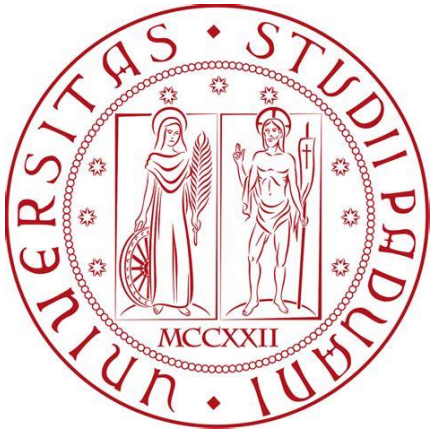


QtDrive

Progetto di Programmazione ad Oggetti

Luca Polese 1225425

Alessandro Poloni 1224444

A.A. 2020-2021

1 Scopo del Progetto

Con il recente avvento della *rivoluzione 4.0*, sempre più aziende hanno offerto, al pubblico e al privato, vari servizi tecnologici che hanno così permesso di diffondere rapidamente l'innovazione digitale.

Al giorno d'oggi, ciascuno di noi possiede almeno un account di un qualche servizio di cloud storage. Il problema maggiore che però possiamo riscontrare durante il suo uso quotidiano è proprio l'assenza di un modo per poter accedere alle informazioni in essi contenuti tutte in una volta sola.

L'idea alla base del nostro progetto, è stata perciò quella di simulare e semplificare l'accesso contemporaneo a questi servizi, utilizzando un solo software.

Quello che abbiamo messo a punto è un prodotto che intende consentire la molteplice gestione e memorizzazione di account *rigorosamente* aggiunti dall'utente tramite una maschera d'accesso.

Ogni account presenta caratteristiche comuni, ma differiscono principalmente nella quantità massima di spazio incluso nel servizio e dalle opportunità offerte singolarmente dalle aziende esistenti, che però non dipendono strettamente da quanto abbiamo sviluppato.

È tuttavia possibile visualizzare alcune di queste informazioni in una tabella riassuntiva con le specifiche divise a seconda dell'account selezionato.

Per quanto riguarda i file, invece abbiamo definito una particolare gamma di tipologie che si distinguono a seconda del contenuto che rappresentano.

In questa lista troviamo: *File di Testo, Immagini, Audio, Video e Archivi*.

Avviato il programma, una volta aperto il proprio profilo memorizzato localmente, l'utente potrà accedere liberamente ai propri documenti nella specifica sezione *File*.

In questa area sarà possibile perciò visualizzare, aggiungere, modificare o eliminare ciò che appartiene ad ogni singolo account.

Nella sezione *Ricerca* l'utente potrà selezionare ciò che più gli interessa tramite un'apposita schermata studiata per trovare un particolare contenuto fra quelli disponibili.

Tale ricerca fornisce delle impostazioni predefinite, che permettono di semplificare questa operazione, in base alle proprie esigenze.

1.1 Funzionalità del programma

Le principali funzionalità sono già state descritte in precedenza, perciò di seguito verranno elencate solamente quelle operazioni messe a disposizione dal programma non citate finora:

- Apertura, chiusura e salvataggio di un profilo utente (anche su nuovo file)
- Apertura di una nuova sessione utente (gestita con la funzione *Nuovo*)
- Visualizzazione informazioni relative al contenuto degli account (previa selezione)
- Operazioni specifiche su ogni file appartenente ad un account (vedi sopra)
- Ricerca testuale per Nome, Descrizione, Informazioni aggiuntive, Case-sensitive
- Collegamento diretto al file selezionato dopo la ricerca (doppio click riporta alla relativa finestra nella tab *File*).
- Uscita dal programma (con conseguente richiesta di salvataggio, in caso di modifiche)
- Guida completa sulle modalità di utilizzo del programma con relative immagini

1.2 Premesse rispetto all'idea del progetto

L'idea è quella di poter interfacciare più servizi eterogenei fra loro e di ottenere dunque l'accesso alle loro informazioni.

Per una questione di tempo e di mancanza di risorse, questo progetto rappresenta una simulazione di quanto descritto nell'idea del progetto nei primi paragrafi del punto 1.

Si ipotizza inoltre (per semplificazione) che la gestione dei tipi di dati sia equivalente per ogni servizio. Anche in questo caso si necessiterebbe di ulteriore tempo di studio e analisi delle infrastrutture, con conseguente sforamento del tetto orario dato.

Quanto descritto in precedenza risulterebbe oltre le conoscenze apprese nel corso di Programmazione ad Oggetti e quindi in buona parte fuori dall'obiettivo dello sviluppo di questa tipologia di progetto.

Gli studenti si riservano pertanto di riprodurre tutte le suddette azioni in modo totalmente manuale, salvando l'ipotetico contenuto degli account in file esterni di cui fa uso questo software.

Ciononostante, la struttura di base di questo progetto rimane valida per sviluppi futuri, una volta fatta esperienza dell'utilizzo di API e di trasmissione dell'informazione per mezzo della rete.

2. Progettazione del Software

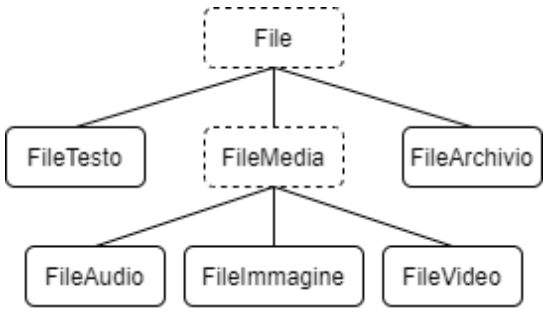
Il programma è stato sviluppato secondo il pattern Model-View-Controller di Qt: le varie classi generate pertanto sono state distribuite nelle relative cartelle.

2.1 Modello e Gerarchia

La gerarchia alla base del nostro progetto, vuole rappresentare le tipologie di dati che possono essere memorizzati in un cloud storage.

2.1.1 File

I File saranno identificati come segue:



File e FileMedia hanno il bordo tratteggiato, poiché rappresentano delle classi astratte

Si evidenziano a seguire, i tipi di File disposti nella gerarchia con le loro caratteristiche:

- File:
  Tipo generico di File, contiene tutte le informazioni che accomunano i sottotipi di file.
  Saranno identificati per mezzo del *nome*, dell'*estensione*, della *dimensione* (in MB), della *data di creazione* e di *caricamento* ed infine dalla *descrizione* (opzionale) del contenuto.
  È una classe astratta e non può essere istanziata.
- FileTesto:
  Tipo specifico di File che permette di memorizzare le informazioni relative a dei file testuali (di tutte le estensioni).
  È caratterizzata da due campi interi: *numeroCaratteri* e *numeroParole*.

- FileArchivio:
  Tipo specifico di File che permette di gestire i file archivio (di tutte le estensioni).
  Ogni FileArchivio è caratterizzato da:
    - *dimensioneOriginale*, che rappresenta la dimensione dei file contenuti nell'archivio senza compressione
    - *numeroDiFile*, che indica quanti file sono contenuti nell'archivio
    - *protetto*, un booleano che distingue se l'archivio è protetto da password
- FileMedia:
  Si tratta della seconda classe astratta della gerarchia.
  Contiene tutte le informazioni che accomunano i formati di file multimediali.
  Ha un campo unico, *tipoCompressione*, che permette di gestire la tipologia di compressione applicata al File (possibilmente anche nulla),
  I vari tipi di compressione sono riassunti mediante una enumerazione contenente *Nessuna*, *Lossless*, *Lossy*.
- FileImmagine:
  Tipo derivato da FileMedia, rappresenta i file di tipo immagine (di tutte le estensioni).
  Ogni immagine è caratterizzata da due campi interi: *larghezza* e *altezza*, che permettono di rappresentarne la risoluzione, e da *tipo* un enumeratore che permette di definire il formato della foto (nello specifico raster o vettoriale).
- FileAudio:
  Anche questo è un tipo derivato da FileMedia, che identifica le varie tipologie di file audio, distinte per mezzo dei campi interi *bitrate* e *durata*, specifici di ogni formato di file audio.
- FileVideo:
  Ultimo dei tipi derivati da FileMedia, permette di memorizzare il contenuto di un filmato tramite molteplici campi:
    - *larghezza* e *altezza*, di formato intero, che allo stesso modo dei FileImmagine, definiscono la risoluzione del filmato salvato
    - *durata* (in secondi) del filmato
    - *codec* enumeratore che indica la codifica del segnale
    - *fps* numero di frame per secondo

2.1.2 Account

La classe concreta *Account* non appartiene effettivamente alla gerarchia sopra citata, ma risulta rilevante nello sviluppo del nostro programma poiché rappresenta tutte le informazioni relative ad ogni servizio che viene aggiunto manualmente dall'utente nel proprio profilo.

Ogni account potrà perciò essere rappresentato per mezzo di caratteristiche comuni, che identificano univocamente ogni servizio di memorizzazione a cui l'utente è abbonato.

La chiave di verifica di unicità è rappresentata dalla coppia *email*, *host* (relativamente una stringa ed un enumeratore).

Il programma permette l'aggiunta dei più comuni servizi di cloud disponibili sul mercato, la maggior parte dei quali forniscono una quantità limitata di memoria gratuitamente.

Nello specifico, la lista comprende: `AmazonDrive, Box, Dropbox, GDrive, iCloud, Mediafire, Mega, Next, OneDrive` e `Qihoo360`.

Ogni account sarà provvisto inoltre di *password* e *spazioFornito* (relativamente stringa e intero richiesti in fase di aggiunta dell'account),

Tutti i file appartenenti all'account saranno salvati in un apposito contenitore di File, chiamato *listaFile* del tipo `Container<Depptr<File>>`.

Ogni account possiede un vettore chiamato *tipiDiFile* che permette una gestione semplificata ed estendibile della gerarchia di file.
Questa scelta permette di tracciare tutti i tipi concreti dei sotto oggetti di File.
Il vettore viene infatti utilizzato all'interno della funzione di *deserializzazione* per permettere la gestione automatica dei tipi di dati, adottando l'uso del polimorfismo.
Infatti è possibile effettuare delle chiamate ai metodi virtuali (presentati successivamente).

2.1.3 Container e Deepptr

Per quanto concerne il Container<T> che è stato sviluppato per questo progetto, si tratta di una lista doppiamente linkata che contiene tutte le principali funzionalità che si possono trovare in `std::deque`. Sono infatti definiti `iterator` e `const_iterator` (che a loro volta contengono le operazioni `begin`, `end`, gli operatori di equivalenza, quelli di incremento e quelli di dereferenziazione).
Troviamo in aggiunta le funzioni di `push_front, push_back, pop_front, pop_back, erase, clear` e gli operatori di subscripting.

Nel nostro programma il template Container è solitamente istanziato con dei tipi Deepptr<T>, anch'essi implementati tramite template. Deepptr è sviluppato per essere compatibile con tutti i tipi delle classi presenti nel programma grazie anche alla gestione profonda della memoria.

2.2 Graphical User Interface

L'interfaccia grafica del software è gestita per mezzo di molteplici finestre, che compaiono a seconda dell'evento che viene generato dall'utente.
Una volta avviato il programma infatti, verrà mostrata all'utente una finestra di avvio, con l'immagine in movimento del logo del progetto.
Prima di chiudersi (in modo automatico), la finestra effettua la chiamata `execute` che invoca la funzione `show` della finestra principale.

La finestra principale, chiamata anche `MainWindow`, deriva direttamente da `QWidget` e gestisce per mezzo di vari `connect` tutti gli eventi generati dalle azioni dell'utente.
Tutti i membri relativi a questa classe non sono accessibili al di fuori della classe stessa, e sono per lo più gestiti tramite puntatori.

Tutti gli elementi che si possono visualizzare all'interno di questa schermata, verranno aggiornati solamente in caso di modifica/eliminazione/aggiunta di nuovi oggetti, oppure in caso di apertura di file o di nuove sessioni.

3. Polimorfismo

L'uso del polimorfismo nel progetto è generalmente distribuito all'interno dei sottotipi di File presenti nella gerarchia di cui sopra.

In molti punti del programma, in particolar modo nelle Tab *Account* e *File*, sono effettuate delle chiamate polimorfe che permettono di recuperare alcune informazioni relative ai File o agli Account selezionati.

Di seguito sono elencati i metodi polimorfi che è possibile individuare all'interno del progetto con le relative spiegazioni:

3.1 File

- `Distruttore` e `clone()`
  Sia il distruttore che il metodo di clonazione, vengono riscritti in tutte le classi derivate di File, e consentono di gestire correttamente la memoria evitando leak.

- `getInformazioniFile() const`
  Questo metodo restituisce una stringa con le informazioni specifiche di ogni sottotipo di File. Ciò ha reso necessaria una definizione virtuale del metodo.
- `serializza(QXmlStreamWriter) const`
  Questo metodo, ha l'obiettivo di salvare il contenuto di ogni componente di un sottotipo di file a seconda della struttura che lo costituisce.
  Il polimorfismo risulta necessario, vista l'esigenza di costruire un file contenente tutti gli elementi specifici del sottotipo
- `deserializza(QXmlStreamReader)`
  Questo metodo, ha l'obiettivo di leggere il contenuto di ogni file XMl e di costruire a partire da esso ogni componente dei vari sottotipi di file (a seconda della struttura che lo costituisce).
- `ricercaAvanzata(QString, Qt::CaseSensitivity)`
  Questo metodo viene utilizzato per effettuare la ricerca di una stringa definita dall'utente nella barra di ricerca. Questa nello specifico viene eseguita sulle informazioni aggiuntive di ogni sottotipo di file.
  Per effettuare la ricerca, questo metodo invocherà la funzione polimorfa `getInformazioniFile() const`.
- `getTipoFile() const`
  Funzione polimorfa utilizzata nei metodi di lettura e scrittura di file XML. Restituisce la stringa con il nome del tipo di sotto oggetto che è stato chiamato, limitando così la necessità di un type checking dinamico ripetuto.

3.2 Account

- Template di funzione `contaFile() const`
  Questa funzione della classe Account è utilizzata per il calcolo della quantità di file di un determinato sottotipo presenti all'interno di un Account.
  È necessario scorrere il container dei file dell'oggetto Account ed effettuare un controllo sul tipo dinamico tramite dynamic_cast.

Tutte le classi che abbiamo precedentemente descritto forniscono una implementazione dei metodi virtuali già descritti. L'unica eccezione, seppur ovvia, è la classe FileMedia, che essendo astratta, per definizione non li implementa.

4. Funzionalità di Input e Output

Come già anticipato in precedenza, tutte le informazioni relative al profilo di ogni utente, vengono serializzate e/o deserializzate tramite il metalinguaggio XML.

I dati relativi ad account e files vengono gestiti a livello di oggetti tramite la classe `Xmlify`.

Qualora la serializzazione o la deserializzazione non andassero a buon fine, l'utente verrebbe notificato con una `QMessageBox` contenente il motivo dell'errore.

Il file XML che viene creato è contraddistinto da vari punti:
- `listaAccount` è l'elemento radice, che determina l'intestazione del file.
- `account`, identifica l'inizio delle informazioni di un account.
- `listaFile`, identifica l'inizio dei file contenuti in un account
- `file`, contiene le informazioni di un singolo file inserito dall'utente.

Con il codice del programma, si può trovare anche un file XML di esempio funzionante.

5. Manuale Utente

Con questo programma, viene fornito un manuale utente completo, che può essere visionato all'avvio dell'eseguibile.

Si può raggiungere la guida in due modi:

- Cliccando sulla voce del menu *Aiuto* e successivamente selezionando l'opzione *Guida*
- Per mezzo dello shortcut *Ctrl+H*

A quel punto comparirà una finestra contenente un indice sotto forma di elenco numerato cliccabile, che permetterà agilmente all'utente di visualizzare direttamente la voce ricercata.

6. Istruzioni di Compilazione ed Esecuzione

Per compilare correttamente il programma è necessario utilizzare il file progetto.pro fornito insieme a tutto il codice sorgente del programma.

I comandi da utilizzare sono:

```
qmake progetto.pro
```

```
make progetto
```

A questo punto, il programma può essere eseguito per mezzo del comando:

```
./progetto
```

7. Tempistiche

Analisi preliminare del problema: 4 ore

Progettazione modello: 3 ore

Progettazione GUI: 4 ore

Apprendimento libreria Qt: 5 ore

Codifica modello: 20 ore

Codifica GUI: 10 ore

Debugging: 6 ore

Testing: 5 ore

In totale il progetto ha richiesto circa 57 ore di lavoro individuale, 7 ore in più rispetto a quelle previste. Motivo di tale sforamento è causato principalmente dallo sviluppo della parte del modello dell'applicazione, che ha richiesto vari aggiustamenti dovuti alla codifica delle parti grafiche e della gerarchia.

Va inoltre sottolineato che una parte del tempo extra è stato utilizzato per il debug ed il collaudo.

Infine, per quanto riguarda l'apprendimento della libreria, non si è tenuto conto delle ore di tutorato frequentate e della consultazione della documentazione di Qt in corso d'opera.

8. Suddivisione del Lavoro Progettuale

Classi a cui ha lavorato individualmente lo studente Luca Polese:

- Container
- DeepPtr
- Xmlify
- Intro

Classi a cui ha lavorato individualmente lo studente Alessandro Poloni:

- File
- AccountWidget
- GuidaWidget
- InfoFileWidget
- ModificaAccountWidget
- NuovoFileWidget

Segue ora una lista delle classi a cui entrambi i componenti del gruppo hanno lavorato con una spiegazione di quanto implementato dal sottoscritto Luca Polese

- Sottotipi di File: le funzioni `serializza` e `deserializza`
- Account: per quanto riguarda la parte di gestione dei dati di ogni Account, le funzioni `serializza(QXmlStreamWriter) const`, `deserializza(QXmlStreamReader)`, `riempiTipiDiFile()` e template di funzione `contaFile() const`
- MainWindow: gestione degli eventi di apertura, chiusura file (più genericamente la parte logica del menù), l'evento di chiusura dell'applicazione ed in parte i `connect`
- Controller, per la parte di gestione delle chiamate di modifica degli account e tutto ciò che concerne l'XML.

9. Ambiente di Sviluppo

Sistema Operativo	Microsoft Windows 10 Home
Compilatore	MinGW 8.1.0 64-bit
Libreria Qt	Qt 5.15.2

Il progetto è stato inoltre testato con la macchina virtuale fornita dal Docente:

Sistema Operativo	Ubuntu 18.04.3 LTS
Compilatore	GCC 64-bit
Libreria Qt	Qt 5.9.5