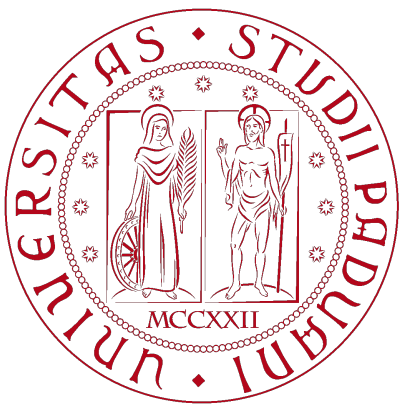
Filament3dPrint

PROGETTO DI PROGRAMMAZIONE AD OGGETTI

Anno Accademico 2021/2022

Realizzato da
Andrei Cristian Bobirica 1224449



UNIVERSITÀ DEGLI STUDI DI PADOVA

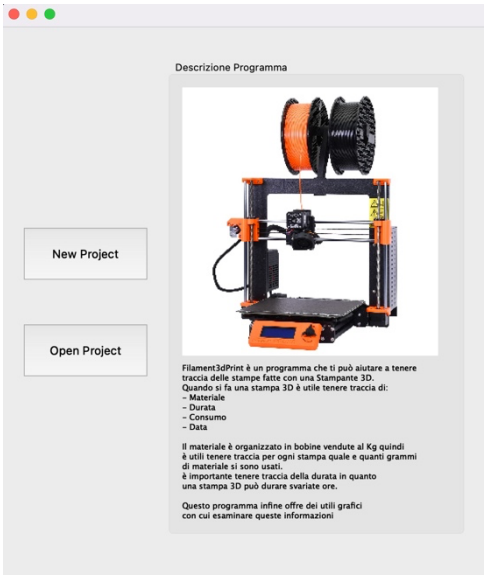
1. **Introduzione**

Negli ultimi anni la stampa 3D è diventata sempre più diffusa e chiunque può con facilità progettare, disegnare e stampare degli oggetti in 3D.
Per la stampa si utilizzano come materiali dei polimeri sotto forma di filamento arrotolato ad una bobina;
esistono diversi materiali e questi possono avere diverse caratteristiche e specifiche sia per il processo di stampa che per la qualità della stampa finale.
Il processo di stampa 3D FDM può variare in base alle configurazioni che si danno alla stampante quindi è importante tenere traccia delle stampe effettuate.
È importante registrare il materiale utilizzato per tenere traccia delle bobine usate.
Serve la durata della stampa in quanto il processo può durare svariate ore.
Successivamente è utile registrare il consumo del materiale in grammi, questo dato è utile per pianificare le future stampe in base al consumo.
L'ultimo parametro di fondamentale importanza è la data della stampa; un materiale può avere una diversa igroscopicità e una più facile stampa durante i mesi invernali piuttosto che i mesi estivi umidi.
Tenere traccia delle date serve anche a vedere i periodi più produttivi della stampante.

2. **Descrizione programma e manuale d'uso GUI**

Fialment3dPrint è un programma utile per chi stampa in 3D, fornisce un applicativo con cui registrare e visualizzare i dati delle stampe 3D, inoltre rielabora questi dati e mostra dei utili grafici.

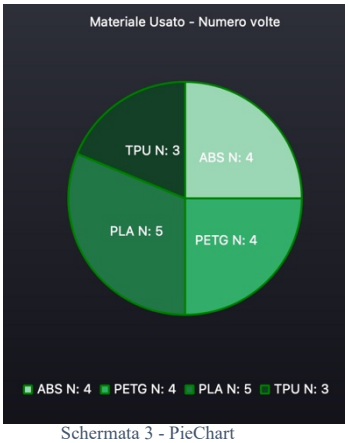
Una volta aperto il programma la prima schermata Home offre la possibilità di iniziare un nuovo progetto senza dati oppure di aprirne uno già esistente prendendo i dati da un precedente progetto salvato su file.
È stato fornito insieme al progetto anche il file **ProgettoProva.json**
Esso è utile per fare delle iniziali prove e vedere il funzionamento del programma.



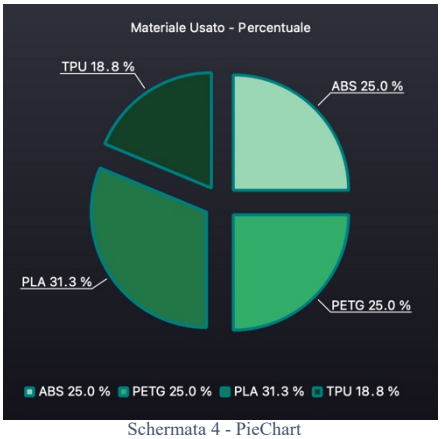
Una volta aperto un progetto viene mostrata la schermata Admin in cui sono presenti due strutture tabellari su cui immettere i dati.
La prima a sinistra rappresenta i Record ovvero i dati delle stampe effettuate.
La seconda a destra rappresenta la lista dei Materiali disponibili.
Nella tabella delle stampe si può selezionare il materiale con cui si è effettuato la stampa, la durata di essa, il consumo in grammi e la data in cui è stata eseguita.
Una volta immessi i dati, con i pulsanti in alto si può salvare il progetto oppure salvarlo con un altro nome.
Sono presenti altri due pulsanti con cui aprire un nuovo progetto o tornare alla schermata Home.

Schermata 1 - Admin

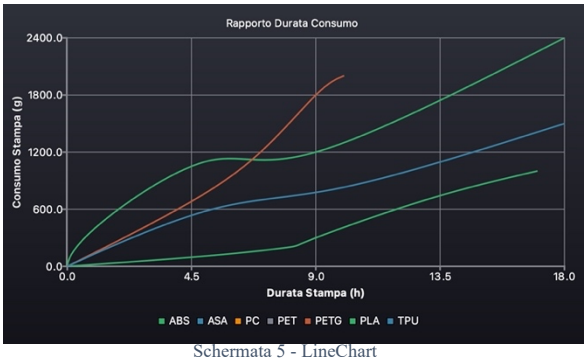
Il programma infine elabora questi dati ed offre all'utente degli utili grafici con cui si possono prendere delle considerazioni sulle stampe effettuate.



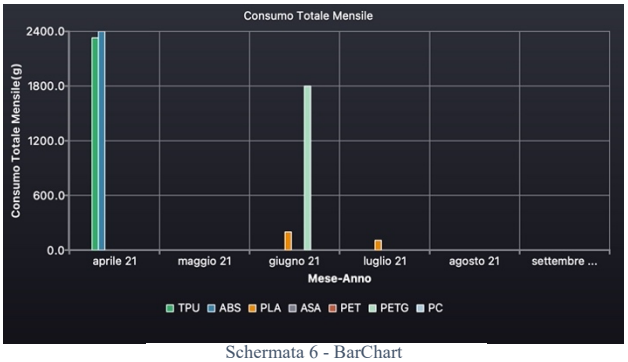
Schermata 3 - PieChart



Schermata 4 - PieChart



Schermata 5 - LineChart



Schermata 6 - BarChart

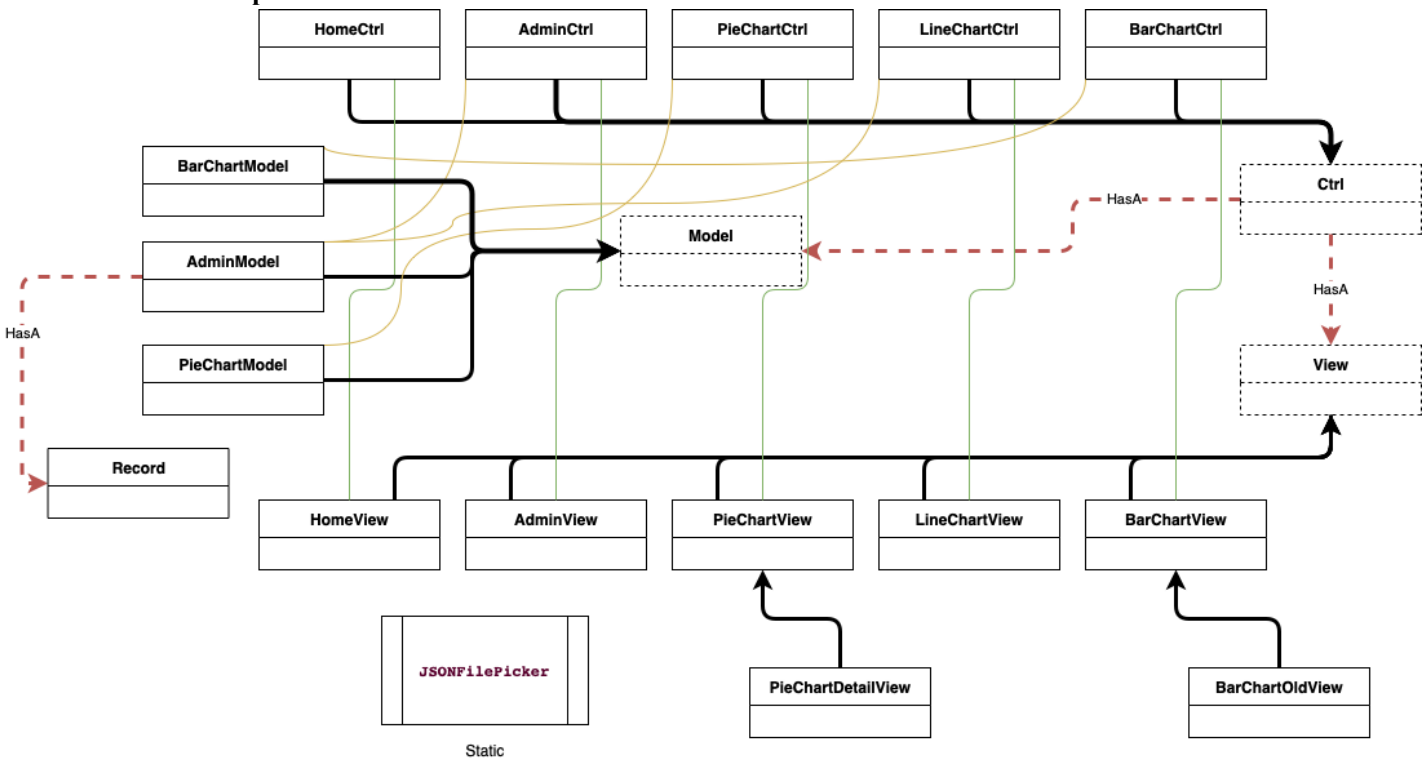
3. Funzionalità

- Inserimento e controllo dei dati immessi mantenendo una integrità referenziale tra Record e Materiali.
  La tabella dei Record, ovvero delle stampe, e la tabella della lista di materiali sono in integrità referenziale tra di loro e l'applicazione gestisce aggiunta, modifica ed eliminazione di un nuovo materiale gestendo anche i rispettivi Record collegati.
  Viene aggiornato in tempo reale il ComboBox (SelectBox) nella lista dei Record con le effettive modifiche effetuate nella lista dei materiali.
  Viene inoltre gestita la lista dei materiali in maniera tale che essi abbiano dei nomi univoci.

- Salvataggio del progetto su file, i dati vengono salvati in formato JSON.
  Il nome del progetto è il nome del file ed esso ha una estensione JSON.
  Viene gestito il caso in cui si da un file non valido al programma e viene effettuato un controllo anche sul JSON che viene fornito.
  In particolare viene controllato che il JSON abbia come struttura almeno
  `{"materiali": [],"records": []}`
  Il file può essere usato per riaprire il progetto in un secondo momento ed effettuare ulteriori modifiche oppure per essere usato con altri programmi.

- Elaborazione dei dati e visualizzazione di essi tramite dei grafici.
  Viene offerto un **PieChart** in 2 versioni con cui si può vedere la tipologia di materiale utilizzata più volte sia in forma numerica che in forma percentuale.
  Successivamente è disponibile un **LineChart** che rappresenta sulle ascisse la durata e sulle ordinate il consumo, il grafico che viene disegnato è utile per gestire il rapporto durata/consumo tra i vari materiali e capire quale materiale ha un consumo più veloce a parità di durata della stampa.
  l'ultimo grafico che viene fornito è un **BarChart** in due versioni con cui è possibile vedere per ogni mese il consumo totale di ogni materiale.

4. Gerarchia di tipi



Si è deciso di utilizzare il modello MVC per dividere parte Logica dalla parte Grafica.
La classe base astratta "**Ctrl : public QObject**" ha il compito di essere Controller di una vista (View / Schermata).
Un Controller ha una View e un Model ed in linea generale riceve segnali dalla View, li elabora, e ne ritorna il responso alla View oppure fa delle modifiche al Model di dati. Un Controller ha una lista di Slot collegati a segnali della View.
Un Controller ha anche la possibilità di creare un altro Controller e ciò corrisponde ad aprire una nuova schermata.
La classe base astratta "**View: public QWidget"** ha il compito di essere la parte visiva di una specifica schermata, contenendone dei Widget e fornendo segnali e metodi al Controller, il quale in base alle sue operazioni ne modifica lo stato degli Widget e ne aggiorna il Model.
Un "**Model"** è una collezione e rappresentazione di dati per una specifica schermata o funzionalità.
Tutte le schermate della applicazione hanno un specifico **Ctrl** (HomeCtrl, AdminCtrl, PieChartCtrl, etc…) che sono classi derivate ed introducono specifiche funzioni per ogni schermata in questione.
Per esempio "`void AdminCtrl::onMaterialTableAdded(const QString &m);`" offre uno SLOT per la gestione di un materiale aggiunto alla lista di materiali nella schermata Admin. Invece "`void HomeCtrl::onOpenProject() const;`" offre uno SLOT per l'apertura di un progetto esistente nella schermata Home.
Lo stesso ragionamento viene effettuato anche per le **View** (HomeView, AdminView, PieChartView, etc…) le quali hanno una specifica implementazione per ogni schermata con ciascuna i propri Widget, segnali e i propri metodi.
Infine ogni schermata può avere un suo **Model** (AdminModel, PieChartModel, etc)
La gerarchia di classi si può capire con molta facilità guardando i costruttori dei Controller in quanto essi descrivono con precisione di cosa ha bisogno ogni schermata.

```
HomeCtrl(HomeView*, Ctrl*);
AdminCtrl(AdminView*, AdminModel*, Ctrl*);
PieChartCtrl(PieChartView*, PieChartModel*, Ctrl*);
LineChartCtrl(LineChartView*, AdminModel*, Ctrl*);
BarChartCtrl(BarChartView*, BarChartModel*, Ctrl*);
```

Si ricorda infine che oltre al meccanismo di distruzione di QT attraverso i parent, si sono utilizzati anche i distruttori virtuali profondi ridefiniti dove servisse e aggiungendo al loro normale comportamento anche la rimozione del parent prima della distruzione in maniera da non creare conflitti con il distruttore di QT (doppio delete su un ptr).
Si è infine deciso di utilizzare la classe statica **JSONFilepicker** che offrisse solo metodi statici e quindi che fosse usata come un coltellino svizzero all'occorrenza per la lettura e la scrittura di file.
Si è optato per questa soluzione perché non serviva istanziare questa classe ad ogni occorrenza ed in ogni caso perché offriva solamente una sovrastruttura al meccanismo di lettura di file di QT e non implementava nulla di complicato.

5. **Uso di Polimorfismo**

- **`Distruttori Polimorfi`**

Un esempio è quando si chiude una schermata, in particolare quando si cancella il Controller, è stato utilizzato il meccanismo di distruttori virtuali :

```
Ctrl::~Ctrl() {
    setParent(nullptr);
    for(auto child : children())
        delete child;
    delete view; delete model;
}
```

In questo caso viene invocato il distruttore sulla **view** e sul **model** e non importa quale è il vero tipo dinamico di questi due puntatori, vengono comunque eliminati correttamente dallo Heap.

- **`closeEvent`**

Quando si preme sul pulsante **X** di una finestra **View** essa essendo un **QWidget** invoca automaticamente il metodo

```
virtual void QWidget::closeEvent(QCloseEvent* event);
```

Esso è in overriding in :

```
void AdminView::closeEvent(QCloseEvent* event) override;
void HomeView::closeEvent(QCloseEvent* event) override;
void View::closeEvent(QCloseEvent *event) override;
```

In base al tipo dinamico di una vista viene invocato la giusta versione del metodo che effettua diverse operazioni come chiudere conferma con un PopUp a schermo, oppure effettuare delle operazioni di pulizia prima.
Questo metodo viene definito dalla libreria QT ma viene usato in overriding nella gerarchia del progetto per tanto è un ottimo esempio della utilità del polimorfismo.

- **`setViewTitle`**

È stato definito :

```
virtual void View::setViewTitle(const QString &title);
```

Questo metodo serve a impostare alla schermata un Titolo da vedere sulla finistra del Desktop.
La sua prima implementazione all'interno di **View** si limita solamente a trascrivere la stringa title sulla finestra.
Esso è in overriding in :

```
void AdminView::setViewTitle (const QString &title) override;
```

Questo metodo non si limita a trascrivere la stringa title sulla finestra ma aggiunge anche l'etichetta "Progetto :" davanti al titolo. Esso è infati ridefinito per la **AdminView** la quale ha necessità di specificare che si tratta di un Progetto.
L'invocazione polimorfa avviene all'interno del **AdminCtrl** :

```
view->setViewTitle(last);
```

Se in futuro ci fosse una evolvibilità della **AdminView** questo metodo in overriding potrebbe implementare altre funzionalità quali modificare anche l'icona in base al title , etc.

- **`onViewClosed`**

Quando una **View** viene chiusa essa emette il segnale : "`void viewClosed() const;`"
Il Ctrl connette a questo segnale uno SLOT virtuale puro.

```
connect(view,SIGNAL(viewClosed()),this,SLOT(onViewClosed()));      // this == const Ctrl*
```

Lo SLOT è:

```
virtual void Ctrl::onViewClosed() const = 0;
```

Questo SLOT è in overriding in tutte le Classi derivate di Ctrl, ecco alcuni esempi:

```
void HomeCtrl::onViewClosed() const override;
void AdminCtrl::onViewClosed() const override;
void HomeCtrl::onViewClosed() const override;
```

Una volta chiusa una View questo SLOT decide cosa fare dal **Ctrl** ed in base alla sua implementazione e al tipo dinamico del **Ctrl** possono essere eseguite diverse operazioni in base alla specifica funzionalità della schermata.

- **`PieChartView`**

Per la View **PieChartView** è stato ipotizzato un particolare caso di estensibilità in cui la classe veniva reimplementata con una classe derivata :

```
class PieChartDetailView : public PieChartView
```

È stato implementato il metodo virtuale :

```
virtual void PieChartView::applyGraphics();
```

Esso è in overriding in:

```
void PieChartDetailView::applyGraphics() override;
```

6. Descrizione formati di input/output

Come anticipato precedentemente il programma offre come funzionalità la possibilitò di salvare i dati del progetto sotto forma di file.
I dati immessi da utente tramite tastiera vengono interpretati come Modelli, essi sucessivamente, se richiesto, vengono salvati su file sotto forma di JSON.
Il file JSON ha una struttura ben specifica e l'unico test semantico che viene effettuato è che esso assomigli a
`{"materiali": [],"records": []}`
Viene rigettato ogni altro tentativo di imettere dei File con altre estensioni, che non siano JSON oppure che non abbiano una struttura simile a quella su mostrata.
Attenzione al fatto che tuttavia non sono stati implementati ulteriori meccanismi di protezione e si potrebbe andare incontro ad U.B. se si passano file JSON alterati esternamente con una struttura,specialmente per i records, diversa.
Il file salvato può essere successivamente riutilizzato per riaprire il progetto in altre esecuzioni del programma.

Si fa notare che nel progetto è stato incluso il file **ProgettoProva.json** utile per fare prove, test e vedere il funzionamento del programma. Tutti i dati al suo interno sono a scopo dimostrattivo e non rappresentano un vero caso d'uso nel mondo della stampa 3D.

7. Istruzione compilazione ed esecuzione

Il Progetto è fornito insieme al file **progetto.pro**
Deve essere uttilizzato questo file .pro e non se ne devono generare di altri.
Al progetto servono i sudetti pachetti QT:

```
qt5-default

libqt5charts5-dev
```

Le istruzioni per la compilazione sono le seguenti :

```
qmake → make
```

Il programma può essere eseguito con doppio click oppure con :

```
./Filament3dPrint
```

8. Ore di lavoro richieste

Analisi dei requisiti	1 ore
Progettazione GUI e funzionalità	3 ore
Documentazione Qt	6 ore
Studio modello MVC	1 ora
Creazione View GUI - View Home/Admin - Creazione Grafico PieChart - Creazione Grafico LineChart - Creazione Grafico BarChart	 8 ore 3 ore 4 ore 4 ore
Creazione Controller - Modello	14 ore
Scrittura su file e salvataggio progetto	6 ore
Debuging testing	11 ore
TOTALE	**61 ore**

Sfortunatamente sono state superate le 50 ore di lavoro per i seguenti motivi :
La schermata Admin ha richiesto pià tempo del previsto, in particolare per rispettare la funzionalità di integrità referenziale tra le tabelle e garatire il controllo sui dati immessi si sono dovuti fare tutti i controlli necessari e ciò ha richiesto molto tempo.
In aggiunta ci sono stati dei problemi iniziali in cui il programma andava incontro a continui Crash e per trovare la causa ci sono volute molte ore.

9. Ambiente di Sviluppo

L'IDE di sviluppo usato è stato Qt Creator.
Lo sviluppo principale è stato effettuato sulla piattaforma :

Sistema Operativo	macOS Monteray 12.1
Compilatore	Apple clang version 13.0.0
Libreria Qt	Qt 5.9.5

Il programma è stato comunque regolarmente testato sulla Virtual Machine:

Sistema Operativo	Ubuntu 18.04.3 LTS
Compilatore	GNU g++ 7.3
Libreria Qt	Qt 5.9.5

Inoltre è stato utilizzato lo strumento GIT nonostante il progetto sia stato sviluppato in singolo.
Questo strumento ha permesso di tenere traccia dei cambiamenti e ha permesso di utilizzare uno strumento molto importante cioè le ACTIONS.
Il progetto ad ogni commit è stato infatti testato su una macchina virtuale Ubuntu messa a disposizione da Github.
Di seguito i test effettuati ad ogni commit insieme alla repository.
https://github.com/andreibobirica/Filament3dPrint/actions
Le operazioni che la action faceva ad ogni test sono state:

```
- name: Install Qt
  run: |
    sudo apt-get install qt5-default
    sudo apt install libqt5charts5-dev
- name: Test
  run: |
    qmake
    make
```

Nonostante gli innumerevoli test bisogna ricordare che questi coprivano solamente la compilazione del progetto.
Per trovare i problemi a Runtime sono stati eseguiti dei test manuali.