MAY 2022

Project Report: Box.com/eDefender Integration and Document Tag Parser

PRESENTED BY

Daniel Guevara-Dominguez | Jesica Lopez De Leon | Luke Williams | Sergio Tapia | Shaocheng Shi | Chuang Huang | Marco De La Torre | Joshua Cabrera | Raul Gallegos | Dang Le

FACULTY ADVISOR

Dr. Jungsoo (Soo) Lim

SANTA BARBARA PUBLIC DEFENDER OFFICE

Deepak Budwani | Angella Stokke | Bryan Burzon | Sarah Rothschild | Aidan Bassett

DEPARTMENT OF COMPUTER SCIENCE

College of Engineering, Computer Science, and Technology
California State University, Los Angeles

Box.com/ eDefender Integration

Table of Contents

1. Introduction

1.1 Background

Santa Barbara Public Defender has transitioned to a fully paperless case management system. To support this new project, they are using Box.com as their cloud storage provider to store data. This requires over 50 terabytes of data to be moved to the cloud, and their data needs are increasing exponentially each year. As a part of this process, our team's objective was to assist Santa Barbara Public Defender with the integration of the Box cloud provider with their case management system eDefender. The role of our team was to provide a way for a transcription of case discovery media (video, and audio) to be created and then uploaded to the cloud. This would allow for a more efficient review of evidence by their attorneys and paralegals. Due to the complexity of this transition, contracts with Box and Azure are still pending and need to be finalized for this project to be completed. After discussing the options with our sponsor, we agreed to put this part of the project on hold to assist with a separate task, the Document Tag Parser mentioned above.

1.2 Design Principles

The Box.com/eDefender project utilizes Box.com, AWS Lambda, and Azure video analyzer to create a transcription for case discovery media. The project initiates when a user uploads a media file to Box.com which is then sent to the Azure video analyzer via AWS lambda, which analyzes the media file and generates metadata. This metadata is returned to the user providing valuable information on the media file. This functionality of this project is designed where the user should just upload a media file to Box.com. The main procedure of this project is processed by Box.com, AWS Lambda, and the third-party tool Azure video analyzer. This being the main design allows for the user to upload any file easily without having to worry about additional steps. Since the project heavily relies on Box.com, AWS Lamda, and Azure video analyzer it allows for a clear, quick, and easy way to have media files processed.

1.3 Design Benefits

Using external providers allows for seamless integration between Box.com and E-Defender with transcribing services. To set up the system the user simply needs to follow step-by-step instructions in order to connect the services. It is fast, easy, and requires little background knowledge to implement.

1.4 Achivements

Over the academic school year, our team was able to set up and create a test environment where this application was tested. The team was successfully able to upload media files to our application which then processed the request and returned valuable metadata. We then presented our application to the SBPD office and demonstrated its abilities. Additionally, we were able to update an existing user manual which was created by the previous team. By updating the user manual, we were able to create a step-by-step instruction manual to set up the application.

Once creating our test environment we then conducted meetings with one of the IT members at the SBPD office to discuss ideas to implement the eDefender notification system. Our team explained the details of the Box.com application and provided information on the systems architecture. By having various meetings we were able to come up with various ideas for this implementation. The IT team was informed of how this application worked and how it can possibly be scaled to include this eDefender notification system.

2. Related Technologies

2.1 Existing Solutions

CSU Los Angeles in partnership with the Public Defenders Office of Los Angeles created similar software for transcribing and processing video evidence. The work described in this document is an extension of that software and demonstrates how manageable and flexible our solution is.

2.2 Reused Products

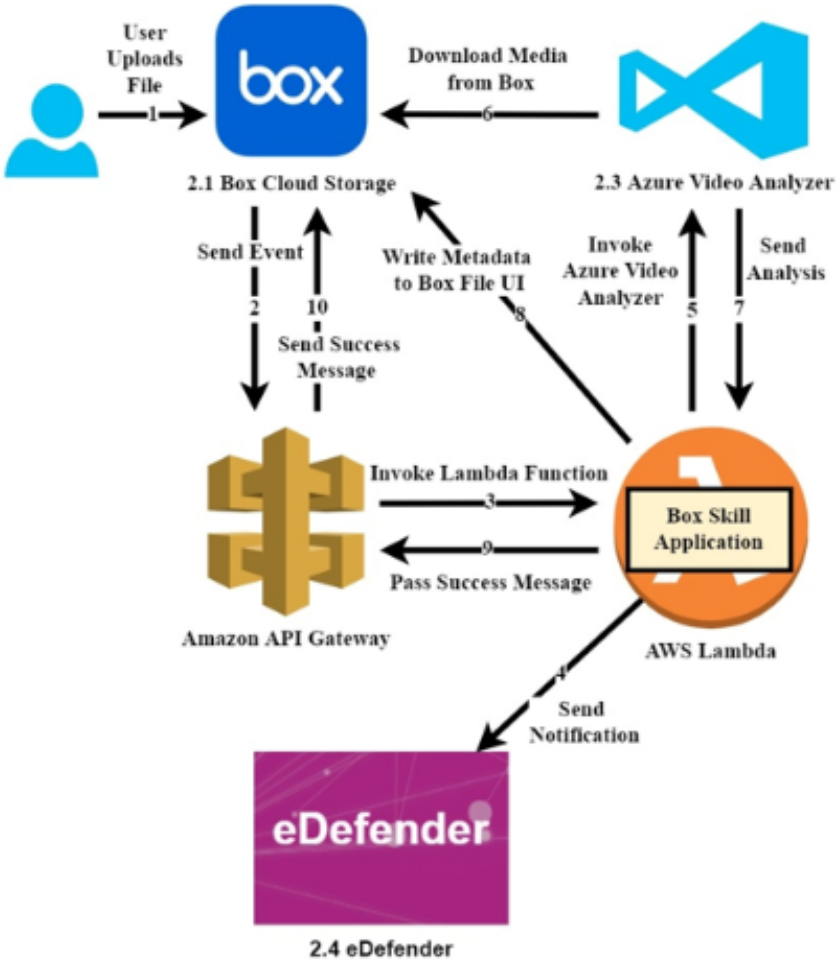
Los Angeles Public Defender Office and California State University Capstone 2020.

3. System Architecture

3.1 Overview

The architecture for this application is composed of five main components. These components consist of Box.com, Amazon API Gateway, AWS Lambda, Azure Video Analyzer, and eDefender. All of these components work together to receive and analyze media files, where valuable metadata is then returned to the user. Along with an alert system that notifies eDefender and its user.

The diagram below shows the overall architecture of the system and how these components work together to process requests.



3.2 Data Flow

There are five main components in this system, which are described in detail below. Here is an overview of how they function.

- **Box.com:**
  - 1**:** User uploads file to Box.com.
  - 2: The event is sent to the Amazon API Gateway
- **Amazon API Gateway**:
  - 3: Amazon API invokes Lambda Function.
- **AWS Lambda**:
  - 4: Notification is sent to eDefender
  - 5: AWS Lamda invokes Azure Video Analyzer.
  - 8: Metadata is sent to Box File UI.
  - 9: Success message is passed to Amazon API Gateway.
  - 10: Success message is passed to Box.com
- **eDefender**:
  - 2.4: Notification is received via AWS Lambda
- **Azure Video Analyzer**:
  - 6: The media file is downloaded from Box.com.
  - 7: Media file is processed and the analysis is sent to AWS Lambda.

3.3 Implementation

The project was split into three sections based on three platforms: Box.com, AWS Lambda, and Microsoft Azure Video Analyzer. All three work together to provide the transcription and facial recognition for media and audio files. Each section plays a key role in presenting, handling, and interpreting the files.

3.3.1 Box.com

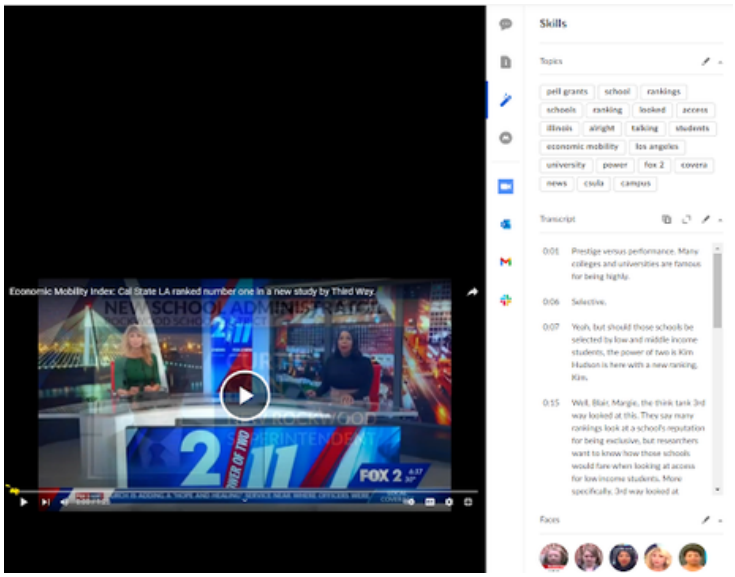
Box.com is the cloud based management platform that provides the cloud storage architecture for the Santa Barbara Public Defender office. At the same time, it is the starting point of this whole transcription and facial recognition process. It starts the process by triggering the Box Skill service hosted in AWS lambda and presents the metadata to users once they are finished.

3.3.2 AWS Lambda

AWS Lambda is the connection point where sensitive data has been transferred in and out between Box.com and Microsoft Azure Video Analyzer. It receives an access token from Box.com and gives it to the Analyzer for media/audio files access and writes metadata back to Box.com once the Analyzer finishes the analysis.

3.3.3 Microsoft Azure Video Analyzer

Microsoft Azure Video Analyzer is the last stage of the whole process. It will perform the essential machine learning algorithms on media/audio files uploaded to Box.com, and return that analysis back to AWS Lambda to generate the metadata.



4. Conclusions

4.1 Results

We have adapted code written by previous year students for a different project and made changes and updates to the user guide to ensure all the links are up to date. After we fixed a small error that only exists on Windows computers, we were able to deploy the serverless Lambda function onto AWS lambda and successfully transcribe a video that has been uploaded to Box.com

4.2 Future

The Santa Barbara Public Defender's office wanted to have an alert email sent to their system once the metadata is ready to review. However, due to time constraints, our team did not have a chance to start the alert system. Thus future teams can create this alert system to make the program more useful for the SBPD office because they and other law enforcement agencies could get updates immediately. Another possible improvement will be the language choices for the transcription. As of now, only English transcription is possible and it will be better if transcription in multiple languages is available for the SBPD office. When we discussed with the SBPD office about the most used languages in their discovery they said English, Spanish, and Mandarin. We believe if future teams want to implement other languages these should be the priority.

5. References

5.1 REST+API.doc

Link: https://csns.cysun.org/department/cs/project/view?id=7873056

5.2 user_manual_box_indexer.pdf

Link: https://csns.cysun.org/department/cs/project/resource/view?projectId=7873056&resourceId=7886247

5.3 Journal Tech

Link: https://journaltech.com/

5.4 Box

Link: https://box.com

5.5 Lambda

Link: https://aws.amazon.com/lambda/


Document Tag
Parser

Table of Contents

1. Introduction

1.1 Background

Bates Stamps are a common convention used to identify documents and aid in organization. Each page in each document has a number that is separate from an individual document's page number. This allows pieces of evidence to be more easily identified within the context of a given case. Our sponsor, the Public Defender's Office of Santa Barbara, frequently receives case evidence that uses Bates Stamps. Our sponsor renames each document based on the bates stamps found on the first and last page. By their estimate, their employees spend approximately 20 hours a week on this process. Due to the repetitive and time-consuming nature of this task, it is a perfect candidate for automation. Our team has designed an application to handle this process for them automatically and greatly improve effeciency.

1.2 Design Principles

Document Tag Parser is a Python-based desktop GUI application that utilizes several popular libraries to assist in the processing of pdf documents as well as images. It is designed to run on Windows 10, the operating system used by our sponsor. In addition to standard pdf text documents, it uses the machine learning based pytesseract library to convert pdf images, png, and jpg files to text for the parsing process. The libraries are grouped together in three separate modules: Text Parsing, Pytesseract, and the UI Module. Please refer to the system architecture section for a detailed explanation of these modules.

It is also important to note that this application is intended to be easily modified to handle changing business requirements. The overall architecture and the simplicity of the python programming language makes it easier for developers to quickly understand the data flow and application logic.

For example, during our final release and testing, Santa Barbara Public Defender made us aware of a strange edge case. In one document, the Bates stamps were rotated 90 degrees instead of being printed in a left to right orientation. This issue will likely require a major update and could not be completed due to time constraints. However, the problem was caught by our error log. While this issue is addressed through future work, the user will be made immediately aware so it may be named manually.

1.3 Design Benefits

Python is an easy-to-use and widely supported programming language. By choosing it as our platform, we gained access to many libraries which made the application possible and greatly reduced the potential development time.

The addition of multi-threading was a major key in improving our initial design of the application. We found the processing of image-based documents to be especially slow. By dividing the processing work among four threads, we were able to take advantage of powerful modern processors found in most desktop computers. This greatly reduced the processing time.

Another issue with our initial design was that the UI would become unresponsive during processing. To fix this, we added a fifth thread that only runs the UI module. It also supports a loading bar in order to show the status of processing. Finally, we added an error log to make the user aware of any edge cases or problems that had occurred during processing.

1.4 Achievements

Santa Barbara Public Defender's office roughly spends 20+ hours per week manually checking for court document bate stamps and renaming their files. Instead of spending 20+ hours doing it manually, with the Document Tag Parser Application, SPBD will spend less than 3 hours per week. The Document Tag Parser Application will process folders and files in under one minute utilizing multithreading saving time for productive work at SPBD. It is also 99.99 percent accurate and will help against human error with its error log.

2. Related Technologies

2.1 Existing Solutions

Document Tag Parser is a custom purpose-built application specifically made to handle the renaming of files based on Bates Stamps. To date, we are not aware of any applications which include all the features our program does.

2.2 Reused Products

Our application uses multiple libraries which handle key parts of our application. Without them, our application would not have been possible to develop in the timeframe we had. We would like to take this opportunity to thank the developers of the following packages:

- Pytesseract OCR
- Pdf2image
- PyMuPDF
- Tkinter

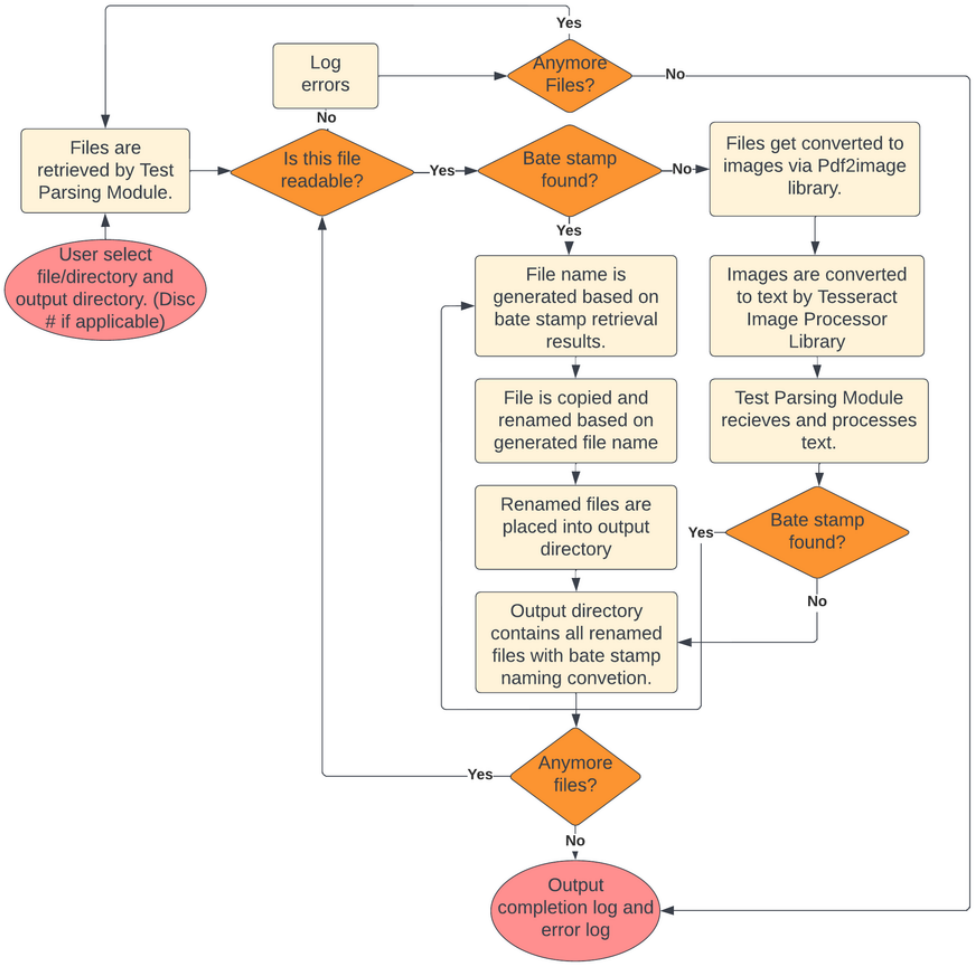
3. System Architecture

3.1 Overview

As discussed earlier, our application is divided into three distinct modules: text parsing module, pytesseract, and the UI module. The text parsing module interacts directly with the GUI to show the status of files and folders selected for processing. It uses the poppler pdf library to manipulate documents and relies on multiple threads to perform processing.

Some pdfs are image-based, and the text is not readable. Pytesseract is a third-party library that handles the conversion of image-based pdfs into text files. This makes the text machine readable for parsing to be performed by the text parsing module.

The UI module uses the Tkinter framework to render an easy-to-use visual interface. It uses a folder of custom assets consisting of icons, buttons, and other decorations. This module also runs on its own separate thread.



3.2 Data Flow

Each file opened by the Document Tag Parser has a new copy made and stored in the directory specified by the user. No files are modified, and any temporary files created are processed for Bate stamps only. For more details on this process, see System Architecture, section 3.1.

3.3 Implementation

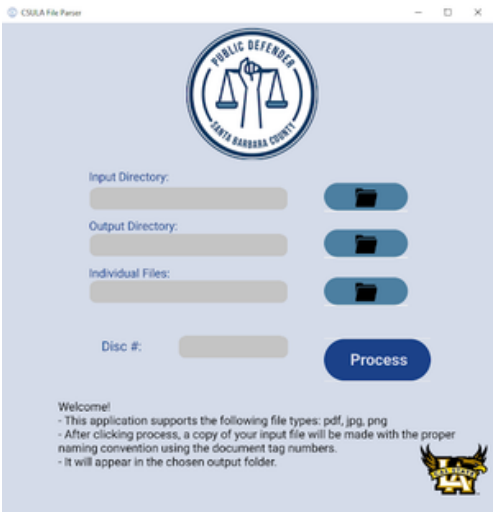
The document parser app was divided into four parts the UI/UX, the backend involving algorithms, testing, and deployment. Each part was important to achieve our requirements and get our final results.

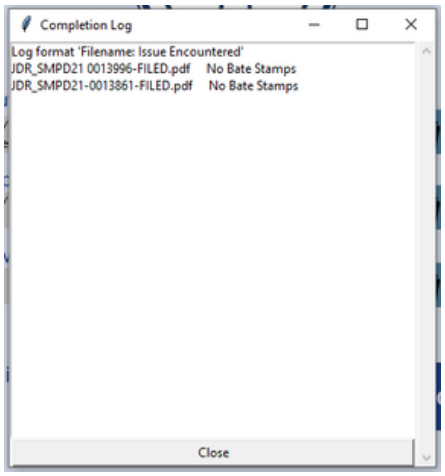
3.3.1 User Interface

The user interface was created to allow users to easily understand and access the functions of the app. The head image shows it's a program sponsored by Santa Barbara Public Defender. Under the image, four fields and their corresponding buttons simply tell the user what their functions are. The Completion log will pop up as soon as the program finishes the processing, showing what has been achieved and what went wrong. The CSULA logo on the bottom right represents those who implement the application.

3.3.2 Testing

For testing purposes, the Santa Barabara Public Defender's office provided 27 folders. Within each folder, there were two additional folders that contained the non-renamed files and the correctly renamed files. Our goal was to process the non-renamed files and accurately match them with the correctly renamed files. These 27 folders consisted of real documents that had many types of formats. Within each folder, there were text-based files, image-based files, and files that were not supported by our application. Once testing all 27 folders our team resulted in an accuracy of 90%. After reconstructing our algorithm we were able to reach an accuracy of 99% for these testing folders. Each folder was processed in roughly under one minute.



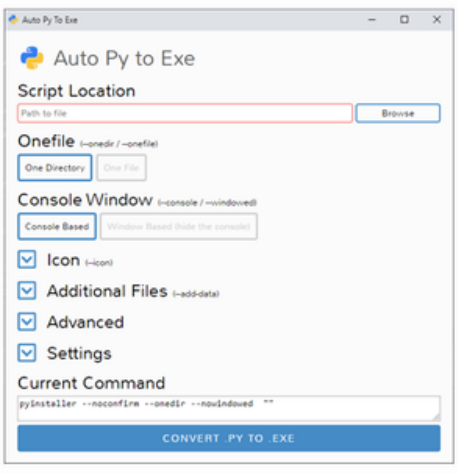


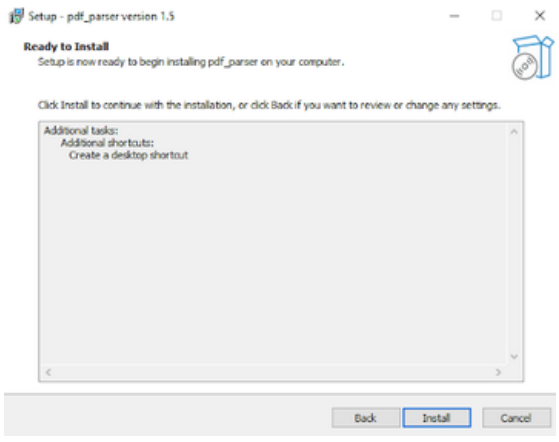
3.3.3 Backend Algorithms

The backend functions consisted of various algorithms that helped process the user's request. After reading pdf text files or pdf image files, all the string input will be read individually, and regular expression will be used to filter out all the strings containing exactly 6 digits. Once all the 6 digits strings have been stored into a list, another filter will be applied to pick up strings that start with at least two leading zeros. After two filter processes, the correct bate stamps will get picked up and stored in a list. To help with processing time, multiple threads have been utilized to process files in parallel time.

3.3.4 Deployment

In the beginning, the program was released as an executable program using pyinstaller with a folder containing all the necessary design files. Later, the team found a better way to make the process of finding the executable program within the folder easier by creating an installer using InnoSetup. With the installer, the user can simply install the program onto their computer just like they normally do with any other software.





4. Conclusions

4.1 Results

Our team fulfilled all requirements stated by our application sponsor. Document Tag Parser is a fully functioning desktop application that can be easily installed on any Windows computer using the installer found on our CSNS page. Large batches of files can be processing and renamed in under a minute and is approximately 99% accurate. According to our sponsor, when taking human error into account, this is a great improvement on manually renaming files. We have also made the source code available so that the application can be improved upon by both the Santa Barbara Public Defender's Office and future teams. We have also created user and developer guides in addition to the standard design and requirements documents.

4.2 Future

As discussed earlier in our report, there are edge cases which may be encountered in the future that we have not been able to anticipate. Case documents are sometimes delivered to our sponsor in an improper format. Sometimes bates stamps may be missing on certain pages, obscured, or unrecognizable due to printing and other external errors.

In addition, we know that currently Bates Stamps are 6 characters long and numeric only. it is possible that the format of the documents or the Bates Stamps themselves could change and cause the application's accuracy to decrease. This creates a potential for improvements and additional modifications to be done by a future CSULA team or our sponsor's developers.

5. References

5.1 pytesseract documentation

Link: https://pypi.org/project/pytesseract/

5.2 tkinter documentation

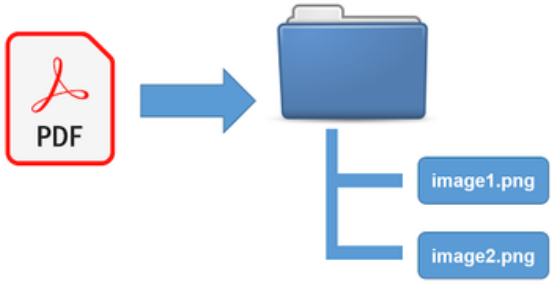
Link: https://docs.python.org/3/library/tkinter.html

5.3 pdf2image library documentation

Link: https://pypi.org/project/pdf2image/

5.4 Python Documentation

Link: https://www.python.org/doc/



Thank you!

Explore More.