

**Università degli Studi di Bergamo**

**Dipartimento di Ingegneria Gestionale, dell’Informazione e della trasmissione**

**VolText**

**Manuale utente**

**Colombi Simone**

**Gambarara Alberto**

**Scarpellini Stefano**

**Indice**

[1. Introduzione 3](#_Toc71294394)

[2. Grammatica 4](#_Toc71294395)

[2.1. Descrizione generale 4](#_Toc71294396)

[2.2. Elementi di una pagina 7](#_Toc71294397)

[2.2.1. Immagini 7](#_Toc71294398)

[2.2.2. Testi 8](#_Toc71294399)

[2.2.3. Liste 9](#_Toc71294400)

[2.2.4. Div 10](#_Toc71294401)

[2.3. Scelte grammaticali 11](#_Toc71294402)

[2.4. Errori della grammatica 11](#_Toc71294403)

[3. Tool utilizzati 12](#_Toc71294404)

[4. Interfaccia utente 13](#_Toc71294405)

[4.1. Caso d’uso generale 13](#_Toc71294406)

[4.2. Reliability 15](#_Toc71294407)

[5. Possibili sviluppi futuri 17](#_Toc71294408)

# Introduzione

VolText è un progetto sviluppato nell’ambito del corso di Linguaggi Formali e Compilatori, tenuto dal professor Giuseppe Psaila nell’Anno Accademico 2020/2021.

L’obiettivo è la progettazione e implementazione di una libreria in Java per la generazione di un documento PDF e di una interfaccia grafica che permetta l’utilizzo semplificato di essa. La parte più inerente al corso è stata la realizzazione di una grammatica che descrive la struttura di un documento PDF usando i costrutti tipici della sintassi HTML, quali div, immagini, testi e liste, ma sfruttando una sintassi simile a quella di JSON.

La libreria è incentrata in particolare sullo sviluppo di documenti che abbiano una grafica semplice come volantini, piccoli annunci, avvisi, biglietti da visita, etc.

Il tool non vuole e non funge da sostituto di LaTeX ma prende gli elementi essenziali, creando una versione diversa sia in termini di linguaggio sia in termini di componenti, eliminando formule, grafici, tabelle, etc. non utili allo sviluppo di documenti grafici.

1. **Grammatica**

## 2.1. Descrizione generale

La grammatica è stata progettata e implementata pensando a chi lavora con linguaggi web e che è familiare ad HTML/JSON.

Sfruttando la tipica sintassi del linguaggio JSON per la costruzione di tag personalizzati si è strutturata la grammatica come un documento PDF composto da vari elementi che possono essere disegnati all’interno in posizioni e dimensioni a piacimento.

Si riporta la grammatica e, per ogni regola, se ne descrive la funzione.

**grammar** VolText;

@header {

**package** antlr;

}

pdf : *A* pdfattr\* (stylesheet)? (page)+ *C*;

pdfattr: 'title:' *STRING* *ENDNLINE*

| 'author:' *STRING* *ENDNLINE*

| 'path:' *STRING* *ENDNLINE*;

stylesheet: 'stylesheet' *O* element\* *C*;

element: '@' *STRING* *O* attrStyle\* *C*;

attrStyle: 'cross-point:' *NVAL* *ENDNLINE*

| 'shape:"' ('RECTANGLE' | 'CIRCLE' | 'TRIANGLE') *ENDLINE*

| ('fit-x' | 'fit-y') ':' *TFVAL* *ENDNLINE*

| ('pos-x' | 'pos-y') ':' *NOTVAL*? *NVAL* (*UNIT*)? *ENDNLINE*

| 'angle-rotation' ':' *NOTVAL*? *NVAL* *ENDNLINE*

| ('height' | 'width') ':' *NVAL* (*UNIT*)? *ENDNLINE*

| ('p\_height' | 'p\_width') ':' *NVAL* *ENDNLINE*

| 'ordered:' *TFVAL* *ENDNLINE*

| 'bullet:' *STRING* *ENDNLINE*

| ('font-family:' | 'font-family-ttf:' | 'font-family-otf:') *STRING*

*ENDNLINE*

| 'font-size:' *NVAL* *ENDNLINE*

| ('bold:' | 'italics:' |'underline:') *TFVAL* *ENDNLINE*

| 'colorT-bullet:' *STRING* *ENDNLINE*

| 'color-bullet:' *COLORVAL* *ENDNLINE*

| 'colorT:' *STRING* *ENDNLINE*

| 'color:' *COLORVAL* *ENDNLINE*

| 'position:' *POSVAL* *ENDNLINE*

| 'alignment:' *ALIGNVAL* *ENDLINE*

| 'orientation:' *ORIENTATION* *ENDNLINE*

| 'oob:' *TFVAL* *ENDNLINE*

| 'format:' *FORMATVAL* *ENDNLINE*;

page: 'page' *O* pageattr\* pae\* *C*;

pae: elemd | div;

div: 'div' *O* (color | idval | imganumber | elemd | positionv | fitAttr |

figure | tvalue)\* *C*;

tvalue: 'cross-point:' *NVAL* *ENDNLINE*;

figure: 'shape:"' ('RECTANGLE' | 'CIRCLE' | 'TRIANGLE') *ENDLINE*;

elemd: text

| list

| img;

img: 'img' *O* imgattr\* imgElem imgattr\* *C*;

imgattr: (idval | imganumber | positionv | fitAttr);

imgElem: 'URL:' *STRING* *ENDNLINE*;

list: 'list' *O* (listattr | listElem)\* *C*;

listElem: 'item:' *STRING* *ENDNLINE*;

text: 'text' *O* txtattr\* txtElem\* txtattr\* txtElem\* txtattr\* *C*;

txtattr: (color | idval | imganumber | positionv | alignment | txtval |

fitAttr);

txtElem: 'string:' *STRING* *ENDNLINE*;

//ATTRIBUTES

fitAttr: ( 'fit-x'

| 'fit-y') ':' *TFVAL* *ENDNLINE*;

imganumber: ( 'pos-x'

| 'pos-y' ) ':' *NOTVAL*? *NVAL* (*UNIT*)? *ENDNLINE*

| 'angle-rotation' ':' *NOTVAL*? *NVAL* *ENDNLINE*

| ('height'

| 'width') ':' *NVAL* (*UNIT*)? *ENDNLINE*;

idval: 'id:' *STRING* *ENDNLINE*;

listattr: 'ordered:' *TFVAL* *ENDNLINE*

| 'bullet:' *STRING* *ENDNLINE*

| idval

| imganumber

| txtval

| positionv

| color

| colorBullet

| fitAttr;

txtval: ('font-family:'

| 'font-family-ttf:'

| 'font-family-otf:') *STRING* *ENDNLINE*

| 'font-size:' *NVAL* *ENDNLINE*

| ('bold:'

| 'italics:'

| 'underline:') *TFVAL* *ENDNLINE*;

colorBullet: 'colorT-bullet:' *STRING* *ENDNLINE*

| 'color-bullet:' *COLORVAL* *ENDNLINE*;

color: 'colorT:' *STRING* *ENDNLINE*

| 'color:' *COLORVAL* *ENDNLINE*;

positionv: 'position:' *POSVAL* *ENDNLINE*;

alignment: 'alignment:' *ALIGNVAL* *ENDLINE*;

pageattr: 'orientation:' *ORIENTATION* *ENDNLINE*

| 'oob:' *TFVAL* *ENDNLINE*

| 'p\_width' ':' *NVAL* *ENDNLINE*

| 'p\_height' ':' *NVAL* *ENDNLINE*

| 'format:' *FORMATVAL* *ENDNLINE*

| idval;

//TERMINALI

*NOTVAL*: '-';

*UNIT*: 'mm' | '%' | 'pt';

*FORMATVAL*: 'A0'

| 'A1'

| 'A2'

| 'A3'

| 'A4'

| 'A5'

| 'A6';

*ORIENTATION*: ('hor' | 'ver');

*COLORVAL*: '#' ([0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]

[0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]);

*ALIGNVAL*: 'left'

| 'center'

| 'right'

| 'justify';

*TFVAL*: 'true'

| 'false';

*POSVAL*: 'lt'

| 'ct'

| 'rt'

| 'lc'

| 'cc'

| 'rc'

| 'lb'

| 'cb'

| 'rb' ;

*NVAL*: [0-9]+('.'[0-9]+)?;

*ENDLINE*: '";';

*ENDNLINE*: ';';

*O* : ':{';

*A* : '{';

*C* : '}';

*STRING*: '"' (~[\t\r\n])+ '"';

*WS* : [ \t\r\n]+ **->** skip ; // skip spaces, tabs, newlines

La prima regola, *pdf*, permette l’apertura del documento, l’inserimento di attributi quali titolo, autore e percorso di salvataggio del pdf, relativi al documento, la possibilità di inserire una parte di stile del documento, una o più pagine e la chiusura del documento.

Una struttura sommaria del tag *stylesheet* è stata realizzata come

stylesheet:{

@IdElemento1:{

<< attributi relativi a quell’elemento>>

}

}

La pagina può contenere vari attributi quali orientamento, oob (out-of-bound), larghezza, altezza e formato, come specificato nella seguente tabella:

|  |  |
| --- | --- |
| **attributo** | **valori possibili** |
| orientation | hor, ver |
| oob | true,false |
| p\_width | numero decimale |
| p\_height | numero decimale |
| format | A0, A1, A2, A3, A4, A5, A6 |

Ogni pagina contiene al suo interno o dei div, quindi dei blocchi che possono contenere a loro volta altri elementi, oppure elementi stessi quali immagini, liste e testi. Questi elementi e i loro attributi verranno trattati nel capitolo 2.2.

Una pagina, come anche segnalato nella specifica ISO 32000-1, sezione 11.4.7, deve avere colore di fondo bianco. Quindi se si vuole colorare completamente una pagina è possibile inserire un div con dimensioni fittate rispetto alla pagina e inserirci altri elementi (in questa versione, data la scelta di non avere contenitori interni ad altri, non è possibile crearne diversi in una pagina con colore di fondo diverso dal bianco, appunto perché è già esso un contenitore colorato).

## 2.2. Elementi di una pagina

Trattiamo ora i vari elementi che possono comporre una pagina. Tratteremo per prime le immagini, i testi e le liste che possono essere inserite riferendosi alle dimensioni della pagina. Per ultimi tratteremo i div, elementi più complessi e che possono contenere al loro interno gli altri tipi di elemento ma con riferimenti di misura legati al div nel quale sono incapsulati. Ogni attributo di ogni elemento deve sempre essere chiuso da un ; mentre la chiusura degli elementi è realizzata tramite una graffa chiusa.

### 2.2.1. Immagini

Un’immagine deve essere inserita attraverso il tag *“img:{ <<attributi>> }”*.

Essa può contenere attributi messi in posizione anche successiva all’unico attributo obbligatorio, ovvero l’URL. L’*url* è una stringa che specifica il percorso relativo o assoluto di quella immagine all’interno del computer. Se relativo, il riferimento partirà dalla cartella dove si trova il documento testuale e, come vedremo in seguito, se non viene aperto un documento ma solo incollato il testo all’interno della GUI avremo degli errori riguardo la mancanza dell’oggetto all'indirizzo specificato.

Si elencano ora i vari attributi disponibili per le immagini:

* *id*, ovvero l’identificativo dell’elemento all’interno del documento. Questo è stato pensato per l’uso soprattutto nel foglio di stile e per la segnalazione degli errori relativi al posizionamento e dimensionamento degli oggetti.
* *pos-x* e *pos-y*, indicano la posizione dell’elemento sull’asse orizzontale e verticale. Il valore può essere positivo o negativo e contenere anche l’unità di misura (millimetri (mm), percentuale (%) o punti (pt), di default mm).
* *angle-rotation*, indica la rotazione dell’elemento rispetto al suo centro in gradi. Se il valore specificato è positivo verrà effettuata una rotazione in senso antiorario, al contrario con un valore negativo verrà svolta una rotazione in senso orario.
* *height* e *width*, indicano l’altezza e la larghezza dell’immagine nel caso serva ridimensionare l’elemento all’interno della pagina o del div. Anch’esse possono essere espresse in millimetri, percentuale o punti.
* *position*, contiene una coppia di lettere che permettono di inserire l’elemento nel suo contenitore (pagina o div) in una posizione prefissata (immaginando una suddivisione in 9 quadrati del contenitore). Essi possono quindi essere *l,c* e *r* per segnalare se l’elemento deve stare rispettivamente a sinistra, centrato o a destra nel contenitore, e *t,c* o *b* per segnalare se l’elemento deve essere in alto (top, *t*), centrale (*c*) o in basso (bottom, *b*).
* *fit-x* e *fit-y*, contengono un valore booleano (true, false) per segnalare se l’elemento deve essere ridimensionato sull’intera dimensione del contenitore (sia esso la pagina o il div) rispetto all’asse x o y.

### 2.2.2. Testi

Un testo può essere inserito tramite il tag “*text:{ <<attributi>> }”*.

Esso può contenere attributi misti a stringhe di testo senza vincoli di precedenza.

Le stringhe/capitoli sono contenute/i in attributi *“string”* e vengono scritti nel PDF nell’ordine che si è usato nella scrittura dell’elemento *text*.

Gli attributi dell’elemento *text* sono i seguenti:

* *id*, come già descritto per le immagini, identifica l’oggetto.
* *colorT* e *color*, attributi che segnalano il colore del testo (univoco per tutti le stringhe). *color* specifica un colore in formato *#RGBA* esadecimale, dove R è il colore rosso, G il colore verde, B il colore blu e A l’alpha, ovvero il valore di trasparenza dell’elemento. Nei testi e nelle liste l’alpha non è considerato, al contrario di ciò che accade per i div. *colorT* specifica un colore sotto forma di testo (black, blue, red, green, …). L’ultimo attributo che viene dichiarato verrà usato nella redazione del documento.
* *pos-x, pos-y, angle-rotation, height, width, position, fit-x* e *fit-y*: si veda la trattazione nel capitolo 2.2.1 sulle immagini.
* *alignment*, utilizzato per l’allineamento del testo (left per sinistra, center per centrato, right per destra e justify per giustificato)
* *font-family, font-family-ttf* e *font-family-otf*, specificano il font da utilizzare nella redazione del documento. Se si utilizza font-family si possono utilizzare i font standard quali helvetica, courier e times. Se si utilizza font-family-ttf, allora si può specificare il path dove è disponibile il file con estensione ttf (True Type Font) che descrive il font da usare. Se la cartella dove risiede il font contiene anche il relativo font in grassetto, corsivo e/o grassetto corsivo allora il path da specificare sarà quello del font regolare (terminante con Regular.ttf, e in cui gli altri file termineranno con Bold.ttf, Italic.ttf e BoldItalic.ttf, rispettivamente per grassetto, corsivo e grassetto corsivo): in questo caso verranno elaborati correttamente anche eventuali testi o parti di testo in grassetto (scritte come *\bold testo in bold \bold*), corsivo (scritte come *\italic testo in corsivo \italic*) o grassetto corsivo (scritte come *\bold\italic testo in bold italic \italic \bold*). Le stesse regole valgono per l’attributo font-family-otf, a cui si applicano font con estensione otf (Open Type Font)
* *font-size*, contiene la dimensione del carattere da usare nei testi.
* *bold, italic* e *underline*, sono attributi booleani che permettono di specificare se l’intero testo debba essere scritto in grassetto, in italico o sottolineato (quest’ultima specifica è stata implementata ma non conclusa per difficoltà di integrazione della libreria PDFBox-layout con PDFBox).

### 2.2.3. Liste

Una lista può essere inserita attraverso il tag *list:{ <<attributi>> }*

Essa può contenere attributi misti a stringhe di testo (voci della lista, *item*) senza vincoli di precedenza.

Gli attributi di una lista possono essere:

* *id*, come i precedenti elementi
* *ordered*, che serve per segnalare se è una lista ordinata di elementi (lista numerica) oppure se è una lista non ordinata (lista a punti)
* *bullet*, specifica il tipo di “punto” che fa da inizio dell’elemento della lista. I possibili punti sono *odd* (pallino pieno, ), *even* (doppia freccia, ), oppure *new(Bullet)* (in cui il *Bullet* è una stringa decisa dall’utente e usabile come punto).
* tutti gli attributi relativi ai testi, tranne l’allineamento, sono implementati anche per le liste
* *colorT-bullet* e *color-bullet* , attributi che segnalano il colore del bullet (univoco per tutti i bullet). *color-bullet* specifica un colore in formato *#RGBA* esadecimale, dove R è il colore rosso, G il colore verde, B il colore blu e A l’alpha, ovvero il valore di trasparenza dell’elemento. Il valore di alpha non è considerato. *colorT-bullet* specifica un colore sotto forma di testo (black, blue, red, green, …). L’ultimo attributo che viene dichiarato verrà usato nella redazione del documento.

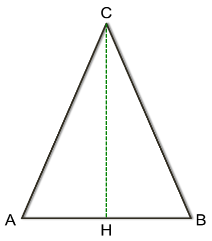
### 2.2.4. Div

Un div può essere inserito attraverso il tag *div:{ <<attributi e/o elementi>>}*.

Il div, come descritto anche in html, è un contenitore che permette di raggruppare elementi al suo interno. In questa versione del progetto un div può contenere solo immagini, testi e liste ma non altri contenitori. Il contenitore può essere personalizzato attraverso vari attributi e gli elementi che sono al suo interno prendono come riferimento dimensionale la dimensione del contenitore.

Esso può contenere attributi misti a elementi (*img*, *text* e *list*) senza vincoli di precedenza.

Si analizzano ora i vari attributi di un contenitore:

* *id,* come ogni elemento, specifica un identificativo che permette di riconoscere l’elemento.
* *color* e *colorT*, come già presentato precedentemente per quanto riguarda i testi e le liste, specifica il colore di riempimento del contenitore. Non è stata realizzata una suddivisione in colore di riempimento e colore del bordo, che potrà essere inserita in futuri sviluppi. *color* specifica il colore in formato *#RGBA* e in questo caso l’alpha (A) specifica la trasparenza del contenitore, per fornire la possibilità di vedere altri elementi sottostanti ad esso. *colorT* invece permette l’uso di colori attraverso l’uso di nomi di colori.
* *pos-x*, *pos-y*, *angle-rotation*, *height*, *width*, *fit-x* e *fit-y*, come già detto nelle precedenti sezioni, specificano la posizione, l’angolo di rotazione, le dimensioni e l’adattamento rispetto all’asse orizzontale e verticale.
* *position*, specifica la posizione del contenitore rispetto alla pagina nel quale viene inserito senza bisogno di specificare le coordinate spaziali
* *shape*, specifica la forma del contenitore, tra *RECTANGLE*, *CIRCLE* e *TRIANGLE*. Nel caso del rettangolo verranno usati i valori di posizione e dimensione per il disegno di esso. Nel caso del cerchio verranno utilizzati i valori di posizione e dimensione per disegnare un cerchio o una ellisse. Nel caso del triangolo oltre a posizione e dimensione sarà necessario l’inserimento di un attributo *cross-point* espresso come percentuale che specifica il punto H specificato in figura:
  1. **Scelte grammaticali**

La principale scelta è dovuta alla grammatica elaborata per implementare il foglio di stile. Si tratta di una scelta legata alla comodità di implementare gli attributi direttamente come figli di *attrStyle*, rispetto a implementarli come figli degli attributi degli elementi presenti nelle pagine.

Per come la grammatica legge il foglio, la posizione dello *stylesheet* come seconda parte del non terminale *pdf* non permette di avere una idea dell’elemento al quale le specifiche si riferiscono, dato che l’id presente nel foglio di stile fa riferimento a un oggetto generico, e quindi non verifica la correttezza dei singoli attributi. Non avendo controlli di alcun tipo sullo stylesheet, è possibile l’utilizzo di un id per più elementi di natura diversa e di poter inserire attributi anche se non appartenenti al tipo di oggetto che si vuole personalizzare.

* 1. **Errori della grammatica**

Trattiamo ora i vari errori e le varie situazioni che possono capitare nella redazione di un documento di testo e che il programma rileva e interpreta. In nessun caso il programma segnala un errore ma mostra solo avvisi per segnalare un errore di battitura o di tipo lessicale.

Il primo caso di errore (che verrà anche trattato come esempio di errore nel capitolo 4) è il caso di una dimenticanza di un punto e virgola (o una parentesi graffa). In questo caso viene riportato all’utente che in una specifica linea/posizione è mancante un simbolo ma il sistema riesce comunque a proseguire nella redazione. Questo è il tipico caso di un errore lessicale.

Lo stesso vale anche se un attributo non appartiene a un elemento (ad esempio un attributo *bullet* per il tag *img*). Questo è il tipico caso di errore sintattico.

Una segnalazione che il programma restituisce è anche riferita, identificandosi con l’*url* dell’immagine, all’errato percorso dove si trova l’immagine. Infatti, il sistema specifica che non è riuscito a leggere l’immagine in quello specifico path. Il sistema però evita il caricamento dell’immagine e prosegue con la continuazione dell’analisi del testo.

Lo stesso “errore” viene riportato se il path specificato non è relativo a un file per l’uso di font specifici. Questi ultimi due esempi son riferibili a errori semantici.

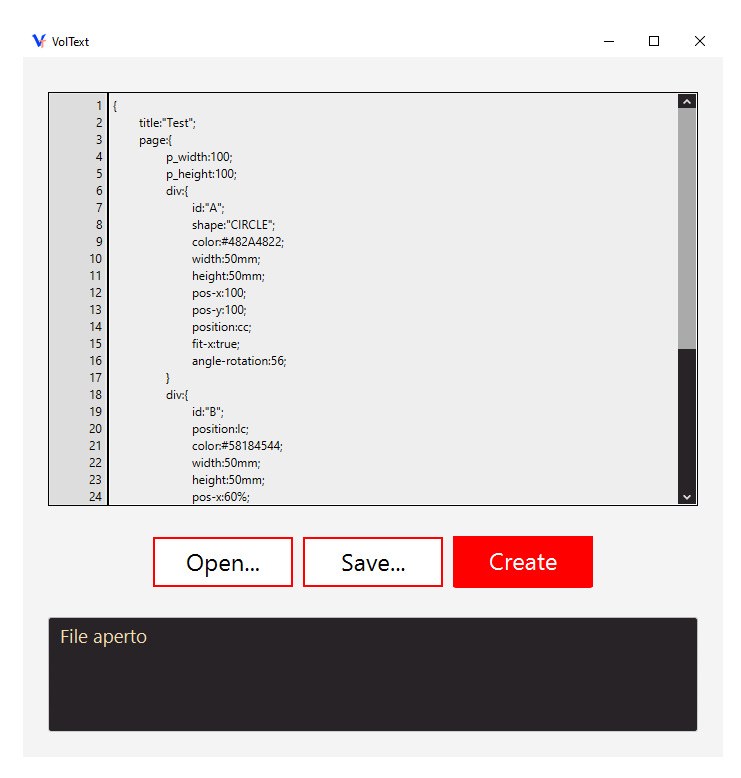
Altre segnalazioni vengono effettuate anche se la posizione inserita dell’elemento non è corretta o esterna alla pagina, se la pagina è specificata sia come formato che come dimensioni (in quel caso viene preso il formato della pagina) o se le dimensioni sono maggiori rispetto al contenitore che contiene l’elemento.

# Tool utilizzati

Per lo sviluppo sono stati utilizzati diversi strumenti:

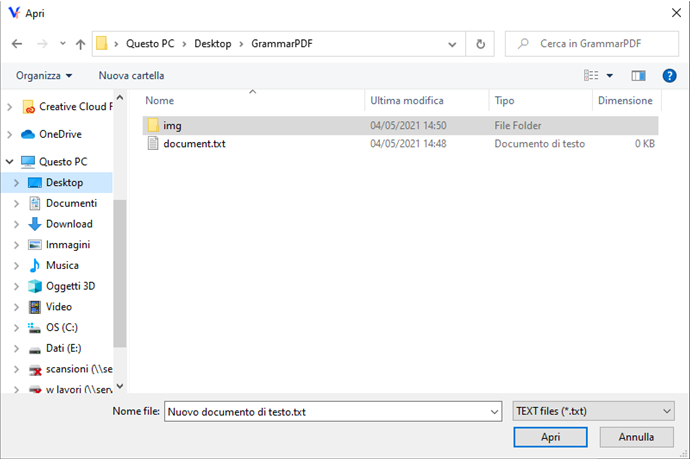
* GitHub: GitHub è un servizio di Version Control utilizzato nel nostro progetto per tenere traccia delle modifiche e dei mutamenti implementati nel nostro software. Ciò ha aiutato anche la condivisione quasi real-time di codice per un lavoro di gruppo migliore.
* GitHub Desktop: client che permette un utilizzo semplificato e rapido di GitHub, utile anche per tenere traccia dei cambiamenti e delle versioni del progetto.
* ANTLR 4: ANTLR (ANother Tool for Language Recognition) è un potente generatore di parser per leggere, elaborare, eseguire o tradurre testo strutturato o file binari. Da una grammatica, ANTLR genera un parser che può costruire e analizzare alberi di analisi. La versione 4 è stata utilizzata per stare al passo con le versioni più recenti e per una comodità con l’IDE Eclipse, vista la presenza di un plugin apposito.
* Eclipse: IDE per lo sviluppo della libreria e della GUI, che supporta anche vari plugin per ANTLR e JavaFX.
* PDFBox: libreria java per la creazione e modifica di documenti PDF.
* PDFBox-layout: libreria java aggiuntiva a PDF-box che permette una gestione (non del tutto compatibile) semplificata di testi e liste.
* JavaFX: libreria che permette l’implementazione di una interfaccia grafica per un programma java. In particolare, per Eclipse, è stato utilizzato il plugin e(fx)clipse che permette la creazione di progetti JavaFX e l’utilizzo delle librerie annesse.
* Launch4j: software per l’incapsulamento di eseguibili di tipo exe altamente personalizzabili a partire da eseguibili di tipo jar.

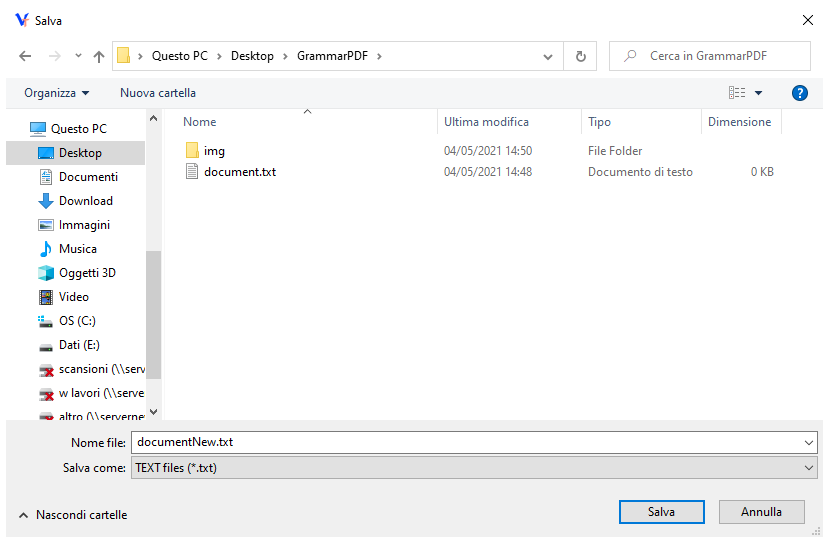
1. **User interface**

The application can be used through a graphical interface developed with JavaFX. At the execution it comes with a GUI consisting of a text area where the user can write and edit the grammar of a PDF document, two buttons similar to each other, respectively for opening and saving a text file, a red button for creating the document, and a non-editable text area that is used to communicate any messages and errors to the user.

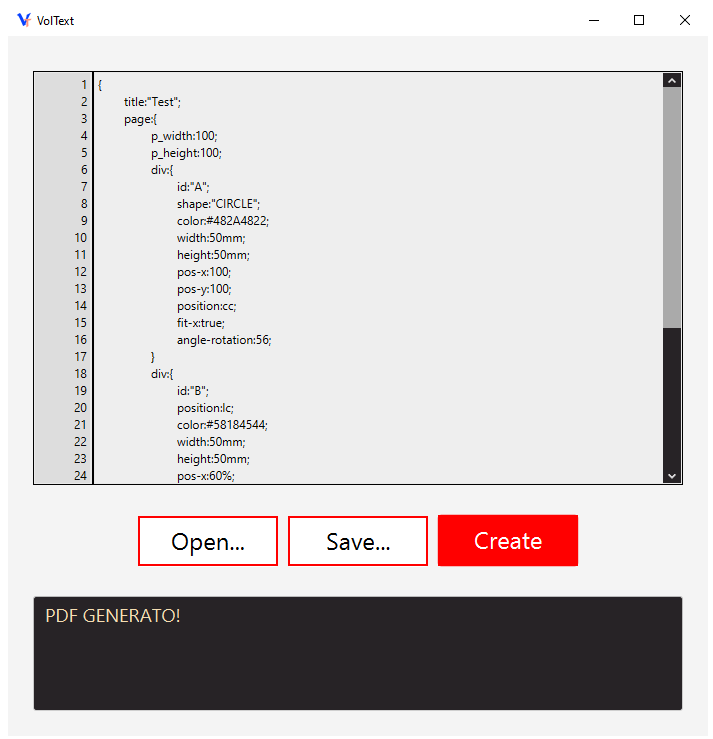
* 1. **General use case**

The user can write the grammar from scratch, but the most common use case is certainly the one in which the grammar, or a part of it, is already in an existing text file. In this case, with the "Open ..." button is possible to select the file to be loaded through a file system navigation interface, receiving a confirmation message.



 After opening the file, the user can make all the necessary changes to the grammar, and by pressing the "Save ..." button it can be saved to a file using an interface similar to the one used during the loading process.

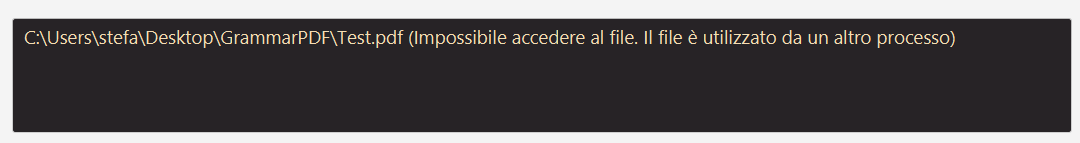
When the user is ready to start generating the document, it can be done by pressing the "Create" button and, at the end of the procedure, if it is successful, a confirmation message is shown, in addiction to any warnings that need to be reported to the user.



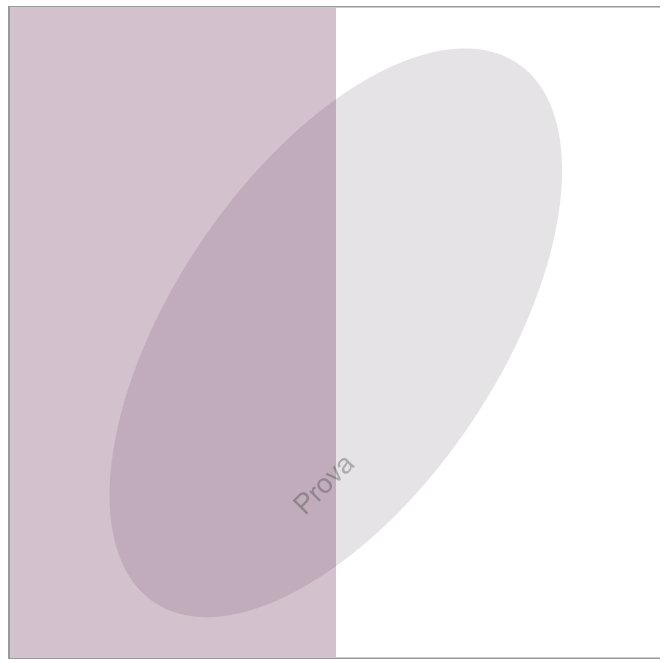
## Reliability

The application was developed to tolerate incorrect usage by the user without compromising the execution of the software, but instead showing clear error messages that can help the user to understand the cause.

For example, if the user, while opening a document, enters the name of a non-existent file, in the first version, he returned the message "The specified file cannot be found" in the textual error area, while currently the The system explorer helps to check the existence of the document. Another example is the situation where the destination path of the pdf file already exists and is used by another process; In this case, the message “Cannot access the file. The file is being used by another process”.

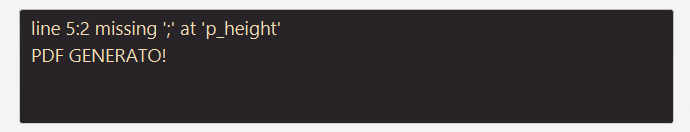


A possible example of grammar is the following one, which describes a simple PDF document called Test containing a single page of 100x100 size, two coloured containers and a rotated text:



It is interesting to note that the application generates a PDF document even with the presence of errors in the grammar, however, returning a warning message to the user.

For example, by eliminating the semicolon after 'p\_width: 100' in the previous grammar, the software produces a correct document, but reporting the error through the graphical interface.



# Possible future developments

During the development of the application, some interesting possible additions emerged for the expansion of the project. Here are exposed some ideas:

* The *underline* attribute, even if implemented, does not work correctly due to effective methods in the PDFBox library and incompatibility between PDFBox and PDFBox-layout.

The generation of a function that allows to underline text with a more simple mode in terms of programming and calculation is in future plans.

* The implementation of constructs such as tables, links, mathematical formulas and graphs.
* A possible improvement in grammar such as simplification of constructs, id without double quotes, colour management also through other formats.
* Code optimization, especially for the VolTextListener.java class.
* Management of containers within other containers, management of objects internal to non-rectangular containers, management inside the grammar of particular constructs to reduce the document drafted by the user.
* Implementation of auxiliary tools for previewing the document before the generation.