Documentazione Tecnica Text2HTML

Benedetta Lenuzza, Matr. 1068745
Aurora Zanenga, Matr. 1054891

August 20, 2024

Contents

1 Introduzione

Text2HTML è stato creato per soddisfare l'esigenza dei professionisti del settore editoriale di ottimizzare le proprie attività permettendo la generazione automatica di pagine web per libri senza la necessità di compiere compiti ripetitivi.
Analizzando questo settore è emerso il seguente problema: i web developer che operano nell'editoria devono frequentemente compiere azioni ripetitive per libri diversi ottenendo però risultati molto simili. Di conseguenza il codice prodotto da questi sviluppatori risulta essere quasi sempre identico seppur con lievi variazioni, che tuttavia aumentano significativamente il loro carico di lavoro portandoli spesso a commettere piccoli errori che prolungano i tempi di completamento.

Questi elementi hanno guidato la creazione del compilatore Text2HTML; tramite un'interfaccia grafica intuitiva (sviluppata per rendere l'esperienza più piacevole) consente di redigere la scheda di un libro e definire le caratteristiche visive della pagina web.
Text2HTML genera la pagina web corrispondente effettuando una serie di controlli sulla struttura e sul contenuto per fornire messaggi utili allo sviluppatore.

2 Requisiti funzionali

L'obiettivo principale di Text2HTML è convertire un testo in una pagina web personalizzata.
Il testo deve seguire una struttura ben definita:
la parte iniziale deve specificare tutte le caratteristiche relative al CSS mentre la parte centrale deve includere i campi della scheda libro i quali non sono tutti obbligatori.

Text2HTML fornisce una semplice ed intuitiva interfaccia grafica:

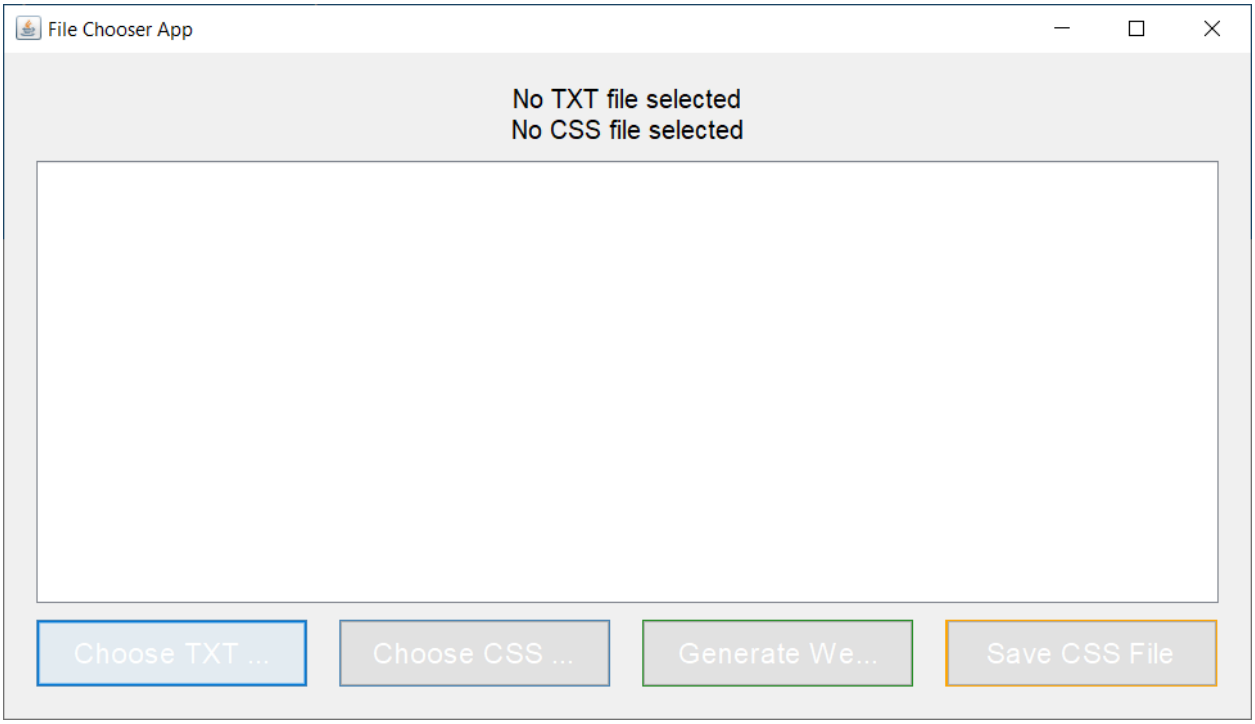

Figure 1: Schermata principale applicazione

- Premendo il tasto **Choose TXT File** si apre una finestra di dialogo che permette di selezionare il file di testo da caricare all'interno dell'applicazione.
- Dopo aver selezionato il file, in alto nella schermata principale dell'applicazione, viene mostrato il nome del file txt selezionato.
- Premendo il tasto **Choose CSS File** si apre una finestra di dialogo che permette di selezionare il file css da utilizzare per lo stile della pagina.
- Dopo aver selezionato il file, in alto nella schermata principale dell'applicazione, viene mostrato il nome del file css selezionato.
- Premendo il tasto **GenerateWebPage** si aprirà una nuova finestra di dialogo in cui sarà possibile scegliere dove salvare il file contente il codice HTML della pagina web.
  - Se non sono presenti errori nel file caricato si apparirà un popup che confermerà che la pagina è stata creata correttamente:

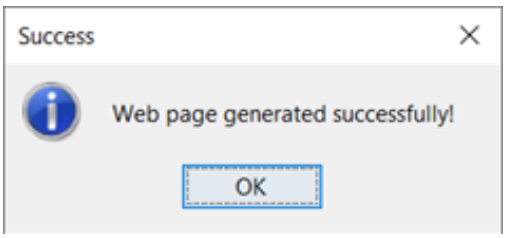

Figure 2: Alert

– Se sono presenti errori verrà mostrato un alert in cui verrà specificata la riga dove è presente l'errore.
  Un esempio di alert è il seguente:

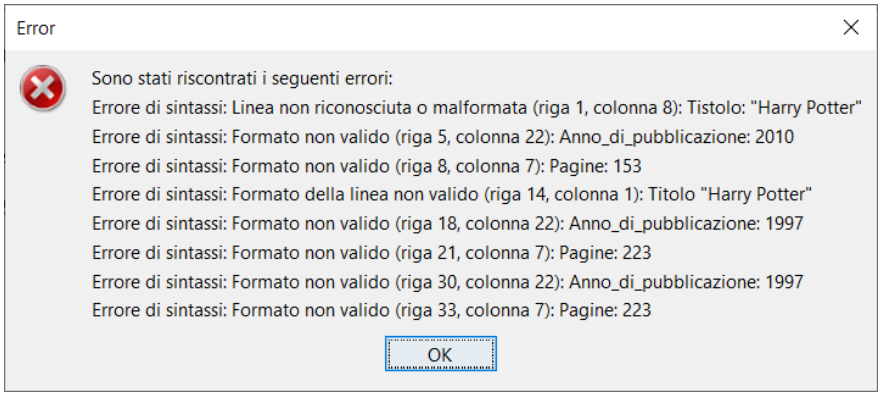

Figure 3: Alert

Il funzionamento di Text2HTML è riassunto nel seguente statechart diagram:

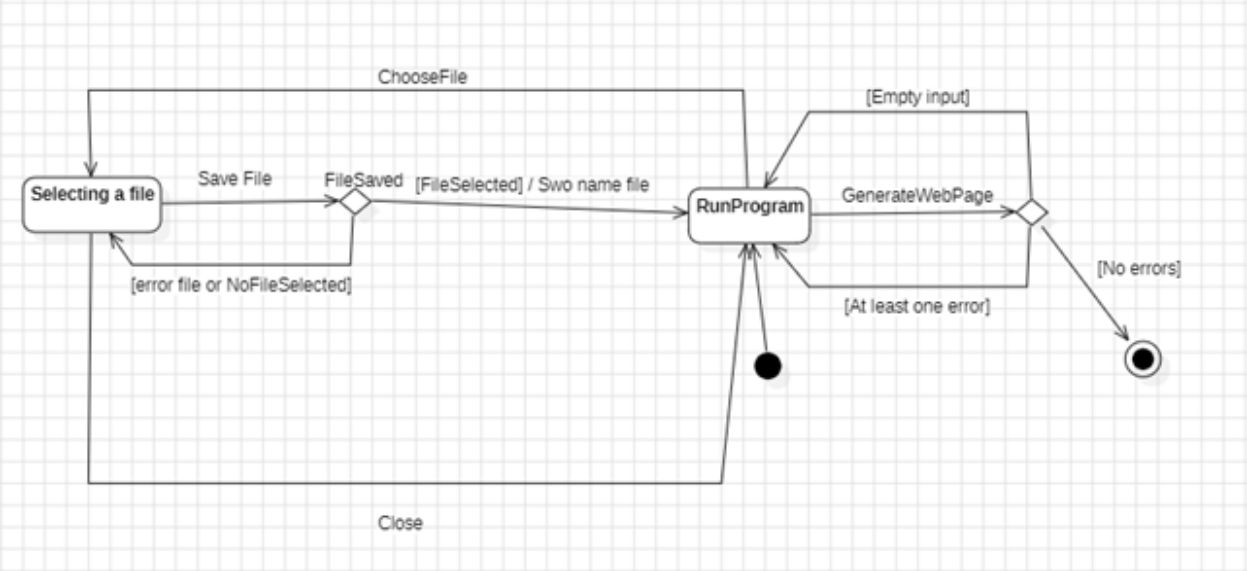

Figure 4: Statechart Diagram di Text2HTML

3 Tecnologie

Text2HTML è stato sviluppato con l'aiuto dei seguenti tool e delle seguenti tecnologie:

- Il linguaggio di programmazione utilizzato è Java e l'IDE utilizzato è Eclipse.
- Il framework utilizzato per lo sviluppo dell'interfaccia grafica è Java Swing.
- La grammatica per Text2HTML è stata definita attraverso l'uso di ANTLR 3.4 con l'aiuto dell'IDE ANTLRWorks.

4 Struttura del Progetto

Il progetto è composto dai seguenti file:

- **HtmlHandler**: Una classe Java responsabile della generazione del codice HTML. E' contenuta nel package myCompilerPackage.
- **Grammatica**: Un insieme di regole definite con ANTLR per il parsing del file di testo. E' contenuta nel package myCompilerPackage.
- **ParserLauncher**: Una classe Java che avvia il processo di parsing e gestisce l'output HTML. E' contenuta nel package myPackage.
- **FileChooserApp**: Un'applicazione Java Swing che consente agli utenti di selezionare file TXT e CSS, visualizzare e modificare il contenuto del file CSS, e generare una pagina web HTML basata sui file selezionati. È contenuta nel package myPackage.

5 HtmlHandler

La classe `HtmlHandler` è responsabile della creazione del contenuto HTML. Definisce vari metodi per aggiungere elementi HTML come titoli, autori, generi e altre informazioni relative ai libri. Di seguito vengono esposte alcune delle sue funzioni principali:

- `startHtml()`: Inizia il documento HTML.
- `endHtml()`: Termina il documento HTML.
- `addTitle(String text)`: Aggiunge un titolo di libro al documento HTML.
- `addAuthor(String text)`: Aggiunge l'autore di un libro.
- `addGenre(String text)`: Aggiunge il genere di un libro.
- `addPublicationYear(String text)`: Aggiunge l'anno di pubblicazione di un libro.
- `addBookSeparator()`: Aggiunge un separatore tra le schede di diversi libri.

5.1 Panoramica

HtmlHandler è una classe Java progettata per generare un documento HTML che rappresenta schede di libri. La classe legge dati da un file di testo strutturato e li trasforma in un formato HTML. La classe supporta anche la lettura di un file CSS per applicare stili personalizzati.

5.2 Attributi Principali

- **errors**: Una lista di stringhe che memorizza eventuali errori di sintassi incontrati durante la lettura dei file.
- **generatedHtml**: Un StringBuilder che costruisce l'output HTML finale.
- **bgColor, fontType, marginTop, marginBottom, marginLeft, marginRight, textAlign, textLineHeight**: Variabili per configurare lo stile CSS della pagina.
- **bookIndex**: Contatore per il numero di libri elaborati.
- **side_bar_content**: Stringa che costruisce il contenuto della barra laterale di navigazione.

5.3 Metodi Principali

- **startHtml()**: Inizia il documento HTML aggiungendo le intestazioni e i riferimenti al CSS.
- **endHtml()**: Chiude il documento HTML, chiudendo l'ultima scheda del libro e aggiungendo la barra laterale di navigazione.
- **addCss(String cssContent)**: Aggiunge il contenuto CSS in linea all'HTML generato.

- **addTitle(String text)**, **addAuthor(String text)**, **addGenre(String text)**, ecc.: Metodi per aggiungere varie informazioni sul libro all'HTML generato.
- **readCssFile(String cssFilePath)**: Legge un file CSS e aggiunge il suo contenuto all'HTML generato.
- **readTextFile(String textFilePath)**: Legge un file di testo strutturato e processa ogni linea per aggiungere le informazioni sul libro all'HTML.
- **validateRequiredFields()**: Verifica che i campi richiesti (Titolo, Autore, Trama) siano presenti nel file di testo.
- **processLine(String line, int lineNumber)**: Elabora una singola linea del file di testo, identificando l'etichetta e il valore, e aggiornando l'HTML di conseguenza.

5.4 Funzionamento Dettagliato

- **Inizializzazione**: Quando un'istanza di HtmlHandler è creata, *errors* e *generatedHtml* sono inizializzati vuoti.
- **Inizio HTML**: Il metodo *startHtml()* prepara l'intestazione del documento HTML, includendo il titolo e i riferimenti al CSS.
- **Lettura File**:
  - *readCssFile(String cssFilePath)* legge un file CSS e lo aggiunge all'HTML.
  - *readTextFile(String textFilePath)* legge un file di testo riga per riga, processando ogni linea con *processLine(String line, int lineNumber)*.
- **Elaborazione Linee**: Ogni linea del file di testo viene analizzata per etichetta e valore. Se la linea è corretta, viene aggiunta all'HTML, altrimenti un errore viene registrato in *errors*.
- **Validazione**: *validateRequiredFields()* assicura che i campi necessari siano stati forniti nel file di testo.
- **Fine HTML**: Il metodo *endHtml()* chiude la costruzione dell'HTML, aggiungendo la barra laterale e chiudendo i tag HTML aperti.

5.5 Uso

Un tipico uso di HtmlHandler coinvolge la creazione di un'istanza della classe, l'uso dei metodi *startHtml()*, *readCssFile()*, *readTextFile()*, e *endHtml()*, e infine la chiamata a *getGeneratedHtml()* per ottenere il documento HTML generato.

5.6 Codice Java

```
package myCompilerPackage;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class HtmlHandler {
    private static final String ERROR_PREFIX = "Errore di sintassi:
        ";
    private List<String> errors;
    private StringBuilder generatedHtml;

    public HtmlHandler() {
        this.errors = new ArrayList<>();
        this.generatedHtml = new StringBuilder();
    }

    public List<String> getErrors() {
        return errors;
    }

    // Variabili CSS e HTML
    String bgColor = "\"grey\"";
    String fontType = "\"sans-serif\"";
    String marginTop = "\"0px\"";
    String marginBottom = "\"0px\"";
    String marginRight = "\"0px\"";
    String marginLeft = "\"0px\"";
    String textAlign = "\"center\"";
    String textLineHeight = "\"normal\"";
    int bookIndex = 0;
    String side_bar_content = "<div class = 'sidebar'>";

    private boolean hasTitle = false;
    private boolean hasAuthor = false;
    private boolean hasPlot = false;

    public static String removeFirstAndLastChar(String str) {
        if (str == null || str.length() < 2) {
            return "";
        }
        if (str.charAt(0) == '\"' && str.charAt(str.length() - 1) ==
            '\"') {
```

```
            return str.substring(1, str.length() - 1);
        } else {
            return str;
        }
    }

    public void startHtml() {
        generatedHtml.append("<!DOCTYPE␣html>\n<html>\n<head>");
        generatedHtml.append("<title>Schede␣Libro</title>");
        generatedHtml.append("<link␣rel=\"stylesheet\"␣href=\"styles
              .css\">");
        generatedHtml.append("<style>")
                     .append(".page-break␣{␣page-break-before:␣
                           always;␣}")
                     .append("</style>");
        generatedHtml.append("</head>\n<body>");
        generatedHtml.append("<div␣class=\"container\">");
    }

    public void endHtml() {
        // Chiudi l'ultima pagina
        closeCurrentPage();
        generatedHtml.append("<div␣class=\"content\">");
        generatedHtml.append("</div>␣<!--␣end␣of␣content␣-->");
        generatedHtml.append("</div>␣<!--␣end␣of␣container␣-->");
        generatedHtml.append(side_bar_content + "" + "</div>");
        generatedHtml.append("</body>\n</html>");
    }

    public void addCss() {
        // CSS is included by link, we don't need inline CSS
    }

    public void addPageTitle() {
        // Title is included in startHtml
    }

    public void setBgColor(String text) {
        bgColor = text;
    }

    public void setFontType(String text) {
        fontType = text;
    }

    public void setMarginTop(String text) {
        marginTop = text;
```

```
}

public void setMarginBottom(String text) {
    marginBottom = text;
}

public void setMarginLeft(String text) {
    marginLeft = text;
}

public void setMarginRight(String text) {
    marginRight = text;
}

public void setTextAlign(String text) {
    textAlign = text;
}

public void setTextLineHeight(String text) {
    textLineHeight = text;
}

public void addTitle(String text) {
    text = removeFirstAndLastChar(text);
    if (bookIndex > 0) {
        closeCurrentPage(); // Chiudi la pagina corrente prima
              di iniziarne una nuova
    }
    bookIndex++;
    generatedHtml.append("<div␣id=\"book" + bookIndex + "\"␣
          class=\"book-card\">");
    generatedHtml.append("<center><h1>" + text + "</h1></center>
          ");
    side_bar_content += "\n␣<a␣href='#book" + bookIndex + "'>␣"
          + text + "␣</a>␣\n";
}

public void addAuthor(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<center><div>" + text + "</div></
          center><br>");
}

public void addGenre(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Genere</i>:␣" + text + "</div
          ><br>");
```

```
}

public void addCompositionDate(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Data di composizione</i>: " +
          text + "</div><br>");
}

public void addPublicationYear(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Anno di pubblicazione</i>: " +
           text + "</div><br>");
}

public void addPublisher(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Editore</i>: " + text + "</div
          ><br>");
}

public void addNarrator(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Narratore</i>: " + text + "</
          div><br>");
}

public void addPages(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Pagine</i>: " + text + "</div
          ><br>");
}

public void addPrice(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Prezzo</i>: " + text + "</div
          ><br>");
}

public void addStructure(String text) {
    text = removeFirstAndLastChar(text);
    generatedHtml.append("<div><i>Struttura</i>: " + text + "</
          div><br>");
}

public void addPlot(String text) {
    text = removeFirstAndLastChar(text);
```

```
        generatedHtml.append("<div><i>Trama</i>: " + text + "</div><
            br>");
    }

    public void addMessage(String text) {
        text = removeFirstAndLastChar(text);
        generatedHtml.append("<div><i>Messaggio</i>: " + text + "</
            div><br>");
    }

    public void addCharacters(String text) {
        text = removeFirstAndLastChar(text);
        generatedHtml.append("<div><i>Personaggi</i>: " + text + "</
            div><br>");
    }

    private void closeCurrentPage() {
        generatedHtml.append("</div> <!-- end of book-card --> ");
        generatedHtml.append("<div class=\"page-break\"></div>"); //
            Interruzione di pagina
    }

    public void readCssFile(String cssFilePath) {
        try (BufferedReader br = new BufferedReader(new FileReader(
            cssFilePath))) {
            StringBuilder cssBuilder = new StringBuilder();
            String line;
            while ((line = br.readLine()) != null) {
                cssBuilder.append(line).append("\n");
            }
            addCss(cssBuilder.toString()); // Assicurati di
                aggiungere il CSS al documento HTML
        } catch (IOException e) {
            errors.add(ERROR_PREFIX + "Impossibile leggere il file
                CSS: " + cssFilePath);
            e.printStackTrace();
        }
    }

    private void addCss(String cssContent) {
        generatedHtml.append("<style>\n")
                     .append(cssContent)
                     .append("\n</style>\n");
    }

    public void readTextFile(String textFilePath) {
```

```
        try (BufferedReader br = new BufferedReader(new FileReader(
              textFilePath))) {
            String line;
            int lineNumber = 0;
            while ((line = br.readLine()) != null) {
                lineNumber++;
                processLine(line, lineNumber);
            }
            validateRequiredFields();
        } catch (IOException e) {
            errors.add(ERROR_PREFIX + "Impossibile leggere il file
                  di testo: " + textFilePath);
            e.printStackTrace();
        }
    }

    private void validateRequiredFields() {
        if (!hasTitle) {
            errors.add(ERROR_PREFIX + "Manca il Titolo nel file di
                  testo.");
        }
        if (!hasAuthor) {
            errors.add(ERROR_PREFIX + "Manca l'Autore nel file di
                  testo.");
        }
        if (!hasPlot) {
            errors.add(ERROR_PREFIX + "Manca la Trama nel file di
                  testo.");
        }
    }

    public String getGeneratedHtml() {
        return generatedHtml.toString();
    }

    private void processLine(String line, int lineNumber) {
        int colonIndex = line.indexOf(":");
        if (colonIndex != -1) {
            String label = line.substring(0, colonIndex).trim();
            String value = line.substring(colonIndex + 1).trim();

            // Calcola la colonna dell'inizio del valore
            int column = colonIndex + 1; // La colonna inizia subito
                  dopo i due punti

            // Controlla se il valore inizia e finisce con
                  virgolette
```

```
if (value.startsWith("\"") && value.endsWith("\"")) {
    value = removeFirstAndLastChar(value);
    switch (label.toLowerCase()) {
        case "title":
            addTitle(value);
            hasTitle = true;
            break;
        case "author":
            addAuthor(value);
            hasAuthor = true;
            break;
        case "genre":
            addGenre(value);
            break;
        case "composition_date":
            addCompositionDate(value);
            break;
        case "publication_year":
            addPublicationYear(value);
            break;
        case "publisher":
            addPublisher(value);
            break;
        case "narrator":
            addNarrator(value);
            break;
        case "pages":
            addPages(value);
            break;
        case "price":
            addPrice(value);
            break;
        case "structure":
            addStructure(value);
            break;
        case "plot":
            addPlot(value);
            hasPlot = true;
            break;
        case "message":
            addMessage(value);
            break;
        case "characters":
            addCharacters(value);
            break;
        default:
```

```
                        errors.add(ERROR_PREFIX + "Etichetta non
                              riconosciuta alla riga " + lineNumber
                              + ", colonna " + column + ": " + label
                              );
                        break;
                    }
                } else {
                    errors.add(ERROR_PREFIX + "Valore non valido alla
                          riga " + lineNumber + ", colonna " + column +
                          ": " + value);
                }
            } else {
                errors.add(ERROR_PREFIX + "Formato non valido alla riga
                     " + lineNumber + ": manca il carattere \":\".");
            }
        }
    }
```

6 Grammatica

La grammatica definita in ANTLR specifica le regole sintattiche per il parsing del file di testo di input. Alcune delle regole principali includono:

- `startRule`: Regola iniziale che avvia il parsing.
- `bookRule`: Regola che definisce la struttura di una scheda libro.
- `tagTitleDefinitionRule`: Regola per il parsing del titolo di un libro.
- `tagAuthorDefinitionRule`: Regola per il parsing dell'autore di un libro.
- `tagGenreDefinitionRule`: Regola per il parsing del genere di un libro.

6.1 Panoramica

Il codice fornito è la definizione di un parser scritto utilizzando ANTLR, un generatore di parser che converte file di grammatica in codice sorgente per l'analisi sintattica.
Questo parser è progettato per analizzare una grammatica che viene utilizzata per la gestione di un documento HTML con tag specifici e informazioni sui libri.

6.2 Struttura del Codice

6.2.1 Header di Lexer e Parser

Il codice inizia con l'intestazione del lexer e del parser che definisce i pacchetti Java utilizzati.

```
@lexer::header{
    package myCompilerPackage;
}

@parser::header{
    package myCompilerPackage;
    import myCompilerPackage.HtmlHandler;
}
```

6.2.2 Membri del Parser

Definisce un oggetto `HtmlHandler` che gestirà la manipolazione dei dati durante il parsing.

```
@parser::members{
    HtmlHandler handler = new HtmlHandler();
}
```

6.2.3 Regole Sintattiche

Le regole sintattiche specificano come analizzare i diversi elementi del documento.

```
startRule
    : {handler.startHtml();}
      tagBgColor?
      tagFontType?
      tagMarginTop?
      tagMarginBottom?
      tagMarginLeft?
      tagMarginRight?
      tagTextAlign?
      tagTextLineHeight?
      tagCss
      (bookRule)*
      {handler.endHtml();}
      EOF
    ;
```

6.2.4 Regole per le Informazioni del Libro

Gestisce le informazioni relative ai libri e può includere vari tag definiti.

```
bookRule
    :
    tagTitleDefinitionRule
    tagAuthorDefinitionRule( tagGenreDefinitionRule
    tagCompositionDateDefinitionRule
    tagPublicationYearDefinitionRule
    tagPublisherDefinitionRule
    tagNarratorDefinitionRule
    tagPagesDefinitionRule
    tagPriceDefinitionRule
    tagStructureDefinitionRule
    tagPlotDefinitionRule
    tagMessageDefinitionRule
    tagCharactersDefinitionRule )*
    {handler.addBookSeparator();}
    ;
```

6.2.5 Definizione dei Tag: Tag di Colore di Sfondo, Tipo di Font, Margini, Allineamento del Testo, e Altri Tag CSS

```
tagBgColor
  : BGCOLOR COLON text = STRING {handler.setBgColor($text.text);}
```

```
  ;

tagFontType
  : FONT COLON text = STRING {handler.setFontType($text.text);}
  ;

tagMarginTop
  : MARGINTOP COLON text = STRING {handler.setMarginTop($text.text);}
  ;

tagMarginBottom
  : MARGINBOTTOM COLON text = STRING {handler.setMarginBottom($text.text);}
  ;

tagMarginLeft
  : MARGINLEFT COLON text = STRING {handler.setMarginLeft($text.text);}
  ;

tagMarginRight
  : MARGINRIGHT COLON text = STRING {handler.setMarginRight($text.text);}
  ;

tagTextAlign
  : TEXTALIGN COLON text = STRING {handler.setTextAlign($text.text);}
  ;

tagTextLineHeight
  : TEXTLINEHEIGHT COLON text = STRING {handler.setTextLineHeight($text.text)
  ;

tagCss
  : {handler.addCss();}
  ;
```

6.2.6 Definizione dei Tag: Tag per le Informazioni del Libro

```
tagTitleDefinitionRule
    : TITOLO COLON text = STRING {handler.addTitle($text.text);}
    ;

tagAuthorDefinitionRule
    : AUTORE COLON text=STRING {handler.addAuthor($text.text);}
    ;
```

```
tagGenreDefinitionRule
    : GENERE COLON text=STRING {handler.addGenre($text.text);}
    ;

tagCompositionDateDefinitionRule
    : DATA_DI_COMPOSIZIONE COLON text=STRING {handler.addCompositionDate($tex
    ;

tagPublicationYearDefinitionRule
    : ANNO_DI_PUBBLICAZIONE COLON text=INT {handler.addPublicationYear($text.
    ;

tagPublisherDefinitionRule
    : EDITORE COLON text=STRING {handler.addPublisher($text.text);}
    ;

tagNarratorDefinitionRule
    : NARRATORE COLON text=STRING {handler.addNarrator($text.text);}
    ;

tagPagesDefinitionRule
    : NUMERO_PAGINE COLON text=INT {handler.addPages($text.text);}
    ;

tagPriceDefinitionRule
    : PREZZO COLON text=STRING {handler.addPrice($text.text);}
    ;

tagStructureDefinitionRule
    : STRUTTURA COLON text=STRING {handler.addStructure($text.text);}
    ;

tagPlotDefinitionRule
    : TRAMA COLON text=STRING {handler.addPlot($text.text);}
    ;

tagMessageDefinitionRule
    : MESSAGGIO_LANCIATO COLON text=STRING {handler.addMessage($text.text);}
    ;

tagCharactersDefinitionRule
    : PERSONAGGI COLON text=STRING {handler.addCharacters($text.text);}
    ;
```

6.2.7 Token

Definisce i token che il lexer utilizza per riconoscere i vari componenti della grammatica.

```
TAG_OPEN : '<';
TAG_CLOSE : '>';
TAG_TITLE: 'title';
TAG_HEAD: 'head';
SLASH : '/';
SC : ';';
DOT : '.';
COLON : ':';
COMMA : ',';
TRATTINO: '-';
EURO : '€';
```

Esempi di token per le definizioni di libro:

```
AUTORE : 'Autore';
TITOLO : 'Titolo';
BGCOLOR : 'BgColor';
MARGINTOP : 'margin-top';
MARGINBOTTOM : 'margin-bottom';
MARGINLEFT : 'margin-left';
MARGINRIGHT : 'margin-right';
TEXTALIGN : 'text-align';
TEXTLINEHEIGHT : 'line-height';
FONT : 'fontType';
```

6.2.8 Token di Spazi Bianchi e Commenti: Commenti

```
COMMENT
    : '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    | '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
    ;
```

6.2.9 Token di Spazi Bianchi e Commenti: Spazi Bianchi

```
WS : ( ' ' | '\t' | '\r' | '\n' ) {$channel=HIDDEN;} ;
```

7 ParserLauncher

La classe `ParserLauncher` gestisce l'intero processo di parsing e generazione dell'output HTML. Essa:

- Inizializza il lexer e il parser ANTLR.
- Reindirizza l'output del parser verso un buffer.
- Scrive l'output HTML in un file specificato.
- Copia il file CSS nel percorso di destinazione dell'HTML.

7.1 Panoramica

La classe `ParserLauncher` è progettata per gestire il parsing di un file di testo e la generazione di un file HTML, utilizzando ANTLR per l'analisi sintattica. Include anche una funzionalità per copiare un file CSS in una posizione specificata.

7.2 Struttura del Codice

7.2.1 Importazioni

Il codice importa le seguenti classi e pacchetti:

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.UIManager;
import org.antlr.runtime.ANTLRReaderStream;
import org.antlr.runtime.CommonTokenStream;
import myCompilerPackage.grammatica_completaLexer;
import myCompilerPackage.grammatica_completaParser;

import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
```

7.3 Costruttore `ParserLauncher`

Il costruttore della classe `ParserLauncher` accetta tre percorsi di file come argomenti: `txtFilePath`, `cssFilePath`, e `htmlFilePath`. Esegue le seguenti operazioni:

- **Copia del File CSS:**

```
try {
    String cssDestPath = Paths.get(htmlFilePath).getParent().
        resolve("styles.css").toString();
    Files.copy(Paths.get(cssFilePath), Paths.get(cssDestPath)
        , StandardCopyOption.REPLACE_EXISTING);
    System.out.println("File CSS copiato correttamente in: "
        + cssDestPath);
} catch (IOException e) {
    System.err.println("Errore durante la copia del file CSS:
        " + e.getMessage());
}
```

- **Parsing del File di Testo e Creazione del File HTML:**

```
try {
    OutputStream outputStream = new OutputStream() {
        private StringBuilder buffer = new StringBuilder();

        @Override
        public void write(int b) throws IOException {
            buffer.append((char) b);
        }

        @Override
        public String toString() {
            return buffer.toString();
        }
    };

    PrintStream customOut = new PrintStream(outputStream);
    System.setOut(customOut);

    grammatica_completaLexer lexer = new
        grammatica_completaLexer(new ANTLRReaderStream(new
        FileReader(txtFilePath)));
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    grammatica_completaParser parser = new
        grammatica_completaParser(tokens);
    parser.startRule();

    System.setOut(standardOut);

    try (BufferedWriter writer = new BufferedWriter(new
        FileWriter(htmlFilePath))) {
        writer.write(outputStream.toString());
        System.out.println("File salvato correttamente in: "
            + htmlFilePath);
    } catch (IOException e) {
        System.err.println("Errore durante il salvataggio del
            file: " + e.getMessage());
    }
} catch (Exception e) {
    System.out.println("Parsing con ANTLR abortito");
    e.printStackTrace();
```

```
    }
```

7.4 Metodo `main`

Il metodo `main` è il punto di ingresso dell'applicazione e si occupa di:

```
public static void main(String[] args) {
    String txtFilePath = "C:\\Users\\auror\\Desktop\\file\\file.
        txt";
    String cssFilePath = "C:\\Users\\auror\\Desktop\\file\\file.
        css";
    String htmlFilePath = "C:\\Users\\auror\\Desktop\\file\\file
        .html";

    new ParserLauncher(txtFilePath, cssFilePath, htmlFilePath);
}
```

7.5 Dettagli Tecnici: Gestione degli Stream

- **Reindirizzamento dell'Output:** Utilizza un `OutputStream` personalizzato per catturare l'output generato dal parser. Questo output viene poi scritto nel file HTML.
- **Copia del File CSS:** La copia del file CSS viene eseguita utilizzando `Files.copy` e gestita con `StandardCopyOption.REPLACE_EXISTING` per sovrascrivere eventuali file esistenti con lo stesso nome.

7.6 Gestione delle Eccezioni

- **Eccezioni di I/O:** Le eccezioni durante la copia del file CSS e la scrittura del file HTML vengono catturate e stampate su `System.err`.
- **Eccezioni Generali:** Le eccezioni generali durante il parsing vengono catturate e stampate, ma non interrompono il flusso del programma.

8 FileChooserApp

La classe `FileChooserApp` è un'applicazione Java Swing che fornisce un'interfaccia grafica per selezionare file TXT e CSS, visualizzare e modificare il contenuto del file CSS, e generare una pagina web HTML.

8.1 Panoramica

La `FileChooserApp` è progettata per facilitare la selezione di file di testo e fogli di stile, permettere la modifica dei CSS direttamente dall'interfaccia e infine generare un file HTML che combina il contenuto del testo e lo stile definito.

8.2 Attributi Principali

- **frame**: La finestra principale dell'applicazione.
- **chooseFileButton**: Bottone per scegliere un file TXT.
- **chooseCssButton**: Bottone per scegliere un file CSS.
- **generateWebPageButton**: Bottone per generare la pagina web HTML.
- **saveFileButton**: Bottone per salvare il file CSS.
- **fileLabel**: Etichetta che mostra il file TXT selezionato.
- **cssLabel**: Etichetta che mostra il file CSS selezionato.
- **fileContentTextArea**: Area di testo per visualizzare e modificare il contenuto del file CSS.
- **selectedTxtFile**: Il file TXT selezionato.
- **selectedCssFile**: Il file CSS selezionato.

8.3 Metodi Principali

- **FileChooserApp()**: Costruttore che inizializza l'interfaccia grafica e i relativi listener per i bottoni.
- **main(String[] args)**: Metodo principale che avvia l'applicazione.

8.4 Funzionamento Dettagliato

- **Inizializzazione**: Viene creata la finestra principale e configurata con i vari componenti (bottoni, etichette, area di testo).
- **Scelta File**: Gli utenti possono selezionare file TXT e CSS tramite i relativi bottoni. Il contenuto del file CSS viene visualizzato nell'area di testo.

- **Salvataggio File**: Le modifiche al contenuto del file CSS possono essere salvate nel file originale.
- **Generazione Pagina Web**: Utilizzando i file selezionati, viene generata una pagina web HTML che combina il testo e lo stile CSS. Eventuali errori vengono mostrati all'utente.

8.5 Codice Java

```
package myPackage;

import javax.swing.*;

import myCompilerPackage.HtmlHandler;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;

public class FileChooserApp {
    private JFrame frame;
    private JButton chooseFileButton;
    private JButton chooseCssButton;
    private JButton generateWebPageButton;
    private JButton saveFileButton;
    private JLabel fileLabel;
    private JLabel cssLabel;
    private JTextArea fileContentTextArea;
    private File selectedTxtFile;
    private File selectedCssFile;

    public FileChooserApp() {
        FileProcessor   fileProcessor = new FileProcessor();
        WebPageGenerator  webPageGenerator = new
              WebPageGenerator();
        // Creazione della finestra principale
        frame = new JFrame("File Chooser App");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 400);
        frame.setLocationRelativeTo(null); // Centra la
              finestra sullo schermo

        // Imposta il look and feel di sistema
        try {
            UIManager.setLookAndFeel(UIManager.
                  getSystemLookAndFeelClassName());
```

```
} catch (Exception e) {
    e.printStackTrace();
}

// Inizializzazione dei componenti
chooseFileButton = new JButton("Choose TXT File");
chooseFileButton.setFont(new Font("Arial", Font.PLAIN,
       18));
chooseFileButton.setBackground(new Color(70, 130, 180)
      );
chooseFileButton.setForeground(Color.WHITE);
chooseFileButton.setFocusPainted(false);
chooseFileButton.setBorder(BorderFactory.
      createEmptyBorder(10, 20, 10, 20));

chooseCssButton = new JButton("Choose CSS File");
chooseCssButton.setFont(new Font("Arial", Font.PLAIN,
      18));
chooseCssButton.setBackground(new Color(70, 130, 180))
      ;
chooseCssButton.setForeground(Color.WHITE);
chooseCssButton.setFocusPainted(false);
chooseCssButton.setBorder(BorderFactory.
      createEmptyBorder(10, 20, 10, 20));

generateWebPageButton = new JButton("Generate Web Page
      ");
generateWebPageButton.setFont(new Font("Arial", Font.
      PLAIN, 18));
generateWebPageButton.setBackground(new Color(34, 139,
       34));
generateWebPageButton.setForeground(Color.WHITE);
generateWebPageButton.setFocusPainted(false);
generateWebPageButton.setBorder(BorderFactory.
      createEmptyBorder(10, 20, 10, 20));

saveFileButton = new JButton("Save CSS File");
saveFileButton.setFont(new Font("Arial", Font.PLAIN,
      18));
saveFileButton.setBackground(new Color(255, 165, 0));
saveFileButton.setForeground(Color.WHITE);
saveFileButton.setFocusPainted(false);
saveFileButton.setBorder(BorderFactory.
      createEmptyBorder(10, 20, 10, 20));

fileLabel = new JLabel("No TXT file selected");
fileLabel.setFont(new Font("Arial", Font.PLAIN, 16));
```

```
fileLabel.setHorizontalAlignment(SwingConstants.CENTER
    );

cssLabel = new JLabel("No CSS file selected");
cssLabel.setFont(new Font("Arial", Font.PLAIN, 16));
cssLabel.setHorizontalAlignment(SwingConstants.CENTER)
    ;

fileContentTextArea = new JTextArea();
fileContentTextArea.setFont(new Font("Arial", Font.
    PLAIN, 14));
fileContentTextArea.setLineWrap(true);
fileContentTextArea.setWrapStyleWord(true);
JScrollPane scrollPane = new JScrollPane(
    fileContentTextArea);

// Layout manager
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout(10, 10));
panel.setBorder(BorderFactory.createEmptyBorder(20,
    20, 20, 20));

// Panel per i bottoni
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(1, 4, 20, 0));
buttonPanel.add(chooseFileButton);
buttonPanel.add(chooseCssButton);
buttonPanel.add(generateWebPageButton);
buttonPanel.add(saveFileButton);

JPanel labelPanel = new JPanel();
labelPanel.setLayout(new GridLayout(2, 1));
labelPanel.add(fileLabel);
labelPanel.add(cssLabel);

panel.add(buttonPanel, BorderLayout.SOUTH);
panel.add(labelPanel, BorderLayout.NORTH);
panel.add(scrollPane, BorderLayout.CENTER);

frame.getContentPane().add(panel);

// Listener per il bottone chooseFileButton
chooseFileButton.addActionListener(new ActionListener
    () {
    @Override
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileChooser = new JFileChooser();
```

```
            int result = fileChooser.showOpenDialog(frame)
                  ;
            if (result == JFileChooser.APPROVE_OPTION) {
                selectedTxtFile = fileChooser.
                      getSelectedFile();
                fileLabel.setText("Selected TXT file: " +
                      selectedTxtFile.getName());
            }
        }
    });

// Listener per il bottone chooseCssButton
    chooseCssButton.addActionListener(new ActionListener()
           {
        @Override
        public void actionPerformed(ActionEvent e) {
            JFileChooser fileChooser = new JFileChooser();
            int result = fileChooser.showOpenDialog(frame)
                  ;
            if (result == JFileChooser.APPROVE_OPTION) {
                selectedCssFile = fileChooser.
                      getSelectedFile();
                cssLabel.setText("Selected CSS file: " +
                      selectedCssFile.getName());

                // Legge il contenuto del file CSS e lo
                      visualizza nella JTextArea
                try (BufferedReader reader = new
                      BufferedReader(new FileReader(
                      selectedCssFile))) {
                    fileContentTextArea.read(reader, null)
                          ;
                } catch (IOException ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(frame, "
                          Error reading file", "Error",
                          JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    });


    // Listener per il bottone saveFileButton
    saveFileButton.addActionListener(new ActionListener()
          {
        @Override
```

```
        public void actionPerformed(ActionEvent e) {
            if (selectedCssFile != null) {
                // Salva il contenuto della JTextArea nel
                    file CSS
                try (BufferedWriter writer = new
                    BufferedWriter(new FileWriter(
                    selectedCssFile))) {
                    fileContentTextArea.write(writer);
                    JOptionPane.showMessageDialog(frame, "
                        CSS file saved successfully", "
                        Success", JOptionPane.
                        INFORMATION_MESSAGE);
                } catch (IOException ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(frame, "
                        Error saving file", "Error",
                        JOptionPane.ERROR_MESSAGE);
                }
            } else {
                JOptionPane.showMessageDialog(frame, "No
                    CSS file selected", "Error",
                    JOptionPane.ERROR_MESSAGE);
            }

        }
    });

// Listener per il bottone generateWebPageButton
    generateWebPageButton.addActionListener(new
        ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (selectedTxtFile != null && selectedCssFile
                != null) {
                // Salva il contenuto della JTextArea nel
                    file CSS prima di generare la pagina
                    web
                try (BufferedWriter writer = new
                    BufferedWriter(new FileWriter(
                    selectedCssFile))) {
                    fileContentTextArea.write(writer);
                } catch (IOException ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(frame, "
                        Error saving CSS file", "Error",
                        JOptionPane.ERROR_MESSAGE);
                    return;
```

```
}

JFileChooser fileChooser = new
      JFileChooser();
fileChooser.setDialogTitle("Save HTML File
      ");

// Imposta un filtro file per HTML
fileChooser.setFileFilter(new javax.swing.
      filechooser.FileFilter() {
    @Override
    public boolean accept(File f) {
        return f.isDirectory() || f.
              getName().toLowerCase().
              endsWith(".html");
    }

    @Override
    public String getDescription() {
        return "HTML Files (*.html)";
    }
});

int userSelection = fileChooser.
      showSaveDialog(frame);

if (userSelection == JFileChooser.
      APPROVE_OPTION) {
    File htmlFile = fileChooser.
          getSelectedFile();
    String txtFilePath = selectedTxtFile.
          getAbsolutePath();
    String cssFilePath = selectedCssFile.
          getAbsolutePath();
    String htmlFilePath = htmlFile.
          getAbsolutePath();

    // Aggiungi estensione .html se manca
    if (!htmlFilePath.toLowerCase().
          endsWith(".html")) {
        htmlFilePath += ".html";
    }

    // Esegui il parsing e genera la
          pagina HTML
    HtmlHandler htmlHandler = new
          HtmlHandler();
```

```
htmlHandler.startHtml();
htmlHandler.readCssFile(cssFilePath);
htmlHandler.readTextFile(txtFilePath);
htmlHandler.endHtml();

// Controlla se ci sono errori di
      sintassi
if (!htmlHandler.getErrors().isEmpty()
      ) {
    StringBuilder errorMessage = new
          StringBuilder("Sono stati
          riscontrati i seguenti errori
          :\n");
    for (String error : htmlHandler.
          getErrors()) {
        errorMessage.append(error).
              append("\n");
    }
    JOptionPane.showMessageDialog(
          frame, errorMessage.toString
          (), "Error", JOptionPane.
          ERROR_MESSAGE);
} else {
    // Salva il documento HTML su file
    try (BufferedWriter htmlWriter =
          new BufferedWriter(new
          FileWriter(htmlFilePath))) {
        String generatedHtml =
              htmlHandler.
              getGeneratedHtml();
        htmlWriter.write(generatedHtml
              );
        JOptionPane.showMessageDialog(
              frame, "Web page
              generated successfully!",
               "Success", JOptionPane.
              INFORMATION_MESSAGE);
        frame.dispose();
    } catch (IOException ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(
              frame, "Error saving HTML
               file", "Error",
              JOptionPane.ERROR_MESSAGE
              );
    }
}
```

```
                    } else {
                        JOptionPane.showMessageDialog(frame, "
                            No files selected", "Error",
                            JOptionPane.ERROR_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(frame, "
                        Please select both TXT and CSS files"
                        , "Error", JOptionPane.ERROR_MESSAGE)
                        ;
                }
            }
        });


        frame.setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new FileChooserApp();
            }
        });
    }
}
```

9 Considerazioni

- **Inizializzazione dei Percorsi dei File:** Assicurarsi di inizializzare `txtFilePath`, `cssFilePath`, e `htmlFilePath` con i percorsi corretti prima di eseguire l'applicazione.
- **Dipendenze:** Questo codice richiede ANTLR e i file di grammatica generati (`grammatica_completaL` e `grammatica_completaParser`) per funzionare correttamente.

10 Funzionamento del Progetto

Il progetto funziona come segue:

1. **Inizializzazione**: `ParserLauncher` riceve i percorsi dei file di input (file di testo e file CSS) e del file di output (file HTML).
2. **Parsing**: ANTLR utilizza la grammatica definita per analizzare il file di testo.
3. **Generazione HTML**: `HtmlHandler` crea il contenuto HTML basato sui dati estratti dal file di testo.
4. **Output**: Il contenuto HTML viene scritto nel file di output e il file CSS viene copiato nella stessa directory.

11 Esecuzione del Progetto

Per eseguire il progetto è necessario avere i percorsi corretti per i file di input e di output. Questi percorsi vengono passati al costruttore di `ParserLauncher` che avvia il processo di parsing e genera l'output HTML.

Il progetto "Compilatore HTML per Schede Libro" fornisce un modo efficiente e automatizzato per generare pagine HTML contenenti informazioni su libri a partire da un file di testo strutturato. Utilizzando ANTLR per il parsing, il sistema è in grado di gestire facilmente diverse strutture di input e produrre output HTML ben formattati e pronti per la pubblicazione.