Software Requirement Specification

Version 2.0

<<Final Report Version>>

December 5, 2020

CMPT275 Group 18

Aki Zhou - 301336141

Hamza Kamal - 301333339

Kirill Melnikov - 301312400

Table of contents

Introduction

Problem

Audio-books are expensive and many readings required for school do not have an audiobook available. As such, we want a program to convert PDFs into audiobooks that sound more natural than screen reading software bundled with operating systems.

Solution

Let users make their own audio-book files from their PDFs using cloud text-to-speech services. Users will have the ability to select page numbers, language, speed and pitch of audio, and gender of the voice output. The ability to convert reading texts to audio is especially useful for use on transit, while cooking, running, or walking. Ideal PDFs would include text only pre-reading material, short stories without an audio-book available, or news articles.

Requirements

Functional Requirements

- The user should be able to receive an audiofile after inputting a pdf file.
- The application should allow the user to specify a voice.
- The user shall be able to select pages in the pdf to convert.
- The user shall be able to queue multiple PDFs for conversion.
- ~~The user shall be able to choose an audio file type.~~
- The user shall be able to select a file save location.
- The user shall be able to vary pitch of voice.
- The user shall be able to choose a filename.
- The user should be able to close the application mid-conversion.
- ~~The user shall be able to view the estimated time for conversion.~~
- The user shall be able to resize the program windows.
- The user shall be notified when the conversion is finished.
- ~~The user shall be able to drag and drop PDFs into the program to queue for conversion.~~
- ~~The user shall be able to view the PDF in the program.~~
- ~~The user should be able to edit the pdf before conversion.~~

Non-functional Requirements

- The audio shall be generated within 10 minutes.
- The application should be responsive while the pdf is processing (i.e async operation).
- The system should run on modern operating systems (Linux, Windows, Mac).
- The software must connect with google cloud service.
- The software can only support pdf or text files.
- The UI must be accessible to color blind people.
- Must be intuitive for users through modern UI.
- The application should be safe for work use (no intrusive sounds/images).

- The application should have a limit of 1GB in disk space.
- The audio files should be limited to 1GB.

Constraints

- The program shall not violate any intellectual property including but not limited to copyrights, and software licenses.
- The program shall not gather any privacy sensitive information without consent.
- The program shall not introduce any security vulnerabilities to the system which operates it.
- The program shall not generate any audio that may cause hearing problems.

User Stories

- As a user, I want to select a PDF so that it can be processed.
- ~~As a user, I want to annotate text so it excludes irrelevant text, images.~~
- As a user, I want to save audio files in a specific folder so I can refer to them later.
- As a user, I want to select the pages of the PDF to convert into Audio so I can only listen to information I need.
- As a user, I want to choose a voice, so that I can comfortably listen to an audiobook.
- As a user, I want to select certain text to choose to read, so that I am not listening to information that I am not interested in.
- ~~As a user, I want to view the pdf in the application, so that I can see what I am choosing to read in the application.~~
- As a user, I want to queue multiple books to parse for audio reading, so that I can efficiently create multiple audio files.
- As a user, I want to see the progress of the conversion so I can see how long I need to wait for the file.
- As a user I want to be able to rename the audio file for easy access later.
- As a user, I want to change the pitch of the voice reader, so that I can be comfortable while listening.
- As a user, I want to see the pdf within the application, so that I can choose what I need to parse.
- As a user, I want relevant info (like the title) to be displayed in the current audio book about to be processed, so that I know which audio file is about to be created.
- As a user, I want to search for current audio files that are in the queue, so that I can't narrow down and find which audio books are currently waiting to be processed.
- As a user, I want to be able to remove an audio book from the queue, so that I do not needlessly create an audio file.
- As a user, I want to upload my audio files to the web, so that I can download it later for use.
- As a user, I want to be alerted when I did not select a pdf to convert, so that I can know what is the proper procedure to convert the pdf.
- ~~As a user, I want the audio files to be auto-stored, if I close the application without selecting a place to store it, so that I do not lose the audio files.~~
- As a user, if I close the application while it is processing a file, I want to be notified that I would lose my audio file, so that I do not accidentally terminate the task.
- As a user, I want to get a warning so I know if a PDF may be broken or invalid.
- As a user, I want to report errors/send feedback, so I can expect improvement on the app.

- ~~As a user, I want to have an estimated time remaining on my conversion so I can know how long it should take.~~
- As a user, I want to queue multiple PDFs to convert at a time so I can start more than one conversion at a time.
- As a user, I want to be notified that the size is too big to be processed, so that I know that I have to make the audio file smaller.
- As a user, I want to save my settings for downloading pdfs, so that later I do not have to redo them every single time, while the application is running.
- As a user, I want to resize my application window, to fit whatever I am currently doing on my computer.
- As a user, I want to be notified when the audiobook is done processing, so I can start listening as soon as possible.
- As a user, I want to specify the scope of the page so it can cut the page numbers for all pages.
- As a user, I want to prioritize certain pdfs in the queue to go first in processing, so that I can choose what I want to listen to first (goto top of queue).
- As a user, I want to be able to set a default folder from which all PDFs will automatically be converted.
- As a user, I want to see when I am not connected to the internet, so that I do not spend time using the application, which requires internet connection.
- As a user, I want to download the location of the audiofile, so that I can quickly get access to the file.
- As a user, I want to choose the accent of the voice, so that I'm comfortable listening to it,
- As a user, I want to choose a language for the voice, so that I can understand the audiofile.
- As a user, I want to change the speed of how fast the voice reads the audiofile, so that I can breeze through the audiofile at my own pace.
- As a user, I want to specify the gender of my voice, so that I can be comfortable.

Dropped requirements and why we did not do these use-cases

- **The user shall be able to choose an audio file type:** Options were only mp3 and wav. Since mp3 is much more common nowadays, and users can convert mp3 to wav on their own, the option to choose audio type is irrelevant to the goal of the project.

- **The user shall be able to view the estimated time for conversion:** There is an absolute limitation to how long text-to-speech API executes for and the bottleneck of the conversion time is the internet connectivity which varies a lot depending on many variables, making it difficult to be measured.

- **The user shall be able to drag and drop PDFs into the program to queue for conversion:** While this improves user experience, it is an extension to the program and not a requirement that is fundamental to the working software.

- **The user shall be able to view the PDF in the program, The user should be able to edit the pdf before conversion:** This is unnecessary to the fundamental goal of the program and requires much more complex tasks to complete.

- **Annotating text:** We decided to parse using pdfbox, and scan using this software, which means that we have no way of image processing. This would be a full on feature on its own, separated from our existing architecture. Therefore, we decided not to include it, as it is not a fundamental addition to our application.

- **Auto-storing audio files:** We've decided not to include this user story, as it is anti user-friendly to be downloading something to the user, while the application is not running. We want to ensure that everything the user is doing in the application is transparent.

- **Estimating download-time:** There is no direct way to retrieve how long it takes to process the audiofile from the API. Even if we can estimate it, it will be inaccurate. Not to mention that there is a hard-limit of 9 minutes for the audio file to process.

- **Viewing pdf within application:** This was not achievable for similar reasons as the use-cases "annotating text" and "selecting text from pdf." This was not a feasible feature for us to complete, in the time-frame, due to our chosen third-party software.

Modified Use-cases

- **Prioritizing pdfs in the queue:** This user story is completed in essence since the user can select any file from the queue to process, so a priority system isn't necessary.

- **Selecting text to process from pdf:** This is correlated to how we are parsing the text, using the PDFBoxConverter. Again, to select text, it would require image processing. As such, we decided to instead stick with the simplistic design of selecting pages only.

Design and Analysis

Why we chose our design

For our design, we've decided to keep the general structure simplistic. We have a main window class, which is instantiated in main. This would hold all our relevant objects, such as the Cloud API, pdf converter, and the pdf queue. Note that each object added is the interface object, to ensure abstraction. Each class has its own associated interface. This is useful in the event that we wanted to have a different implementation for a particular class structure. For example, though PDF text parsing is done on our end locally, we could have done this in the cloud. By having the interface, it is possible to have different types of implementation, while keeping the same specification. The same reasoning is applied for why we have a Cloud API. If we were for example to use Microsoft Azure, or Amazon Web Services, we can very easily switch the implementation.

In our PDFQueue, there exists a list that holds FileData objects. This is linked to the factory method, which we use to add a generic data object to the queue. The value of using the FileData object is that in the situation where we need multiple different file types, it is very easy to implement, since the interface object is added to the queue. It was ensured that the pdfQueue cannot be directly accessed, you must use the helper methods provided in the specification to access the list. To ensure abstraction, the pdf queue was a separate entity unrelated to the pdf converter. It would be easy to couple these two objects together, as they are quite intertwined and co-exist together. However, in the case that we modify the pdf converter, this would have drastic impacts on the queue, requiring us to make large changes to the implementation. As such, it is separated.

In our case, we have separated the call to main from MainWindow by adding the class PDF2Audiobook. This is the object that will start the whole application. This simplifies the program and shows a clear hierarchy in the initial boot of the program. Note that the mainWindow is essentially the whole UI. data objects are instantiated within this class, but whatever the user sees will be present within this class. All other data classes deal strictly with logic/data. This way, there is a clear separation on where the UI implementation starts and ends.

For debugging purposes, we've decided to make a couple of our own throwables to handle specific error situations. Namely the MACAddressError and the CredentialKeyError. The reason for these classes are more so for our own purpose, to ensure that we know the type of error that cannot be handled as an exception during runtime. The class that uses these is the GoogleCloudAPI where exceptions such as IOException, StorageException are thrown and caught, then a self diagnostic is done in order to determine which type of error it is or if it is an exception that should be handled.

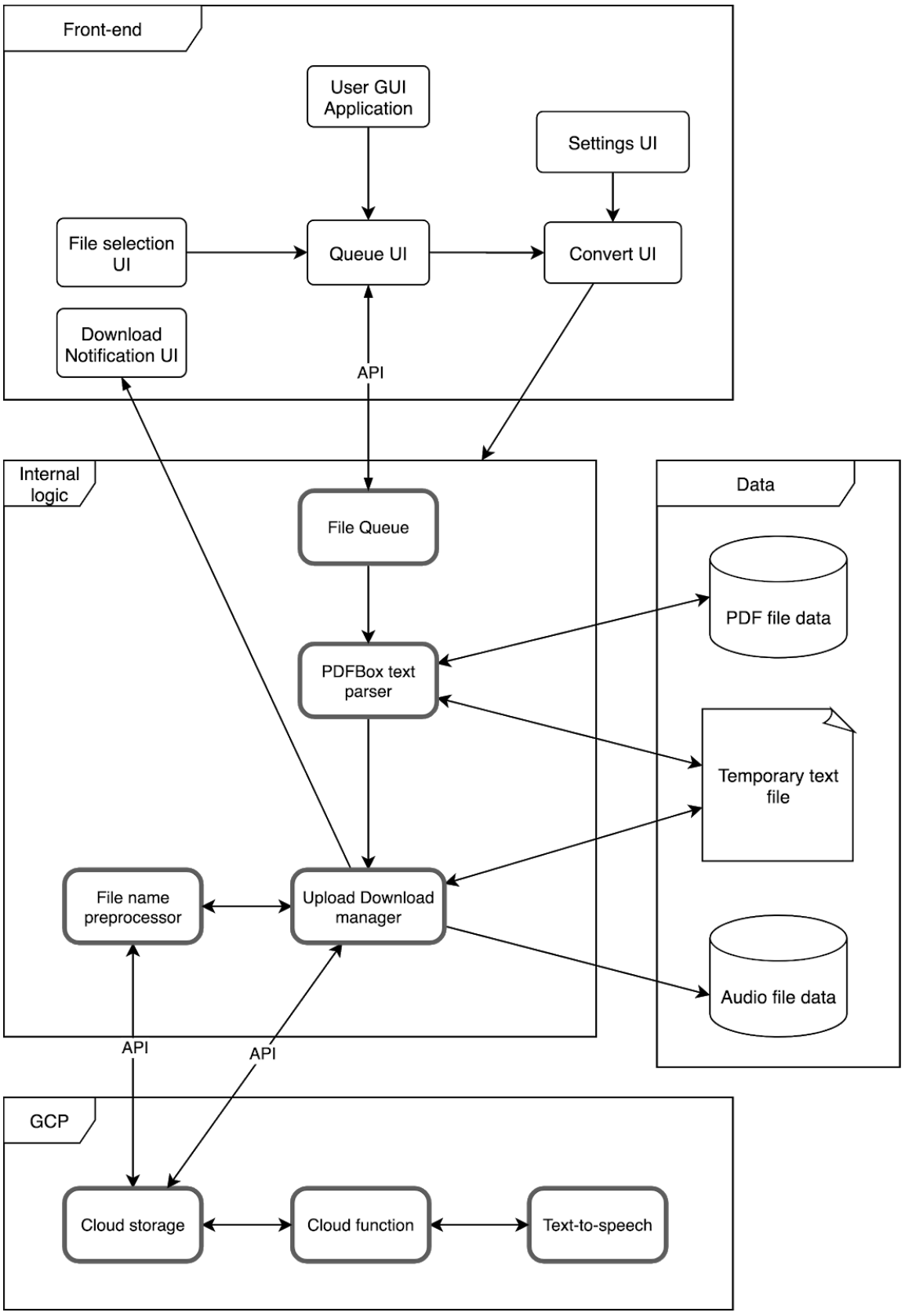
Once the error or exception is thrown, they will be caught in MainWindow. By having the different types of exceptions, it makes it easy to differentiate in the main window where issues occur and to display corresponding messages and/or call to action to the user.

Design patterns we used

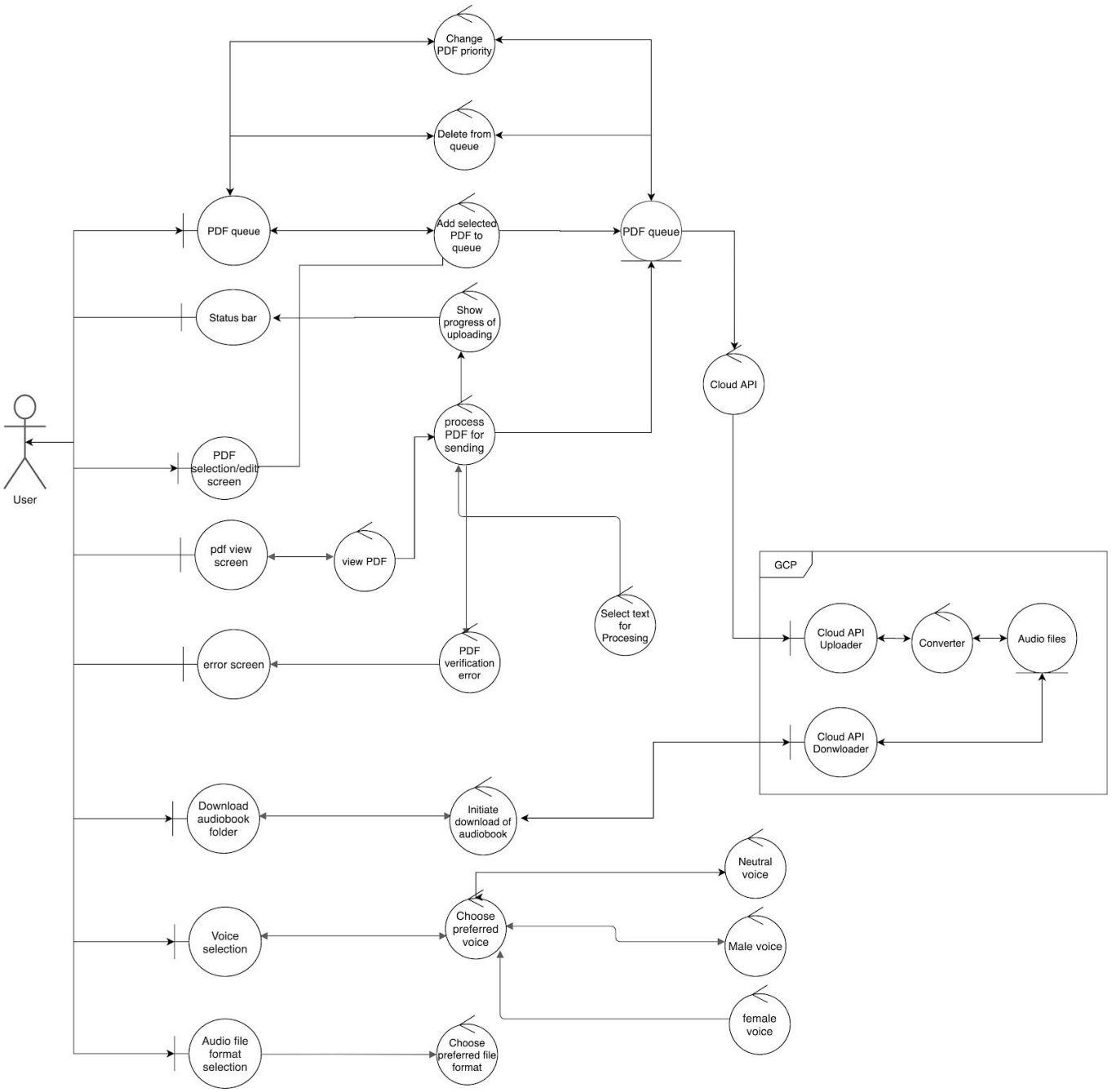
Factory Pattern: We adopted the factory pattern to instantiate the file data object, which was then added to the queue. The point of having this factory method, is that if we were to extend our queue to hold multiple types of data objects; for example, specific text files, pdfs, word documents, etc, then we have the liberty of doing so, without changing the implementation of the instance creation.

Adapter Pattern: We used the adapter pattern in GoogleCloudAPI in order to validate numerical inputs, voice settings, and also to set values in filenameManager indirectly without invocation of this class. This allows us to have a layer of abstraction between the UI and the cloud services.

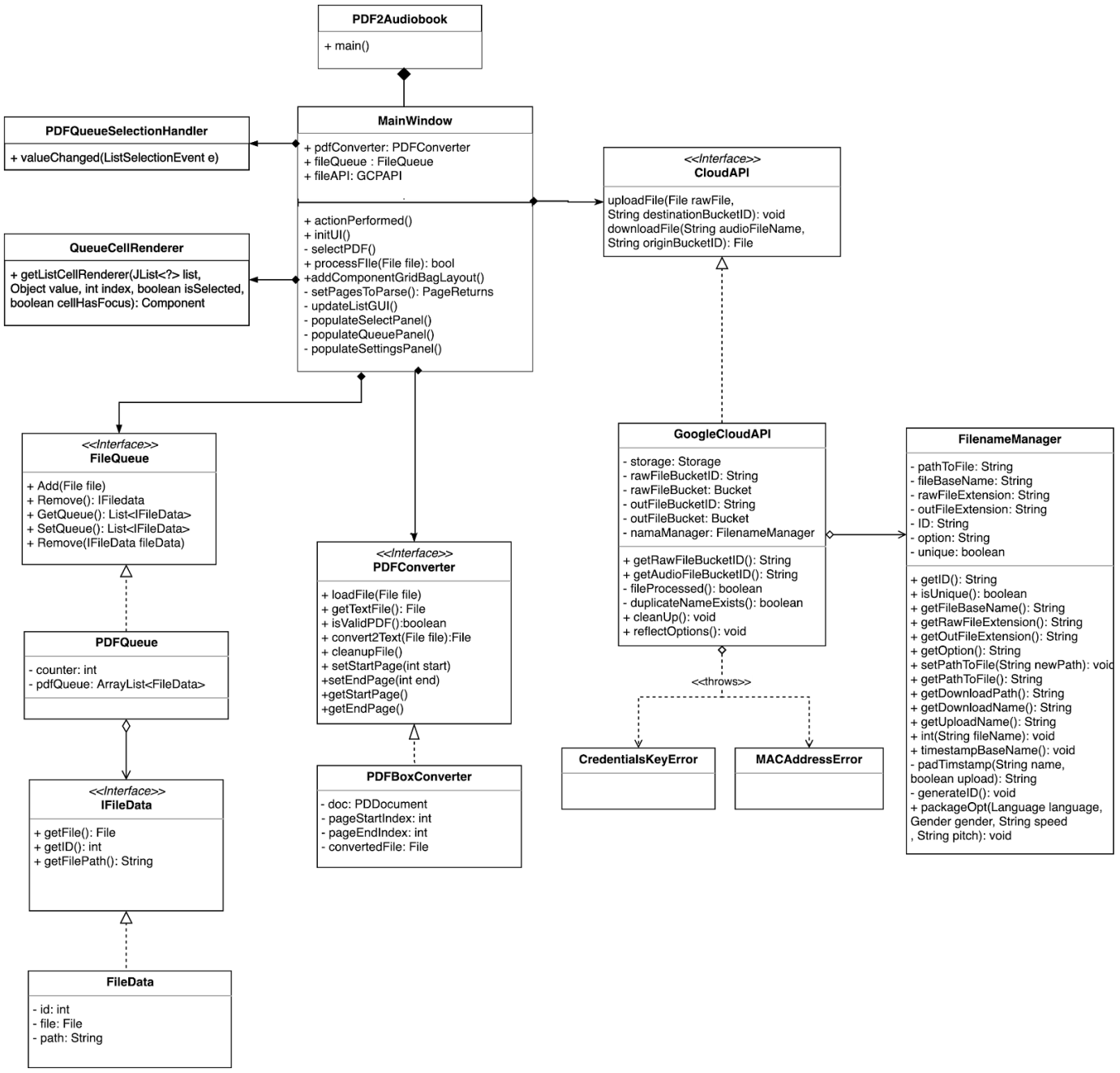
High Level Architecture



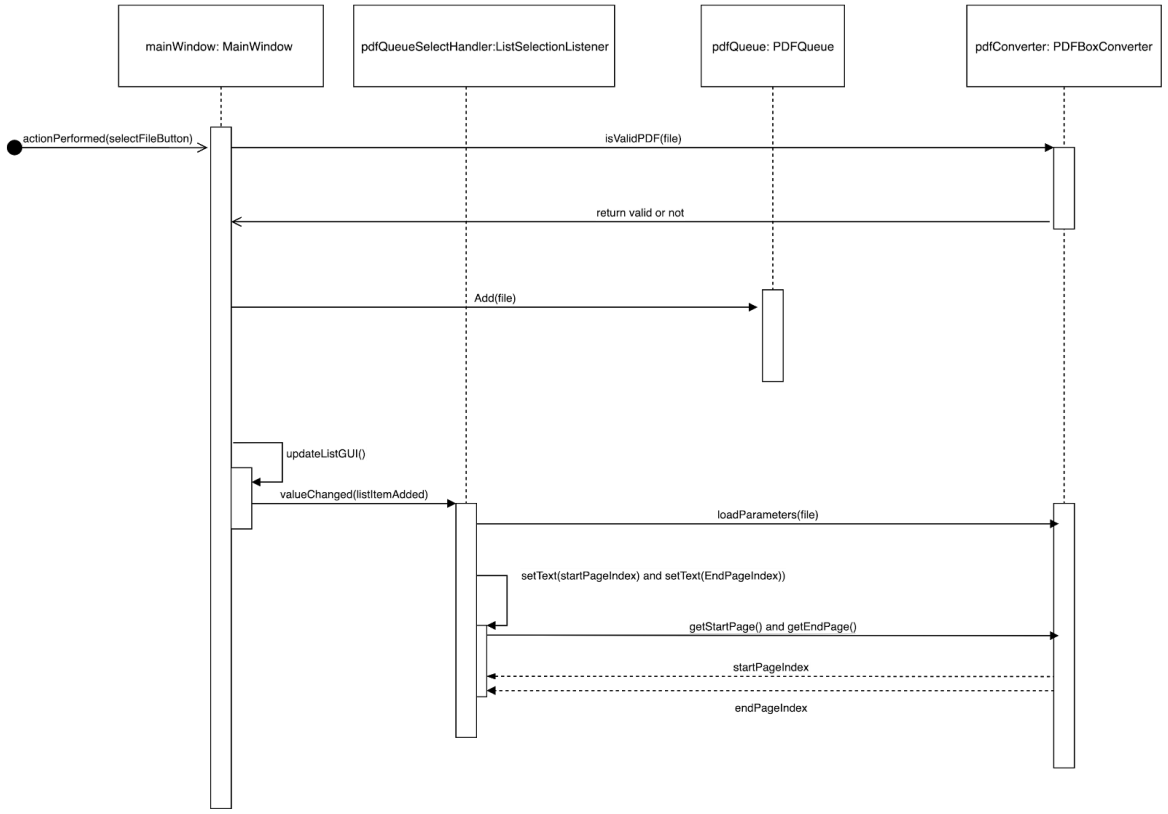
ECB Diagram


Change PDF priority
Delete from queue
PDF queue
Add selected PDF to queue
PDF queue
Status bar
Show progress of uploading
Cloud API
User
process PDF for sending
PDF selection/edit screen
pdf view screen
view PDF
Select text for Procesing
GCP
Cloud API Uploader
Converter
Audio files
error screen
PDF verification error
Cloud API Donwloader
Download audiobook folder
Initiate download of audiobook
Neutral voice
Voice selection
Choose preferred voice
Male voice
female voice
Audio file format selection
Choose preferred file format

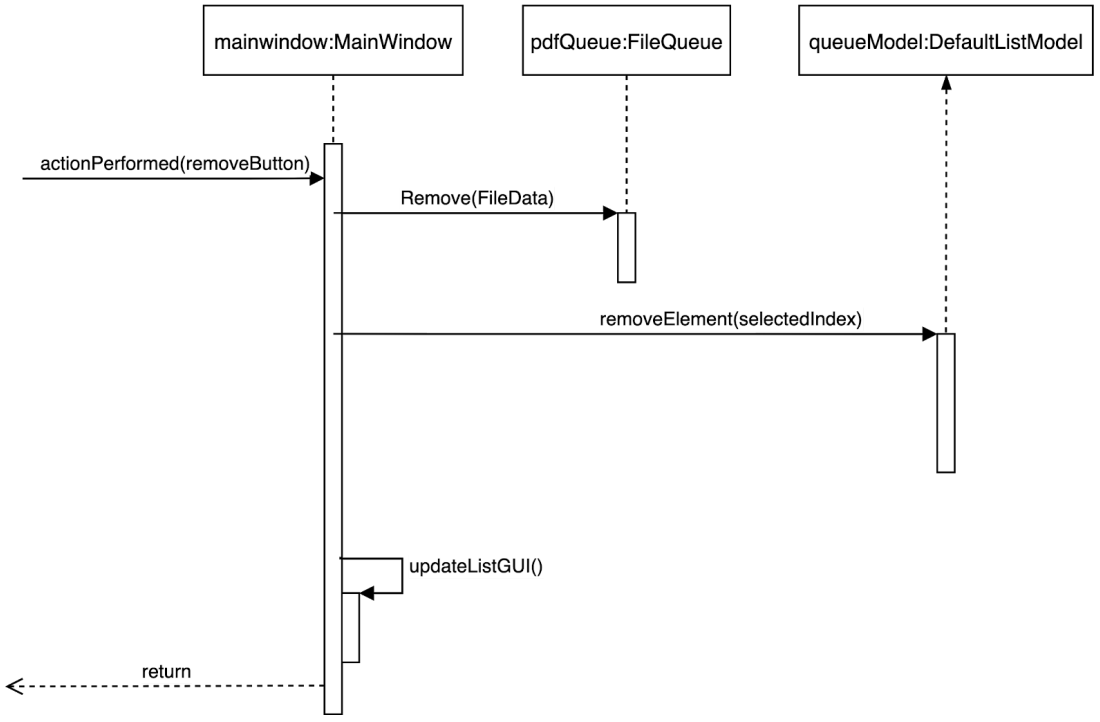
Class Diagram



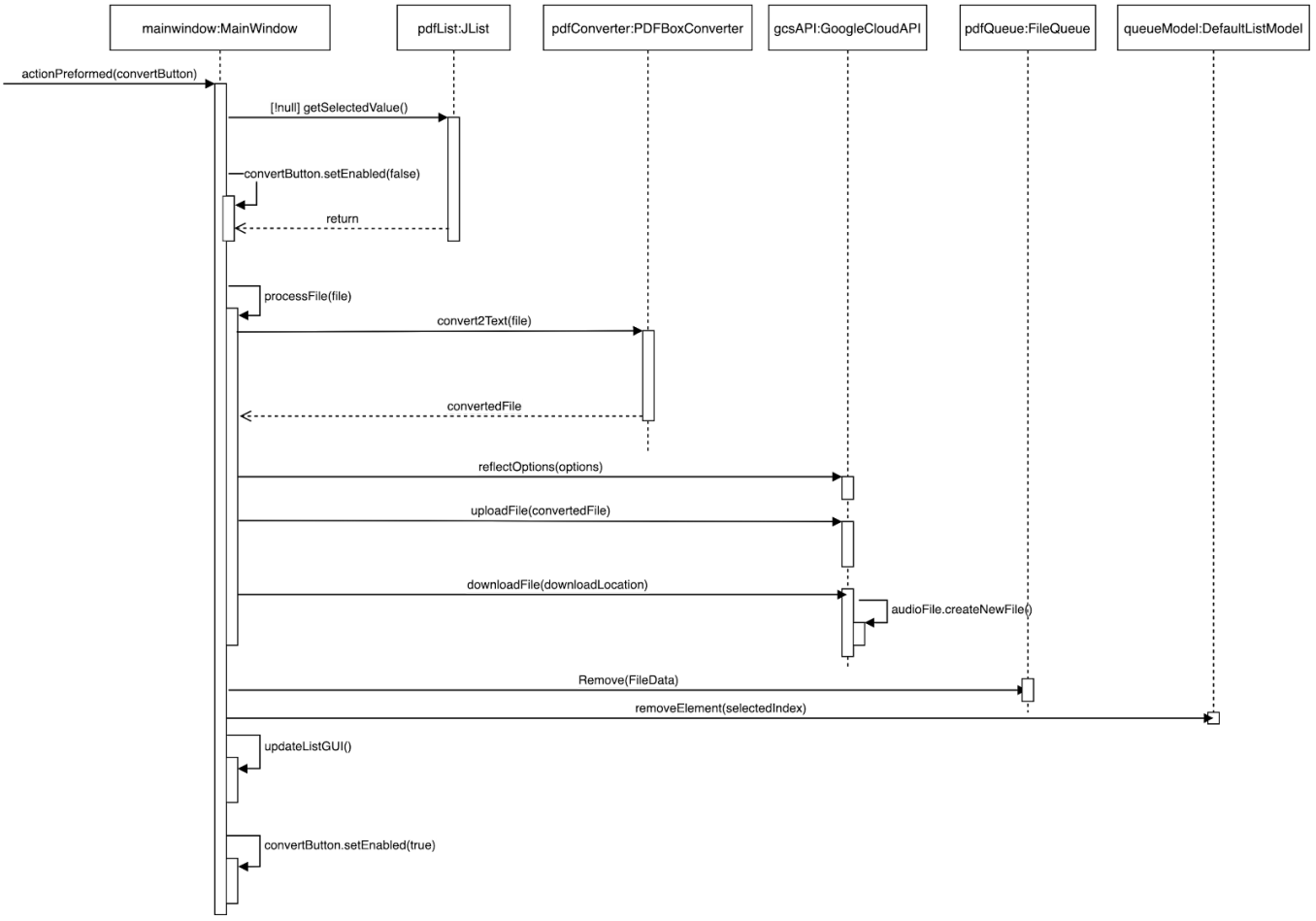
Sequence Diagrams



Selecting file to queue



Removing file from queue



Convert file

Test Cases

Unit Tests Summary & Description

PDFBox

The PDFBoxConverter class reads in a file and first tests if it's a PDF document, and then parses it into text and writes to a file. In between, it reads some meta-data of the file through the loadParameters method. We tested if it can differentiate between a PDF file and a non PDF file and whether it can correctly parse it.

isValidPDF:
Description: Tests whether the program can correctly identify a PDF document from a non PDF document.
Inputs: PDF document, and docx document
Result: When inputting a PDF document, the method returns true passing the assertion. When the input is not a PDF document, the method returns false which is caught by the assertion.
Boundaries: We expect an error when sending in a non PDF document.

loadParameters:
Description: Tests whether a PDF document is able to be parsed for its meta-data such as start and end pages.
Inputs: PDF document with 1 page
Result: In inputting a 1 page PDF document, the start and end pages are set to 1
Boundaries: Start and end pages being at minimum equal to 1. End page being larger or equal to the start page. Entering a non PDF document will result in an exception which will be dealt by the test failing.

convert2Text:
Description: Tests whether the loaded PDF document is able to be converted into a text file with all of its content
Inputs: PDF document
Result: When inputting a PDF document, it returns a converted text file.
Boundaries: Entering a non PDF document should throw an exception which will be dealt with the test failing.

PDFQueue

The priority of testing the pdf queue was to ensure that the methods worked currently. This would be the main structure that the user interacted with, as it gave a list of pdf that the user wanted to download. The tests covered were for all the publicly used methods of the queue, which included add, remove, remove with file return, and finally getters and setters for the queue. The goal of these unit tests is to limit/segregate the functions, so that each individual unit-test tested each method such that they were all separated.

Add:
Input: The input values were a simple file.
Result: Testing the adding of a file to the queue, where we add multiple objects to the queue, and then check the expected size of the queue to the actual size.

Remove:
Input: example file input
Result: Testing by adding a file to the queue through a file input, and then creating a similar fileData object. The idea behind this test was that the remove should not be able to delete the fileData object in the queue, since after all the queue is removed based on the memory address value, and not the ID or file value.

RemoveWithReturn:
Input: The inputs were a simple pdf file.
Result: The goal of this test was to ensure that using the remove function, when we check the file return with getFile(), it corresponds to the same file signature.

GetQueue:
Input: A new instance of an example queue array list, with data type IFileData (same as the PDFqueue).
Result: We ensure that by using the getQueue function, that the queue list example is not pointing to the same list inside the PDFQueue.

SetQueue:
Input: A simple example queue list.
Result: We check that by using the SetQueue function, that the queue list example is the same object that is present within the PDFQueue.

FileData

The file data object is the object that composes the PDF queue. The goal of the unit tests were to ensure that all the method values worked correctly. Note that the fileData object is generic, accepting any type of file. Actual validation of file type is done in the PDF Converter. The tests were done on the two publicly used getters getFile, and getID. Note that the ID is never exposed to the user, it is internal to the queue.

GetFile:
Input: a simple text file
Result: Test that the file added through the constructor is the same as the file example.

GetID:
Input: simple text file
Result: Test that the ID added through the constructor has the same ID as the expected value, and that the ID is a valid value greater than zero. Note that in the queue, the ID is a counter that starts at one, and increments on each addition to the queue.

FileDataFactory

The FileDataFactory was the primary object used to construct the fileData object used within the PDFQueue. There was only one test, corresponding to the "Create" method.

Create:
Input: test example file
Result: Ensure that the fileData object returned from the Create() method has the same file signature as the example file, and has the same ID as the expected value.

FilenameManager

The filenameManage is a class used to manage filename for upload and download in order to avoid race conditions caused by duplication filenames.

setPathToFile_ForEmptyString_ReturnsDefaultPath:
Input: empty string
Result: Ensure getPathToFile() returns the default download path.

setPathToFile_ForStringWithoutTrailingSlash_ReturnsPathWithSlash:
Input: path not ending with slash as string
Result: Ensure getPathToFile() appends a trailing slash to the path.

setPathToFile_ForStringWithTrailingSlash_RetunsSamePathAsArgument:
Input: path name ending with slash as string
Result: Ensure getPathToFile() returns the same string as input.

getDownloadPath_BeforeInit_ThrowException,
getDownloadName_BeforeInit_ThrowException,
getUploadName_BeforeInit_ThrowException,
timestampBaseName_BeforeInit_ThrowException:
Input: none
Result: Ensure getDownloadPath(), getDownloadName(), getUploadName(), timestampBaseName(), throws IllegalStateException if called before init() was invoked.

getDownloadPath_AfterInit_ReturnPathOfDownloadFile:
Input: test filename as string, false for timestamp option
Result: Ensure getDownloadPath returns a path for a mp3 file having the same base name as the test file under default download location.

getDownloadName_AfterInit_ReturnNameOfDownloadFileWithID:
Input: test filename as string
Result: Ensure getDownloadName() returns a mp3 filename having the same base name as the test file prepended with MAC address of local host.

getUploadName_AfterInit_ReturnNameOfUploadFileWithIDAndOption:

Input: test filename as string
Result: Ensure getUploadName() returns a mp3 filename having the same base name as the test file prepended with MAC address of local host and appended with options.

init_SetsUniqueToTrue:
Input: test filename as string
Result: Ensure init() sets private boolean member variable unique to true so get methods will now execute without throwing IllegalStateException.

init_GetsFileExtension:
Input: test filename as input
Result: Ensure init() reads the file extension of the test file and sets that as rawFileExtension.

timestampBaseName:
Input: test filename as string
Result: Ensure getFileName after invocation to timestampBaseName() returns a name that is not equal to the original test filename.

packageOpt_packsOptionsTo25Characters:
Input: sample options, Language, Gender, speed, pitch
Result: Ensure the length of option is formatted to 25 characters regardless of the option chosen.

packageOpt_packsOptionsToCommaSeparatedString:
Input: sample options, Langue, Gender, speed, pitch
Result: Ensure packaged option string is in the format or (language,gender,speed,pitch) prepended with some number of "-" to make the total length 25.

GoogleCloudAPI

The GoogleCloudAPI class is responsible for uploading text files and downloading converted audio using Google Cloud services client libraries.

GoogleCloudAPI_ForNoKeyGiven_ThrowError:
Input: None
Result: Ensure instantiating GoogleCloudAPI without a key throws CredentialKeyError.

reflectOptions_ForSpeedLessThanMIN_SPEED_ThrowException:
Input: numeric value less than MIN_SPEED which is 0.25
Result: Ensure reflectOptions throws IllegalArgumentException.

reflectOptions_ForSpeedMoreThanMAX_PITCH_ThrowException:
Input: numeric value greater than MAX_PITCH which is 4
Result: Ensure reflectOptions throws IllegalArgumentException.

reflectOptions_ForSpeedLessThanMIN_PITCH_ThrowException:
Input: numeric value less than MIN_SPEED which is -20

Result: Ensure reflectOptions throws IllegalArgumentException.

reflectOptions_ForPitchLessMoreMAX_PITCH_ThrowException:
Input: numeric value greater than MAX_PITCH which is 20
Result: Ensure reflectOptions throws IllegalArgumentException.

Language

The Language enum is a user defined enum with methods that returns a custom string as the name of language code.

codeOfReturn_LanguageCode:
Input: language name as string
Result: Ensure codeOf() returns the correct language code as a language enum.

noMatchingCode_ReturnNull:
Input: language name not supported
Result: Ensure codeOf() returns null.

langNames_ReturnListOfNames:
Input: none
Result: Ensure langNames() returns the correct list of language names.

noMaleVoice_forLanguageWithoutMaleVoice_ReturnTrue:
Input: language names that do not have male voice available
Result: Ensure noMaleVoice() returns true.

System testing

Invalid credentials

In case the credential key of the program is invalid, a message (Fig 1) shows up.

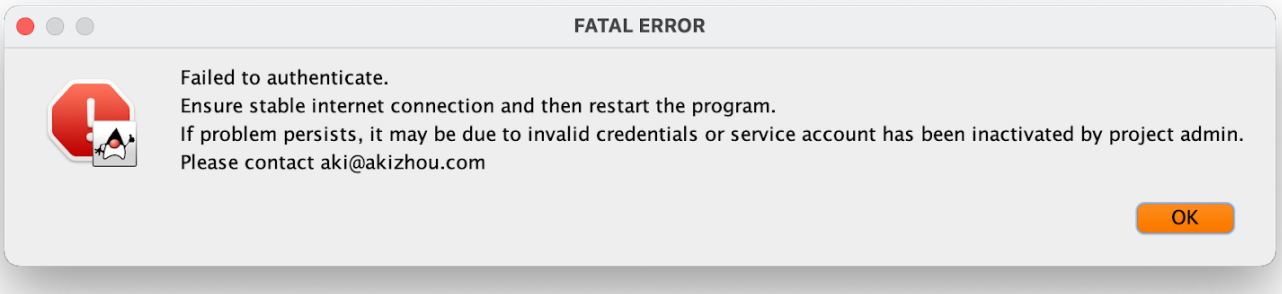
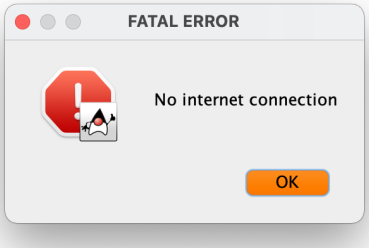


Fig 1

No internet connection

In case of bad internet connection or offline, the program will try to establish connection for a while and if no conversion was initiated within 1 minute after the start of the application, the authentication will fail when 1 minute passes and the same message (Fi 1) as the invalid credentials will pop up.
If the user attempts to convert a file before authentication fails without internet connection, another error message (Fig 2) will pop up notifying no internet connection and will keep showing the same message until authentication passes or authentication fails after 1 minute.
If internet connection drops after authentication succeeds, error indicating bad server connection (Fig 3) will pop up.



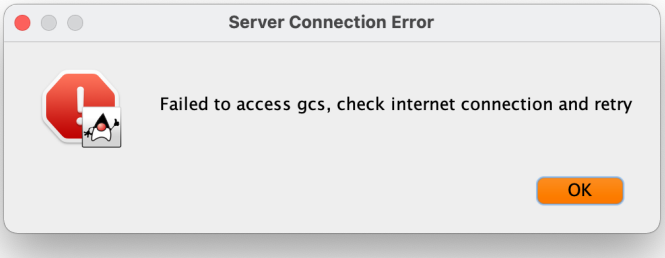


Fig 2

Fig 3

Attempt to convert without selecting any files

It is trivial that a file must be selected for conversion (Fig 4), message is shown to notify this step.

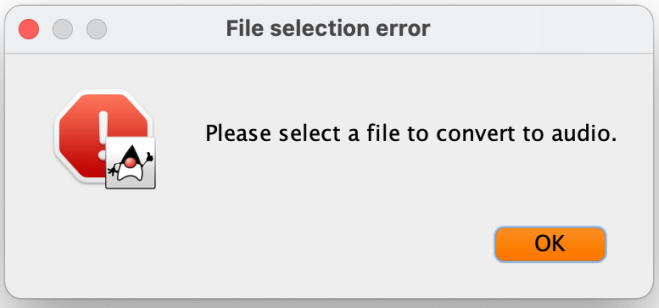


Fig 4

Select invalid PDF

The program file selection window has a filter that only allows files with .pdf extension to be selected, but in case of a corrupted PDF file is selected the program notifies by a message pop up (Fig. 5).

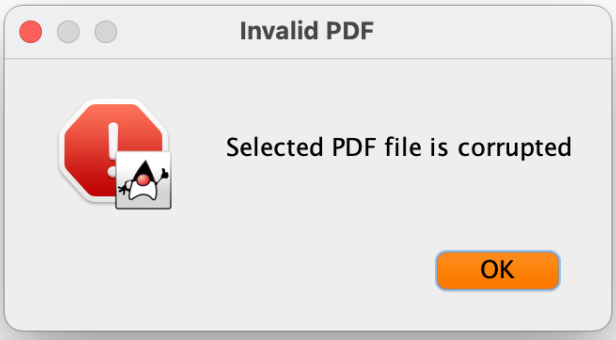


Fig 5

Select invalid page number

In case a start page or end page is out of range or non-numeric value was entered in the text boxes, settings input error (Fig 6) pops up. Fig 6 is an example of settings input error for a 1 page pdf document.

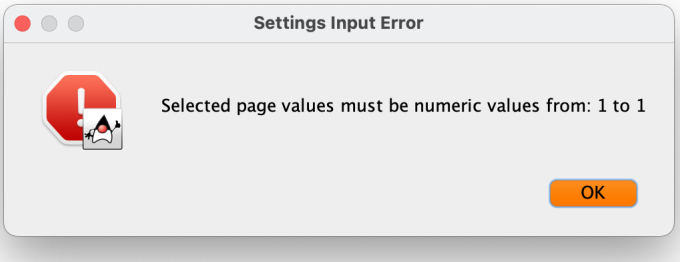


Fig 6

Select language without male or neutral voice type

There are languages that only support female voices and neutral voices are going to be implemented soon by google text-to-speech thus for these option choices, a warning (Fig 7) will show up and the option will default to female voice instead.

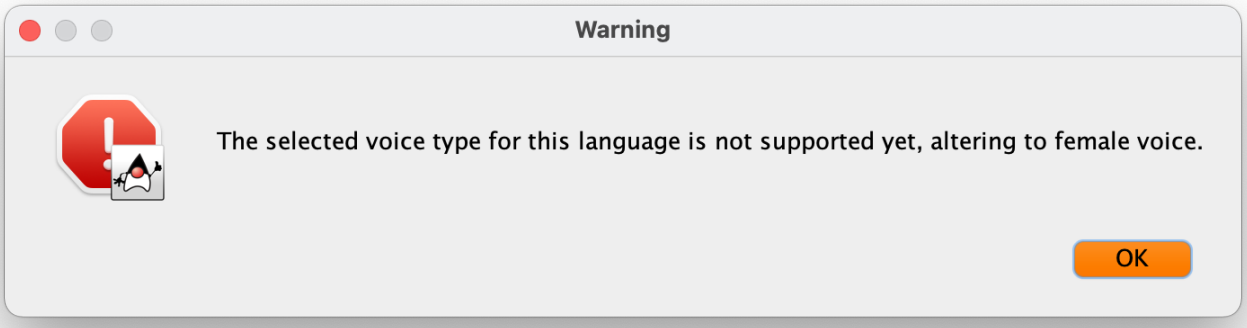


Fig 7

Original file renamed, moved, removed

The program file queue does not store the data but where to get the file on the user's file system. For this reason, if a file is selected and added to the application's queue but renamed, moved, or removed before conversion, the program will throw a file access exception and a message (Fig 8) will popup.

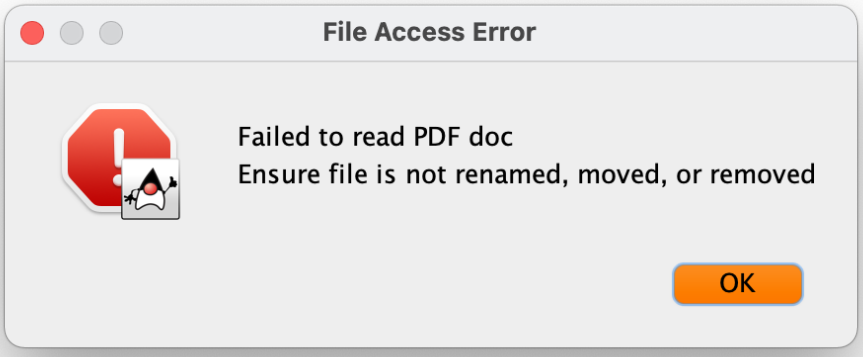


Fig 8

Conversion timeout

The program conversion process has a limit of 9 minutes on Google Cloud, thus documents that are very large may cause not have enough time to be converted or if anything happens on the cloud nothing will be returned so the program will throw a timeout exception after 10.5 minutes (allowing some time for local processing) from the start of the conversion and notifies the failure of conversion and recommends the user to retry with smaller sized PDF (Fig 9).

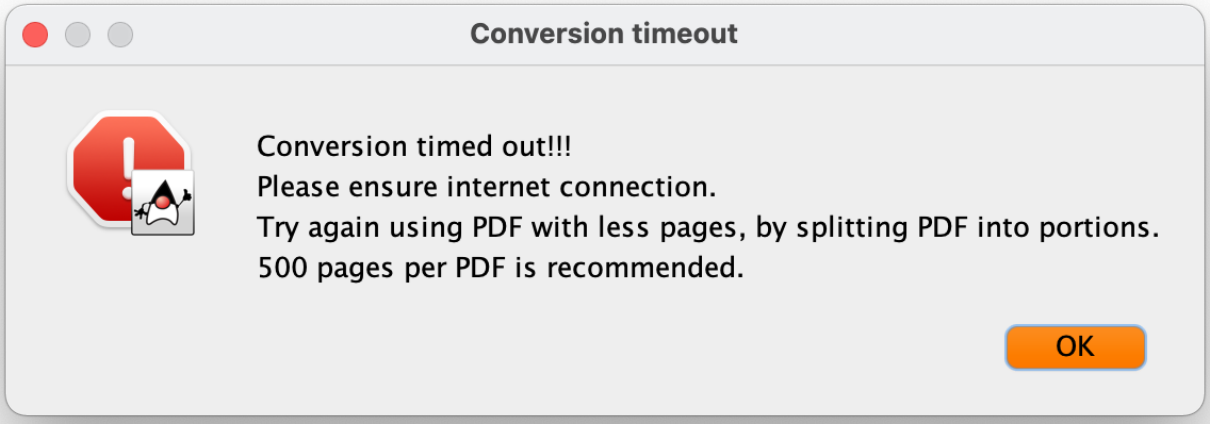


Fig 9

Exiting program

The program does not retain any information about a conversion thus terminating the application means all incomplete conversion progress will be lost. The user will be asked (Fig 10) to confirm exiting regardless of if there is an ongoing conversion.

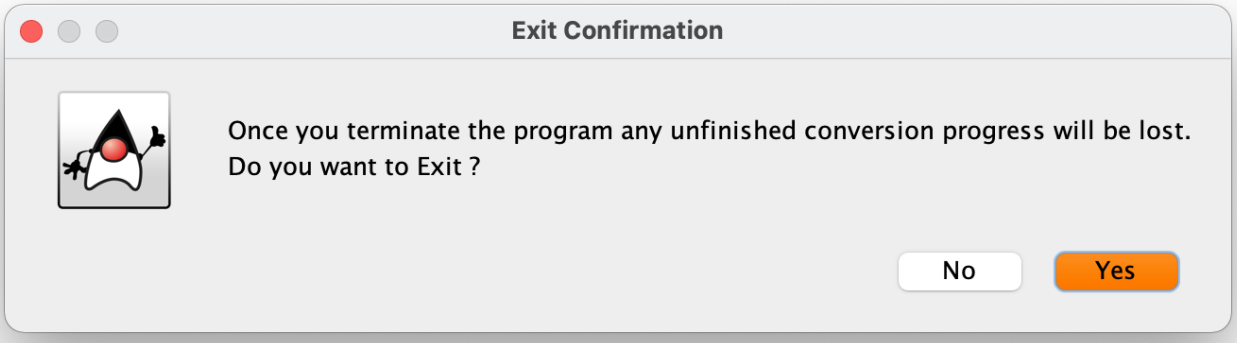
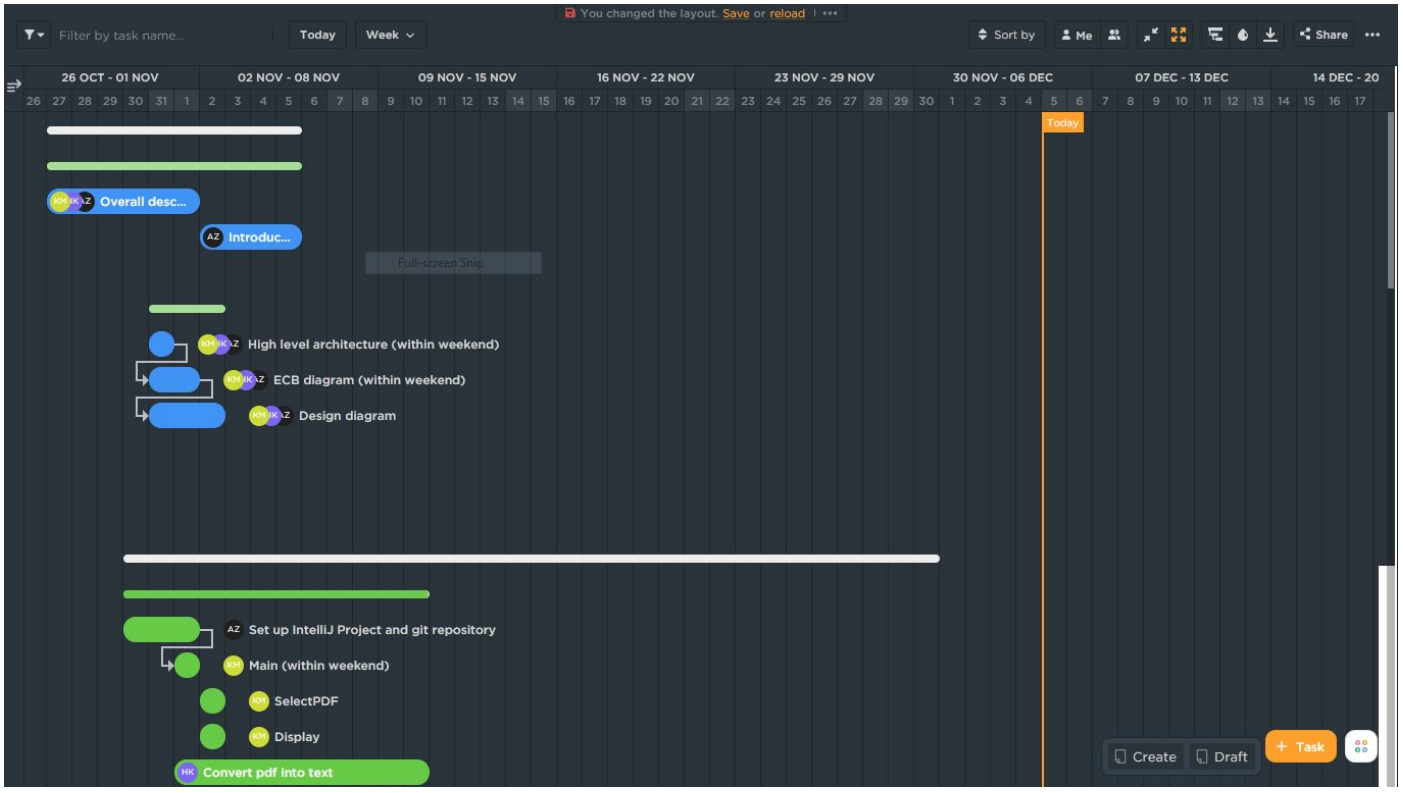
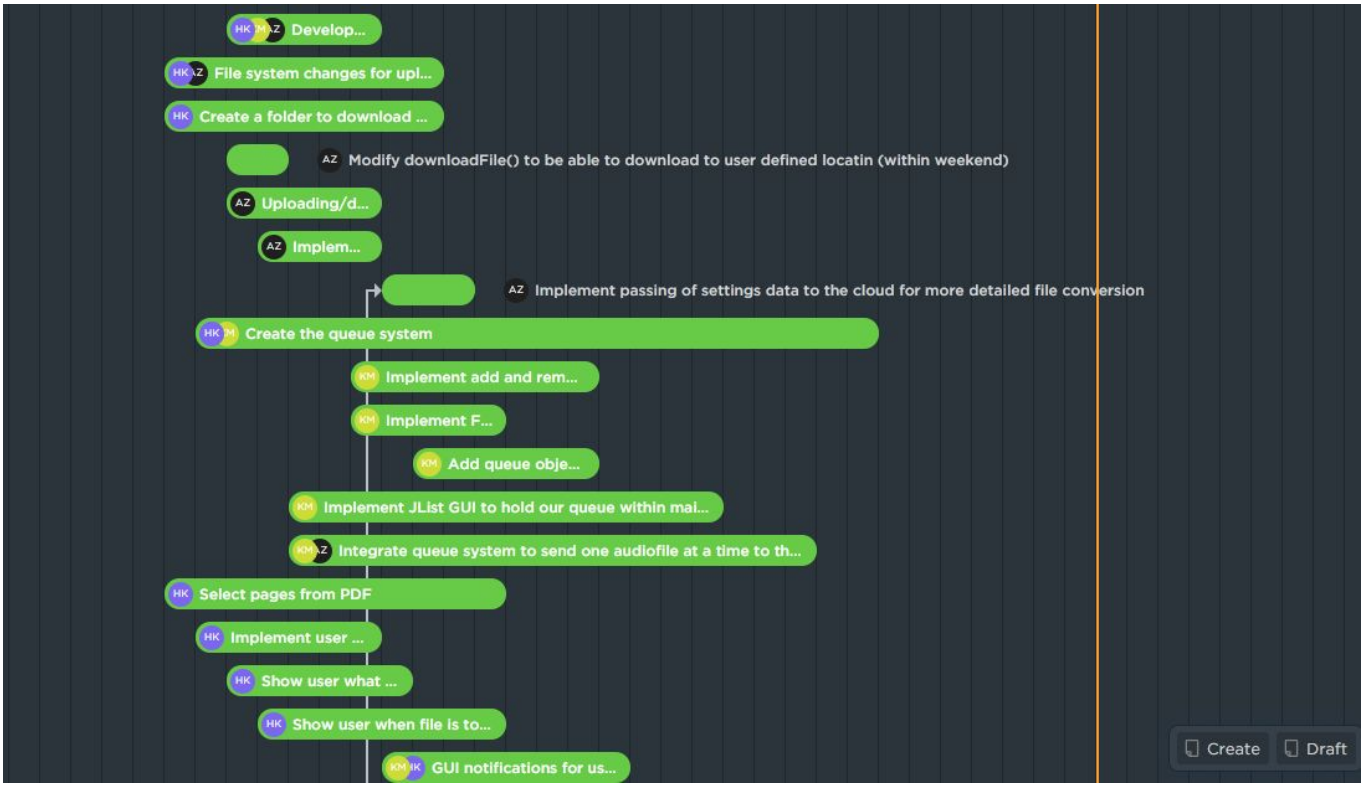


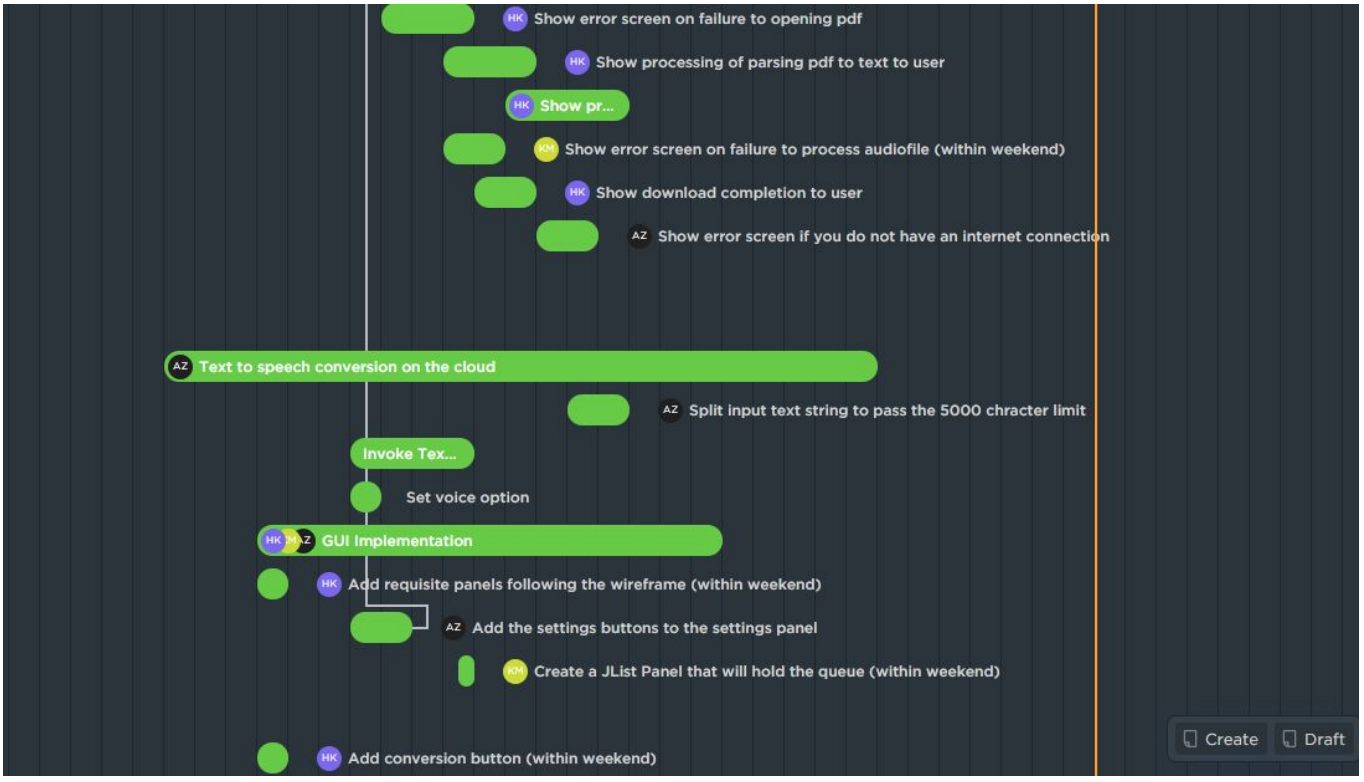
Fig 10

Gantt Chart


You changed the layout. Save or reload
Filter by task name...
Today
Week
Sort by
Me
Share
26 OCT - 01 NOV
02 NOV - 08 NOV
09 NOV - 15 NOV
16 NOV - 22 NOV
23 NOV - 29 NOV
30 NOV - 06 DEC
07 DEC - 13 DEC
14 DEC - 20
Today
Overall desc...
Introduc...
High level architecture (within weekend)
ECB diagram (within weekend)
Design diagram
Set up IntelliJ Project and git repository
Main (within weekend)
SelectPDF
Display
Convert pdf into text
Create
Draft
Task


GCP API
Write unit tests for PDFQueue
Write unit tests for Language
Write unit tests for GoogleCloudAPI
Write unit tests to fileData (within weekend)
Write unit tests for FilenameManager
Error notifcations & UI Polish
Voice options
Create user choices for pitch
Create user choices for gender
Create GUI radiobuttons & checkboxes for
Ensure program is platform independent (within weekend)
Create
Draft


Develop...
File system changes for upl...
Create a folder to download ...
Modify downloadFile() to be able to download to user defined locatin (within weekend)
Uploading/d...
Implem...
Implement passing of settings data to the cloud for more detailed file conversion
Create the queue system
Implement add and rem...
Implement F...
Add queue obje...
Implement JList GUI to hold our queue within mai...
Integrate queue system to send one audiofile at a time to th...
Select pages from PDF
Implement user ...
Show user what ...
Show user when file is to...
GUI notifications for us...
Create
Draft


Show error screen on failure to opening pdf
Show processing of parsing pdf to text to user
Show pr...
Show error screen on failure to process audiofile (within weekend)
Show download completion to user
Show error screen if you do not have an internet connection
Text to speech conversion on the cloud
Split input text string to pass the 5000 chracter limit
Invoke Tex...
Set voice option
GUI Implementation
Add requisite panels following the wireframe (within weekend)
Add the settings buttons to the settings panel
Create a JList Panel that will hold the queue (within weekend)
Add conversion button (within weekend)
Create
Draft


Add the add,remove buttons to GUI (within weekend)
Disable or handle textbox for choosing pathname on clicking download location (within weeken
Add default download p...
Create
Draft
Task

Post Performance Analysis

Successes:
The PDF can be converted into an audiobook quite fast. A 300 page document on average can be converted and downloaded into the users folder in about 1 minute. This is done by utilizing Google Cloud Services' Text-To-Speech converter and multithreading which allowed us to significantly speed up the conversion time.
A working UI in place that does not block at any point, allowing the user to have full control over the application. A simplistic design that allows us to choose voice, pitch, gender, and language/accent for the conversion from pdf to audio.

Failures:
In the project, we had not separated the presentation layer from the logical layer enough. In the future we should utilize an architecture like MVC where we can split up the presentation layer to just UI elements and have a controller for it which talks to the logical layer. Without this separation, too much of the logic is coupled with the UI and leads to a large singular file.

Exceptional Cases:
If a user wants to convert an exceptionally large PDF, the application will time out before it can convert all of the PDF and thus return an error to the user to try and convert a smaller PDF. The timeout value is set by Google Cloud Function and although it is multithreaded so it can handle very large files, for instance 1400-page long textbook takes 6 minutes, some extremely large files just cannot be handled by Google Cloud Function.

Project Management:
We utilized ClickUp for our project management tool. It gave us the ability to set up tickets with estimated times and linkages between blocking tickets. We also utilized a Github hosted repository and git which allowed us to quickly collaborate on the same codebase. Within GitHub, we made sure to create new branches whenever we wanted to implement a new feature or fix a bug on the master branch. Then we utilized pull requests before merging so changes could be reviewed by the other members. We also had bi-weekly meetings and were constantly in communication over messaging applications to discuss where we were blocked or what task was needed to do. All together, this project management strategy was a success as we had a clear idea of what each member was working on at all times and what needed to be done. Throughout the duration of this project, we did not have leading personnel, such as a project manager. All of our decisions were done as a whole, following a more agile paradigm.

Things we would do differently:
Instead of constraining the UI to a desktop application, using a web based service and calling google cloud services via https would have been a much more robust implementation. We used Java swing for our UI, which was expected since we used it for most of our assignments. However, this is a limiting library, as it is quite old and has not been updated in a while. If we used a more modern library, this would improve the general looks of our UI, and allow us to adopt a variety of different methods to accomplish this. We should also do more continuous integration testing, to ensure that with whatever big changes we did, when we put all the components together the product was still

working. An issue we had was generating the jar file, as some components were not linked correctly. This took time, and our testing depended on it, which lost us a significant amount of time.

Things that could have done to improve the process:

- Follow the gitflow convention to manage branching when using git.
- We should have done estimations that represent both the complexity and expected time taken, whereas all we did was track the due date of the story/feature we were working on.
- We could have written unit tests before writing code that passes it so the process is test driven and avoids unnecessary features.

Are we proud?

Yes. The application meets the core requirements we set forth from the beginning. We also are proud of the functionality as this project is an interesting problem to solve. It was also a good learning experience with Google Cloud Services, JUnit, Maven, and Java.

Communication-level:

The communication level was good. Whenever we needed to schedule a meeting, there were clear and quick responses. If there was a need to reschedule meetings, this was also done proficiently with no issues. We always discussed what changes each person was doing, so it was clear what each person was currently working on. This made it easy to track existing issues/stories in progress. We also ensured that in meetings, it was always in voice. In our meetings, we discussed any existing problems we have, and brought forth discussions to solve them. This allowed us to have a steady track of progress within the duration of this projection, without any large setbacks.

Did we do the best we could?

It depends. Strictly speaking from the requirements, we fulfilled them, meaning that our goal for this project was complete. However, there was always space to improve. The GUI could have been polished more, and we could have explored many other features, such as drag and drop. As for the conversion, a great improvement to performance would have been allowing us to convert multiple pdf to audio once. Currently, our software is converting a pdf file to audio one at a time. This is inefficient, but this was the simplest way to ensure that the cloud side was synchronized with our program.

Video Presentation

Youtube link: https://youtu.be/Nhq4IrdsqME
Raw video download: https://vault.sfu.ca/index.php/s/HQWcF5921ofwnM1

Contribution List

Requirements

Aki Zhou	33.33%
Hamza Kamal	33.33%
Kirill Melnikov	33.33%

Design

Aki Zhou	33.33%
Hamza Kamal	33.33%
Kirill Melnikov	33.33%

Working Software

Aki Zhou	51.02%
Hamza Kamal	24.49%
Kirill Melnikov	24.49%

Documentation

Aki Zhou	24.00%
Hamza Kamal	38.00%
Kirill Melnikov	38.00%