EXP. NO: 1 **DATE:**

PROBLEM STATEMENT

AIM :

To write a problem statement for the mini project

PROBLEM STATEMENT:

Currently there are many online converters available to convert various file types into desired file type. Conversion is a simple thing but in a workspace it makes big deal and till now there is no such tool which would combine all the file conversion mechanisms and provide it as single application. Though some exist in market, those are paid and not many people could afford.

The sole aim of this project is to create an opensource all-in-one file converter and make it available to everyone. And we try out best to make it happen.

RESULT :

Problem statement for the mini project is successfully completed

EXP. NO: 2 **DATE:**

SOFTWARE REQUIREMENTS SPECIFICATIONS

AIM :

To write a Software Requirements Specification for the toolkit application.

PROCEDURE :

Steps you can follow to write an effective SRS document

1. Introduction
   - 1.1. Purpose
   - 1.2. Intended Audience
   - 1.3. Intended Use
   - 1.4. Scope
   - 1.5. Definitions and Acronyms
2. Overall Description
   - 2.1. User Needs
   - 2.2. Assumptions and Dependencies
3. System Features and Requirements
   - 3.1. Functional Requirements
   - 3.2. External Interface Requirements
   - 3.3. System Features
   - 3.4. Non Functional Requirements
4. System Features
   - 4.1. System Feature 1
5. Other Non Functional Requirements
   - 5.1. Performance Requirements
   - 5.2. Safety Requirements
   - 5.3. Security Requirements
   - 5.4. Software Quality Attributes
   - 5.5. Business Rules
6. Other Requirements

Appendix A : Analysis Model

Software Requirements Specification

for

Toolkit

Version 1.0 approved

Prepared by Dheephiga A M, Maarcus Reniero L

Rajalakshmi Engineering College

09-03-2022

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1. Purpose

This SRS covers all the major features and requirements of the Toolkit software in a detailed manner. This documentation gives an overall idea about the software.

1.2. Document Conventions

This SRS follows standard documentation. Italics are used to highlight specific requirements and features. Bold text formatting is used for headings and subheadings.

1.3. Intended Audience and Reading Suggestions

This SRS documentation is mainly intended for developers,marketing managers,testers and documentation writers. Every headings in this SRS are arranged according to the index on the first page. Readers are recommended to go through the index before reading the documentation for better understanding.

1.4. Product Scope

The main purpose of this software is to provide users an all in one software for document, speech to text, compressor and media like audio and video conversion purposes. This software will have a huge potential for buyers and will gradually generate a considerable amount of revenue in a very short period of time for the company.

1.5. References

https://betterprogramming.pub/build-a-pdf-merger-in-python-c582607a9d29

https://pypi.org/project/AudioConverter/

https://pypi.org/project/Pillow

2. Overall Description

2.1. Product Perspective

In this project, the main aim is to enable the user to perform conversions locally instead of depending on a website that the majority use, which restricts users to upload the files for a particular threshold and also has daily limits for conversion per user.

2.2. Product Functions

We are using functions such as

- *Graphical User Interface*
- *Conversion*
- *Multiple convertors in one place*
- *Downloading the file after conversion*

2.3. User Classes and Characteristics

There is only one user. The user can use the system to convert files from various sources.

2.4. Operating Environment

- *Windows 7 or above*
- *Linux 18.04 or above*

2.5. Design and Implementation Constraints

User Interface: This offers free online conversion services for a wide range of file formats and units of measurement, this is a versatile and multipurpose converter, a handy tool for your work and personal life.

2.6. User Documentation

In this software, we prepare a help page so that the user can read it and can execute the software properly. It can be seen at the top of the Menu page labeled 'Help'. This page will help the users to work the software.

2.7. Assumptions and Dependencies

There are many dependencies and assumptions associated with the software. They are

- *Every user is expected to have a valid file*
- *The user can do unlimited conversions.*

3. External Interface Requirements

3.1. Functional Requirements

- *A window to upload the required files which needed to be converted.*
- *An option to choose the download path*
- *A functionality to download the file multiple times*

3.2. User Interfaces

The interface between the user and the system include many provisions from where they can access the whole system. It contains

- *4 options - Document Converter, Media Converter, Speech to Text Converter and Compressor*

3.3. Hardware Interfaces

- *Processor, Pentium G and above*
- *Minimum 1 GB RAM*

3.4. Software Interfaces

- *Operating System: Windows 7 and above*
- *Python*

3.5. Communications Interfaces

Network server communications protocols are required.

4. System Features

The Toolkit Conversion Software is loaded with a lot of features for its users. Furthermore new features will be added in the upcoming updates based on user experience. Major features of the software are mentioned below.

4.1. System Feature 1

4.1.1. File Converter:

Convert your video, audio, images, e-books and documents to another file format.

4.1.2. Free to Use:

This is a completely free service

4.1.3. Simple:

Without need to download and install any software, safer for you

4.1.4. Convenience:

You can access this online service from your PC or mobile anytime and anywhere.

4.1.5. Easy To Use:

With a few mouse clicks, you can easily convert your files by yourself.

5. Other Nonfunctional Requirements

5.1. Performance Requirements:

The performance of the system lies in the way it is handled. Every user must be given proper guidance regarding how to use the system. The other factor which affects the performance is the absence of any of the suggested requirements.

5.2. Safety Requirements

Regular safety checkups and monitoring of the device to ensure proper working of it User should be aware of the malfunctions and should have the basic ability to solve the issues that occurred.

5.3. Security Requirements

Unauthorized users should be prevented from accessing the data.

5.4. Software Quality Attributes

5.4.1. Quality

This software provides 99.99% quality to users' data.

5.4.2. Niche Focused

Each of our 11 tools follows exactly one purpose. For instance, our online file converter converts files and our PDF editor allows us to edit PDF files.

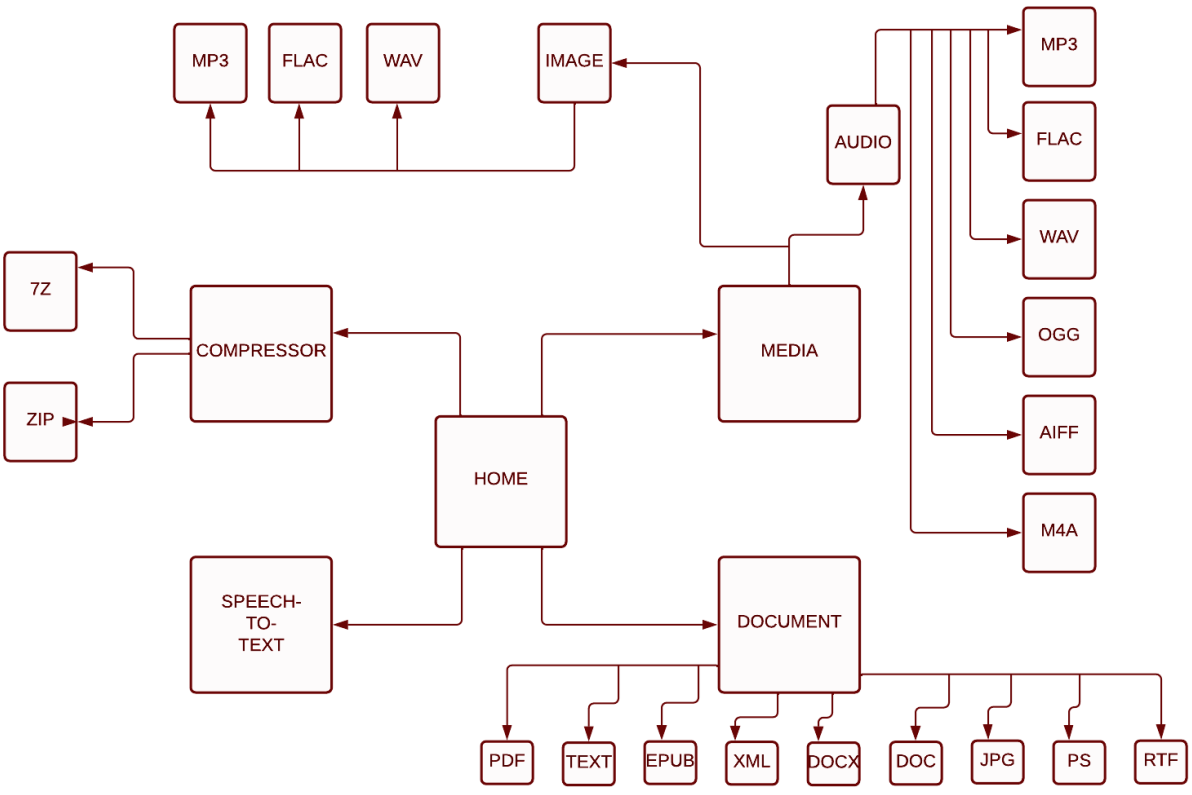
5.4.3. Reliability

The software is 100% reliable to users with easy user interface and navigation.

5.5. Business Rules

The software focuses on maintaining the privacy of its users' data. An age where data is increasing and so do the breaches, this software has a wide scope of market in the upcoming days.

Appendix A : Analysis Models



RESULT :

Software Requirements Specification for the toolkit application is successfully completed.

EXP. NO: 3 **DATE:**

FORMAL USE CASE

AIM :

To write a use case specifications for the toolkit application.

PROCEDURE :

1. Create a use case model for the toolkit application.
2. Identify the use cases and actors for the same
3. Draw the table with the following details
    - Use case name,
    - Participating actors,
    - Flow of events,
    - Entry condition,
    - Exit condition,
    - Quality requirement
4. Finally end the use case

USE CASE

Usecase name	Document conversion
Participating actors	Initiated by user, communicates with server
Flow of events	1. The user selects the zone in which the document converter is located. 2. The document converter shows different formats of conversion available. 3. The user selects the desired format and uploads the file. 4. The document converter shows the converted document available for download to the user.
Entry condition	The user uploads an valid file
Exit condition	● The user selected a format to convert

	• The user is able to download the converted file • The user doesn't choose any option
Quality Requirements	The file is at a negligible size. The user would be able to download the file

RESULT :

Use case specification for the toolkit application is successfully completed

EXP. NO: 4 **DATE:**

PROJECT PLANNING WITH PERT DIAGRAM

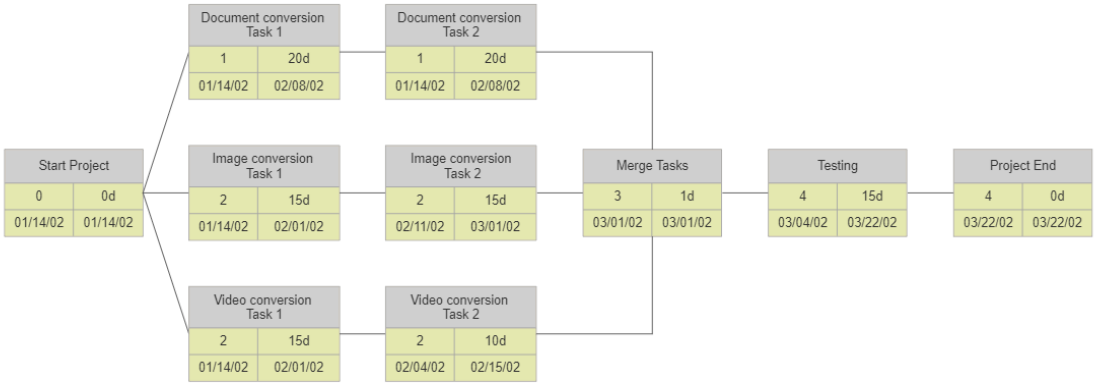
AIM :

To draw a pert diagram for the toolkit application.

PROCEDURE :

1. Identify project tasks.
2. Define task dependencies.
3. Connect project tasks.
4. Estimate project time frame
5. Manage task progress.

DIAGRAM :



RESULT :

Pert chart for the toolkit application is successfully completed

EXP. NO: 5 **DATE:**

USE CASE DIAGRAM

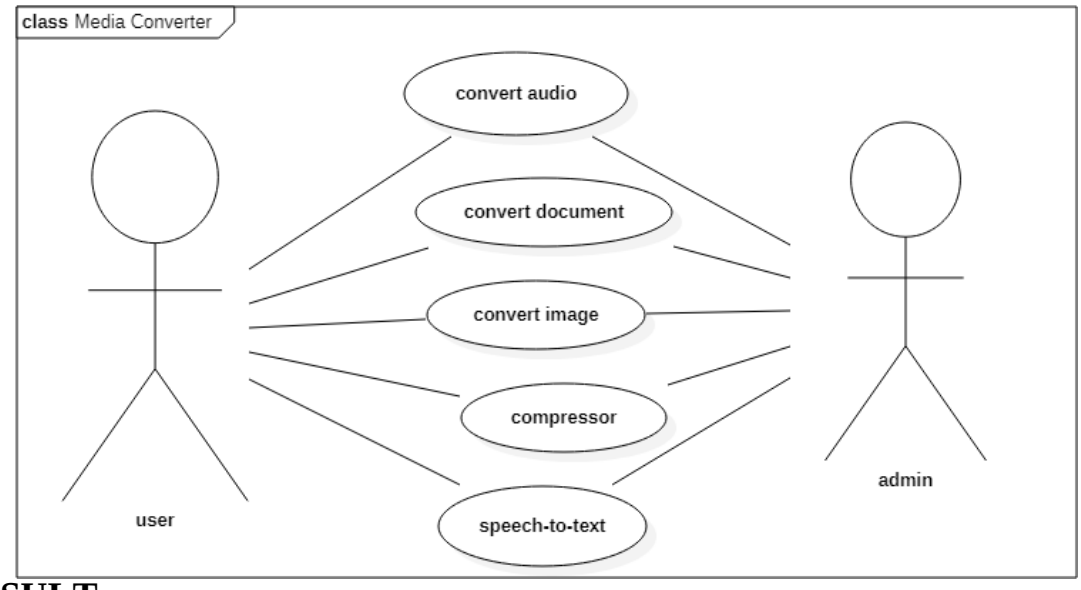
AIM :

To draw a use case diagram for the toolkit application.

PROCEDURE :

1. Identify the actors of the system.
2. For each category of users, identify all roles played by the users relevant to the system.
3. Identify what are the users required for the system to be performed to achieve these goals.
4. Create use cases for every goal.
5. Structure the use cases.
6. Prioritize, review, estimate and validate the users.
7. Add the relationship wherever needed

USE CASE DIAGRAM :



RESULT :

Use case for the toolkit application is successfully completed.

EXP. NO: 6A **DATE:**

INTERACTION DIAGRAM

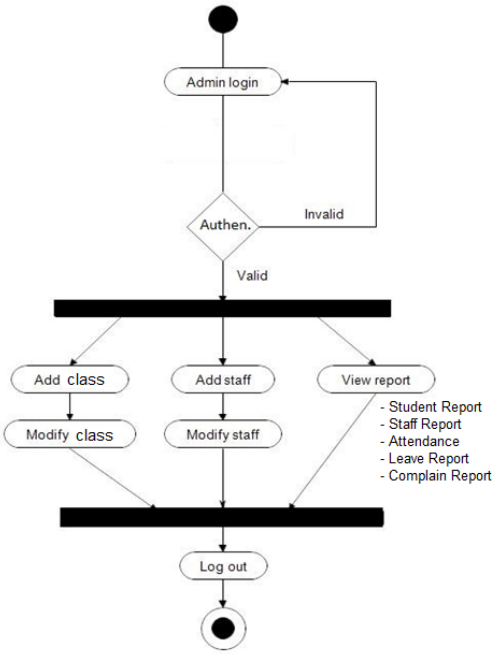
AIM :

To draw a interaction diagram for the toolkit application.

PROCEDURE :

1. Define who will interact with the interaction
2. Draw the first message to a sub-system
3. Draw the message to other sub-system
4. Draw return message to actor
5. Send/respond to anonymous actors

DIAGRAM :



RESULT :

Thus the interaction diagram for the toolkit application is successfully completed.

EXP. NO: 6B **DATE:**

COLLABORATION DIAGRAM

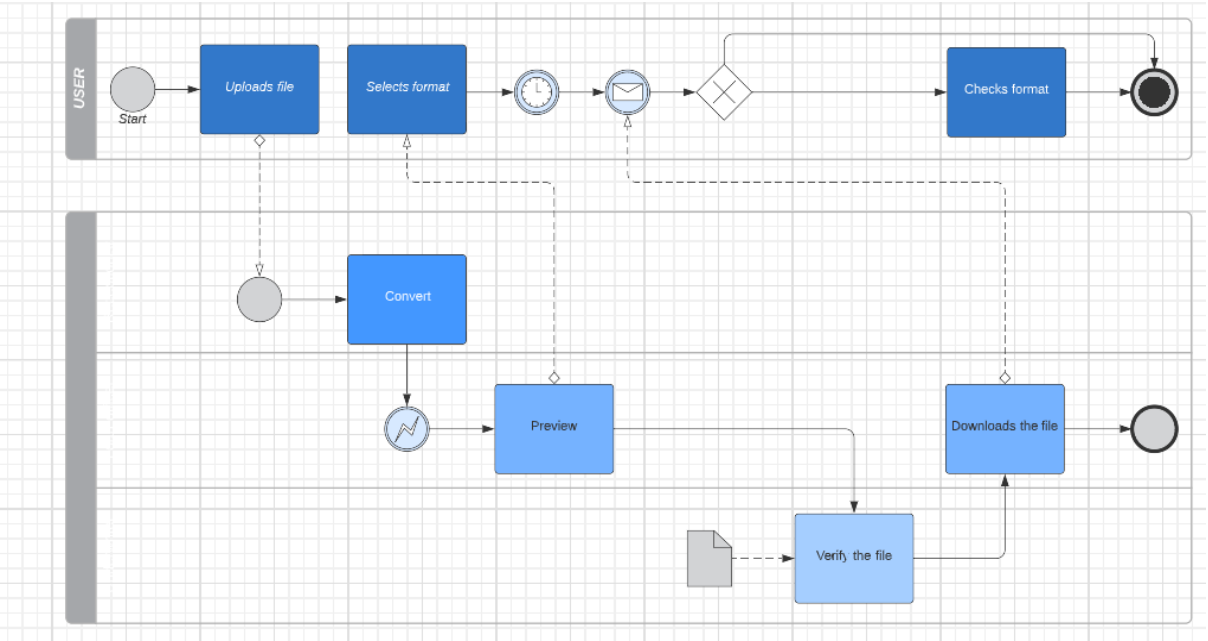
AIM :

To draw a collaboration diagram for the toolkit application.

PROCEDURE :

1. Identify behavior whose realization and implementation is specified
2. Identify the structural elements (class roles, objects, subsystems) necessary to carry out the functionality of the collaboration
3. Decide on the context of interaction: system, subsystem, use case and operation
4. Model structural relationships between those elements to produce a diagram showing the context of the interaction.

DIAGRAM :



RESULT :

Thus the collaboration diagram for the toolkit application is successfully completed.

EXP. NO: 7 **DATE:**

STATE CHART AND ACTIVITY DIAGRAM

AIM :

To draw State Chart diagrams and activity diagrams for the toolkit application.

PROCEDURE :

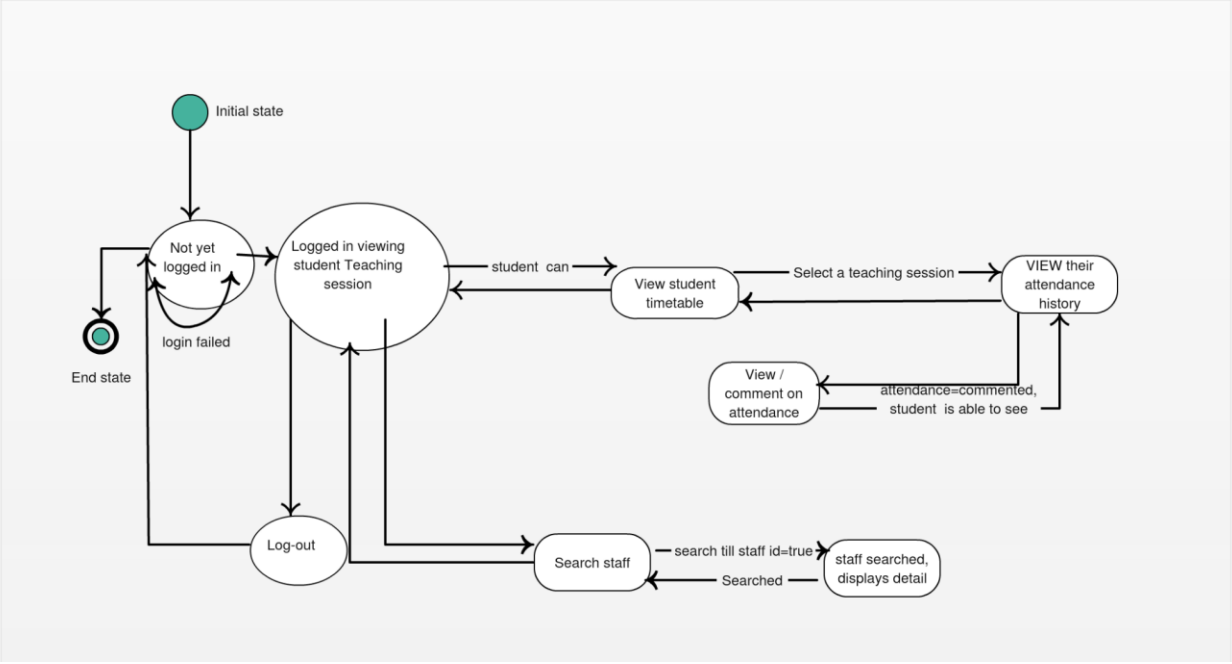
STATE CHART DIAGRAM:

1. Identify the initial state and the final terminating states.
2. Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states).
3. Label the events which trigger these transitions.

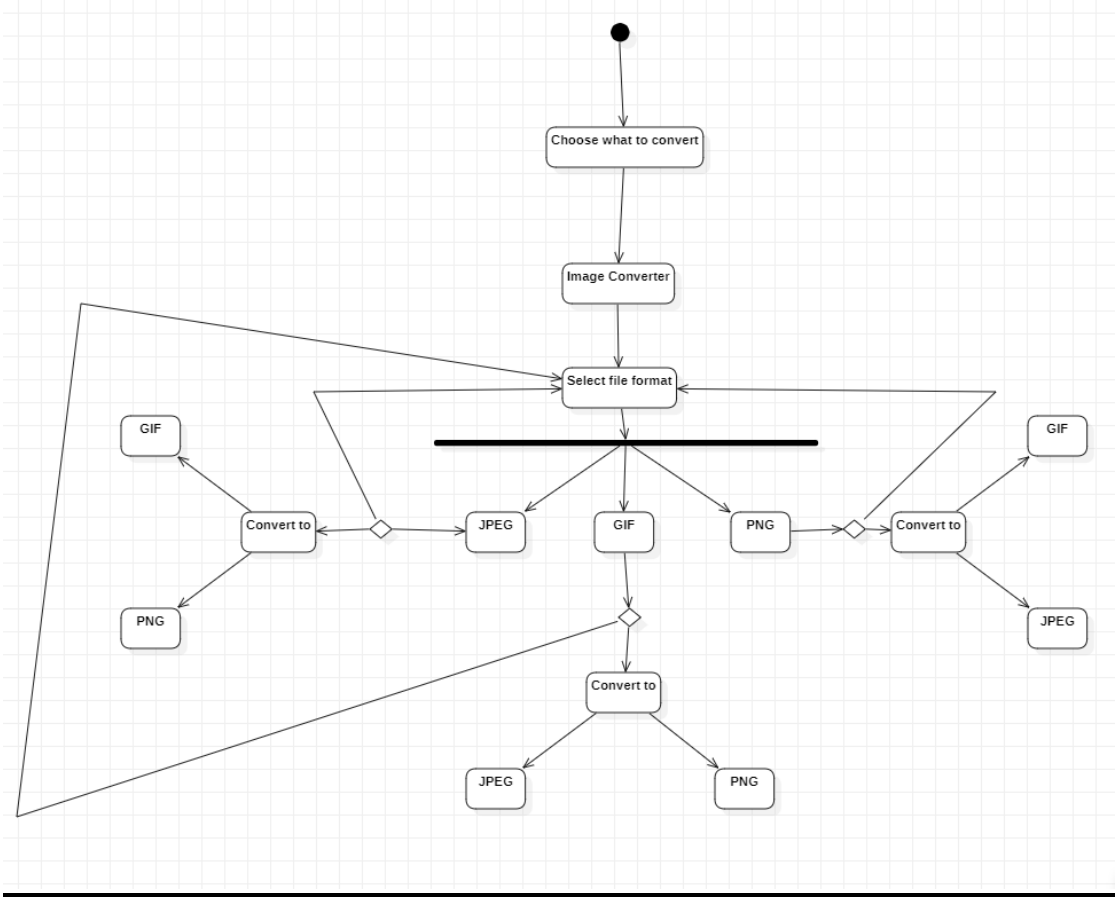
ACTIVITY DIAGRAM :

1. Figure out the action steps from the use case. Here you need to identify the various activities and actions your business process or system is made up of.
2. Identify the actors who are involved. ...
3. Find a flow among the activities. ...
4. Add swimlanes.

STATE CHART DIAGRAM :



ACTIVITY DIAGRAM :



RESULT :

State Chart diagrams and activity diagrams for the toolkit application are successfully completed.

EXP. NO: 8 **DATE:**

CLASS DIAGRAM

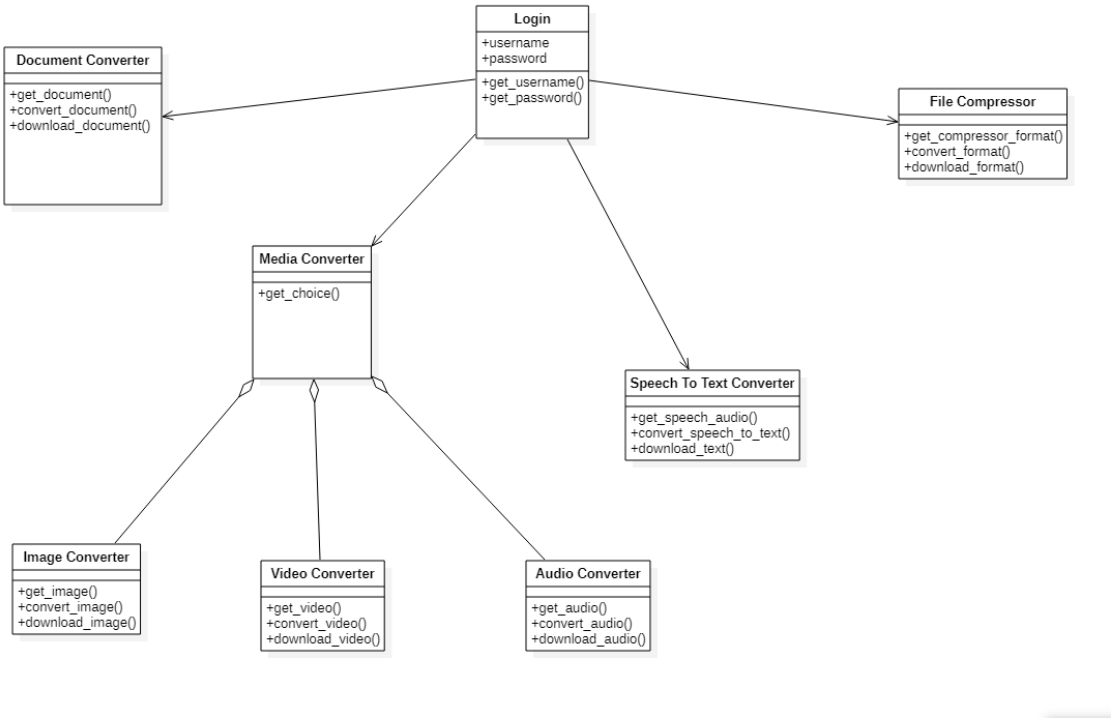
AIM :

To draw class diagrams for the toolkit application.

PROCEDURE :

1. Identify the class names.
2. Identify the primary objects of the system.
3. Distinguish the classes
4. Determine how each of the classes or objects are related to one another.
5. Create the Structure.

DIAGRAM :



RESULT :

Thus the class diagrams for the toolkit application are successfully completed.

EXP. NO: 9 **DATE:**

PACKAGE DIAGRAM

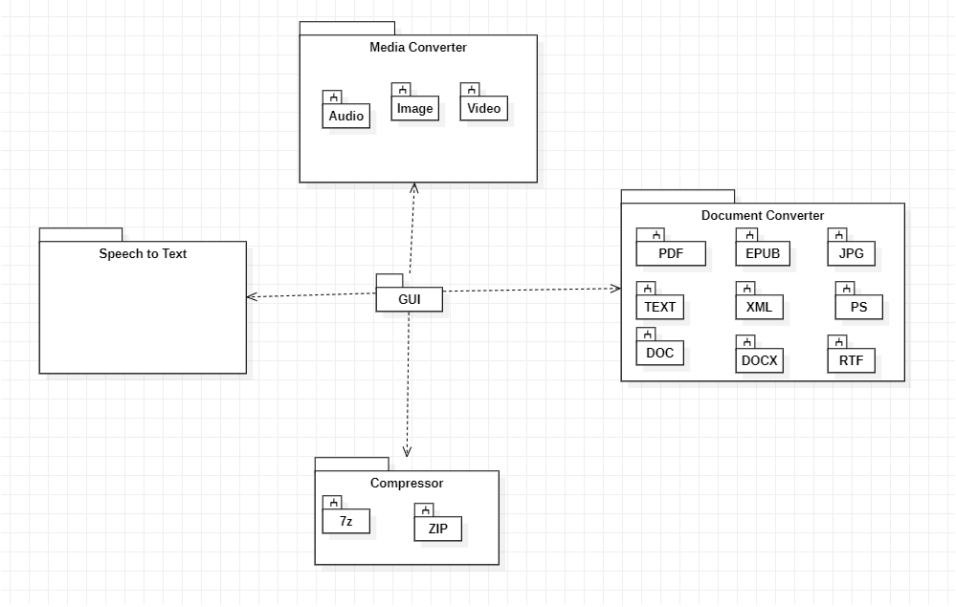
AIM :

To draw package diagrams for the toolkit application.

PROCEDURE :

1. Select Diagram > New from the application toolbar.
2. In the New Diagram window, select Package Diagram.
3. Click Next.
4. Enter the diagram name and description.
5. The Location field enables you to select a model to store the diagram.
6. Click OK.
7. Draw the package diagram with different notations.
8. Specify the links for different packages.

DIAGRAM :



RESULT :

Thus the package diagrams for the toolkit application is successfully completed.

EXP. NO: 10 **DATE:**

COMPONENT AND DEPLOYMENT DIAGRAM

AIM :

To draw Component and deployment diagrams for the toolkit application.

PROCEDURE :

COMPONENT DIAGRAM :

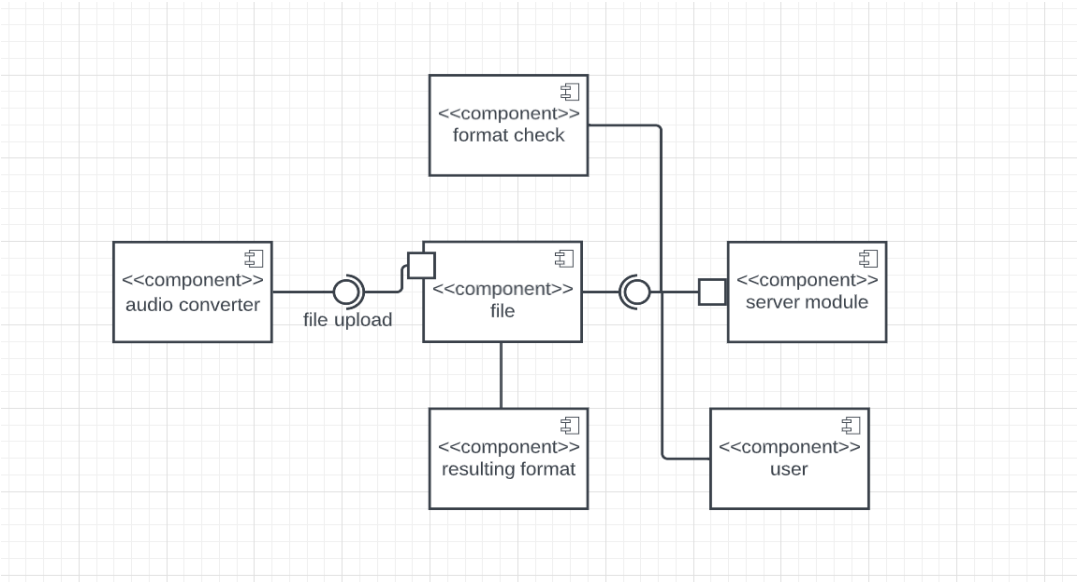
1. Decide on the purpose of the diagram.
2. Add components to the diagram, grouping them within other components if appropriate.
3. Add other elements to the diagram, such as classes, objects and interface.
4. Add the dependencies between the elements of the diagram.

DEPLOYMENT DIAGRAM :

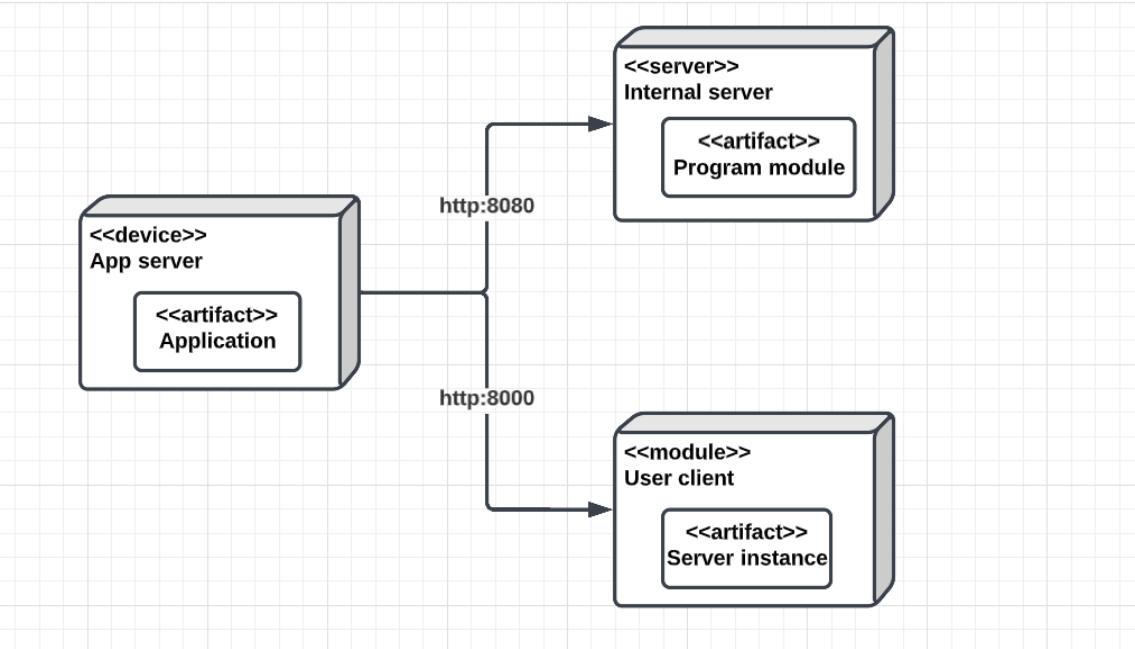
1. Identify the purpose of your deployment diagram.
2. Figure out the relationships between the nodes and devices.
3. Identify what other elements like components, active objects you need to add to complete the diagram.

DIAGRAM :

COMPONENT DIAGRAM :



DEPLOYMENT DIAGRAM :



RESULT :

Thus the Component and deployment diagrams for the toolkit application are successfully completed.

EXP. NO: 11 **DATE:**

MAPPING DESIGN TO CODE

AIM :

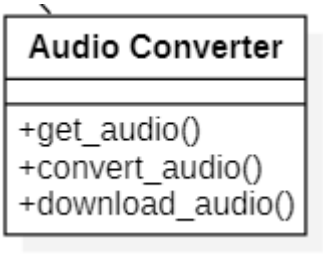
To generate code from class diagram for the toolkit application.

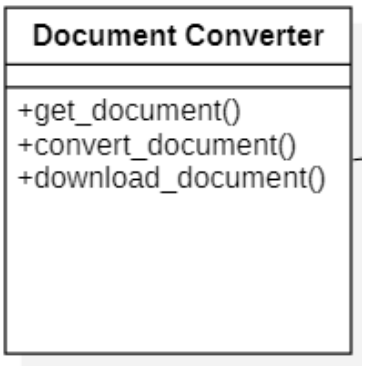
PROCEDURE:

1. Select Tools -> Python ->Generate code
2. In the Instant Generator window, choose the model
3. Fill in the Output Path, which is the directory where you want the code to generate to.
4. Select the classes to generate code.
5. Save the code

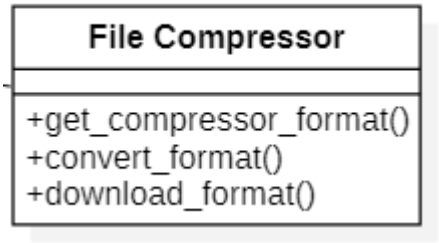
CODE:

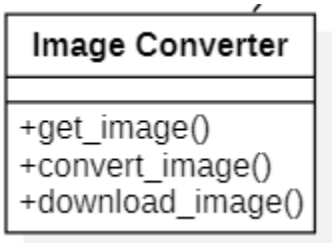
```
class Audio Converter:
    def __init__(self):
        pass

    def get_audio(self, ):
        pass

    def convert_audio(self, ):
        pass

    def download_audio(self, ):
        pass

class Document Converter:
    def __init__(self):
        pass

    def get_document(self, ):
        pass
```



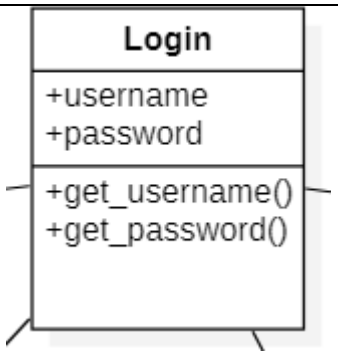


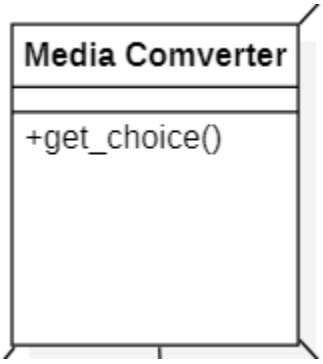
```
    def convert_document(self, ):
        pass

    def download_document(self, ):
        pass

class File Compressor:
    def __init__(self):
        pass

    def get_compressor_format(self, ):
        pass

    def convert_format(self, ):
        pass

    def download_format(self, ):
        pass

class Image Converter:
    def __init__(self):
        pass

    def get_image(self, ):
        pass

    def convert_image(self, ):
        pass

    def download_image(self, ):
        pass

class Login:
    def __init__(self):
```

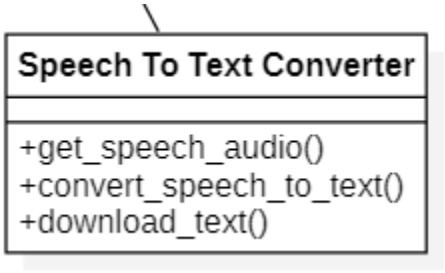


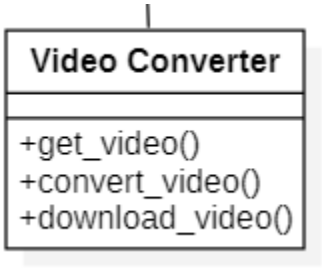


```
        self.username = None
        self.password = None

    def get_username(self, ):
        pass

    def get_password(self, ):
        pass

class Media Comverter:
    def __init__(self):
        pass

    def get_choice(self, ):
        pass
class Speech To Text Converter:
    def __init__(self):
        pass

    def get_speech_audio(self, ):
        pass

    def convert_speech_to_text(self, ):
        pass

    def download_text(self, ):
        pass

class Video Converter:
    def __init__(self):
        pass

    def get_video(self, ):
        pass
```









```
def convert_video(self, ):
    pass

def download_video(self, ):
    pass
```

RESULT :

Thus the code from class diagram for the toolkit application is successfully generated.

IMPLEMENTATION

```
import os
from tkinter import *
from tkinter import filedialog, messagebox
from gtts import gTTS
from playsound import playsound
from docx2pdf import convert
import aspose.words as aw
import fitz
from PIL import Image
import time


def main():
    root = Tk()
    app = Toolkit(root)
    root.mainloop()


class Toolkit:

    def __init__(self, master):
        self.master = master
        self.master.title("Toolkit")
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")
        self.master.resizable(False, False)

        self.title = Label(self.master, text="TOOLKIT", bg="#041C32", font=('arial', 10, 'bold'), fg="white")
        self.title.place(x=300, y=15)
        self.TextToSpeechConverter_button = Button(self.master, text="Text To Speech Converter",
                                        command=self.TextToSpeechConverter_Window, bg='#ECDBBA',
                                        fg='#191919', padx=25, pady=25)
        self.TextToSpeechConverter_button.place(x=80, y=100)

        self.DocumentConverter_button = Button(self.master, text="Document Converter",
                                    command=self.Document_Converter_Window, bg='#ECDBBA',
fg='#191919',
                                    padx=25, pady=25)
        self.DocumentConverter_button.place(x=380, y=100)

        self.Compressor_button = Button(self.master, text="Compressor Converter",
command=self.CompressorWindow,
                                bg='#ECDBBA', fg='#191919', padx=25, pady=25)
```

```
        self.Compressor_button.place(x=80, y=300)

        self.MediaConverter_button = Button(self.master, text="Media Converter",
command=self.Media_Converter_Window,
                                    bg='#ECDBBA', fg='#191919', padx=25, pady=25)
        self.MediaConverter_button.place(x=380, y=300)

        self.master.mainloop()

    def Document_Converter_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = DocumentConverter(self.newWindow)

    def TextToSpeechConverter_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = TextToSpeechConverter(self.newWindow)

    def CompressorWindow(self):
        self.newWindow = Toplevel(self.master)
        self.app = Compressor(self.newWindow)

    def Media_Converter_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = MediaConverter(self.newWindow)


# ================================================== COMPRESSOR
CONVERTER MAIN=========================================

class Compressor:

    def __init__(self, master):
        self.master = master
        self.master.title("Compressor Converter")
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")

        self.master.mainloop()


# ================================================ TXT TO SPEECH CONVERTER
==============================================
class TextToSpeechConverter:

    def __init__(self, master):
```

```
        self.master = master
        self.master.title("Document Converter")
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")
        self.master.resizable(False, False)

        self.title = Label(self.master, text="TEXT TO SPEECH CONVERTER", bg="#041C32",
font=('arial', 10, 'bold'),
                           fg="white")
        self.title.place(x=250, y=15)

        self.BrowseLabel = Label(self.master, text="Enter the text to be spoken", bg='#041C32', fg='white')
        self.BrowseLabel.place(x=80, y=100)

        self.BrowseLocationEntry = Entry(self.master, width=15, bd=4, font=10)
        self.BrowseLocationEntry.place(x=300, y=100)
        self.BrowseLocationEntry.insert(0, "")

        self.SubmitButton = Button(self.master, text="SUBMIT", width="15", command=self.play,
                                   bg='#ECDBBA', fg='#191919')
        self.SubmitButton.place(x=300, y=200)

        self.master.mainloop()

    def play(self):
        self.language = "en"

        self.myobj = gTTS(text=self.BrowseLocationEntry.get(), lang=self.language, slow=False)
        self.myobj.save("convert.wav")
        time.sleep(4)
        os.system("convert.wav")

    def Exit(self):
        self.master.destroy()

    def Reset(self):
        self.BrowseLocationEntry.set("")



class DocumentConverter:

    def __init__(self, master):
        self.master = master
        self.master.title("Document Converter")
```

```
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")
        self.master.resizable(False, False)

        self.title = Label(self.master, text="DOCUMENT CONVERTER", bg="#041C32", font=('arial', 10,
'bold'), fg="white")
        self.title.place(x=250, y=15)

        self.convertDOCXtoPDF = Button(self.master, text="Convert DOCX to PDF",
command=self.DOCXPDF_Window,
                                       bg='#ECDBBA',
                                       fg='#191919', padx=25, pady=25)
        self.convertDOCXtoPDF.place(x=80, y=100)

        self.convertPDFtoDOCX = Button(self.master, text="Convert PDF to DOCX",
command=self.PDFDOCX_Window,
                                       bg='#ECDBBA',
                                       fg='#191919', padx=25, pady=25)
        self.convertPDFtoDOCX.place(x=380, y=100)

        self.convertDOCXtoTXT = Button(self.master, text="Convert DOCX to TXT",
command=self.DOCXTXT_Window,
                                       bg='#ECDBBA',
                                       fg='#191919', padx=25, pady=25)
        self.convertDOCXtoTXT.place(x=80, y=200)

        self.convertTXTtoDOCX = Button(self.master, text="Convert TXT to DOCX",
command=self.TXTDOCX_Window,
                                       bg='#ECDBBA',
                                       fg='#191919', padx=25, pady=25)
        self.convertTXTtoDOCX.place(x=380, y=200)

        self.convertTXTtoPDF = Button(self.master, text="Convert TXT to PDF",
command=self.DOCXTXT_Window, bg='#ECDBBA',
                                      fg='#191919', padx=25, pady=25)
        self.convertTXTtoPDF.place(x=250, y=300)

        self.master.mainloop()

    def DOCXPDF_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = DOCXtoPDF_Converter(self.newWindow)

    def PDFDOCX_Window(self):
        self.newWindow = Toplevel(self.master)
```

```
        self.app = PDFtoDOCX_Converter(self.newWindow)

    def DOCXTXT_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = DOCXtoTXT_Converter(self.newWindow)

    def TXTPDF_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = TXTtoPDF_Converter(self.newWindow)

    def TXTDOCX_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = TXTtoDOCX_Converter(self.newWindow)


# ========================================================== DOCX TO PDF
CONVERTER ======================================
class DOCXtoPDF_Converter:

    def __init__(self, master):
        self.master = master
        self.master.title("Document To PDF Converter")
        self.master.geometry("700x450+0+0")
        self.master.config(bg="#041C32")
        self.master.resizable(False, False)

        self.title = Label(self.master, text="DOCX TO PDF CONVERTER", bg="#041C32", font=('arial',
10, 'bold'), fg="white")
        self.title.place(x=250, y=15)

        self.BrowseFile = Label(self.master, text="Choose a File : ", bg="#041C32", fg="white")
        self.BrowseFile.place(x=100, y=200)

        self.ChooseFileButton = Button(self.master, text="Select", command=self.open_and_convert,
bg='#ECDBBA',
                                       fg='#191919')
        self.ChooseFileButton.place(x=250, y=200)

        self.master.mainloop()

    def open_and_convert(self):
        self.file = filedialog.askopenfilename(filetypes=[("Word Files", "*docx")])
        convert(self.file, r'C:\Users\maarc\OneDrive\Desktop\Converted.pdf')
        messagebox.showinfo("Done", "File successfully converted")
```

```
# ========================================================== PDF TO DOCX
CONVERTER ====================================
class PDFtoDOCX_Converter:

    def __init__(self, master):
        self.master = master
        self.master.title("PDF to DOCX Converter")
        self.master.geometry("700x450+0+0")
        self.master.config(bg="#041C32")
        self.master.resizable(False, False)

        self.title = Label(self.master, text="PDF TO DOCX CONVERTER", bg="#041C32", font=('arial',
10, 'bold'),
                           fg="white")
        self.title.place(x=250, y=15)

        self.BrowseFile = Label(self.master, text="Choose a File : ", bg="#041C32", fg="white")
        self.BrowseFile.place(x=100, y=200)

        self.ChooseFileButton = Button(self.master, text="Select", command=self.open_and_convert,
bg='#ECDBBA',
                                       fg='#191919')
        self.ChooseFileButton.place(x=250, y=200)

        self.master.mainloop()

    def open_and_convert(self):
        self.file = filedialog.askopenfilename(filetypes=[("PDF Files", "*pdf")])
        self.doc = aw.Document(self.file)
        self.doc.save("pdf-to-doc-converted.docx")
        messagebox.showinfo("Done", "File successfully converted")


# ========================================================== DOC TO TXT
CONVERTER ====================================
class DOCXtoTXT_Converter:

    def __init__(self, master):
        self.master = master
        self.master.title("DOCX To TXT Converter")
        self.master.geometry("700x450+0+0")
        self.master.config(bg="#041C32")
        self.master.resizable(False, False)
```

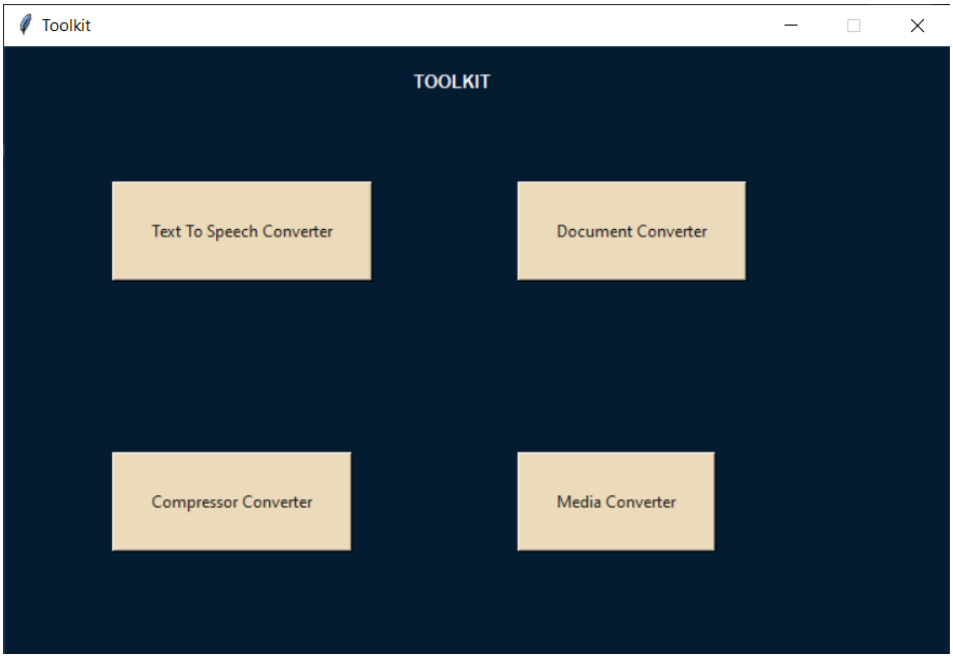
```
        self.title = Label(self.master, text="DOCX TO TXT CONVERTER", bg="#041C32", font=('arial',
10, 'bold'),
                           fg="white")
        self.title.place(x=250, y=15)

        self.BrowseFile = Label(self.master, text="Choose a File : ", bg="#041C32", fg="white")
        self.BrowseFile.place(x=100, y=200)

        self.ChooseFileButton = Button(self.master, text="Select", command=self.open_and_convert,
bg='#ECDBBA',
                                       fg='#191919')
        self.ChooseFileButton.place(x=250, y=200)

        self.master.mainloop()

    def open_and_convert(self):
        self.file = filedialog.askopenfilename(filetypes=[("Word Files", "*docx")])
        self.document = aw.Document(self.file)
        self.document.save("result.txt")
        messagebox.showinfo("Done", "File converted successfully")


# ========================================================== TXT TO DOC
CONVERTER ========================================
class TXTtoDOCX_Converter:

    def __init__(self, master):
        self.master = master
        self.master.title("TXT To DOCX Converter")
        self.master.geometry("700x450+0+0")
        self.master.config(bg="#041C32")
        self.master.resizable(False, False)

        self.title = Label(self.master, text="TXT TO DOCS CONVERTER", bg="#041C32", font=('arial',
10, 'bold'),
                           fg="white")
        self.title.place(x=250, y=15)

        self.BrowseFile = Label(self.master, text="Choose a File : ", bg="#041C32", fg="white")
        self.BrowseFile.place(x=100, y=200)

        self.ChooseFileButton = Button(self.master, text="Select", command=self.open_and_convert,
bg='#ECDBBA',
                                       fg='#191919')
        self.ChooseFileButton.place(x=250, y=200)
```

```
        self.master.mainloop()

    def open_and_convert(self):
        self.file = filedialog.askopenfilename(filetypes=[("Text Files", "*txt")])
        self.document = aw.Document(self.file)
        self.document.save("result.docx")
        messagebox.showinfo("Done", "File converted successfully")


# ================================================ MEDIA CONVERTER MAIN
================================================
class MediaConverter:

    def __init__(self, master):
        self.master = master
        self.master.title("Media Converter")
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")

        self.text = Label(self.master, text="MEDIA CONVERTER", bg="#041C32", fg="white")
        self.text.place(x=250, y=15)

        self.ImageConverter_button = Button(self.master, text="IMAGE CONVERTER",
command=self.Image_Converter_Window,
                                    bg='#ECDBBA', fg='#191919', padx=25, pady=25)
        self.ImageConverter_button.place(x=80, y=100)

        self.master.mainloop()

    def Image_Converter_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = ImageConverter(self.newWindow)


# ================================================== IMAGE CONVERTER
MAIN ========================================
class ImageConverter:

    def __init__(self, master):
        self.master = master
        self.master.title("Image Converter")
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")
```

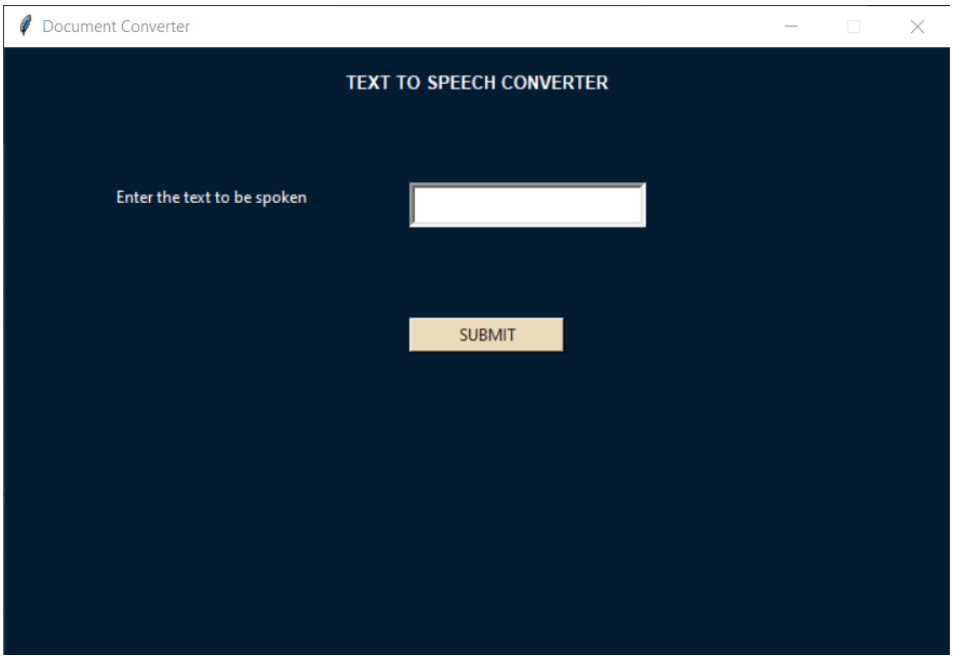
```
        self.text = Label(self.master, text="IMAGE CONVERTER", bg="#041C32", fg="white")
        self.text.place(x=250, y=15)

        self.PNGtoJPG_button = Button(self.master, text="PNG to JPG Converter",
command=self.PNGtoJPG_Window,
                                      bg='#ECDBBA', fg='#191919', padx=25, pady=25)
        self.PNGtoJPG_button.place(x=80, y=100)

        self.JPGtoPNG_button = Button(self.master, text="JPG to PNG Converter",
command=self.JPGtoPNG_Window,
                                      bg='#ECDBBA', fg='#191919', padx=25, pady=25)
        self.JPGtoPNG_button.place(x=300, y=100)

        self.master.mainloop()

    def PNGtoJPG_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = PNGtoJPG_Converter(self.newWindow)

    def JPGtoPNG_Window(self):
        self.newWindow = Toplevel(self.master)
        self.app = JPGtoPNG_Converter(self.newWindow)

# ========================================================= PNG TO JPG
CONVERTER ======================================
class PNGtoJPG_Converter:

    def __init__(self, master):
        self.master = master
        self.master.title("Image Converter")
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")
        self.master.resizable(False, False)

        self.text = Label(self.master, text="PNG TO JPG CONVERTER", bg="#041C32", fg="white")
        self.text.place(x=250, y=15)

        self.convertToJPGButton = Button(self.master, text="Convert PNG to JPG",
command=self.open_and_convert,
                                         bg='#ECDBBA', fg='#191919', padx=25, pady=25)
        self.convertToJPGButton.place(x=80, y=100)

        self.master.mainloop()

    def open_and_convert(self):
```

```
        self.import_file_path = filedialog.askopenfilename(filetypes=[("PNG Files", "*png")])
        self.imageJPG = Image.open(self.import_file_path).convert("RGB")
        self.export_file_path = filedialog.asksaveasfilename(defaultextension=".jpg")
        self.imageJPG.save(self.export_file_path)

        messagebox.showinfo("Done", "File converted successfully")

# ========================================================= JPG TO PNG
CONVERTER ========================================
class JPGtoPNG_Converter:

    def __init__(self, master):
        self.master = master
        self.master.title("JPG to PNG Converter")
        self.master.config(bg="#041C32")
        self.master.geometry("700x450+0+0")
        self.master.resizable(False, False)

        self.text = Label(self.master, text="JPG TO PNG CONVERTER", bg="#041C32", fg="white")
        self.text.place(x=250, y=15)

        self.convertToJPGButton = Button(self.master, text="Convert JPG to PNG",
command=self.open_and_convert,
                                         bg='#ECDBBA', fg='#191919', padx=25, pady=25)
        self.convertToJPGButton.place(x=80, y=100)

        self.master.mainloop()
    def open_and_convert(self):
        self.import_file_path = filedialog.askopenfilename(filetypes=[("JPG Files", "*jpg")])
        self.imageJPG = Image.open(self.import_file_path)
        self.export_file_path = filedialog.asksaveasfilename(defaultextension=".png")
        self.imageJPG.save(self.export_file_path)
        messagebox.showinfo("Done", "File converted successfully")
if __name__ == "__main__":
    main()
```

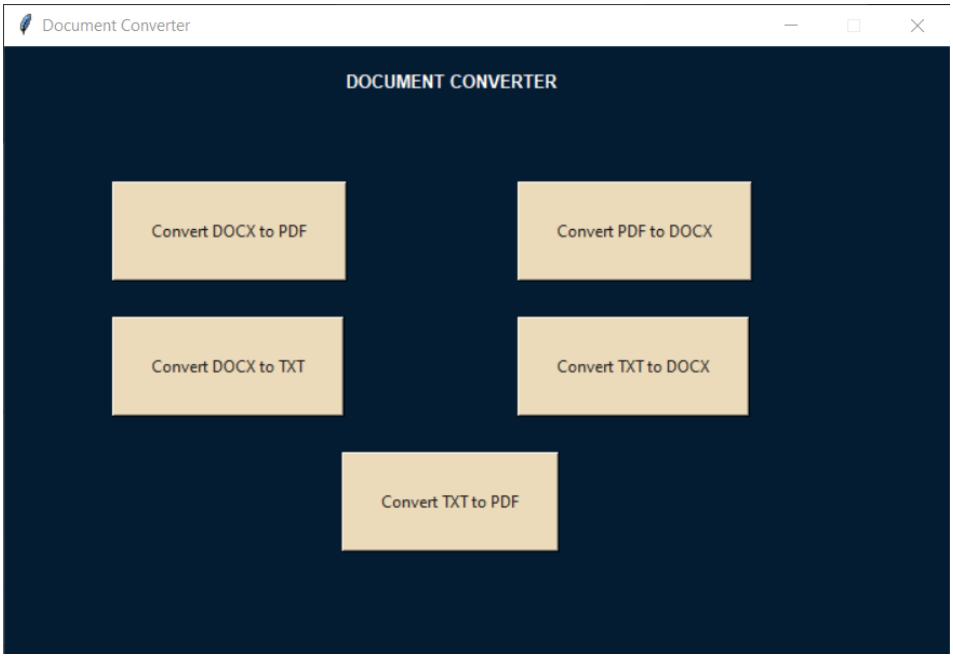
SCREENSHOTS



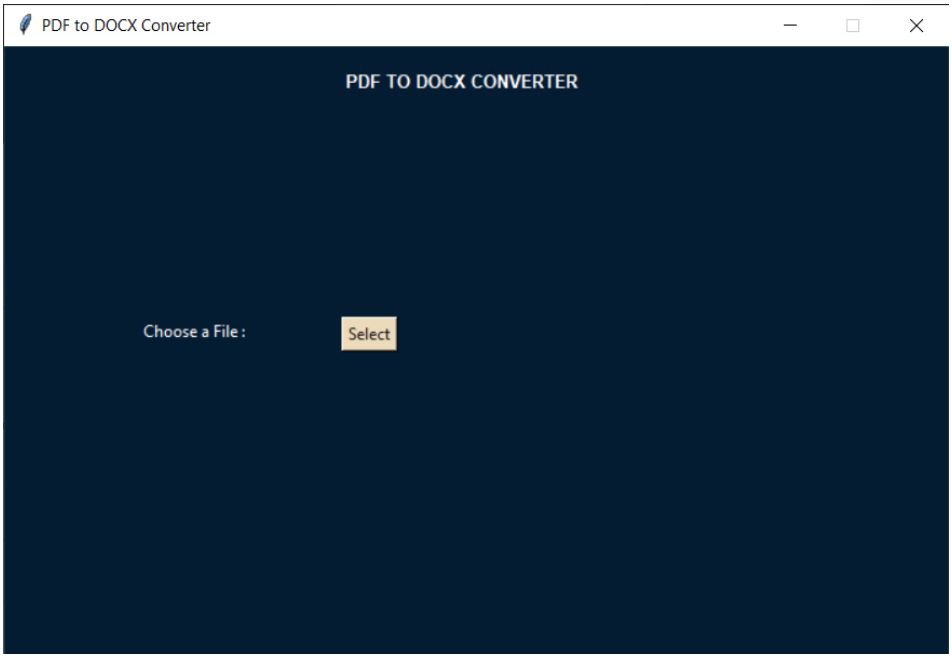
1. Main window



2. Window of text-to-speech-converter

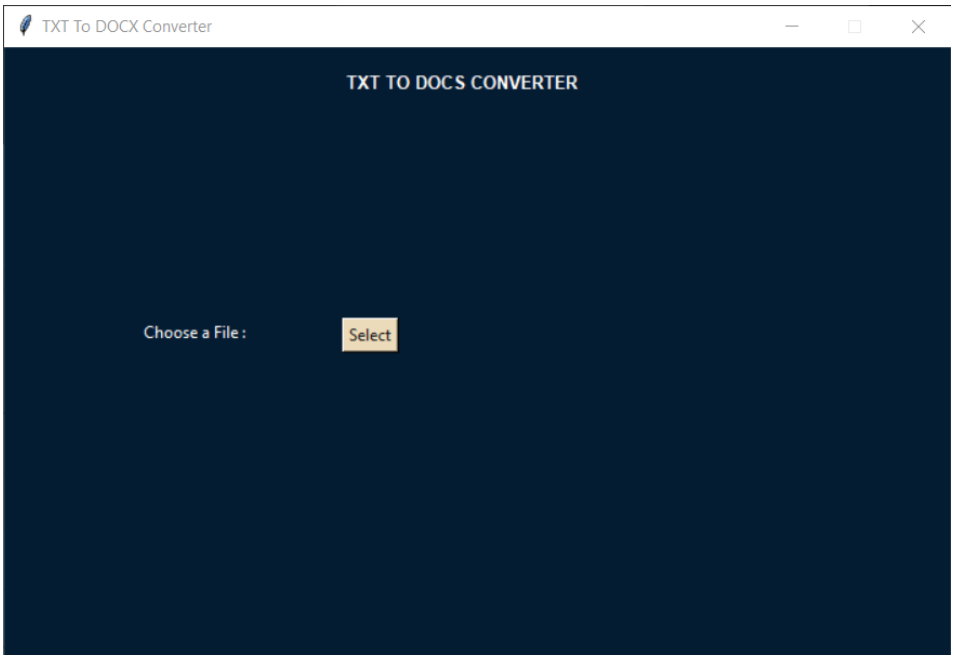


3. Window of document converter



4. Window to choose file to convert it from PDF to DOCX

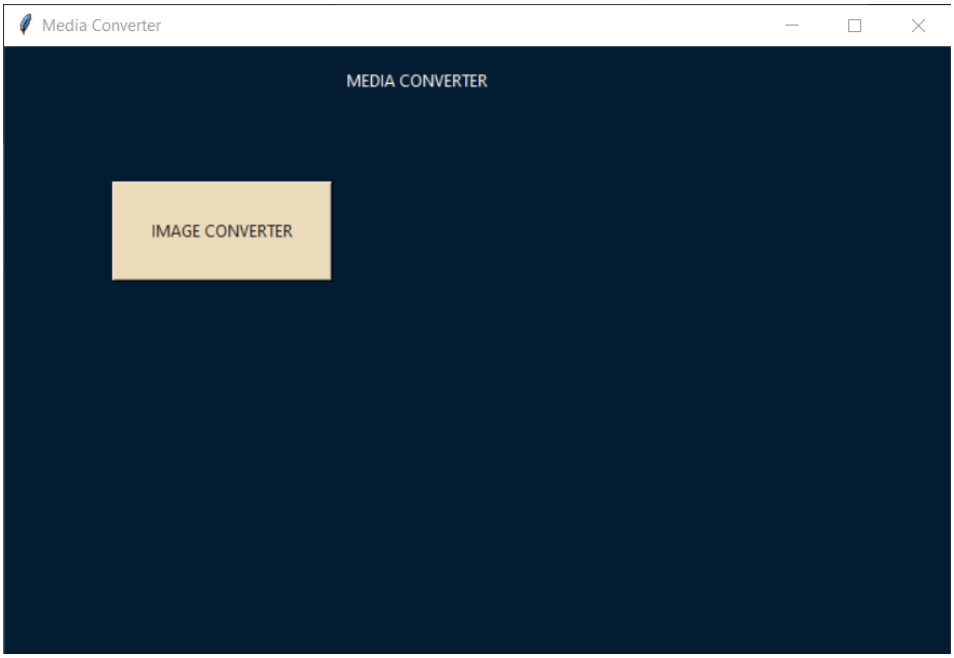
5. Window to choose file to convert it from DOCX to TXT



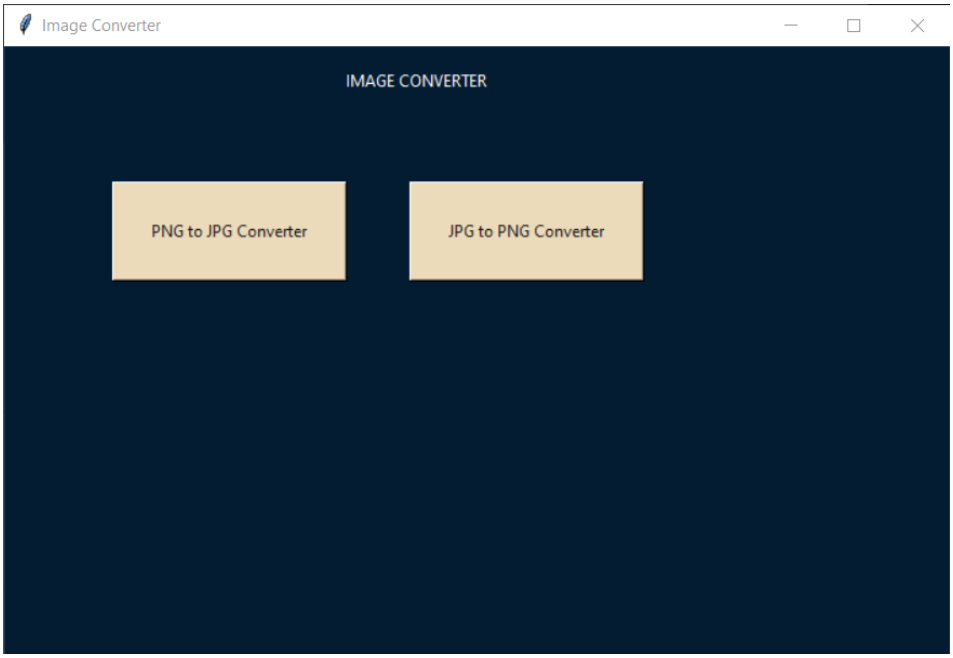
6. Window to choose file to convert it from PDF to DOCX

7. Window to choose file to convert it from DOCX to TXT

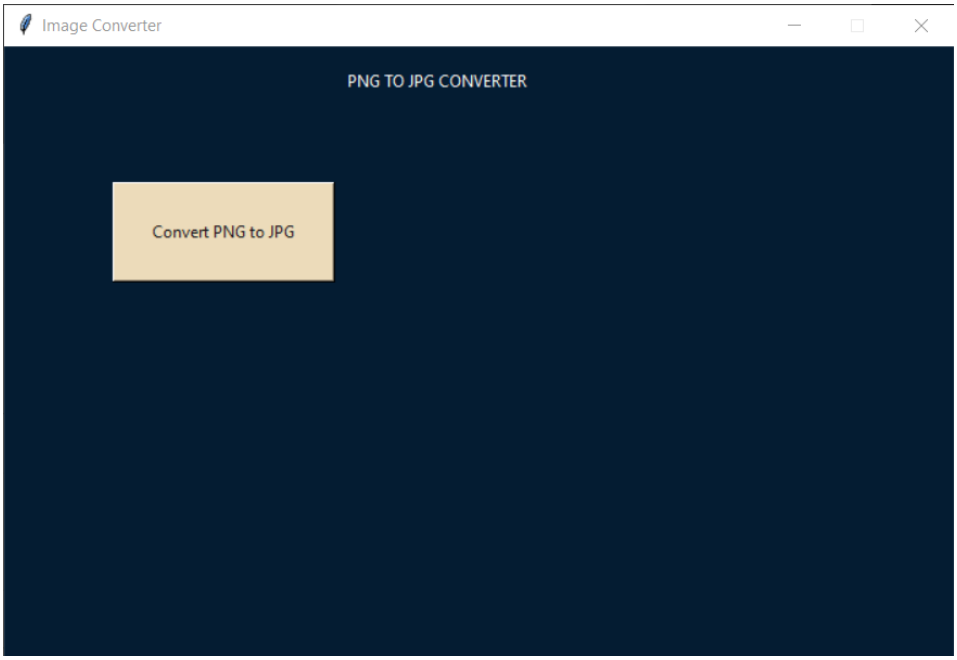
8. Window to choose file to convert it from DOCX to PDF



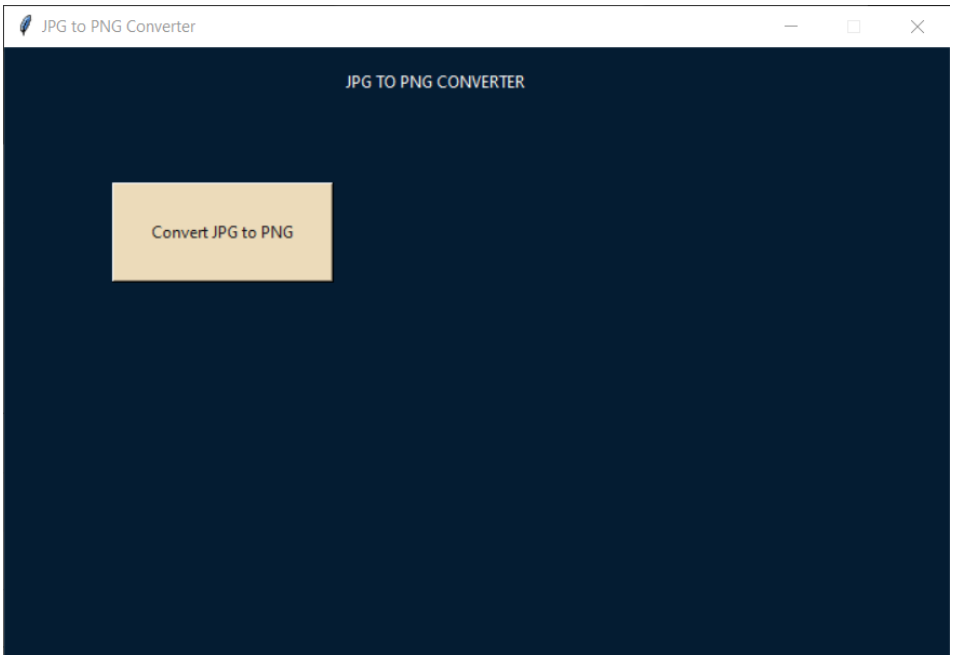
9. Window of media converter



10. Window of Image converter



11. Window to convert PNG to JPG



12. Window to convert JPG to PNG