# Team 37 - Design Document

Aneesh Pendyala, Sergio Alvarez, Zhenyao (Michael) Yang, Kye Jocham, and Brennan Frank

## Project: EasyDraft

# 1 Purpose

Businesses, like law firms and medical offices, have to create the same document for multiple people. They have to change, usually by hand, some parts of each document like client names, employee numbers, etc. Our solution is to create a template processor where there are placeholders in a document and the user has a set of information (client names, employee numbers, etc.) ready to fill those placeholders; after having the information for the placeholders ready, with the press of a few buttons they have all their documents filled out.

We intend to target this application to users that need to send the same document to multiple different clients. These include people from various businesses such as law/accounting firms, medical offices, etc. These users require an efficient tool to manage large amounts of documents without the need for repetitive and manual updates. With this application, users can reduce manual input, minimize errors and increase the document processing efficiency.

Functional requirements:
1. Template Management and Reuse
    a. As a professional,
        i. I want to save and reuse templates for efficiency and consistency.
        ii. I desire the ability to name the templates for easy identification.
2. Learning and Onboarding
    a. As a user,
        i. I would like to have visually guided instructions to teach me how to use the software, enhancing my learning experience.
        ii. I would like to be able to re-do this onboarding experience whenever I need it, as often as I need it.
3. Template Personalization and Version Control
    a. As a client,
        i. I'd like to open a template and get my own version to fill out; the server will create this for me the first time I open the template, and subsequently reuse it for efficiency and personalization.
4. Personalized Bulk Output
    a. As a user,
        i. I want to specify a list of users for personalized bulk output (e.g., many PDFs) from a text box within the program, streamlining the document creation process.
5. Document Exportation
    a. As a client,
        i. I would like to export the documents created in a PDF format for standardization and ease of sharing.
        ii. I would like to choose from multiple export formats for my documents to meet different submission standards, enhancing flexibility.
6. Template Accessibility
    a. As a user, a. I want to see the top three templates I use most often at the top of a list when selecting a template, improving accessibility and efficiency.

7. Text Customization
    a. As a client,
        i. I would like to bold, italicize, or underline the text I input for enhanced document styling.
        ii. I desire more customization with the text I input, such as changing font style and size, for greater control over document appearance.
8. Placeholder and Document Management
    a. As a user,
        i. I want to save the information for placeholders I've used before to complete other documents, enhancing convenience and efficiency.
        ii. I would like to preview documents before finalizing to ensure all information is correctly placed, improving accuracy.
        iii. I would like to input where in the template should have placeholders for customized document creation.
        iv. I want to select pre-filled inputs from a list that I have created to fill the template fields with, for streamlined document preparation.
9. Additional Features for Efficiency and Accessibility
    a. As a user,
        i. I would like to be able to export my documents to email software to send emails to each client when bulk outputting multiple documents, adding efficiency to the communication process
        ii. I desire the ability to have a signature input box for authenticated document signing.
        iii. I want the application to be usable with a visual disability, including a high contrast mode, ensuring accessibility.
        iv. I would like to create my own template document from scratch in the application, offering flexibility in document creation.
10. Organization and Feedback
    a. As a user,
        i. I want to tag and categorize my documents and templates for easier organization, enhancing findability.
        ii. I desire a feedback button on the software to give developers suggestions or report bugs, contributing to the software's improvement.
11. Software Usability and Security
    a. As a user,
        i. I would like my work to be auto-saved so I don't lose progress during system or software crashes, ensuring reliability.
        ii. I wish to receive regular updates for security and feature upgrades, maintaining the software's integrity and usability.
        iii. I should be able to use the application on different screen sizes and resolutions, ensuring accessibility across devices.
12. Advanced Template and Document Handling
    a. As a user,
        i. I want the ability to organize and sort my templates into folders for better management.

ii.   I desire the option to print the finished document for physical record-keeping.

iii.   I would like to be warned if I'm creating a template that is not completely filled, preventing incomplete document submissions.

Non functional requirements:

1. Main Function
    a. As a developer,
        i.   There will be a document in Word with placeholders possibly with special language (e.g $firstName, $Date, $Employee_pay or something similar) that will be inputted. These special inputs define the places in the document where the specified text will be replaced in the output.
        ii.   They import their docx document in our application,

        iii.   They input information for the placeholders related to each receiver of the document

        iv.   Then bulk output multiple documents each related to a receiver.

2. Architecture
    a. As a developer,
        i.   I aim for the application to be cross-platform across Windows, MacOS, and Linux.
        ii.   I intend for the docx files provided to the user to be edited through the xml files that underlie the docx format.
        iii.   I plan to develop the application using C++ for both frontend and backend.
        iv.   I will utilize cross-platform frameworks such as Qt for development.

3. Performance
    a. As a developer,
        i.   I want the template filling response time to be under 500 ms.
        ii.   I aim for the application to use less than 300 MB of memory.
        iii.   I want the software to handle thousands of documents simultaneously.
        iv.   I expect the application to process and export up to 100 pages per document quickly.

4. Security
    a. As a developer
        i.   I would like to develop modern security features and protections
        ii.   I expect to provide a secure login page with password authentication.
        iii.   The passwords will be authenticated using modern standards for a "good and secure" password
5. Usability
    a. As a developer
        i.   I would like to develop and application with easy to navigate UI
        ii.   The application would also support a template up to 100 pages

# 2 Design Outline

Our program will be installed on the user's computer for quick and easy access. The program will have the option to locally store all files or utilize the user's cloud storage service, such as Google Drive or Dropbox. The user will also have the option to export their document to an email service of their choosing to easily send documents to clients. The program itself will have three main components: parser, input, and processor modules.

## Computer Application
We are utilizing a computer application as the interface for the user. This allows quick and easy access to the program in comparison to a web application where the speed is dependent on server resources and internet connection.

## Cloud Storage
When importing and exporting templates, the user will be given the option of using local storage or cloud storage. Instead of providing our own cloud storage solution, the user will be able to connect their pre-existing cloud storage, such as Google Drive or Dropbox. This will make it more convenient for the user. This also allows easy template sharing among employees in a company.

## Parser Module
When opening a template, it could take the application a long time to parse through and find the areas for input. To fix this issue, we will parse the template once when it is first uploaded. This parsing will save the input locations in a much smaller file for quick loading when the user selects the template for use.

## Input Module
Once the template is loaded, the user will need to input the information they want in the template. The information will be saved in memory to be used by the processor module. Of the three modules, this is the only one the user directly interfaces with.

## Processor Module
The processor module is used to tie everything together. It will utilize the user inputs, parser file, and duplicated template. Using these, the program will process all the information and create the final document from the user information. This will make the document ready to be exported.
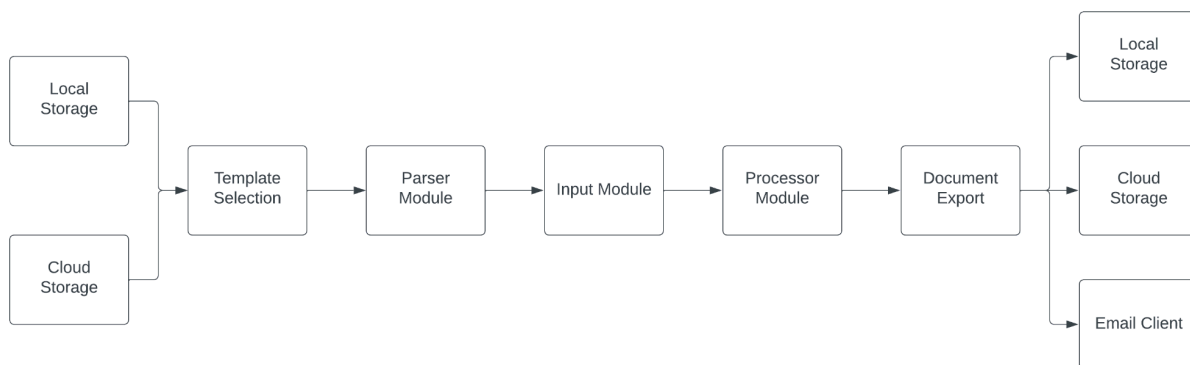
# 3 Design Issues

## Functional Issues

1. How should users export their document?
   a. Option 1: Save as a PDF and export the pdf manually
   b. Option 2: Output the document to a cloud storage
   c. Option 3: Export the PDF directly from the application to email
   d. Option 4: Option 1 and 2

Decision: We decided that it would be best for now to have the user save the document as a PDF and also export their document to a cloud storage. We decided not to directly email documents to others since we aren't familiar with a simple way to connect a user's email service (Gmail, Yahoo, etc.) with the application. Also, we believe that since cloud storage (i.e Google Drive) is fundamental to storing documents, we need to apply it to our application. Also, it's obvious that documents will be in your local machine.

2. How should a user login?
   a. Option 1: They can use the application anonymously (no login)
   b. Option 2: They can login with a separate account only for our application
   c. Option 3: They can login with their company credentials

Decision: We decided to let the users use the application anonymously with the option of syncing to their preferred cloud service (such as Google Drive or OneDrive). This is because having one account per organization, having one account per person, or having them log in with their company credentials all seems clunky and almost impossible to sync and integrate for all companies that use our product.

3. Should a user go through a tutorial for how to use the application?
   a. Option 1: Yes
   b. Option 2: No

Decision: We decided to have a tutorial for the application. This is due to some of the unusual behavior that our application contains such as our special language(eg. $firstName, $Date, $Employee_pay). Therefore, having a visually-guided tutorial could save a lot of time for our users when using the application.

4. Should a user be able to repeat the tutorial?
   a. Option 1: Yes
   b. Option 2: No

Decision: In case that the user needs a refresher or accidentally skipped the tutorial on the first time through, we decided it best that they would be able to repeat the tutorial as many times as they wish, probably launched from something like a settings menu. This also works well if multiple users from the same organization use the same computer.

5. How many documents should the user be able to export at once?
   a. Option 1: Yes
   b. Option 2: No

Decision: We decided that what could set our project apart from competitors and similar products could be its ability to export many PDFs from one template using multiple replacement texts for one input on the

template. That functionality is one of the core reasons our software is the better choice over something like Microsoft Word's find and replace feature.

## Non-Functional Issues
1. What programming language should our project be in?
    a. Option 1: C++
    b. Option 2: Python
    c. Option 3: Go


Decision: The programming language our implementation of EasyDraft is implemented in was a critical choice, but ultimately, we chose to base the whole project in C++ for a couple reasons. One of the main reasons we chose C++ was for its support for the object oriented programming paradigm, but even more important than that was its very impressive speed. To get our program to process the documents in the way intended consistently under 500 ms, we needed to choose a language fast for file processing, and we believe C++ is a great choice. We also chose C++ because the libraries for C++ for things like networking and graphics have existed for a longer time, so we believe they will be more stable and more feature-rich.

2. What GUI Library will we implement?
    a. Option 1: GTK
    b. Option 2: QT
    c. Option 3: wxWidgets


Decision: We decided for our graphical interface to use QT due to its popularity and because it also supports formatted text. wxWidgets and GTK don't have that advantage over QT.

3. How should we approach cloud support?
    a. Linking through services such as Google Drive
    b. Have a link with AWS
    c. Set up our own database through a server we set up
    d. Have companies link with their own cloud services

Decision: We decided to allow support by linking to services like Google Drive and OneDrive. We did this because businesses already have cloud storage solutions set-up, and for them to have to manage another storage service and account login would be cumbersome and annoying. Therefore, for convenience, they can sync to the cloud service of their choice.

4. Should the application be hosted on the web or through a local application?
    a. We host it on a website
    b. We set up a client-side graphical application

Decision: Because the application is going to be parsing through local files on the client's system, we decided it best to have the application hosted on their own computer. We also chose this for the reason that the application would be more secure if the only times it connects to the internet is to sync with cloud services. That way, with less traffic, our client will be less prone to hacks such as networks intercepting traffic. This is of high importance because often businesses have sensitive documents they send around.

# 4 Design Details
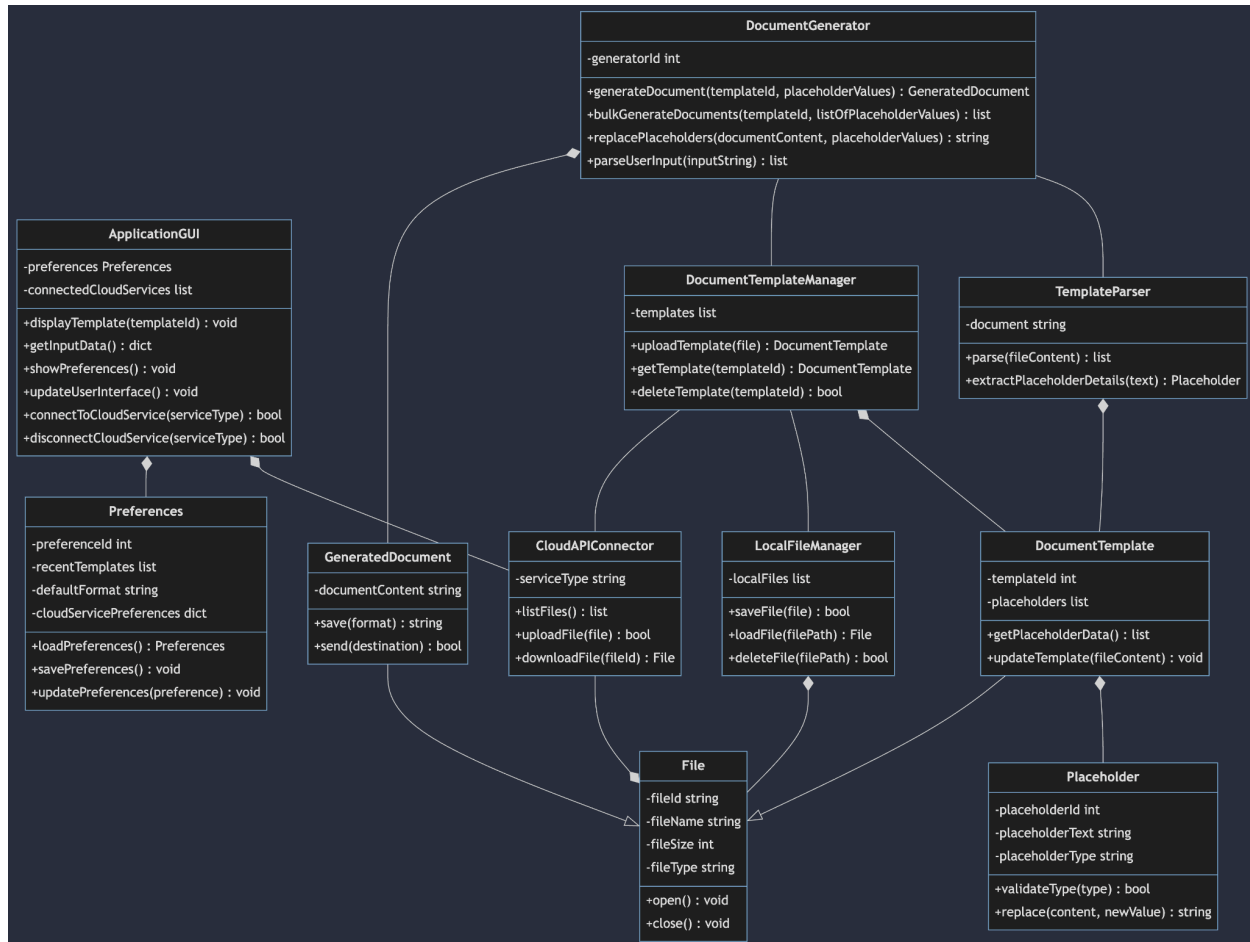


1. **ApplicationGUI:**
   - **Description:** Acts as the primary interface between the user and the application. It manages user interactions, displays templates, collects input data, and allows users to access and modify preferences and connect to cloud services.
   - **Interactions:**
     - Directly interacts with **Preferences** to load and save user settings.
     - Utilizes **CloudAPIConnector** to list files, upload templates, and download templates from cloud services like Google Drive or Dropbox.
     - Interacts with **DocumentTemplateManager** to retrieve and display document templates for the user to select and edit.

2. **LocalFileManager:**
   - **Description:** Manages files stored locally within the system. It handles operations such as saving, loading, and deleting files.
   - **Interactions:**
     - **DocumentTemplateManager** uses **LocalFileManager** to save and load document templates locally.
     - **CloudAPIConnector** may interact with **LocalFileManager** to download files from the cloud and save them locally or to upload local files to the cloud.

3. **File**
    - **Description:** Acts as the abstract base class for various types of files managed within the system. It encapsulates common file attributes and functionalities such as opening and closing files, which are applicable across different types of files.
    - **Interactions:**
        - Serves as a foundation for **DocumentTemplate** and **GeneratedDocument** subclasses, providing shared attributes and methods.
        - Interacted with by **LocalFileManager** for basic file operations like saving, loading, and deleting.
4. **DocumentTemplate (Subclass of File)**
    - **Description:** Specializes the **File** class to represent document templates. These templates include placeholders that are to be replaced with dynamic content when generating documents. Document templates are the blueprints used by **DocumentGenerator** to produce customized documents.
    - **Interactions:**
        a. Managed by **DocumentTemplateManager**, which is responsible for uploading, updating, and retrieving document templates. This includes parsing the document for placeholders and storing template information.
        b. **TemplateParser** is used to identify and extract placeholders when a new template is uploaded or an existing template is updated.
        c. Serves as input for **DocumentGenerator** when creating **GeneratedDocument** instances by replacing placeholders with user-provided data.
5. **GeneratedDocument (Subclass of File)**
    - **Description:** Inherits from the **File** class to represent the final, user-customized documents generated by replacing placeholders in a **DocumentTemplate** with specific user data. Each **GeneratedDocument** is a unique instance created based on user inputs and the selected template.
    - **Interactions:**
        - Produced by **DocumentGenerator** through the process of dynamically replacing placeholders in a **DocumentTemplate** with actual data provided by the user. This process involves parsing the template, identifying placeholders, and substituting them with relevant content.
        - Once generated, these documents can be saved, exported in various formats (as supported by the system), or sent directly to the user. **LocalFileManager** or **CloudAPIConnector** can manage the storage and retrieval of these generated documents, depending on user preferences for local or cloud storage.
6. **Preferences:**
    - **Description:** Stores user-specific settings, such as recent template selections and default formats for documents.
    - **Interactions:**
        - Accessed and modified by **ApplicationGUI** when a user updates their preferences.
7. **DocumentTemplateManager:**
    - **Description:** Manages the lifecycle of document templates, including uploading, retrieving, updating, and deleting templates.
    - **Interactions:**
        - Interacts with **LocalFileManager** and **CloudAPIConnector** to save and retrieve templates from local storage or cloud services.

- ○ Uses **TemplateParser** to parse new or updated document templates for placeholders.
8. **TemplateParser:**
   - ● **Description:** Parses document templates to identify and extract placeholders that will be replaced with dynamic content.
   - ● **Interactions:**
     - ○ Used by **DocumentTemplateManager** to parse templates upon upload or update.
9. **Placeholder:**
   - ● **Description:** Represents placeholders within a document template that are to be replaced with dynamic content.
   - ● **Interactions:**
     - ○ Managed within **DocumentTemplate** instances. **DocumentGenerator** uses the information from placeholders to replace them with user input.
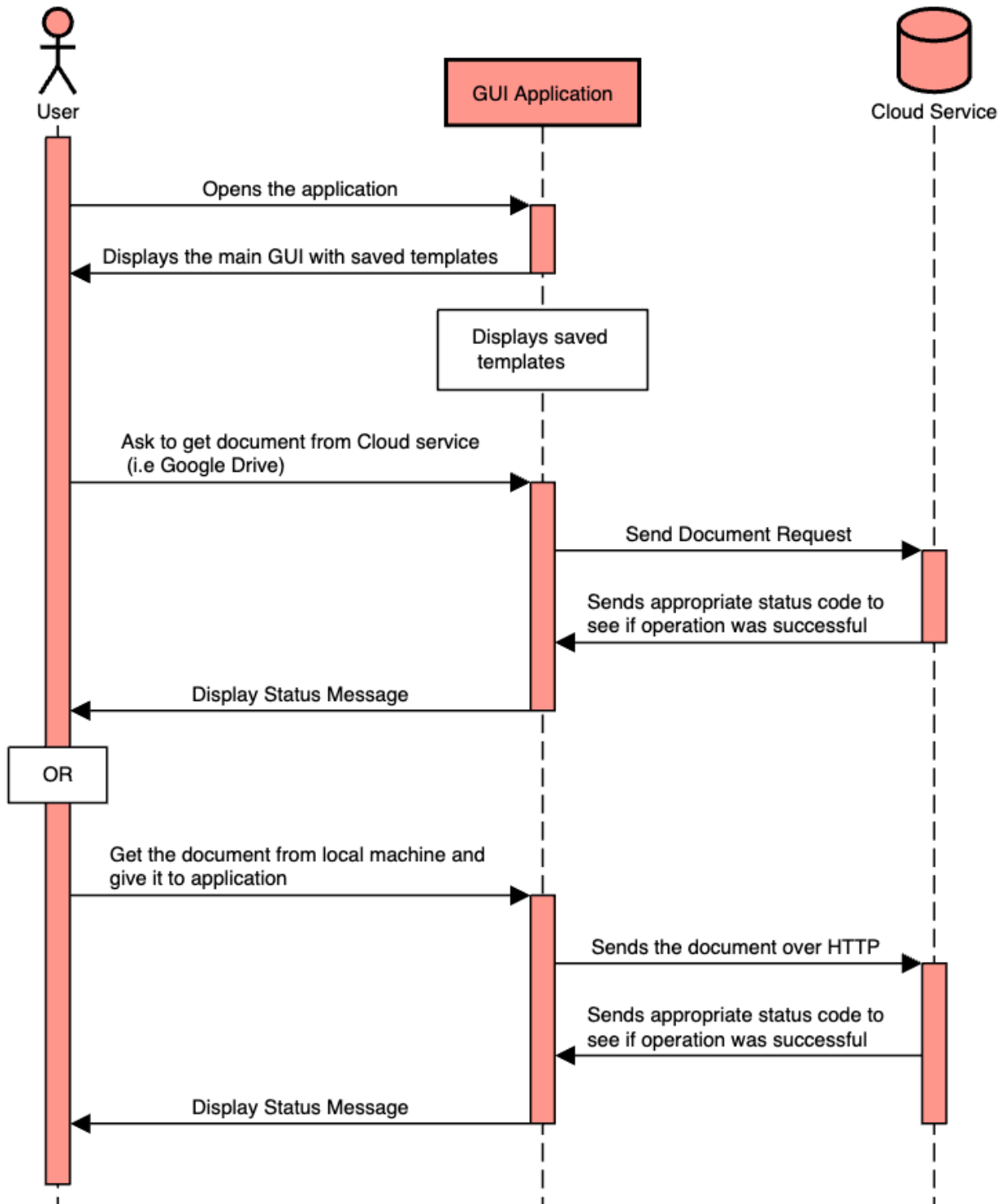10. **DocumentGenerator:**
    - ● **Description:** Responsible for generating final documents by replacing placeholders in templates with user-provided data.
    - ● **Interactions:**
      - ○ Retrieves templates from **DocumentTemplateManager**.
      - ○ Uses **TemplateParser** to ensure placeholders are correctly identified before replacement.
      - ○ Generates **GeneratedDocument** instances with customized content for the user.
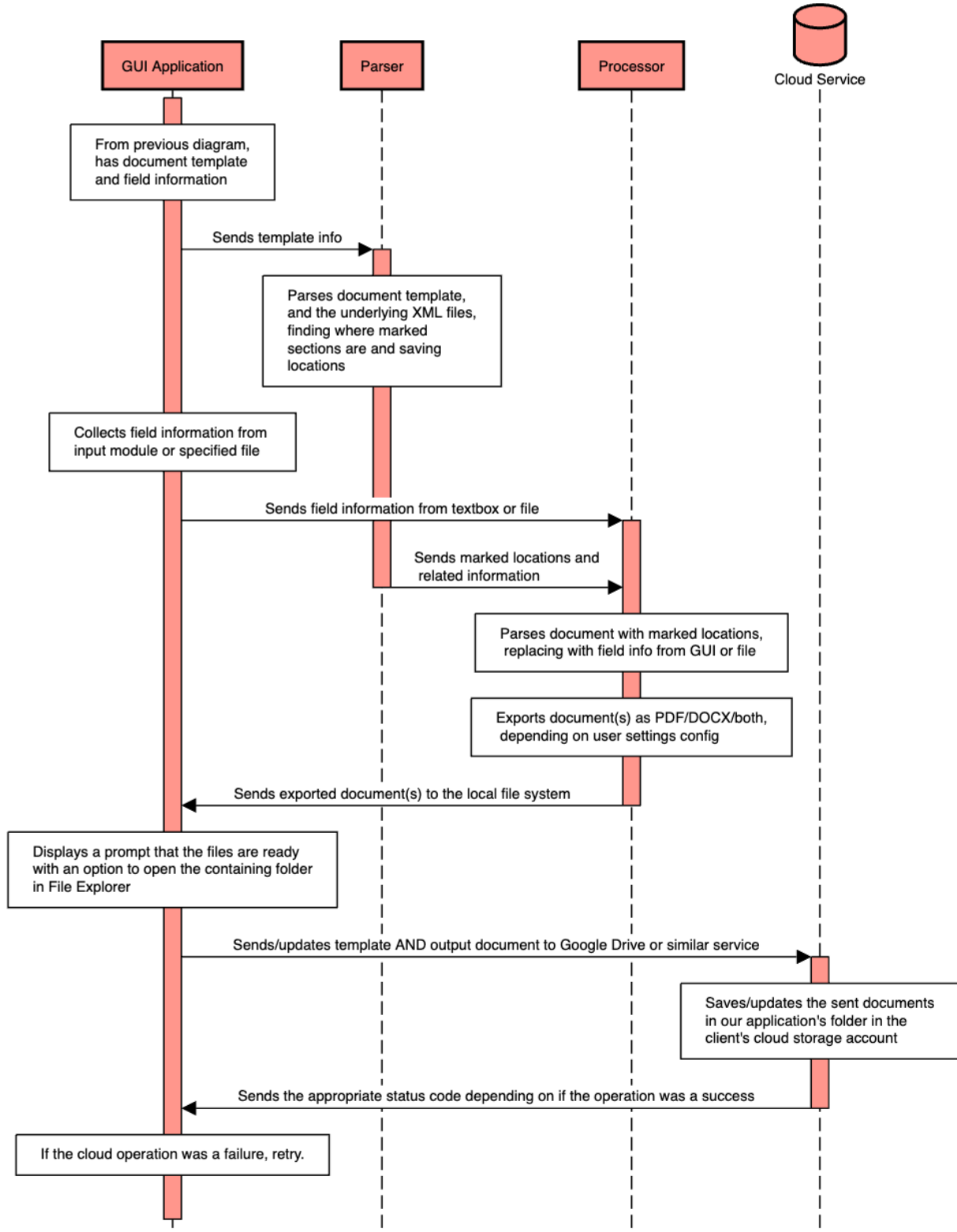11. **CloudAPIConnector:**
    - ● **Description:** Connects to external cloud services for file storage and retrieval, enabling operations like listing, uploading, and downloading files.
    - ● **Interactions:**
      - ○ **ApplicationGUI** uses **CloudAPIConnector** to provide users with the ability to save and access templates in the cloud.
      - ○ Interacts with **LocalFileManager** when downloading files from the cloud to local storage or uploading local files to the cloud.
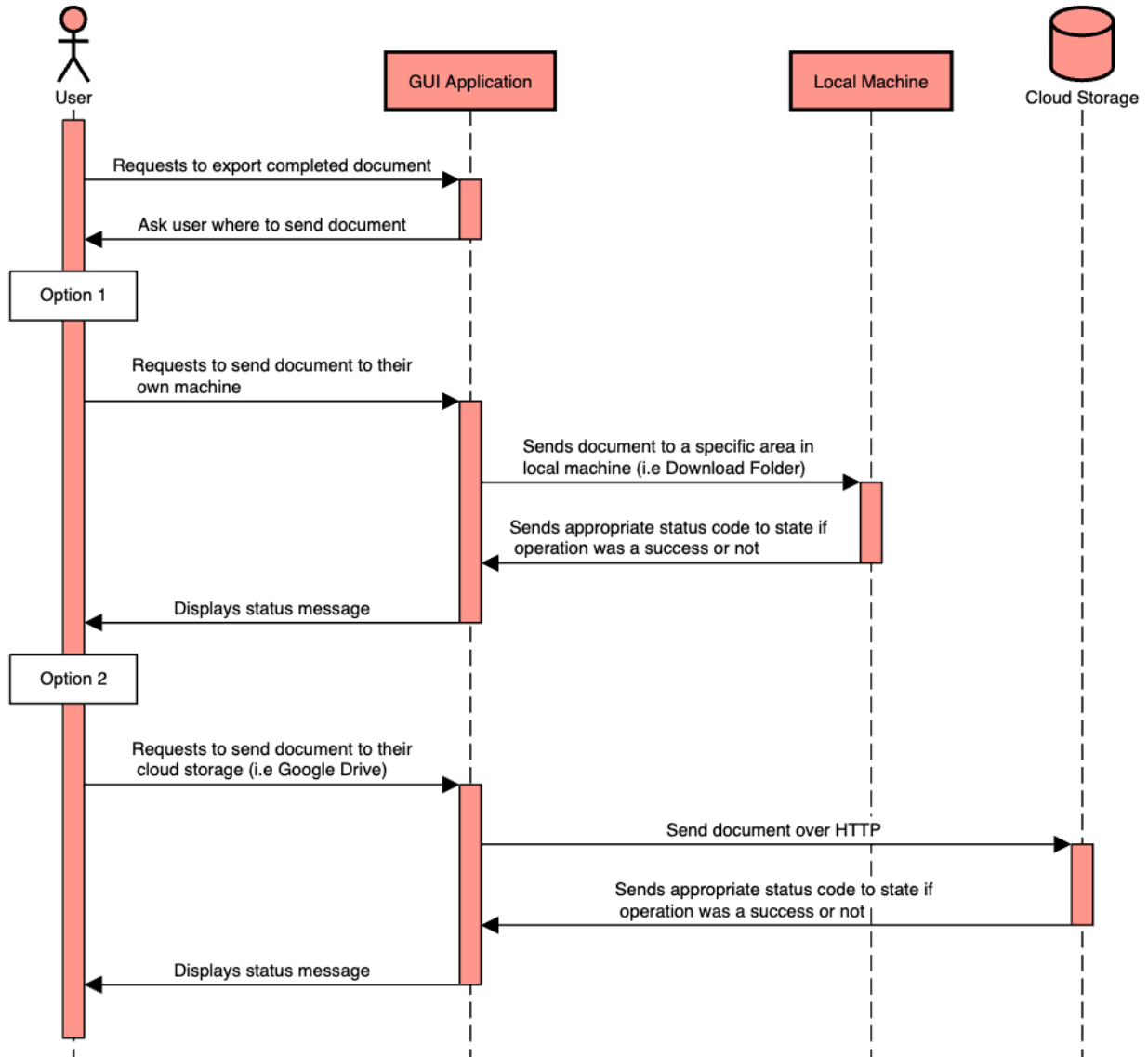
# Sequence Diagrams

## When a user starts the application
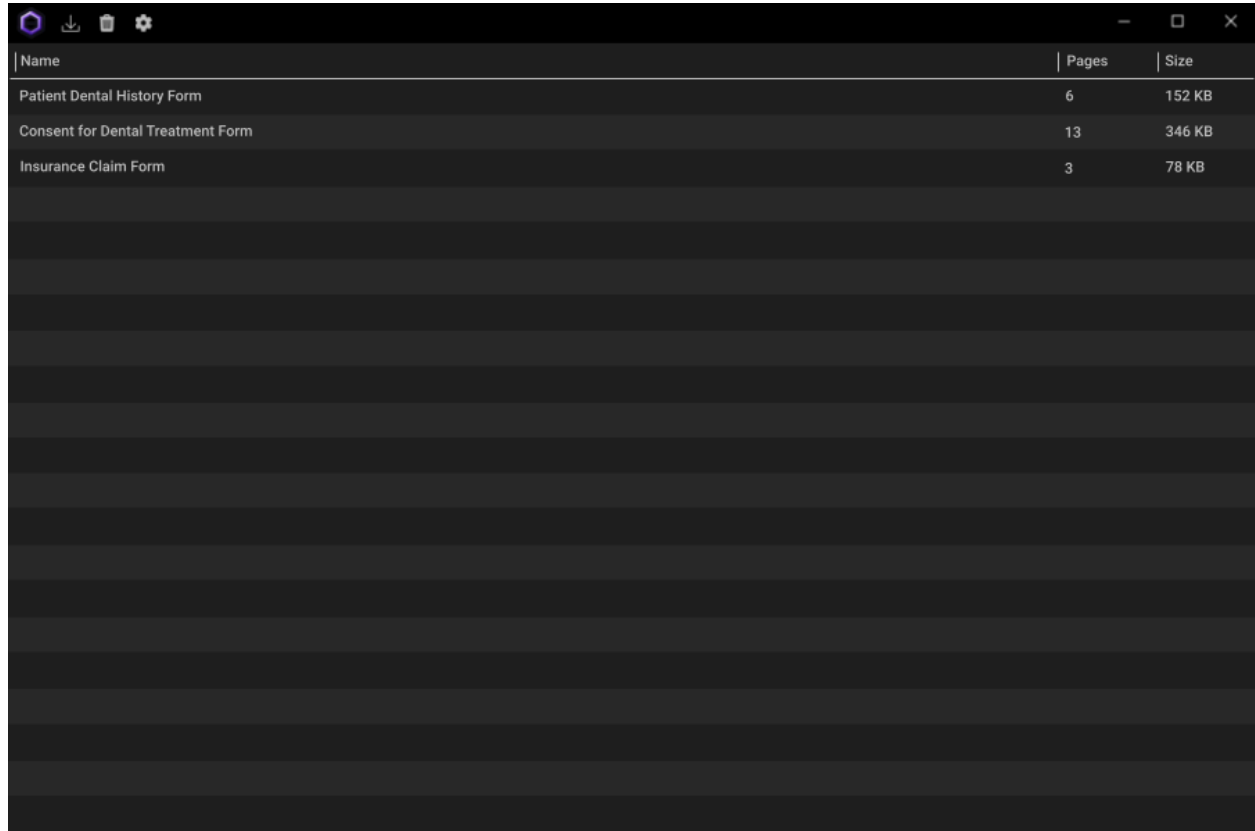
# File Parsing and Processing Step

**GUI Application** | **Parser** | **Processor** | **Cloud Service**

From previous diagram, has document template and field information

*Sends template info* →

Parses document template, and the underlying XML files, finding where marked sections are and saving locations

Collects field information from input module or specified file

*Sends field information from textbox or file* →

*Sends marked locations and related information* →

Parses document with marked locations, replacing with field info from GUI or file

Exports document(s) as PDF/DOCX/both, depending on user settings config

← *Sends exported document(s) to the local file system*

Displays a prompt that the files are ready with an option to open the containing folder in File Explorer

*Sends/updates template AND output document to Google Drive or similar service* →

Saves/updates the sent documents in our application's folder in the client's cloud storage account

← *Sends the appropriate status code depending on if the operation was a success*

If the cloud operation was a failure, retry.

# When a User wants to export template to pdf

User     GUI Application     Local Machine     Cloud Storage

Requests to export completed document

Ask user where to send document

**Option 1**

Requests to send document to their own machine

Sends document to a specific area in local machine (i.e Download Folder)

Sends appropriate status code to state if operation was a success or not

Displays status message

**Option 2**

Requests to send document to their cloud storage (i.e Google Drive)

Send document over HTTP

Sends appropriate status code to state if operation was a success or not

Displays status message

# UI Mockups

## Template Selection Screen
This is a UI mockup for the template selection screen. The overall design is supposed to be simple to keep it easy to use. It is designed after a file selection window. The top left of the screen has options for uploading templates, deleting uploaded templates, and settings. To access a template, the user just needs to click on the name. It will then take them to the screen for inputting their information for the template.

| Name | Pages | Size |
|------|-------|------|
| Patient Dental History Form | 6 | 152 KB |
| Consent for Dental Treatment Form | 13 | 346 KB |
| Insurance Claim Form | 3 | 78 KB |

## Input Screen

This is the screen where the users fill out the template, either by typing in the placeholders or importing a file with already saved placeholders.
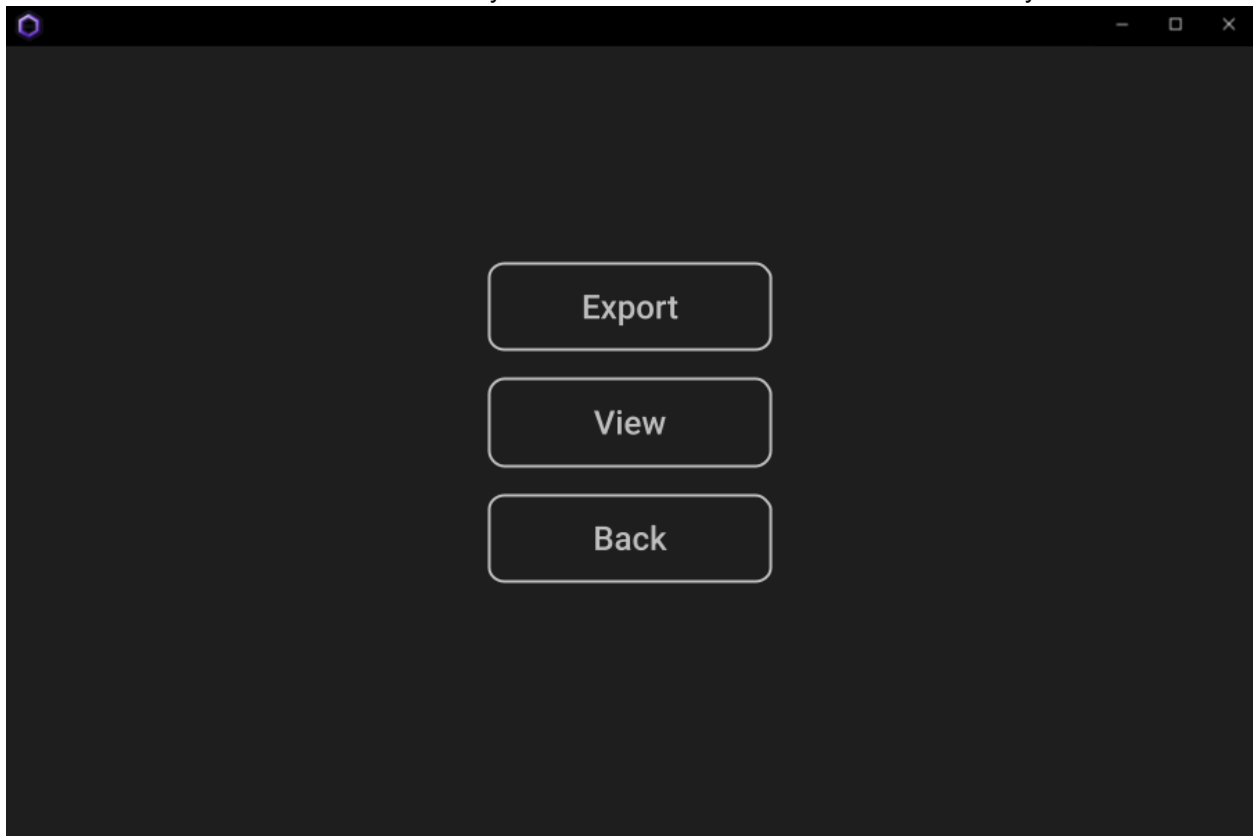
## Post Processing Screen

This is the screen that will be given after the user has inputted their data and the program has processed the document. On this screen, the user has the option of continuing to the export screen, viewing the document before exporting, and going back to change inputs. If the user selects "View", the document will be shown on their PDF viewer so that they can make sure the document looks how they want.

## Export Screen

This is the screen that is shown after choosing "Export" in the post processing screen. This will give the user the option to select where they want to export their document to. "Local" allows them to export it locally, "Cloud" allows them to export it to their cloud server, and "Email" allows them to export it to their email client.