

Documentazione Progetto

Informatica III - Progettazione e Algoritmi

Prof.ssa Patrizia Scandurra

Benedeta Lenuzza 1068745

Gloria Pasinetti 1066654

Andrea Lenzi 1066922

Laurea Magistrale in Ingegneria Informatica Febbraio 2023

# Indice

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [**1**](#_bookmark0) | [**Iterazione 0**](#_bookmark0) | | | | **5** |
|  | [1.1 Introduzione](#_bookmark1) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | | | 5 |
|  | [1.2 Requisiti Funzionali](#_bookmark2) . . . . . . . . . . . . . . . . . . . . . . . . | | | | 5 |
|  | [1.3 Analisi dei casi d’uso](#_bookmark3) . . . . . . . . . . . . . . . . . . . . . . . | | | | 7 |
|  | [1.4 Diagramma UML dei casi d’uso](#_bookmark7) . . . . . . . . . . . . . . . . . | | | | 8 |
|  | [1.5 Requisiti non funzionali](#_bookmark9) . . . . . . . . . . . . . . . . . . . . . | | | | 8 |
|  | [1.6 Pattern di progettazione](#_bookmark10) . . . . . . . . . . . . . . . . . . . . . | | | | 9 |
|  | [1.7 Topologia del sistema](#_bookmark12) . . . . . . . . . . . . . . . . . . . . . . . | | | | 10 |
|  | [1.8 ToolChain](#_bookmark14) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | | | 12 |
| [**2**](#_bookmark15) | [**Iterazione 1**](#_bookmark15) | | | | **13** |
|  | [2.1 UC1: Gestione corriere](#_bookmark16) . . . . . . . . . . . . . . . . . . . . . . | | | | 13 |
|  | | [2.1.1](#_bookmark17) | [UC1.1:](#_bookmark17) | [Inserimento corriere](#_bookmark17) . . . . . . . . . . . . . . . | 14 |
|  | | [2.1.2](#_bookmark18) | [UC1.2:](#_bookmark18) | [Visualizzazione corrieri](#_bookmark18) . . . . . . . . . . . . . | 14 |
|  | | [2.1.3](#_bookmark19) | [UC1.3:](#_bookmark19) | [Modifica corriere](#_bookmark19) . . . . . . . . . . . . . . . . . | 15 |
|  | | [2.1.4](#_bookmark20) | [UC:1.4](#_bookmark20) | [Elimina corriere](#_bookmark20) . . . . . . . . . . . . . . . . . | 15 |
| [2.2](#_bookmark21) | | [UC2:](#_bookmark21) | [Gestione](#_bookmark21) | [prodotto](#_bookmark21) . . . . . . . . . . . . . . . . . . . . . | 16 |
|  | | [2.2.1](#_bookmark22) | [UC2.1:](#_bookmark22) | [Inserimento prodotto](#_bookmark22) . . . . . . . . . . . . . . | 16 |
|  | | [2.2.2](#_bookmark23) | [UC2.2:](#_bookmark23) | [Visualizzazione prodotti](#_bookmark23) . . . . . . . . . . . . . | 17 |
|  | | [2.2.3](#_bookmark24) | [UC2.3:](#_bookmark24) | [Modifica prodotto](#_bookmark24) . . . . . . . . . . . . . . . . | 17 |
|  | | [2.2.4 UC:2.4 Elimina prodotto](#_bookmark25) . . . . . . . . . . . . . . . . . | | | 18 |
| [2.3](#_bookmark26) | | [UC3: Razionamento dei prodotti](#_bookmark26) . . . . . . . . . . . . . . . . | | | 18 |
| [2.4](#_bookmark27) | | [UML Component Diagram](#_bookmark27) . . . . . . . . . . . . . . . . . . . . | | | 19 |
| [2.5](#_bookmark29) | | [UML Class Diagram per le interfacce](#_bookmark29) . . . . . . . . . . . . . . | | | 20 |
| [2.6](#_bookmark32) | | [UML Class Diagram per i tipi di dato](#_bookmark32) . . . . . . . . . . . . . . | | | 21 |
| [2.7](#_bookmark34) | | [Testing](#_bookmark34) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | | 22 |
|  | | [2.7.1 Analisi Dinamica](#_bookmark35) . . . . . . . . . . . . . . . . . . . . . | | | 22 |
|  | | [2.7.2 Test API tramite Postman](#_bookmark38) . . . . . . . . . . . . . . . . | | | 23 |
| [**3**](#_bookmark41) | [**Iterazione 2**](#_bookmark41) | | | | **26** |
|  | [3.1 UC4: Gestione ordini](#_bookmark42) . . . . . . . . . . . . . . . . . . . . . . . | | | | 26 |
|  | [3.1.1 Algoritmo per l’allocazione degli ordini](#_bookmark43) . . . . . . . . . | | | | 26 |

* 1. [UML Component Diagram](#_bookmark50) 34
  2. [UML Class Diagram per le interfacce](#_bookmark52) 35
  3. [UML Class Diagram per i tipi di dato](#_bookmark54) 35
  4. [Testing](#_bookmark56) 36
     1. [Analisi Dinamica](#_bookmark57) 36
     2. [Test API tramite Postman](#_bookmark60) 37

1. [Iterazione 3](#_bookmark62) 38
   1. [UC5: Registrazione cliente](#_bookmark63) 38
   2. [UML Component Diagram](#_bookmark64) 39
   3. [UML Class Diagram per le interfacce](#_bookmark66) 40
   4. [UML Class Diagram per i tipi di dato](#_bookmark68) 40
   5. [Testing](#_bookmark70) 41
      1. [Analisi Dinamica](#_bookmark71) 41
      2. [Test API tramite Postman](#_bookmark74) 42
2. [Analisi Statica](#_bookmark76) 44
   1. [Grafo strutturale](#_bookmark77) 44
   2. [TreeMap](#_bookmark79) 45
   3. [Project Outline](#_bookmark81) 46

Elenco delle figure

1. [Diagramma UML dei casi d’uso](#_bookmark8) 8
2. [Model View Presenter](#_bookmark11) 10
3. [Topologia del sistema](#_bookmark13) 11
4. [Component Diagram delle parti implementate](#_bookmark28) 19
5. [Class Diagram dei metodi interfaccia GestioneCorriereIF](#_bookmark30) 20
6. [Class Diagram dei metodi interfaccia GestioneProdottoIF](#_bookmark31) 20
7. [Class Diagram per i tipi di dato](#_bookmark33) 21
8. [Codice Unit Test sui metodi Get e Set della classe prodotto.java](#_bookmark36) 22
9. [Risultato Unit Test sui metodi Get e Set della classe prodot-](#_bookmark37) [to.java](#_bookmark37) 23
10. [Chiamata POST inserisci prodotto](#_bookmark39) 24
11. [Chiamata GET visualizza prodotti](#_bookmark40) 25
12. [Grafo della citt`a](#_bookmark44) 27
13. [Soluzione del problema di flusso di costo minimo iniziale](#_bookmark45) 28
14. [Risultato finale dell’algoritmo](#_bookmark46) 29
15. [Pseudocodice parte 1](#_bookmark47) 31
16. [Pseudocodice parte 2](#_bookmark48) 32
17. [Flow chart dell’algoritmo](#_bookmark49) 33
18. [Component Diagram delle parti implementate](#_bookmark51) 34
19. [Class Diagram dei metodi dell’ interfaccia gestioneOrdiniIF](#_bookmark53) 35
20. [Class Diagram dei tipi di dato](#_bookmark55) 35
21. [Codice Unit Test per la classe Algoritmo.java](#_bookmark58) 36
22. [Risultato Unit Test per la classe Algoritmo.java](#_bookmark59) 36
23. [Class Diagram dei tipi di dato](#_bookmark61) 37
24. [Component Diagram delle parti implementate](#_bookmark65) 39
25. [Class Diagram dei metodi dell’ interfaccia gestioneOrdiniIF](#_bookmark67) 40
26. [Class Diagram dei tipi di dato per il Cliente](#_bookmark69) 40
27. [Codice Unit Test sui metodi Get e Set della classe Cliente.java](#_bookmark72) 41
28. [Esito Unit Test sui metodi Get e Set della classe Cliente.java](#_bookmark73) . 42 [29 Chiamata GET](#_bookmark75) 43
29. [Grafo Strutturale](#_bookmark78) 44
30. [TreeMap](#_bookmark80) 45
31. [Project Outline](#_bookmark82) 46

**Elenco delle tabelle**

1. [Casi d’uso ad alta priorit`a](#_bookmark4) 7
2. [Casi d’uso a media priorit`a](#_bookmark5) 7
3. [Casi d’uso a bassa priorit`a](#_bookmark6) 7

# Iterazione 0

## Introduzione

Lo scopo di questo progetto `e quello di creare un sistema distribuito per la gestione dei parcheggi coperti nella citta di Bergamo.

L’applicativo `e sviluppato per far fronte alle esigenze di un attore:

* + - L’utente che cerca un parcheggio disponibile.

Ogni parcheggio ha a disposizione un numero limitato di parcheggi che vanno costantemente contati, per cui e in grado di ospitare un numero massimo di utenti contemporaneamente. Per questo motivo, per migliorare l’efficienza sia in termini di costo che di tempo `e stato sviluppato un apposito algoritmo grazie al quale si riesce ad abbinare l’utente al parcheggio più adatto alle sue esigenze. Il focus del progetto `e quindi quello di gestire la prenotazione di posti auto online effettuata da parte dei clienti ottimizzando le risorse necessarie per effettuare la prenotazione.

## Requisiti Funzionali

Il sistema consente all’utente che cerca parcheggio di sovraintendere agli aspetti principali della gestione dei posti liberi. In particolare, inserendo il nome utente e la password per la login l’utente sarà in grado di:

* Accedere alla propria area personale, visualizzando i suoi dati di accesso.
* Visualizzare i parcheggi disponibili e, attraverso un algoritmo, verificare quali tra questi si adatta al meglio alle proprie esigenze (vicino, posti liberi, posizione, recensione, orario…).
* Visualizzare le recensioni dei singoli parcheggi disponibili all’interno della applicazione.
* Creare recensioni per un determinato parcheggio.
* Possibilità di prenotare il posto auto, specificando orario e quantità di veicoli.

## Analisi dei casi d’uso

Al fine di procedere ad uno sviluppo efficente, `e stato deciso di dividere le specifiche funzionali in tre code di priorit`a: alta, media, bassa. Nella coda ad alta priorit`a si trovano i casi d’uso fondamentali al corretto funziona- mento dell’applicazione, nella coda a media priorit`a sono inserti i casi d’uso riguardanti le funzionalit`a aggiuntive e nella coda a bassa priorit`a ci sono le funzionalit`a non strettamente necessarie.

|  |  |
| --- | --- |
| **CODICE** | **DESCRIZIONE** |
| UC1 | Prenotazione parcheggi e visualizzazione dei singoli parcheggi disponibili. |

Tabella 1: Casi d’uso ad alta priorit`a

|  |  |
| --- | --- |
| **CODICE** | **DESCRIZIONE** |
| UC2 | Registrazione Utente |
| UC3 | Login Utente |
| UC4 | Logout Utente |
| UC5 | Visualizzazione recensioni |
| UC6 | Creazione di recensioni |

Tabella 2: Casi d’uso a media priorit`a

|  |  |
| --- | --- |
| **CODICE** | **DESCRIZIONE** |
| UC7 | Modifica e visualizzazione dati utente |

Tabella 3: Casi d’uso a bassa priorit`a

## Diagramma UML dei casi d’uso

## 

Figura 1: Diagramma UML dei casi d’uso

## Requisiti non funzionali

Il progetto verr`a sviluppato tenendo in considerazione anche alcuni requisiti non funzionali, quali la portabilit`a, l’efficienza e l’usabilit`a.

### Efficienza

Il requisito dell’efficienza `e anche l’obiettivo primario del progetto, ovvero la divisione ottimale delle prenotazioni in modo da ottimizzare i tempi e i posti disponibili.

### Usabilit`a

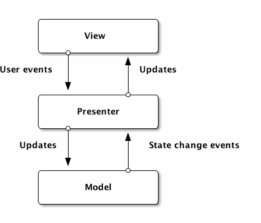
L’usabilit`a `e garantita dalla decisione di sviluppare il programma attraverso un’applicazione Android, di facile utilizzo sia per gli utenti che per il proprietario. Inoltre, `e possibile accedere al sistema tramite una pagina Web dotata di chiare e semplici interfacce.

### Portabilit`a

Il sistema sar`a sviluppato come una Web App il che render`a piu` semplice la portabilit`a su un’App Android.

## Pattern di progettazione

Il design pattern scelto per lo svilupppo del progetto `e il Model View Pre- senter (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: `e il layer intermedio tra Model e View, prende i dati dal Model, li elabora e li restituisce alla View.

## Topologia del sistema

La topologia del sistema, mostrata in Figura [3,](#_bookmark13) evidenzia come il progetto sia stato sviluppato utilizzando un’architettura Three-Tier. L’applicazione

`e 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, ossia l’interfaccia utente.

Nello specifico il modulo Data `e costituito da un Database gestito con My- SQL, 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 tramite una WebApp realizzata in JavaScript, HTML/CSS o un app Android.

Figura 3: Topologia del sistema

## 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

# Iterazione 1

Nella prima iterazione si `e deciso di implementare i primi casi d’uso ad alta priorit`a riportati nella tabella [1](#_bookmark4) di pagina [7:](#_bookmark4)

UC1: Prenotazione parcheggi e visualizzazione dei singoli parcheggi disponibili.

1. Visualizzazione dei parcheggi disponibili
2. Prenotazione di un parcheggio

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

## UC1.1: Visualizzazione dei parcheggi disponibili

*Breve descrizione:* All’utente che vuole prenotare un parcheggio verrà messa a disposizione una mappa e vare opzioni di ricerca calcolate tramite un algoritmo a seconda delle necessita dello stesso. Successivamente, il cliente avrà la possibilità di scegliere la propria opzione definitiva prenotando uno o più posti macchina nello specifico parcheggio specificando: Orario, Numero di veicoli, durata della sosta.

1. Visualizzazione della mappa e dei parcheggi
2. Calcolo della migliore opzione disponibile
3. Visualizzazione dei parcheggi (eventualmente con foto)

*Attori coinvolti:* Sistema, Utente.

*Procedimento:* il Sistema mostra la pagina home dedicata alla visualizzazione di tutti i parcheggi, L’utente inserisce il parametro da prioritizzare nella ricerca, il sistema aggiorna la pagina mostrando le migliori opzioni calcolate tramite un algoritmo di ricerca applicato su tutti i parcheggi precedentemente mostrati, l’utente visualizza il parcheggio a lui più adatto.

### UC1.1: Visualizzazione della mappa e dei parcheggi

*Breve descrizione:* Nella schermata verrà visualizzata una mappa o lista dei parcheggi totali registrati nel sistema.

Attori coinvolti: Utente, Sistema.

*Procedimento:*

* + - 1. Il sistema mostra la pagina home.
      2. L’utente clicca sulla icona della mappa per visualizzare tutte le opzioni
      3. Il sistema mostra a schermo la relativa pagina (*mappa.html* ).

### UC1.2: Calcolo della migliore opzione disponibile

*Breve descrizione. Utente decide il parametro che deve avere piu priorita nella ricerca del parcheggio e il sistema risponde con una lista che tiene conto di questo.*

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

* + - 1. Il sistema mostra la pagina home della mappa
      2. L’utente clicca sulla finestra dei parametri e decide quello che si adatta meglio alle sue necessita
      3. Il sistema risponde con una lista che rappresenti al meglio la ricerca dell’utente.

### UC1.3: Visualizzazione dei parcheggi

*Breve descrizione:* L’utente dopo aver fatto la ricerca può visualizzare parcheggio per parcheggio al fine di trovare quello che si adatta meglio alle proprie esigenze.

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

* + - 1. Il sistema mostra le varie opzioni ottimizzate
      2. L’ Utente sceglie la sua ipotetica scelta definitiva.
      3. Il sistema mostra la descrizione del singolo parcheggio.
      4. L’utente può decidere se procedere alla prenotazione o cambiare parcheggio scelto.

## UC2: Prenotazione di un parcheggio

*Breve descrizione:* Una volta scelto definitivamente il parcheggio, l’utente puo procedere alla sua prenotazione specificando al sistema: il numero di macchine coinvolte, l’orario di arrivo, e il tempo complessivo di sosta del singolo.:

* + - UC2.1: Effettua Prenotazione
    - UC2.2: Conferma prenotazione

*Attori coinvolti:* Utente, Sistema.

*Procedimento:* Il sistema dopo aver mostrato all’utente il singolo parcheggio che si addice meglio alle sue esigenze, mostra sulla schermata un bottone dove l’utente può prenotare il posto auto specificando i parametri descritti precedentemente. Il sistema confermerà la prenotazione e aggiornerà i posti disponibili nel parcheggio.

### UC2.1: Effettua Prenotazione

*Breve descrizione:* L’utente decide il parcheggio e lo prenota.

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

* + - 1. L’utente dopo aver visualizzato un parcheggio clicca su prenota
      2. Il sistema mostra a schermo la relativa pagina (*prenota.html* ) in cui `e presente un apposito form.
      3. L’utente compila il form e clicca sul bottone *Prenota*.
      4. Il sistema aggiorna la pagina mostrando a schermo la prenotazione.

### UC2.2: Conferma prenotazione

*Breve descrizione:* Una volta compilato il form il sistema prenota e aggiorna i posti relativi al parcheggio scelto.

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

* + - 1. Il sistema mostra la pagina di prenotazione all’utnete
      2. L’ utente prenota il posto.
      3. Il sistema mostra a schermo la relativa pagina (*prenotato.html* ) dove mostra la avvenuta prenotazione e aggiorna i posti disponibili nel parcheggio scelto dall’utente.

## UML Component Diagram

Analizzando i casi d’uso implementati in questa iterazione `e stato possibile progettare il Component Diagram mostrato in Figura [4.](#_bookmark28) Il grafico evidenzia le interazioni tra i componenti dal punto di vista delle funzionalit`a

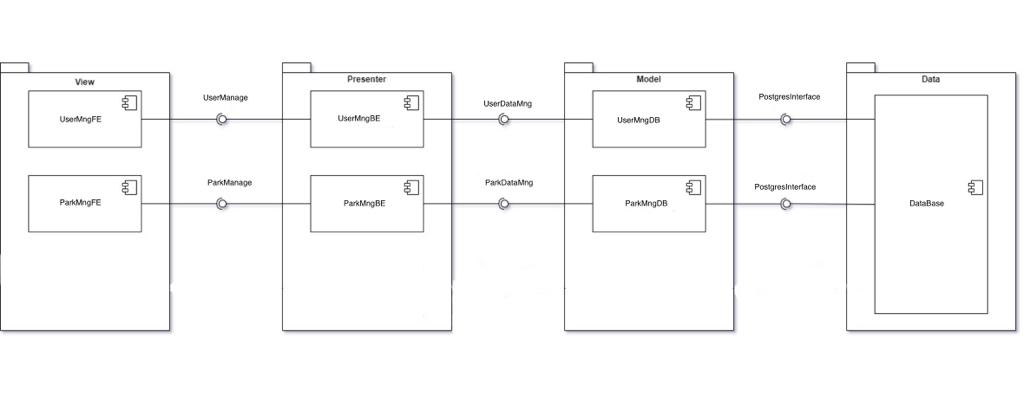


Figura 4: Component Diagram delle parti implementate

## UML Class Diagram per le interfacce

Il class diagram mostrato in Figura [5](#_bookmark30) riferito all’interfaccia *GestioneCorrie- reIF* implementata dalla classe *CorriereController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

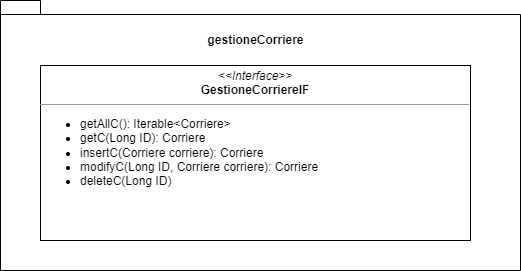


Figura 5: Class Diagram dei metodi interfaccia GestioneCorriereIF

Il class diagram mostrato in Figura [6](#_bookmark31) riferito all’interfaccia *GestioneProdot- toIF* implementata dalla classe *ProdottoController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

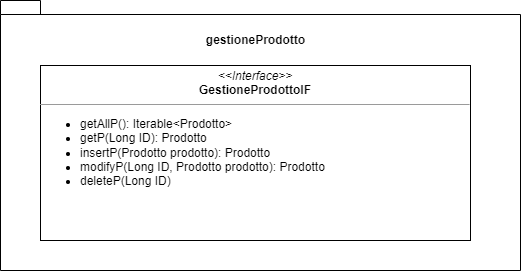


Figura 6: Class Diagram dei metodi interfaccia GestioneProdottoIF

## UML Class Diagram per i tipi di dato

Il diagramma in Figura [7](#_bookmark33) mostra i tipi di dato necessari allo sviluppo del- l’applicazione.

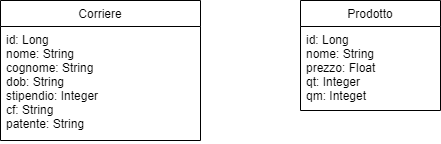


Figura 7: Class Diagram per i tipi di dato

## Testing

### Analisi Dinamica

Per effettuare l’analisi dinamica dell’applicazione `e 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 `e stata creata una classe: *TestSetandGet.java* che testa il corretto funzionamento di questi metodi. Il relativo codice `e mostrato in Figura [8.](#_bookmark36)

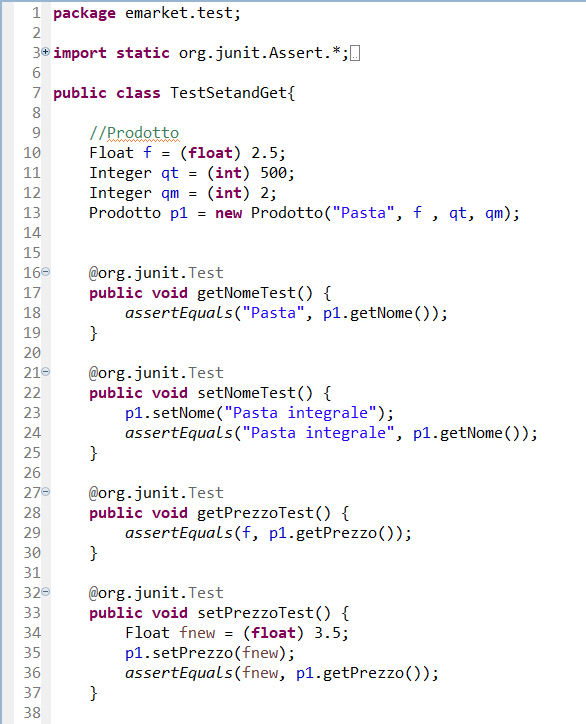


Figura 8: Codice Unit Test sui metodi Get e Set della classe prodotto.java

In Figura [9](#_bookmark37) si mostra che il test di unit`a fatto con JUnit `e andato a buon fine.

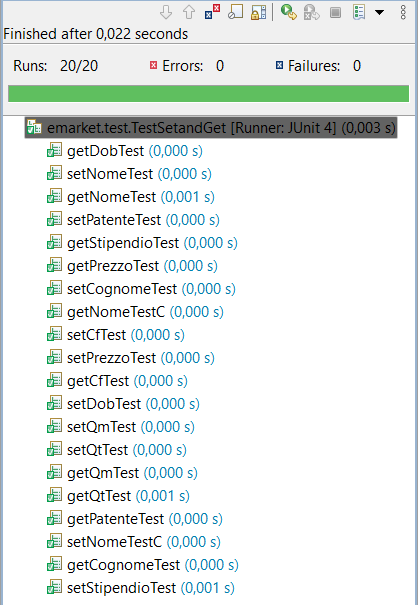


Figura 9: Risultato Unit Test sui metodi Get e Set della classe prodotto.java

### Test API tramite Postman

La verifica del buon funzionamento delle API REST esposte dai vari con- troller `e stata effettuata tramite il software PostMan. In particolare `e stato testato il funzionamento di una chiamata GET e di una chiamata POST esposte dalla classe *ProdottoController.java* attraverso l’interfaccia *Gestio- neProdottoiF*

### Test chiamata POST Prodotto

La chiamata POST mostrata in Figura [10](#_bookmark39) consente alla WebApp di salvare nel database le informazioni di un singolo prodotto passando i valori inse- riti dall’utente nei relativi campi. Il metodo invocato `e *insertP(Prodotto prodotto)*

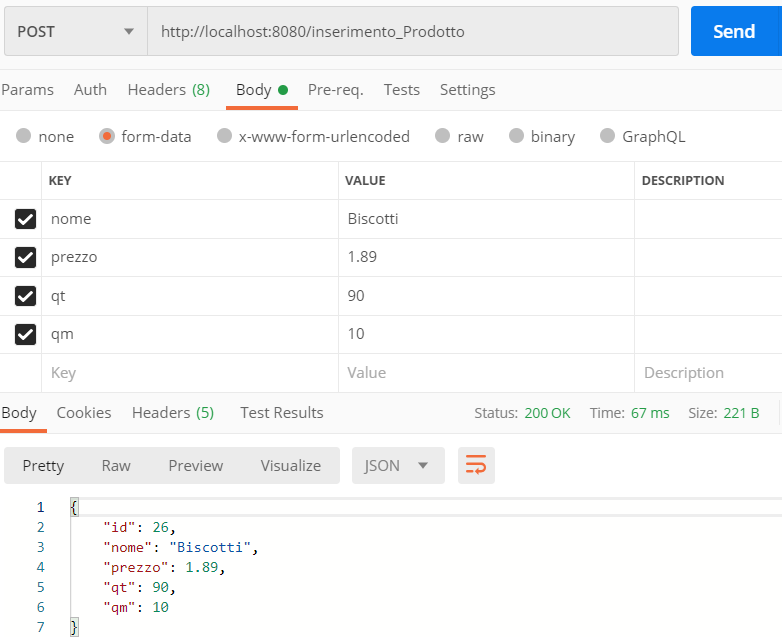


Figura 10: Chiamata POST inserisci prodotto

### Test chiamata GET Prodotto

La chiamata GET mostrata in Figura [11](#_bookmark40) consente alla WebApp di recuperare dal database le informazioni di tutti prodotti. Il metodo invocato `e *getAllP()*

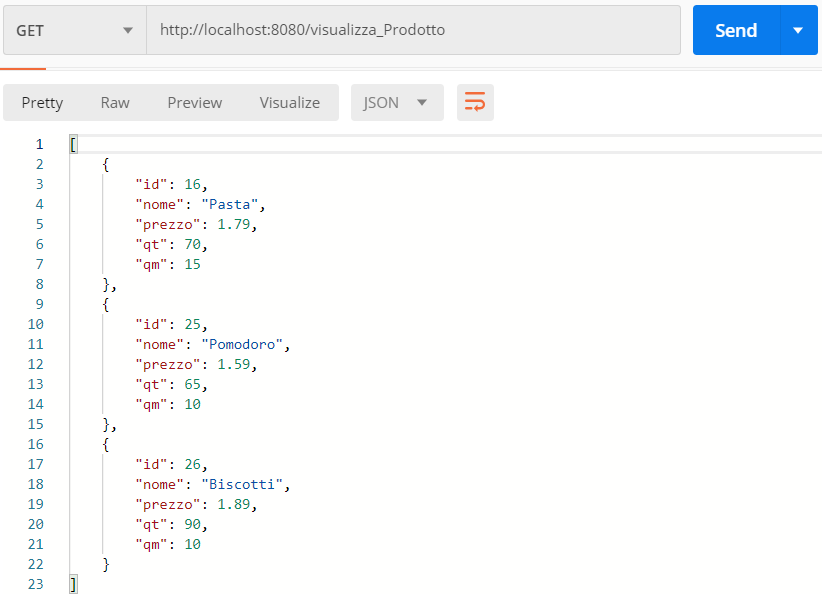


Figura 11: Chiamata GET visualizza prodotti

# Iterazione 2

Durante questa iterazione il team si `e concentrato principalmente sullo svi- luppo della parte algoritmica. L’obiettivo `e quello di implementare un algo- ritmo 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 ta- bella [1](#_bookmark4) dei casi d’uso ad alta priorit`a. Di seguito `e riportata una descrizione di questo caso d’uso.

## UC4: Gestione ordini

*Breve descrizione:* dopo che i clienti hanno effettuato la spesa online ge- nerando 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

`e prensente l’elenco dei camion e per ogni camion viene specificata la tratta che deve essere percorsa.

L’assegnamento degli ordini ai camion `e effettuato da un apposito algoritmo descritto di seguito.

### Algoritmo per l’allocazione degli ordini

Lo scopo principale dell’algoritmo `e quello di minimizzare il numero di camion utilizzati per effettuare le consengne in modo da ridurre i costi totali di trasporto. Come prima cosa `e stata definita la mappa della citt`a tramite un apposito grafo, riportato in Figura [12.](#_bookmark44)

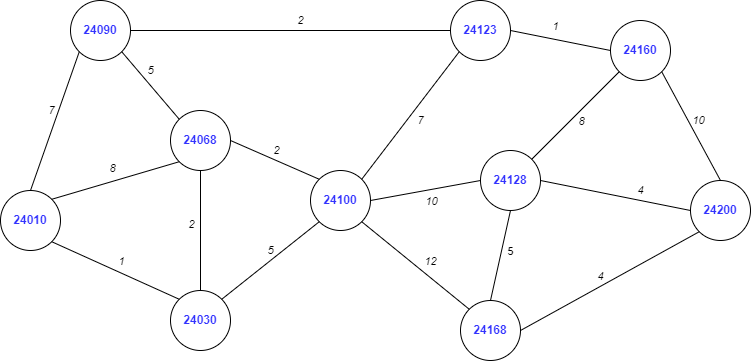


Figura 12: Grafo della citt`a

Ogni nodo `e caratterizzato da due etichette:

* + - * Il CAP
      * Il peso che rappresenta il numero di ordini da portare in quel nodo. Ogni arco `e 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* `e presente il supermercato da cui vengono spediti tutti gli ordini.

Il problema per come `e stato modellizzato `e un problema di Min Cost Flow os- sia 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 quantit`a 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 `e 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 `e 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 flus- so di costo minimo del grafo, ovvero per ogni arco viene speficiato il flusso (il numero di ordini) che lo attraversa.

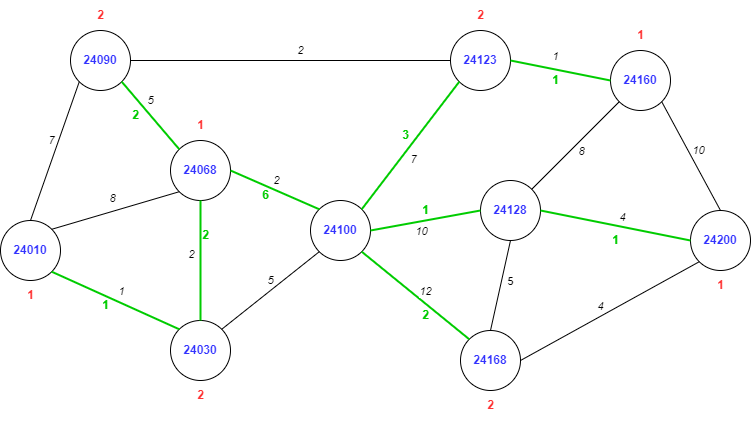


Figura 13: Soluzione del problema di flusso di costo minimo iniziale

1. 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 aggiun- gendolo 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 `e possibile trovare un ul- teriore emax, ovvero sono arrivato ad un nodo che ha tutti i suoi archi uscenti con flusso pari a 0.

1. Ora cerco un nodo adiacente alla posizione corrente con peso *>* 0 che sia percorribile.
2. Se esiste mi sposto in quel nodo, aggiorno il contatore dei km percorsi e azzero il peso sul nodo raggiunto.
3. Se non esiste faccio tornare il camion alla sorgente aggiornando i km percorsi.
4. 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](#_bookmark46) `e mostrato il risultato finale dell’algoritmo

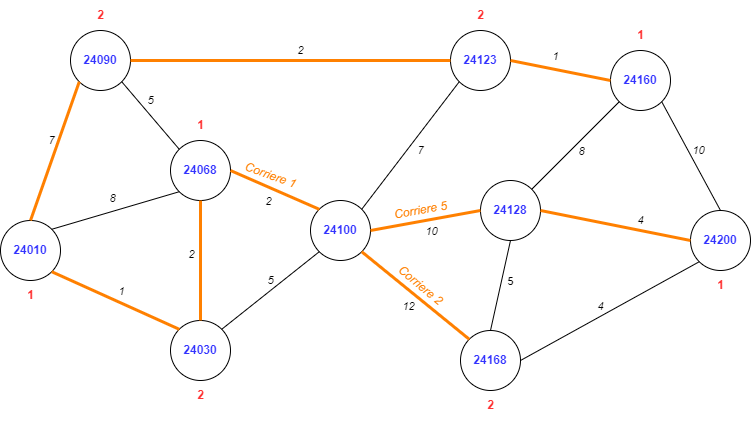


Figura 14: Risultato finale dell’algoritmo

### Pseudocodice

In Figura [15](#_bookmark47) e [16](#_bookmark48) `e riportato lo pseudocodice dell’algortimo con l’analisi di complessit`a. Nell’eseguire l’analisi di complessit`a si considera il numero di nodi del grafo pari a *n* e il numero di archi pari a *m*. La complessit`a dell’algoritmo `e mostrata di seguito.

*O*((*m* + *n*)*n* log *n*) (1)

L’espressione pu`o essere riscritta nel seguente modo nel caso in cui il numero di archi `e molto maggiore al numero di nodi ossia *m* ≫ *n*

*O*((*mn* log *n*) (2)

L’elemento principale a cui corrisponde questa elevata complessit`a `e la chia- mata della funzione *getMinimumCostFlow()* fornita dalla libreria *jGraphT* usata per calcolare il flusso di costo minimo del problema.



Figura 15: Pseudocodice parte 1

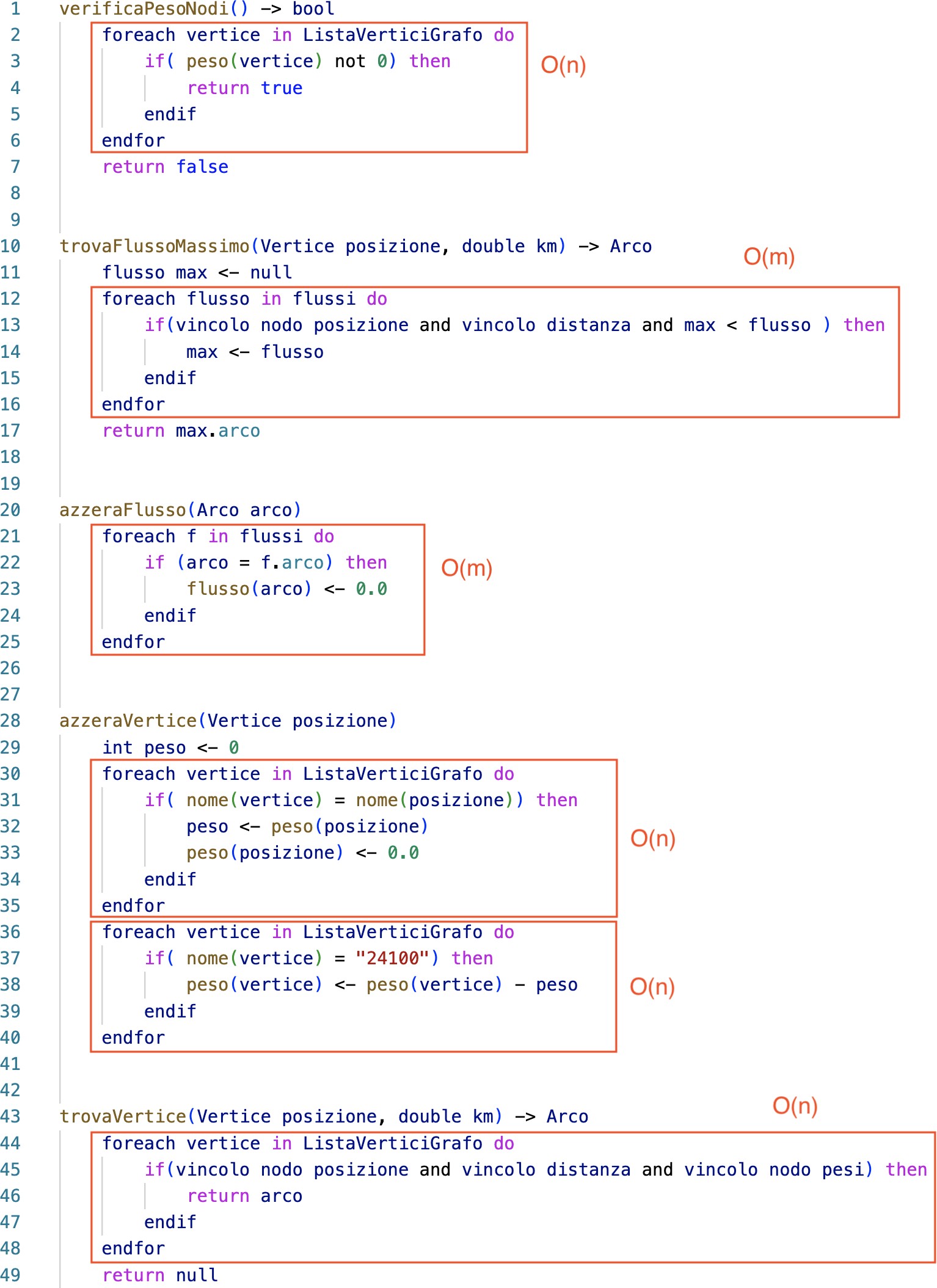


Figura 16: Pseudocodice parte 2

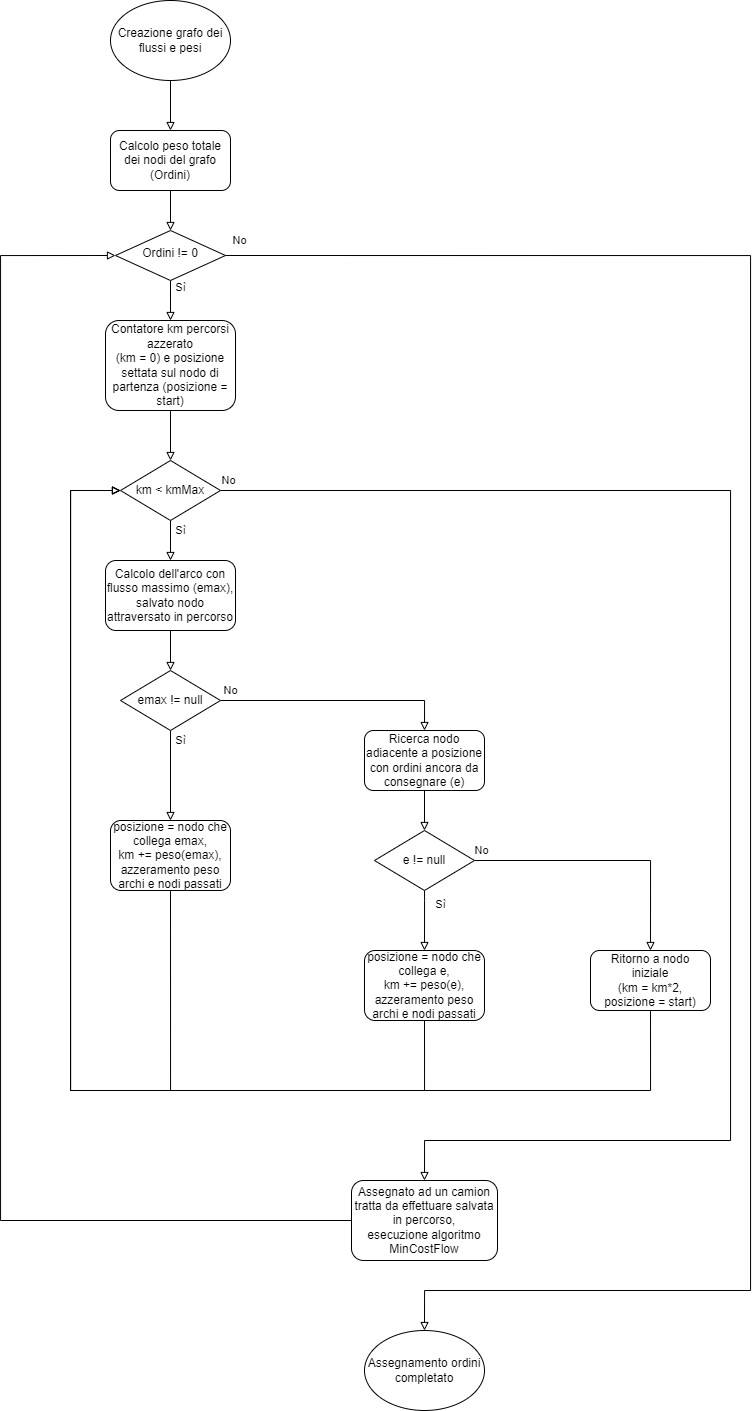


Figura 17: Flow chart dell’algoritmo

## UML Component Diagram

Il caso d’uso implementato in questa iterazione `e stato aggiungo al Com- ponent Diagram dell’iterazione precedente (Figura [4](#_bookmark28)). Il nuovo diagramma

`e mostrato in Figura [18.](#_bookmark51) In particolare in blu sono riportati i componenti aggiunti in questa iterazione.

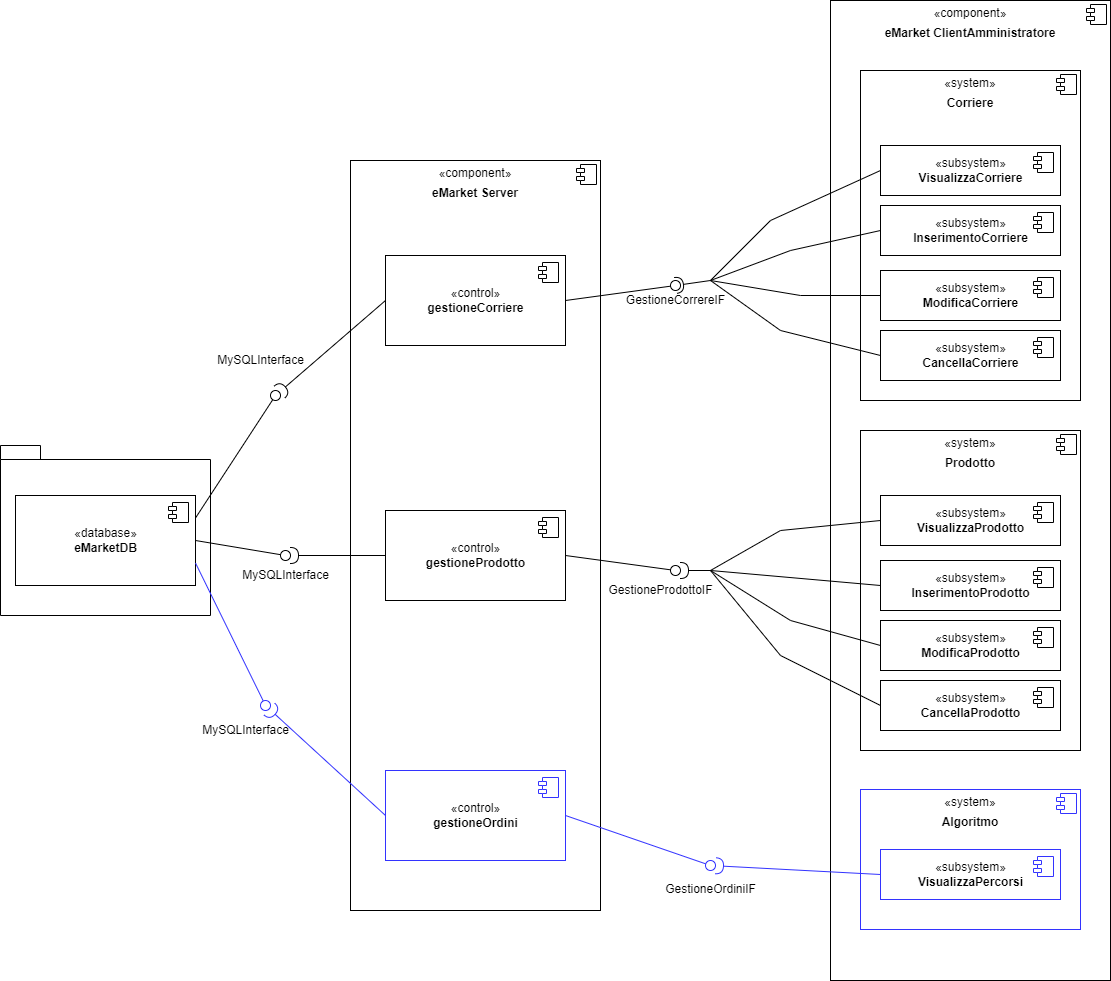


Figura 18: Component Diagram delle parti implementate

## UML Class Diagram per le interfacce

Il class diagram mostrato in Figura [19](#_bookmark53) riferito all’interfaccia *gestioneOrdi- niIF* implementata dalla classe *AlgoritmoController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

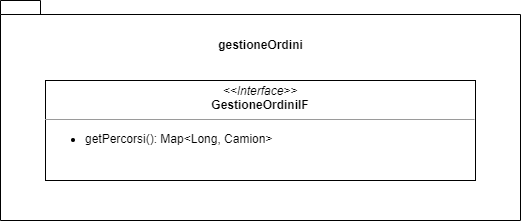


Figura 19: Class Diagram dei metodi dell’ interfaccia gestioneOrdiniIF

## UML Class Diagram per i tipi di dato

La classe *Algoritmo* introdotto in questa iterazione, `e stata inserita nel Class Diagram per tipo di dato dell’iterazione precedente (Figura [7).](#_bookmark33) Il nuovo diagramma, mostrato in Figura 20, mostra anche le relazioni che si sono instaurate tra i vari tipi di dato.

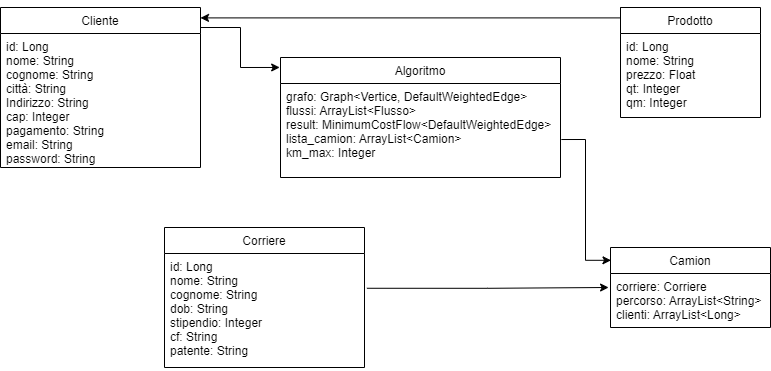


Figura 20: Class Diagram dei tipi di dato

## Testing

### Analisi Dinamica

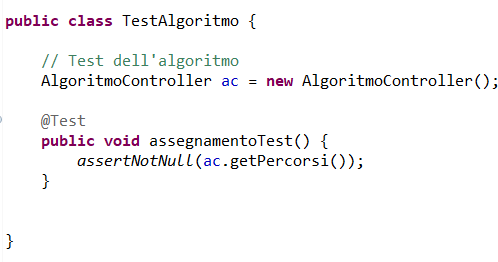
Come per l’iterazione precedente `e stato utilizzato JUnit per verificare la correttezza dei metodi implementati nello sviluppo dell’algoritmo. Il relativo codice `e mostrato in Figura [21.](#_bookmark58)

Figura 21: Codice Unit Test per la classe Algoritmo.java In Figura [22](#_bookmark59) si mostra che il test di unit`a `e andato a buon fine.

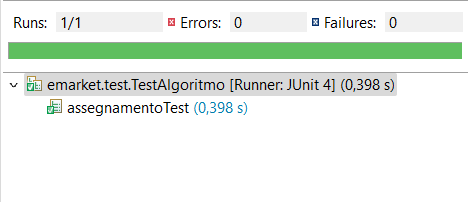


Figura 22: Risultato Unit Test per la classe Algoritmo.java

### Test API tramite Postman

Anche in questa iterazione `e stato utilizzato il software PostMan per verificare la corretta esecuzione delle chiamate REST API. In particolare la chiamata GET mostrata in Figura [23](#_bookmark61) 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 `e *getPercorsi()*.

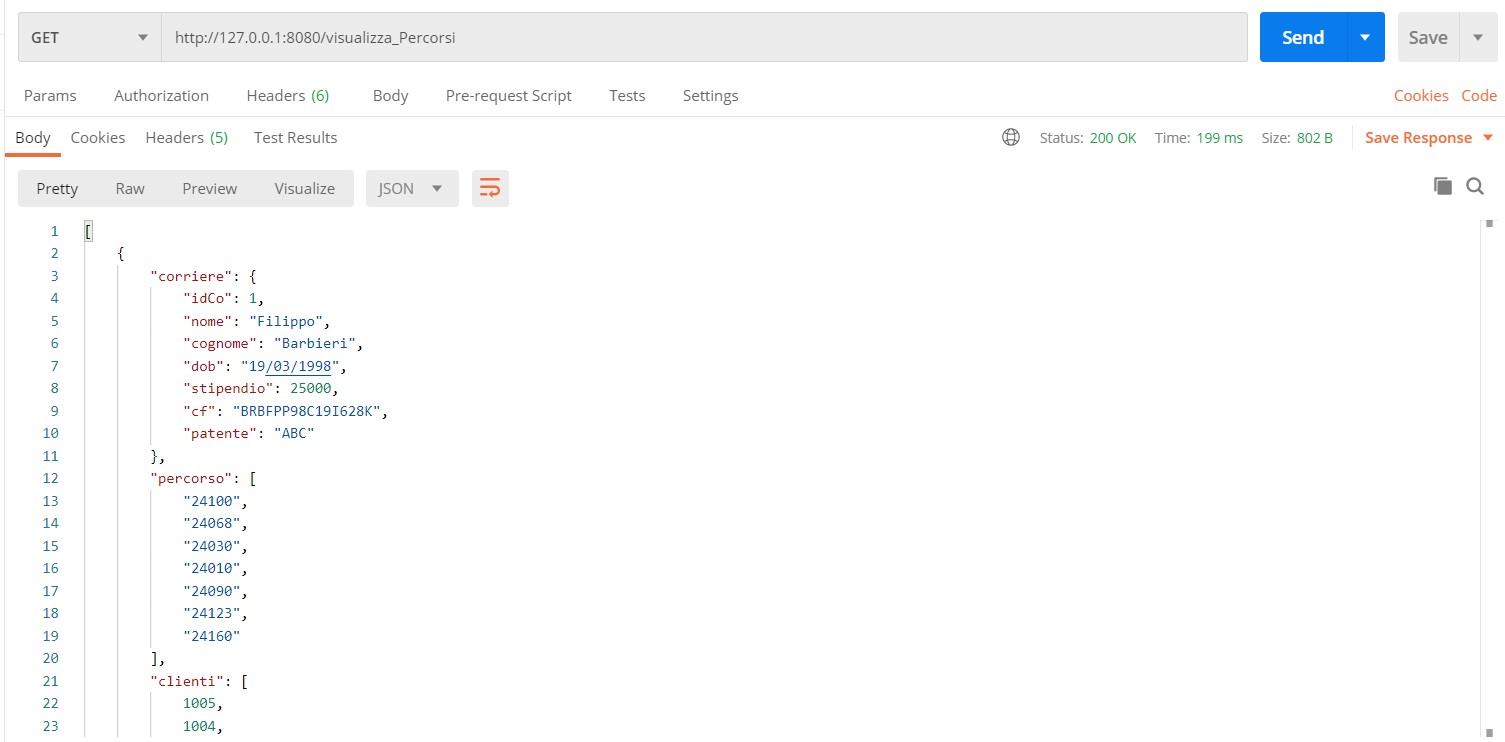


Figura 23: Class Diagram dei tipi di dato

# Iterazione 3

Fino ad ora il software implementato `e destinato agli utilizzatori interni del supermercato come l’amministratore o i corrieri. In questa iterazione si `e 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](#_bookmark5) dei casi a media priorit`a: la registrazione del cliente.

## UC5: Registrazione cliente

*Breve descrizione:* il cliente deve compilare l’apposito form per la registra- zione alla WebApp. Senza registrazione non `e 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
   * Citt`a
   * CAP
   * Indirizzo
   * Email
   * Password
   * Metodo di pagamento

Infine clicca sul bottone *Registrati*.

1. Il sistema recupera i dati nella form e inserisce nel database il nuovo cliente.

## UML Component Diagram

Il caso d’uso implementato in questa iterazione `e stato aggiungo al Compo- nent Diagram dell’iterazione precedente (Figura [18).](#_bookmark51) Il nuovo diagramma

`e mostrato in Figura [24.](#_bookmark65) In particolare in blu sono riportati i componenti aggiunti in questa iterazione.

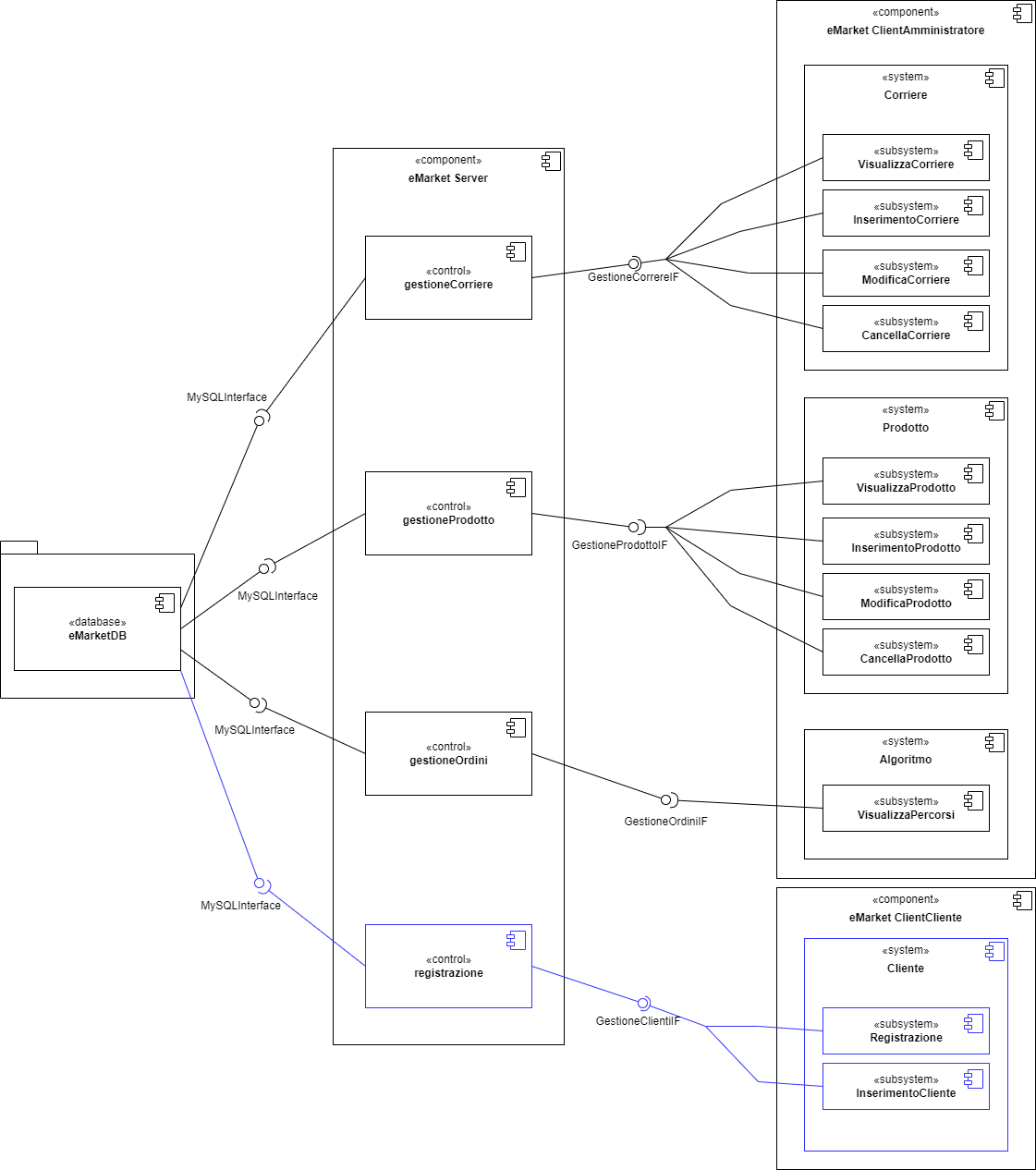


Figura 24: Component Diagram delle parti implementate

## UML Class Diagram per le interfacce

Il class diagram mostrato in Figura [25](#_bookmark67) riferito all’interfaccia *GestioneClien- teIF* implementata dalla classe *ClienteController.java* mette in evidenza la segnatura specifica dei metodi e i valori ritornati.

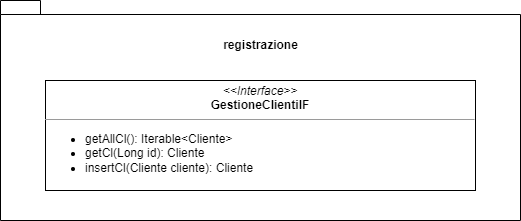


Figura 25: Class Diagram dei metodi dell’ interfaccia gestioneOrdiniIF

## UML Class Diagram per i tipi di dato

Il diagramma in Figura [26](#_bookmark69) mostra il tipo di dato *Cliente* implementato in questa iterazione.

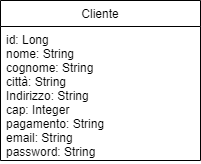


Figura 26: Class Diagram dei tipi di dato per il Cliente

## Testing

### Analisi Dinamica

Come per l’iterazione precedente `e 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 `e mostrato in Figura [27](#_bookmark72)



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

In Figura [28](#_bookmark73) si mostra che i test di unit`a effettuati hanno avuto esito positivo

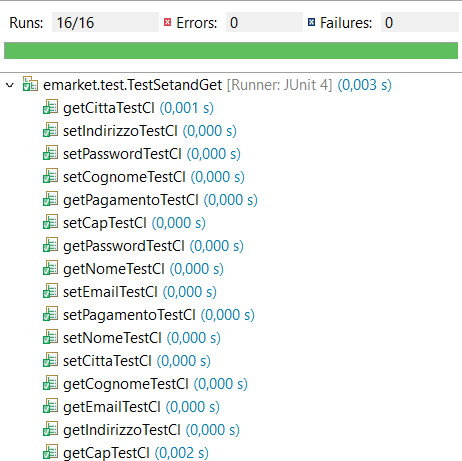


Figura 28: Esito Unit Test sui metodi Get e Set della classe Cliente.java

### Test API tramite Postman

In Figura [29](#_bookmark75) `e riportato il test delle API effettuato sulla chiamata GET *vi- sualizza Clienti* che restituisce tutti i clienti presenti nel database. Il metodo invocato `e *getAllCl()*.

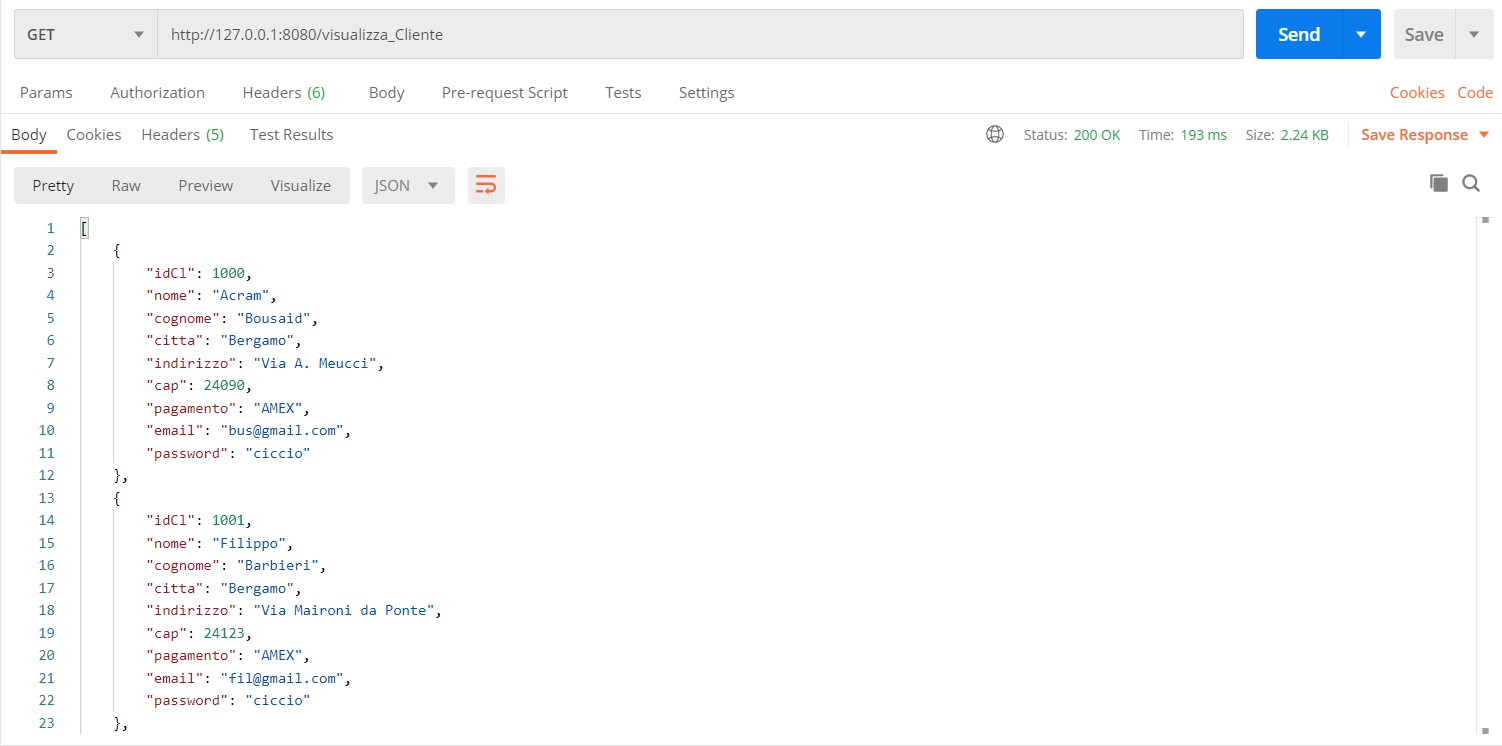


Figura 29: Chiamata GET

# Analisi Statica

Al termine dell’iterazione 3 `e stato utilizzato il tool CodeMr per effettuare un’analisi statica e della qualit`a 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.

## 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.

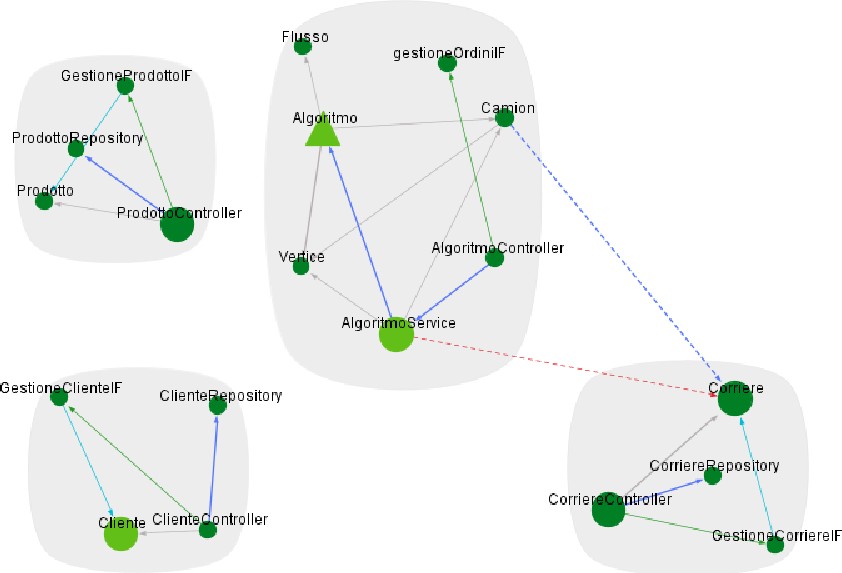


Figura 30: Grafo Strutturale

## 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 `e possibile interagire con il grafico e visualizzare nel dettaglio ci`o che accade all’interno di un singolo package.



Figura 31: TreeMap

## 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](#_bookmark82) sono riportati i valori di alcune delle metriche piu` importanti quali:

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

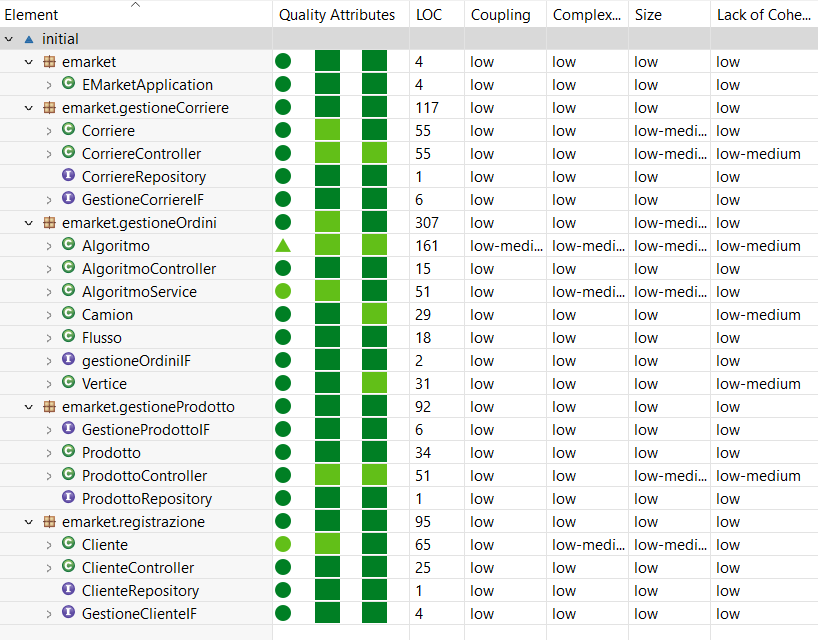


Figura 32: Project Outline