

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 |
|  |  | | | |  |
|  |  | | | |  |
|  |  | | | |  |

# 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

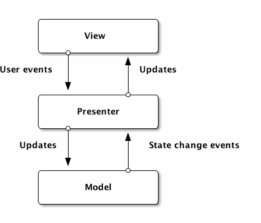
L’usabilit`a `e garantita dalla decisione di sviluppare il programma attraverso un’applicazione Android, di facile utilizzo sia per gli utenti che per il proprietario. Inoltre, `e possibile accedere al sistema tramite una pagina Web dotata di chiare e semplici interfacce.

### 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

# 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 un form che do ra compilare con varie informazioni. In base alle informazion inserite, tramite un algoritmo verrà calcolato il parcheggio più vicino a lui che rispetta che caratteristiche richieste. 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 dell’elenco 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 dell’elenco dei parcheggi

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

Attori coinvolti: Utente, Sistema.

### 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

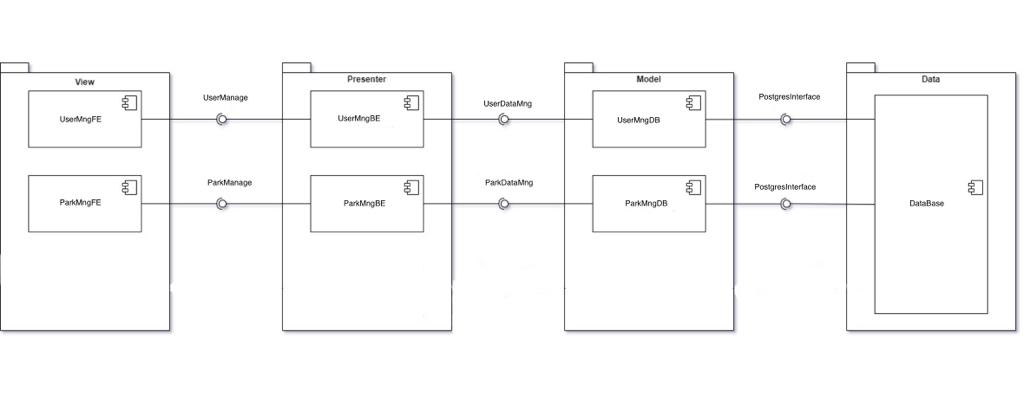


Figura 4: Component Diagram delle parti implementate

## UML Class Diagram per i tipi di dato

Il diagramma in Figura [5](#_bookmark33) mostra i tipi di dato necessari allo sviluppo del- l’applicazione.

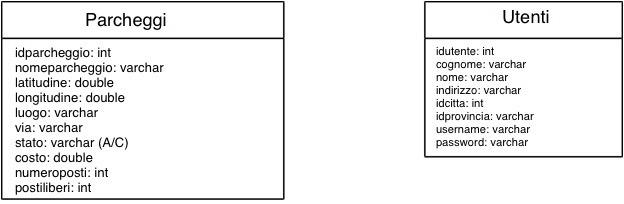


Figura 5: Class Diagram per i tipi di dato

## 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)

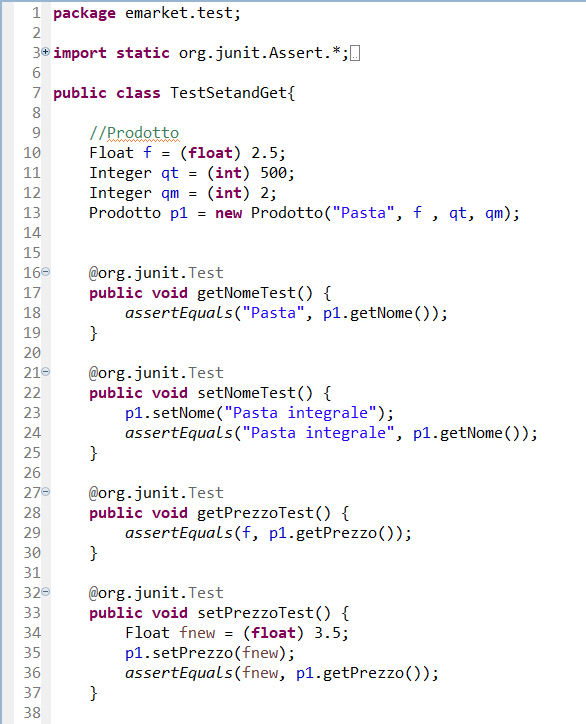


Figura 8: Codice Unit Test sui metodi Get e Set della classe prodotto.java

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

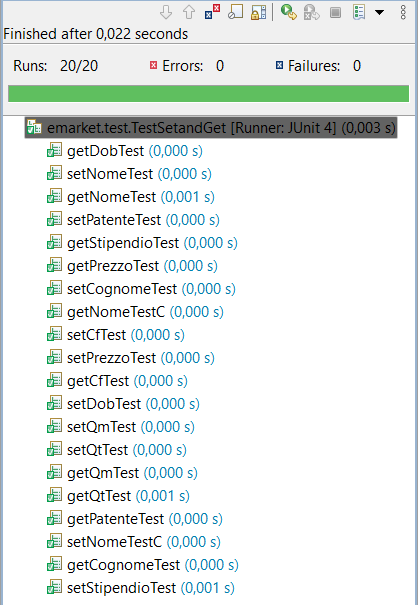


Figura 9: Risultato Unit Test sui metodi Get e Set della classe prodotto.java

### 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)*

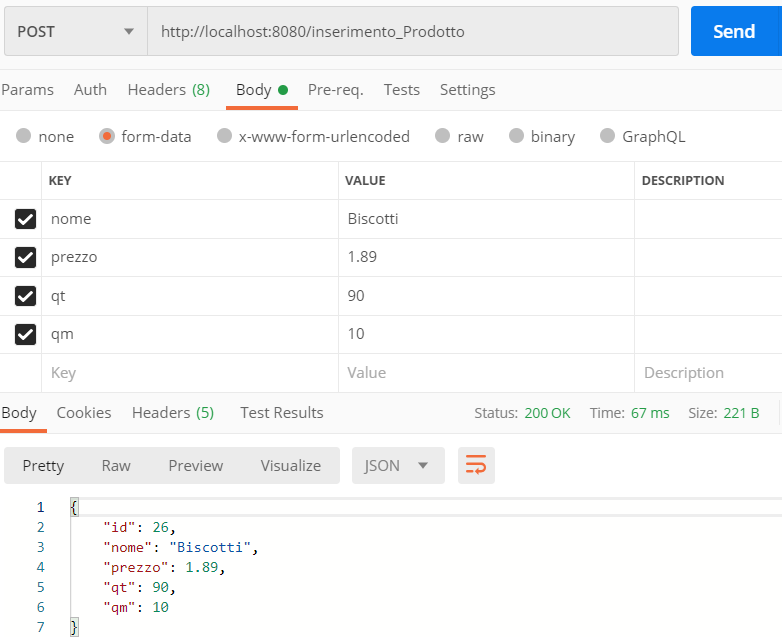


Figura 10: Chiamata POST inserisci prodotto

### 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()*

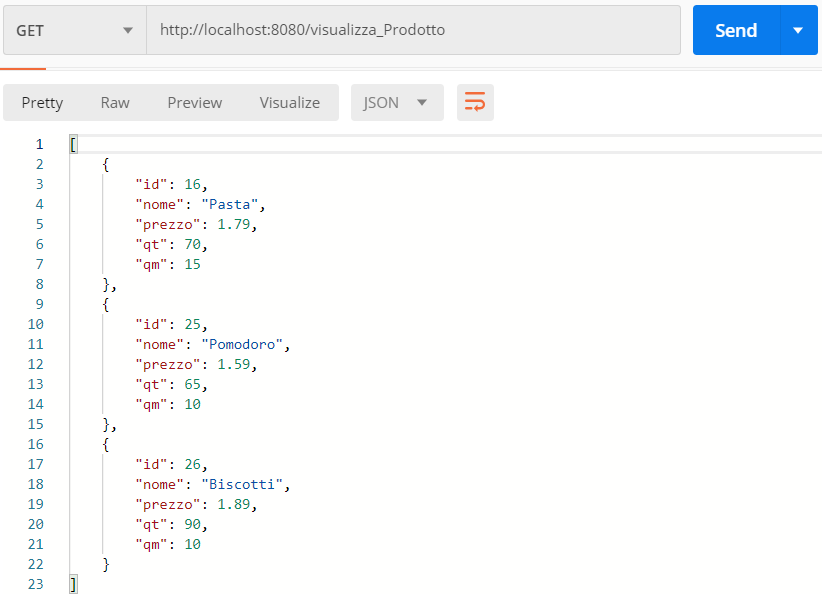


Figura 11: Chiamata GET visualizza prodotti

# 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.

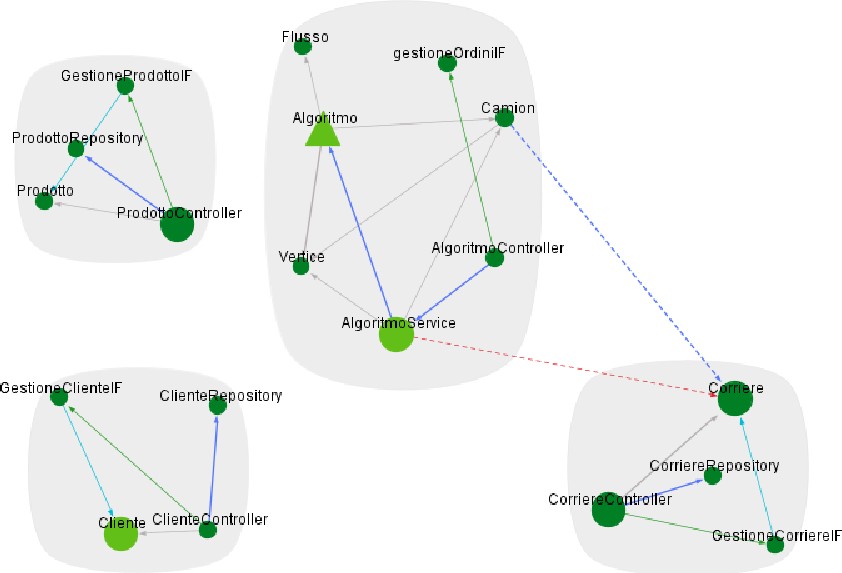


Figura 30: Grafo Strutturale

## 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

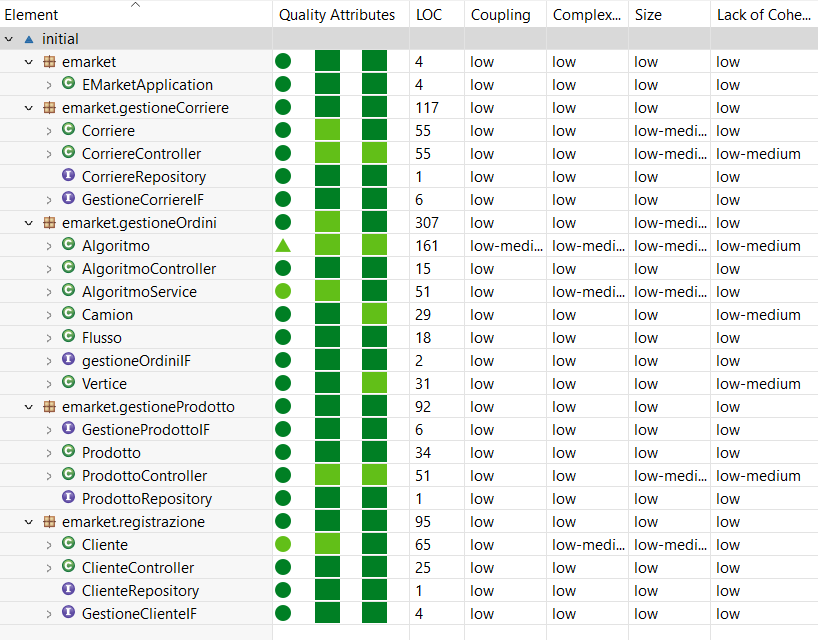


Figura 32: Project Outline