
Politecnico di Torino

Collegio di Ingegneria Informatica, del Cinema e della Meccatronica

**CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA**

# *eeBook*: sviluppo di un Epub Reader in linguaggio Rust con integrazione del sistema OCR

Anno accademico 2021-2022

Sviluppatori: Di Maida Claudio

Luppina Giorgio Daniele

# INDICE

INTRODUZIONE ..... 1

1. INTERFACCIA GRAFICA ..... 3

1.1. Divisione file del progetto ..... 3

1.2. La GUI ..... 3

2. APRERTURA DI UN LIBRO ..... 5

2.1. *Open Book* ..... 5

3. CORRETTORE DI BOZZA ..... 8

3.1. Modifica del testo ..... 8

3.2. Salvataggio del libro ..... 9

4. FUNZIONALITÀ AGGIUNTIVE ..... 10

4.1. Schermata di help ..... 10

4.2. Scorrimento delle pagine ..... 10

4.3. *Go to Page/Chapter* ..... 11

4.4. Ricerca Testuale ..... 12

4.5. Font Size ..... 13

5. RICONOSCIMENTO OCR ..... 14

5.1. *OCR Direct* ..... 14

5.2. *OCR Reverse* ..... 15

INDICE DELLE FIGURE ..... 17

BIBLIOGRAFIA ..... 18

# INTRODUZIONE

La seguente relazione discute del processo di realizzazione di un Epub Reader, passando per i punti salienti del suo sviluppo, usando il linguaggio di programmazione Rust, che hanno portato il software ad avere una corretta *user-experience*. Verranno pertanto descritte e discusse le varie scelte implementative che hanno portato alla realizzazione di un'idea iniziale avuta dal docente Alessandro Savino di Programmazione di Sistema e portata avanti da due studenti universitari.

Lo sviluppo del progetto è andato avanti per diversi obiettivi principali che saranno oggetto di analisi nei successivi capitoli.

Nel primo capitolo verrà brevemente presentata l'interfaccia GUI dell'applicazione con una descrizione generale dei vari strumenti presenti sulla schermata iniziale. L'interfaccia è stata realizzata con il supporto del framework *Druid*[1] di Rust.

Nel corso del secondo capitolo verrà trattata la parte inerente all'apertura di un determinato libro di tipo ".epub" e cosa comporta in termini di scelte implementative. Ne verrà pertanto esposto l'intero processo di funzionamento.

Il terzo capitolo verterà invece sulla parte relativa al "correttore di bozza", un tool per la modifica del libro, precedentemente aperto, in *real-time* andando a ritoccare la relativa sorgente HTML[2]. Verrà, quindi, trattata anche la parte di come avviene il salvataggio del libro modificato.

---

[1] Druid è un framework dichiarativo basato sui dati. È possibile descrivere il modello di applicazione in termini di tratto Dati, quindi creando un albero di *widgets* in grado di visualizzare e modificare tali dati. https://docs.rs/druid/latest/druid/

[2] Un file ".epub" ha una suddivisione interna in capitoli scritti secondo il formato HTML

Nel quarto capitolo si darà spazio a tutte le altre feature implementate al fine di rendere l'applicazione più solida dal punto di vista funzionale restituendo all'utente finale delle meccaniche che favoriscano la fruibilità del software.

Il quinto capitolo affronterà l'integrazione del sistema di riconoscimento del testo OCR per passare, tramite relative immagini, dalla versione cartacea a quella digitale e viceversa. Come motore di riconoscimento del testo è stato scelto *Tesseract*[3] per via della sua natura *open-source*.

Il software *eeBook* è gratuito, eseguibile su PC dotati di sistema operativo Windows® che abbiano istallato Tesseract 4® ed è possibile scaricarlo visitando la pagina GitHub® del progetto[4].

[3] Per maggiori informazioni: https://tesseract-ocr.github.io
[4] Il link della pagina GitHub® di *eeBook* è: https://github.com/giorgioDaniele/eeBOOK.

# 1. INTERFACCIA GRAFICA

## 1.1. Divisione file del progetto

Per una migliore organizzazione delle funzionalità, il progetto *eeBook* consta di un *main* e di ben otto file di libreria:

- chapter.rs
- command.rs
- constants.rs
- ebook.rs
- event.rs
- screen.rs
- search.rs
- toolbar.rs

## 1.2. La GUI

Nel *main* viene generata una nuova finestra, con la funzione *WindowDesc::new(GUI)*, settata inizialmente a schermo intero. La GUI, creata con la libreria *Druid*, parte da un *widget* principale richiamato con *Flex::column()* al quale vengono agganciati due oggetti figli (Figura 1.1):

- La *toolbar*: contiene una colonna ed una riga con al suo interno diciassette figli di tipo *button*, *slider* e *bar text*.
- Lo *screen*: contiene due colonne in cui verranno inserite rispettivamente la copertina e il testo.

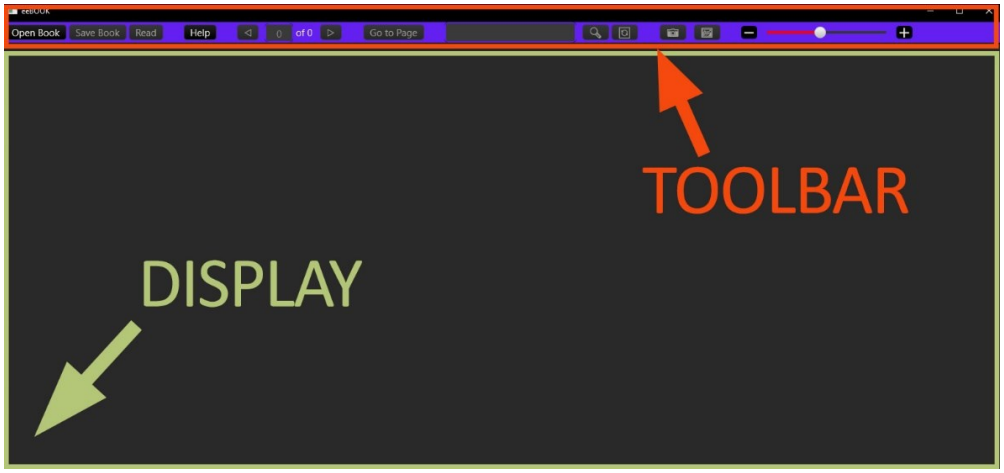


**Figura 1.1. Interfaccia di *eeBook*.**

Successivamente, dopo aver generato tutti i componenti, viene istanziato un *BookState* che corrisponde alla struttura principale. Questa contiene le informazioni sul libro che verrà aperto. Infine, viene lanciata l'applicazione.

Come è possibile distinguere dalla Figura 1.1, nella *toolbar* (in *toolbar.rs*) si hanno:

- *Open Book*: apre un nuovo libro di formato ".epub".
- *Save Button*: salva in una nuova destinazione il libro.
- *Edit/Read*: permette lo *switch* dalla modalità lettura a quella di "correttore bozza" e viceversa.
- *Help*: apre una schermata dove vengono spiegate brevemente le funzionalità, gli scopi e gli autori dell'applicazione.
- ◁, ▷: tasti per scorrere i capitoli o le pagine rispettivamente se si è in modalità *Edit* o in *Read*.
- *Go to Page/Go to Chapter*: tasto per andare direttamente ad una pagina (o capitolo) specificato nel relativo *TextBox*.
- , : tasti per fare una determinata ricerca di una parola o frase all'interno della pagina visualizzata ed eventualmente fare un *refresh* dei risultati.
- *Slider per il font size*: tramite i tasti +, – è possibile ingrandire o rimpicciolire la dimensione del testo.
- , : tasti per passare dalla versione cartacea a quella digitale e viceversa tramite riconoscimento del testo OCR.

# 2. APRERTURA DI UN LIBRO

La prima cosa da fare, dopo aver curato l'interfaccia, è quella di dare la possibilità all'utente di aprire un libro scelto per la lettura. Per fare questo, si descriverà il funzionamento del tasto *Open Book* contenuto nella *toolbar*, di ciò che innesca e di tutti i passaggi si svolgono prima di visualizzare il libro nella schermata principale.

## 2.1. *Open Book*

*Open Book* invoca l'unico metodo implementato in *AppDelegate* (contenuto in *command.rs*) che permette di richiamare il sistema operativo per accedere, in questo caso, al *File Explorer* (Figura 2.1) in cui l'utente seleziona il percorso della risorsa: il libro di formato ".epub".

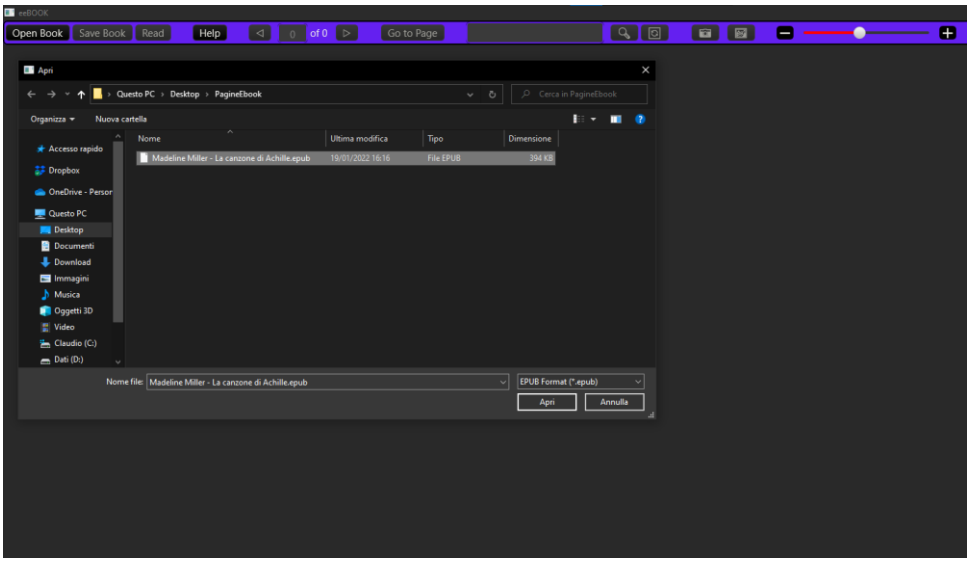


**Figura 2.1. In primo piano il *File Explorer* dedito ad aprire un nuovo libro.**

A questo punto, la risorsa viene caricata tramite il metodo statico *loading()* (in *ebook.rs*) che prende in ingresso un'istanza di *BookState* da valorizzare.

All'interno di *loading()* avviene la costruzione dei capitoli che costituiscono il libro (attraverso la struttura dati *Chapter*). Una volta ricostruiti i capitoli, con il supporto

del *crate epub*[5] avviene il *parsing* dell'HTML (struttura dati contenuta in un *epub*) e della copertina.

Dopodiché avviene il setup del libro tramite la funzione *setup()* (contenuta in *ebook.rs*). In pratica, dato in ingresso il capitolo corrente viene generato un vettore che raccoglie tutte le stringhe che lo compongono con il relativo formato (Titolo, Intestazione o Paragrafo). Questa specifica implementazione è stata pensata per poter adattare il *rendering* del capitolo ad eventi come la ricerca testuale, lo scorrimento delle pagine e l'ingrandimento del testo che comportano la modifica di alcuni parametri del *RichText*[6].

Nel procedere con l'implementazione, si è deciso di mantenere sia lo scorrimento per capitoli (per agevolare la modifica dei capitoli in modalità *Edit*) che lo scorrimento per pagine (per rendere comoda la lettura in modalità *Read*).

Ogni pagina digitale è settata per contenere al massimo 10 elementi tra Titolo, Intestazioni e Paragrafi catturati dall'HTML. La creazione di una pagina digitale consiste nel concatenare le parti che costituiscono un capitolo. Ad esempio, per la pagina 1 si ha: 1 Titolo + 1 Intestazione + 8 Paragrafi; dalla la pagina 2 si ha: 10 Paragrafi.

Per numerare le pagine (e i capitoli) sono state create due mappe:

- La prima, dato il numero (ID) del capitolo, associa il vettore di numero di pagine corrispondenti di quel capitolo;
- La seconda associa la pagina (con il relativo numero) al capitolo al quale appartiene.

In seguito, nella funzione *setup()* viene inizializzato l'HTML da visualizzare in modalità *Edit* e la pagina da visualizzare in modalità *Read.*

[5] Pagina web del *crate*: https://crates.io/crates/epub
[6] Tipo di dato di Druid per contenere testo in formato ".rtf".

Infine, viene settato *ViewState::ReadMode*. La *ViewState* definisce gli stati in cui si trova l'applicazione (nel caso d'esame sono *ReadMode*, *EditMode*, *HelpMode*, *Idle*) modificando a sua volta il display usando l'astrazione *ViewSwitcher*[7] di *Druid*.


**Figura 2.2. Schermata con un libro aperto in modalità *Read*.**

[7] Il *ViewSwitcher* è un widget definito in *Druid* che passa dinamicamente tra due o più figli.

# 3. CORRETTORE DI BOZZA

L'implementazione di un correttore di bozza segue due passi principali:

- Modifica del file HTML relativo al capitolo da correggere;
- Salvataggio del libro modificato.

## 3.1. Modifica del testo

Il tasto *Edit/Read* presente nella *toolbar* permette di passare dalla modalità *Read* a *Edit* (e viceversa) innescando il *ViewSwitcher*.

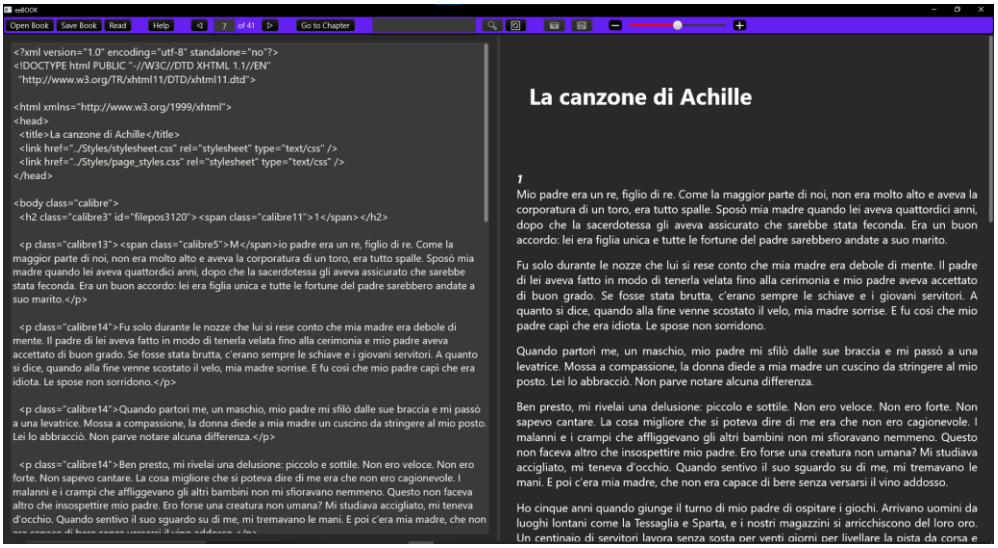


**Figura 3.1. Schermata con un libro aperto in modalità *Edit*.**

Da come è possibile notare dalla Figura 3.1, il *ViewSwitcher* modifica il display in modo che a sinistra della schermata compaia il testo HTML modificabile del capitolo ed a lato il *RichText* corrispondente.

Per vedere gli effetti della modifica del testo in *real time* ciò che viene fatto è aggiungere un controllore di eventi (in *screen.rs*, nella voce riguardante il *widget TextBox* contenente l'HTML) che ritorna un booleano in base al fatto se l'evento coincide o meno alla pressione (o rilascio) di un tasto.

La funzione *reactToHTMLModification()* (in *event.rs*) fa da *wrapper* per *updateHTML()* che aggiorna:

- L'HTML corrente nel vettore di HTML contenuto nel *BookState*;
- Il rendering del testo in formato ".rtf" invocando la *update()*.

## 3.2. Salvataggio del libro

Il salvataggio del libro, per quanto riguarda il sistema operativo, procede allo stesso modo di quanto già visto per l'apertura ovvero invoca l'unico metodo implementato in *AppDelegate* (contenuto in *command.rs*) che permette di chiamare il sistema operativo per accedere al *File Explorer* in cui l'utente seleziona il percorso della risorsa da salvare (il file ".epub").

Si prende a questo punto la risorsa, gli si cambia l'estensione in .zip così che il sistema operativo Windows lo veda come un archivio estraibile all'interno di una cartella. In questo modo è possibile modificare l'archivio inserendo i file HTML modificati. La cartella modificata viene poi compressa e si procede al ripristino dell'estensione ".epub" originaria.

# 4. FUNZIONALITÀ AGGIUNTIVE

In questo capitolo verranno descritte tutte le funzionalità aggiuntive rispetto a quelle base di cui si è già discusso in precedenza.

## 4.1. Schermata di help

Il *button* Help modifica il valore del *ViewSwitcher* settandolo nella modalità *Help*. In questo modo viene visualizzata una schermata dove vengono descritte le funzionalità dell'applicazione (Figura 4.1).

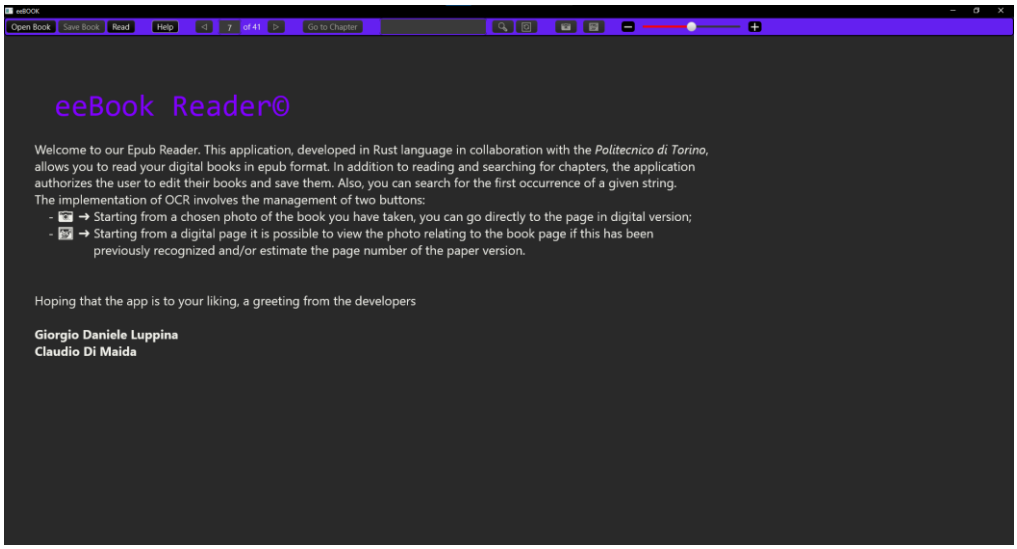


**Figura 4.1. Schermata della modalità *Help*.**

Tramite il tratto *Lens* di *Druid* si accede ad un campo del *BookState* in cui viene inizializzato il *RitchText* che descrive le funzioni dell'applicazione.

## 4.2. Scorrimento delle pagine

I pulsanti [◁], [▷] servono per scorrere i capitoli quando si è in modalità *Edit* o le pagine in modalità *Read*. Entrambi invocano una funzione *scroll()* (in *ebook.rs*)

che, in funzione della direzione e della modalità di visualizzazione, salvano eventualmente l'HTML modificato e aggiornano la vista con la pagina successiva (o precedente) richiamando la funzione *update()*. Questa funzione:

- In modalità *Read*: recupera dal *BookState* la pagina da visualizzare con le sue opzioni di formattazione e modifica la pagina .rtf corrente[8]. Recupera poi anche il capitolo corrispondente alla pagina utilizzando la mappa pageBelongs (che restituisce l'indice del capitolo) e precalcola il rendering. Infine, recupera anche l'HTML del capitolo corrente.
- In modalità *Edit*: anziché iterare sulle pagine, itera sui capitoli. Recupera dal *BookState*: il *plaintext* del capitolo (e lo renderizza), l'HTML e la prima pagina del capitolo.

Infine, il metodo *update()* aggiorna una variabile "*cursor*" che contiene l'informazione su quale sia la pagina (o il capitolo) corrente in formato stringa così da essere visualizzata in un'apposita *TextBox* (Figura 4.2) in mezzo ai due pulsanti.

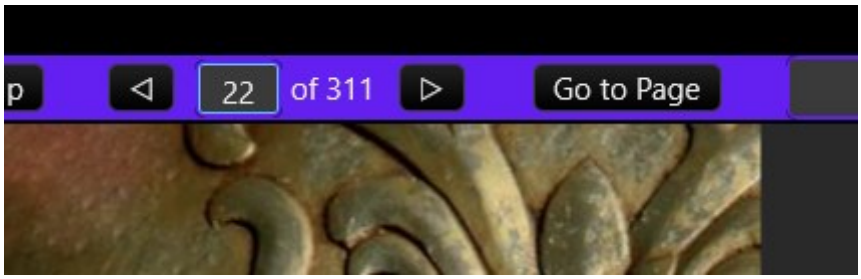


**Figura 4.2. Particolare della *TextBox* dove è contenuto il numero di pagina.**

## 4.3. *Go to Page/Chapter*

L'utente può specificare la pagina (in modalità *Read*) o il capitolo (in modalità *Edit*) da visualizzare cambiando valore al TextBox, oggetto della Figura 4.2, e premendo il relativo pulsante *Go to Page/Chapter*.

[8] La variabile *bookState.currentRTFpage* è contenuta dentro una struttura dati che implementa il tratto *Lens* che permette dall'esterno di accedere ad uno o più dei suoi campi. In tal modo il *widget* che espone il metodo *Lens* accede alla variabile in modo dinamico mostrandone in output i suoi cambiamenti: in questo caso, il testo all'interno di un etichetta *RawLabel*.

Il pulsante fa da *wrapper* per il metodo statico *jump()* (in *ebook.rs*). Questo metodo, una volta verificato che l'input sia un numero valido, esegue la funzione *update()* sulla pagina o sul capitolo.

## 4.4. Ricerca Testuale

La ricerca testuale è formata da una *searchBar* e da due *button*:

- *SearchButton* [🔍]
- *refreshSearcButton* [🔄]

La *searchBar* è un front-end per le espressioni regolari[9] (*Regex*). La *SearchButton* distingue dapprima i casi in cui si trova in modalità *Edit* o modalità *Read* facendo rispettivamente una ricerca nel capitolo o nella pagina corrente.

La funzione *looksForMatchesInPage()* (contenuta in *search.rs*) recupera dal *BookState* il vettore di stringhe (10 stringhe per pagina in accordo con quanto specificato nel capitolo 1.2) riguardante la pagina corrente ed applica l'espressione regolare data in input a ciascuna di queste stringhe.


**Figura 4.3. Risultato di una ricerca di una parola.**

[9] Un'espressione regolare è una sequenza di simboli che identifica un insieme di stringhe.

Per ogni stringa possono esserci più *match*; per questo motivo vengono registrati tutti gli intervalli possibili dei *match* tramite la funzione *captures_iter()*. Tali intervalli servono per rigenerare la pagina in formato ".rtf" con i *match* evidenziati tramite la funzione *reactToSearch()*. Questa, per ogni *match*, attribuisce un colore differente alla porzione di testo da visualizzare (Figura 4.3).


**Figura 4.4. Risultato di una successiva ricerca con memoria della precedente.**

Inoltre, la feature prevede di avere memoria dei precedenti risultati che continuano a rimanere evidenziati nelle ricerche successive (Figura 4.4).

La *refreshSearchButton()* è un *wrapper* alla funzione *update()* che ristampa la pagina su schermo senza le evidenziazioni del testo.

## 4.5. Font Size

La feature di modifica della dimensione del font è costituita da una slide e da due bottoni [+], [-]. I bottoni o la slide eseguono entrambi i seguenti passi:

1. Prendono in input la dimensione corrente del testo;
2. Modificano il valore di una quantità prestabilita;

Una volta fatto ciò, tramite il controller su *ZoomEvent*, il widget del display viene aggiornato se la dimensione del font è cambiata.

# 5. RICONOSCIMENTO OCR

In quest'ultimo capitolo verranno discusse le implementazioni riguardanti il riconoscimento del testo. L'obiettivo è integrare il sistema di fotocamera con OCR per collegare la versione cartacea del libro con quella digitale nei seguenti modi:

- *OCR Direct*: nella versione digitale saltare al punto in cui si è arrivati nella versione cartacea caricando una foto di una pagina del libro.
- *OCR Reverse*: da una o più pagine cartacee riconosciute indicare a che pagina si trova il testo della versione digitale rispetto a quella cartacea.

## 5.1. *OCR Direct*

All'inizio la procedura di apertura di una foto (o immagine) segue lo stesso *flow* dell'apertura del libro. Viene quindi sfruttato *AppDelegate*, solamente che, in questo caso, vengono aperte delle immagini (Figura 5.1).

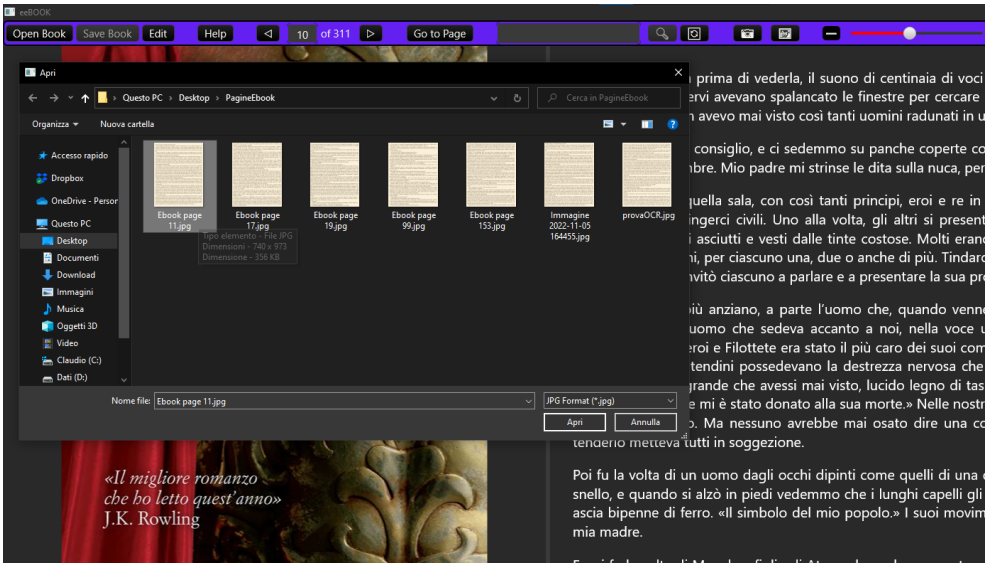


**Figura 5.1. Apertura di una foto da ricercare nel libro digitale.**

L'apertura dell'immagine è seguita dal *Command* che salva in una variabile il *path* dell'immagine ed esegue la funzione *loading()* (in *ebook.rs*) che fa da *wrapper* per la funzione di ricerca del testo OCR.

La ricerca OCR inizia specificando la lingua da utilizzare per il riconoscimento. Questa la si ricava dalle informazioni contenute all'interno del file ".epub". Segue l'esecuzione di un processo OCR da linea di comando che salva in un file temporaneo il risultato dell'analisi testuale. Questo file viene letto e salvato in una stringa *tessOutput* e successivamente cancellato.

La stringa *tessOutput* va in ingresso alla funzione *directOCR()* (contenuta in *search.rs*) che recupera il vettore contente il testo di tutte le pagine ed inizializza un vettore di threads. Viene impiegata una tupla comune *bestHit* protetta da *mutex*.

Il funzionamento è il seguente: ogni thread lavora su un capitolo specifico del libro (un sottoinsieme di pagine) e verifica sulle sue pagine la presenza o meno delle stringhe acquisite dall'immagine: viene contato, per ogni pagina, la quantità dei *match* modificando il valore di *bestHit* solo se maggiore di quello attuale. La tupla *bestHit* contiene il numero di hit e la pagina alla quale questi sono stati ottenuti.

I thread vengono attesi dal *main* e l'algoritmo riconosce banalmente la pagina dove sono state ottenute più corrispondenze e salta ad essa tramite il metodo *jump()* che fa semplicemente da *wrapper* per la funzione *update()*.

## 5.2. *OCR Reverse*

I vari *path* delle immagini precedentemente acquisite costituiscono un piccolo database per questo punto.

Per stimare il numero di pagina della versione cartacea, all'inizio vengono aperte tutte le immagini in database, ne viene acquisito il testo tramite la funzione *reverseOCR()*. Con questa funzione ogni thread lavora su un'immagine in cui estrae il testo confrontandolo con la pagina digitale corrente. Se testo digitale trova una completa corrispondenza con una delle immagini in database, questa viene eletta

come equivalente cartacea con lo stesso principio del *bestHit* e il relativo *path* costituisce l'output della funzione. Quest'immagine viene aperta e proposta in una finestra *Druid* distinta da quella principale (Figura 5.2).

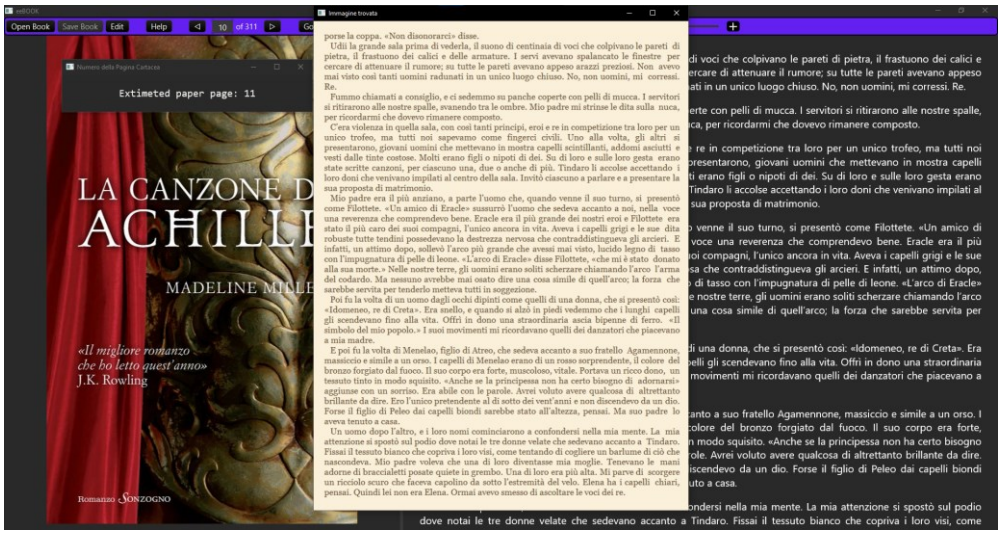


**Figura 5.2. Risultato dell'operazione di stima della pagina cartacea.**

In ogni caso, anche se l'immagine cartacea corrispondente non si trova in database, viene comunque mostrata una finestra *Druid* che propone una stima del numero di pagina reale del libro secondo un algoritmo che è possibile riassumere in due punti:

1. Ad ogni fotografia caricata all'interno di *eeBook* viene aggiornata, in *directOCR()*, la variabile *paperWordsEvaluetion* contenuta nel *BookState*. Banalmente essa viene inizializzata al *count* delle parole della prima immagine e viene modificata con una media aritmetica per i *count* delle parole delle immagini successive.
2. Ad ogni innesco della funzione *reverseOCR()*, vengono prese tutte le pagine digitali precedenti a quella di cui stimare il numero cartaceo e viene fatto un *count* totale delle parole per capitolo. Al *count* relativo al capitolo viene ricavato, dividendo per *paperWordsEvaluetion*, il numero di pagine stimate. Se c'è del resto nella divisione per *paperWordsEvaluetion*, al capitolo viene stimata una pagina in più. Iterando questo procedimento per tutti i capitoli fino alla pagina in esame, si ricava la stima del numero di pagina che viene settato nella variabile *digitalPageEvaluetion* del *BookState* e mostrato a schermo in una finestra apposita.

# INDICE DELLE FIGURE

Figura 1.1. Interfaccia di *eeBook*. .....4
Figura 2.1. In primo piano il *File Explorer* dedito ad aprire un nuovo libro. .....5
Figura 2.2. Schermata con un libro aperto in modalità *Read*. .....7
Figura 3.1. Schermata con un libro aperto in modalità *Edit*. .....8
Figura 4.1. Schermata della modalità *Help*.....10
Figura 4.2. Particolare della *TextBox* dove è contenuto il numero di pagina. .....11
Figura 4.3. Risultato di una ricerca di una parola. .....12
Figura 4.4. Risultato di una successiva ricerca con memoria della precedente. ....13
Figura 5.1. Apertura di una foto da ricercare nel libro digitale. .....14
Figura 5.2. Risultato dell'operazione di stima della pagina cartacea. .....16

# BIBLIOGRAFIA

AA.VV., Codice sorgente di *eeBook* disponibile sulla pagina GitHub del progetto: https://github.com/giorgioDaniele/eeBOOK

AA.VV., *Crate "epub" latest documentation* a disposizione sul sito internet: https://crates.io/crates/epub

AA.VV., *Druid latest documentation* a disposizione sul sito internet: https://docs.rs/druid/latest/druid/

AA.VV., *Tesseract OCR documentation* a disposizione sul sito internet: https://tesseract-ocr.github.io

S. Klabnik, C. Nichols, *The Rust Programming Language* disponibile sul sito: https://doc.rust-lang.org/book/