HealthUp

PROGETTO DI PROGRAMMAZIONE AD OGGETTI

Studente: Matteo Bando

Matricola: 1226287

Professore: Francesco Ranzato

Corso: Programmazione ad oggetti

Università: Università degli Studi di Padova

Contenuti

1 PROGRAMMA

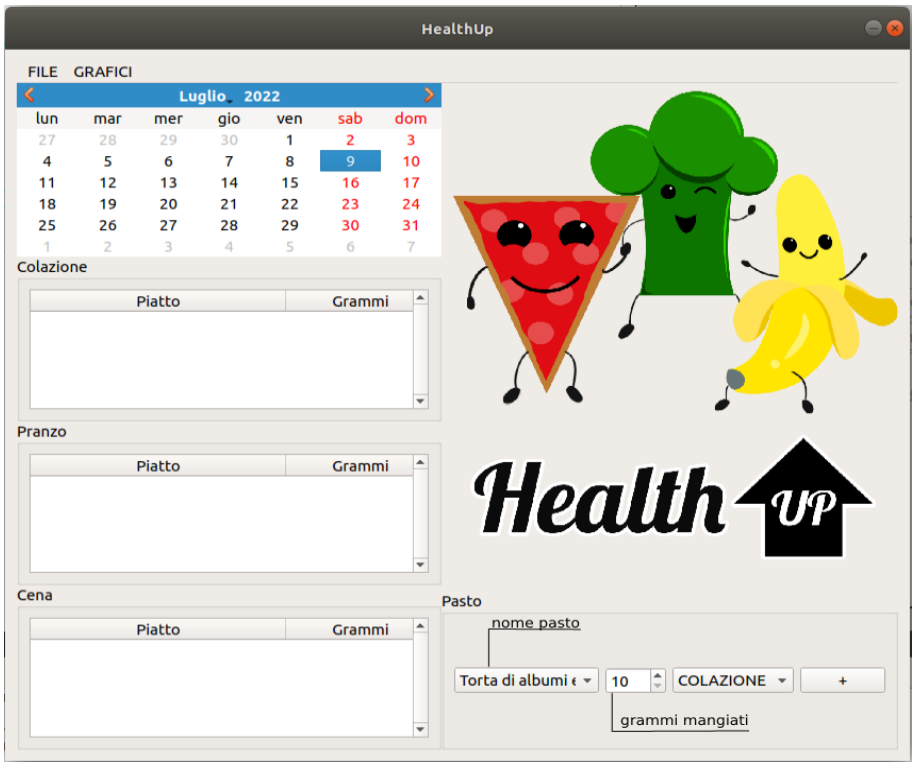
1.1 Introduzione e descrizione programma

HealthUp è un programma sviluppato per chi vuole seguire uno stile di vita sano. Permette infatti di tenere traccia degli alimenti assunti, così da potersi gestire i vari pasti della giornata.

2 MANUALE GUI

2.1 Pagina Principale

La pagina principale risulta semplice ed intuitiva:



2.1.1 Menù File

Premendo il menù **File** ci compiranno due sottomenù:

- Apri : Apre il il file *save.json*
- Salva : Salva tutti gli input in un file chiamato *save.json*

2.1.2 Menù Grafici

Saranno disponibili tre possibilità premendo il menù **Grafici**

- Macronutrienti giornalieri Figura 1a
- Andamento mensile assunzione calorie Figura 1b
- Andamento settimanale assunzione delle calorie Figura 1c

2.2 Macronutrienti Giornalieri

La pagina si presenta con un Pie Chart (Figura 1a) dove mostra i grammi assunti suddivisi in **Proteine**, **Grassi** e **Carboidrati**.
I valori presenti sono proporzionati su grammi 100 di ogni pasto selezionato nella schermata principale, ergo per avere una buona rappresentazione dei dati è opportuno inserire una quantità discreta di grammi.

2.3 Andamento mensile assunzione calorie

La pagina si presenta mostrando un Bar Chart (Figura 1b) dove nell'asse delle ascisse prendono posto i giorni tracciati dall'utente. All'interno di ogni blocco si trova il corrispondete numero di calorie assunte nel corrispondete giorno.

2.4 Andamento settimanale assunzione delle calorie

La pagina ci mostra un Line Chart (Figura 1c) che rappresenta il numero di calorie assunte durante la settimana selezionata dall'utente nel calendario. Per ottenere un risultato ottimale è opportuno che tutti i giorni della settimana in questione siano stati tracciati.

3 FUNZIONALITÀ OFFERTE

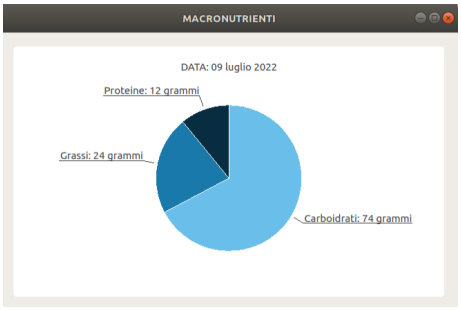
3.1 Salvataggio su .JSON

Si offre la possibilità di salvare i diversi input dell'utente in un file in formato JSON che prenderà il nome di **save.json**.

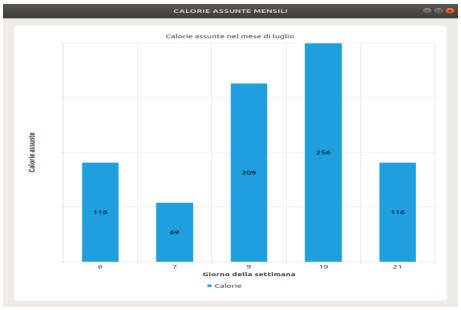
3.2 Apertura files .JSON

Si offre all'utente l'apertura di un salvataggio **save.json** precedentemente creato.

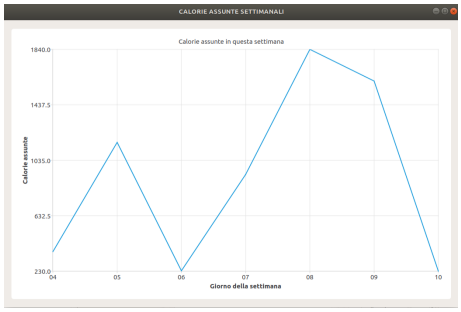
3.3 Visualizzazione grafici



(a) Macronutrienti giornalieri

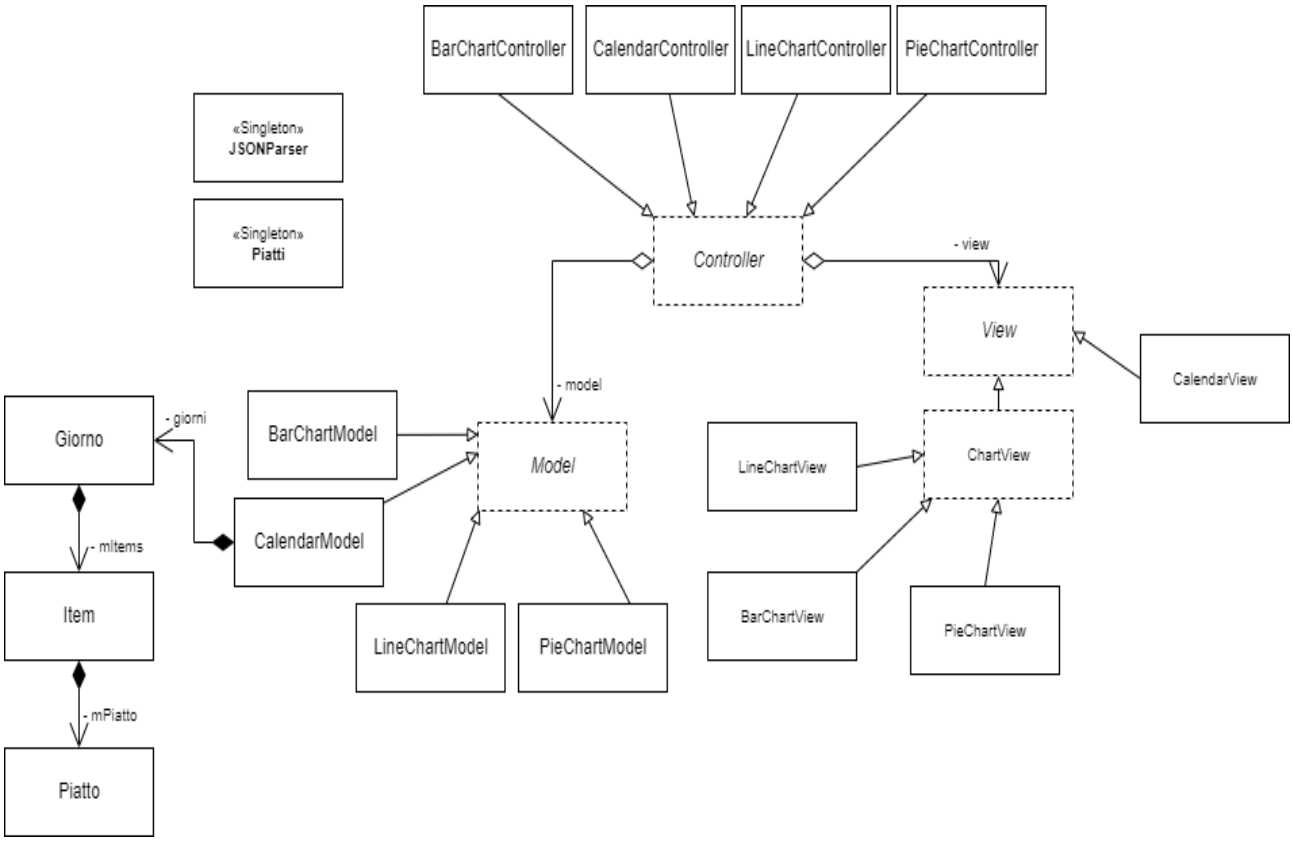


(b) Andamento mensile assunzione calorie



(c) Andamento settimanale assunzione delle calorie

4 GERARCHIA DI TIPI



Si è deciso di seguire due patterns per lo sviluppo del progetto:

MVC Per separare la logica di presentazione dei dati dalla logica di business

Singleton per la garanzia che di una determinata classe venga creata una e una sola istanza

4.1 Spiegazione gerarchia di tipi

Vi sono quattro principali classi basi astratte:

Controller

Model

View

ChartView

4.1.1 Controller

Il Controller è quella classe che fa da collegamento tra Model e View mediante SIGNALS emanati dalla vista e collegati agli SLOT del controller in modo che apportino modifiche al model. Una volta apportate le modifiche al model si lascia sempre alla classe Controller il compito di segnalare alla View che ci sono nuovi dati da mostrare. Da questa classe derivano e implementano i metodi virtuali puri:

CalendarController si occupa della comunicazione tra la vista che si occupa di rappresentare il la schermata principale ed il model che si occupa della gestione dei dati della schermata principale. Si occupa inoltre di creare il file **piatti.json** in caso mancasse all'avvio del programma. Sempre lo stesso, si occupa di segnalare all'utente un errore in caso il file **save.json** non fosse corretto o non presente.

BarChartContoller si occupa della comunicazione tra la vista che si occupa di rappresentare il BarChart ed il model che si occupa della gestione dei dati del BarChart.

LineChartController si occupa della comunicazione tra la vista che si occupa di rappresentare il LineChart ed il model che si occupa della gestione dei dati del LineChart

PieChartContoller si occupa della comunicazione tra la vista che si occupa di rappresentare il PieChart ed il model che si occupa della gestione dei dati del PieChart

4.1.2 Model

Si occupa della rappresentazione dei dati. Incapsula la logica business ed i dati.
Da questa classe derivano:

CalendarModel Rappresenta tutti i dati della scheramata CalendarView

Giorno Rappresenta il singolo giorno del calendario. Contiene sia la data che la lista di items(pasti) divisi per tipo consumati dall'utente

Item Rappresenta il singolo pasto (Piatto, grammi e tipo) consumato durante la giornata dall'utente

Piatto Rappresenta il singolo piatto con i suoi valori nutrizionali (per grammi 100)

BarChartModel Rappresenta tutti i dati della scheramata BarChartView

LineChartModel Rappresenta tutti i dati della scheramata LineChartView

PieChartModel Rappresenta tutti i dati della scheramata PieChartView

4.1.3 View

E' la parte visuale di una schermata.
Da questa classe derivano:

CaledarView Ha il compito di far visualizzare i dati della schermata principale

ChartView Classe astratta che ha il compito di rappresentare un generico Chart.Esso aggiunge 2 campi dati marcati *protected* che accomunano tutte le classi derivanti. Da questa classe derivano ed implementano i metodi virtuali puri:

LineChartView Ha il compito di far visualizzare i dati del LineChartModel

BarChartView Ha il compito di far visualizzare i dati del BarChartModel

PieChartView Ha il compito di far visualizzare i dati del PieChartModel

4.2 Singleton

Per la garanzia che di una determinata classe venga creata una e una sola istanza si sono costruite due classe seguento il Singleton Pattern:

JSONParser Ha il compito di salvare o creare un file di formato .json

Piatti Contiene tutti i piatti con i relativi valori nutrizionali caricati dal file *piatti.json*

5 USO DEL POLIMORFISMO

5.1 Distruttori

Il distruttore (profondo) è stato dichiarato virtuale così quando viene chiamatato il distruttore nel Controller la *delete* invocherà il distruttore del tipo dinamico in modo corretto:

```
Controller::~Controller(){
    while ( QWidget* w = findChild<QWidget*>() )
        delete w;

    delete model;
    delete view;
}
```

5.2 void Controller::ShowView() const

Nel main la chiamata *ShowView* dichiarata virtuale, invocherà il metodo overridato nel tipo dinamico.

```
Controller* controller = new CalendarController(view);
controller->ShowView();
```

Così se in futuro ci dovessero essere altre schermate prima della CalendarView, basterà creare un altro controller derivante dalla classe *Controller*, utilizzarlo con la *new* e invocare il metodo *ShowView()*, il quale come detto in precedenza invocherà il metodo corretto del tipo dinamico.

5.3 virtual Model* getModel() const = 0;

Nella classe derivante per non poter essere più considerate astratte dovranno overridare il metodo. Un esempio:

```
CalendarModel* CalendarController::getModel() const{
    return static_cast<CalendarModel*>(model);
}
```

che ritorna il tipo corretto per covarianza.

5.4 virtual View* getView() const = 0;

Nella classe derivante per non poter essere più considerate astratte dovranno overridare il metodo. Un esempio:

```
CalendarView* CalendarController::getView() const{
    return static_cast<CalendarView*>(view);
}
```

che ritorna il tipo corretto per covarianza.

5.5 virtual void showChart() = 0;

Questa chiamata definita come virtuale pura nel file chartview.h

```
virtual void showChart() = 0;
```

Viene overridata in ogni classe derivante di showChart e in base al tipo dinamico chiamante, setterà le impostazioni del proprio grafico. Se per esempio il tipo dinamico sarà un PieChartView*, allora verrà eseguito il metodo

```
void PieChartView::showChart(){
......
}
```

6 DESCRIZIONE DEI FILE UTILIZZATI PER L' INPUT E L' OUTPUT

6.1 Salvataggio

Si offre la possibilità di salvare i diversi input dell'utente in un file in formato JSON che prenderà il nome di **save.json** La classe che offre queste funzionalità segue il Singleton Pattern in quanto è necessaria un'unica istanza della classe per tutto il ciclo di vita del programma

6.2 Apertura

L' apertura di un file .json avviene in due parti del programma.

Apertura "save.json" La prima viene offerta come funzionalità all'utente, il quale può aprire un suo vecchio salvataggio (che dovrà essere per forza chiamato save.json e trovarsi nella directory del programma) e permette di caricare nel programma i vecchi dati precedentemente salvati. Se il file non dovesse essere presente oppure non dovesse contenere le parole chiave *data* o *items* il programma segnalerà un errore. All'apertura del salvataggio la data selezionata sarà spostata nella data odierna.

Apertura "piatti.json" La seconda invece, avviene quando si avvia il programma: esso infatti necessita necessariamente di un file chiamato piatti.json che rappresenta tutti i valori dei piatti necessari al corretto funzionamento del programma. Se non dovesse essere presente, si occuperà il programma di crearne uno con dei piatti preimpostati

6.2.1 File di prova

Nel progetto sono stati lasciati due file per provare il corretto funzionamento del programma:

piatti.json Lista di piatti pre caricati

save.json Salvataggio di prova

7 INDICAZIONE DI ISTRUZIONE ED ESCUZIONE

7.1 Compilazione

Il progetto viene munito del proprio file .pro chiamato **HealthUp.pro** il quale deve essere *necessariamente e obbligatoriamente* utilizzato per compilare il progetto

7.1.1 Istruzioni di compilazione (steps)

1. `qmake`
2. `make`

7.1.2 Pacchetti necessari

- qt5-default
- libqt5charts-dev

I pacchetti sono ottenibili con il comando

```
sudo apt install qt5-default libqt5charts5-dev
```

7.2 Test

Come precedentemente detto in 6.2.1 vengono dati due file per provare il programma. Basterà digitare

```
./HealthUp
```

oppure fare doppio click sull'icona per testare il programma.

8 CONCLUSIONE

8.1 Ore di lavoro

Operazione	Ore
Analisi dei requisiti	1
Sviluppo models	21
Sviluppo views	15
Sviluppo controllers	6
Sviluppo helpers	4
Apprednimento liberia Qt	10
Bugs fix finali	2
Grafiche e icone	1
ORE TOTALI	60

Sono state superate le ore di lavoro perchè ci sono stati diversi problemi nel corso dello sviluppo e sono state spese diverse ore nel capire i motivi dei vari crash. In tutte le ore degli "sviluppi" sono calcolate anche le ore di debugging e fix tra i collegamenti Model-View-Controller.

8.2 Ambiente di sviluppo

Il progetto è stato sviluppato sulla macchina virtuale che ci è stata fornita.

Sistema operativo	Ubuntu 18.04.3 LTS
Compilatore	gcc 7.3.0
IDE	Qt Creator 4.5.2
Versione Qt	5.9.5