Software Specification Document

for

iText Based PDF Viewer

Version 1.0

Prepared by

Dhruvkumar Patel

Jace Robinson

Pranav Pranav

CS 7140 Advanced Software Engineering

July 13th 2016

Table of Contents

1. Introduction

This is a software specification document for the product, *iText Based PDF Viewer*. This document will significantly refer to the requirements document of the same product [6].

1.1 Purpose

The purpose of this document is to completely describe and define the specification for the *iText Based PDF Viewer* product. These specifications must meet the requirements in the previous document [6]. This application will allow users to view and annotate files following the Portable Document Format (PDF) version 1.7 [4]. The viewer will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

iText Based PDF Viewer will be developed using the free and open source library iText [1]. This extensive library can assist in manipulating the PDF internals while using the Java programming language. As a classroom project, the requirements, specification, design, implementation, and testing documents will all be created for this product.

1.2 Intended Audience and Reading Suggestions

There are several types of readers who may view this document. Below are various categories of users along with suggested sections to read.

Developers:
A developer will be interested in learning the specifications necessary to create the product. This group is suggested to read sections 2.2, 2.5, and all of section 3. Section 2.2 will give a high level summary of the functionality, section 2.5 will discuss important constraints, and section 3 lists the formal specifications.

Project Manager:
A project manager will want to understand a bigger picture view of the product in order to manage the work. He or she is encouraged to read all of section 2. This will give a big picture view of the product, users and constraints.

Users:

Users of the product may read this document for a number of reason such as wanting to know some motivation for the product as discussed in section 2.1, or a list of functionality in section 2.2.

Students:
Students may read this document as an example of requirements document. For educational purposes, it is suggested the student reads the entire document to gain the most understanding.

1.3 Product Scope

The goal of this project is to develop a Java and iText based PDF viewer. The viewer should be able to open files following the PDF format. The user should also be able to annotate the PDF, and save these changes with the file. For more detailed description of functionality see sections 2.2 and 3.

As this is an educational project, the scope will be severely limited when compared to a professional product. The primary focus of the project is to develop the documentation for a software life cycle of requirements, specification, design, implementation, and testing. The product will not allow replacement of text at the level of lines, words, and paragraphs. The viewer is only expected to display textual content. This means the product will not be able to display vector graphics, raster images, or any other types of content commonly stored in PDF. The product will not support hand drawn annotations such as created by a stylus pen on a tablet device.

The product must be built using the iText library to handle transfer of information between PDF and this viewer. iText was chosen due to size, popularity, and quality of user documentation. Some of the other choices considered were jPod, PDFBox, and ICEpdf. An investigation comparing and contrasting the various PDF libraries was not performed and the choice of iText should not imply higher quality.

1.4 References

[1] "iText Developers", *Developers.itextPDF.com*, 2016. [Online]. Available: http://developers.itextPDF.com/. [Accessed: 15- Jun- 2016].

[2] "Tracker Software Products :: PDF-XChange Editor", *Tracker-software.com*, 2016. [Online]. Available: https://www.tracker-software.com/product/PDF-xchange-editor.

[Accessed:15- Jun- 2016].

[3] "XODO PDF Reader & Annotator", *Xodo.com*, 2016. [Online]. Available: http://xodo.com/. [Accessed: 15- Jun- 2016].

[4] *PDF Reference*, 6th ed. Adobe Systems Incorporated, 2006.

[5] "Doxygen: Main Page", *Stack.nl*, 2016. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/. [Accessed: 16- Jun- 2016].

[6] D. Patel, J. Robinson, and P. Pranav, “Software Requirements Document for iText Based
PDF Viewer”, June 2016.

1.5 Document Terminology

“PDF File” - For the purposes of this Requirements Document, “PDF file” refers to a file format in compliance with the standards defined in the PDF reference document version 1.7 [4].

"text" - For the purposes of this Requirements Document, "text" refers to "Text Objects" described by Section 5.3 of the PDF reference document [4] with the added restriction that ”text" is composed solely of sequences of Latin characters irrespective of font specified in the PDF for display of the characters. Latin characters are defined in Appendix D.1 of the PDF reference [4].

"annotation" - refers to a subset of annotations described in section 8.4 of the PDF reference document [4]. The subset chosen are underline, highlight, strikethrough, rectangular box, and comment. Table 8.28 of the PDF reference [4] describes rectangular box, and table 8.30 of the PDF reference [4] describe underline, highlight and strikethrough, and table 8.23 describes comment. A visual example of each annotation is given in figure 3.2.1 and figure 3.2.2 of this document.

“upload” - refers to the display of only text and annotations encoded within PDF documents selected from within a file system by a user. The text will display to the output monitor device of the computer.

“save” - refers to storing PDF documents with user selected file names at user selected file system locations.

“insert” - “insert” refers to adding content to the current state of the PDF document.

“delete” - refers to removing content from the current state of the PDF document.

“location” - refers to the position in the grid defined in section 4.2 of the PDF reference document [4].

“search” - refers to matching input text with all occurrences of text contained in the current page of document such that textInput = textDocument. The locations of the matching text within the document are determined.

“PDF page” - refers to a grouping of PDF content as defined in section 3.6 of PDF reference document [4].

1.6 Definitions, Acronyms, and Abbreviations

JAR	Java Archive
JDK	Java Development Kit
PDF	Portable Document Format
UI	User Interface

1.7 Figures

2. Overall Description

2.1 Product Perspective

As discussed, in section 1.3, the goal of this project is to develop a Java and iText based PDF viewer where the user should also be able to annotate the PDF, and save these changes with the file. And as discussed, in section 1.1, the project will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

As a class project, will be limited to a proof of concept. It will be written in Java 8 and JavaFX will be used for its front end. Exploiting the iText library, the ability of extracting texts and adding annotations will be required. See sections 2.2 and 3 for more details.

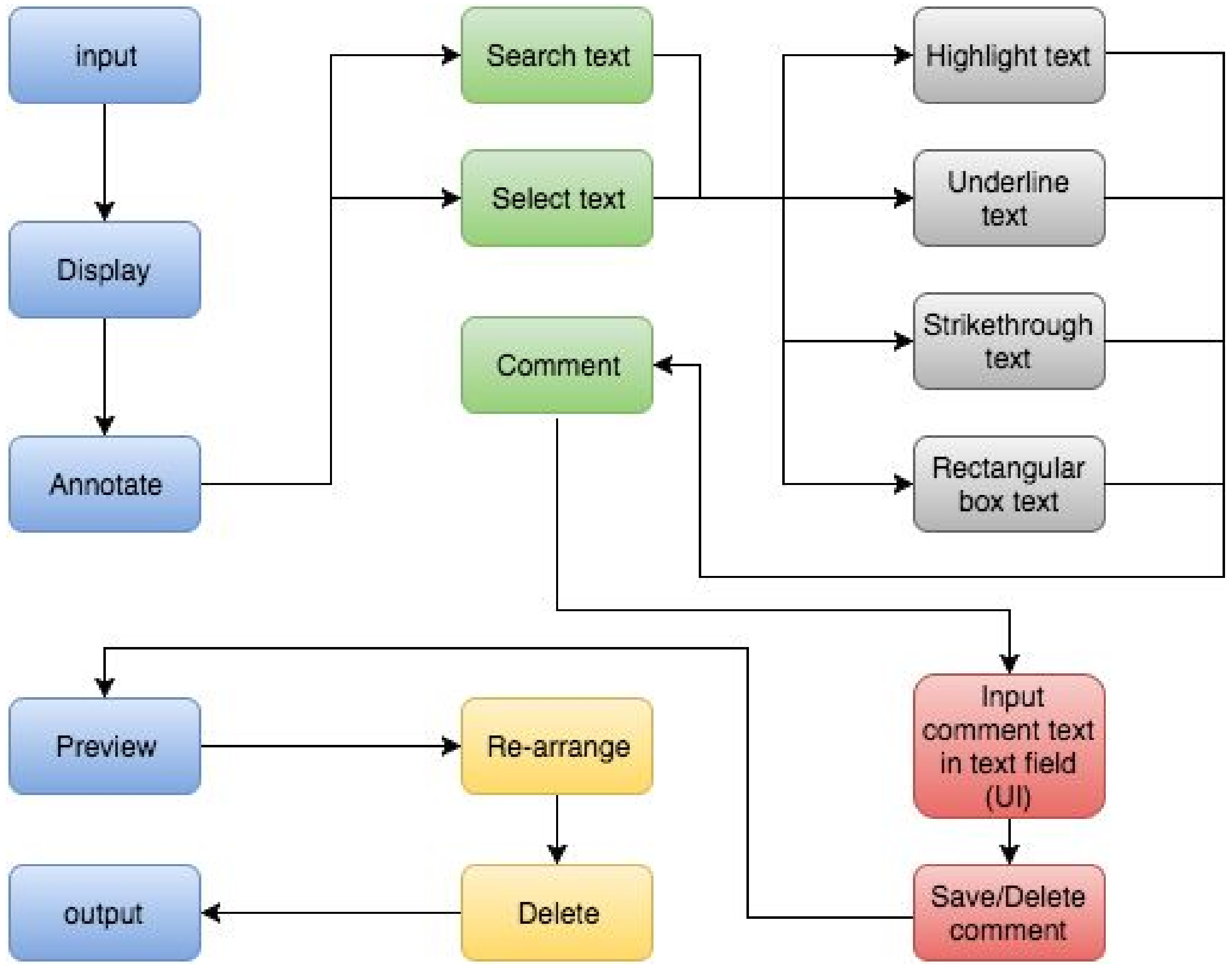


Figure 2.1.1: Work flow of the product.

As seen in figure 2.2.1, all blue color boxes are the basic functionalities of the product. Input, display, annotation, preview and output are discussed in more detail in following sections [2.2]. All green color boxes are sub functionality of the "Annotation". All grey color boxes are different options for "Select text". All red color boxes are sub functionality of the "Comment". All yellow color boxes are sub functionality of the "Preview".

2.2 Product Functions

Following are the functionalities given by the product

1. Input: User Interface (UI) which accepts input as uploaded file defined by PDF [4].
2. Display: UI to display the uploaded PDF in a way that content of the given PDF are not editable at the level of lines, words, or paragraphs, but to be viewed as same. Further the display will be used to annotate text as well.
3. Annotate: UI in the display will be used to annotate text in a number of ways and operations performed given below.
    a. Select text: There are four ways to select text.
        i. Text Highlighting: Mouse cursor will be provided to select and highlight the text
        ii. Under lining: Mouse cursor will be provided to select and underline the text
        iii. Strikethrough: Mouse cursor will be provided to select and strikethrough the text
        iv. Rectangular Box: Mouse cursor will be provided to select and box the text
    b. Comment: operations can be done as part of the annotation to the selected text.
        i. Open text field for keyboard input which can be save or delete. No edit strictly. UI should have fresh look and feel.
        ii. Save or delete the comment.
    c. Search text: operations will be created to allow user to search for specific text and display the location of matching text within the document.
4. Preview: An option provided to user to do below operations.
    a. Rearrange: the user will able to re-arrange all the pages of the given PDF pages.
    b. Delete: the user will able to delete the pages he/she wish to. Comments will be also deleted with the deleted pages if any.
5. Output: Save the project including the given preview page(s) and associated comment into a PDF [4] file as output.

2.3 Operating Environment

The application will be developed to work on up-to-date software at the time of the creation of this document. As the software will be built using Java and delivered as a .jar file, to run the software a user must have the Java Runtime Environment (JRE) version 1.8.0_91 or higher installed. The application will be usable with Windows 10 version 1511, and Ubuntu version 14.04.

2.4 Design and Implementation Constraints

There are several constraints affecting the development of this product. The constraints are listed below.

1. The product must be developed in six weeks. Due to this short time frame, the scope has been severely reduced.
2. The software development life cycle will follow a waterfall model. This model has the sequence of deliverables of requirements, specifications, design, implementation, and testing.

2.5 User Documentation

The product will be delivered with two manuals.

1. The first manual will describe the source code. This is intended for developers and not for users. The document will be created using *Doxygen* to describe the source code and display class diagrams [5].
2. A second document will be created to describe all of the features available to the user. This non-technical document will provide written tutorials on how to open a PDF, search for text, create annotations, and save PDF.

2.6 Assumptions and Dependencies

There are a few critical dependencies for this product that are listed below.

1. The viewer is developed to display PDF version 1.7 formatted files [4]. It is assumed that future versions of PDF will remain backwards compatible with this version.
2. The viewer is developed using free and open source library developed by iText. It is assumed the library remains free in the future.
3. The viewer is to be developed using JavaFX available in Java 8. It is assumed current and future JRE's past version 1.8.0_91 will be backwards compatible and execute the deliverable .jar file.
4. As a test to the saving and uploading of PDF documents, the PDF viewer XODO [3] will be used. A requirement defined in section 3 is a PDF file created in this product must be

viewable in XODO. It is assumed that XODO will continue to be available during the development process. It is also assumed XODO will continue to follow PDF standards.

3. Specifications

The iText Based PDF Viewer product can be specified as separate components. The five components are the PDF Document, PDF Input-Output, Display, Toolbar and Cursor and Keyboard. Specifications will reference the numbered requirements from the requirements document [6].

3.1 PDF Document

First we will define some of the math objects used throughout the specification. The PDF Document is the primary object upon which everything operates.

Define: PDFDoc[1...n] as array of PDFPage where each PDFDoc[i] is the ith PDFPage of the document, 1 <= i <= n, where n is total number of pages (natural number).

Define: PDFCanvas[1...m] as array of elements of type *Text Element* or type *Annotation Element*, such that PDFCanvas[i] is the ith element of the PDFCanvas of type PDFCanvas[i].type == (Text || Annotation), 0 <= i <= m, where m number of objects in the page.

Define: *Text Element* as an object with three attributes of content, position and type, with notation Text.content, Text.position, or Text.type. The content is a stream of characters where the length is >= 0. This stream of characters completely describes both the “characters” and “glyphs” necessary to display to screen as defined in PDF reference [4]. The position is 2D rectangle represented as four number (i,j,k,l) where i,j>=0 and k,l >0. The four vertices of the rectangle in cartesian format would be (i,j), (i+k,j), (i,j+l), and (i+k,j+l). Text.type == Text.

Define: *Annotation Element* as an object with three attributes of annotType, position, and type. annotType can be one of four values, highlight, underline, strikethrough, or box. These annotTypes completely describe how to display to the screen as defined in PDF reference [4]. The position is 2D rectangle represented as four number (i,j,k,l) where i,j>=0 and k,l >0. The four vertices of the rectangle in cartesian format would be (i,j), (i+k,j), (i,j+l), and (i+k,j+l). Annotation.type == Annotation.

Next we will define some basic operations on PDFDoc and PDFCanvas.

3101: When a page number c is removed, all pages before c are unchanged, and all pages after have their page number reduced by 1. This operation meets requirement 3206.

removePage(PDFDoc, c), where c is a natural number.
Pre: PDFDoc[1...n] not null and 1<= c <= n.
Post: n == n.old-1 and and PDFDoc[1...c-1].old == PDFDoc[1...c-1] and PDFDoc[c...n] == PDFDoc[c+1...n.old].

3102: To add an annotation to canvas of a page, simply add the annotation element to the existing canvas of page array without changing other elements.

addAnnotation(PDFCanvas, AnnotationElement)
Pre: PDFCanvas[1...m] not null and AnnotationElement not null
Post: m == m.old + 1 and PDFPage[1...m-1] == PDFPage[1...m-1].old and PDFPage[m] == AnnotationElement

3103: To remove annotation on canvas of a page, simple remove the existing annotation element from the canvas of a page.

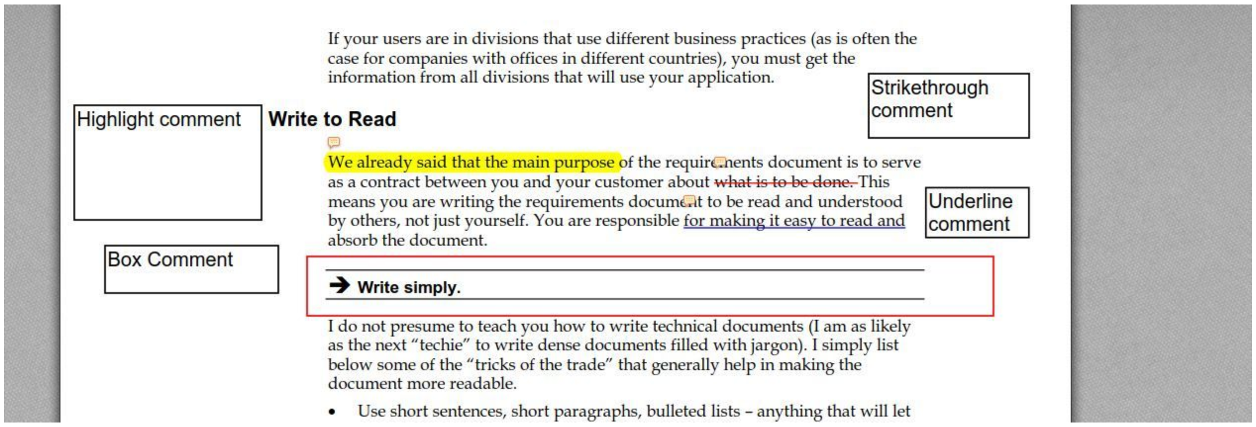
removeAnnotation(PDFCanvas, AnnotationElement)
Pre: PDFCanvas[1...m] not null and AnnotationElement in PDFCanvas
Post: m == m.old - 1 and PDFCanvas[1...m] == PDFCanvas[1...m-1].old and PDFCanvas[1...m] != AnnotationElement and PDFCanvas[1...m.old].old == AnnotationElement

3104: If keyPhrase in PDFCanvas, return position of matching keyphrase.

searchText(PDFCanvas, keyPhrase) where keyPhrase is a sequence of characters
Pre: PDFCanvas not null and keyPhrase not null
Post: ((keyPhrase == substring(PDFCanvas[i].text) and position == PDFCanvas[i].position for some i) || (keyPhrase != substring(PDFCanvas[1...m].text)) && PDFCanvas == PDFCanvas.old

The specifications 3102-3104 meet requirements 3202-3204.

In figure 3.1.1, we can see a specification by example of the “look and feel” of each type of annotation.



If your users are in divisions that use different business practices (as is often the case for companies with offices in different countries), you must get the information from all divisions that will use your application.

Write to Read

We already said that the main purpose of the requirements document is to serve as a contract between you and your customer about ~~what is to be done.~~ This means you are writing the requirements document to be read and understood by others, not just yourself. You are responsible for making it easy to read and absorb the document.

➔ **Write simply.**

I do not presume to teach you how to write technical documents (I am as likely as the next "techie" to write dense documents filled with jargon). I simply list below some of the "tricks of the trade" that generally help in making the document more readable.

- Use short sentences, short paragraphs, bulleted lists – anything that will let

In Figure 3.1.1 the "look and feel" of the four types of annotations are shown. The *highlight* annotation is shown by the yellow background surrounding the phrase "We are said that the main purpose". The strikethrough annotation is shown by the red line through "what is to be done". The underline annotation is shown in blue on the phrase "for making it easy to read and". The box annotation is shown as the red box surrounding "Write simply.". Lastly for each type of annotation, there is a comment, which contains text provided by the user.

3.2 PDF Input-Output

PDF Input-Output are the components of "uploading" a document that satisfies PDF reference [4] and "saves" a document that satisfies the reference.

Define: *isPDFCompliant(file)* as a boolean test of whether the bytes read from any file satisfy PDF reference [4].

Define: *textEqual(file1,file2)* as a boolean test of whether all the text elements as defined in section 3.1 of this document for file1 exist in file2. Notice that file2 may have more text elements than file1.

3201: isPDFCompliant(input) and isPDFCompliant(output), where input is input file before uploading to software, and output is output file after save operation.

3202: upload(inputFile)
Pre: true
Post: isPDFCompliant(inputFile) || (NOT isPDFCompliant(inputFile) and exit program)

3203: textEqual(output,input). As stated in the requirements, pages of the PDF can be deleted, but not added.

These specifications meet requirements 3201 and 3205-3208.

3.3 Display

The *display* is the finite 2-dimensional grid where the PDFDocument is visible to the user.

Define: grid[1...max_x][1...max_y] as 2-dimensional array such that grid[i,j] contains either a Text element, Annotation element, or is empty. If the grid[i,j] contains a Text element, display the content of the element. If grid[i,j] contains an Annotation element, display the annotType of the element. If the grid[i,j] is blank, display white background.

3.4 Toolbar

The types of components (save button, display field, etc…) are the only detail from the figure to be interpreted as formal specifications. The positioning of the components in the figure are not to be taken as specifications. The visual appeal of the figure is also not the be taken as specification. These specifications meet the requirements 3101-3112.

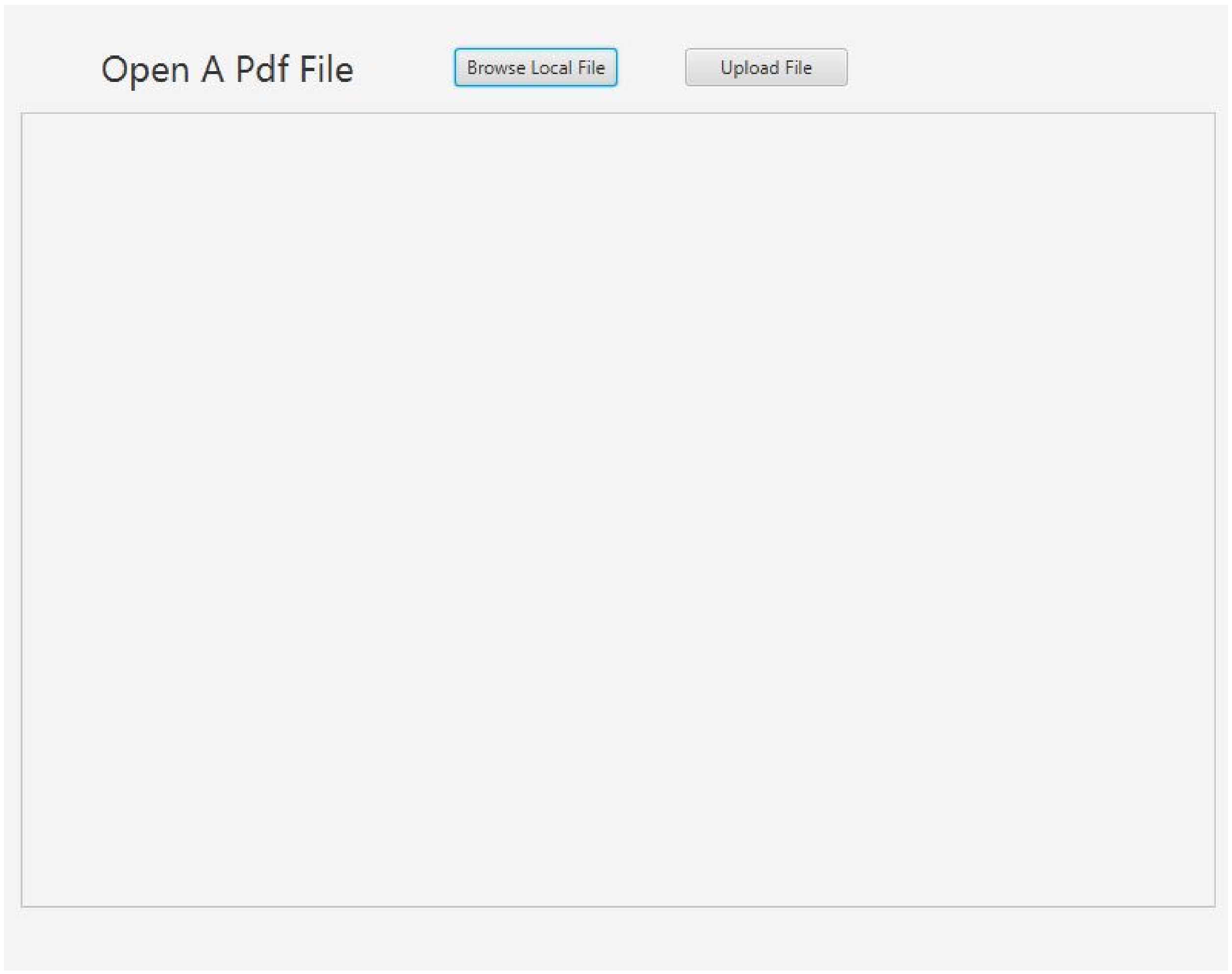


Figure 3.4.1: screen 1: browse, upload button and display content.

A second screen will contain buttons save, annotate, search, and split functionality button.
Screen 2:- save,annotate, search, split functionality

3.5 Cursor and Keyboard

The user is able to interact with the software through a cursor and keyboard.

The cursor is a device that allows the user to interact with the grid and toolbar components as defined in sections 3.3 and 3.4 of this document. The cursor when interfacing with the display, will be located at a specific (i,j) location corresponding to grid[i,j]. Define an event as an action taken by the user. An event can be used to invoke a response at location (i,j). When interfacing with the toolbar, the cursor will be located at a specific button. When an event is started by the user, if the cursor is located at a button, the button event begins.

The keyboard is a device that allows character input. A sequence of characters can be collected from sequential use of the keyboard.

3.6 Additional Specifications

Requirements 3301-3303, 3401-3403, and 3501-3502 are already unambiguously specified in the requirement document and are therefore not mentioned in this document. Requirement 3503 has been removed now and in all future documents.

4. Author Journals

Below are the journals of each of the authors maintained throughout the development process.

4.1 Dhruvkumar Patel

Start Day 7-11-16

we decided to implement specification with the same template of document what we used for Requirement documents.
I learned lifecycle of software engineering system how the system is build? in real world. As of now what i have developed in my career i just start with the
analysis and then dirrectly move on implementation part.

Start Day 7-12-16
we start with specification document and discussed all the math logic, pre and post condition of all the objects used throughout specification document.
also start to think about design of the system we dicussed classes, attributes and operations of the system.
designed UML and Data Flow Diagram using Microsot Visio tool.
learned how to design wwhole system and documents.
meanwhile we started coding to make product and whcih will be helpful to write implementation documents.
discussed design of all the features what we mentioned in the requirement and specification document.
start writing specification and design document.

Start Day 7-13-16
We had a difficulty to display pdf in javafx scrollpane but, then piazza discussion and talked with jace and pranav. Atlast we reached the solutions.
we all divided the section of both documents and we did within time limit.

4.2 Jace Robinson

Start Day 7-11-16
Today we are beginning the specification document. The first task is to find and understand an appropriate template. I am confident in our requirements document so now it is a matter of writing a spec which meets this. We decided to use basically the same template as requirements but create specifications that meet each requirement.

Start Day 7-12-16
Today we are beginning the design document. We have a rough outline of the main components of our viewer.
A pdf is a collection of objects completely describing the content. Specifically, there exist *text objects* which completely describe the text (this includes font).
PDF *Document Structure* describes pages, fonts, and annotations.
3.4 File Structure contains body which make up document. Sequence of indirect objects
File Structure
How is text stored?? Text is font, position, and characters (text objects)
Glyphs are organized into fonts
Character is abstract entity such as “A” (and not its font)
How does text “wrap” and not extend past border of page?? Using matrix operations to determine “text space”

Annotations
Text Markup Annotation (highlight, underline, strikethrough), 4 (x,y) pairs describing coordinates of rectangle to markup (text will be in the rectangle).
Square and Circle Annotation – rectangle is represented as difference of two rectangles, along with their positions
Annotation are separate objects from the text, they are simply given the same coordinates.
How are pages determined? A page is a collection of content stream
Entire pdf is collection of pages (with other stuff but not relevant to us)
Now looking at iText
Relevant Classes/Functions:
Canvas: Where content is placed, does not know what page it is
Document: root object
DocumentRenderer:
LayoutPosition:
Paragraph: self contained block of text
IElement: takes physical space on canvas
PDFAnnotation
PDFArray
PDFDocument
PDFMarkupAnnotation
PDFPage
PDFSquareAnnotation
PDFTextMarkupAnnotation
Rectangle
Examples and APIs worth keeping track of:
com.itextpdf.kernel.pdf.annot
com.itextpdf.kernel.pdf.canvas
TextMarkup example (with highlight)
http://developers.itextpdf.com/content/itext-7-jump-start-tutorial/chapter-4-making-pdf-interactive
TextExtractionStrategy() to read text from pdf
http://developers.itextpdf.com/examples/content-extraction-and-redaction/parsing-pdfs
PDFStamper to use modify/create new pdf from existing pdf
Start Day 7-13-16
First task is to continue writing specification. I have decided to overhaul a large portion of our existing specification to give more rigorous “math” solution. The main idea to array of pages, where each page is array of text and annotations. The grid is a two-grid. All other operations are built around these basic ideas.
There were some difficulties writing the specification for how the content is “displayed” but a reasonable solution was met.

4.3 Pranav Pranav

Start Day 7-11-16
Group meeting 1: Decide the template and break up that task.
From old experience, defined specs for each of our requirement.
Pre, post and relation are defined for all 13 requirements.
This gives us comprehensive idea of how to convert requirement into specs.

Start Day 7-12-16

Group meeting 2: Discuss and review among us shortcoming of the specs document.

Start Day 7-13-16

Review section 3 and did changes required.