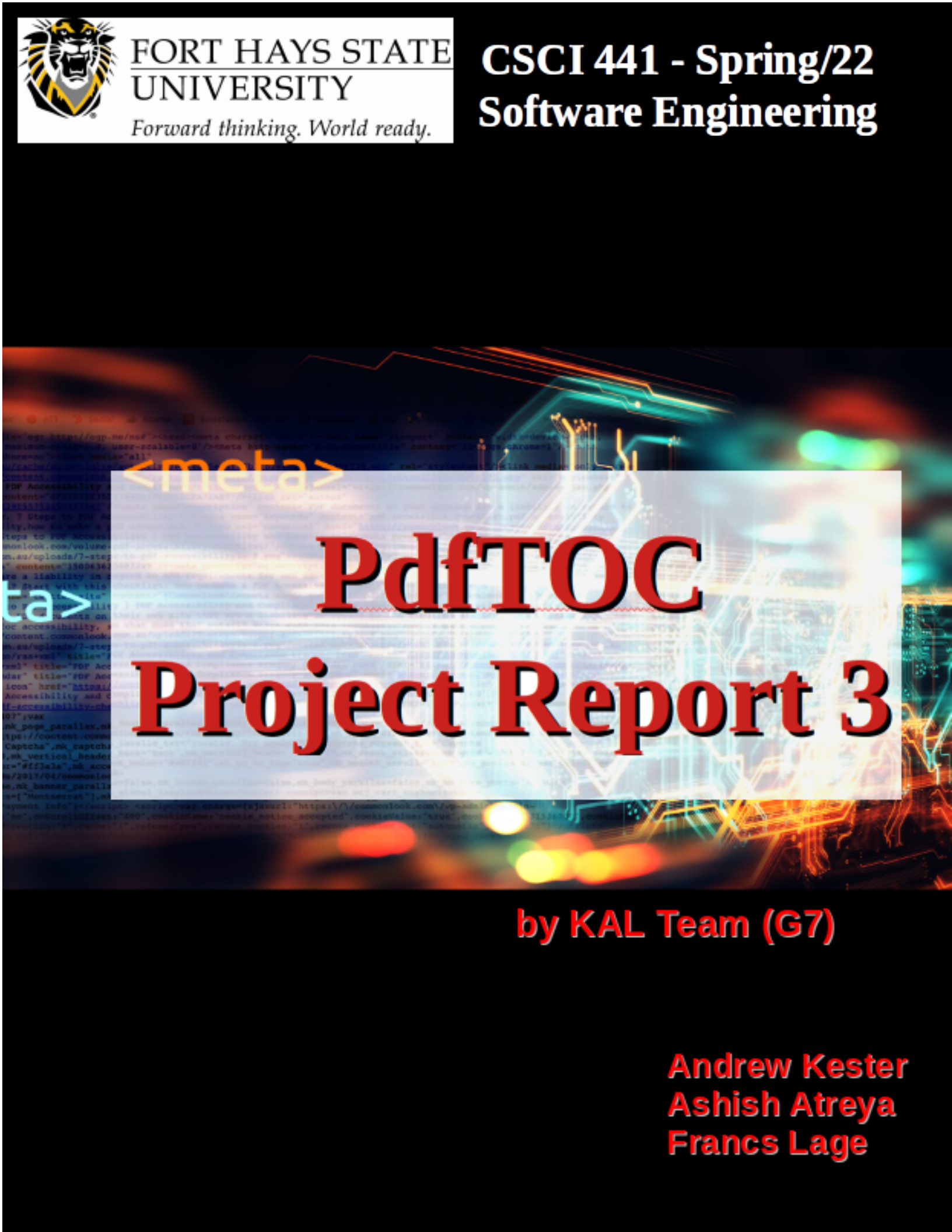

FORT HAYS STATE UNIVERSITY
Forward thinking. World ready.
CSCI 441 - Spring/22
Software Engineering
PdfTOC
Project Report 3
by KAL Team (G7)
Andrew Kester
Ashish Atreya
Francs Lage

FORT HAYS STATE UNIVERSITY

Forward thinking. World ready.

CSCI 441 - Software Engineering

Professor: Dr. Shukri Abotteen

PdfTOC

Project Report 3

By KAL Team (G7)

Andrew Kester

Ashish Atreya

Francs Lage

Table of Contents

Summary Of Changes

Customer Problem Statement

The Customer problem statement has not been modified much since the start of the project. The customer's problem was clearly and well defined initially and the focus of the project has remained true to the problem at hand.

Glossary of Terms

The glossary of terms has been updated with new terms. Terms that no longer apply have also been removed.

System Requirements

The system requirements have been updated to remove requirements that were no longer in line with the functional requirements, such as allowing users to OCR documents.

Functional Requirements

This section has been updated to highlight work that is anticipated to be done by the final demonstration.

Effort Estimation

This section was updated to focus on development effort rather than user effort and utilize the productivity factor and use case points.

Class Diagrams

The class diagrams and class descriptions have been updated with further implementation details and additional details regarding internal structure.

Algorithms and Data Structures

Data structures that have been identified during implementation have been added.

User Interface Design

Images of current user interface implementation added.

1 Customer Problem Statement

The PDF file format is becoming more and more ubiquitous in today's paperless culture. These files allow for users to quickly and easily exchange documents while preserving formatting and allowing for advanced features such as organization, searching, and filling in forms.

PDFs have become so commonplace that many office suites export to PDF files natively, and many scanners allow users to scan physical documents into a PDF file. The features of the platforms creating these files vary widely, from a simple export button to a complex system that exposes many of the features PDFs can offer.

Alongside creating PDFs, almost all major platforms (and many web browsers) have methods to view PDF files without the need for any additional software. At one point, PDF files required a specific software viewer published by Adobe (the originator of the PDF standard) to view, limiting their use. With the added ease of viewing PDF files, their usage has become even more commonplace and easy.

With the proliferation of these file types, many individuals and organizations are presented with the task of organizing and storing these files. Since PDF files have internal data features that are designed to help users organize these files, files that take advantage of these features can be much easier to organize. Specialized software tools exist to help organize these files as well, but again the internal features of the PDF must be utilized in many cases to let the software operate as intended.

One key problem, however, is that many PDF files don't take full advantage of the features available within the PDF standard. These features can be added when the document is created, or added later using a separate program. Many of these features may not be immediately obvious to a user, so they may not notice if they are missing.

As an example, the PDF standard allows for an internal table of contents, called "bookmarks" to be present in a file to aid in navigation. These bookmarks are different from the Table of

Contents included in many larger documents, as rather than being a series of pages or rows, these are stored internally in the PDF document.

If a file has these bookmarks, the viewer frequently will display these on the side of the document in a sidebar for the user's ease of use. Since these typically mirror the headings and the table of contents already, it is a great use to navigate large documents quickly.

If a document lacks these bookmarks, a user is forced to manually search or scroll the document to a particular section. Once a document becomes quite large, this can be cumbersome and inefficient. This may seem mundane, but as documents become larger and its use becomes more frequent, the inefficiencies will slowly add up to become an incredible burden.

PDF files are commonly used for many long-form documents, such as books and complex reports, which highlights the need for utilizing these internal bookmarks for organization even more. An end user will likely not enjoy scrolling hundreds or thousands of pages to find sections of interest for them, and repeated access would only cause more frustration to an end user.

A solution for this is to add bookmarks into existing PDFs, but then a new issue is presented: how to accomplish this easily? There are tools, but many require usage on a command line interface, which makes them inaccessible to non-technical users. Other solutions too may no longer be maintained, causing compatibility and security issues as well.

Another issue is some of these solutions require software to be installed on the user's computer, which now creates issues with compatibility and security. If a user is in a large IT environment, they may not have access to install new software onto their computer. Likewise, software they want to use may not be compatible with the platform they are using.

The ideal solution would be two-fold: allow users to use the software from any device and provide a GUI for the users who may not be as technical. This project solves both of those by proposing a web service that will allow users to add bookmark metadata to PDFs.

At the heart of the service is a program called PdfTK, which is a command line program designed to accomplish many tasks related to editing PDF files. PdfTK doesn't solve one key issue though: since it's a CLI program it is not suitable for many end users.

The web service will allow an end user to upload files, define bookmarks using a graphical interface without the need for any advanced technical knowledge, and make the changes to their file.

With the web service, a new set of options and features also become available. Users would be able to save partially completed files to their account and work on them in multiple sessions. This is crucial if the file is large (such as a book), since the task of editing metadata may take some time.

Additionally, other features and services can be added to allow for the processing of other PDF files. As another example, many scanners produce a file with no searchable text embedded in the file, as each page is stored as the raw image of the scanned paper. Adding features to allow OCR processing for files would be simple to add, but add great value to end users.

In the future, even more features to allow for advanced PDF editing could also be added to make PDFToc a full-featured PDF editing platform, something that is also becoming more crucial to many users. In order to keep the scope attainable, though, all of these advanced editing features will not be included in the initial project scope.

2 Glossary Of Terms

Bookmarks - A type of metadata in PDF Files that acts as a table of contents. These store the location of sections within the document and can be displayed by PDF viewers to aid in navigation.

GUI - Graphical User Interface, interfacing with a program using a mouse and keyboard and using images. This is the most common way to use software for non-technical end users.

Index table - Similar to a Table of contents, this stores locations of sections within the document, but rather than doing so on pages it does so using document metadata.

Metadata - Additional information about a file, such as bookmarks, authors, versions, etc, that is stored in an internal format and not typically visible to end users. This data can be added by the program that created the PDF file, or added later by other utilities.

OCR - Optical Character Recognition, a process to convert images of documents into a digital representation of the words and text on the pages.

PDF - Portable Document Format, a standard for document file storage commonly used and originally developed by Adobe.

PDF Editor - A tool to edit an existing PDF file. These can modify the document itself, metadata, or other features of the PDF.

PDF Generator - A software program that creates a PDF file. This can be a standalone program, part of another program (like a word processor), or embedded in a device such as a scanner. These can add metadata, but some support certain PDF features and others do not.

PDF Viewer - A software program that displays the contents of a PDF. Many viewers are available for many computer platforms, and each have a range of features. Many are able to display the internal bookmarks and index table to help users navigate.

PDFTk - A CLI software toolkit that allows modification of PDF files, including their metadata and bookmarks.

Table of Contents - Organization in a document that lists sections and what page they begin on. This can either be a section within the document that is visible to the end user or stored in the file's metadata.

TOC - Acronym for Table of Contents. A digital file has two TOC: one that is within the document,formatted as the rest of the content and the Index TOC that is stored inside the file metadata, they both have the same information. Most of the time in this Report the TOC acronym will be referring to the second one, the digital index of the file.

3 System Requirements

The overall set of problems above can be divided into a series of subproblems that need to be individually addressed.

Allow Users to Create PDF Bookmarks

The key problem highlighted above is allowing users to add bookmarks into the internal structure of a PDF. Without these, the overall navigation through the document can become cumbersome. Since there is a wide variety of tools used to create PDFs, a standalone tool to modify bookmarks would be ideal.

The application must provide a way to get the current metadata, add to it and modify it, then store it into the original PDF file. To aid in this, the PdfTk program will be used. This is a CLI program that performs many of the functions needed, but as mentioned above the usage of CLI programs is out of the knowledge of may end users. The program must therefore provide an interface that is acceptable for most users, but also put the data in a format that PdfTK can use and invoke PdfTK.

Provide a GUI for PDF Modification

One key problem outlined above was that many solutions don't support GUIs and may be challenging or impossible to use for most end users. By using a web-based service, and therefore creating a GUI, non-technical end users will be able to access and use the service.

Allow Users to Work on large PDF files in Sections

Since PDF files can be used for long documents such as books, the time needed to organize the internal bookmarks of files increases. Since users may want to accomplish these longer tasks in installments, the solution will need to handle allowing users to save and resume incomplete progress.

This will require an internal database and data structures to represent an incomplete set of document metadata. The data must then be stored and retrieved from the database as needed.

Handle Authentication and User Sessions

Since a web service is being proposed, allowing users to login to a personalized account to securely manage their files and progress is a new problem that must be overcome. Since the documents are now stored in a shared third-party service, the files must also be assigned to individual users and kept separate from each other.

Allow File Uploading and Downloading

Another new issue presented by utilizing a web based solution is providing the ability for users to upload and download files, as well as store the files while they are being edited and processed. As before, keeping the files private and secure is a key issue.

4 Functional Requirements

These are the requirements from the customer's narrative. Each row has a unique requirement. The requirements will be classified by priority number ranging 1 to 5 that means: Any requirement priority 1 is a nonfunctional requirement and it will be placed on the next table. The requirements 2 to 5 are functional requirements and they will be implemented.
5: is the highest priority, the system doesn't work or doesn't serve a purpose without it.
4: means a very important requirement but the system would be able to work without it.
3: important but can be implemented later.
2: not important, it will be implemented after all the most important aspects have been resolved.

Requirements that are planned to be implemented for the final demo are highlighted. Other requirements will be considered future work for the project.

Identifier	Priority	Requirement
REQ1	5	The System shall allow user to upload a pdf document
REQ2	4	The System shall allow user to download the modified version of the document
REQ3	3	The System shall allow user to login, so partial work can be saved
REQ4	5	The System shall be able to extract the metadata of the uploaded file
REQ5	4	The System shall be able to insert the revised metadata back to the file
REQ6	5	The System shall allow user to edit title bookmarks from metadata
REQ7	2	The System shall allow user to batch edit bookmark's titles
REQ7	5	The system shall allow user to edit page number of a bookmark
REQ9	4	The system shall allow user to do batch edit to bookmark's page numbers
REQ10	3	The system Shall allow user to search for a pattern in the titles list
REQ11	4	The system shall allow user to insert a bookmark title at any position

Identifier	Priority	Requirement
REQ12	3	The system shall allow batch insert of bookmark titles
REQ13	2	The system shall allow user to save bookmark title list as .txt file
REQ14	2	The system shall allow user to save bookmarks as .txt file

Some of these requirements, like REQ15, could be a nice option and it wouldn't require any deviation from the actual course of the project. Some of those requirements, such as REQ20 and REQ21 would change the application to an PDF editor instead of being only a metadata editor.

Identifier	Requirement
REQ15	The system should help user to reduce pdf size. CLI application ps2pdf can be invoked.
REQ16	The system shall do a batch delete of empty spaces in the bookmark titles list
REQ17	The system shall do a batch insert of characters or spaces in the bookmark titles list
REQ18	The system shall be able to remove page numbers from the titles list
REQ19	The System shall be able to add pages to the pdf document
REQ20	The System shall be able to delete pages from the pdf document
REQ21	The system could be able to add pages to pdf document.
REQ22	The System could edit other parts of metadata and not only the TOC

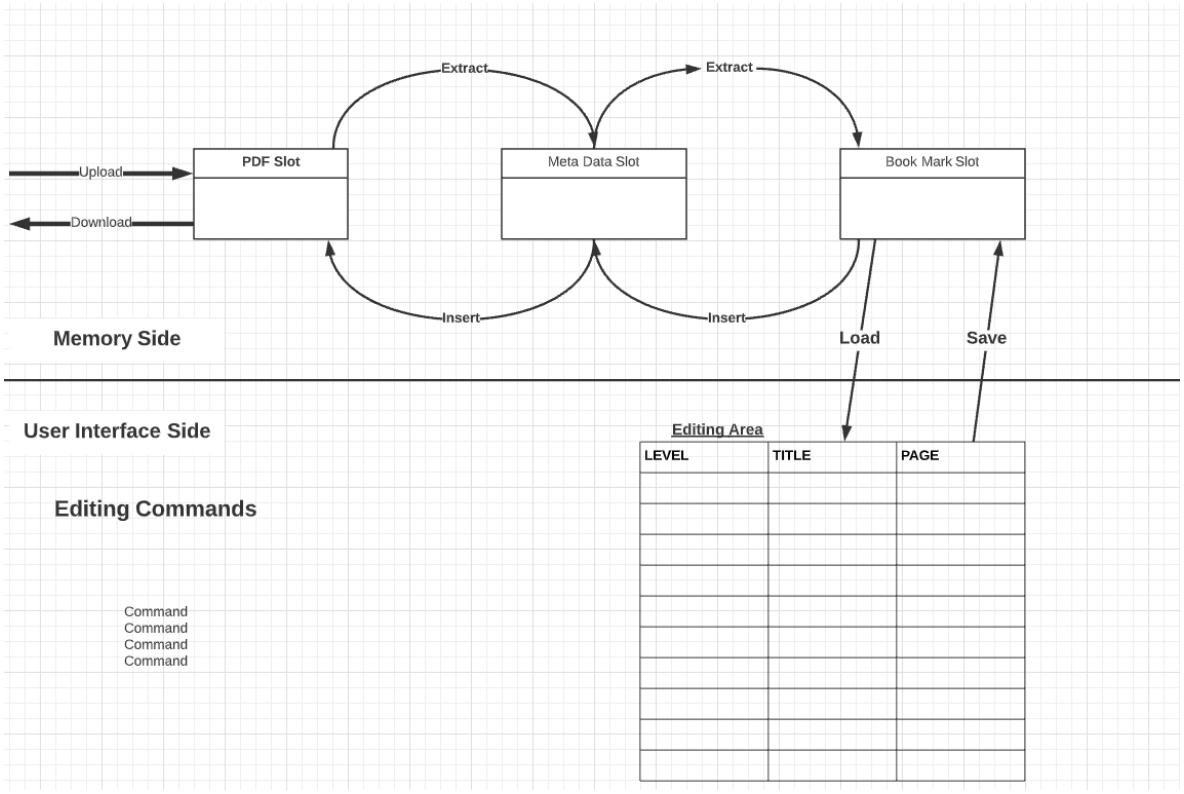
5 Effort Estimation

The expected development effort for key portions of the application are estimated below.

Operation	UCP	PF	Total Effort
Register as new user	0.5	28	**14**
Login	0.5	28	**14**
Upload PDF document	1	28	**28**
Download PDF document	1	28	**28**
Upload Toc	2	28	**56**
Download Toc	2	28	**56**
Operations for editing Toc	3	28	**84**
Operations for Extracting and Inserting Toc on PDF	3	28	**84**

6 Interaction Diagrams

Before starting with a UML sequence diagram, we would like to show a sketch of the general system's view. The below picture illustrates the back side and the interface side of the system.

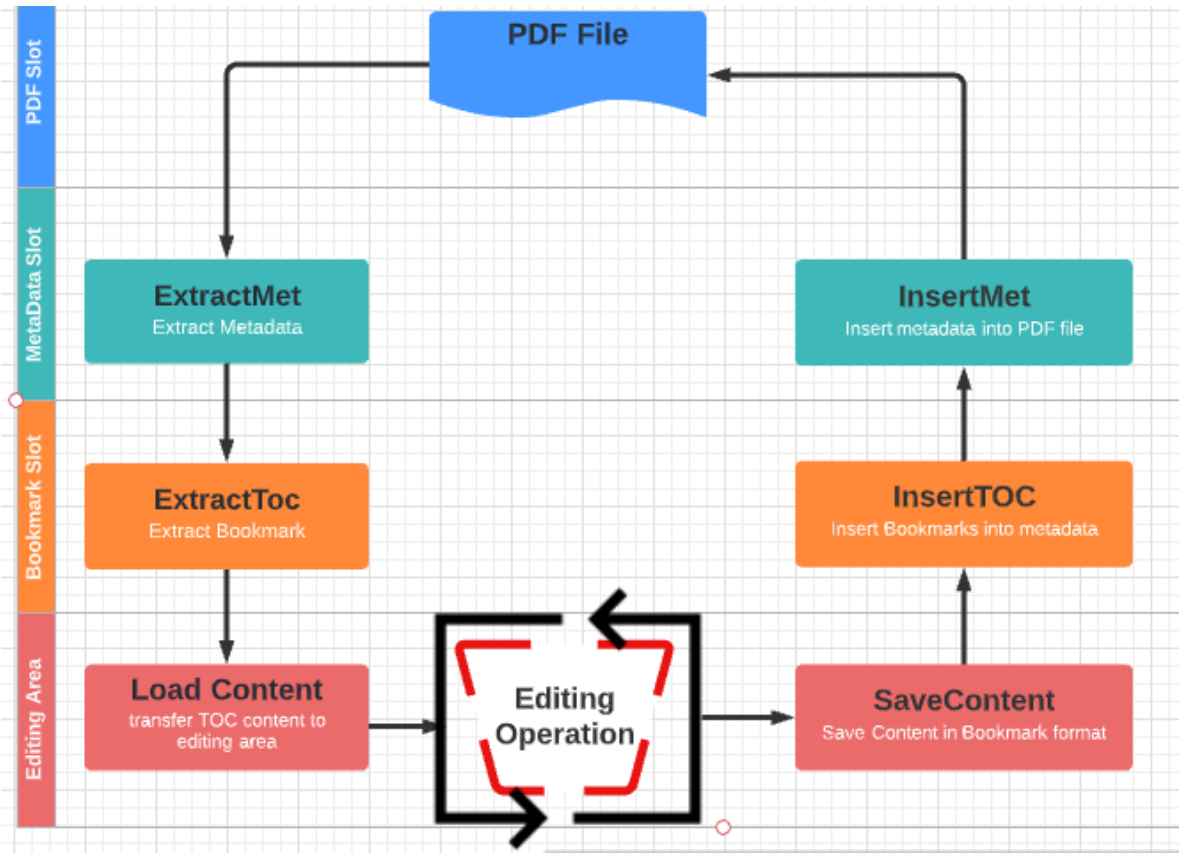


To simplify the sketch, the registration and login aspects of the system were left out. The main system's operation starts with the user uploading a file and ends with the user downloading the updated file. At the editing area, there will be several approaches to get the titles fixed and ready to be saved into the Bookmark slot.

Also it's important to mention that after the file upload, which is the first step of the process, the following two steps (extract metadata from file and extract bookmarks from metadata) can be

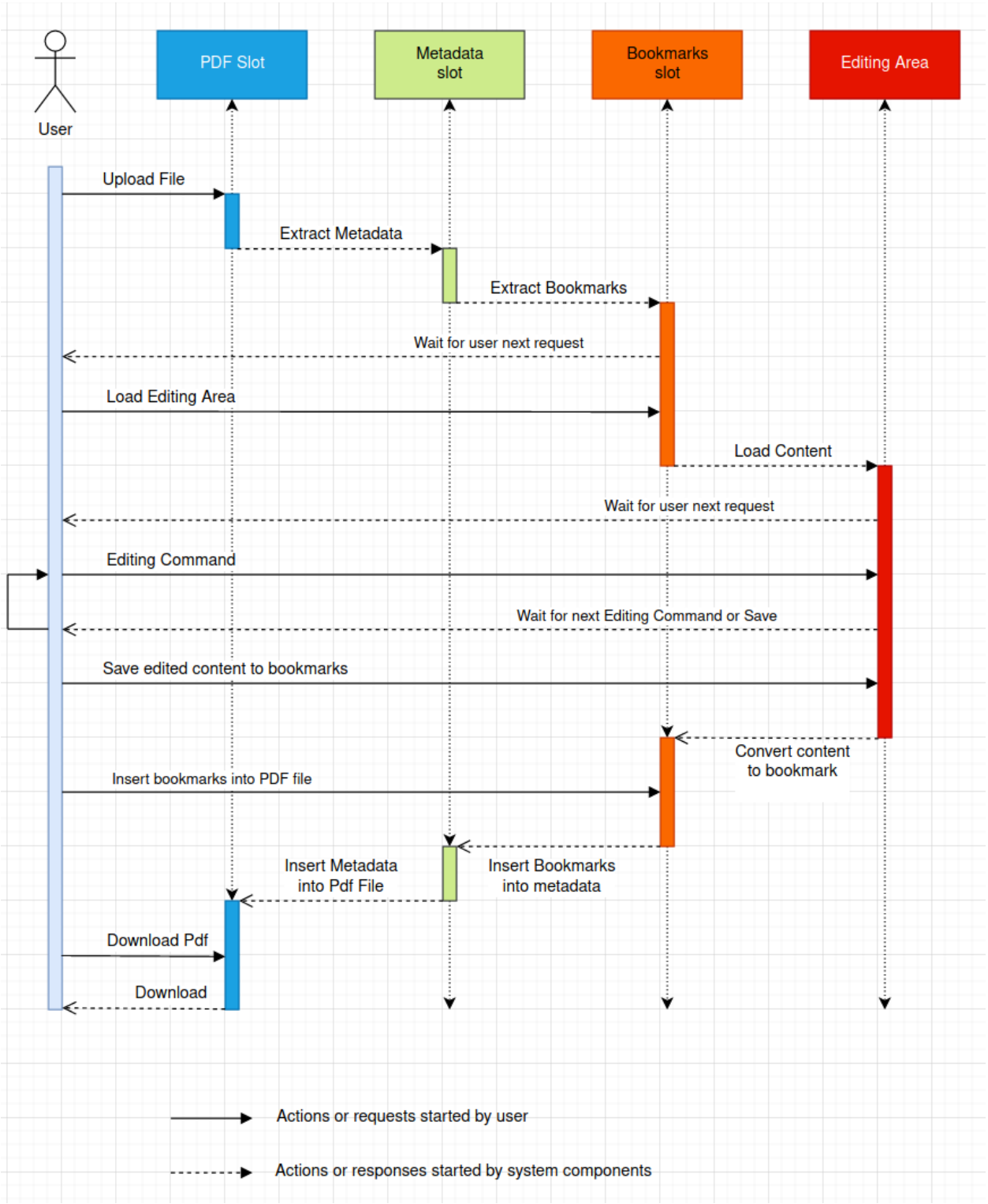
automated. The Loading editing area with the extracted bookmarks must be user-requested, because there will be several possibilities to enter data into the editing area.

The next picture will show a flow diagram of the complete cycle operation. This diagram shows the main use cases. Other use cases are inside the “Editing Operation” and they will be detailed later.



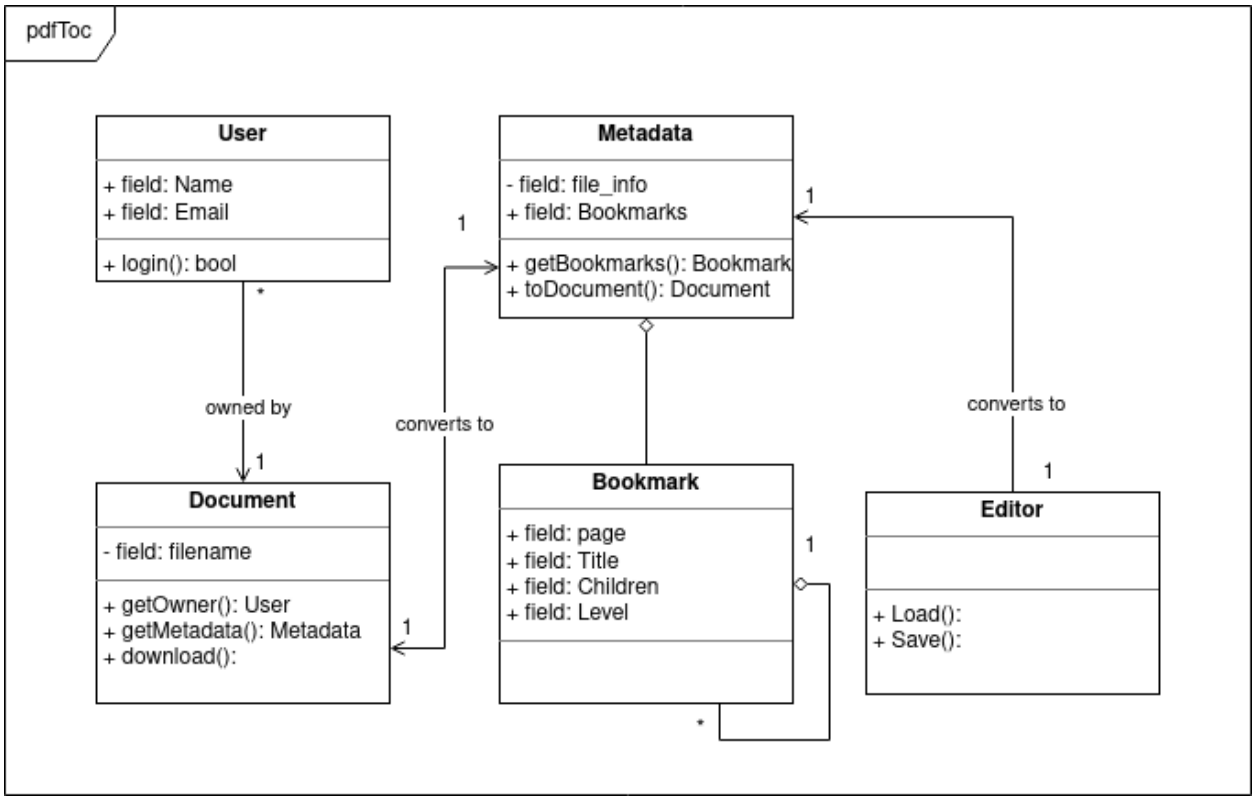
The flow diagram starts after the file being uploaded and it ends with the metadata being inserted back into the PDF file. Each stage on this flow diagram (the squares) represents the arrows in the system draw picture.

The combination of the system draw with the flow diagram gives us enough information to build our UML sequence diagram. The top of the diagram shows the actors: user and main components of the system. The arrows represent the actions and responses from the actors.



7 Class Diagram and Interface Specification

7.1 Class Diagram



The application is broken into 5 major classes: A User, A Document, Metadata, Bookmarks, and an Editor. The relationship between each is linear, with each class being able to convert into its "neighbor" classes. The conversions allowed are Document to Metadata (and vice versa), Metadata to Bookmark (and vice versa), and Bookmark to Editor (and vice versa).

Each class will be responsible for these conversions themselves. Since these conversions are highly dependent on the data within an object, creating an external conversion class is not immediately useful.

7.2 Data Types and Operation Signatures

A user represents the registered and anonymous users utilizing the system. For logged-in users, this stores information needed to process their logins, such as the username and password.

User	
+ Name + Password	User's name. Hash of User's Password.
login(Password): bool getDocuments(): Document	Check if a given password is the correct user password. Get an array of documents owned by this user.

A document is the representation of an uploaded PDF file, and is able to extract metadata from the document in its raw form (no bookmark data is extracted at this stage). Uploading and downloading is also handled by this class.

Document	
- filename + UserFilename	Location where this file is stored internally. The name of the file provided by the user.
getOwner(): User getMetadata(): Metadata download():	Get the user object that owns this document. Extract metadata from this PDF, giving a Metadata object. Prepare this document into a PDF to be downloaded.

The metadata contains the raw information extracted from the PDF, and is able to convert it into an array of Bookmark objects. Adding, removing, and updating the metadata is achieved by modifying the array of bookmarks.

Metadata	
- file_info + Bookmarks	Raw metadata associated with the file. Bookmarks contained in this metadata.
toDocument(): Document getBookmarks(): Bookmark	Convert this metadata back into a Document object. Get bookmarks from this metadata information.

The Bookmark is the representation of a bookmark in the document. It contains the Title, Level and page number. Any nested bookmarks, such as subheadings in a table of contents, are stored in the Children field.

Bookmark	
+ Title + Level + Children + Page	Title of this bookmark. Bookmark Level of this bookmark. Bookmarks that are nested under this bookmark. Page number for this bookmark.
getLevel(): int	Get the number of nesting levels for this bookmark.

The Editor interfaces with the web frontend of the site, and handles the loading and saving of documents. It uses an internal state object that keeps track of the state the interface and editor are both in, and can load and save bookmarks from this state.

The state object is coupled tightly with the frontend, and is directly modified by the editor component on the website. The state will be a JSON representation of the editor's main table data. The frontend will be able to take this data and format it into the visible table and also take the visible data table and format it into the JSON state object.

Further objects were not used here as the frontend framework that is used works natively with JSON formatted data, and parsing the data into and out of the required objects is trivial. To add additional objects and models for this section would add unneeded complexity to the application, development, and testing.

Editor	
- state	State of the editor
Load(Bookmark): Save(): Bookmark	Loads a series of bookmarks into the editor's state. Saves the editor's state into a series of bookmarks.

7.3 Traceability Matrix

The below table illustrates the mapping of Domain Concept with the various Classes listed in the prior section.

	Classes				
Domain Concept	**User**	**Document**	**MetaData**	**Bookmarks**	**Editor**
Register	X				
Login	X				
UploadPdf		X			
DownloadPdf		X			
DownloadTOC		X	X		X
UploadTOC		X	X		
Save					X
extractToc		X	X		
InsertToc			X		
InsertTitle				X	X
DeleteTitle				X	X
UpdatePageNumber				X	X
DeleteTitlePN				X	X
DeleteES					X
AddES					X
DeletePattern					X

8 System Architecture and System Design

8.1 Architectural Styles

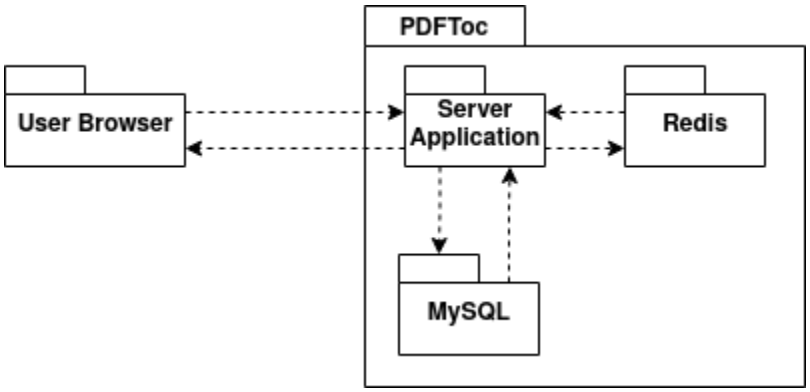
The system will be developed in a client-server type model, as that is the standard model for web based applications. The server portion will be a monolithic application written in PHP and utilize a Model-View-Controller structure internally to accomplish the required tasks.

The framework in use is Laravel, which is designed around an Event-Driven architecture, as the primary driver for code execution are HTTP requests coming from end users. There will also be internal events that will respond to document states, such as processing tasks completing.

Processes for updating documents need to be performed linearly, and this is a requirement that is more or less handled by the user interface and user interaction. As an example, a document must be uploaded before metadata can be modified on that document. The interface is intended to guide a user through the required steps in the correct order, and the server process will verify required steps are completed before allowing actions to complete.

Since the server is event based, part of each event response needs to be spent validating the current state of a document before allowing users to proceed, to ensure the linear steps outlined above are completed in a reasonable order. While the UI should prevent out-of-order steps, bugs or malicious requests could cause the steps to be requested incorrectly.

8.2 Identifying Subsystems



The PDFToc application will utilize a server application, the monolithic PHP application that handles requests from users, as well as a MySQL/MariaDB[1] server and Redis Key-Value store. The user's web browser will communicate with the server application, and the server application in turn will communicate with both the external services as needed.

8.3 Mapping subsystems to Hardware

The client component will be any standard web browser and run on the individual users' computer. The main server application will run on a centrally hosted web platform. A MySQL/MariaDB database and a Redis Key-Value store are also utilized by the server application, and are hosted on the web platform as well.

The main server application, MySQL, and Redis will all be run in a web hosting platform that allows these services to exist in an isolated area on a larger shared system. These systems will run in individual containers, but since things are effectively isolated and interconnected, they logically exist in one central platform.

8.4 Persistent Data Storage

There will be 2 persistent data stores for the application: A MySQL database and flat file storage. The database will be used to store structured data, such as user information,

[1] MariaDB is a drop-in replacement for MySQL chosen for this project. These programs are effectively the same, but differ in some licensing areas. For clarity and compatibility with other documents, the term MySQL is used in this document, but is in fact referring to MariaDB.

document information, etc. The schema for each table is managed by a set of migration classes within the application that perform any updates needed to the database structure as the application is designed and built.

The flat file storage will be used to store user documents as they are processed by the system. These will be normal PDF files stored in a shared directory, with each file mapped to a document by an entry in the structured database. The files will simply be stored in a single directory, each being given a unique, generated name.

A Redis Key/Value store is also used, but this is not persistent. It stores data related to sessions that are in progress, such as login cookies, but the data is stored only in memory and cleared automatically. Being a Key/Value store, data sent to Redis is unstructured, but is data is assigned a unique key that can be used to retrieve the data later.

8.5 Network Protocol

Data coming to and from the users will be sent over HTTP/HTTPS. Since PdfToc is intended to be used as a web-based application, this is a requirement to ensure web browsers are able to use the system.

Internally, data will also be transmitted between the main server process and the Redis and MySQL services. These services run in an isolated network space provided by the hosting platform, and communication is done via standard TCP sockets. The exact protocol used is defined by the server process, and no additional features or modifications are made to these protocols.

8.6 Global control Flow

Execution Orderness

The system uses both a linear style and event driven style to complete tasks. The worker processes operate as an event driven system, as they primarily respond to HTTP requests coming from users' computers

While responding to certain events, the processes must verify that a linear set of steps is also followed to ensure correct document processing. As multiple users or multiple documents are being processed concurrently, individual events may be received out of order. As an example, Document A may be in a state and ready for a certain a hypothetical step 7, but Document B may not be ready for the same state. Part of the response for each event would be to ensure that the document is ready for the step requested.

Time Dependency

There are no timers or time based components in the system.

Concurrency

The system uses an event driven model, so concurrency is expected and safe. For the linear procedures, the state of documents will be stored in the database to allow for multiple threads to work on the same document at different times. The state can be checked by each request worker to ensure that steps are performed in order.

8.7 Hardware Requirements

To run the main server application:

- A web server able to run PHP scripts and ~1 GB of disk storage.
- A MySQL database with at least 100 MB of storage space.
- A Redis instance.

For the main server, there are no hard memory or CPU requirements.

For end users, a functional web browser is required. The exact requirements depend on the browser and operating system in use by the user. Any system capable of handling an office suite would be capable of running this software.

Also, a stable internet connection is desirable. There’s no need for high speed internet, but it’s important to mention that the waiting time for downloading and uploading files to the server is directly influenced by the size of the file and internet speed.

9 Algorithms and Data Structures

9.1 Algorithms

No advanced algorithms are utilized in the application.

9.2 Data Structures

The Document object will contain a list of bookmarks stored in an Array format. Adding, removing, or re-ordering this array will add, remove, and re-order the bookmarks within the document as well. This data will use the built-in PHP array data type, as it is able to store ordered collections of objects.

This array will also be used to define a relationship in the database between bookmarks and documents. Each document will have an “order” key that corresponds to the index it has in the array and a “document_id” key that defines which document is associated with the bookmark. Loading the bookmarks by the “order” key will yield the final ordered bookmark array.

Bookmarks can also be nested, such as in the case of sub sections. When a bookmark is nested, a new relationship called “parent” will be defined as well. The parent will contain a reference to the bookmark that should contain this bookmark as well. The document_id and order keys will remain the same. Loading the bookmarks by the “order” key and filtering on the “parent” key as well will yield the subsections and given bookmark should contain.

As an example, say the document has the following structure:

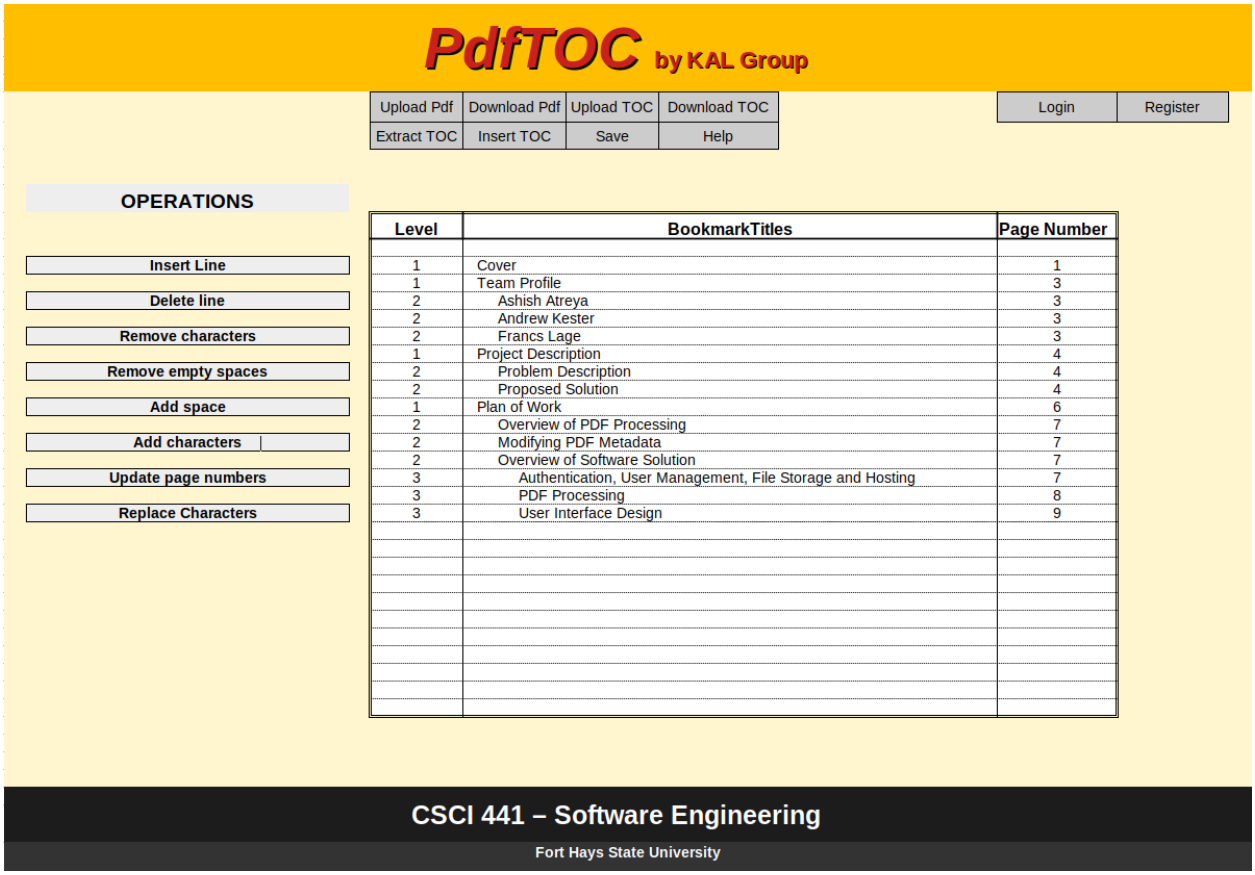
- Section 1
    - Subsection A
    - Subsection B

The object for section 1 will contain order 1 and have an empty parent relationship. The object for Subsection A will have section 1 listed in its parent relationship and order 1 as well. Subsection B will likewise have the parent listed, but order will be 2. When loading the bookmarks for a document using the bookmarks function, only section 1 would be returned. In section 1’s object, the children method would then return Subsections A and B.

10 User Interface Design and Implementation

The goal for our UI is to be simple and intuitive. Before drawing the first mock up, we wrote down all the necessary steps and actions to accomplish the task. The users must know what they need, the system interface goal is to show them right away how to get the results.

The mock up was built using a spreadsheet. There's still not so many modifications since the start of the project to this current stage. A few more buttons were added on the top area to correspond to users actions shown on the UML diagram. The edit operations on the left were updated as well.



One of the ways to reach our goal to build an intuitive interface is to use intuitive names in the buttons. Currently, we have as example “Insert TOC” and “Upload TOC”, this is not intuitive and needs further explanation. The insert means to load the current work done with the editing back to the file, and the upload means to fulfill the editing area with content from different sources, perhaps a text file inside the user machine.

Another possibility to make the UI simple and intuitive is group related resources. We let the registration and login on the up right corner because average internet users will expect to find those resources in that location. Also, the buttons in the top center will be focused on operating with files, while the buttons on the left side will be dedicated to editing the loaded content.

The final version of UI will be implemented using HTML and CSS. The buttons and functionalities of the page will be indexed by PHP scripts. There are no plans to change the visual aspect from the mock-up picture to the final version. The actual colors and shapes will be reproduced on the final version.

11 Design of Tests

Due to the nature of the web application, much of the test regiment must be completed manually. While automated browser based tests exist, the implementation of these tests was deemed too cumbersome for the application at the current time.

The manual tests will follow a documented manual process that tests individual use cases. The preliminary manual test steps are as follows:

1. Create a new user account on the system and login with the new account. (REQ3)
2. Upload a test PDF document. (REQ1)
3. Extract the metadata from the PDF. (REQ4)
4. Perform modifications to the metadata. These modifications are the minimum: (REQ 6,7,8,9,11,12)
    a. Add a new PDF bookmark.
    b. Remove a PDF bookmark.
    c. Edit a PDF bookmark's title and page number.
    d. Batch insert a series of bookmarks.
    e. Batch edit a series of bookmarks.
5. Search for patterns in title list (REQ10)
6. Save partially completed work and reload it. (REQ3)
7. Update the PDF file with the updated bookmarks (REQ5)
8. Download the PDF metadata as a txt file (REQ13,14)
9. Download the updated PDF file (REQ2)

Unit tests may be implemented for some individual class functions, but integration testing will be utilized as an overall method for acceptance testing of new changes.

References and Resources

The list of references should contain exact *references and URLs* of any material that supports the project.

Books and Tutorials:
https://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf
https://www.w3schools.com/sql/
https://en.wikipedia.org/wiki/Unified_Modeling_Language

Reports:
https://docs.google.com/document/d/1qeXLQGgb9OFnt-mcjdDhByVvpm6g8_WQN3ewMzfYzPg/edit?usp=sharing

Tools:
https://moqups.com/
https://www.diagrams.net/

Web Resources:
https://www.youtube.com/watch?v=ImtZ5yENzgE
https://laravel.com/docs/8.x/installation
https://laravel.com/docs/8.x
https://www.pdfescape.com/
https://www.pdflabs.com/docs/pdftk-man-page/
https://www.pdflabs.com/docs/pdftk-cli-examples/