QANALYTICS

Relazione progetto Programmazione ad Oggetti
A.A. 2019/20

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Gianpiero Giuseppe Tovo 1193350
Ayoub Maher 1187406
Giacomo Sassaro 1187566

Abstract

L'applicazione ***QANALYTICS*** è uno strumento di social media managing utile ad analizzare e visualizzare statistiche ricavate dai profili dei propri clienti.
Le informazioni vengono disposte in grafici popolati, concettualmente, tramite l'interfacciamento dell'applicazione ad un database o alle API dei social network; operazione che a fine esemplificativo è stata realizzata come una deserializzazione di un file *JSON.*

Una volta scelta la fonte dei dati e selezionato il cliente da esaminare, è possibile visualizzarne I dati personali e le diverse statistiche dei social network supportati, singolarmente o a confronto tra di loro, con la possibilità di salvare un'istantanea dei grafici in formato *PNG*.

Dall'interfaccia grafica sono disponibili solamente statistiche relative agli account ma la logica del programma è predisposta anche alla gestione e visualizzazione dei singoli contenuti.

Per la realizzazione dei grafici l'applicazione sfrutta il modulo ***QTCHARTS*** di Qt e necessita di flag specificati alla fine del documento.

Implementazione MVC

La classe ***Controller*** implementa il controller che comunica con il modello, rappresentato dalla classe ***Model***, e con la vista, composta dalle finestre ***Landingwindow*** e ***Graphswindow***, tramite i riferimenti a questi nei campi dati.

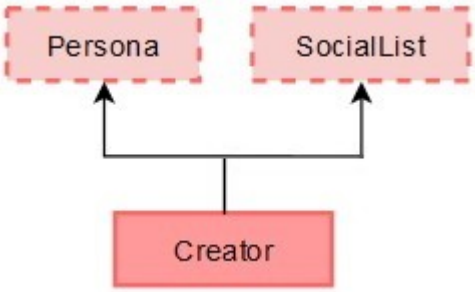
Il controller, che all'avvio dell'applicazione importa I dati tramite la funzione ***importData*** e popola il modello, risponde quindi ai segnali delle finestre, ne permette la visualizzazione dei grafici richiesti e gestisce l'operazione di esportazione dei dati dallo slot ***exportBtnClick***.
I grafici vengono costruiti nel modello che raccoglie quindi le informazioni necessarie e li invia al controller che li dispone nel layout adeguato su `Graphswindow`.
La funzione di salvataggio dell'istantanea dei grafici viene invece gestita interamente dalla `Graphswindow`.

Viene rispettata l'autonomia del modello dal controller mentre è presente una certa dipendenza dal framework *Qt* per quanto riguarda l'utilizzo dei tipi e la creazione dei grafici, che viene compiuta nella classe `Model`.

Nella logica del programma è largamente utilizzato il marcatore *const,* mentre è rara la presenza di metodi *set* per i campi dati per via della premesse dell'applicazione, intesa solamente per la rappresentazione grafica "in sola lettura" delle statistiche scelte.

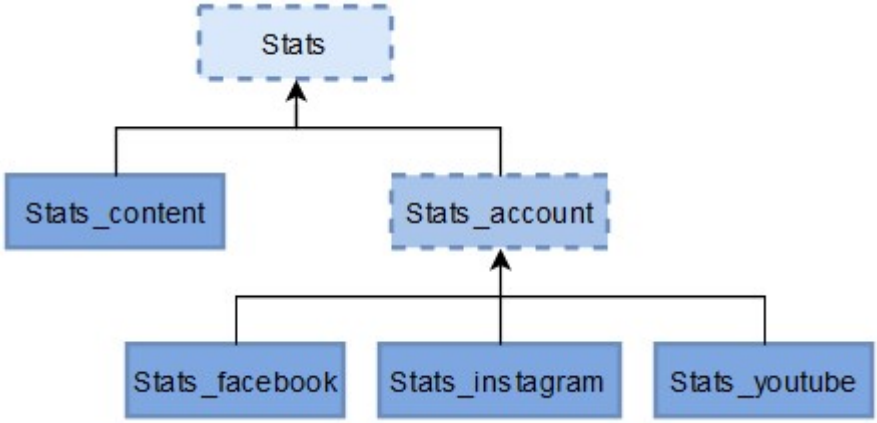
Gerarchia di tipi e Contenitori

Gli oggetti ***`Creator`***, raccolti in un oggetto ***`CreatorList`*** accessibile dal modello, derivano dalle classi astratte *`Persona`*, per memorizzare le generalità dei clienti e dalla classe ***`SocialList`*** per gli account corrispondenti.



Gerarchia Creator

Nel contenitore ***`StatsList`*** vengono memorizzare le statistiche per mezzo di ***`Stats`***: una classe polimorfa e astratta utile a raccogliere le statistiche comuni di contenuti ed account, che viene resa concreta nelle classi **`Stats_content`** e, dopo un'ulteriore derivazione astratta intermedia di ***`Stats_account`***, in ***`Stats_facebook`***, ***`Stats_instagram`*** e ***`Stats_youtube`***.



Gerarchia Stats

Le classi contenitori ***`SocialList`*** e ***`CreatorList`*** sono implementate rispettivamente come vettori di *`Account`* e *`Creator`*, mentre ***`StatsList`*** contiene una lista doppiamente linkata di oggetti polimorfi *`Stats`*, dotata di campi testa e coda, iteratore e un'interfaccia minimale per la sua gestione (costruttore di copia, assegnazione, distruzione profonda, operatore di subscripting, rimozione nodi e informazioni sulla capacità della lista).

Polimorfismo

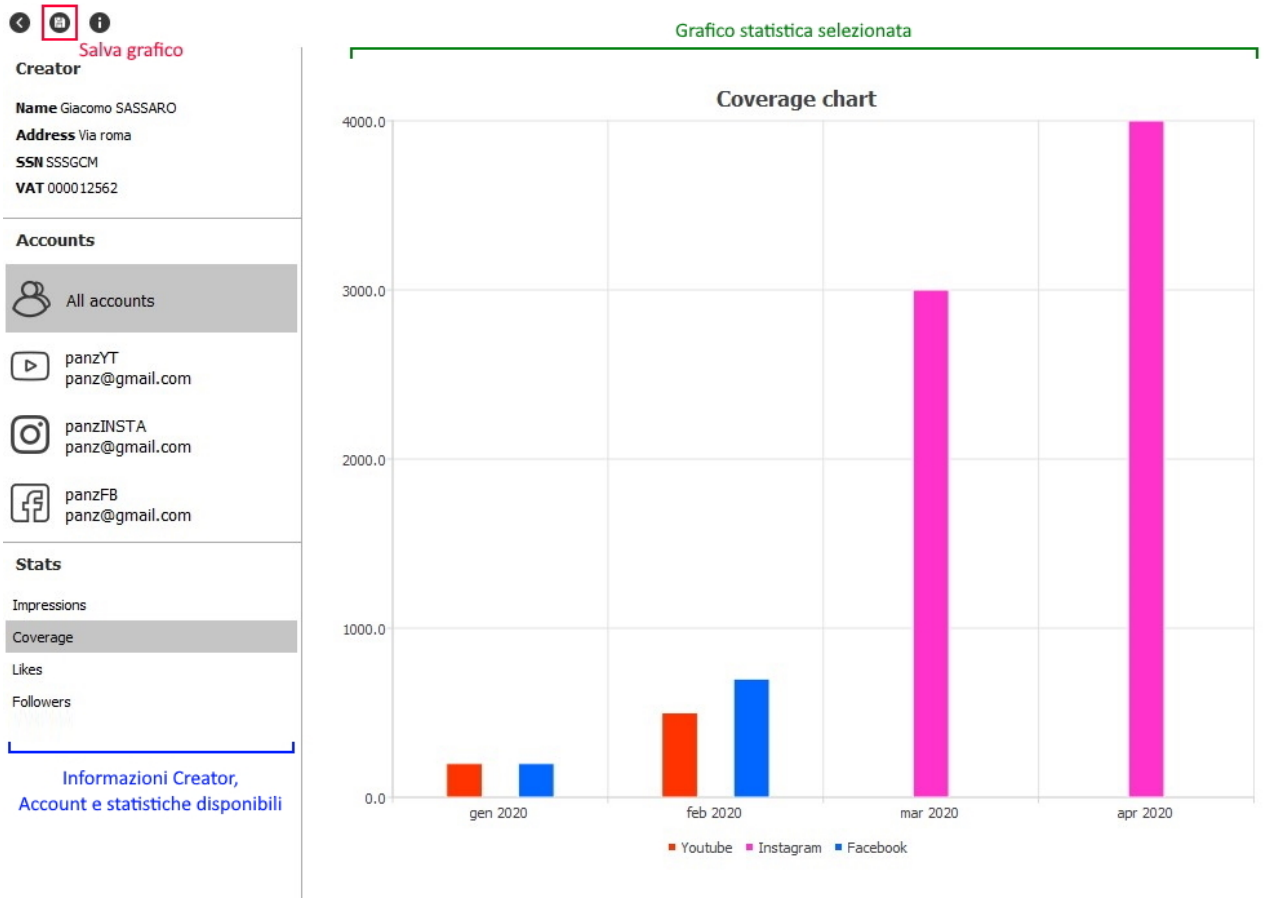
Nel progetto non vi è una grande presenza di chiamate polimorfe data l'assenza di comportamenti comuni identificabili tra le differenti statistiche.
Un esempio di caso d'uso del concetto di polimorfismo si può notare nella gerarchia di `Stats` dove sono implementati metodi virtuali utili alla creazione e distruzione degli oggetti (metodo `clone` e distruttori).
In particolare, al riempimento di `StatsList`, ***`clone`*** permette l'inserimento in lista di oggetti con tipo adatto.
Risulta utile anche l'override delle funzioni ***`import`*** ed ***`export`*** per consentire l'esecuzione corretta della procedura di importazione/salvataggio dei dati.

Interfaccia grafica

L'interfaccia, semplice ed intuitiva e' composta da due finestre distinte, entrambe derivate da `QWidget`: la ***`Landingwindow`***, che è la prima a presentarsi all'utente, permette di gestire la fonte dei dati e ricercare il cliente da esaminare tramite una barra di ricerca; e la ***`Graphswindow`***, che mostra le informazioni del cliente selezionato e ne cataloga le statistiche disponibili alla visualizzazione sui grafici.
La ricerca viene effettuata tramite una *regular expression* sul nome e sul codice fiscale dei clienti.

Istanza di LandingWindow

Dalla `Graphswindow` è possibile salvare un'istantanea del grafico attualmente visualizzato in formato immagine, mentre da entrambe le finestre è accessibile una finestra di dialogo, derivata da `QDialog`, contenente informazioni generali sull'applicazione.



Istanza di GraphsWindow

File I/Output ed Error Handling

L'avvio dell'applicazione è vincolato dall'importazione dei dati da un file in formato *JSON*, scelto per la sua efficienza e facilità d'uso.
Tramite I pulsanti nella parte superiore della `Landingwindow` la fonte delle informazioni può essere cambiata ed I dati esportati in locale (concettualmente i dati vengono richiesti online da interfacce API).
L'import/export viene quindi portato a termine attraverso le classi **`QJSonObject`** e **`QJsonDocument`** dal controller, che per mezzo di blocchi `try-catch` e `runtime_error` notifica l'utente del fallimento delle operazioni all'occorrenza di valori o formato dei dati non corretti.

Suddivisione compiti e reso conto ore

- **Gerarchia e contenitore:** *Gianpiero Tovo, Giacomo Sassaro, Ayoub Maher*
- **Modello:** *Gianpiero Tovo, Giacomo Sassaro*
- **Vista**
  - **GUI:** *Gianpiero Tovo, Giacomo Sassaro, Ayoub Maher*
  - **Grafici:** *Giacomo Sassaro*
  - **Image Export:** *Gianpiero Tovo*

- **Controller:** *Gianpiero Tovo, Giacomo Sassaro*
- **Funzione Import/Export:** *Ayoub Maher*

Monteore personale:

Come i miei compagni ho contribuito generalmente a quasi tutte le parti del progetto, occupandomi però principalmente della stesura del contenitore `CreatorList`, della funzione di export dei grafici su immagine e della progettazione e creazione dell'interfaccia grafica con lo sviluppo dei relativi slot.

- **Analisi problema e progettazione soluzione**
  Ideazione progetto: *4 ore*
  Studio sistema di versioning: *3 ore*
  Sviluppo classi gerarchia: *14 ore*

- **View**
  Progettazione e sviluppo GUI: *16 ore*
  Funzione export dei grafici: *2 ore*

- **Model e Controller:** *8 ore*

- **Debugging e Testing:** *15 ore*

***Totale**: 62 ore*

Note

Le statistiche sono identificate dalla data e l'applicazione assume quindi che per ogni mese ci sia solamente un valore (comportamento ipotetico delle API alla quale verrebbero chiesti i dati).
L'asse temporale dei grafici è infatti scandito solo da mese e anno.

La compilazione necessita della disponibilità del modulo `QtCharts`, su ambiente Ubuntu reperibile attraverso il comando `sudo apt install libqt5charts5 libqt5charts5-dev`.
I flag necessari son inclusi nel file `QANALYTICS` allegato, compilabile attraverso `qmake QANALYTICS.pro` e `make`.

Ambiente di sviluppo

- Sistema operativo: `Windows 10`, debug finale su macchina virtuale `Ubuntu`
- IDE e Framework: `Qt 5.9.5`
- Compilatore: `MinGW 5.3.0`
- Sistema di versionamento e host codebase: `Github`
- Moduli QT supplementari: `QtCharts`
- Flags QMake: `QT += core gui charts`