Article to Powerpoint Converter
Design Document
July 13th 2023

Ahmed Shahid, 16904248
Aleksander Sienkiewicz, 210222490
Charles Rocchi, 210608250
Duaa Ghauri, 200658020
Omar Abdeen, 190777000

Context

The system we are building is a software that will convert a PDF article into a powerpoint presentation. The purpose of this product is to provide a quick and efficient way to convert articles into slides.The target consumer is anyone who needs a succinct way to give an overview of their work.

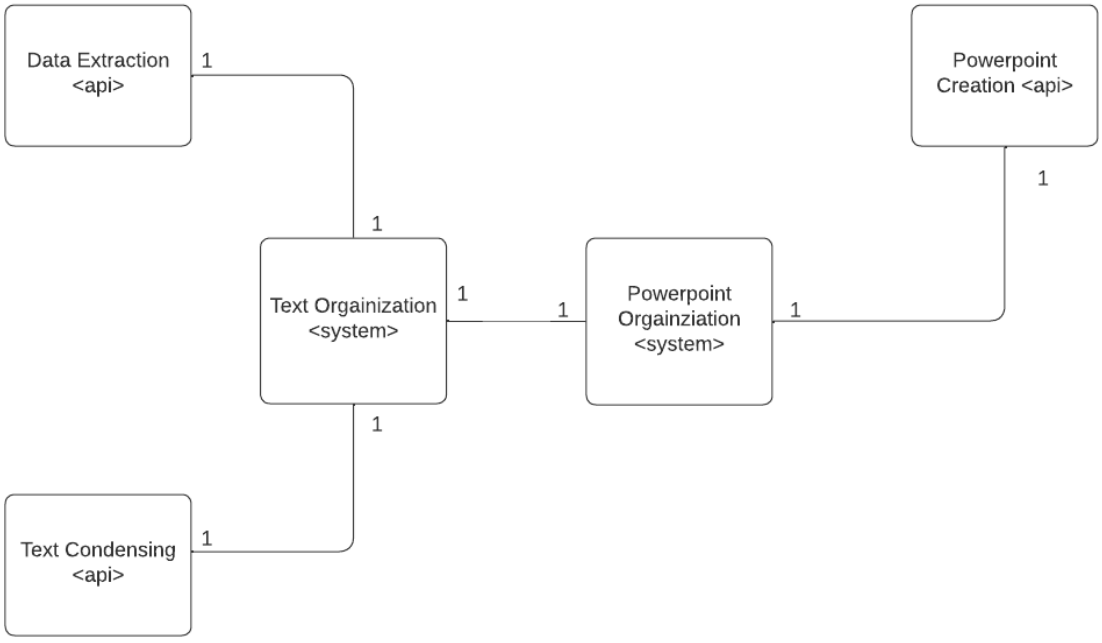
Boundaries

The boundaries we will set onto our project is that the extracting of data, the condensing of the text, and the creation of the powerpoint will be done by outer systems or apis. The operations we are creating will be more tailored towards organizing the data taken from the api's and putting it into a formatted powerpoint for the users. With this we will have 2 main classes, one that organizes the data and another that organizes the powerpoint. These are the main systems.

Target Environment

Our target environment will be a single threaded class based environment. The environment will be a folder of classes on Github. This folder can be downloaded and run on the user's computer using a function. These classes will be connected to the api's they need to use and connect to each other. The program will run simply by the user inputting the article they want to convert into a function. The program will then convert the article and give them the resulting powerpoint. This environment will also allow for our code to be taken and used by others to access and build upon the script. The user will have to download the needed apis to run the script. The instructions to download and run the script will be found in Github.

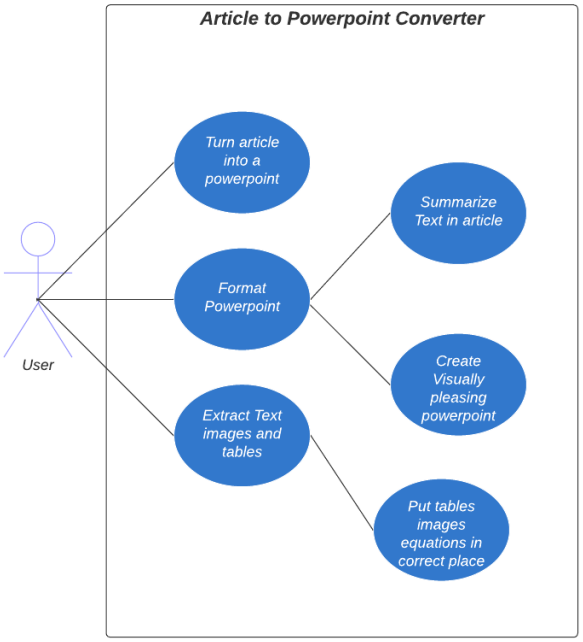
System Context Diagram



High Fidelity System Diagram

User needs

The users of our product are anyone who wants to convert a pdf article to a powerpoint. This specifically focuses on scholars such as professors, researchers and students. For instance a math student may have several equations in their article or a science student's article may be more image and diagram heavy. Our product must be able to extract as much data as possible from the article to create an accurate overview of it in a powerpoint format. The main four types of data that need to be extracted are text, tables, images, and equations. We anticipate that these are the types of data that will occur most frequently. The tables, images and equations must be put near the corresponding text to create a logical powerpoint. Another problem in converting an article to a powerpoint is that articles can be very long and if the powerpoint created is extremely long it can be confusing and unpresentable. This is why we need to summarize the text from the article reducing the slides to key points to create less cluttered slides. Lastly and most importantly the powerpoint needs to be organized and visually pleasing so that the end result is usable for the user.



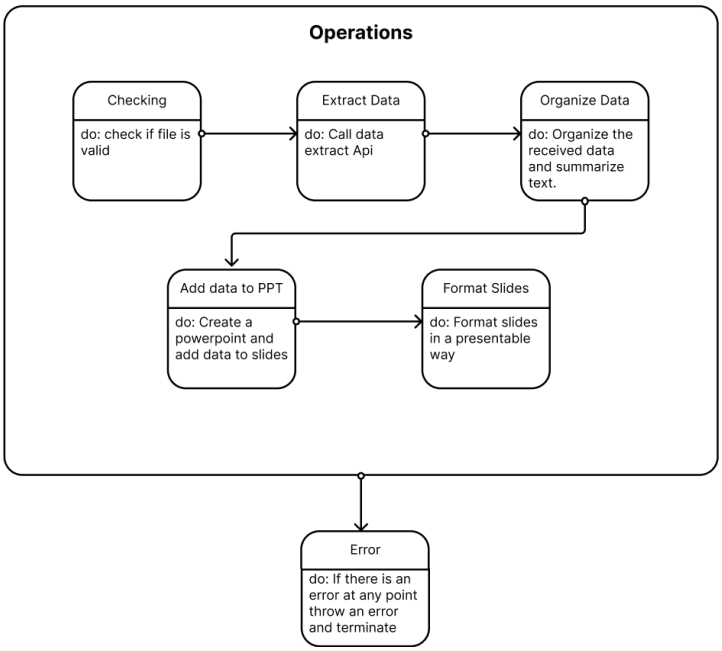
System	Article to PDF Converter
Use Case	Extract Tables and images
Actors	User (professor, scholar, student)
Description	The system needs to be able to find the images, tables and equations. Once found the system will extract the data and organize them in a way that can be correlated to the text.
Stimulus	Script has started and article being extracted from
Response	Arrays are created of each data type by page
Comments	System will not be able to account for data types other than the ones mentioned

System	Article to PDF Converter
Use Case	Format Powerpoint
Actors	User (professor, scholar, student)
Description	The powerpoint needs to be formatted in a way that is appealing and readable by its user. This means the system must make sure the text matches the tables, equations and images to prevent confusion. Also the text must be summarized in a way that users can read and decipher what is happening in the presentation
Stimulus	Data has finished being extracted from the article
Response	Text is summarized then put into the powerpoint and the images, tables and equations are inputted in their correct spots.
Comments	N/A

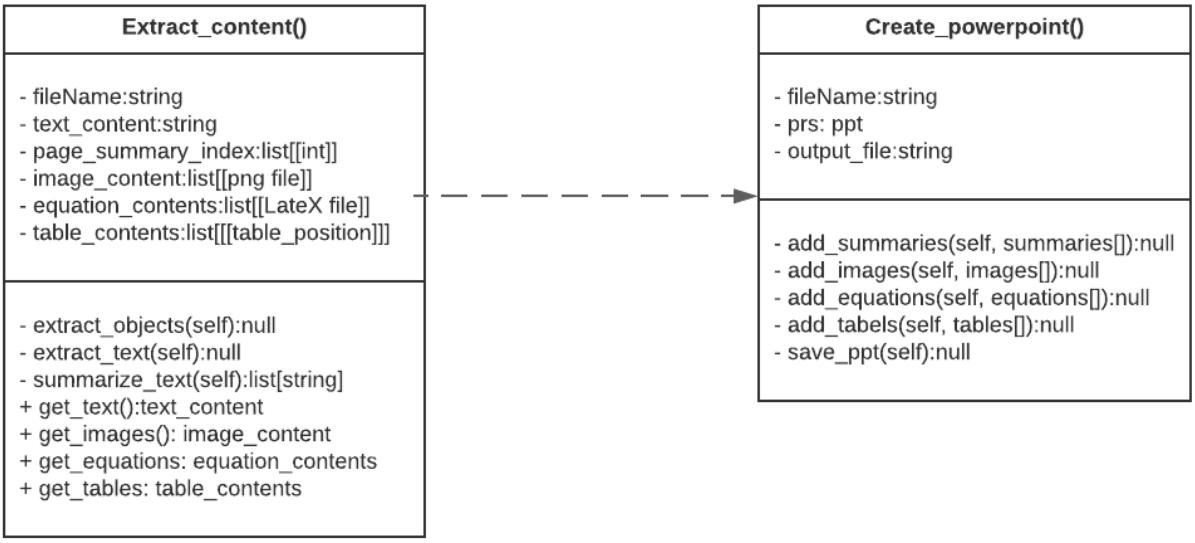
System State

The system we are building is a system that converts an article to a powerpoint. The basis of how this will work is that all required data will be extracted from the article in a way that can be used in the code (e.g. lists and strings). Once we have data that can be used by the code we

need to organize it in a way that can be put into the powerpoint. This is essentially making sure that the text aligns with the images and that the powerpoint can be produced in an order that makes sense. Lastly we need to condense the long text into something shorter and more readable for the powerpoint. This requires essentially 4 main states in our project: extracting data, organizing data after extraction, putting it into the powerpoint, and formatting the slides. If these states fail at any point they will throw an error terminating the process.



Classes



In our product we have 2 classes Extract_content and Create_powerpoint. The **Extract_content class** is used to extract the images, equations, tables and text from the pdf article given by the user. This class will also summarize the text to make it more presentable in the powerpoint. The dependent attributes are the file path of the article that it is to be extracted from and a list for each of the diagrams, tables and equation data types being extracted from the article. These lists are initially blank when the class is invoked. There is also an attribute called page_summary_index, where the ending of each summarized page will be. This is used to help put the images, equations, and tables near the text they correspond with.

The **extract_objects()** class is used to extract images, equations and tables from the article and populate the given attributes. We decided to extract all these types of data in one function to make our script more efficient as we can get all this data in one loop of the article. The function will populate all these attributes as a 3D array. The outer list will be the page that the piece of data is on and the inner list will be the actual pieces of data from the article in order of when they are seen in the page (top to bottom). This allows us to see where each piece of data is in the article and correlate that with the text when putting them in the powerpoint. Each of the different data types will be stored in different ways. The images will be stored as a .png, the tables will be stored as arrays and the equations will be stored in a LateX file.

The **extract_text()** function is used to extract the actual text of the article into string formatting stored in one large string. This function will also keep the index of where the page ends and divide that by 1024 letters which is the length of the summaries created in extact_summaries() to get the where the end of the page would be in relation to the summaries. This is used to align tables, images and equations with the text. Next the **extract_summaries()** function will take the output of extract text and create summaries. A summary is made for every 1024 letters. Each summary will get its own slide and will be stored in a list. In the Extract_content class all attributes, extract_objects(), extract_text() and extract_summaries will be private. This is because tampering with these will crash the code. There will still be functions to view attributes in case the user wants to see a specific data type.

The **Create_powerpoint class** will be used to create a powerpoint presentation, by putting the previously extracted data into that powerpoint and formatting it in a presentable way. The attributes of this class are the file name of the output presentation and a link to the powerpoint that the functions can use to reference the powerpoint. The **add_summaries()** function will be used to create a new slide for each summary in the list extracted from the article. This will be done before the other data is added to create a base for the slide. Then the images, equations and tables will be added using their respective functions: **add_images, add_equations and add_tables**. The format of each created slide varies depending on the data type that is being added. The position of the image, equation and table slides will be after the last summary corresponding to its page (page_summary_index). This makes sure they are in the relative area of their corresponding text. Slides can be moved around easily after the powerpoint is created by the user to perfectly match where these slides need to go. Lastly the **save_ppt()** function saves the powerpoint under the given output name allowing the users to access the completed

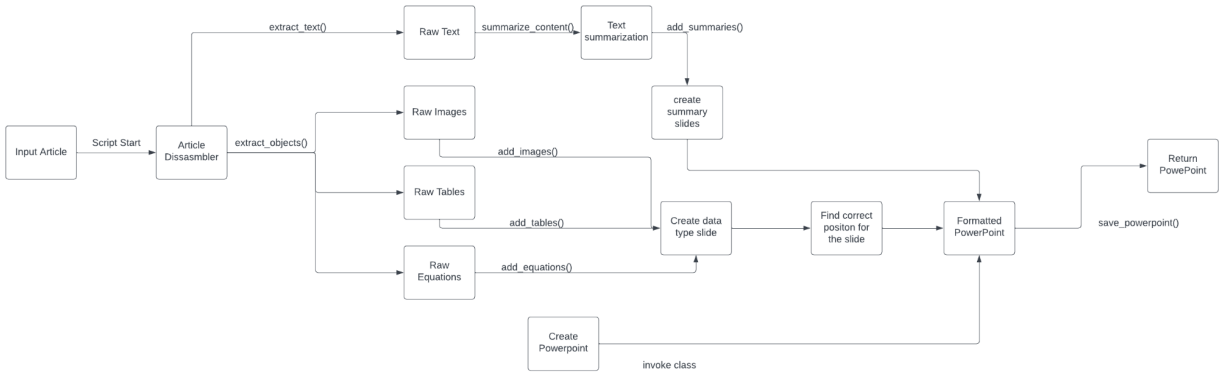
powerpoint. In this class all of the attributes and functions will be private to make sure our program is safe.

	Extract_text_content class
Attribute	description
- fileName:string	Is the path to the pdf article that is to be used by the program
- text_content:string	Is the raw text from the article. populated by extract_text()
- page_summary_index:list[int]	Is the summary that would represent the ending of a specific page. Used to organize data in powerpoint. Populated by extract_text()
- Image_content:list[[png]], - Equation_content:list[[LateX]], - Table_content:list[[[string]]]	These are the images, equations and tables from the article. Stored by page order then first to last seen. Populated by extract_objects()
Functions	
- extract_objects(self):null	Loops through the article and extracts the images, equations and tables. Put them in their respective attributes.
- extract_text(self):null	Loops through the article and extracts the text. Populates page_summary_index with (len(end of page)/ 1024) to correlate pages with the summaries.
- summarize_text(self):list[string]	Take the text extracted from extract_text() and divide the text in blocks of 1028 letters. Summarizes those blocks and returns a list of all the summaries
+ get_text(), get_images(), get_equations(), get_tables()	Allows users to see a specific type of data extracted from the article.

	Create_powerpoint() class
attributes	description
- output_name:string	What the output powerpoint filename will be called
- prs:powerpoint	Is a reference to the powerpoint that is being created and reformatted
functions	

- add_summaries():null	Creates a new slide for each summary in the list. Formats and add the respective text
- add_images():null - add_tables():null - add_equation():null	Creates a new slide for each data type. Put the slide behind the summary of the corresponding page. Formats and adds the respective data type.

Interactions



This sequence diagram shows how the script will run from start to finish and the order in which each function will be called. We first start with the user inputting the article to begin the script. The path of this input file will be taken by the script and used to open it. Next we will use the Extract_text_content class to extract the images, tables, and equations into a usable format. Then the text will be extracted and summarized. We must extract the text before summarizing because the text attribute will initialize as empty and the summarize function relies on the text. After the text extraction and summarization is finished we will invoke the Create_powerpoint class. This will automatically create a blank powerpoint which we will be formatting. We create a new slide for each summary. This must be done first because where we put the image, table and equation slides relies on the corresponding text slides. After the summary slides are created, the data type slides will be created and placed behind the last slide that corresponds with the specific page. Finally we will save the powerpoint which will upload the complete powerpoint to the correct output place allowing the user to see the powerpoint and ending the script.

Functional design issues

Problem 1: Many articles have different formatting which affects extraction.

Issue

- There are many different formatting methods for an article (ex IEEE or Springer). This can create issues when extracting from the article because the formatting is different. An example of us experiencing this error is when trying to extract a table from IEEE

formatting. In IEEE formatting the text is divided down the middle so the code saw the regular text as a table. This resulted in a combined table and text extraction thereby rendering the system useless.

Solution

- Since there are many different formatting methods for an article. It would be too time consuming to make sure that the system works for every formatting method. Because of this we decided to test and make sure our code works with the 2 main formatting styles: Springer and IEEE.

Problem 2: How do we select what goes on which slide?

Issue

- How will we figure out what content will go on each slide? The problem here is that we need the end powerpoint to be fluent and in order. We also need to make sure that there are not too many or too little slides.

Choices

- Divide text by page
- Divide text by paragraph
- Divide text by a certain number of letters

Solution

- We decided to divide slides into a certain amount of letters. The system will divide the text by 1024 letters. After that the system will summarize the chunk and use the summary as an individual slide. We decided that 1024 was a good number as it gives us enough content to summarize in one slide and will make around 2-3 slides per page. The summarization of the text allows us to make sure that the powerpoint is fluid and that each slide plays into the next. We chose this over the other options because a whole page is too much to summarize on one slide; this would lead to vague and convoluted slides. Dividing by paragraph would have been good but it is very hard for the machine to decipher where a paragraph ends. Dividing by a certain number of letters is quicker than dividing into paragraphs.

Problem 3: How do we figure out where each image, table, and equation will be placed?

Issue

- Because the tables, images and equations are being extracted separately from the text we do not know the exact position of each item. Also because the text is summarized outside of the raw text article, the text may not be exactly the same description of the table, image, or equation.

Choices

- Use keywords in the text to place images
- Place them where the page would end

Solution

- When extracting the images, tables and equations the function tells us what page each item is on in order of first to last. When extracting the text we figure out where each page ends corresponding to the summaries using the formula len (end of page)/1024. Because of this we can put the images, tables, and equations of that specific page

directly behind the slides correlated to that page. This is not perfect as the items are not in the exact position they need to be. We thought this would be better than to use keywords because it is more consistent. If there is a large article with few pictures then using keywords to associate with images may place them in the completely wrong spot.

Problem 4: There is no UI for the project

Issue

- A UI for the project would make it a lot more user friendly and visually pleasing for people who are not as good with computers. The problem is that within the time constraints there may not be enough time to create the UI.

Solution

- If there is time after all the mandatory code is tested and working, we will work on a UI.

Non Functional design issues

Problem 1: The system is very slow

Issue

- The system is very slow, taking several minutes to run if the article is larger than 10 pages. The part of the system that is slowing it down is the summarization of the text. This is done by an api so we cannot speed this up by improving the code.

Choices

- Not use text summarization
- Write a fast text summarization code
- Keep using slow api

Solution

- We decided to keep using the slow system because we feel that the summarization of the text is very important. We do not have time to write our own faster code to summarize the text instead of the API. One thing we did change is that we now send all of the text to be summarized at once instead of each page separately so now we only have to call the api once. This speeds up the process.

Traceability Matrix

Requirement ID	**Requirement Description**	**Design Elements**	**Class**	**Method**
REQ-01	Extract images from PDF	Extract_images	Extract_content	extract_objetcs()
REQ-02	Extract tables from PDF	Extract tables	Extract_content	extract_objetcs()
REQ-03	Extract equations from PDF	Extract equations	Extract_content	extract_objetcs()

REQ-04	Extract text from PDF	Extract text	Extract_content	extract_text()
REQ-05	Summarize extracted text	Summarize text	Extract_content	extract_summaries()
REQ-06	Create a new PowerPoint file	Create PowerPoint	Create_powerpoint	add_summaries()
REQ-07	Add summaries to PowerPoint	Add summaries	Create_powerpoint	add_summaries()
REQ-08	Add images to PowerPoint	Add images	Create_powerpoint	add_images()
REQ-09	Add equations to PowerPoint	Add equations	Create_powerpoint	add_equations()
REQ-10	Add tables to PowerPoint	Add tables	Create_powerpoint	add_tables()
REQ-11	Save and download the PowerPoint	Save PPT	Create_powerpoint	save_ppt()