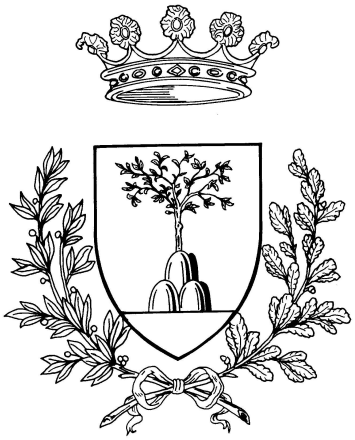

Università degli Studi Di Ferrara

Dipartimento di Ingegneria

Corso di laurea in
Elettronica ed Informatica

Realizzazione di un programma per divisione di file Excel

Relatore:
Prof.
Evelina Lamma

Candidato:
Mirko Covizzi
Matr.: 112394

ANNO ACCADEMICO 2015-2016

INDICE

SOMMARIO

L'obiettivo di questa tesi è descrivere l'implementazione di un programma Java in grado di dividere file Excel (XLS) semplici, cioè formattati con struttura tabulare, utilizzando la libreria Apache POI ed offrendo una moderna interfaccia grafica grazie al framework JavaFX.
Il compito dell'applicativo consiste nell'effettuare la suddivisione del documento Excel in base ai suoi campi attributo (colonne), dando anche la possibilità di scegliere altri campi attributo per la creazione di cartelle e sottocartelle innestate.
L'applicazione verrà poi utilizzata dall'amministrazione Unife per semplificare la suddivisione di documenti Excel che raccolgono dati relativi ai questionari sulla didattica.
La tesi è organizzata nel modo seguente:

- **Capitolo 1**: Vengono presentate le nozioni di linguaggi *Cross Platform*, con un'attenzione particolare a *Java*, il linguaggio di programmazione utilizzato nel progetto; viene trattato il design pattern *Model-View-Controller*, alla base della struttura del progetto; infine sono brevemente descritte le librerie utilizzate dal progetto.

- **Capitolo 2**: Vengono presentate le motivazioni che hanno portato all'idea di progetto insieme ai requisiti richiesti per quest'ultimo; viene inoltre fatta una trattazione dell'ambiente di sviluppo in cui è stato realizzato il software, in particolare viene descritto l'IDE *IntelliJ IDEA* e la sua integrazione con un sistema di *Version Control* (Github).

- **Capitolo 3**: Viene presentata la struttura del progetto, partendo dall'albero della directory, spiegando il compito di ogni componente del software; vengono inoltre mostrate le diverse schermate della *User Interface* (UI).

- **Conclusione**: Vengono indagate possibilità per sviluppi futuri.

CAPITOLO 1

INTRODUZIONE

1.1 Linguaggi Cross Platform

In questa sezione verranno indagate le varie possibilità relative alla realizzazione di un software cross platform. Questa trattazione è particolarmente importante in quanto i requisiti del progetto analizzato in questo elaborato richiedevano esplicitamente la realizzazione di un programma che supportasse diversi sistemi operativi. La scelta del linguaggio di implementazione è perciò ricaduta su Java, il più famoso linguaggio di programmazione cross platform, che viene brevemente descritto nei paragrafi seguenti.

1.1.1 Introduzione

I linguaggi Cross Platform sono linguaggi di programmazione che permettono di sviluppare o eseguire software su più di un tipo di piattaforma hardware. La più utilizzata applicazione Cross Platform è il Web Browser. Sviluppati per qualsiasi piattaforma desktop e mobile, i Browser rappresentano le pagine web "quasi" allo stesso modo, indipendentemente dalla piattaforma su cui queste vengono aperte. Il Browser ricade nella categoria del "Compilare in Linguaggio Macchina". Il più famoso linguaggio di programmazione Cross Platform è Java, che ricade invece nella categoria del "Compilare in un Linguaggio Intermedio".

1.1.2 Problemi per gli Sviluppatori

Il supporto alle diverse piattaforme rappresenta un grande problema per gli sviluppatori software che vogliono rendere il proprio prodotto fruibile da chiunque, indipendentemente dalla piattaforma su cui il software viene eseguito (Windows, Mac, Linux, ecc...). Ci sono due metodi principali per sviluppare i programmi in questo modo. Il primo è compilare un eseguibile per ogni diverso ambiente operativo (sia a livello di linguaggio macchina, sia di Sistema Operativo). Il secondo è utilizzare un linguaggio intermedio, che viene compilato una sola volta e che viene poi interpretato da ogni piattaforma.

1.1.3 Compilare in Linguaggio Macchina

Mantenere gruppi separati di codice sorgente per la stessa applicazione è l'approccio meno desiderato dagli sviluppatori; in alcuni casi, però, questo avviene di routine quando le piattaforme hardware sono diverse. Per esempio, applicazioni in C++ vengono compilate direttamente nel linguaggio macchina del computer designato, utilizzando un compilatore apposito per quell'hardware. Se invece il linguaggio C++ è usato per scrivere un programma sia per Windows che per Macintosh, tipicamente si utilizzano due diversi gruppi di codice sorgente; uno per Windows ed uno per Mac.

1.1.4 Compilare in un Linguaggio Intermedio

Questo secondo metodo utilizza un interprete, come la Java Virtual Machine. Java è cross platform perchè il codice sorgente di un programma viene compilato in un linguaggio intermedio, il "bytecode". Quest'ultimo viene poi eseguito dalla Java Virtual Machine (interprete Java) che viene scritta per la particolare piattaforma hardware su cui esegue. Per design, il bytecode viene eseguito allo stesso modo su tutte le piattaforme hardware su cui è presente una Java Virtual Machine. Emergono però problemi quando l'interprete Java non è aggiornato alla stessa versione del codice che deve eseguire, oppure quando l'interprete non interpreta correttamente il bytecode in base ad un determinato standard.

1.1.5 Java

Java è un linguaggio di programmazione orientato agli oggetti a tipizzazione statica, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. Uno dei principi fondamentali del linguaggio è espresso dal motto WORA (write once, run anywhere, ossia "scrivi una volta, esegui ovunque"): il codice compilato che viene eseguito su una piattaforma non deve essere ricompilato per essere eseguito su una piattaforma diversa. Il prodotto della compilazione è infatti in un formato chiamato bytecode che può essere eseguito da una qualunque implementazione di un processore virtuale detto Java Virtual Machine. Il linguaggio deriva gran parte della sua sintassi dai linguaggi Simula, C e C++, ma ha meno costrutti a basso livello e implementa in modo più puro (rispetto al C++) il paradigma object-oriented. I programmi scritti in linguaggio Java, dopo una fase iniziale di compilazione con ottenimento del cosiddetto bytecode, sono destinati all'esecuzione sulla piattaforma Java attraverso una fase di interpretazione (per questo motivo il linguaggio Java è detto anche semi-interpretato) ad opera di una Java Virtual Machine e, a tempo di esecuzione, avranno accesso alle API della libreria standard. Questi due passi forniscono un livello di astrazione che permette alle applicazioni di essere interamente indipendenti dal sistema hardware su cui esse saranno eseguite. Un'implementazione della piattaforma java è il Java Runtime Environment (JRE), necessario per l'esecuzione del programma compilato, mentre per lo sviluppo dei programmi in Java a partire dal codice sorgente è necessario il Java Development Kit (JDK) che include anche il JRE.

1.2 Model-View-Controller

Il Model-View-Controller è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business. Il componente centrale del MVC, il modello, cattura il comportamento dell'applicazione in termini di dominio del problema, indipendentemente dall'interfaccia utente. Il modello gestisce direttamente i dati, la logica e le regole dell'applicazione. Una vista può essere una qualsiasi rappresentazione in output di informazioni, come un grafico o un diagramma. Sono possibili viste multiple delle stesse informazioni, come ad esempio un grafico a barre per la gestione e la vista tabellare per l'amministrazione. La terza parte, il controller, accetta l'input e lo converte in comandi per il modello e/o la vista. Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- Il **Model** fornisce i metodi per accedere ai dati utili all'applicazione;
- La **View** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- Il **Controller** riceve i comandi dell'utente (in genere attraverso la view) e li attua modificando lo stato degli altri due componenti.

1.3 Librerie

In questa sezione vengono descritte le librerie utilizzate nel progetto analizzato da questo elaborato. Tra queste JavaFX svolge uno dei ruoli più importanti, cioè permette di realizzare interfacce utente moderne e personalizzabili, mantenendo comunque la filosofia cross platform tipica di Java. Apache POI rappresenta invece la principale libreria del progetto, poichè fornisce metodi ed astrazioni comodi e versatili per la manipolazione di documenti XLS. Infine, la libreria Apache Commons è stata utilizzata nel progetto in un paio di occasioni per facilitare la gestione di stringhe e di file.

1.3.1 JavaFX

JavaFX è una famiglia di software applicativi, basati sulla Piattaforma Java, per la creazione di *rich Internet applications*, applicazioni web che hanno tutte le caratteristiche e funzionalità delle comuni applicazioni per computer. L'aspetto delle applicazioni JavaFX può anche essere personalizzato. I *Cascading Style Sheets* (CSS) permettono infatti di separare l'aspetto e lo stile dall'implementazione, così gli sviluppatori possono concentrarsi sulla logica del loro codice mentre i designer si concentrano sullo stile. Per chi avesse esperienza nel web design, o per chi volesse separare l'interfaccia utente (UI) dalla logica back-end, c'è la possibilità di sviluppare gli aspetti della UI in linguaggio di scripting FXML, linguaggio di markup simile ad XML, ed usare il codice Java per la parte logica dell'applicazione. Inoltre, se si preferisce realizzare interfacce senza scrivere codice, si può utilizzare il *JavaFX Scene Builder*. Infatti, mentre viene creata la UI, lo Scene Builder crea un documento di markup in FXML

che può essere poi importato in un *Integrated Development Environment* (IDE). A partire da Java 7, la libreria di JavaFX è distribuita con il Java Runtime Environment. E' inoltre fondamentale sottolineare che per eseguire applicazioni che utilizzano JavaFX 2.0, come l'applicativo analizzato in questo elaborato, è necessario disporre di una versione di JRE 1.8+, in quanto JavaFX 2.0 utilizza le nuove *lambda expressions* introdotte in Java 8, ma anche poichè la nuova iterazione comporta un cambiamento di paradigma rispetto alla versione JavaFX precedente.

1.3.2 Apache POI

Apache POI è un progetto supportato dalla Apache Software Foundation che fornisce librerie Java per leggere e scrivere file in formati Microsoft Office, come Word, PowerPoint ed Excel. Il progetto Apache POI contiene molte sottocomponenti, tra cui HSSF (Horrible SpreadSheet Format), che permette di elaborare file Excel XLS. La libreria Apache POI può essere scaricata dalla pagina `https://poi.apache.org/`, nella sezione *Download*.

1.3.3 Apache Commons

Apache Commons è un progetto supportato dalla Apache Software Foundation che fornisce software Java riutilizzable ed Open Source. Per esempio, Commons IO è una collezione di utilities per l'Input/Output. Commons Lang è invece una libreria che permette di estendere le funzionalità delle classi di java.lang. La libreria Apache Commons si può trovare alla pagina `http://commons.apache.org/`.

CAPITOLO 2
PROGETTO

2.1 Motivazioni e requisiti

La motivazione principale che ha portato alla nascita di questo progetto è stata la necessità di un software in grado di automatizzare un task normalmente svolto dagli operatori dell'amministrazione di Unife. Il task da automatizzare consiste nella divisione di file Excel, contenenti dati e statistiche relativi ai questionari sulla didattica, in base a specifici attributi. Per esempio, dividere un documento in base al campo "Dipartimento" porta ad ottenere un numero di documenti pari al numero di dipartimenti presenti nella colonna "Dipartimento", ognuno contenente le tuple relative a tale dipartimento. Il programma da realizzare deve inoltre essere in grado di creare cartelle e sottocartelle in base ad altri campi attributo selezionati. Ogni cartella e sottocartella avrà il nome corrispondente alle diverse *entries* per l'attributo selezionato, mentre i file singoli dovranno avere un nome ottenuto dalla concatenazione del nome del file originale più le diverse *entries* della colonna selezionata per lo splitting. Il programma da realizzare deve inoltre essere *cross platform*.

2.2 Ambiente di sviluppo

In questa sezione vengono descritti i software e gli strumenti utilizzati durante la realizzazione del progetto.

2.2.1 IntelliJ IDEA

IntelliJ IDEA è un ambiente di sviluppo integrato (IDE) per il linguaggio di programmazione Java. Ogni aspetto di IntelliJ IDEA è specificatamente progettato per massimizzare la produttività dello sviluppatore. Infatti, attraverso un potente sistema di analisi statica del codice ed un design ergonomico, rende la progettazione non solo più produttiva ma anche più comoda. Dopo aver indicizzato il codice sorgente, IntelliJ IDEA offre un'incredibile esperienza di programmazione dando suggerimenti rilevanti a seconda del contesto: un com-

pletamento del codice veloce ed intelligente, un'analisi del codice *on-the-fly* ed affidabili strumenti di refactoring. IntelliJ IDEA offre inoltre una completa integrazione con strumenti di tipico utilizzo nei grandi progetti, come il *Version Control*, ed il supporto nativo a molti linguaggi di programmazione, senza dover installare plugins aggiuntivi. La figura 2.1 mostra la schermata di progettazione di IntellJ IDEA.

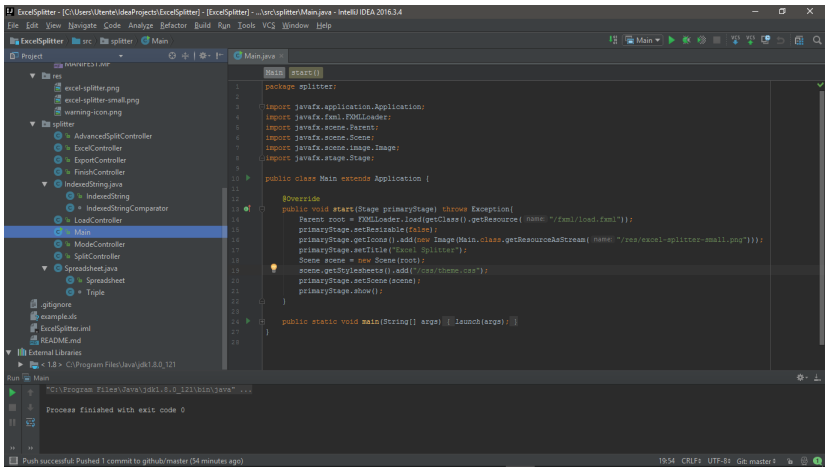

Figura 2.1: Schermata di progettazione di IntelliJ IDEA

2.2.2 JavaFX Scene Builder

JavaFX Scene Builder è uno strumento che permette all'utente di realizzare velocemente interfacce per applicazioni JavaFX, senza dover scrivere codice. L'utente può trascinare componenti della UI nell'area di lavoro, modificare le loro proprietà, applicare loro dei fogli di stile (CSS) ed il codice per il layout viene generato automaticamente. Il risultato è un file FXML che può successivamente essere combinato con un progetto Java associando la UI alla logica dell'applicazione. La figura 2.2 mostra la schermata di design di JavaFX Scene Builder.
Scene Builder permette di posizionare facilmente componenti di interfaccia JavaFX come pulsanti, checkbox, etichette, in modo da prototipare velocemente le interfacce utente. Animazioni ed effetti possono essere applicati in maniera semplice, per ottenere UI più sofisticate. Scene Builder può essere inoltre combinato con qualsiasi IDE Java, come IntelliJ IDEA. La UI può essere quindi associata al codice sorgente che gestisce gli eventi e le azioni di ciascun elemento ed ogni modifica apportata al file FXML all'interno dell'IDE si rifletterà automaticamente nel progetto in Scene Builder, permettendo così allo sviluppatore di modificare l'interfaccia sia dal punto di vista del codice che dal punto di vista visivo, in maniera intercambiabile. Infine, Scene Builder è sviluppato come applicazione JavaFX, supportata da Windows, Mac OS X e Linux ed è perciò un perfetto esempio di applicazione desktop sviluppata in JavaFX.

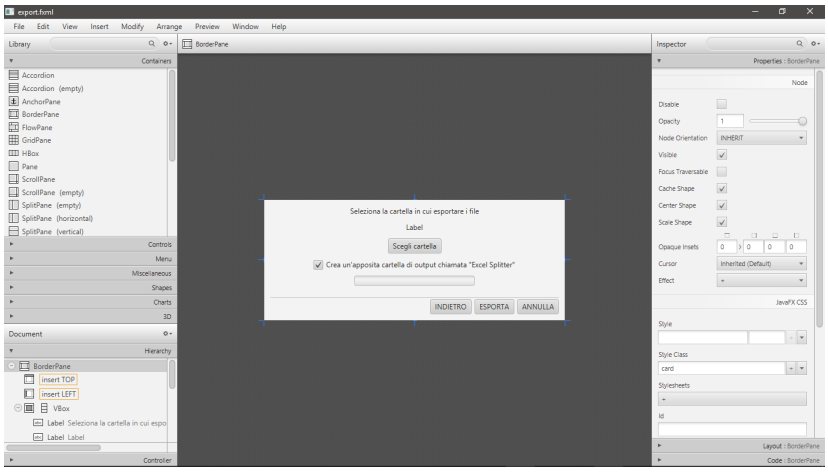

Figura 2.2: Schermata di design di JavaFX Scene Builder

2.2.3 Version Control

Il Version Control è un sistema che registra cambiamenti ad un file o ad un insieme di file nel tempo, così da permettere di poter recuperare specifiche versioni anche dopo le modifiche. Il Version Control permette di far tornare un file ad uno stato precedente, far tornare un intero progetto ad uno stato precedente, comparare i cambiamenti apportati nel corso del tempo, vedere quali file modificati di recente possono aver causato problemi, vedere chi ha introdotto errori e quando, e così via. L'utilizzo di un sistema di version control si è rivelato infatti estremamente utile durante lo sviluppo del programma *Excel Splitter*, poichè in più di un caso è stato necessario far tornare il progetto ad una versione più stabile, in seguito a modifiche del codice che hanno portato ad instabilità.

2.2.3.1 Registrare i cambiamenti

Un sistema di version control è principalmente basato attorno ad un concetto, la registrazione di cambiamenti che avvengono all'interno di cartelle o di file. A seconda del sistema di version control, si può essere in grado di sapere se e quale file è cambiato fino a conoscere quali specifici caratteri o byte siano cambiati all'interno di un file. Nella maggior parte dei casi, l'utente specifica quale cartella o insieme di file debba essere controllato dal sistema di version control. Questo può avvenire, per esempio, clonando una repository da un host esistente, oppure dicendo al software quale file o cartelle si desidera che siano controllate dal sistema. L'insieme di file o cartelle gestiti da un version control viene comunemente detto *repository*. Mentre l'utente apporta cambiamenti ai file, il sistema registra queste azioni dietro le quinte. Questo processo risulta trasparente all'utente, finché quest'ultimo decide di fare *commit* dei cambiamenti.

2.2.3.2 Commit

Mentre l'utente lavora con i suoi file coperti dal version control, ogni cambiamento è registrato automaticamente. Questi cambiamenti includono la modifica di un file, la cancellazione di una cartella, l'aggiunta di un file, lo spostamento

di file o semplicemente qualsiasi azione che possa alterare lo stato di un file. Invece di registrare ogni modifica individualmente, il sistema di version control aspetta che l'utente presenti e sottoscriva i cambiamenti come una singola collezione di azioni. Nel version control, questa collezione di azioni è conosciuta come *commit*.

2.2.3.3 Revisioni e Changeset

Quando viene effettuato un commit, i cambiamenti sono registrati in un changeset a cui viene dato un unico valore di revisione. Questa revisione può essere nella forma di numero incrementale (1, 2, 3) oppure sotto forma di hash unico (come 846eee7d92415cfd3f8a936d9ba5c3ad345831e5) a seconda del sistema di version control utilizzato. Conoscendo la revisione di un changeset diventa molto facile vederlo o referenziarlo in futuro. Un changeset può includere un riferimento alla persona che ha compiuto il commit, la data della modifica, i file o le cartelle coinvolte, un commento e persino i cambiamenti avvenuti all'interno dei file stessi (linee di codice). Quando si parla di collaborazione, vedere revisioni passate e changeset diventa uno strumento di immenso valore, per vedere come il progetto è evoluto nel tempo e per analizzare il codice degli altri membri del team. Infine, ogni sistema di version control ha un suo particolare modo di formattare e visualizzare la storia di ogni revisione e changeset della repository.

2.2.3.4 Github

Github è un esempio di sistema di version control. Si può trovare all'indirizzo `https://github.com/`. Nella figura 2.3 è possibile osservare la pagina della repository relativa al progetto di tesi analizzato in questo elaborato, che si può trovare all'indirizzo `https://github.com/MirkoCovizzi/ExcelSplitter`.

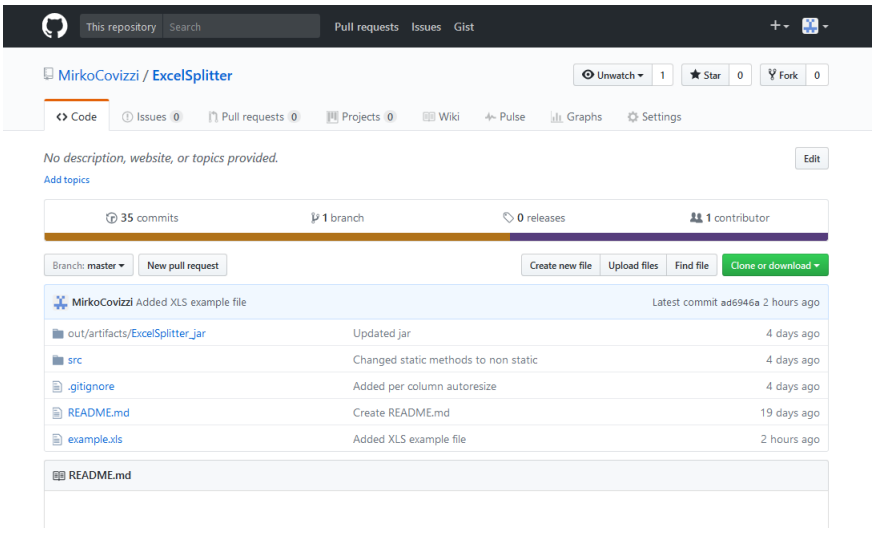

Figura 2.3: Pagina di una repository di Github

CAPITOLO 3
ARCHITETTURA DEL PROGRAMMA

3.1 Struttura del progetto

3.1.1 Albero della directory

```
/
├── css/
│   └── theme.css
├── fxml/
│   ├── advanced_split.fxml
│   ├── dialog.fxml
│   ├── export.fxml
│   ├── finish.fxml
│   ├── help.fxml
│   ├── load.fxml
│   ├── mode.fxml
│   └── split.fxml
├── res/
│   ├── excel-splitter-small.png
│   ├── excel-splitter.png
│   └── warning-icon.png
└── splitter/
    ├── AdvancedSplitController.java
    ├── ExcelController.java
    ├── ExportController.java
    ├── FinishController.java
    ├── IndexedString.java
    ├── LoadController.java
    ├── Main.java
    ├── ModeController.java
    ├── SplitController.java
    └── Spreadsheet.java
```

3.1.2 Descrizione delle cartelle

- **css**: questa cartella contiene lo stile dell'applicazione, implementato in CSS nel file *theme.css*.
- **fxml**: questa cartella contiene i diversi file di markup FXML che rappresentano le diverse schermate dell'applicazione.
- **res**: questa cartella contiene le diverse immagini utilizzate dal programma; *excel-splitter.png* è l'icona del programma, mentre *warning-icon.png* è un'icona utilizzata nelle finestre di warning.
- **splitter**: questa cartella contiene la logica del programma. Vi sono le classi Controller e le altre classi utilizzate dal software.

3.2 Model

In questo progetto, il modello è rappresentato dalla classe *Spreadsheet*. Questa classe raccoglie tutti i metodi per il caricamento, l'elaborazione (splitting) ed esportazione di documenti XLS adeguatamente formattati a struttura tabulare. Essendo una classe Model, essa ha il compito di alleggerire i controller dalla logica di back-end per la manipolazione e la gestione dei dati.

3.2.1 La classe Spreadsheet

La classe Spreadsheet rappresenta il cuore del progetto, in quanto fornisce i meccanismi e le astrazioni per la manipolazione dei file Excel elaborati dall'applicazione.

Spreadsheet
- sheet : HSSFSheet - name : String - directory : String - subdirectory : String - columns : List<String>
+ (All getters and setters) + Spreadsheet(File file) : void + Spreadsheet(String name) : void + Spreadsheet(String name, String directory) : void + Spreadsheet(String name, String directory, String subdirectory) : void - isFormatCorrect() : boolean + copyRow(Spreadsheet s2, int row1, int row2) : void + addRow(Spreadsheet s2, int row1) : void + autoSizeColumns() : void + Spreadsheet(String name, String directory, String subdirectory) : void + split(int directoryCol, int subdirectoryCol, int column) : List<Spreadsheet> + export(String filename) : void

Attributi della classe:

- **sheet**: questo attributo identifica il foglio di calcolo; viene popolato dai costruttori ed acceduto esternamente dal suo metodo getter.
- **name**: questo attributo identifica il nome del foglio di calcolo, epurato della sua estensione. Viene utilizzato durante le fasi di esportazione per determinare i nomi finali dei file da esportare.
- **directory**: questo attributo identifica la directory del foglio di calcolo corrispondente a *sheet*.
- **directory**: questo attributo identifica la sottodirectory del foglio di calcolo corrispondente a *sheet*.
- **columns**: questo attributo identifica la lista di colonne che determinano la struttura tabulare del file Excel da elaborare. Per esempio, un documento può avere colonne *Dipartimento*, *Insegnamento*, *Corso*, ecc...

Principali metodi della classe:

- **Spreadsheet(File file)**: questo metodo rappresenta il principale costruttore della classe Spreadsheet, in quanto prende come argomento il file Excel da elaborare, ne prende il primo foglio inizializzando l'attributo *sheet*. Questo metodo, per la natura delle azioni che compie, prevede la possibilità di lanciare *IOExceptions*, che vengono gestite ad un livello superiore, dalla logica di controllo. Inoltre, all'interno di questo metodo, attraverso l'invocazione di *isFormatCorrect* si verifica se il file XLS importato ha un formato adeguato ad essere manipolato dai metodi della classe Spreadsheet. Se il file non è adeguato, viene lanciata anche in questo caso una eccezione IO. Infine, in questo metodo vengono inizializzati anche gli altri attributi.
- **isFormatCorrect()**: questo metodo svolge il compito di controllare se il file XLS importato ha un formato valido. L'implementazione effettuata dal progetto nella versione corrente prevede un semplice algoritmo di controllo che osserva se la prima riga è vuota o meno; in caso affermativo restituisce *false* altrimenti restituisce *true*. Avendo dedicato un metodo interamente al controllo della formattazione del file si è aumentata la modularità del progetto ed è perciò possibile, in estensioni future, migliorare il progetto aggiungendo in questo metodo altri check più avanzati.
- **split(int directoryCol, int subdirectoryCol, int column)**: questo è il metodo più importante del programma, in quanto svolge l'azione che identifica il software stesso. Come argomenti prevede una *directoryCol*, un numero intero che identifica l'attributo colonna da considerare per effettuare una suddivisione in cartelle: un numero positivo per identficare la colonna, oppure -1 nel caso in cui non si voglia una suddivisione in cartelle; allo stesso modo viene considerato l'argomento *subdirectoryCol*; l'argomento *column* identifica invece la colonna su cui effettuare lo split vero e proprio che conseguirà nella determinazione dei file da esportare. L'argomento *column* è sempre positivo in quanto viene sempre scelta una colonna su cui fare la suddivisione.

- **export(String filename)**: questo metodo permette di esportare il file Excel associato all'attributo *sheet*.

3.3 View

In questo progetto, la View è rappresentata dai documenti di markup FXML contenuti nella cartella *fxml*. Ognuno di questi file identifica una specifica schermata dell'applicazione, ognuna gestita dal corrispondente controller.
Di seguito una breve trattazione di ognuna delle View dell'applicazione:

- **load.fxml**: questa rappresenta la prima schermata che l'utente vede. Dà la possibilità di caricare il documento XLS da dividere sia attraverso un pulsante, *SCEGLI IL FILE*, il quale aprirà la tipica finestra di file-browsing di sistema, oppure è possibile trascinare semplicemente il file all'interno della finestra dell'applicazione.

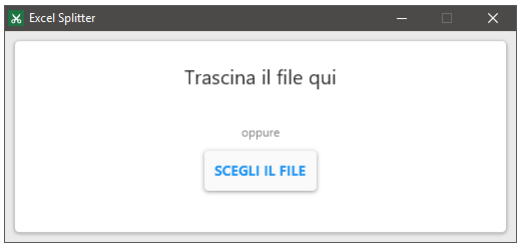

Figura 3.1: Schermata relativa a load.fxml

- **dialog.fxml**: questa schermata rappresenta una finestra di warning, il cui testo può essere impostato in base all'evenienza. Nella figura di seguito, un esempio di schermata di warning, che si presenta quando il file caricato non rispetta le caratteristiche attese.

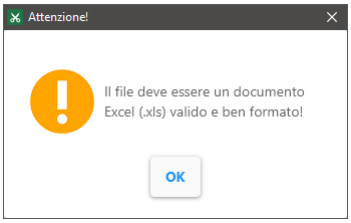

Figura 3.2: Schermata di esempio relativa a dialog.fxml

- **mode.fxml**: questa schermata permette all'utente di scegliere se effettuare una suddivisione semplice, cioè senza creazione di cartelle e/o sottocartelle, oppure se effettuare una suddivisione avanzata.

Figura 3.3: Schermata relativa a mode.fxml

- **help.fxml**: questa schermata acceduta dal pulsante *?* della schermata *mode.fxml* permette all'utente di capire la differenza tra le diverse modalità di suddivisione del documento.

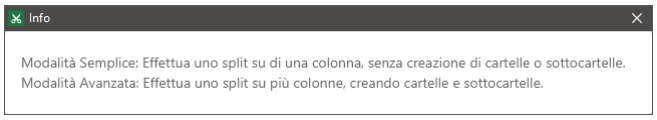

Figura 3.4: Schermata relativa a help.fxml

- **split.fxml**: questa schermata permette all'utente di scegliere, attraverso il menu a tendina, quale colonna designare per la suddivisione del file Excel di partenza.

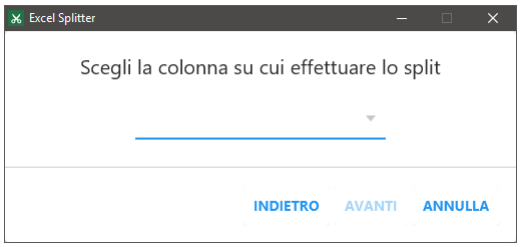

Figura 3.5: Schermata relativa a split.fxml

- **advanced_split.fxml**: questa schermata permette all'utente di scegliere, attraverso i menu a tendina, quali colonne designare per la suddivisione in cartelle, sottocartelle e per lo split.

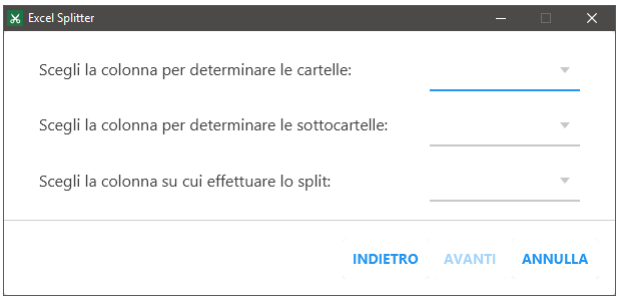

Figura 3.6: Schermata relativa ad advanced-split.fxml

- **export.fxml**: questa schermata permette all'utente di scegliere la cartella di destinazione in cui esportare i file Excel ottenuti dalla suddivisione avvenuta precedentemente. Vi è inoltre la possibilità, attraverso una spunta, di scegliere se creare o meno una cartella apposita (situata comunque nella cartella scelta per l'output) in cui verranno inserite le cartelle ed i file ottenuti dalla suddivisione.

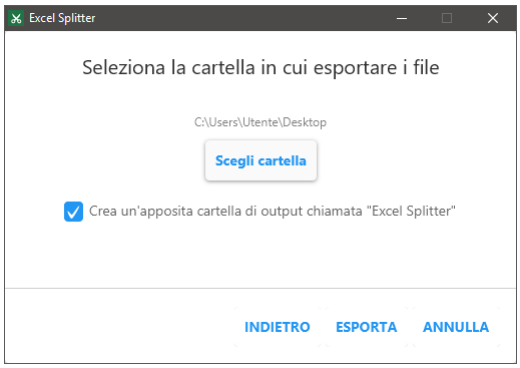

Figura 3.7: Schermata relativa ad export.fxml

- **finish.fxml**: questa schermata avvisa l'utente che l'esportazione dei file è avvenuta con successo e da la possibilità, attraverso una spunta, di scegliere se aprire o meno la cartella di output dopo aver premuto il pulsante *Finito*.

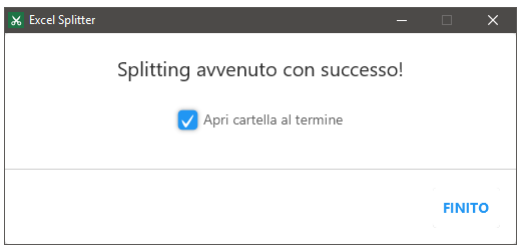

Figura 3.8: Schermata relativa a finish.fxml

3.4 Controller

In questo progetto, i Controller sono rappresentati dalle classi *Controller.java contenute nella cartella *splitter*. Ognuna di queste classi di controllo gestisce la corrispondente interfaccia FXML. Di seguito una trattazione delle azioni svolte da ciascun controller.

3.4.1 ExcelController

Questa classe estende la classe *Controller*, implementando funzionalità comuni alle classi controller presenti nel progetto. In particolare, viene implementato un metodo per la transizione ad altre schermate ed un metodo per il passaggio al controller dell'istanza Spreadsheet contenente il documento Excel caricato nel programma.

ExcelController
- spreadsheet : Spreadsheet - previousFxml : String - nextFxml : String
+ (All getters and setters) + transition(Event event, String fxml) : void + handleBackButton() : void + handleForwardButton() : void + handleCancelButton() : void + errorDialog(Window initOwner, String string) : void

Attributi della classe:

- **spreadsheet**: questo attributo identifica l'istanza della classe Spreadsheet, corrispondente al file XLS importato nel programma.
- **previousFxml**: questo attributo identifica il file di markup FXML corrispondente alla schermata precedente.
- **nextFxml**: questo attributo identifica il file di markup FXML corrispondente alla schermata successiva.

Metodi della classe:

- **transition(Event event, String fxml)**: questo metodo svolge la logica finalizzata alla transizione verso un'altra schermata, a seconda di quale FXML venga specificato nell'argomento del metodo.
- **handleBackButton()**: questo metodo invoca il metodo *transition* passando come argomento la stringa corrispondente al nome del file FXML corrispondente alla schermata precedente.
- **handleForwardButton()**: questo metodo invoca il metodo *transition* passando come argomento la stringa corrispondente al nome del file FXML corrispondente alla schermata successiva.
- **handleCancelButton()**: questo metodo gestisce il pulsante *ANNULLA* e permette all'applicativo di terminare correttamente.

- **errorDialog(Window initOwner, String string)**: questo metodo si occupa di creare una finestra di warning con un messaggio specificato dall'argomento *string*.

3.4.2 LoadController

Questa classe gestisce l'interfaccia espressa da *load.fxml* (fig. 3.1). Estende la classe *ExcelController*.

LoadController
- openButton : Button - progressIndicator : ProgressIndicator
+ handleOpenButtonClick(ActionEvent actionEvent) : void + handleDragOver(DragEvent event) : void + handleDragDropped(DragEvent event) : void - openFileTask(File file, Event event) : void

Attributi della classe:

- **openButton**: questo attributo identifica il pulsante che permette di aprire una finestra del *file explorer* per scegliere il file da caricare nel programma.
- **progressIndicator**: questo attributo identifica il componente di caricamento che mostra all'utente che il caricamento del file è corso.

Metodi della classe:

- **handleOpenButtonClick(ActionEvent actionEvent)**: questo metodo gestisce il pulsante *openButton*. Per il caricamento del file invoca il metodo *openFileTask*.
- **handleDragOver(DragEvent event), handleDragDropped(DragEvent event)**: questi metodi gestiscono l'azione di *drag & drop*. Per il caricamento del file, il metodo *handleDragDropped* invoca *openFileTask*.
- **openFileTask(File file, Event event)**: questo metodo si occupa di aprire un thread, separato dal thread principale della UI, per il caricamento del file *file*. Se non si aprisse il file attraverso un nuovo thread l'interfaccia dell'applicazione si bloccherebbe per un tempo indefinito, tipicamente proporzionale alla dimensione del file da caricare ed, in alcuni casi, potrebbe portare il programma a non rispondere. E' perciò fondamentale in casi come questo di azioni I/O pesanti eseguire un nuovo thread dedicato.

3.4.3 ModeController

Questa classe gestisce l'interfaccia espressa da *mode.fxml* (fig. 3.3). Estende la classe *ExcelController* ed implementa l'interfaccia *Initializable*.

ModeController
+ initialize() : void + handleSimpleButton(ActionEvent actionEvent) : void + handleAdvancedButton(ActionEvent actionEvent) : void + handleHelpButton(ActionEvent actionEvent) : void

Metodi della classe:

- **initialize()**: questo è il metodo che viene importato in seguito all'implementazione dell'interfaccia *Initializable*. Come il nome suggerisce, permette di eseguire un blocco di istruzioni al momento dell'inizializzazione dell'interfaccia FXML associata al controller.
- **handleSimpleButton(ActionEvent actionEvent)**: questo metodo gestisce il pulsante *SEMPLICE*. Quando il pulsante *SEMPLICE* viene premuto, il metodo *handleSimpleButton* invoca il metodo *transition*, ereditato dal padre, e passa come parametro la stringa *split.fxml*, permettendo così di passare alla schermata successiva di splitting semplice.
- **handleAdvancedButton(ActionEvent actionEvent)**: questo metodo gestisce il pulsante *AVANZATO*. Quando il pulsante *AVANZATO* viene premuto, il metodo *handleAdvancedButton* invoca il metodo *transition*, ereditato dal padre, e passa come parametro la stringa *advanced-split.fxml*, permettendo così di passare alla schermata successiva di splitting avanzato.
- **handleHelpButton(ActionEvent actionEvent)**: questo metodo si occupa di creare una finestra di aiuto, la quale fornisce all'utente informazioni relative alla differenza tra la modalità di splitting *SEMPLICE* e quella *AVANZATA*.

3.4.4 SplitController

Questa classe gestisce l'interfaccia espressa da *split.fxml* (fig. 3.5). Estende la classe *ExcelController* ed implementa l'interfaccia *Initializable*.

SplitController
- columnBox : ChoiceBox - forwardButton : Button
+ initialize() : void + setSpreadsheet(Spreadsheet spreadsheet) : void + handleChoiceBox() : void + transition(Event event, String fxml) : void

Attributi della classe:

- **columnBox**: questo attributo identifica il menu a tendina relativo alla scelta di una colonna per la suddivisione del file Excel importato.
- **forwardButton**: questo attributo identifica il pulsante di *AVANTI* che permette di procedere alla schermata successiva.

Metodi della classe:

- **initialize()**: questo è il metodo che viene importato in seguito all'implementazione dell'interfaccia *Initializable*. Come il nome suggerisce, permette di eseguire un blocco di istruzioni al momento dell'inizializzazione dell'interfaccia FXML associata al controller.
- **setSpreadsheet(Spreadsheet spreadsheet)**: questo metodo, ereditato dalla classe padre *ExcelController*, permette di passare al controller l'istanza Spreadsheet che contiene il documento Excel caricato nel programma. Nel caso di questa classe, questo metodo viene esteso aggiungendo alcune funzionalità di inizializzazione che non possono essere eseguite nel metodo *initialize*. Qui è presente infatti un blocco di codice che inizializza la lista *IndexedStringList*, popola le ChoiceBox di conseguenza ed aggiunge a queste delle funzionalità per permettere una selezione disgiunta degli elementi di esse.
- **handleChoiceBox()**: questo metodo viene invocato nel momento in cui viene modificato il valore selezionato nella corrispondente *columnBox*. L'azione che viene compiuta è un controllo finalizzato a determinare se sbloccare o meno il pulsante di *AVANTI*.
- **transition(Event event, String fxml)**: questo metodo, ereditato dalla classe padre, svolge la logica finalizzata alla transizione alla schermata precedente od alla schermata successiva, a seconda di quale FXML venga specificato nell'argomento del metodo. Nel caso in cui l'FXML specificato sia *export.fxml*, al corrispondente controller verrà passata anche la colonna scelta dall'utente attraverso la *ChoiceBox columnBox*.

3.4.5 AdvancedSplitController

Questa classe gestisce l'interfaccia espressa da *advanced-split.fxml* (fig. 3.6). Estende la classe *ExcelController* ed implementa l'interfaccia *Initializable*.

AdvancedSplitController
- directoryBox : ChoiceBox - subdirectoryBox : ChoiceBox - columnBox : ChoiceBox - forwardButton : Button - indexedStringList : List<IndexedString>
+ initialize() : void + setSpreadsheet(Spreadsheet spreadsheet) : void + resetChoiceBox(ChoiceBox choiceBox) : void + handleDirectoryBox() : void + handleSubdirectoryBox() : void + handleColumnBox() : void - checkForwardButton() : void + transition(Event event, String fxml) : void

Attributi della classe:

- **directoryBox**: questo attributo identifica il menu a tendina relativo alla scelta di una colonna per la determinazione delle cartelle.
- **subdirectoryBox**: questo attributo identifica il menu a tendina relativo alla scelta di una colonna per la determinazione delle sottocartelle.
- **columnBox**: questo attributo identifica il menu a tendina relativo alla scelta di una colonna per la suddivisione del file Excel importato.
- **forwardButton**: questo attributo identifica il pulsante di *AVANTI* che permette di procedere alla schermata successiva.
- **indexedStringList**: questo attributo identifica una lista di *IndexedString*, una classe di supporto che rappresenta la composizione di una *String* ed un *Integer*.

Metodi della classe:

- **initialize()**: questo è il metodo che viene importato in seguito all'implementazione dell'interfaccia *Initializable*. Come il nome suggerisce, permette di eseguire un blocco di istruzioni al momento dell'inizializzazione dell'interfaccia FXML associata al controller.
- **setSpreadsheet(Spreadsheet spreadsheet)**: questo metodo, ereditato dalla classe padre *ExcelController*, permette di passare al controller l'istanza Spreadsheet che contiene il documento Excel caricato nel programma. Nel caso di questa classe, questo metodo viene esteso aggiungendo alcune funzionalità di inizializzazione che non possono essere eseguite nel metodo *initialize*. Qui è presente infatti un blocco di codice che inizializza la lista *IndexedStringList*, popola le ChoiceBox di conseguenza ed aggiunge a queste delle funzionalità per permettere una selezione disgiunta degli elementi di esse.

- **resetChoiceBox(ChoiceBox choiceBox)**: questo metodo resetta le ChoiceBox al loro valore di default.
- **handleDirectoryBox(), handleSubdirectoryBox(), handleColumnBox()**: questi metodi vengono invocati nel momento in cui viene modificato il valore selezionato nella corrispondente ChoiceBox. L'azione che viene compiuta è un controllo finalizzato a determinare se sbloccare o meno il pulsante di *AVANTI*, attraverso il metodo *checkForwardButton*.
- **transition(Event event, String fxml)**: questo metodo, ereditato dalla classe padre, svolge la logica finalizzata alla transizione alla schermata precedente od alla schermata successiva, a seconda di quale FXML venga specificato nell'argomento del metodo.

3.4.6 ExportController

Questa classe gestisce l'interfaccia espressa da *export.fxml* (fig. 3.7). Estende la classe *ExcelController* ed implementa l'interfaccia *Initializable*.

ExportController
- backButton : Button - exportButton : Button - cancelButton : Button - chooseDirectoryButton : Button - directoryLabel : Label - progressBar : ProgressBar - checkBox : CheckBox - column : IndexedString - directory : IndexedString - subdirectory : IndexedString - dir : String
+ (All getters and Setters) + initialize() : void + handleChooseDirectory(ActionEvent actionEvent) : void + handleExportButton(ActionEvent actionEvent) : void - formatName(String name) : String

Attributi della classe:

- **backButton**: questo attributo identifica il pulsante *INDIETRO*, che permette di far tornare l'utente alla schermata precedente.
- **exportButton**: questo attributo identifica il pulsante *ESPORTA*, che, una volta premuto, comporterà l'esportazione dei file ottenuti dalla divisione del file originale secondo quanto deciso in precedenza.
- **cancelButton**: questo attributo identifica il pulsante ANNULLA, che permette al programma di terminare correttamente.
- **chooseDirectoryButton**: questo attributo identifica il pulsante *Scegli cartella* che apre il *file explorer* di sistema per la scelta della cartella in cui esportare i file.

- **directoryLabel**: questo attributo identifica la *label* che mostra quale cartella é stata selezionata per l'esportazione.
- **progressBar**: questo attributo identifica la barra di progressione che ha lo scopo di mostrare all'utente la progressione dell'esportazione dei file.
- **checkBox**: questo attributo identifica l'opzione secondo la quale creare o meno una cartella chiamata "Excel Splitter", che contenga tutte le cartelle ed i file esportati, all'interno della cartella selezionata. Per esempio, se questa opzione è selezionata e scelgo di esportare nella cartella *Desktop*, dopo l'esportazione in *Desktop* sarà presente una cartella chiamata *Excel Splitter*, la quale a sua volta conterrà tutti i file e le cartelle risultanti dalla divisione dell'Excel caricato all'inizio.
- **column**: questo attributo identifica la colonna selezionata per lo splitting.
- **directory**: questo attributo identifica la colonna selezionata per la creazione delle cartelle.
- **subdirectory**: questo attributo identifica la colonna selezionata per la crezione delle sottocartelle.
- **dir**: questo attributo identifica la cartella selezionata attraverso il metodo *handleChooseDirectory* invocato premendo il pulsante *Scegli cartella*.

Metodi della classe:

- **initialize()**: questo è il metodo che viene importato in seguito all'implementazione dell'interfaccia *Initializable*. Come il nome suggerisce, permette di eseguire un blocco di istruzioni al momento dell'inizializzazione dell'interfaccia FXML associata al controller.
- **handleChooseDirectory(ActionEvent actionEvent)**: questo metodo gestisce il pulsante *chooseDirectoryButton*. Apre il *file explorer* di sistema per la selezione della cartella di output ed aggiorna la label *directoryLabel* con l'indirizzo assoluto della cartella selezionata.
- **handleExportButton(ActionEvent actionEvent)**: questo metodo gestisce il pulsante *exportButton*. Prima di esportare il file, questo metodo controlla di avere i permessi di scrittura nella cartella selezionata ed in caso negativo lancia una eccezione, che viene gestita mostrando una finestra di warning. Altrimenti il metodo procede, eseguendo un task su un nuovo thread, diverso da quello della UI, nel quale si esportano singolarmente i file risultanti dallo splitting.
- **formatName(String name)**: questo metodo effettua un filtraggio dei nomi dei file da esportare affinché siano compatibili con i diversi sistemi operativi. Per esempio, *Windows* non permette di avere nel nome di file o cartelle i caratteri $<$ o $>$, oppure di avere un nome di file o cartella eccessivamente lungo.

3.4.7 FinishController

Questa classe gestisce l'interfaccia espressa da *finish.fxml* (fig. 3.8). Estende la classe *ExcelController* ed implementa l'interfaccia *Initializable*.

FinishController
- checkBox : CheckBox - directory : String
+ initialize() : void + setDirectory(String directory) : void + handleFinishButton(ActionEvent actionEvent) : void

Attributi della classe:

- **checkBox**: questo attributo identifica l'opzione di aprire, al termine, la cartella in cui sono stati esportati i file.
- **directory**: questo attributo identifica la cartella in cui sono stati esportati i file.

Metodi della classe:

- **initialize()**: questo è il metodo che viene importato in seguito all'implementazione dell'interfaccia *Initializable*. Come il nome suggerisce, permette di eseguire un blocco di istruzioni al momento dell'inizializzazione dell'interfaccia FXML associata al controller.
- **setDirectory(String directory)**: *setter* dell'attributo *directory*.
- **handleFinishButton(ActionEvent actionEvent)**: questo metodo gestisce il pulsante *FINITO*. Il compito di questo metodo è principalmente quello di terminare correttamente l'applicazione; tuttavia, in base alla selezione o meno di *checkBox*, viene aperta o meno la cartella in cui sono stati esportati i file ottenuti dallo splitting dell'XLS originale.

CONCLUSIONE

In conclusione è possibile affermare di aver soddisfatto i requisiti di progetto. Il software realizzato è in grado di suddividere documenti XLS, relativi ai questionari sulla didattica, in base alla colonna attributo scelta, offrendo anche possibilità più avanzate e generali indipendenti dai tipi di informazioni contenute nel documento, a patto che la struttura di questo sia in formato tabulare. L'applicativo è stato comunque progettato per essere esteso. Di seguito, perciò, un elenco delle possibili estensioni future:

- Aggiunta di supporto ad altri formati di file Excel, come XLSX e CSV;
- Aggiunta di ulteriori controlli sulla correttezza della struttura del file importato;

RIFERIMENTI

- http://www.pcmag.com/encyclopedia/term/40495/cross-platform
- https://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione)
- https://it.wikipedia.org/wiki/Model-View-Controller
- http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm
- https://en.wikipedia.org/wiki/Apache_POI
- https://en.wikipedia.org/wiki/Apache_Commons
- https://www.jetbrains.com/idea/
- http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html
- http://guides.beanstalkapp.com/version-control/intro-to-version-control.html