Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Imagik – Image Viewer

Studente/i
Massimo De Santi
Attilio Baldini

Relatore
Amos Brocco

Correlatore
Michele Banfi

Committente
Supsi DTI

Corso di laurea
Ingegneria informatica

Modulo
Ingegneria del Software 1

Anno
2019

09.06.2019

Data

1 Abstract

L'obbiettivo di questo progetto è quello di implementare un software per PC in grado di visualizzare le immagini presenti in una directory.

Il software deve inoltre includere delle funzioni base di image editing, applicabili su una o più immagini selezionate dall'utente.

2 Il Progetto

2.1 Scheda di progetto

Imagik – Image Viewer è un software per elaborare delle immagini in formato JPG e PNG. Il programma permette all'utente di visualizzare le immagini di una directory (in forma di miniatura/anteprima). L'utente può visualizzare i metadati principali (dati dell'immagine e metadati EXIF) selezionando un'immagine.

L'utente può filtrare la visualizzazione attraverso dei pattern applicati al nome del file. L'utente può visualizzare un'immagine con livelli di zoom a sua scelta. Il programma deve inoltre permettere all'utente di applicare alle immagini una o più modifiche (per es. scala, ruota,…) e di salvare il risultato in un nuovo file.

Requisiti

I requisiti di base del programma sono elencati di seguito. Ulteriori requisiti dovranno essere discussi con il committente [1] e potranno essere introdotti durante lo sviluppo.

- Il programma deve permettere la visualizzazione (anteprima) delle immagini di una cartella.
- L'utente può filtrare la lista delle immagini attraverso un pattern (globbing) sul nome del file.
- L'utente deve poter cambiare cartella in qualsiasi momento.
- Il programma visualizza le informazioni dell'immagine selezionata (dati dell'immagine quali tipo, risoluzione, dimensione, dimensione del file, metadati EXIF, …).
- Il programma permette di selezionare una o più immagini per effettuare modifiche.
- Il programma permette diverse operazioni di modifica (scala, ruota, converti in bianco e nero, …).
- Il programma memorizza le operazioni di modifica effettuate su un'immagine in un file di testo.
- Il programma mostra un'anteprima delle modifiche selezionate.
- L'utente permette di salvare l'immagine modificata in un nuovo file.
- Il programma dovrà supportare la localizzazione in almeno due lingue [2].

Ulteriori caratteristiche/funzionalità che non vanno in conflitto con quelle richieste possono essere liberamente implementate (per es. modalità slideshow per visualizzare le immagini selezionate in sequenza).

[1] Ruolo che verrà assunto dai docenti
[2] Non è necessaria una traduzione completa in una seconda lingua, ma il supporto a livello di implementazione deve essere presente.

Tecnologie

- Java e JavaFX
- Librerie di manipolazione di immagini per Java (disponibili su Maven)
- Librerie per l'estrazione dei metadati EXIF (disponibili su Maven)

Altre informazioni importanti:

Lo sviluppo dovrà seguire i principi e le metodologie presentate durante il corso. Il codice sorgente dovrà essere disponibile su un repository git (github.com): tutti i membri del gruppo dovranno dimostrare una partecipazione equa allo sviluppo.

Il progetto sarà consegnato in forma di programma installabile/eseguibile: non sono accettate consegne di progetti non direttamente eseguibili o che richiedono un IDE/Compilatore per essere eseguiti. Le piattaforme sulle quali deve funzionare il programma sono: Linux e Windows (opzionalmente: OSX).

Durante la presentazione è richiesta una dimostrazione del programma (a partire dall'installazione).

2.2 Personalizzazione

Il progetto è stato sviluppato da tre gruppi differenti ognuno dei quali doveva scegliere un nome e un'icona per caratterizzare l'applicativo. Inoltre, ad ogni gruppo è stata assegnata una differente linea guida sul design della User Interface.

Nome e Icona

Il nostro gruppo ha scelto come nome "Imagik - Image Viewer" (o "Imagik - Visualizzatore immaggini" se la lingua di sistema è l'italiano). L'icona invece rappresenta due foto sovrapposte che brillano (piccolo richiamo alla magia).

UI

Al nostro team è stato assegnato il compito di seguire le direttive di GNOME 3 ("Human Interface Guidelines"). Nei capitoli successivi verrà spiegato come e dove abbiamo applicato tali direttive.

Come principio base per il design della UI, abbiamo deciso di usare un aspetto semplice e minimalista, riservando la maggior parte dello spazio disponibile alla visualizzazione dell'anteprima.

Usability

Un secondo aspetto su cui ci siamo concentrati è la semplicità di utilizzo. L'obbiettivo che volevamo raggiungere è che l'utente possa usare l'applicativo in maniera intuitiva senza bisogno di leggere un manuale utente.

3 Requisiti

Prima di iniziare a sviluppare abbiamo analizzato i requisiti base proposti dal committente. Questa è una fase cruciale che ogni buon progetto software deve affrontare. È fondamentale chiarire, senza sorta di ambiguità, cosa deve fare il sistema e come lo deve fare.

La tabella successiva riassume i requisiti base dividendoli in requisiti Funzionali (cosa deve fare il sistema) e Non Funzionali (come il sistema deve fare una determinata funzione).

Funzionali	**Non funzionali**
Visualizzare anteprima delle immagini di una directory	i18n (internazionalizzazione)
Selezionare una directory	Implentazione in Java e JavaFX

Visualizzare metadata	Progetto Maven
Cambiare la lingua dell'interfaccia	Codice su repository Git
Applicare dei filtri grafici	Creazione di un package eseguibile
Salvare l'immagine modificata	Aderente a HIG GNOME 3
Selezionare più immagini	

Generalmente esistono altre fasi precedenti all'analisi dei requisiti, che in questo specifico progetto non abbiamo dovuto affrontare:

- Analisi del dominio
- Definizione del problema
- Raccolta dei requisiti

3.1 Requisiti Funzionali

Per meglio rappresentare i requisiti funzionali abbiamo usato un diagramma UML di tipo Use Case. L'uso di un diagramma facilita la comprensione e valutazione della soluzione proposta nascondendo i dettagli irrilevanti.

In fase iniziale abbiamo prospettato una serie di use case (Fig. 1), di questi la maggior parte sono stati dati dal mandato, altri li abbiamo ipotizzati noi.

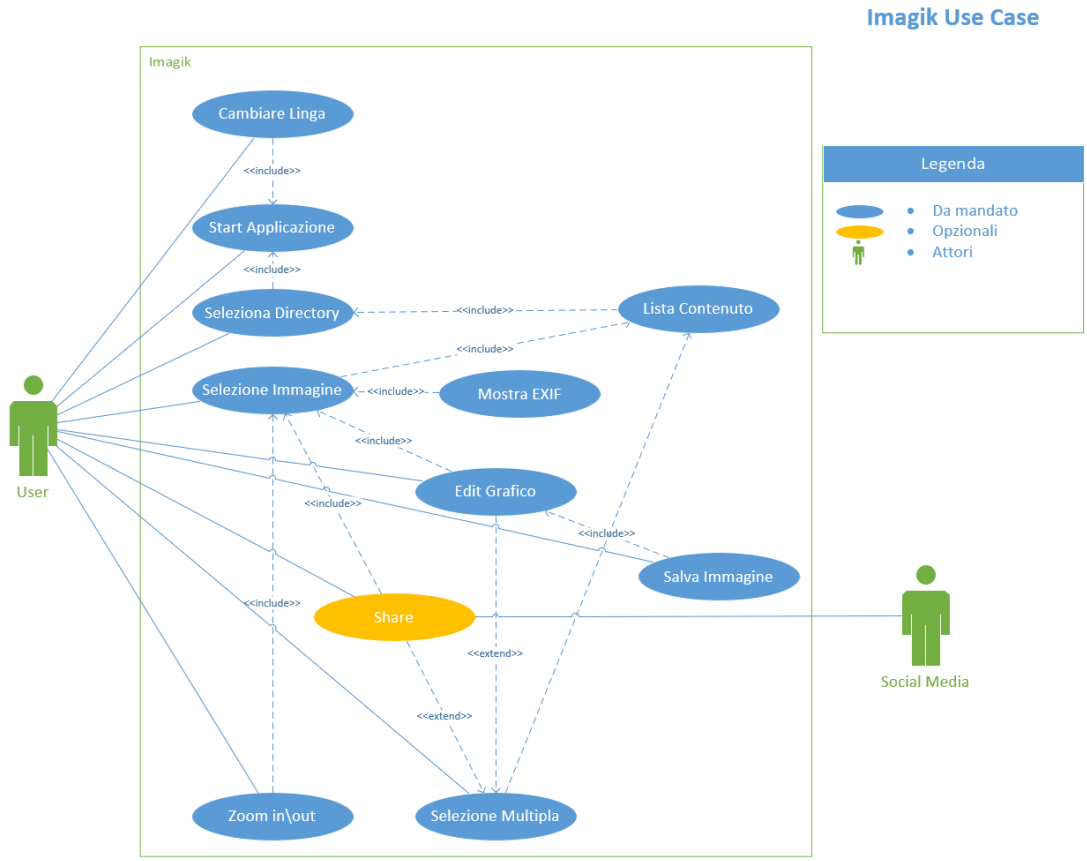

Fig. 1 - Schema UML use case

Un esempio di use case da noi aggiunto è la funzionalità di share verso i social.

Per la creazione di questo schema abbiamo seguito questi step:

1) Identificato gli attori che interagiscono col sistema “Imagik”: Utente e Social.
2) Scelto le azioni che l’attore può fare e che hanno un risultato tangibile.
3) Identificato percorsi alternativi/opzionali (extensions) come la selezione multipla.

In questo schema si può notare l’uso delle parole chiavi *<<include>>* ed *<<extend>>* come collegamenti tra Use Case. Questi collegamenti permettono di semplificare il diagramma nel caso in cui i requisiti di uno Use Case siano già coperti da un altro Use Case.

La relazione *<<include>>* viene usata quando il caso d’uso incluso è sempre e interamente usato. Ad esempio, per poter applicare i filtri grafici è obbligatorio selezionare almeno un’immagine.

La relazione *<<extend>>* viene usata quando il caso d’uso incluso è opzionale. Ad esempio, per poter applicare i filtri non è strettamente necessario selezionare più immagini.

Il diagramma proposto poteva essere scomposto in diagrammi più piccoli accompagnati da una descrizione tabellare degli Use Case. Queste descrizioni aggiungono le informazioni necessarie per modellizzare i requisiti e per poterli implementare:

- Breve descrizione
- Stato iniziale del sistema (precondizioni)
- Stato finale del sistema (Esito positivo ed Esito negativo)
- Interazioni con attori
- Sequenza di passi
- Eventuali punti di estensione

A causa dei tempi stretti ci siamo limitati su un unico scenario che prova ad includere più casi d’uso. Ci siamo concentrati sulle funzionalità di mandato, ma la possibilità di fare un post verso i social di un’immagine editata è allettante vista la diffusione dei social media al giorno d’oggi. Abbiamo deciso di aggiungere questa funzionalità negli sviluppi futuri.

3.2 Requisiti non Funzionali

3.2.1 Strumenti

Il team di sviluppo si è avvalso dei seguenti strumenti e librerie per sviluppare l’applicativo richiesto:

Strumento	**Descrizione**
IntelliJ IDEA	IDE di sviluppo
Trello	Kanban per gestione e avanzamento dei task di sviluppo
GitHub	Source repository

Libreria	**Descrizione**
ImageJ	Elaborazione grafica
Metadata Extractor	Estrazione metadata
Guava Core Libraries	Google Event Bus
OpenJFX	Controlli grafici Java FX

Apache Commons IO	Utility per I/O
Ikonli	Icone

3.2.2 Gestione del progetto

Inizialmente abbiamo diviso il lavoro in Task e creato le corrispondenti card nella prima lista di Trello (Backlog). Ogni membro del team sceglieva una card, la associava al suo nome e la spostava nella lista successiva (DEV). Questo ha permesso un certo grado di parallelismo, mentre per i task più complessi sono stati affrontati assieme.

Come si vede in questo screenshot (Fig. 2) le card avanzano da sinistra verso destra fino a giungere all'ultima lista (Archive) che rappresenta i task completati.

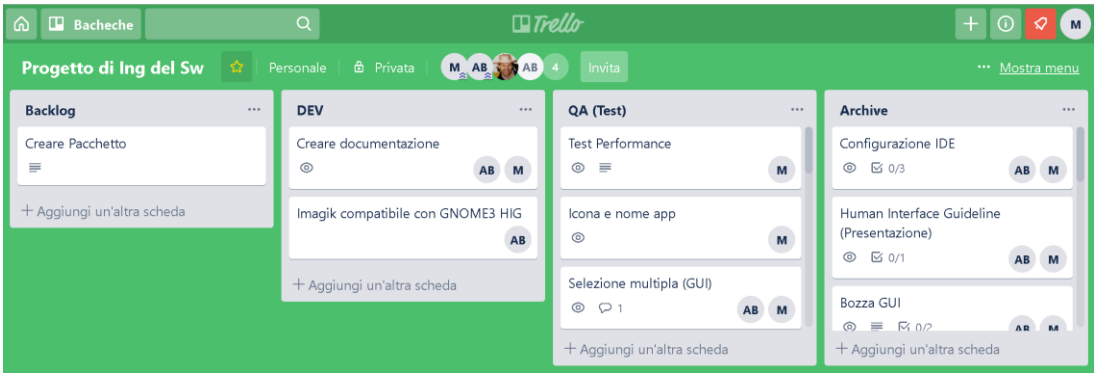

Fig. 2 – La Trello board usata dal gruppo di lavoro

3.2.3 Versioning

Lo strumento di versioning è stato usato con due branch remoti: *master* e *dev*. Sul branch dev abbiamo fatto tutti i push man mano che il programma e le relative funzionalità prendevano forma. Quando tutte le feature sono state sviluppate abbiamo reintegrato (merge) sul master (Fig. 3).

Fig. 3 – Il branching fatto su github

Abbiamo sempre avuto cura di fare il push solo di codice che compilava correttamente e con funzionalità verificate, con questo approccio abbiamo potuto ridurre al minimo l'utilizzo di roll back a versioni precedenti dello sviluppo.

4 Progettazione e Sviluppo

4.1 Mockup

Il presente mock up (Fig. 4) è stato improntato prima che venisse assegnato al nostro team il compito di seguire le direttive HIG di Gnome.

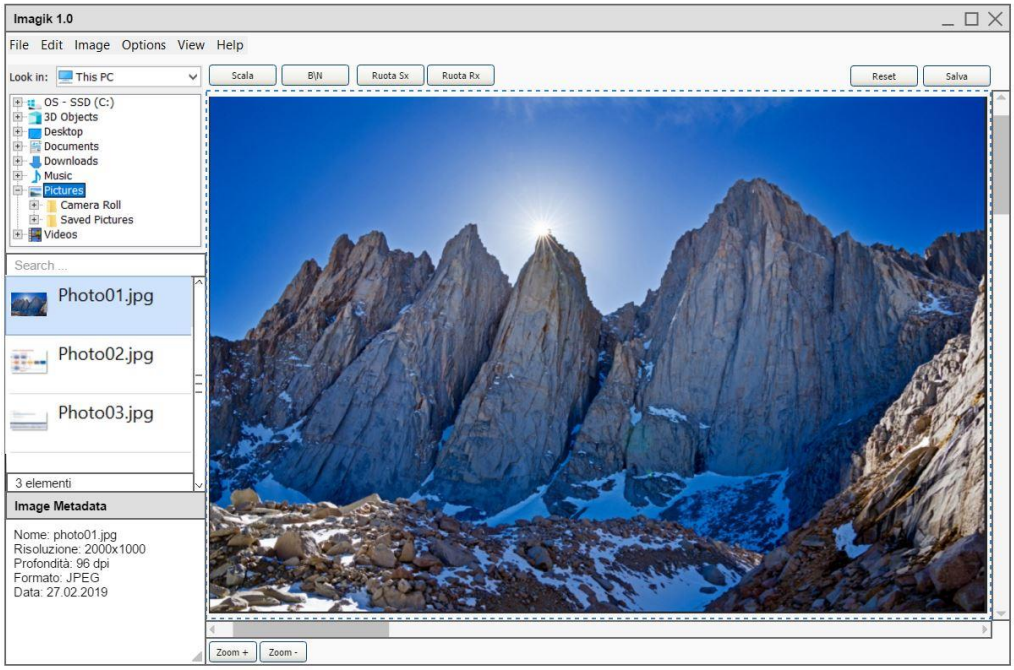

Fig. 4 – Mockup

Nella parte di sviluppo vedremo come è stata sviluppata l'interfaccia grafica del programma. Benché questa si sia evoluta rispetto al mock up proposto, ci sono alcuni elementi che ritroveremo, tra cui:

- Lista immagini disponibili nella cartella selezionata.
- Campo di ricerca in testa alla lista immagini.
- Posizione del metadata viewer.
- Posizione e dimensione dell'area di anteprima immagine.

Per rispettare le direttive GNOME abbiamo nascosto il menu di sistema integrandolo nella toolbar. I bottoni sono stati riorganizzati e provvisti di icona e tooltip. Abbiamo inoltre deciso che la selezione della directory avvenisse tramite dialog di sistema invece che implementare un controllo ad hoc per il browsing delle risorse del computer.

4.2 Human Interface Guidelines

Al nostro team è stato affidato il compito di studiare la H.I.G. di GNOME 3 e di valutarne l'implementazione nel nostro applicativo. Visto l'ampiezza degli argomenti trattati abbiamo scelto di concentrarci su alcuni punti da noi considerati rappresentativi e non da ultimo anche raggiungibili. Segue una schematizzazione di cosa abbiamo scelto.

Nome e icona	Il nome deve comunicare lo scopo dell'applicazione. L'icona deve rendere l'app facilmente riconoscibile, darne un'identità'.
UI semplice	Includere solo informazioni e controlli necessari per rendere l'esperienza più gradevole e meno stancante.
UI progressiva	Nascondere i controlli non necessari e mostrarli solo quando ce n'è bisogno.
Automatizzare	Evitare di far lavorare l'utente quando il programma può farlo autonomamente.
Gerarchia UI	Priorità: Sinistra → Destra e Alto → Basso.

Contenuto	Dare più spazio al contenuto principale e meno ai controlli/info secondarie.
Spacing	Le label, i campi ed i bottoni devono rispettare distanze ben definite.

UI Semplice

Abbiamo optato per un design minimalista.

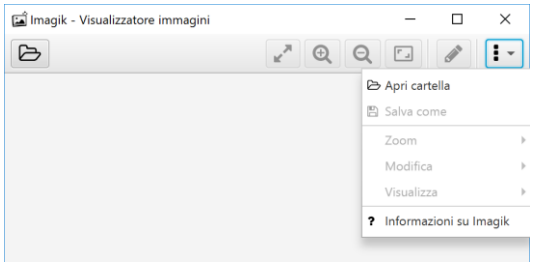

Fig. 5 – Interfaccia di Imagik Image Viewer

Il menu testuale è nascosto e richiamabile con un sistema a tendina (Fig. 5). Il menu presenta tutte le funzionalità dei bottoni, più la parte di About. Solo le funzionalità utilizzabili sono attive/visibili.

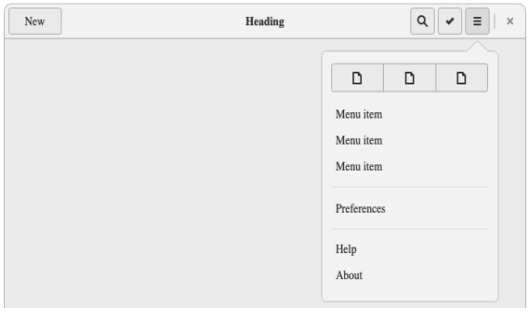

Fig. 6 - Esempio dalla HIG di GNOME

Per referenza riportiamo l'esempio del Primary Menu proposto dalla HIG di Gnome (Fig. 6), a cui ci siamo ispirati per la UI della nostra applicazione. Purtroppo, l'integrazione dei bottoni nell'header avrebbe richiesto troppo tempo per l'implementazione, abbiamo deciso di non farlo in questo primo rilascio.

UI Progressiva

Le funzionalità appaiono solo quando necessarie. La GUI si modifica in base alle necessità.

Per esempio, all'avvio dell'applicazione l'unica operazione possibile è la selezione di una directory (Fig. 7). Per questo motivo la barra di ricerca e la lista delle immagini vengono visualizzate solo dopo aver selezionato una directory (Fig. 8).

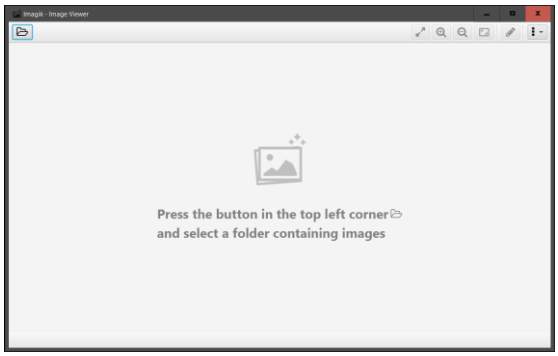

Fig. 7 – In attesa di selezione dela cartella

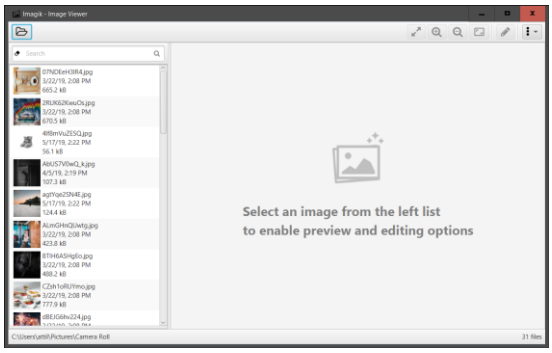

Fig. 8 – La lista di immagini è caricata

Quando non ci sono immagini selezionate i bottoni sono disabilitati (Fig. 9) con lo scopo di comunicare la necessità da parte dell'utente di fare qualcosa prima di poter usare certe funzioni.

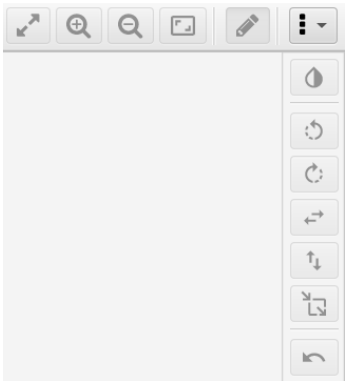

Fig. 9 – Bottoni disabilitati

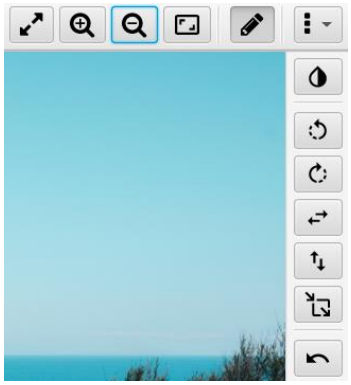

Fig. 10 – Bottoni attivati

Automatizzare

La funzionalità di bulk processing permette all'utente di modificare un gran numero di immagini per volta. Una dialog avverte l'utente su quanti e quali file verrà applicata la modifica (Fig. 11)

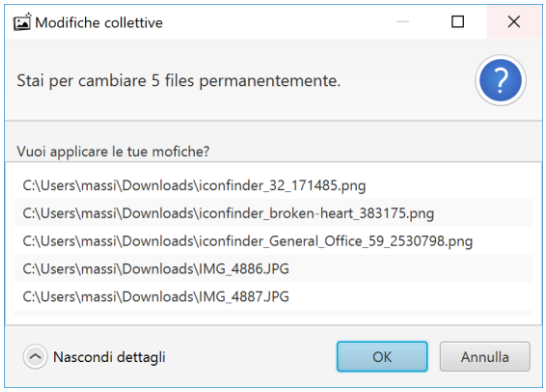

Fig. 11 – Richiesta di conferma per bulk operation

Gerarchia UI

Per organizzare la posizione dei controlli nella UI abbiamo seguito il pattern grafico "F Layout", dove le informazioni e i controlli più importanti sono in alto a sinistra e quelli meno importanti in basso a destra (Fig. 12).

1. In alto a sinistra solo la funzione base di partenza, obbligatoria per aprire il resto delle funzionalità.
2. In alto a destra le funzionalità importanti, dipendenti dalla prima scelta.
3. In basso solo dati informativi.

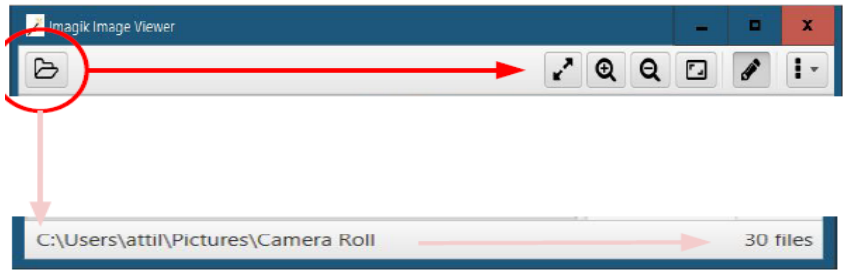

Fig. 12 – F Layout usato in Imagik

Contenuto

Il nostro applicativo riserva la maggior parte dello spazio alla sua funzionalità principale: Visualizzatore di Immagini (Fig. 13).

Fig. 13 – Gran parte dell'interfaccia è dedicata alla visualizzazione dell'immagine selezionata

Spacing

Raggruppare i controlli in gruppi e usare spaziature standard dettate da GNOME 3.

La dialog di ridimensionamento immagine (Fig. 14) rispetta le direttive sulle spaziature standard definite da GNOME 3 (Fig. 15).

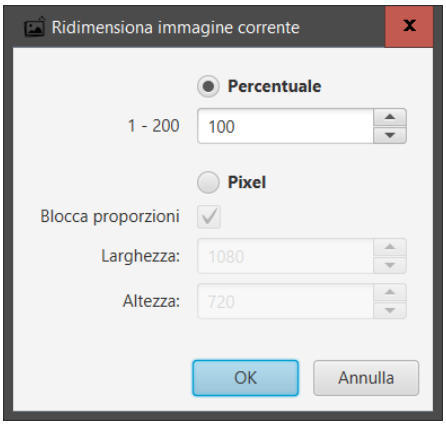

Fig. 14 – Dialog di resize

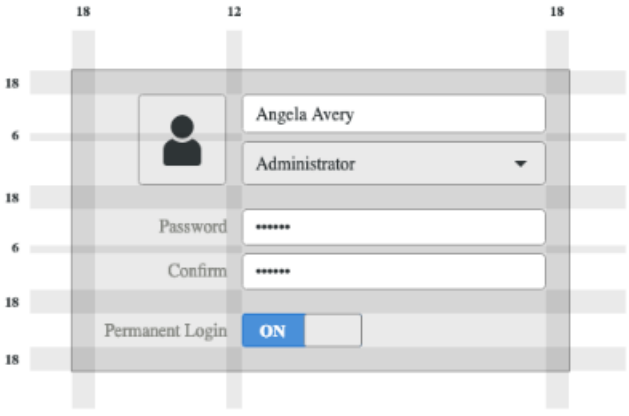

Fig. 15 – Proporzioni consigliate dalla HIG di Gnome

4.3 GUI

Parte dell'interfaccia grafica l'abbiamo indirettamente vista mettendo a confronto la HIG di GNOME con la nostra GUI, di seguito proponiamo alcune particolarità della nostra interfaccia che non sono strettamente legate a GNOME.

4.3.1 Barra di avanzamento lavoro sulle bulk operation

Quando si eseguono operazioni su più file, una dialog mostra l'avanzamento del lavoro (Fig. 16). Le informazioni riportate sono: il nome del file in elaborazione e la percentuale di lavoro raggiunto (in forma di progress bar).

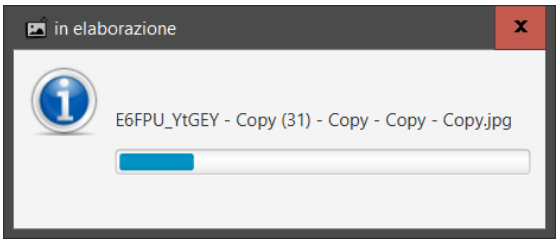

Fig. 16 – Progress bar

4.3.2 Metadata viewer

Attraverso il menu, sotto la voce “Visualizza”, è possibile attivare la finestra “Metadata” (Fig. 17)
La vista “Metadata” mostra le informazioni EXIF contenute nei file di tipo immagine (Fig. 18).

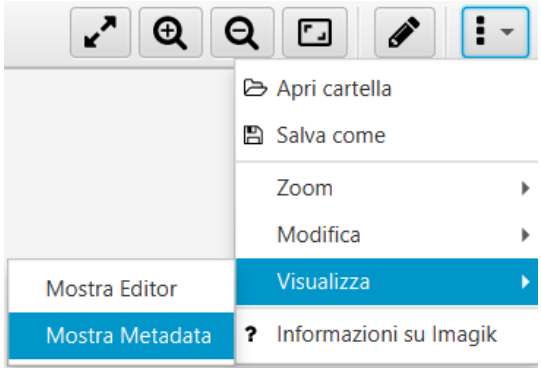

Fig. 17 – Menu

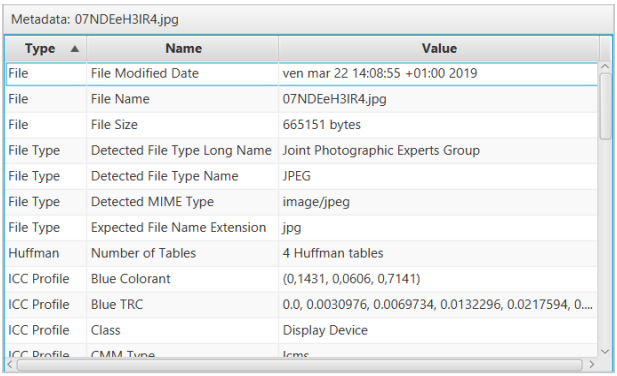

Fig. 18 – Metadati EXIF

4.3.3 Campo Search

Il campo search applica un filtro sulla lista delle immagini ad ogni carattere digitato. Il risultato è una lista di immagini il cui nome contiene il testo digitato (Fig. 19).

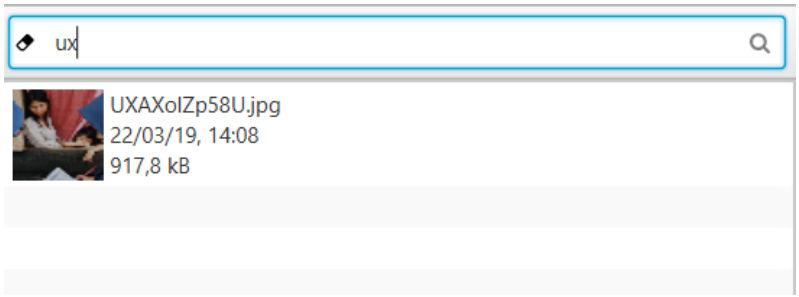

Fig. 19 – Search bar con filtro globbing

4.3.4 Lista Immagini

Ogni elemento della lista immagini è composto da un piccolo Thumbnail di preview e dalle informazioni principali:

- nome file
- data ultima modifica
- dimensione file.

4.4 Diagramma delle classi

4.4.1 Organizzazione del progetto

Per rendere più semplice lo sviluppo della UI, abbiamo deciso di scomporla in diversi controlli grafici ognuno dei quali composto da un Controller e una View secondo il patter MVC (Model View Controller).

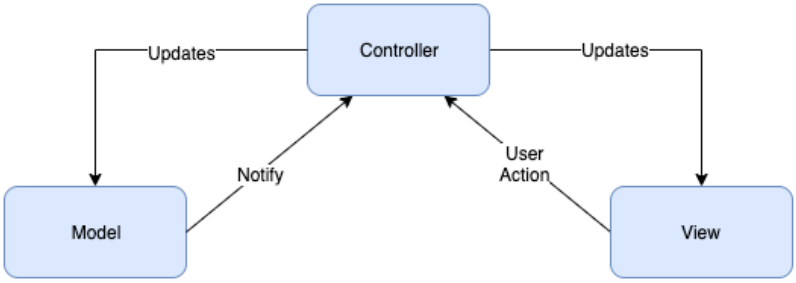

Fig. 20 – Schema pattern MVC

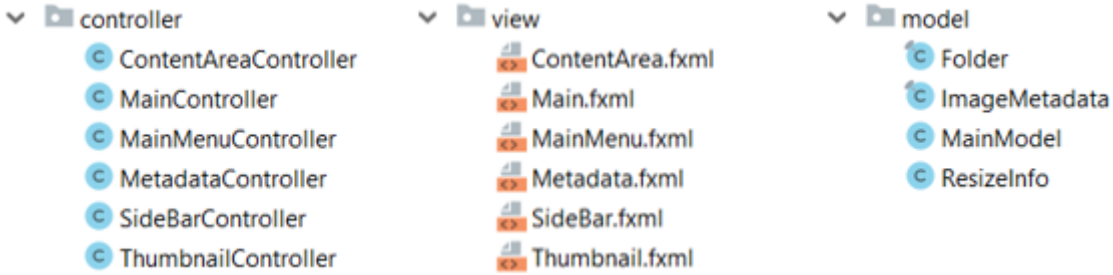

Fig. 21 – Classi e file coinvolti nel pattern MVC

Inoltre il progetto è stato suddiviso in diversi package.
Documentandoci abbiamo visto che fondamentalmente c'erano due scuole di pensiero:

- Organizzare i file per tipologia
- Organizzare i file per funzionalità

Abbiamo scelto la prima opzione, mettendo tutte le classi controller dentro il package controller, tutte i file FXML dentro il package view e tutte le classi modello nel package model.

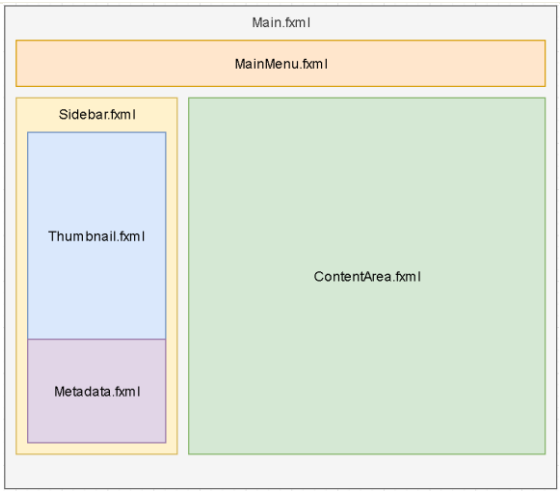

Fig. 22 - Composizione della UI tramite controlli

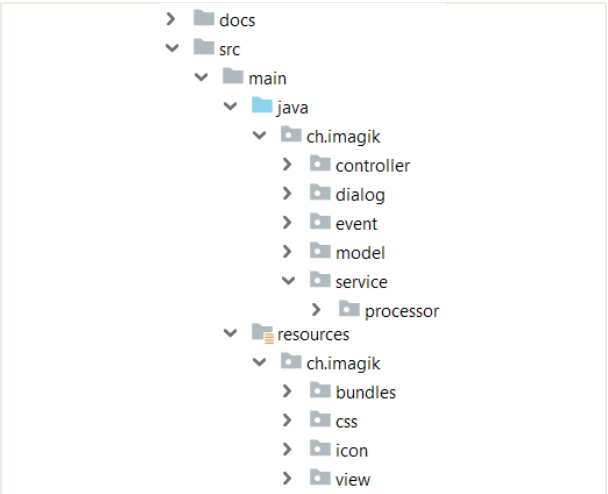

Fig. 23 - Struttura progetto

4.4.2 Events

Per coordinare l'aggiornamento dei vari componenti dell'applicazione e **ridurre l'accoppiamento** tra gli stessi, abbiamo introdotto l'uso di un sistema ad eventi (Google guava Event Bus).

Gli attori principali in questo pattern sono:

- EventBus
- Publisher
- Subscriber
- Event / Message

L'event bus fa da collante tra il Publisher e i Subscribers.

Il publisher invia un messaggio/evento all'event bus, ma non sa chi riceverà il messaggio. Tutti i subscribers, che in fase di inizializzazione si sono registrati sull'event bus, ricevono il messaggio.

La seguente immagine (Fig. 21) mostra come il meccanismo degli eventi è stato sfruttato per implementare la funzione di Logging. Il LogService implementa l'interfaccia EventSubscriber e si registra sui messaggi di tipo EventLoggable. Gli eventi che vogliamo vengano loggatti implementano l'interfaccia EventLoggable oltre all'interfaccia EventBase (es. BlackWhiteEvent).

Gli eventi di zoom invece non implementano l'interfaccia EventLoggable perché non devono essere loggati.

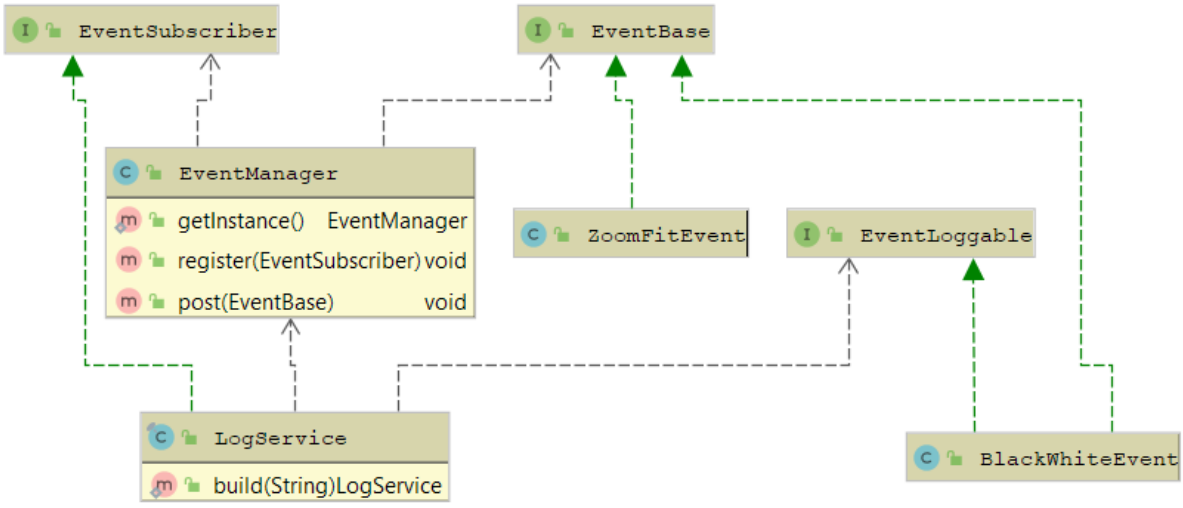

Fig. 24 – Schema UML delle classi del bus degli eventi

4.4.3 Processors e ImageService

Inizialmente la classe ImageService aveva un metodo per ogni filtro grafico (Fig. 22). I filtri venivano chiamati direttamente dal ContentAreaController. Se l'operazione era di tipo bulk, la lista di File in combinazione alla function da applicare, venivano passati al metodo multiSelectionImageEdit.

Fig. 25 – Prima versione dei filtri nella classe ImageService

Per poter soddisfare il principio Open/Closed principle (Open to extension / Closed for modification), abbiamo rifattorizzato la classe ImageService in modo che potesse richiamare i filtri grafici via reflection.

Tutti i filtri grafici sono stati implementati come classi separate che estendono la classe astratta Processor. In questo modo, per introdurre un nuovo filtro è sufficiente implementare una nuova classe che estende Processor senza bisogno di modificare ImageService.

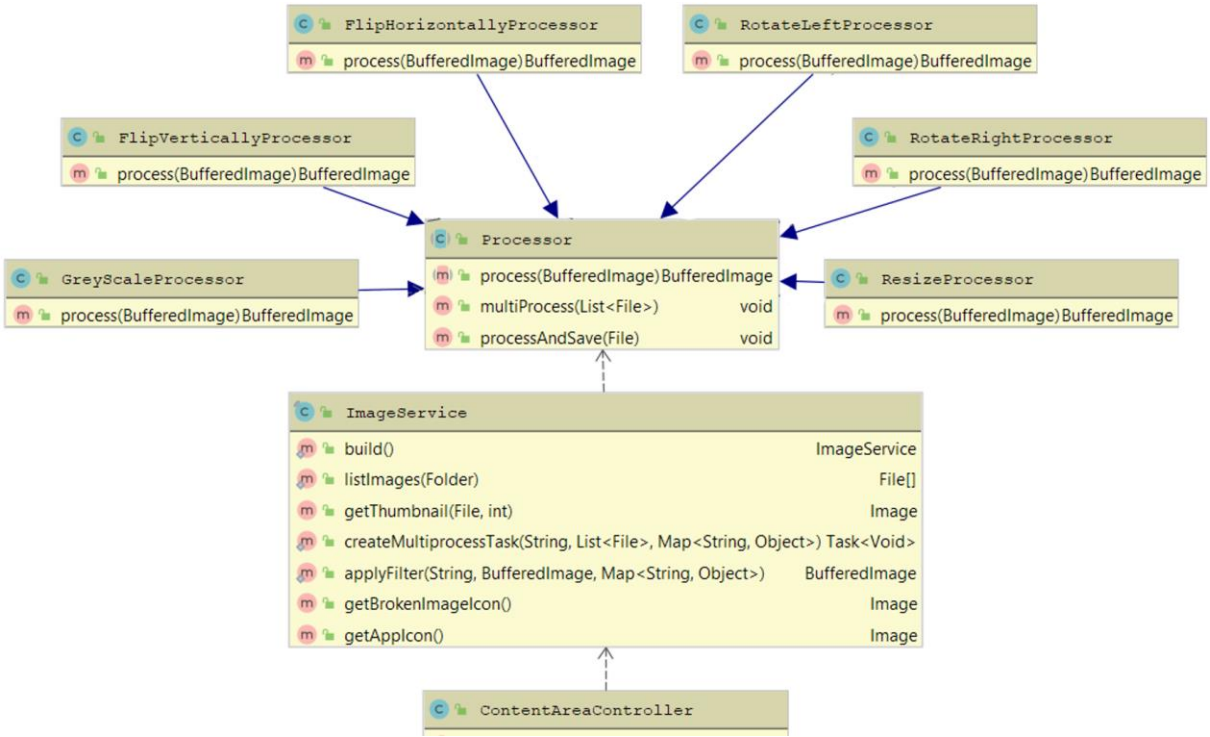

Fig. 26 – Ristrutturazione dei filtri con la classe Processor estesa dalle varie funzionalità

Nell'immagine precedente (Fig. 26) vediamo la versione finale di ImageService e i filtri grafici che estendono Processor.

Come si può notare ci sono due tipi di relazione:

- dipendenza → freccia tratteggiata
- specializzazione → freccia con punta piena

4.5 Altri diagrammi UML

4.5.1 Activity diagram

Imagik è stato tradotto in italiano e inglese. Il software verifica la lingua di sistema e visualizza i testi nella lingua opportuna. In caso di lingua non supportata usa l'inglese come lingua di default.

Se un utente volesse forzare il caricamento di una delle due lingue, può aggiungere la seguente voce nel file di configurazione dell'applicativo.

File di configurazione	`C:\Users\{nomeutente}\.imagik\imagik.properties`
Voce di configurazione	`language=it (o en per inglese)`

Nelle due immagini successive (Fig. 27 e 28) mostriamo il meccanismo di inizializzazione della lingua e della selezione della cartella con relativa lista immagini tramite un activity diagram.

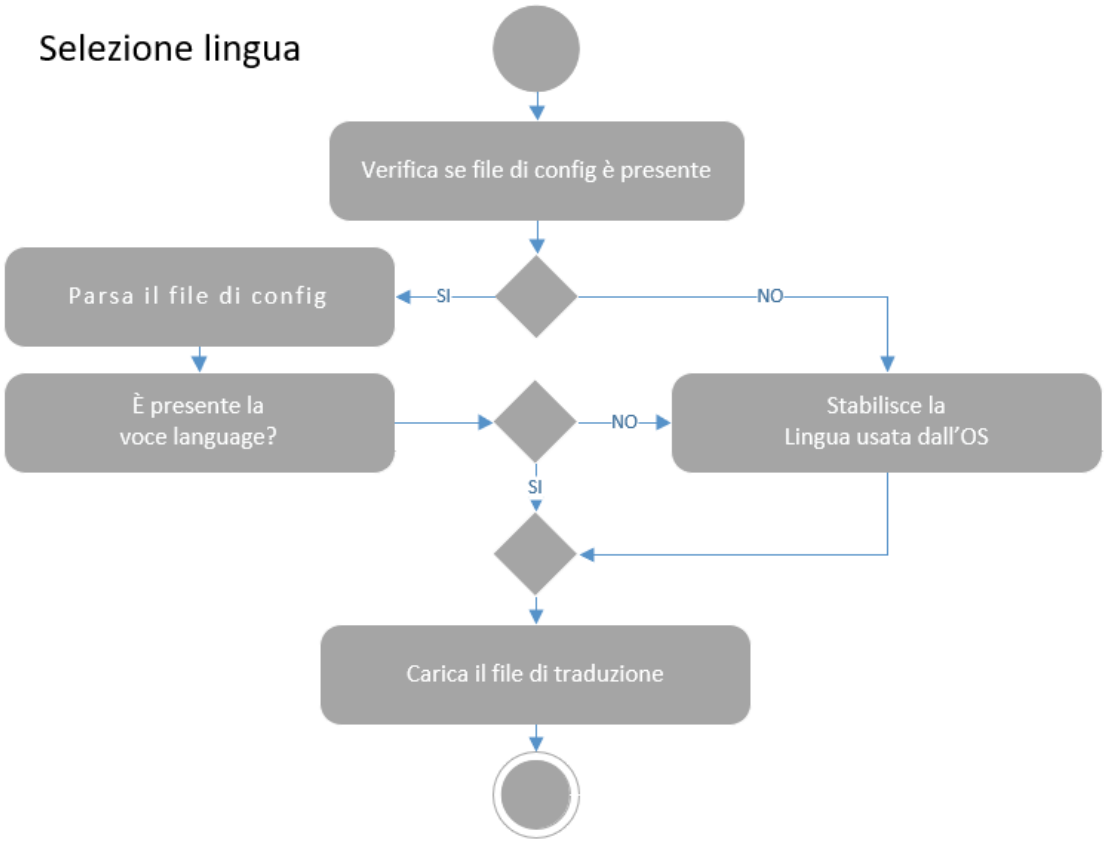

Fig. 27 – Activity diagram selezione lingua

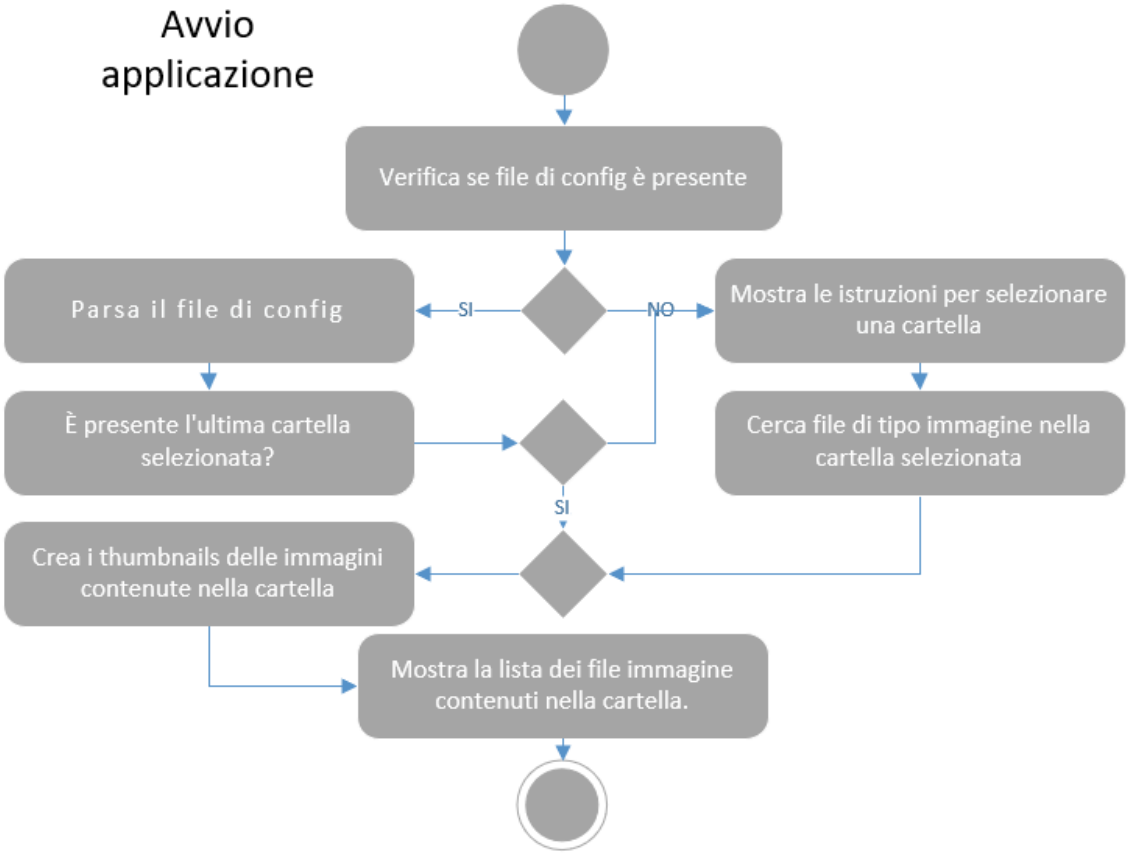

Fig. 28 – Activity diagram caricamento lista immagini

5 Validazione e test

L'applicazione è stata validata tramite test manuali. I test sono stati effettuati principalmente su piattaforme Windows 10 e Mac OS X. Sono stati fatti limitati test su Linux Fedora 29

Ci siamo assicurati che l'applicazione potesse gestire input invalidi senza che andasse in crash. Per esempio, in caso di file invalido, una notifica di tipo "toast message" avvisa l'utente che il file non è supportato. L'immagine in questione viene quindi rimossa dalla lista delle immagini selezionabili.

Abbiamo inoltre verificato che selezionando una cartella con centinaia di immagini, le prestazioni non degradassero (caricamento in background dei thumbnail).

In caso di selezione multipla di immagini, l'applicazione dei filtri grafici può richiedere diverso tempo. Per questo motivo abbiamo deciso di eseguire il calcolo in thread paralleli, notificando l'utente sullo stato di avanzamento tramite una "progress bar". Questo permette di avere una UI sempre responsiva ed evitare l'effetto "not responding…".

6 Problemi e soluzioni

Event Bus

Per rendere più semplice lo sviluppo della UI, abbiamo deciso di scomporla in diversi controlli grafici ognuno dei quali aveva il suo controller.

Il vantaggio di questa tecnica è la riduzione della complessità di ogni controller/vista corrispondente a una riduzione del numero di righe di codice. Un possibile svantaggio è rappresentato dalla necessità di dover coordinare ed eseguire procedure diverse nei vari sottocomponenti a seguito di uno specifico evento. Per esempio, quando un utente seleziona una cartella, i componenti da aggiornare sono diversi: status bar, lista immagini, toolbar, area di preview.

Per risolvere il problema abbiamo introdotto l'uso di un sistema ad eventi (Google Guava Event Bus). Questo sistema ci ha inoltre aiutato nel caso la stessa azione possa essere eseguita tramite diversi comandi (menu di sistema / toolbar). È stato inoltre d'aiuto per l'implementazione del Logging.

Main Model

Un altro problema da risolvere è stato quello di avere uno stato consistente e facilmente accessibile da parte delle viste e dei controller. Per questo motivo abbiamo creato una classe MainModel che implementa il pattern Singleton. Questa classe ha inoltre una serie di proprietà di tipo Observable che permettono l'aggiornamento immediato della UI in caso di cambiamento (Binding).

Source Repository

Ci siamo resi conto che lo strumento di versioning poteva essere usato meglio. Ad esempio, quando si sono decisi importanti re-factoring o sperimentazioni, avremmo potuto creare branch dedicati.

Questo approccio ci avrebbe permesso di valutare meglio i vari design, con i loro pro e contro, potendo switchare facilmente tra i branch.

7 Sviluppi futuri

Abbiamo identificato una lista di possibili funzionalità da implementare:

- automazione dei test
- shortcut da tastiera
- nuovi filtri grafici
- cambio della lingua a runtime tramite un'opzione grafica
- esecuzioni batch tramite linea di comando
- condivisione sui social delle immagini

8 Conclusioni

Possiamo ritenerci soddisfatti del risultato ottenuto considerando che all'inizio del progetto non avevamo nessuna conoscenza del framework JavaFX e dei principi di design di applicativi grafici.

Il progetto è stato molto interessante e la collaborazione all'interno del gruppo proficua.

Come obiettivi futuri abbiamo l'approfondimento dei design patterns e lo studio di sistemi di test automatici.

9 Allegati

- Pacchetto di Rilascio
- Presentazione personalizzazione secondo HIG di Gnome
- Presentazione finale.

10 Bibliografia

Human Interface Guidelines GNOME 3: https://developer.gnome.org/hig/stable

10.1 Strumenti

Strumento	Link
IntelliJ IDEA Ultimate	https://www.jetbrains.com/idea
Trello	https://trello.com/b/TTrP6reB/progetto-di-ing-del-sw
GitHub	https://github.com/code-mds/imagik

10.2 Librerie

Liberia	Link
ImageJ	https://mvnrepository.com/artifact/net.imagej/ij
Metadata Extractor	https://mvnrepository.com/artifact/com.drewnoakes/metadata-extractor
Guava Core Libraries	https://mvnrepository.com/artifact/com.google.guava/guava
OpenJFX	https://mvnrepository.com/artifact/org.openjfx/javafx
Apache Commons IO	https://mvnrepository.com/artifact/commons-io/commons-io
Ikonli	https://mvnrepository.com/artifact/org.kordamp.ikonli/ikonli-javafx