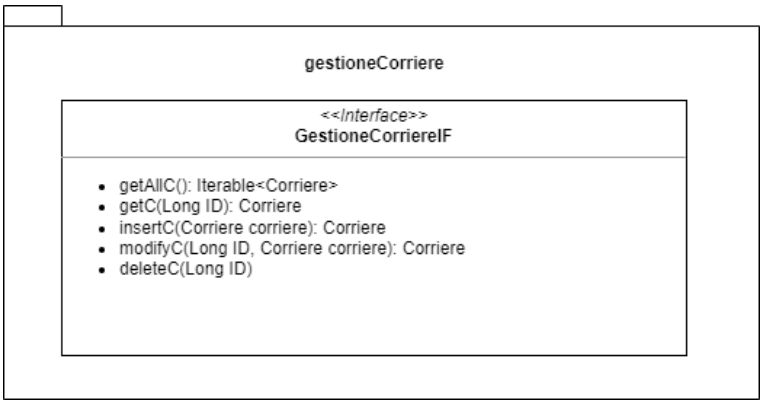

Figura 5: Class Diagram dei metodi interfaccia GestioneCorriereIF

Il class diagram mostrato in Figura 6 riferito all'interfaccia *GestioneProdottoIF* implementata dalla classe *ProdottoController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

Figura 6: Class Diagram dei metodi interfaccia GestioneProdottoIF

2.6 UML Class Diagram per i tipi di dato

Il diagramma in Figura 7 mostra i tipi di dato necessari allo sviluppo dell'applicazione.

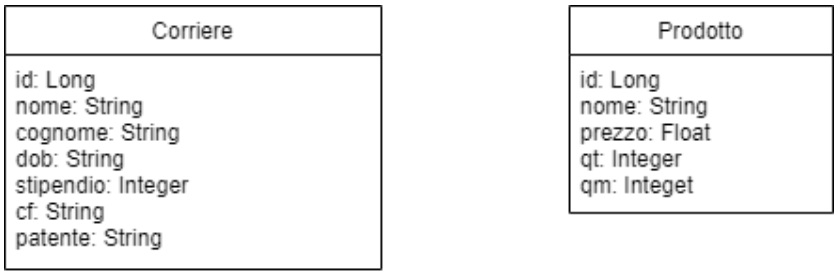

Figura 7: Class Diagram per i tipi di dato

2.7 Testing

2.7.1 Analisi Dinamica

Per effettuare l'analisi dinamica dell'applicazione è stato utilizzato JUnit: un framework di unit testing per il linguaggio di programmazione Java che ha permesso di verificare la corretta esecuzione dei casi di test previsti. Durante l'iterazione 1 sono state create le classi *Corriere.java* e *Prodotto.java* con i relativi metodi get e set su ogni campo della classe, per questo è stata creata una classe: *TestSetandGet.java* che testa il corretto funzionamento di questi metodi. Il relativo codice è mostrato in Figura 8.

```
1  package emarket.test;
2
3  import static org.junit.Assert.*;
6
7  public class TestSetandGet{
8
9      //Prodotto
10     Float f = (float) 2.5;
11     Integer qt = (int) 500;
12     Integer qm = (int) 2;
13     Prodotto p1 = new Prodotto("Pasta", f , qt, qm);
14
15
16     @org.junit.Test
17     public void getNomeTest() {
18         assertEquals("Pasta", p1.getNome());
19     }
20
21     @org.junit.Test
22     public void setNomeTest() {
23         p1.setNome("Pasta integrale");
24         assertEquals("Pasta integrale", p1.getNome());
25     }
26
27     @org.junit.Test
28     public void getPrezzoTest() {
29         assertEquals(f, p1.getPrezzo());
30     }
31
32     @org.junit.Test
33     public void setPrezzoTest() {
34         Float fnew = (float) 3.5;
35         p1.setPrezzo(fnew);
36         assertEquals(fnew, p1.getPrezzo());
37     }
38
```

Figura 8: Codice Unit Test sui metodi Get e Set della classe prodotto.java

In Figura 9 si mostra che il test di unità fatto con JUnit è andato a buon fine.

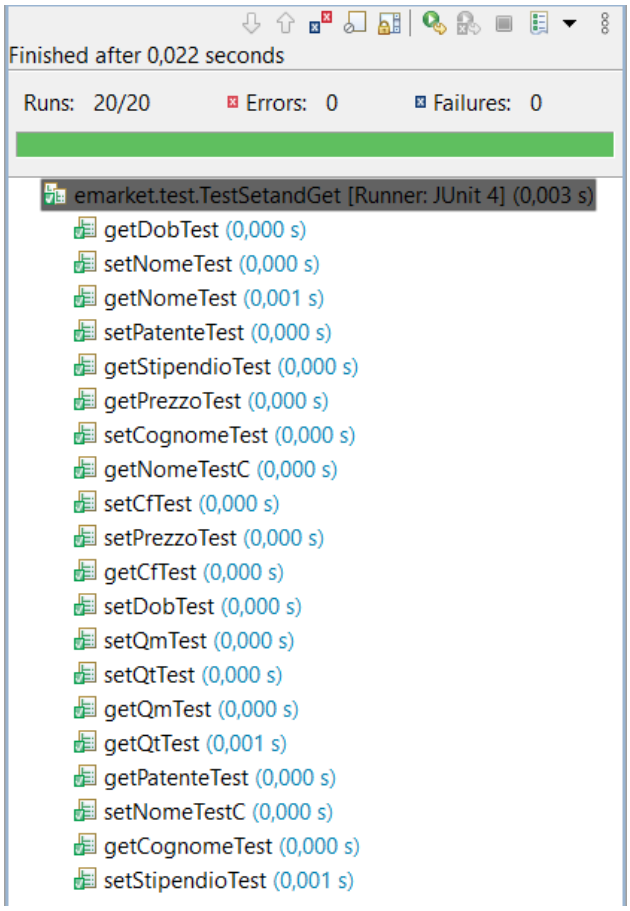

Figura 9: Risultato Unit Test sui metodi Get e Set della classe prodotto.java

2.7.2 Test API tramite Postman

La verifica del buon funzionamento delle API REST esposte dai vari controller è stata effettuata tramite il software PostMan. In particolare è stato testato il funzionamento di una chiamata GET e di una chiamata POST esposte dalla classe *ProdottoController.java* attraverso l'interfaccia *GestioneProdottoiF*

Test chiamata POST Prodotto

La chiamata POST mostrata in Figura 10 consente alla WebApp di salvare nel database le informazioni di un singolo prodotto passando i valori inseriti dall'utente nei relativi campi. Il metodo invocato è *insertP(Prodotto prodotto)*

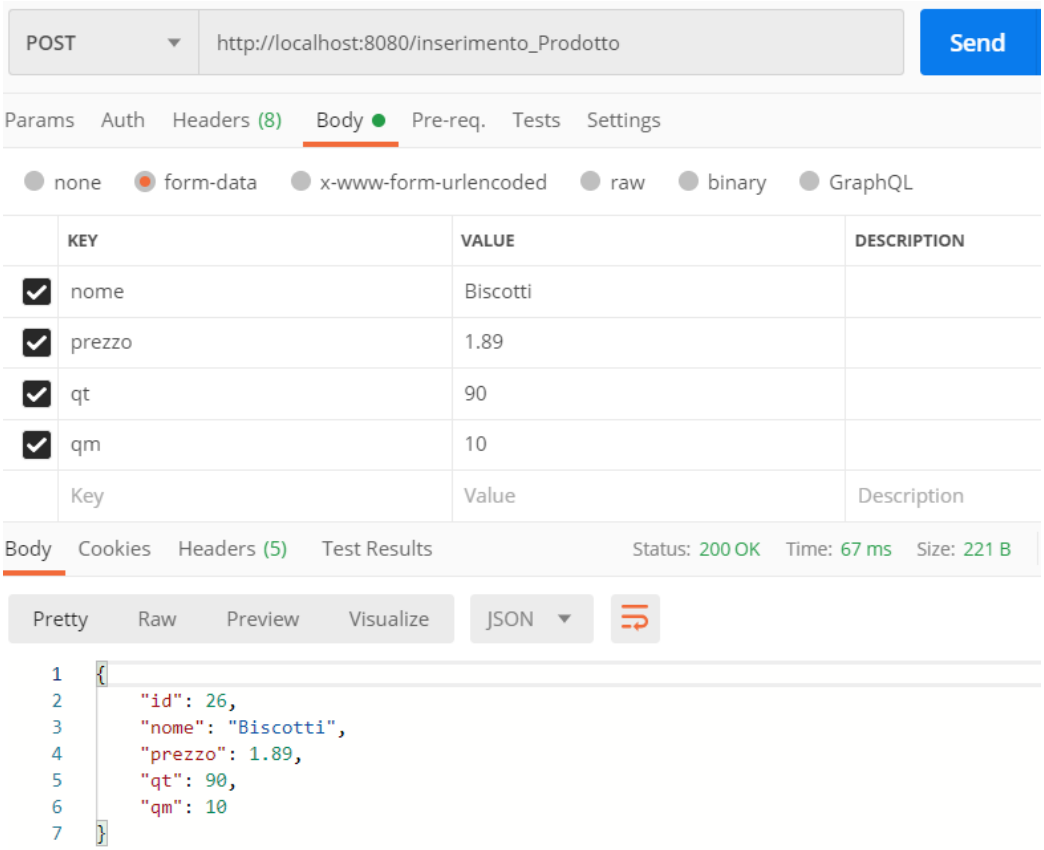

Figura 10: Chiamata POST inserisci prodotto

Test chiamata GET Prodotto

La chiamata GET mostrata in Figura 11 consente alla WebApp di recuperare dal database le informazioni di tutti prodotti. Il metodo invocato è *getAllP()*

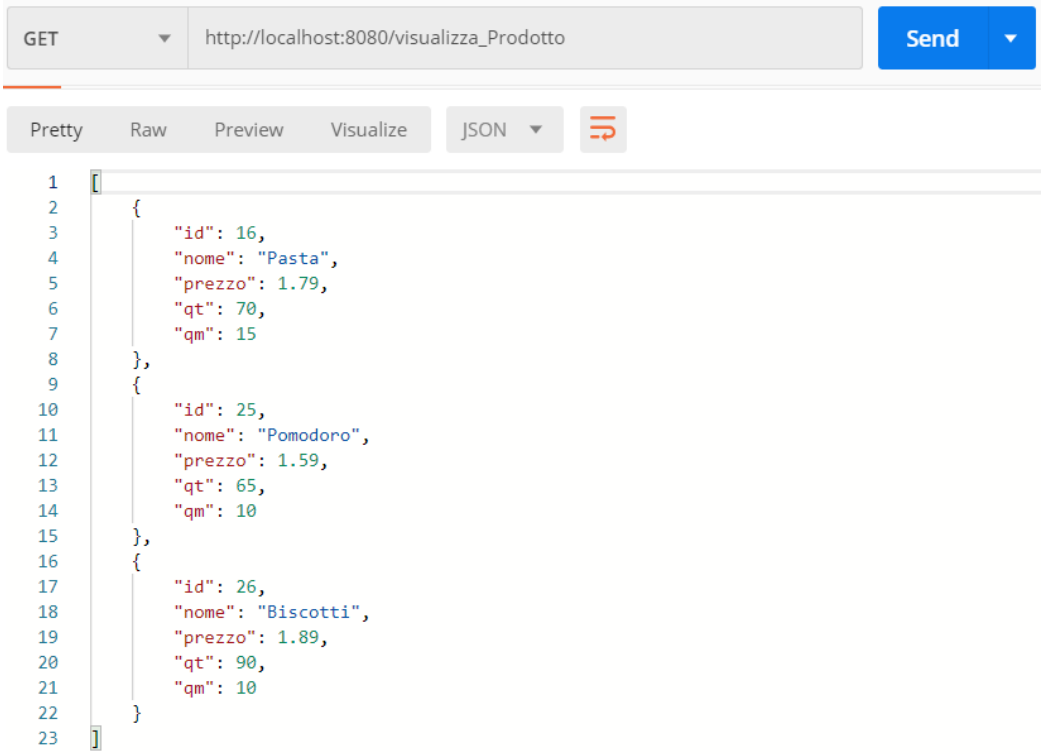

Figura 11: Chiamata GET visualizza prodotti

3 Iterazione 2

Durante questa iterazione il team si è concentrato principalmente sullo sviluppo della parte algoritmica. L'obiettivo è quello di implementare un algoritmo che, dato un vettore di ordini, restituisca all'amministratore il numero di camion necessari per effettuare le consegne. Inoltre per ogni camion viene specificato il percorso che questo deve seguire ed il relativo corriere.
Il caso d'uso implementato corrisponde al caso d'uso UC4 riportato nella tabella 1 dei casi d'uso ad alta priorità. Di seguito è riportata una descrizione di questo caso d'uso.

3.1 UC4: Gestione ordini

Breve descrizione: dopo che i clienti hanno effettuato la spesa online generando un ordine, il sistema deve allocare in modo ottimale gli ordini ai camion disponibili, specificando per ogni campion la tratta da percorrere.
Attori coinvolti: Amministratore, Sistema.
Procedimento

1. Il sistema mostra la pagina *home* all'amministratore
2. L'amministratore clicca sul bottone *Assegnamento Ordini*
3. Il sistema mostra a schermo la relativa pagina (*algoritmo.html*) in cui è prensente l'elenco dei camion e per ogni camion viene specificata la tratta che deve essere percorsa.

L'assegnamento degli ordini ai camion è effettuato da un apposito algoritmo descritto di seguito.

3.1.1 Algoritmo per l'allocazione degli ordini

Lo scopo principale dell'algoritmo è quello di minimizzare il numero di camion utilizzati per effettuare le consengne in modo da ridurre i costi totali di trasporto. Come prima cosa è stata definita la mappa della città tramite un apposito grafo, riportato in Figura 12.

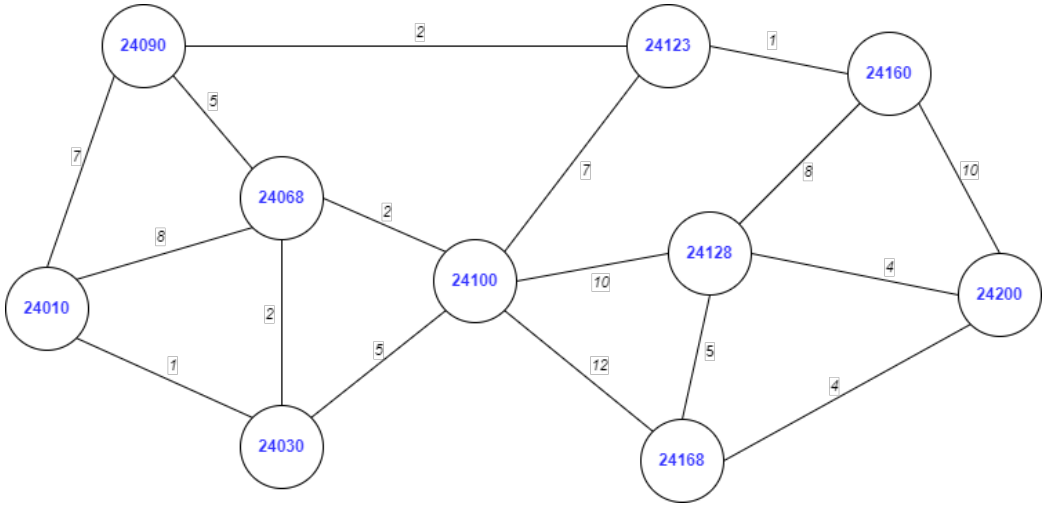

Figura 12: Grafo della città

Ogni nodo è caratterizzato da due etichette:

- Il CAP
- Il peso che rappresenta il numero di ordini da portare in quel nodo.

Ogni arco è composto da due etichette:

- il costo in km da sostenere per spostarsi lungo l'arco
- il flusso che indica il numero degli ordini che passano lungo quell'arco.

Nel nodo con CAP *24100* è presente il supermercato da cui vengono spediti tutti gli ordini.
Il problema per come è stato modellizzato è un problema di Min Cost Flow ossia un problema di decisione e di ottimizzazione che consiste nel determinare la distribuzione del flusso sulla rete nella quale:

- ogni origine (il supermercato) invia tutta la quantità disponibile (gli ordini della giornata).
- ogni destinazione (i vari nodi della rete con peso diverso da 0) riceve tutto il flusso richiesto
- il costo complessivo è minimo.

Passi dell'algoritmo

1. Il primo passo dell'algoritmo consiste nel pesare opportunamente i nodi del grafo. Per farlo vengono prelevati dal database tutti gli ordini da evadere e i nodi vengono pesati nel seguente modo:

   - il peso del nodo sorgente diventa pari all'opposto del numero di ordini da evadere
   - il peso dei singoli nodi è pari al numeri di ordini da consegnare in quel nodo

2. Utilizzando le classi *MinimumCostFlowProblemImpl* e *CapacityScaling-MinimumCostFlow* forniti dalla libreria *jGraphT* viene calcolato il flusso di costo minimo del grafo, ovvero per ogni arco viene speficiato il flusso (il numero di ordini) che lo attraversa.

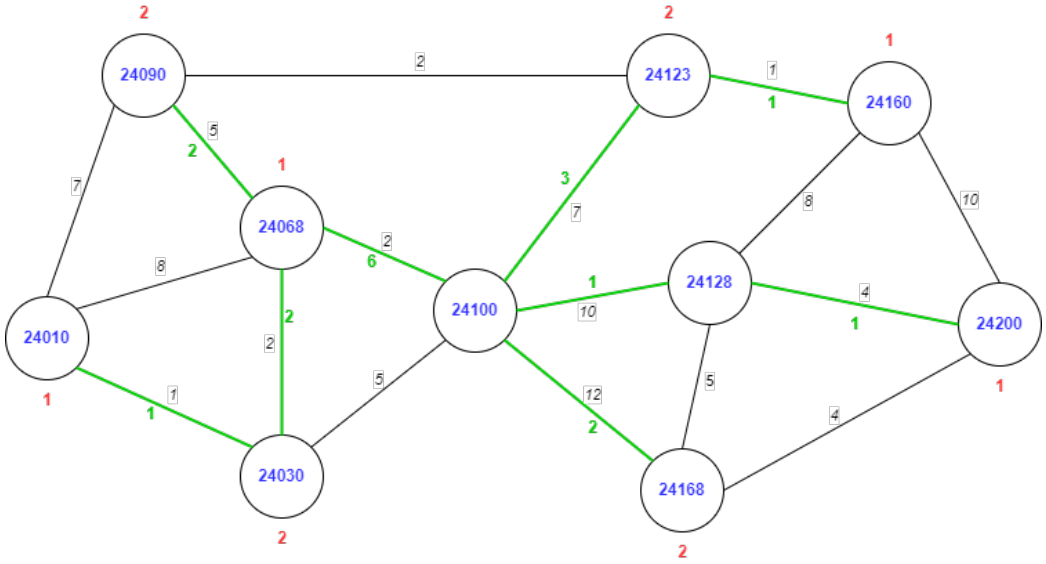

Figura 13: Soluzione del problema di flusso di costo minimo iniziale

3. Partendo dal nodo sorgente cerco l'arco con il flusso maggiore diverso da 0 percorribile (emax), ossia che possa essere percorso senza superare il limite dei km giornalieri. Mi sposto sul nodo che lo collega aggiungendolo al camion. Aggiorno il contatore dei km percorsi, azzero il

flusso dell'arco attraversato e azzero il peso del nodo raggiunto. Proseguo in questo modo fino a quando non è possibile trovare un ulteriore emax, ovvero sono arrivato ad un nodo che ha tutti i suoi archi uscenti con flusso pari a 0.

4. Ora cerco un nodo adiacente alla posizione corrente con peso > 0 che sia percorribile.
   (a) Se esiste mi sposto in quel nodo, aggiorno il contatore dei km percorsi e azzero il peso sul nodo raggiunto.
   (b) Se non esiste faccio tornare il camion alla sorgente aggiornando i km percorsi.
5. Se il camion ha ancora km percorribili allora ricomincio dal passo *3* altrimenti utilizzo un nuovo camion e ricomincio l'algoritmo dal passo *2*.

In Figura 14 è mostrato il risultato finale dell'algoritmo

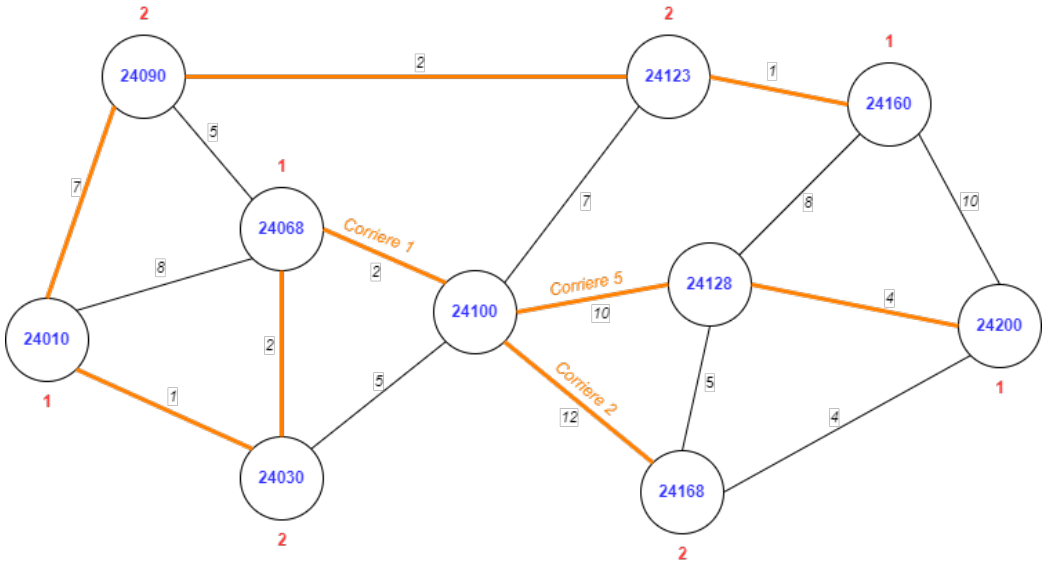

Figura 14: Risultato finale dell'algoritmo

Pseudocodice

In Figura 15 e 16 è riportato lo pseudocodice dell'algortimo con l'analisi di complessità. Nell'eseguire l'analisi di complessità si considera il numero di nodi del grafo pari a n e il numero di archi pari a m. La complessità dell'algoritmo è mostrata di seguito.

$$O((m+n)n\log n) \tag{1}$$

L'espressione può essere riscritta nel seguente modo nel caso in cui il numero di archi è molto maggiore al numero di nodi ossia $m \gg n$

$$O((mn\log n) \tag{2}$$

L'elemento principale a cui corrisponde questa elevata complessità è la chiamata della funzione *getMinimumCostFlow()* fornita dalla libreria *jGraphT* usata per calcolare il flusso di costo minimo del problema.

Figura 15: Pseudocodice parte 1

```
verificaPesoNodi() -> bool
    foreach vertice in ListaVerticiGrafo do                    O(n)
        if( peso(vertice) not 0) then
            return true
        endif
    endfor
    return false


trovaFlussoMassimo(Vertice posizione, double km) -> Arco        O(m)
    flusso max <- null
    foreach flusso in flussi do
        if(vincolo nodo posizione and vincolo distanza and max < flusso ) then
            max <- flusso
        endif
    endfor
    return max.arco


azzeraFlusso(Arco arco)
    foreach f in flussi do                                     O(m)
        if (arco = f.arco) then
            flusso(arco) <- 0.0
        endif
    endfor


azzeraVertice(Vertice posizione)
    int peso <- 0
    foreach vertice in ListaVerticiGrafo do                    O(n)
        if( nome(vertice) = nome(posizione)) then
            peso <- peso(posizione)
            peso(posizione) <- 0.0
        endif
    endfor
    foreach vertice in ListaVerticiGrafo do                    O(n)
        if( nome(vertice) = "24100") then
            peso(vertice) <- peso(vertice) - peso
        endif
    endfor


trovaVertice(Vertice posizione, double km) -> Arco              O(n)
    foreach vertice in ListaVerticiGrafo do
        if(vincolo nodo posizione and vincolo distanza and vincolo nodo pesi) then
            return arco
        endif
    endfor
    return null
```

Figura 16: Pseudocodice parte 2

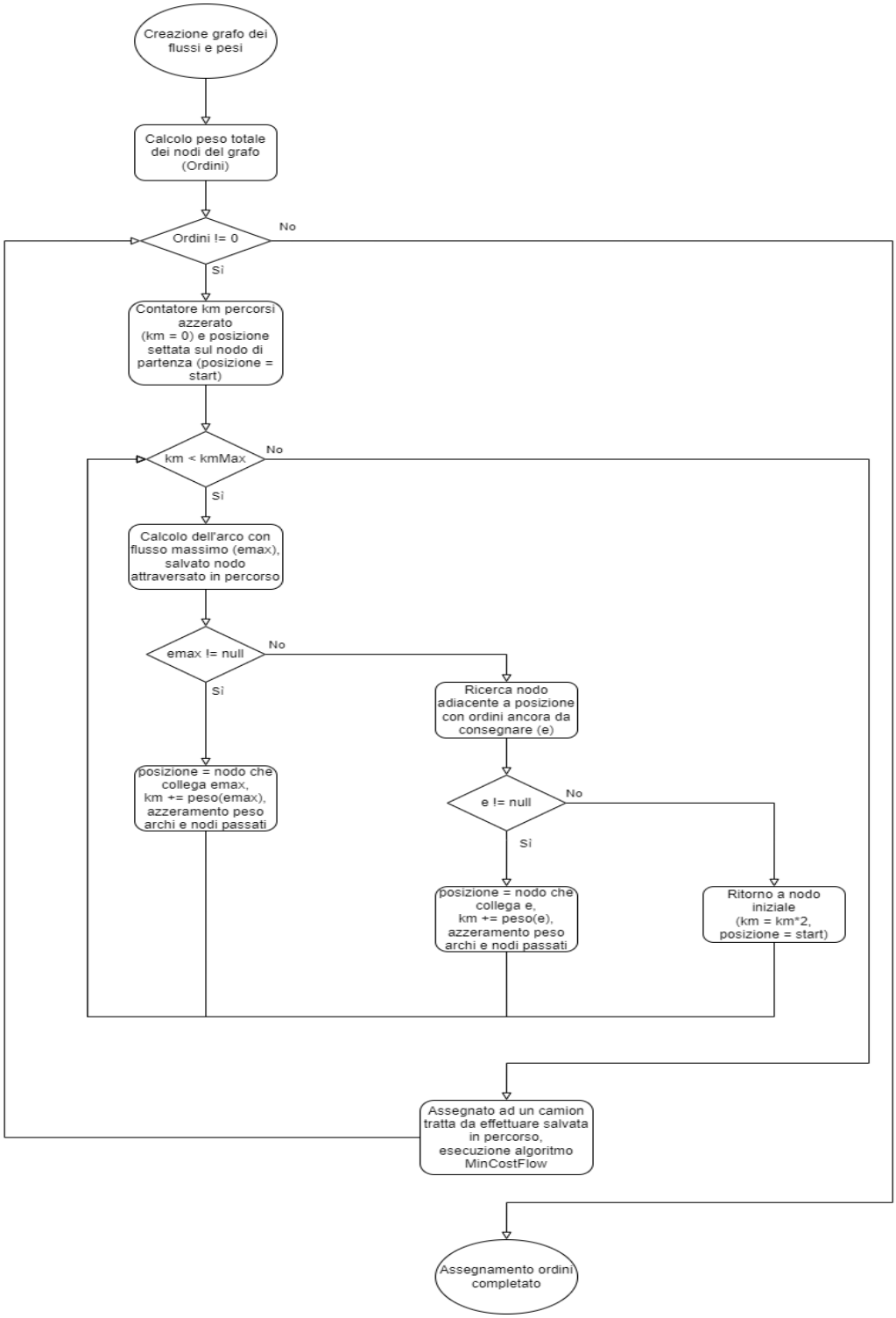

Figura 17: Flow chart dell'algoritmo

3.2 UML Component Diagram

Il caso d'uso implementato in questa iterazione è stato aggiungo al Component Diagram dell'iterazione precedente (Figura 4). Il nuovo diagramma è mostrato in Figura 18. In particolare in blu sono riportati i componenti aggiunti in questa iterazione.

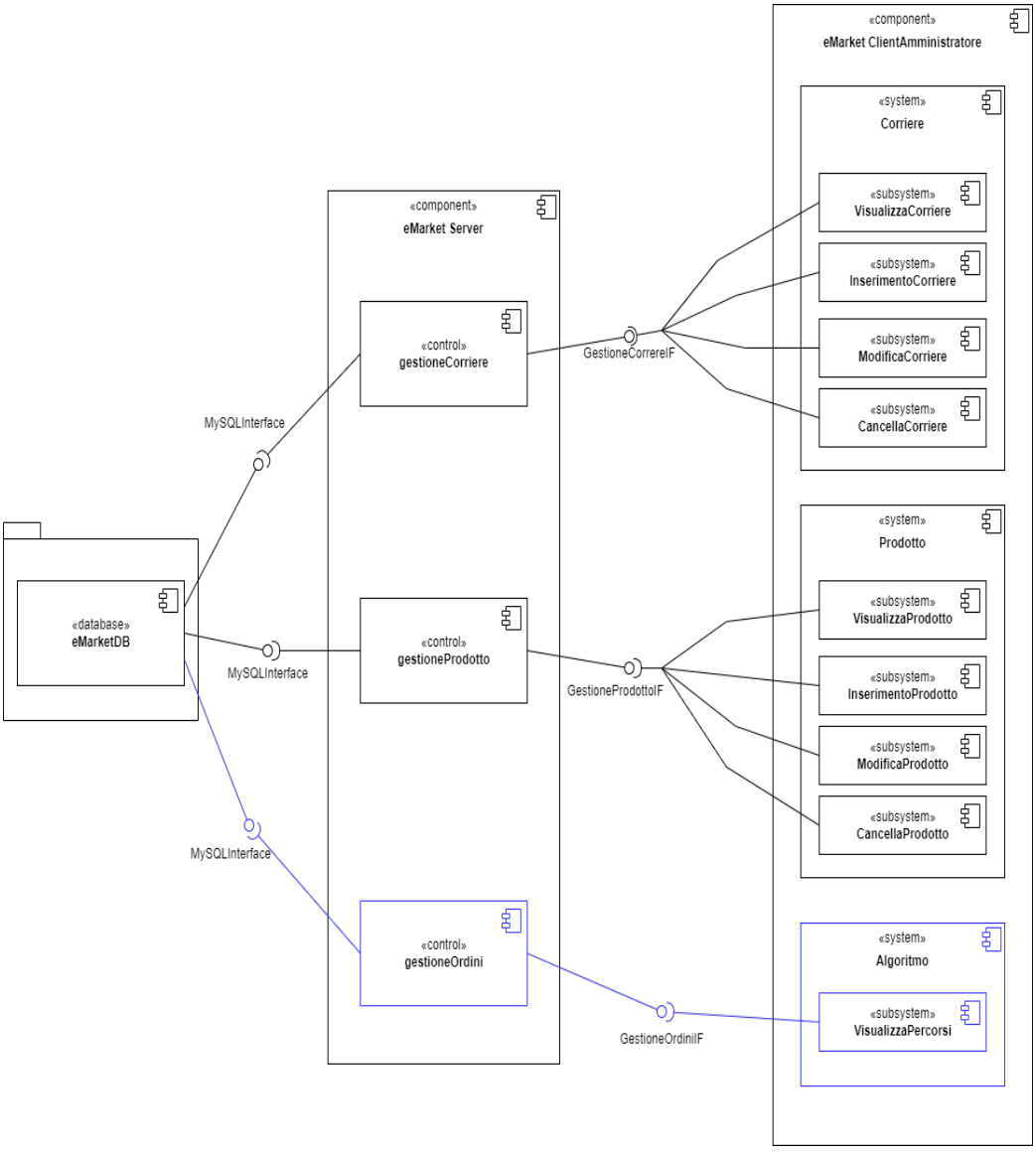

Figura 18: Component Diagram delle parti implementate

3.3 UML Class Diagram per le interfacce

Il class diagram mostrato in Figura 19 riferito all'interfaccia *gestioneOrdiniIF* implementata dalla classe *AlgoritmoController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

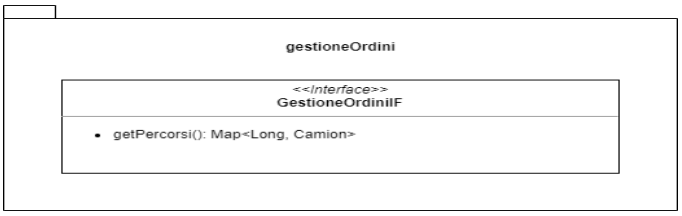

Figura 19: Class Diagram dei metodi dell' interfaccia gestioneOrdiniIF

3.4 UML Class Diagram per i tipi di dato

La classe *Algoritmo* introdotto in questa iterazione, è stata inserita nel Class Diagram per tipo di dato dell'iterazione precedente (Figura 7). Il nuovo diagramma, mostrato in Figura 20, mostra anche le relazioni che si sono instaurate tra i vari tipi di dato.

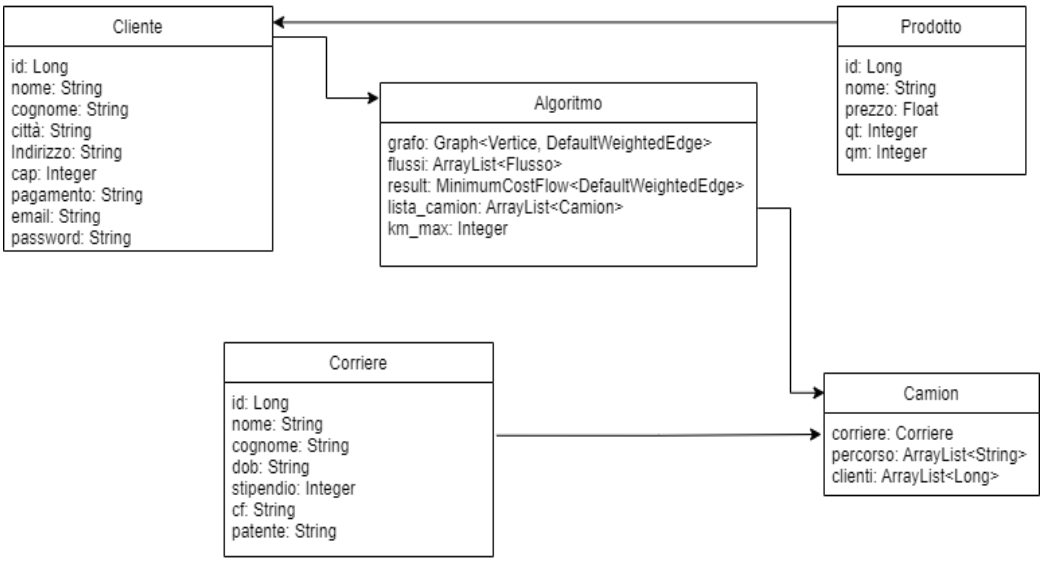

Figura 20: Class Diagram dei tipi di dato

3.5 Testing

3.5.1 Analisi Dinamica

Come per l'iterazione precedente è stato utilizzato JUnit per verificare la correttezza dei metodi implementati nello sviluppo dell'algoritmo. Il relativo codice è mostrato in Figura 21.

```
public class TestAlgoritmo {

    // Test dell'algoritmo
    AlgoritmoController ac = new AlgoritmoController();

    @Test
    public void assegnamentoTest() {
        assertNotNull(ac.getPercorsi());
    }


}
```

Figura 21: Codice Unit Test per la classe Algoritmo.java

In Figura 22 si mostra che il test di unità è andato a buon fine.

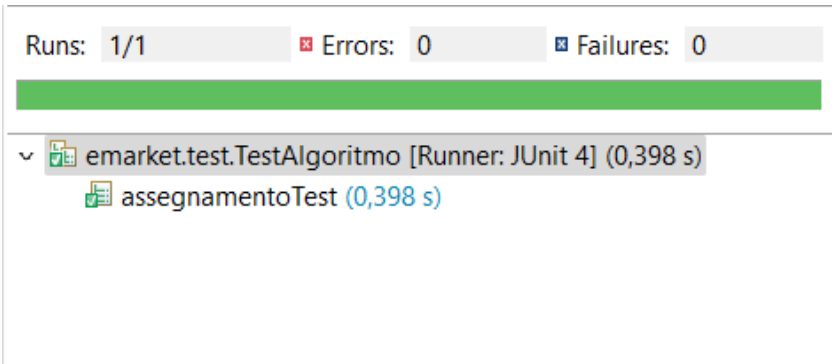

Figura 22: Risultato Unit Test per la classe Algoritmo.java

3.5.2 Test API tramite Postman

Anche in questa iterazione è stato utilizzato il software PostMan per verificare la corretta esecuzione delle chiamate REST API. In particolare la chiamata GET mostrata in Figura 23 consente di recuperare il risultato dell'algoritmo, infatti restituisce i camion necessari per effettuare le consegne e per ognuno specifica il percorso da seguire ed il relativo corriere oltre ai clienti da evadere. Il metodo invocato è *getPercorsi()*.

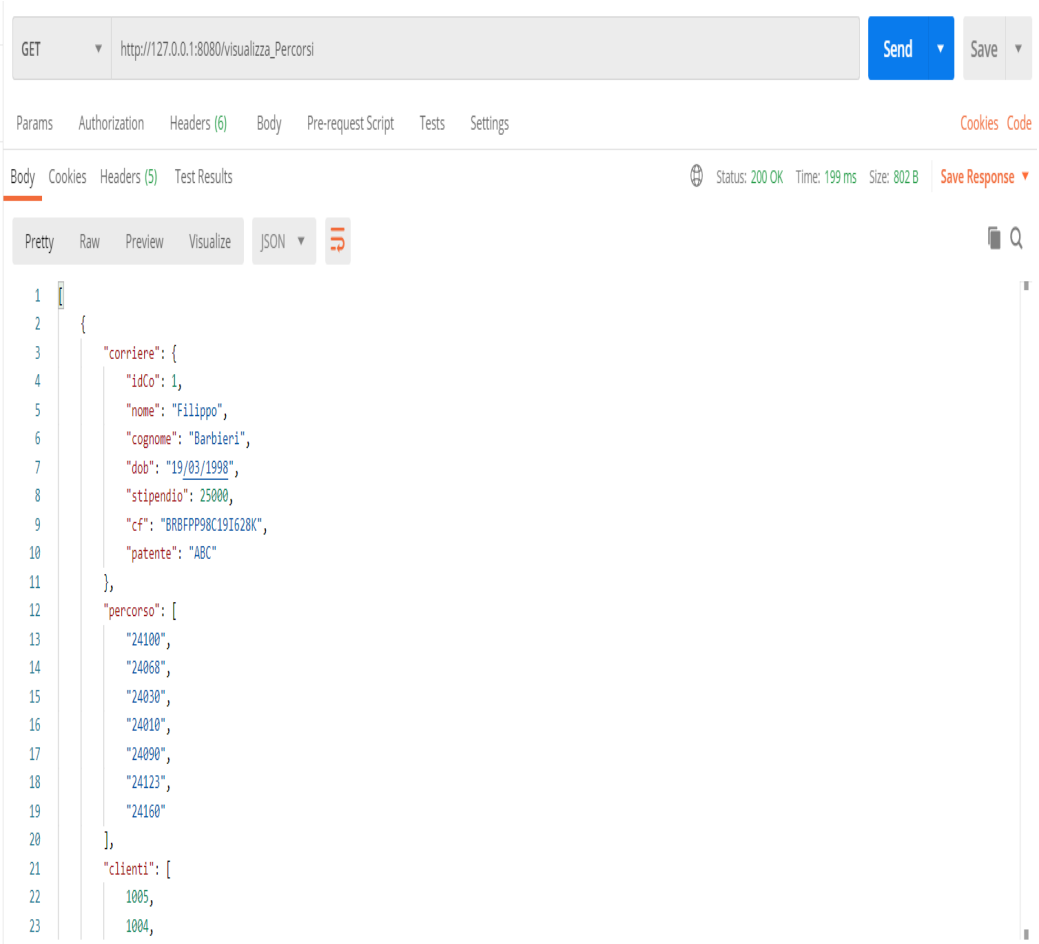

Figura 23: Class Diagram dei tipi di dato

4 Iterazione 3

Fino ad ora il software implementato è destinato agli utilizzatori interni del supermercato come l'amministratore o i corrieri. In questa iterazione si è voluto implementare la prima componente dedicata al cliente vero e proprio del supermercato. La componente corrisponde al caso d'uso UC5 riportato nella tabella 2 dei casi a media priorità: la registrazione del cliente.

4.1 UC5: Registrazione cliente

Breve descrizione: il cliente deve compilare l'apposito form per la registrazione alla WebApp. Senza registrazione non è possibile accedere ai servizi offerti.
Attori coinvolti: Cliente, Sistema.
Procedimento:

1. Il sistema mostra la pagina *Registrazione.html* al cliente
2. Il cliente deve compilare tutti i campi dell'apposito form inserendo le seguenti informazioni:
   - Nome
   - Cognome
   - Città
   - CAP
   - Indirizzo
   - Email
   - Password
   - Metodo di pagamento

   Infine clicca sul bottone *Registrati*.
3. Il sistema recupera i dati nella form e inserisce nel database il nuovo cliente.

4.2 UML Component Diagram

Il caso d'uso implementato in questa iterazione è stato aggiungo al Component Diagram dell'iterazione precedente (Figura 18). Il nuovo diagramma è mostrato in Figura 24. In particolare in blu sono riportati i componenti aggiunti in questa iterazione.

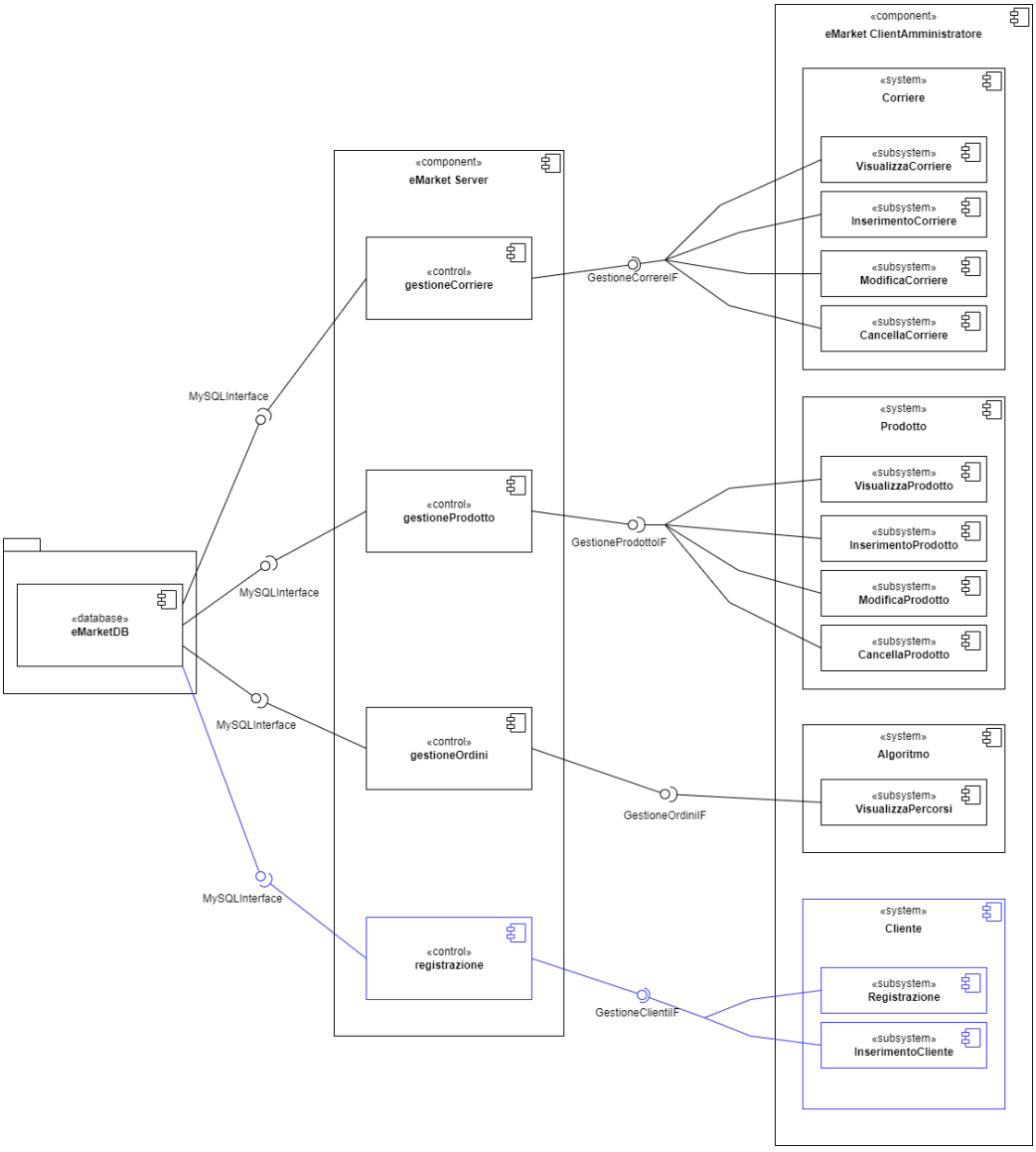

Figura 24: Component Diagram delle parti implementate

4.3 UML Class Diagram per le interfacce

Il class diagram mostrato in Figura 25 riferito all'interfaccia *GestioneClienteIF* implementata dalla classe *ClienteController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

Figura 25: Class Diagram dei metodi dell' interfaccia gestioneOrdiniIF

4.4 UML Class Diagram per i tipi di dato

Il diagramma in Figura 26 mostra il tipo di dato *Cliente* implementato in questa iterazione.

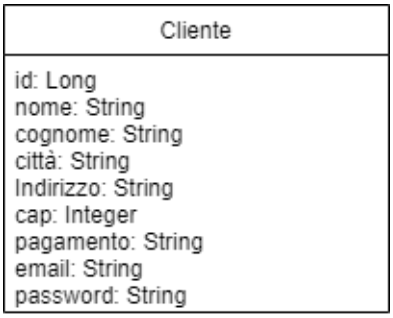

Figura 26: Class Diagram dei tipi di dato per il Cliente

4.5 Testing

4.5.1 Analisi Dinamica

Come per l'iterazione precedente è stato utilizzato JUnit per verificare la correttezza dei metodi implementati in questa iterazione. In particolare sono stati aggiunti dei casi di test alla classe *TestSetandGet.java* per testare il corretto funzionamento dei metodi di get() e set() della classe *Cliente.java*. Il relativo codice è mostrato in Figura 27

```
//Cliente
Cliente cl1 = new Cliente("Mario", "Rossi", "AMEX", "prova@gmail.com", "abcde", "Bergamo", "Via Bergamo", 24100);

@Test
public void getNomeTestCl() {
    assertEquals("Mario", cl1.getNome());
}

@Test
public void setNomeTestCl() {
    cl1.setNome("Filippo");
    assertEquals("Filippo", cl1.getNome());
}


@Test
public void getCognomeTestCl() {
    assertEquals("Rossi", cl1.getCognome());
}

@Test
public void setCognomeTestCl() {
    cl1.setCognome("Barbieri");
    assertEquals("Barbieri", cl1.getCognome());
}
```

Figura 27: Codice Unit Test sui metodi Get e Set della classe Cliente.java

In Figura 28 si mostra che i test di unità effettuati hanno avuto esito positivo

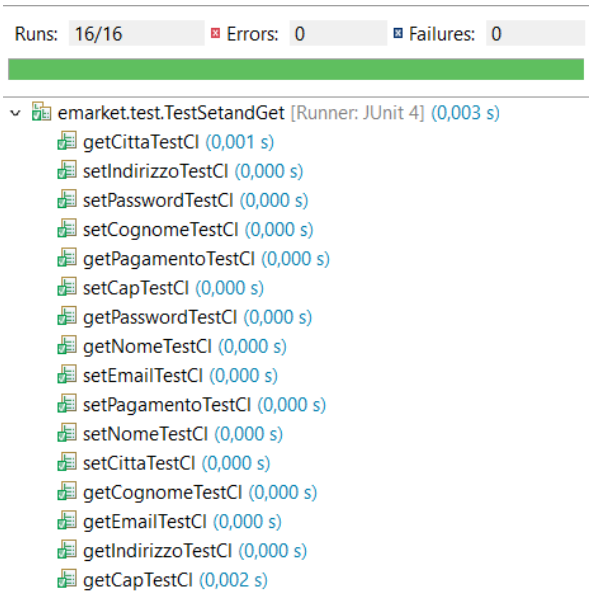

Figura 28: Esito Unit Test sui metodi Get e Set della classe Cliente.java

4.5.2 Test API tramite Postman

In Figura 29 è riportato il test delle API effettuato sulla chiamata GET *visualizza_Clienti* che restituisce tutti i clienti presenti nel database. Il metodo invocato è *getAllCl()*.

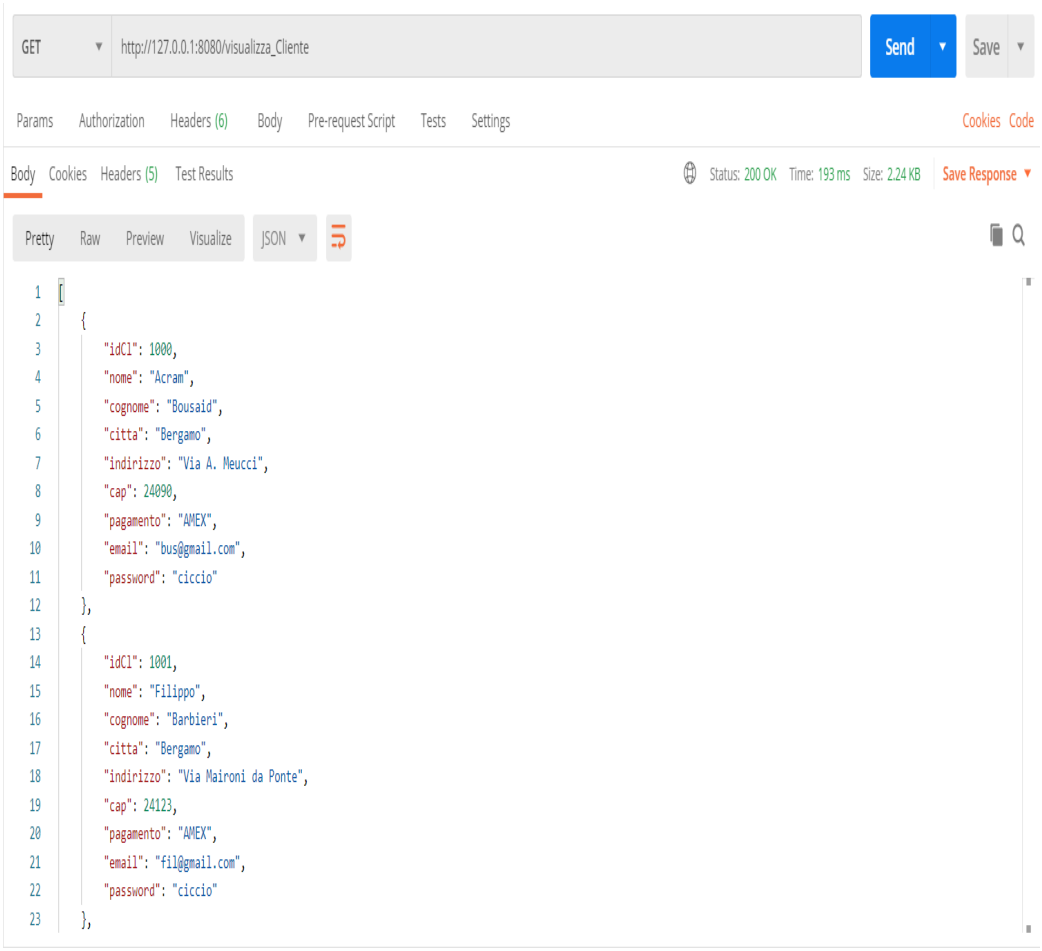

Figura 29: Chiamata GET

5 Analisi Statica

Al termine dell'iterazione 3 è stato utilizzato il tool CodeMr per effettuare un'analisi statica e della qualità architetturale del software implementato. In particolare questo strumento esplora l'intero progetto, analizzando i pacchetti e i moduli presenti evidenziandone le dipendenze.
Di seguito sono riportati alcuni grafici e viste generate dal tool che mostrano alcune caratteristiche e metriche del codice.

5.1 Grafo strutturale

CodeMR permette anche la generazione di grafi per la visualizzazione della struttura del progetto Spring in Java che costituisce il lato server (backend) dell'applicazione software.

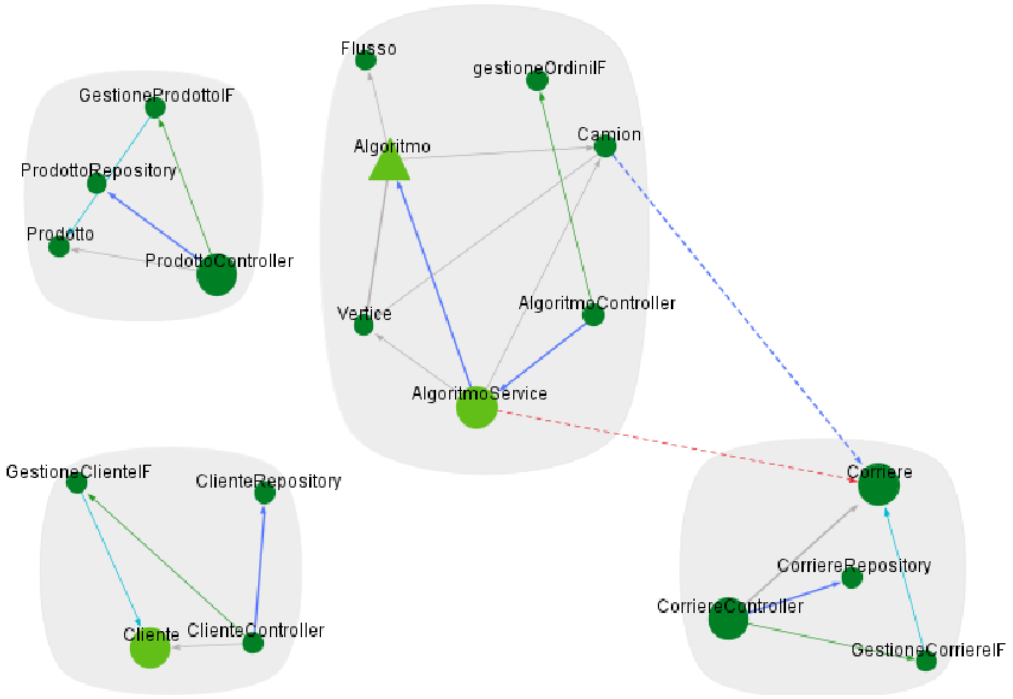

Figura 30: Grafo Strutturale

5.2 TreeMap

la TreeMap consente di individuare le dipendenze attraverso un approccio visivo legato ai colori. Il codice rappresentato da uno dei blocchi, dipende dal codice rappresentato dal blocco sottostante e ha una dipendenza bidirezionale dai blocchi collocati sul suo stesso livello. Nell'interfaccia HTML generata dal tool è possibile interagire con il grafico e visualizzare nel dettaglio ciò che accade all'interno di un singolo package.

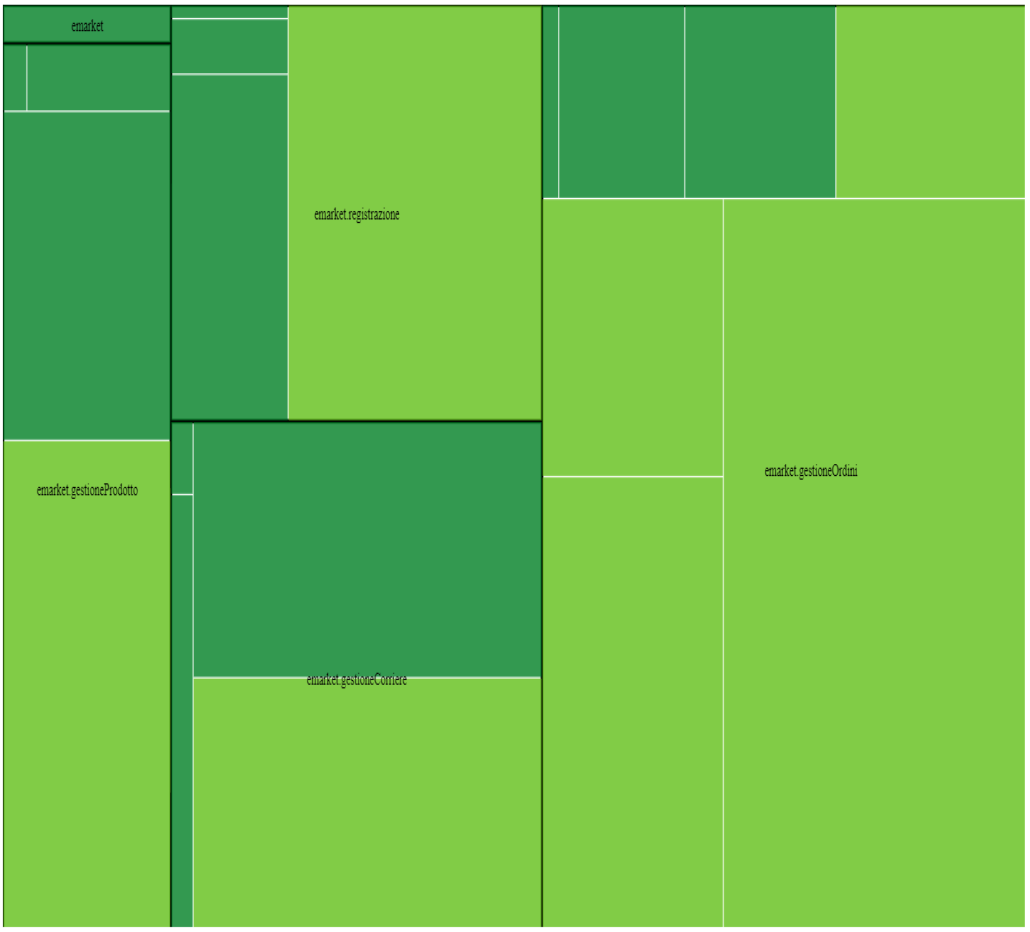

Figura 31: TreeMap

5.3 Project Outline

CodeMR fornisce anche uno schema riassuntivo del progetto che rappresenta i valori di molte metriche relative ad ogni classe del codice. Nella Figura 32 sono riportati i valori di alcune delle metriche più importanti quali:

- Quality Attributes
- Line Of Code
- Coupling
- Complexity
- Size
- Lack of Cohesion

Element	Quality Attributes	LOC	Coupling	Complex...	Size	Lack of Cohe...
initial						
emarket		4	low	low	low	low
EMarketApplication		4	low	low	low	low
emarket.gestioneCorriere		117	low	low	low	low
Corriere		55	low	low	low-medi...	low
CorriereController		55	low	low	low-medi...	low-medium
CorriereRepository		1	low	low	low	low
GestioneCorriereIF		6	low	low	low	low
emarket.gestioneOrdini		307	low	low	low-medi...	low
Algoritmo		161	low-medi...	low-medi...	low-medi...	low-medium
AlgoritmoController		15	low	low	low	low
AlgoritmoService		51	low	low-medi...	low-medi...	low
Camion		29	low	low	low	low-medium
Flusso		18	low	low	low	low
gestioneOrdiniIF		2	low	low	low	low
Vertice		31	low	low	low	low-medium
emarket.gestioneProdotto		92	low	low	low	low
GestioneProdottoIF		6	low	low	low	low
Prodotto		34	low	low	low	low
ProdottoController		51	low	low	low-medi...	low-medium
ProdottoRepository		1	low	low	low	low
emarket.registrazione		95	low	low	low	low
Cliente		65	low	low-medi...	low-medi...	low
ClienteController		25	low	low	low	low
ClienteRepository		1	low	low	low	low
GestioneClienteIF		4	low	low	low	low

Figura 32: Project Outline