Architettura e Design

Gabriele Chignoli

Giugno 2025

Contents

1 Architettura

Pantrymanager è un applicativo che prevede l'utilizzo di:

- Un'interfaccia grafica per permettere all'utente di interagire con il sistema
- Un componente (logica dell'applicazione) che permetta di eseguire operazioni sui dati e fornire un feedback per l'interfaccia
- Un componente per conservare una moderata quantità di dati in modo consistente

Per questi motivi, lo stile architettonico **Model-View-Controller** (**MVC**) è stato scelto come modello più adeguato allo sviluppo dell'applicazione.

Attraverso il MVC il sistema viene suddiviso in macro-componenti, i quali gestiscono ciascuno una certa funzionalità del software; tale suddivisione consente al sistema un buon grado di modularità, che aiuterà il team sia nelle fasi di sviluppo che in quelle di manutenzione.

Di seguito viene mostrata la suddivisione prevista dallo stile MVC.

- **Model** - si occupa di elaborare i dati salvati (e.g. produrre una dieta personalizzata) e gestire la comunicazione con il Database.
- **View** - mostra all'utente le funzionalità disponibili (e.g. pulsanti per aggiungere/-modificare/rimuovere prodotti, creare una dieta personalizzata...), e fornisce l'output del Model all'utente (e.g. i parametri dei vari prodotti, i piatti disponibili...).
- **Controller** - gestisce gli input dell'utente, fornendo un feedback grafico, e comunica direttamente con il model.

1.1 Architectural Views

1.1.1 Punti di vista del Modulo (Vista Statica)

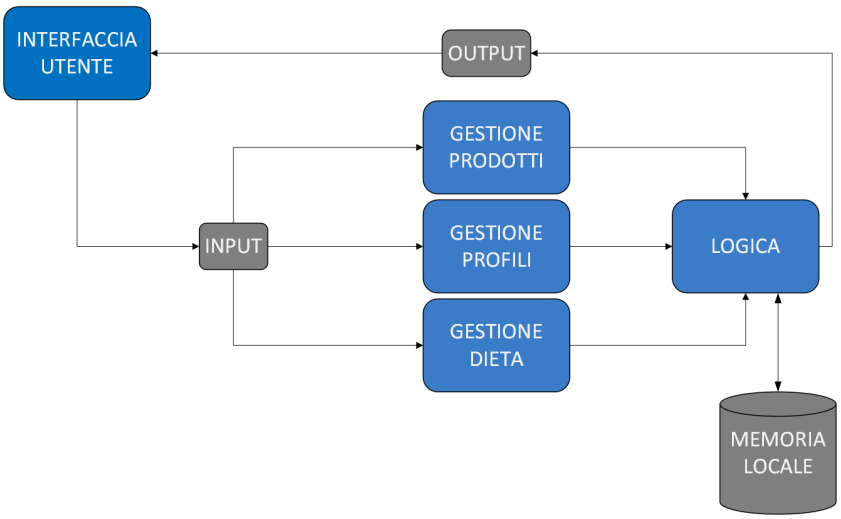


Figure 1: Punto di vista del modulo con relazione "Usa"

1.1.2 Punti di vista dei Componenti e dei Connettori (Vista Dinamica)

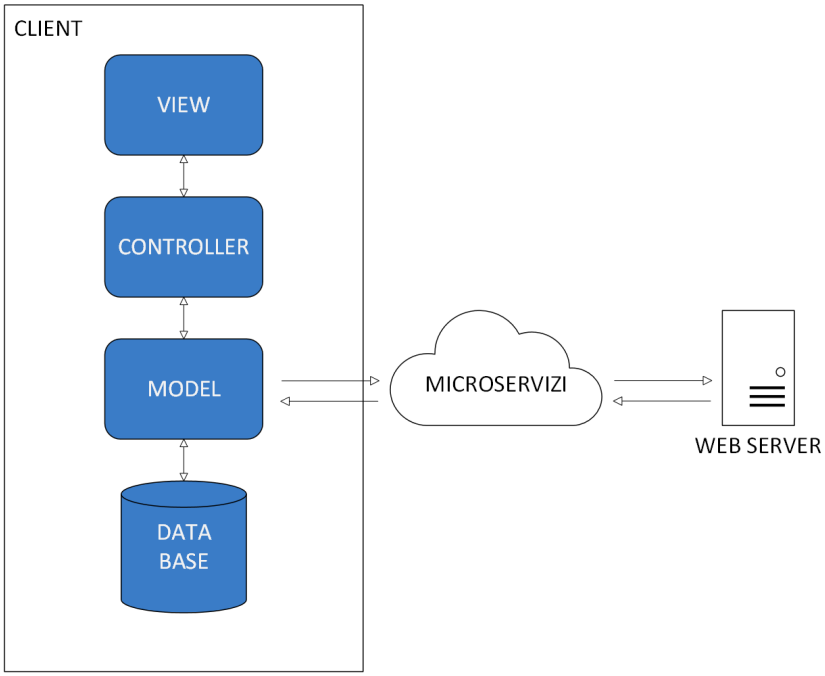


Figure 2: Punto di vista dei componenti e dei connettori

La struttura del client è quella che verrà effettivamente sviluppata e utilizzata come riferimento per lo sviluppo del software; i microservizi sono considerati un possibile ampliamento dell'architettura ma che non sarà implementata nella versione finale del software. Inoltre, le relazioni simboleggiate da due frecce distinte in direzione opposta indicano che la relazione è su richiesta, e non un continuo scambio di dati tra i componenti.

2 Design

Viene presentato di seguito il Diagramma delle Classi, volto a mostrare come si intende effettivamente implementare una delle parti costitutive del software. In particolare viene mostrato quali saranno le classi principali, i loro metodi e attributi e le loro relazioni (gerarchia) e interazioni (comunicazione).

Attraverso Papyrus è possibile generare il codice direttamente dal diagramma UML costruito, tuttavia son stati riscontrati errori (si pensa durante l'installazione) che non permettono il pieno funzionamento dello strumento. Per questo si è deciso di scrivere il codice manualmente utilizzando il diagramma come guida da seguire il più precisamente possibile; si tenterà di tenere aggiornate e coerenti entrambe le versioni.

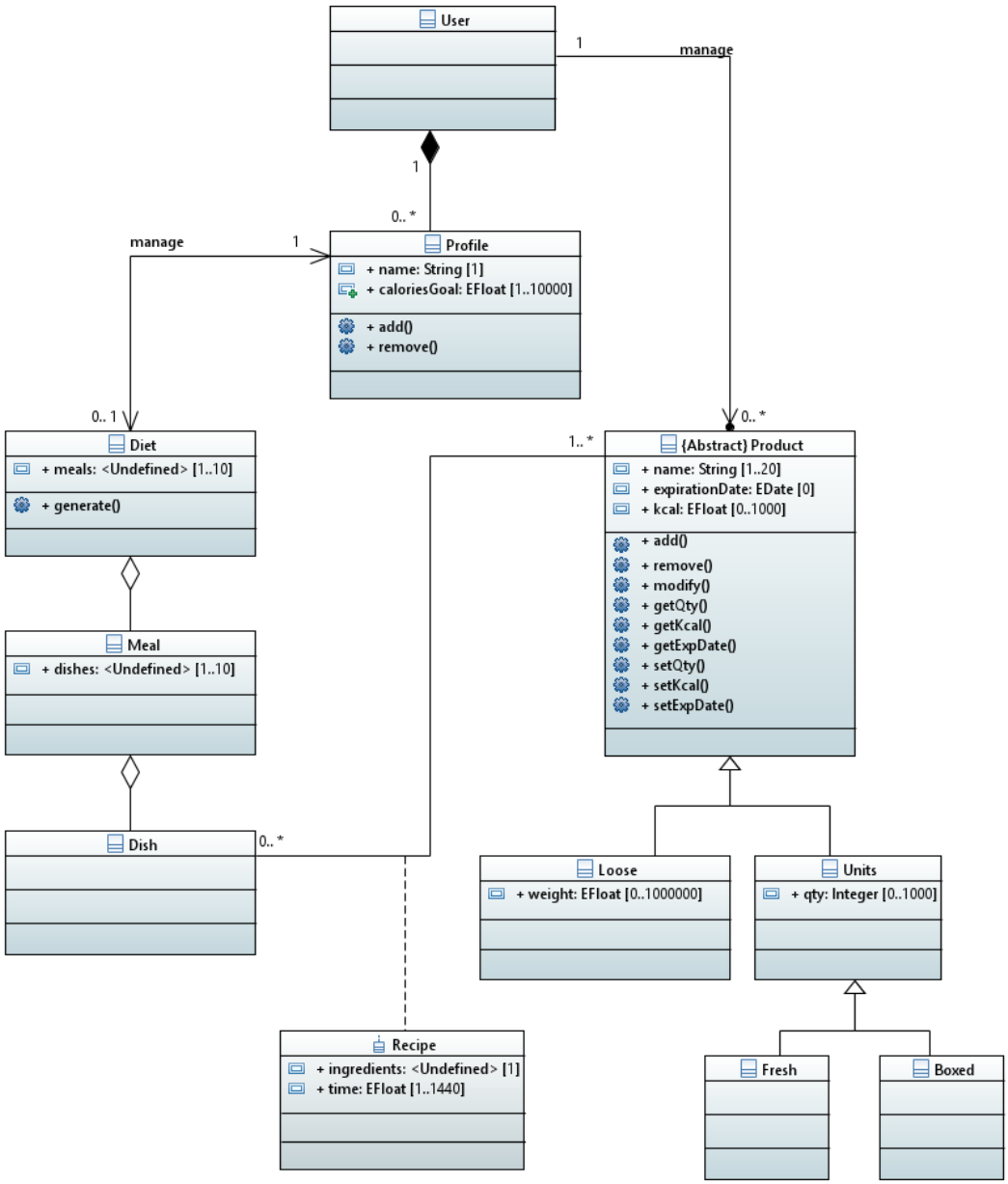


Figure 3: Diagramma delle Classi v0

Si invita il lettore a visionare i diagrammi dalla cartella `/UML` della documentazione, dove sono disponibili le immagini nelle dimensioni originali.

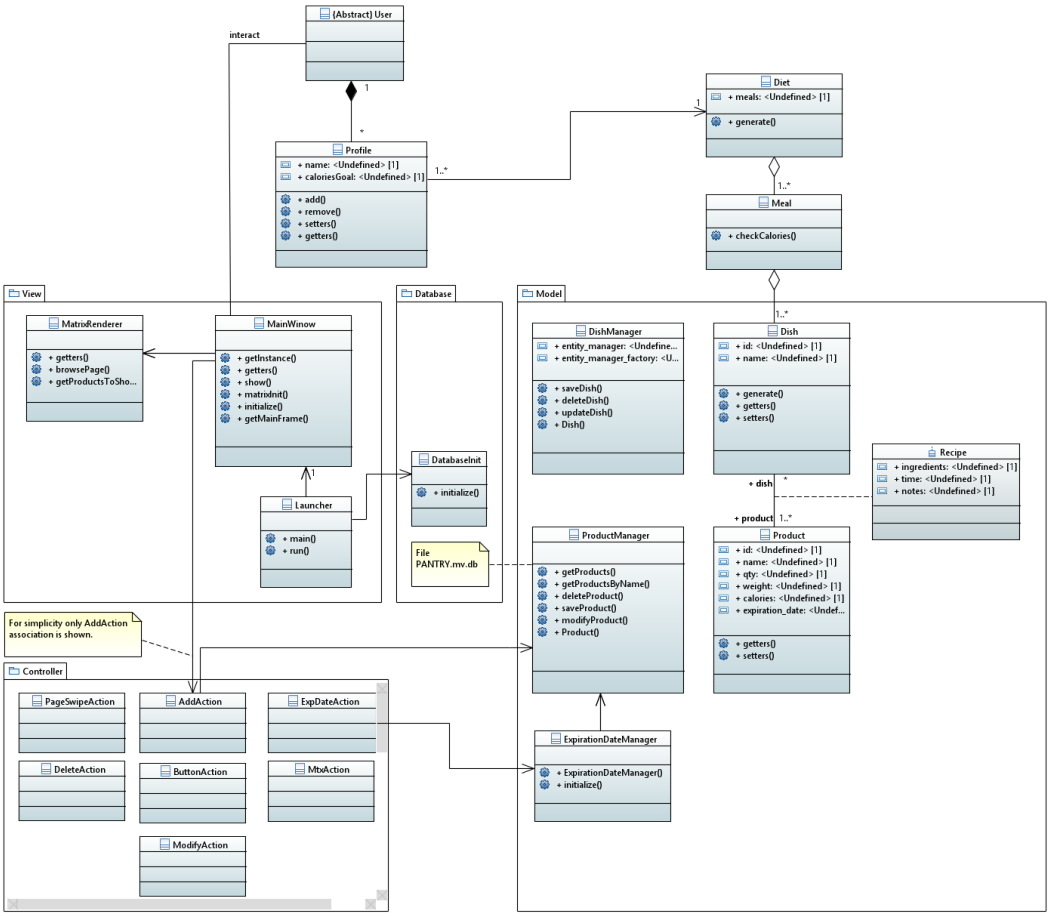


Figure 4: Diagramma delle Classi v1

Viene inoltre presentato il Diagramma delle Componenti (*Figure 5*) al fine di mostrare le interazioni tra le macro-componenti del sistema e le interfacce che questi ultimi devono disporre e utilizzare per garantire il funzionamento dell'applicazione.

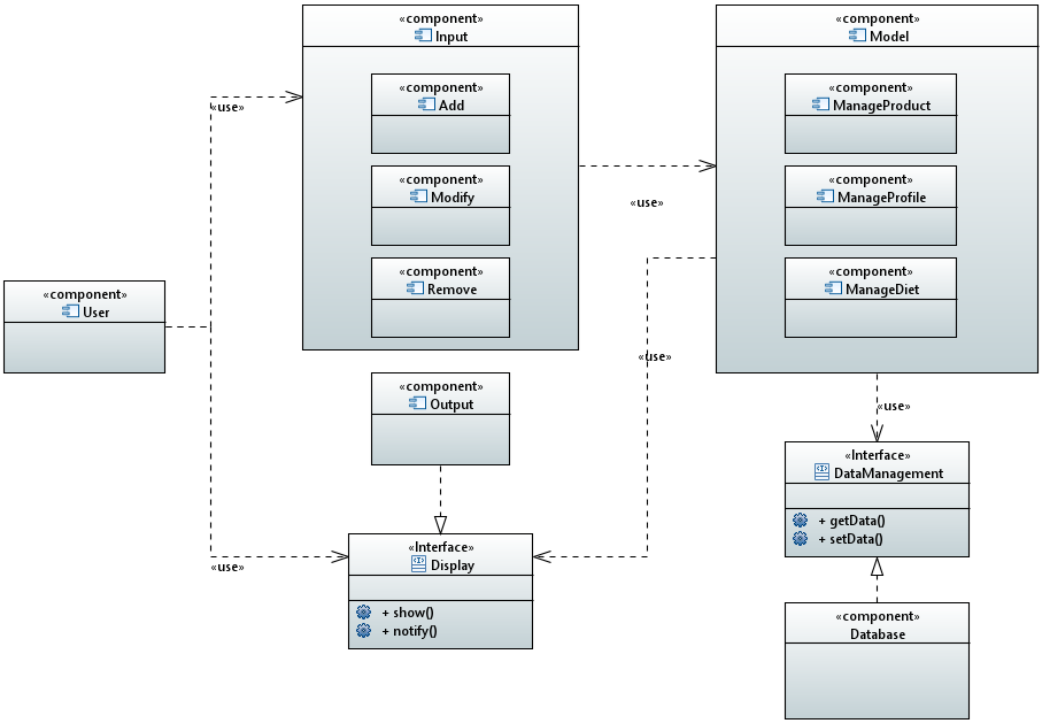


Figure 5: Component Diagram

2.1 Gestione dell'interfaccia

L'interfaccia grafica per l'utente è stata sviluppata utilizzando il framework **Swing** per Java. Swing permette di costruire finestre e di popolarle con pulsanti, campi e etichette attraverso codice (o utilizzando tool grafici). Il framework prevede che, per garantire l'interazione dell'interfaccia con il resto del sistema, venga utilizzato il **paradigma degli eventi**: una interazione dell'utente con l'interfaccia grafica genera un evento che viene catturato da delle componenti (detti listener), che in risposta eseguono altro codice.

In Pantrymanager, ai pulsanti dell'interfaccia (che fa parte del `package view`) viene assegnata un'azione, il cui evento viene catturato dalle classi listener (contenute nel `package controller`), che invocano metodi (definiti nelle classi del `package model`) che agiscono poi effettivamente sui dati in memoria (in lettura e scrittura).

Come è effettivamente organizzata la struttura può essere osservato direttamente dal folder del progetto, oppure dal diagramma delle classi sopra presentato.

2.2 Struttura del Database

Pantrymanager necessita di avere copie persistenti dei dati su cui deve operare; per raggiungere questo obiettivo si è deciso di utilizzare un **database embedded, in-memory**, implementato nell'applicativo attraverso il sistema di gestione di database relazionali **H2**, che permette di creare un database Java SQL.

Per rendere la gestione e la scrittura di codice più semplice, viene integrata la piattaforma **Hibernate**, un middleware che permette il mapping tra la natura ad oggetti di Java e quella relazionale del database, accoppiata all'interfaccia **JPA (Jakarta Persistence API)**, che permette la scrittura di query SQL attraverso la notazione puntata della codifica Object Oriented.

2.3 Design-Patterns

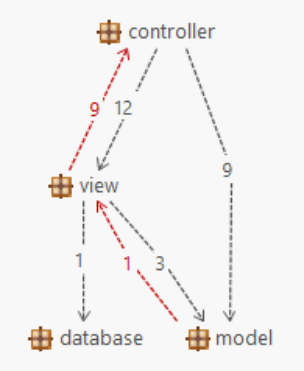
Singleton La classe `MainWindow`, che si occupa di generare e popolare la finestra con cui l'utente interagisce, segue il paradigma del *singleton*: è possibile creare una sola istanza della classe, e si accede ad essa solamente attraverso un metodo pubblico che restituisce tale istanza.

```
public static MainWindow getInstance() {
    if(instance == null) {
        instance = new MainWindow();
    }
    return instance;
}
```

2.4 Complessità - *Stan4j*

Le metriche che vengono presentate in questa sezione sono state estrapolate dal progetto in modo automatico attraverso l'utilizzo del tool *Stan4j*, che permette di eseguire l'analisi statica di un sistema e le sue componenti.

2.4.1 Generale



Metrica	Valori
Cyclomatic Complexity (CC)	1.39
Fat - Libraries	1
Fat - Packages	6
Fat - Units	48
Tangled	28.57%
Distance (D)	-0.62

Figure 6: Grafo dipendenze progetto

Osservazioni

- La complessità è buona (probabilmente dovuto alla semplicità del sistema)
- Assente l'utilizzo di classi astratte e interfacce, segnale negativo data la natura ad oggetti di Java
- Ci sono dipendenze cicliche nel grafo delle dipendenze
  - `view` e `model` sono legati da `ExpirationDateManager.class`, che chiede l'istanza della finestra principale. Si ritiene che sia immediato il fix.
  - `view` e `controller` sono legati da tutte le azioni dei pulsanti. Richiede un'analisi più approfondita.

2.4.2 Model

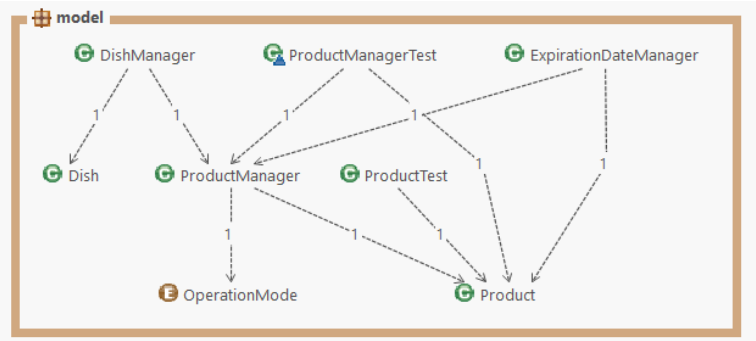


Figure 7: Grafo dipendenze classi

Figure 8: Grafo dipendenze pacchetti

Metrica	Valori
Cyclomatic Complexity (CC)	1.28
Fat	9
Afferent Coupling (Ca)	8
Efferent Coupling (Ce)	1
Instability (I)	0.11
Abstractness (A)	0
Distance (D)	-0.89

Osservazioni

- La complessità è buona
- La stabilità è soddisfacente, il package dipende poco dagli altri componenti
- Assente l'utilizzo di classi astratte e interfacce, segnale negativo data la natura ad oggetti di Java
- Distanza troppo alta, il pacchetto è troppo poco astratto

2.4.3 View

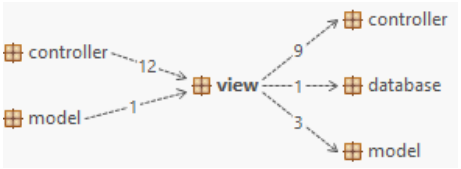


Figure 10: Grafo dipendenze pacchetti

Metrica	Valori
Cyclomatic Complexity (CC)	1.26
Fat	4
Afferent Coupling (Ca)	9
Efferent Coupling (Ce)	12
Instability (I)	0.57
Abstractness (A)	0
Distance (D)	-0.43

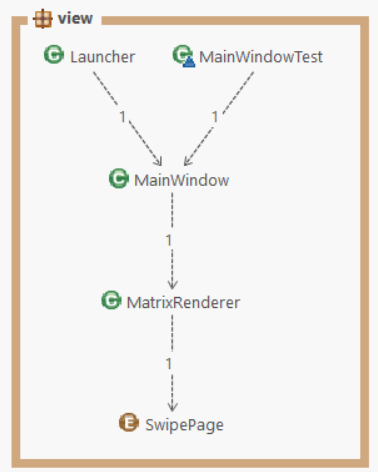


Figure 9: Grafo dipendenze classi

Osservazioni

- La complessità è buona
- L'instabilità richiede attenzione: dal package dipendono molte classi, ma esso stesso ha molte dipendenze
- Astrazione assente
- Distanza gestibile

2.4.4 Controller

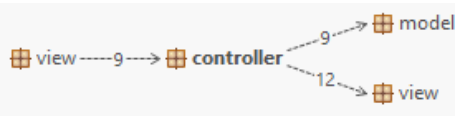


Figure 11: Grafo dipendenze pacchetti

Metrica	Valori
Cyclomatic Complexity (CC)	1.59
Fat	0
Afferent Coupling (Ca)	1
Efferent Coupling (Ce)	6
Instability (I)	0.86
Abstractness (A)	0
Distance (D)	-0.14

Osservazioni

- L'instabilità del è troppo alta, il pacchetto non fornisce abbastanza servizi pur dipendendo da molte altre classi (dato dalla natura del pacchetto, in cui sono contenute le azioni dei pulsanti dell'interfaccia)
- Astrazione assente
- Distanza buona (dovuta all'instabilità molto alta)

2.5 Complessità - *CodeMR*

Stan4j non fornisce per tutte le metriche che calcola dei range per giudicare i valori estrapolati; per questo motivo si è deciso di confrontare i risultati con un altro tool, *CodeMR*, che fornisce un'analisi statica più ampia, graficamente leggibile e soprattutto fornisce un feedback al programmatore più chiaro e diretto.

Per accedere al report (fornito in formato `html`), dopo aver scaricato il progetto, aprire il file `index.html` nella directory

```
docs/Architettura_e_design/codemr/html/main_report/...
```

L'analisi evidenzia una coesione bassa (*medium-high lack of cohesion*) all'interno della classe `Product`. Si pensa che questo sia dovuto al fatto che si tratta di una classe che serve solo come modello per costruire la struttura del database e per istanziare i prodotti, presentando quindi solamente le variabili necessarie e i metodi getter e setter.