# Background:

**CHAPTER-1 INTRODUCTION**

In recent years, Artificial Intelligence (AI) has brought significant changes to the field of software engineering, opening up new opportunities for businesses and individuals alike. One of the most notable developments in this area is the emergence of AI-powered desktop assistants, which mimic human interaction and provide valuable help to users.

An AI desktop assistant is essentially a sophisticated computer program that uses the capabilities of natural language processing (NLP) and machine learning algorithms to understand what users say or type and give them relevant responses. These AI desktop assistants can be smoothly integrated into various applications like websites, messaging apps, and mobile applications. They come in handy for tasks such as answering customer queries or making online shopping easier.

The creation of an AI desktop assistant involves several steps in software development, starting with gathering requirements and progressing through design, coding, thorough testing, and finally deployment. At each stage of this complex process, software engineers collaborate closely with AI experts and data scientists to ensure that the AI desktop assistant accurately understands what users want and responds in a way that truly meets their needs.

Successful AI desktop assistants offer numerous advantages to businesses, including improved interaction with customers, increased efficiency, and cost savings. They also provide a more personalized user experience, which can help build customer loyalty and boost sales. Consequently, the development of an AI desktop assistant is a highly promising project in the field of software engineering, and its importance is likely to grow as technology continues to advance.

In an age marked by a constant pursuit of automation and smart systems, the AI desktop assistant emerges as a crucial tool for enhancing productivity and user satisfaction across various areas. Its potential to simplify daily tasks and interactions makes it an essential asset for businesses and individuals navigating the ever-changing digital world. As we delve deeper into this documentation, we will explore the details of creating and implementing an AI desktop assistant system, shedding light on the combination of cutting-edge technology and software engineering expertise that drives its success.

* 1. **About PYTHON & OpenAI API:**

**PYTHON:**

* + - Python is easy to learn, powerful programming language, created by Guido van Rossum and first released in 1991.
    - It has efficient high-level data structures and a simple but effective approach to object-oriented programming.
    - Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.
    - It reduces the cost of program maintenance.
    - Python supports modules and packages, which encourages program modularity and code reuse.

**OpenAI API:**

* + - APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols. For example, the weather app on your phone “talks” to th system via APIs and shows you daily weather updates on your phone.
    - The OpenAI API allowed developers to integrate GPT-3, a state-of-the-art language model developed by OpenAI, into their own applications, products, or services.
    - GPT-3 stands for "Generative Pre-trained Transformer 3," and it is a deep learning model capable of generating human-like text based on the input provided to it.

# Objective:

The objective of this project is to develop a Desktop Assistant that will have the following functions:

* + - To create a reliable and efficient virtual assistant that can improve communication with users.
    - To develop an assistant that can handle a wide range of tasks, including answering questions, recommendation, etc.
    - To design user friendly interface that is easy to use and provides a positive user experience.
    - To build an Assistant that is scalable and can handle large volumes of user interactions.
    - To Implement machine learning algorithms to train the Assistant to understand user intent and provide relevant responses.
    - To Develop a natural language processing model that can interpret user queries and extract information.
    - To Develop a user interface that is intuitive and easy to use.
    - To Test and refine the performance of desktop assistant through user feedback and Analytics and make improvements.

# Advantages:

* + - Increase productivity.
    - Entertainment and Information.
    - Reduce workload.
    - Reliable and secure.
    - Less Time Consuming.
    - Personalized assistance.

# Scope:

* Information Retrieval: Quickly fetching data, answering questions, and providing information on various topics.
* Personal Organization: Assisting users with scheduling, reminders, and task management.
* Entertainment: Offering entertainment options such as sharing news updates, or suggesting movies and books.
* Desktop Applications Integration: Interacting with users within their desktop environment for convenience.
* Capable of Executing commands and tasks as instructed by the user, automating processes as needed.
* Maintaining a chatlog to store and retrieve personalized user data, enhancing the quality of user   
  interactions and recommendations.

# Existing System:

The current landscape in the realm of desktop assistants presents certain limitations and challenges that prompt the development of our AI Desktop Assistant System. In the existing system, users rely on rudimentary virtual assistants with limited capabilities, primarily offering basic search functions and predefined commands. These conventional systems often struggle with complex user interactions, lack personalization, and do not harness the power of advanced AI technologies for efficient task automation and natural language understanding.

# Proposed System:

Our AI Desktop Assistant System represents a significant leap forward in the domain of desktop assistance. It aims to revolutionize the user experience by offering an intelligent, adaptable, and highly personalized assistant capable of understanding natural language, automating tasks efficiently, and seamlessly integrating with various applications and platforms. Our proposed system leverages cutting-edge AI technologies to enhance user productivity, streamline workflows, and provide a level of assistance that transcends the limitations of the existing system.

# CHAPTER-2

**System Analysis and Requirements**

# System Analysis:

## General:

It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.

## Requirements Specification:

* + 1. **Software requirement:**

### Software Requirements of AI Desktop Assistant System:

A set of programs associated with the operation of a computer is called software. Software is the part of the computer system, which enables the user to interact with several physical hardware devices. The minimum software requirement specifications for developing this project are as follows:

* Operating system: Windows 7 or higher
* Programming Language: Python
* Text Editor: VS code
* Web Browser: Any-Browser

## Hardware requirement:

### Hardware Requirements of AI Desktop Assistant System:

The collection of internal electronic circuits and external physical devices used in building a computer is called the Hardware. The minimum hardware requirement specifications for developing this project are as follows:

* Processor - Intel/ AMD 3rd Gen CPU or Higher
* Ram - 4.00 GB or Higher
* Hard Disk - 256 GB or Higher
* GPU – NVIDIA GeForce MX330 or Higher
* Microphone

## Problem Definition:

Build a virtual voice assistant that will enable users to interact with emerging technologies, manage their devices, and utilize technology for learning.

It serves as a voice assistant for visually impaired people and is a cutting-edge system. By utilizing distinct custom layouts and speech to text, this solution improves system quality while enabling visually challenged users to access the desktop's most crucial functionalities. The user's speech will be the basis for all actions taken by the system. The system assists the user based on voice note, meaning that it follows instructions provided by the user. Because the user cannot see the action going place on the desktop, the system speaks out if the user needs to receive a response.

The blind applicant will also sense independence. Because the system is a machine, it will execute without error. Although many seniors are unable to utilize desktop computers, they can still benefit from this. These assistive technologies will enable users who are blind or visually handicapped to interact with computers.

# Functional Requirements:

## Modules Description:

The AI desktop assistant should have the following functionalities:

* Voice recognition
* Query Answers
* Task execution

## Voice Recognition:

The system should be able to accurately recognize and interpret voice commands from the

user.

## Query Answering:

The system should be able to answer any type of query related to any topic, except for

illegal things.

## Task Execution:

The system should be able to execute tasks such as setting reminders, playing music, sending

emails, and searching the web.

## User Analysis:

User analysis is the process of identifying the different categories of users to the application, their hierarchies and their functionalities.

# CHAPTER-3 SURVEY OF TECHNOLOGIES

1. **SURVEY OF TECHNOLOGIES:**

## BENEFITS:

* + - Increase productivity.
    - Personalized User Experience
    - Reduce workload
    - Reliable and secure
    - Less Time Consuming
    - Efficient Task Automation
    - It saves time.
    - Enhanced Multitasking

## SOFTWARE REQURIMENTS:

* + - Python
    - HTML, CSS, Javascript
    - Speech Recognition
    - Text to Speech (TTS)
    - Machine Learning Framework
    - Web API

## TECHNOLOGY USED:

* + 1. **Python:**

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It is used for:

Web development (server-side), Software Development, Mathematics,

System Scripting.

## What can Python do?

Python can be used on a server to create web applications. Python can be used alongside software to create workflows.

Python can connect to database systems. It can also read and modify files. Python can be used to handle big data and perform complex mathematics.

## Why Python?

Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc). Python has a simple syntax similar to the English language.

Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

**3.3.2 HTML:**

• HTML stands for Hyper Text Mark-up Language.

• HTML describes the structure of Web pages using mark-up.

• HTML elements are the building blocks of HTML pages.

• HTML elements are represented by tags.

• HTML tags label pieces of content such as "heading", "paragraph", "table", and so on.

## Speech Recognition:

* + - * Google Cloud Speech-to-Text API, Amazon Transcribe, and Microsoft Azure Speech Services are a few well-known speech recognition APIs.
      * These APIs offer the ability to convert speech to text and can be incorporated into voice assistant software

## Text to Speech (TTS):

* + - * Popular TTS engines include Google Text-to-Speech, Amazon Polly, and Microsoft Speech Services.
      * These engines can be used to generate human-like speech output from text input.
    1. **WEB API:**
* Web APIs such as OpenWeatherMap, NewsAPI, and OpenAI API can be used to integrate the voice assistant with external services
* These APIs allow the voice assistant to access weather forecasts, news articles, and music streaming.
  + 1. **CSS:**

• CSS is the language we use to style an HTML document.

• CSS describes how HTML elements should be displayed. • This tutorial will teach you CSS from basic to advanced.

• CSS stands for Cascading Style Sheets.

• CSS describes how HTML elements are to be displayed on screen, paper, or in other media.

• CSS saves a lot of work. It can control the layout of multiple web pages all at once.

• External stylesheets are stored in CSS files.

## Feasibility Study:

A feasibility study, as the name suggests, is designed to reveal whether a project/plan is feasible. It is an assessment of the practicality of a proposed project/plan.

A feasibility study is part of the initial design stage of any project/plan. It is conducted in order to objectively uncover the strength and weaknesses of a proposed project or an existing business. It can help to identify and assess the opportunities and threats present in the natural environment, the resources required for the project, and the prospects for success.

## Technical Feasibility:

Technical feasibility is the formal process of assessing whether it is technically possible to manufacture a product or service. Before launching a new offering or taking up a client project, it is essential to plan and prepare for every step of the operation. Technical feasibility helps determine the efficacy of the proposed plan by analyzing the process, including tools, technology, material, labor and logistics. A technical feasibility study helps organizations determine whether they have the technical resources to convert the idea.

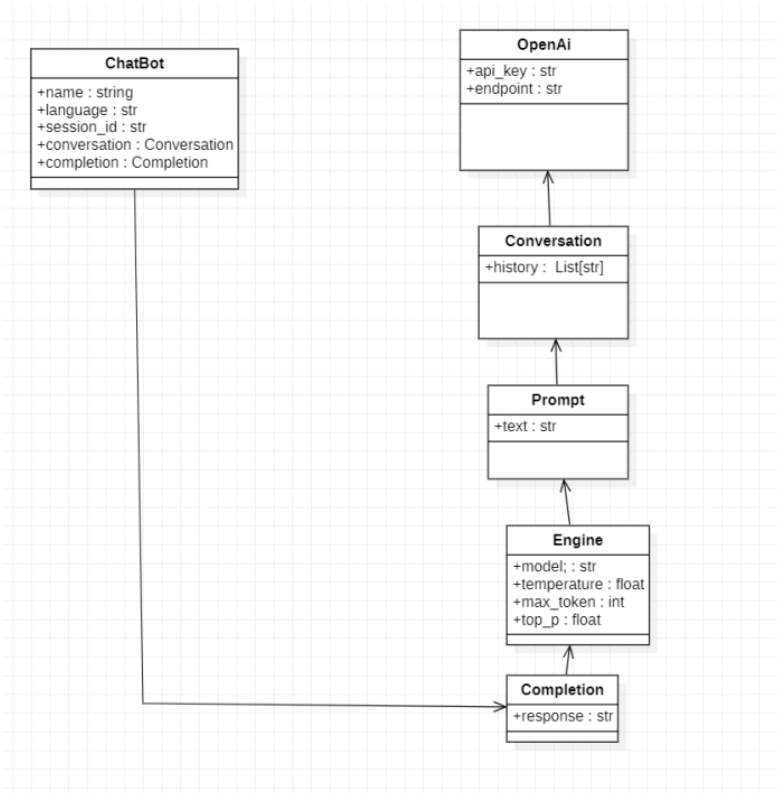
## Limitations:

* Keywords: Certain keywords in the assistant have special functionalities, hence these keywords cannot be used for any other purpose.

1. Visit = used to visit a website (facebook)
2. Launch = used to open an application
3. Search = used to search something on google
4. Open = used to open an application
5. Exit = used to exit the assistant
6. Temperature = used to get the current temperature of specific location
7. Weather = used to get the current weather of specific location
8. Present date = used to get the current date
9. Present time = used to get the current time

**CHAPTER 4 SYSTEM DESIGN**

**Class Diagram**



## Class Diagram: -

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

## Purpose of Class Diagram?

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as –

* + - Analysis and design of the static view of an application.
    - Describe responsibilities of a system.
    - Base for component and deployment diagrams.
    - Forward and reverse engineering

## How to Draw a Class Diagram?

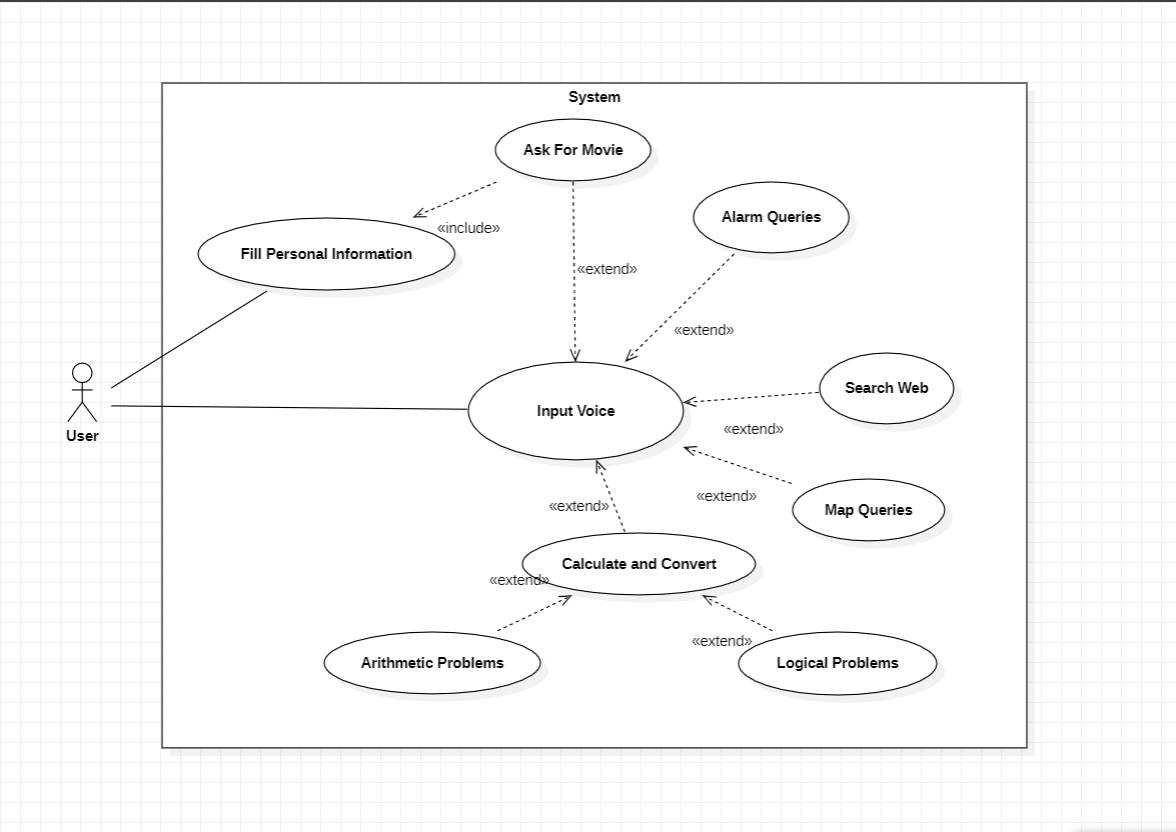
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represents the whole system.

* The following points should be remembered while drawing a class diagram –
  + The name of the class diagram should be meaningful to describe the aspect of the system.
  + Each element and their relationships should be identified in advance.
  + Responsibility (attributes and methods) of each class should be clearly identified
  + For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.

## Where to Use Class Diagrams?

Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system, but they are also used to construct the executable code for forward and reverse engineering of any system.

**Use Case Diagram**



## Use case Diagram: -

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

## Purpose of Use case Diagram?

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

In brief, the purposes of use case diagrams can be said to be as follows −

* + - Used to gather the requirements of a system.
    - Used to get an outside view of a system.
    - Identify the external and internal factors influencing the system.
    - Show the interaction among the requirements are actors.

## How to Draw a Use case Diagram?

A Use Case model can be developed by following the steps below.

1. Identify the Actors (role of users) 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 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.

* Your diagrams will always work with the same three components:
* **Actors.** Actors represent whatever is interacting with your system. An actor could be a person, a business, a group, or something else. Anything can be an actor as long it exists outside of the system and interacts with said system in some way.
* **Systems.** Sometimes known as a ‘scenario,’ your system is made up of a series of actions and interactions made by actors.
* **Goals.** Your goals are the outcome of an actor’s interactions with the system. Your system will lead to multiple outcomes in some cases, while others have one direct outcome.

## Where to Use Class Diagrams?

To understand the dynamics of a system, we need to use different types of diagrams. Use case diagram is one of them and its specific purpose is to gather system requirements and actors.

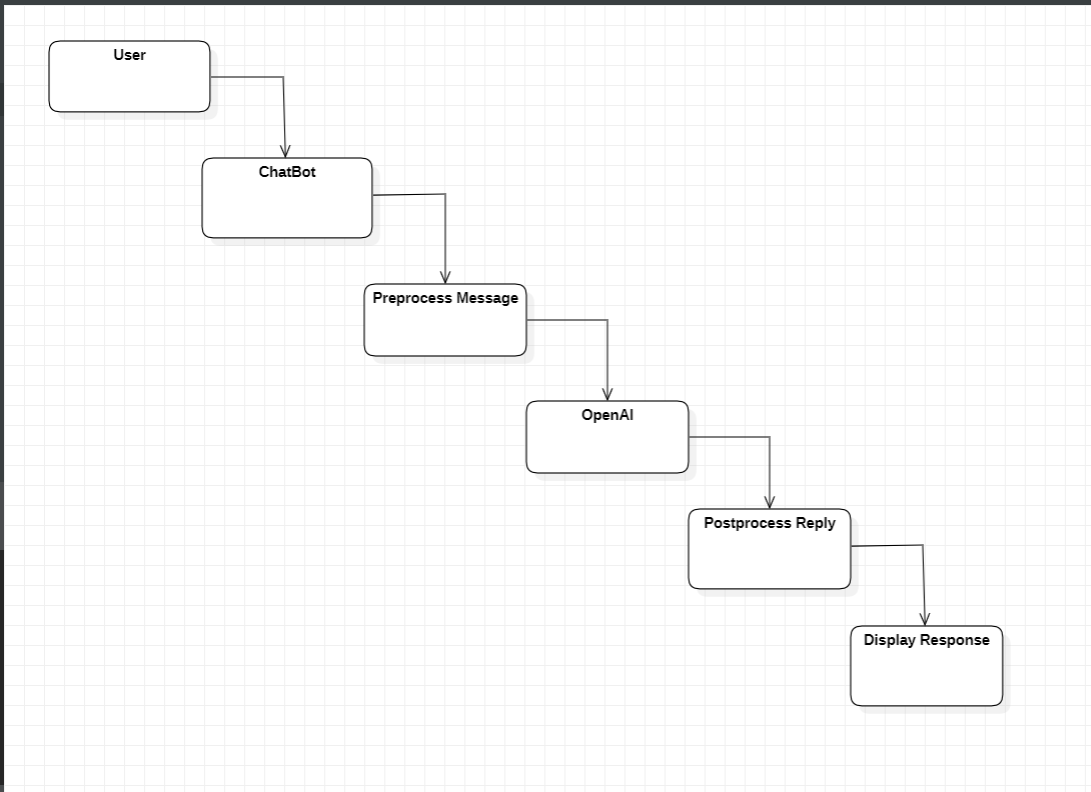
Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box is known.

These diagrams are used at a very high level of design. This high-level design is refined again and again to get a complete and practical picture of the system. A well-structured use case also describes the precondition, post condition, and exceptions. These extra elements are used to make test cases when performing the testing.

Use case diagrams can be used for −

* + Requirement analysis and high-level design.
  + Model the context of a system.
  + Reverse engineering.
  + Forward engineering.

**Activity Diagram**



## Activity Diagram

The Unified Modeling Language includes several subsets of diagrams, including structure diagrams, interaction diagrams, and behaviour diagrams. Activity diagrams, along with use case and state machine diagrams, are considered behaviour diagrams because they describe what must happen in the system being modeled. Stakeholders have many issues to manage, so it's important to communicate with clarity and brevity. Activity diagrams help people on the business and development sides of an organization come together to understand the same process and behavior.

## Purpose of Activity Diagram

In UML, an activity diagram provides a view of the behavior of a system by describing the sequence of actions in a process. Activity diagrams are similar to flowcharts because they show the flow between the actions in an activity; however, activity diagrams can also show parallel or concurrent flows and alternate flows. In activity diagrams, you use activity nodes and activity edges to model the flow of control and data between actions.

## How to Draw Activity Diagram

Activity diagrams can be used to model business requirements, create a high-level view of a system’s functionalities, analyse use cases and for various other purposes. In each of these cases, here’s how to draw an activity diagram from the beginning.

* + - Figure out the action steps from the use case
    - Identify the actors who are involved
    - Find a flow among the activities

## Where to use Activity Diagram

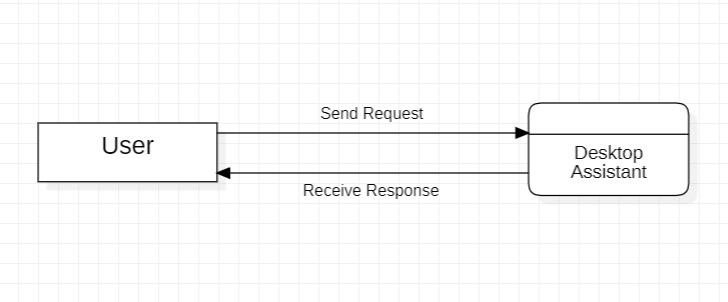
The great strength of activity diagrams lies in the fact that they support and encourage parallel behavior.

This makes them a great tool for workflow modeling and, in principle, for multithreaded programming. Their great disadvantage is that they do not make the links among actions and objects very clear.

I like to use activity diagrams in the following situations:

* + - Analysing a use case.
    - Understanding workflow.
    - Describing a complicated sequential algorithm.
    - Dealing with multithreaded applications.
    - Trying to see how objects collaborate.

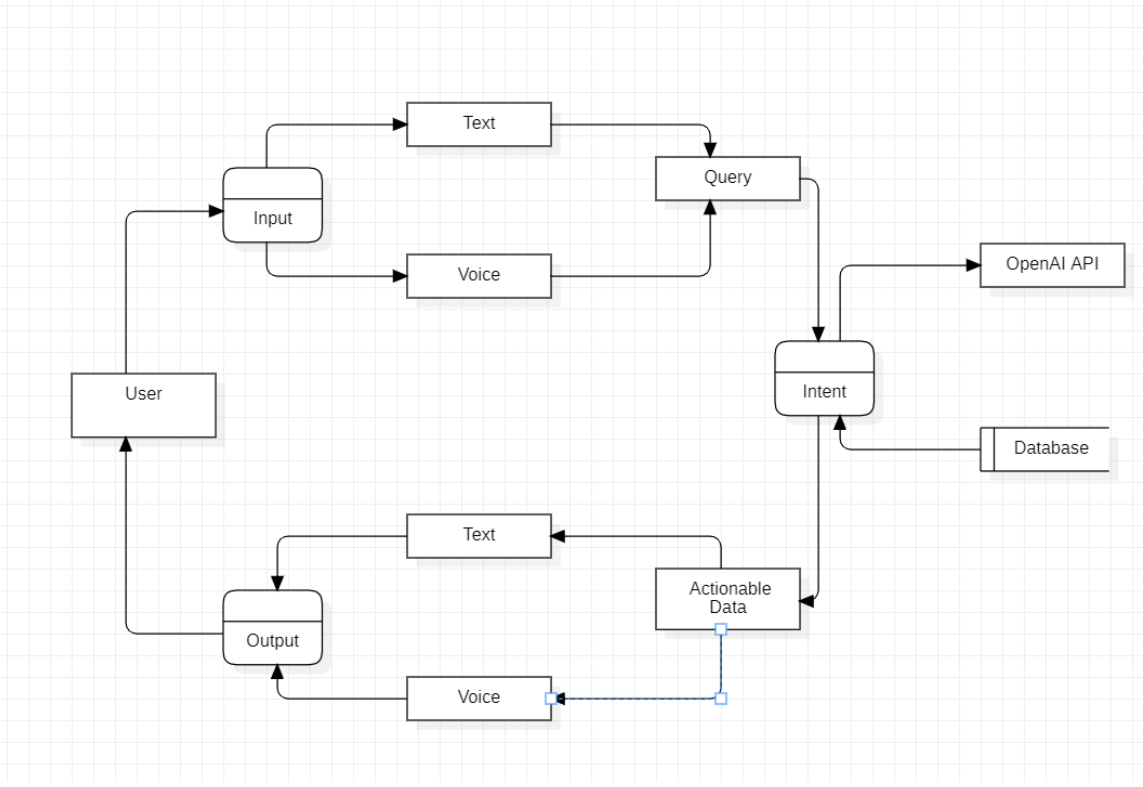
**DFD Zero Level**



* 1. **DFD Zero Level**

DFD Level 0 is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modelled. It’s designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

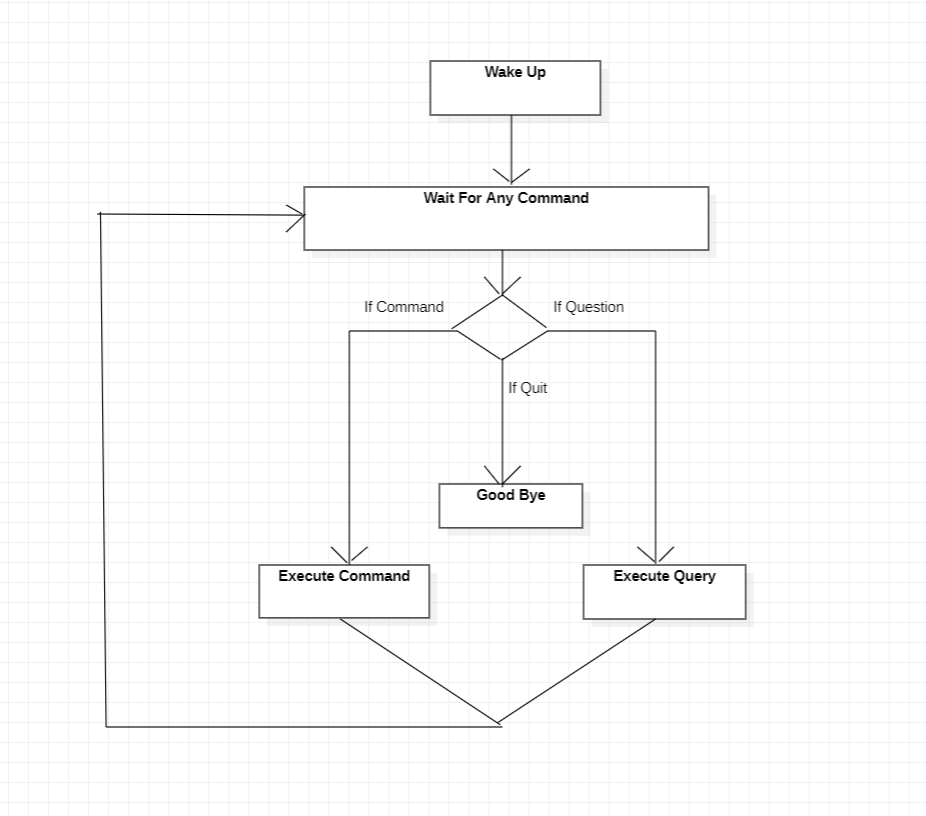
**DFD First Level**



## DFD First Level

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.

**ER Diagram**



## ER Diagram: -

ERD stands for entity relationship diagram, which is a type of diagram that can help you create a clear visualization of your database design.

## Purpose of ER Diagram

An entity-relationship (ER) diagram is a conceptual model that captures an organization's business rules (and any common-sense rules) as concisely as possible.

## How to Draw an ER Diagram

There are a few basic steps to take to draw an ER diagram anywhere: in Gliffy, with PowerPoint, a whiteboard, or even on the back of a napkin. Here’s the basic order to follow.

1. Determine the Entities in Your ERD
2. Add Attributes to Each Entity
3. Define the Relationships Between Entities
4. Add Cardinality to Every Relationship in your ER Diagram
5. Finish and Save Your ERD

This is just a high-level ER model, but it provides enough detail that you should now have a teammate or partner check your work. One of the best ways to do this is to simply have them try to read your diagram out loud. If they end up telling a different story than you intended, you need to do some tweaking.

## Where to use ER Diagram

Database design: ER diagrams are used to model and design relational databases, in terms of logic and business rules (in a logical data model) and in terms of the specific technology to be implemented (in a physical data model.) In software engineering, an ER diagram is often an initial step in determining requirements for an information systems project.

## Entity

Entity keys: Refers to an attribute that uniquely defines an entity in an entity set. Entity keys can be super, candidate or primary. Super key: A set of attributes (one or more) that together define an entity in an entity set. Candidate key: A minimal super key, meaning it has the least possible number of attributes to still be a super key. An entity set may have more than one candidate key. Primary key: A candidate key chosen by the database designer to uniquely identify the entity set. Foreign key: Identifies the relationship between entities.

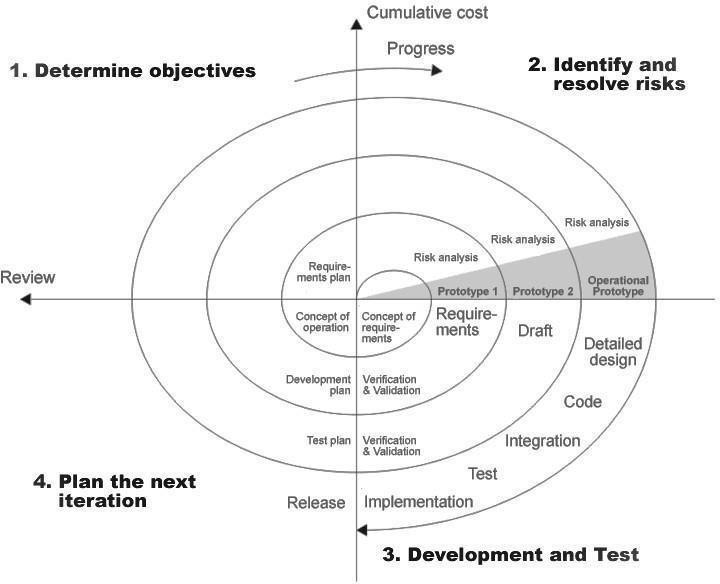
## Relationship

How entities act upon each other or are associated with each other. Think of relationships as verbs. For example, the named student might register for a course. The two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the two entities in that way.

Relationships are typically shown as diamonds or labels directly on the connecting lines. Recursive relationship: The same entity participates more than once in the relationship.

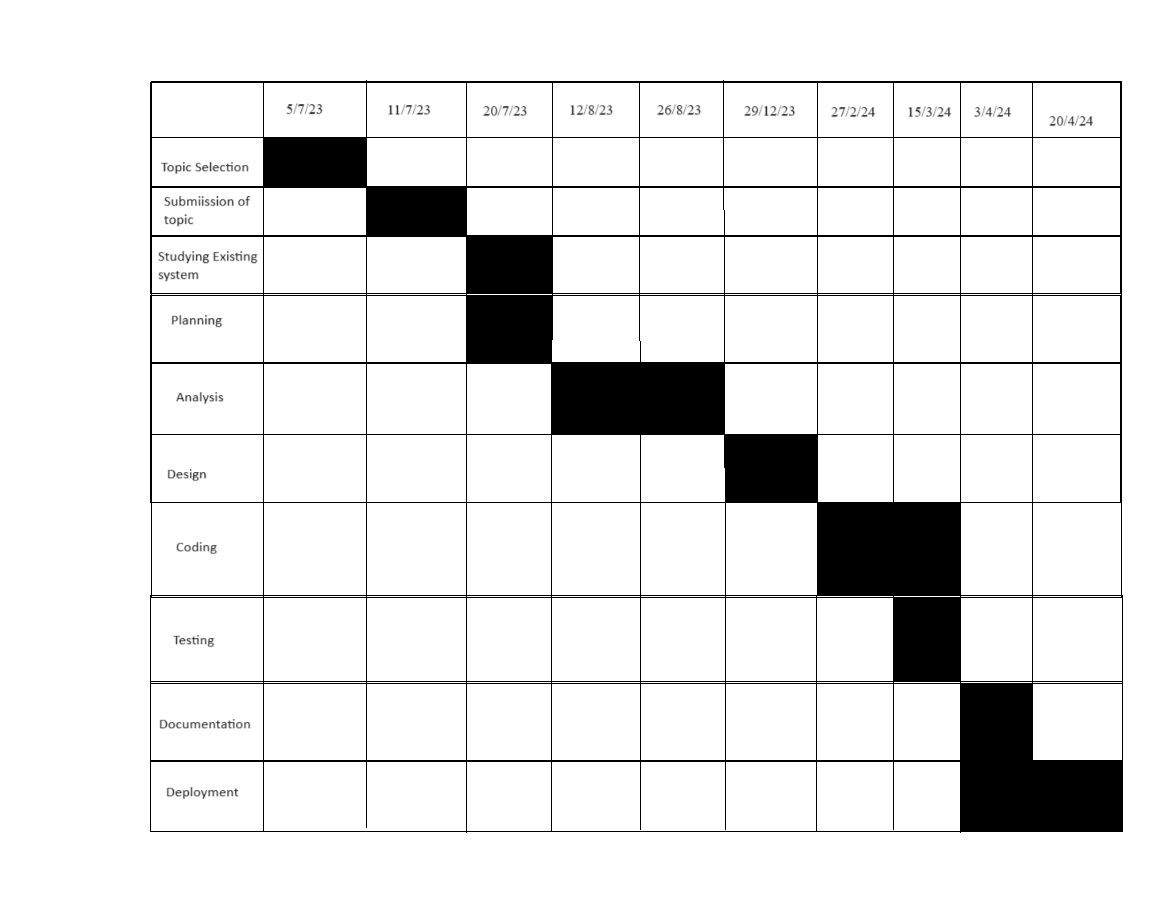
# Chapter-5 Model

## Spiral Model:-



When looking at a diagram of a spiral model, the radius of the spiral represents the cost of the project and represents the progress made in the current phase. Each phase begins with a goal for the design and ends when the developer or client reviews the progress. Every phase can be broken into four quadrants: identifying and understanding requirements, performing risk analysis, building the prototype and evaluation of the software's performance.

# Chapter 6 Gantt Chart



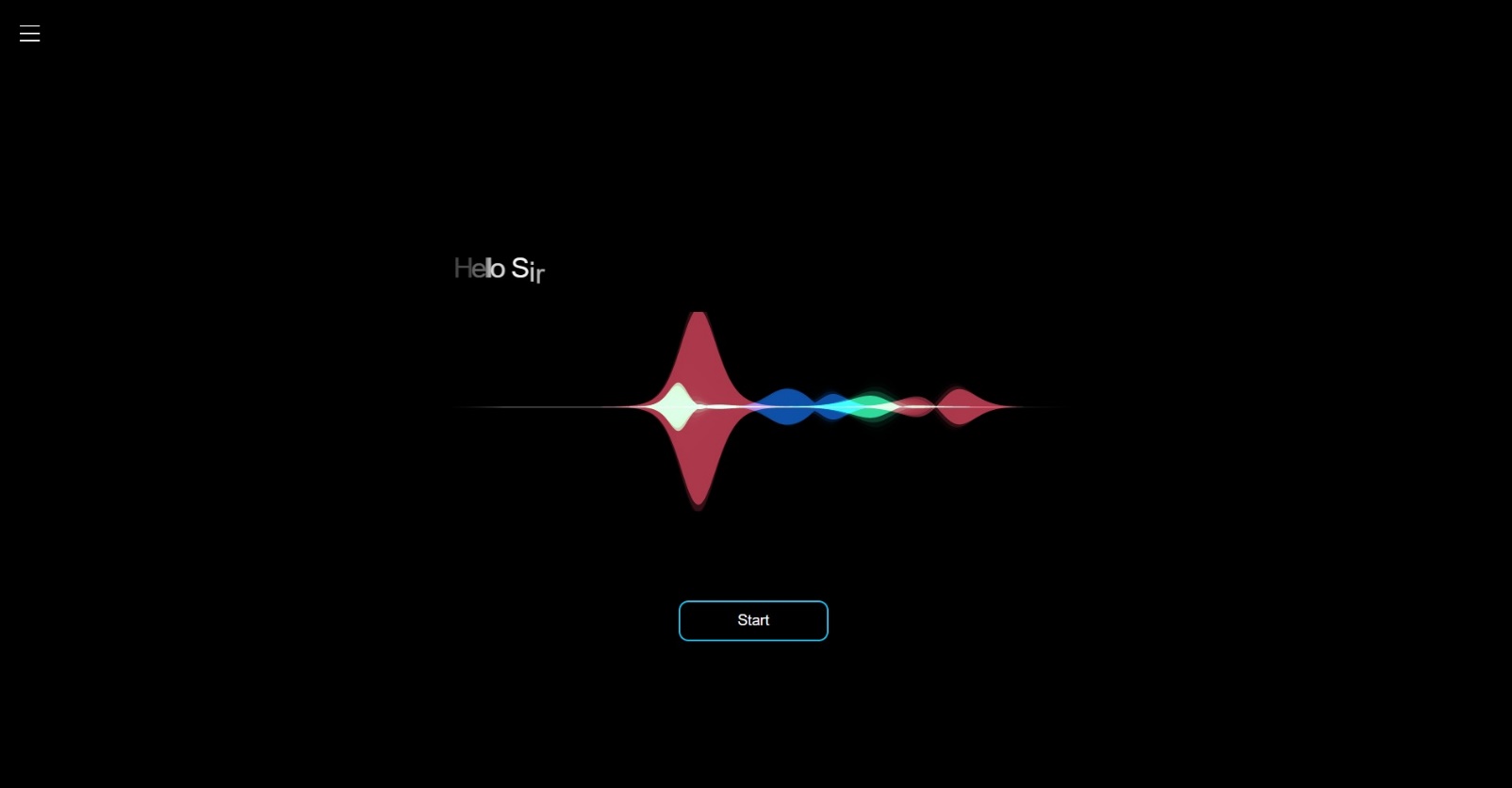
# Chapter-7

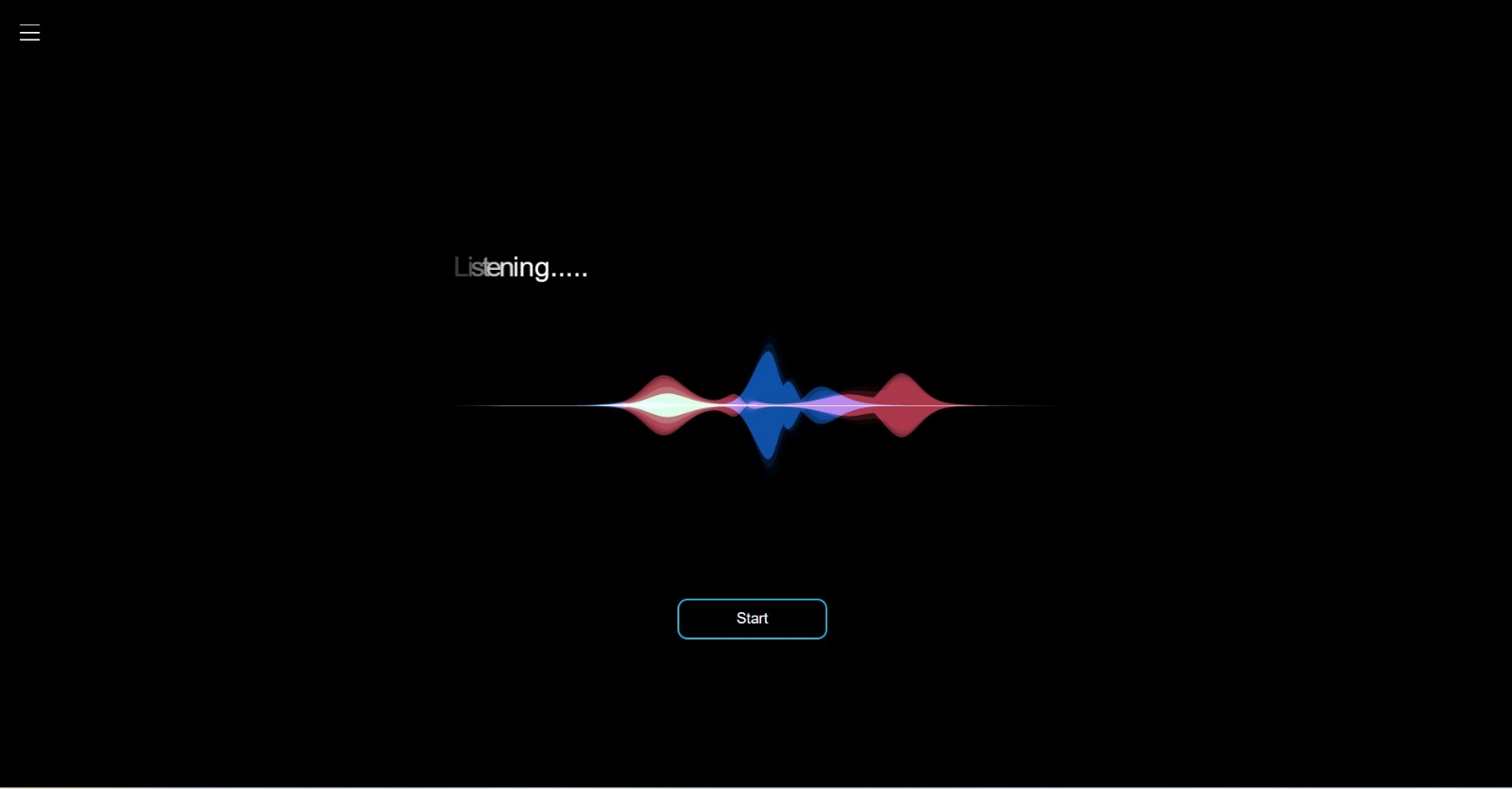
# 

# RESULTS

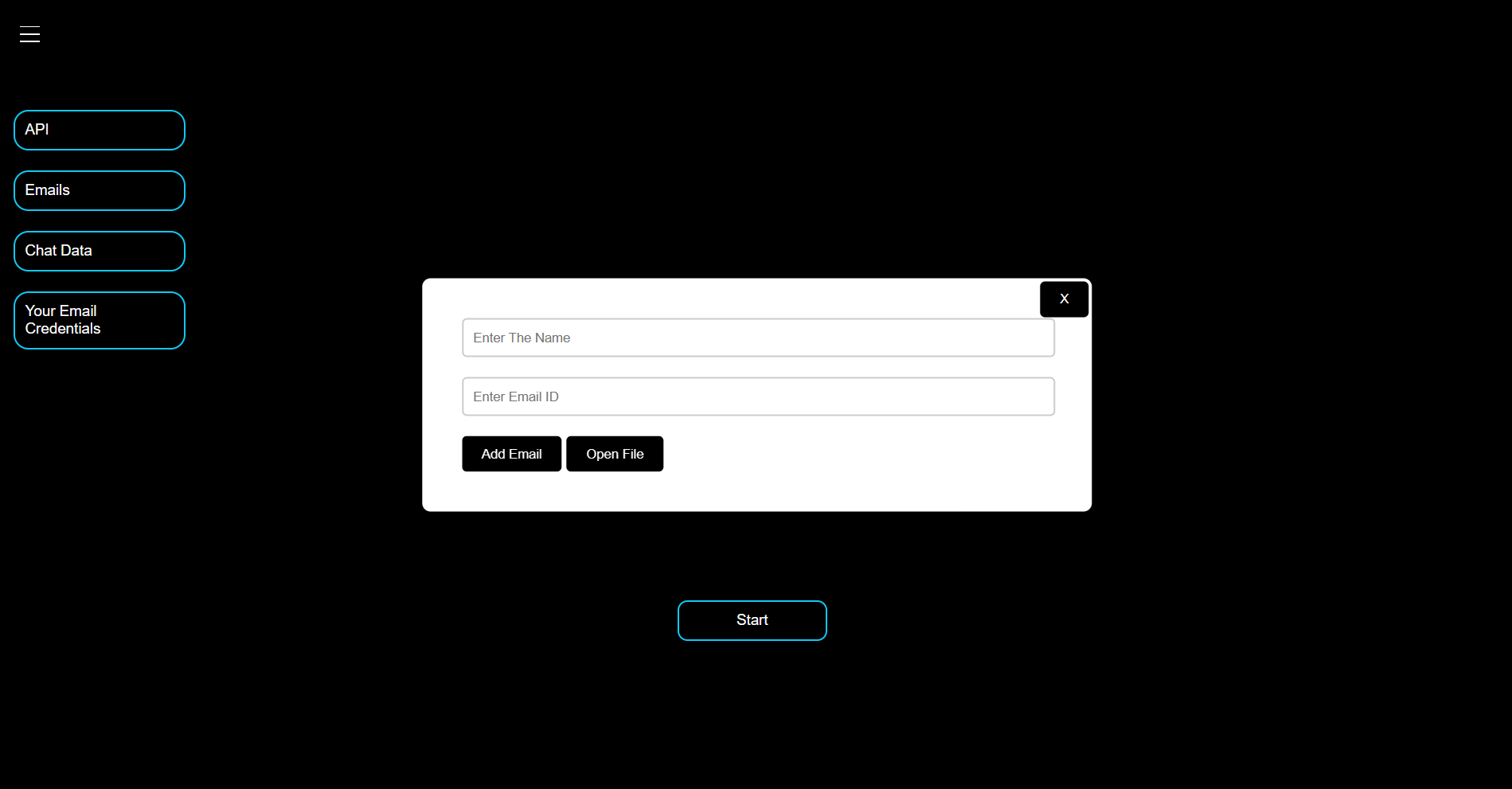
## Project Screenshots:

Home Page:

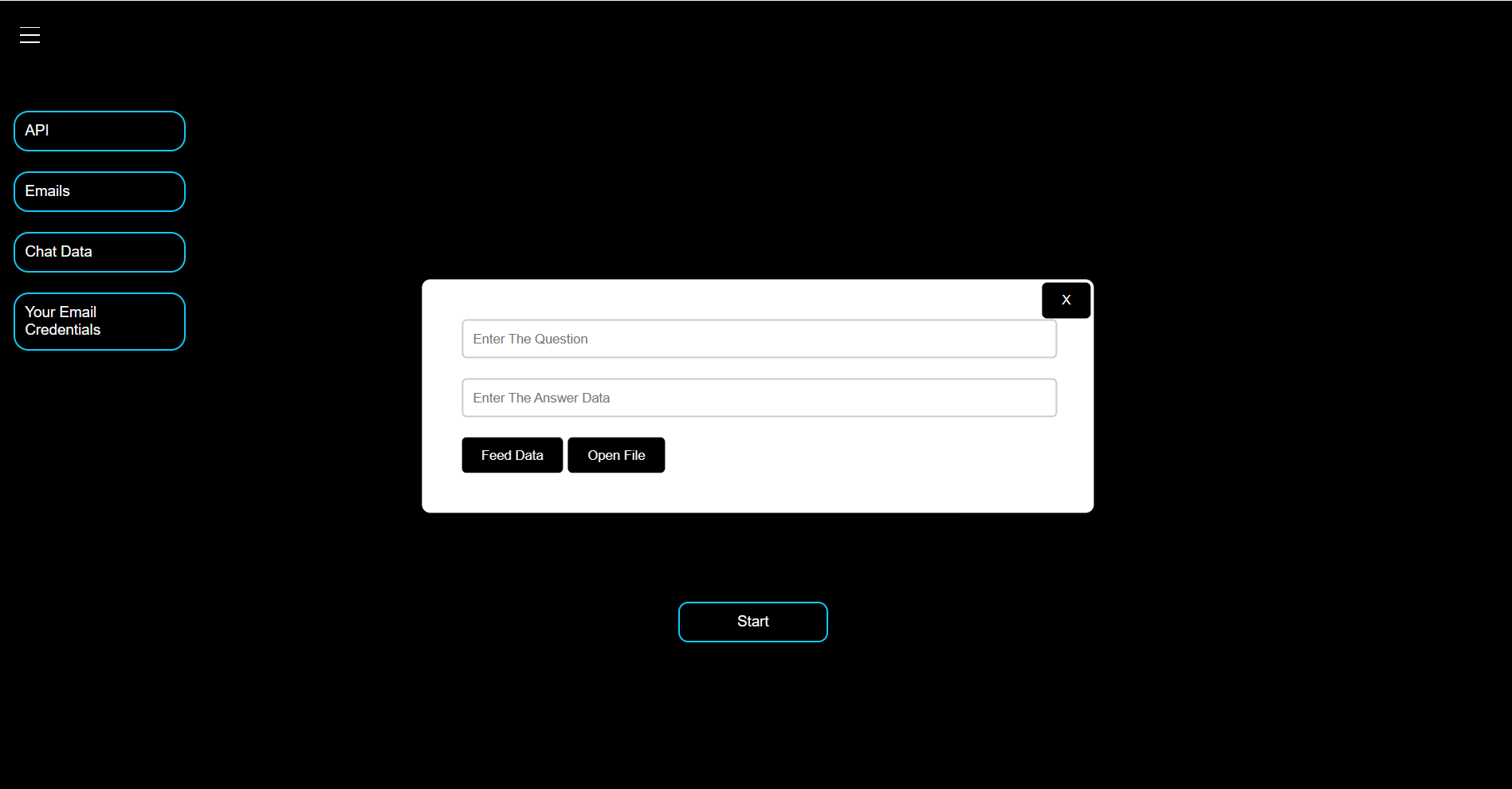




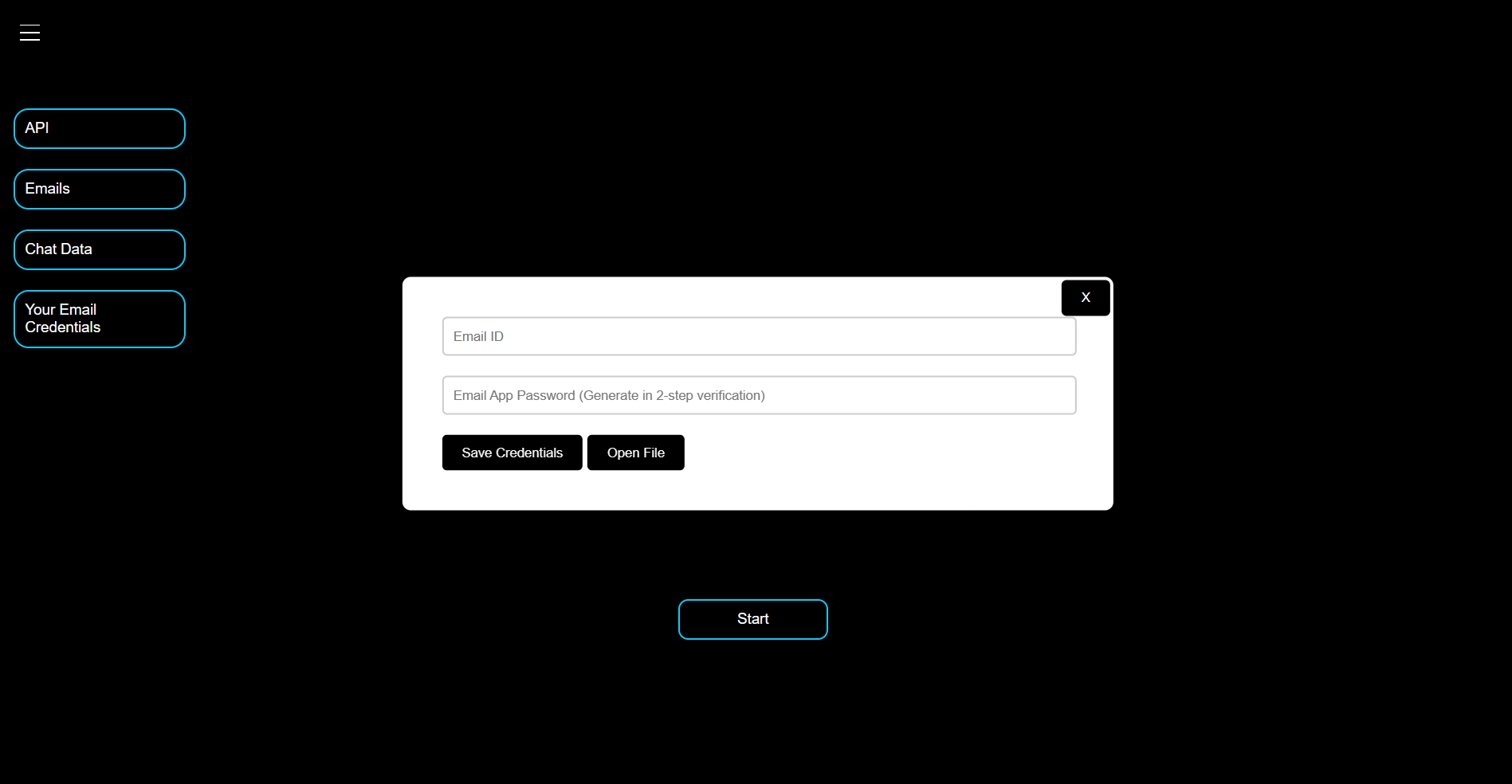
Add Email Data:



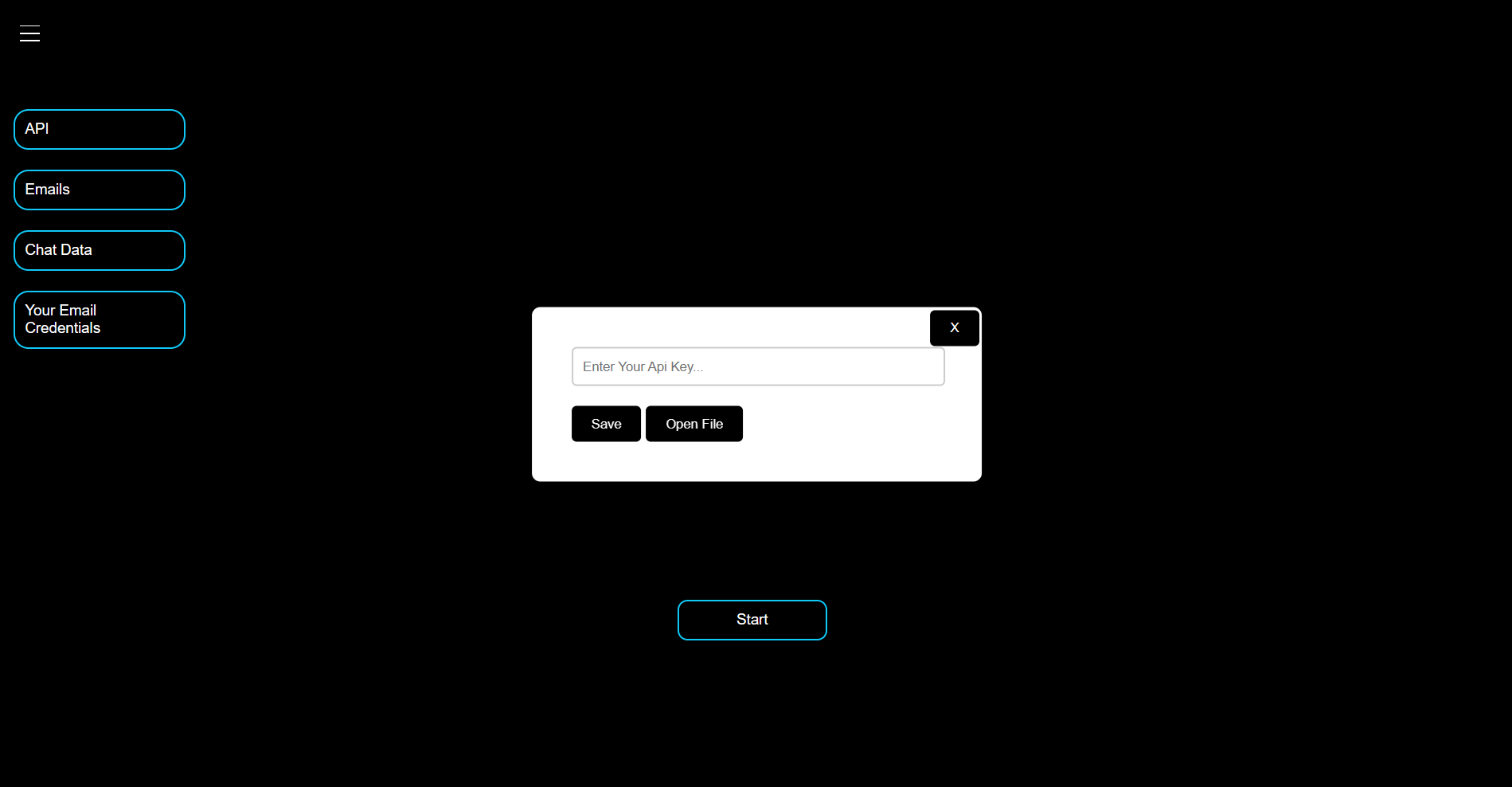
Add Chat Data:



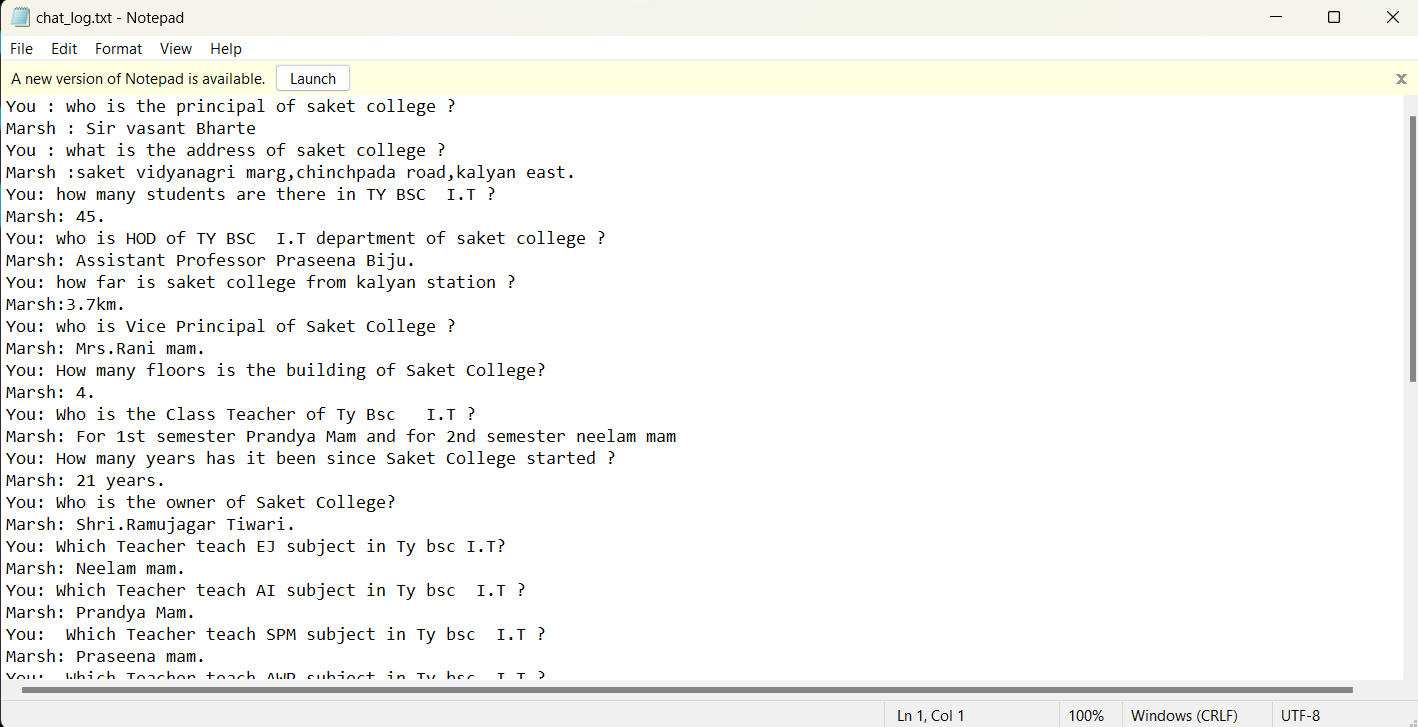
Enter Email Credentials:



Enter API Key:



Database:



# Chapter-8 Implementation and Testing

* 1. **Code Implementation**:

**Marsh.py**

import eel

import sys

import os

import subprocess

from Brain.MarshBrain import ReplyBrain

from Body.Speak import Speak

from Body.Listen import MicExecution

from Main import MainTaskExecution

*# Initialize Eel*

eel.init('web')

os.system('start msedge.exe --start-maximized --app="http://localhost:8000/index.html"')

*# Expose Python functions to JavaScript*

@eel.expose

def start\_assistant():

    Speak("Hello Sir")

    Speak("What can I do for you?")

    while True:

        data = MicExecution()

        data = str(data)

        value\_return = MainTaskExecution(data)

        if value\_return:

            pass

        elif len(data) < 3:

            pass

        else:

            reply = ReplyBrain(data)

            Speak(reply)

@eel.expose

def exit\_program():

    Speak("Ok, I'll see you later")

    sys.exit()

script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'API.txt')

@eel.expose

def saveTextToFile(data):

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'API.txt')

    with open(file\_path, 'w') as file:

        file.write(data)

@eel.expose

def openapi():

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'API.txt')

    subprocess.Popen(['notepad.exe', file\_path])

@eel.expose

def savecredToFile(email, password):

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'email\_id\_pass.txt')

    formatted\_cred = f"{email}\n{password}\n"

    with open(file\_path, 'w') as file:

        file.write(formatted\_cred)

@eel.expose

def opencred():

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'email\_id\_pass.txt')

    subprocess.Popen(['notepad.exe', file\_path])

@eel.expose

def saveemailToFile(data):

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'email\_data.txt')

    with open(file\_path, 'a') as file:

        file.write( '\n' + data )

@eel.expose

def openemail():

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'email\_data.txt')

    subprocess.Popen(['notepad.exe', file\_path])

@eel.expose

def savechatToFile(question, answer):

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'chat\_log.txt')

    formatted\_log = f"You: {question}\nMarsh: {answer}\n"

    with open(file\_path, 'a') as file:

        file.write(formatted\_log)

@eel.expose

def openchat():

    script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

    file\_path = os.path.join(script\_dir, 'Brain', 'Database', 'chat\_log.txt')

    subprocess.Popen(['notepad.exe', file\_path])

eel.start('index.html',mode=None, host='localhost', block=True)

**Listen.py**

import speech\_recognition as sr

import sys

from deep\_translator import GoogleTranslator

import eel

*# Listen*

def Listen():

    r = sr.Recognizer()

    with sr.Microphone() as source:

        print("Listening.....")

        eel.DisplayMessage('Listening.....')

        r.pause\_threshold = 1

        audio = r.listen(source,0,8)

    try :

        print("Recognizing...")

        eel.DisplayMessage('Recognizing.....')

        query = r.recognize\_google(audio, language ="hi")

    except:

        return ""

    query = str(query).lower()

    return query

*# Translation*

def TranslationHintoEng(Text):

    line= str(Text)

    result = GoogleTranslator(source='auto', target='english').translate(line)

    data= result.lower()

    print(f"You :  {data}.")

    eel.DisplayMessage(data)

    if "exit" in data or "goodbye" in data or "go to sleep" in data or "bye" in data or "get exited" in data or "see you later" in data or "farewell" in data:

        from Body.Speak import Speak

        Speak("Ok, I'll see you later")

        eel.close()

        sys.exit()

    return data

*#connecting the ears and translator*

def MicExecution():

    query = Listen()

    data= TranslationHintoEng(query)

    return data

**Speak.py**

import pyttsx3

import eel

def Speak(Text):

    engine = pyttsx3.init("sapi5")

    voices = engine.getProperty('voices')

    engine.setProperty('voice',voices[2].id)

    engine.setProperty('rate',160)

    print("")

    print(f"Marsh : {Text}.")

    print("")

    engine.say(Text)

    eel.DisplayMessage(Text)

    engine.runAndWait()

**Brain.py**

import os

from openai import OpenAI

from dotenv import load\_dotenv

base\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

*# Path to the API key text file*

api\_key\_file\_path = os.path.join(base\_dir, "DataBase", "API.txt")

*# Read the API key from the text file*

with open(api\_key\_file\_path, "r") as api\_key\_file:

    api\_key = api\_key\_file.read().strip()

*# Initialize the OpenAI client with the API key*

client = OpenAI(api\_key=api\_key)

load\_dotenv()

def ReplyBrain(question, chat\_log=None):

*#base\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  # Get the directory of the current script*

    chat\_log\_path = os.path.join(base\_dir, "DataBase", "chat\_log.txt")  *# Construct the full path to the chat log*

    with open(chat\_log\_path, "r", encoding= 'utf-8') as file\_log:

        chat\_log\_template = file\_log.read()

    if chat\_log is None:

        chat\_log = chat\_log\_template

    prompt = f'{chat\_log}You : {question}\nMarsh : '

    response = client.chat.completions.create(

        model="gpt-3.5-turbo",

        messages=[

            {"role": "system", "content": "from now on you are a casual ,friendly, sarcastic AI desktop assistant"},

            {"role": "system", "content": "your name is maarsh"},

            {"role": "system", "content": "you were created by  harsh and  maddy"},

            {"role": "system", "content": " harsh and  maddy are your creator"},

            {"role": "system", "content": " harsh and  maddy created you"},

            {"role": "system", "content": "you were made by  harsh and  maddy"},

            {"role": "system", "content": " harsh and  maddy are your makers"},

            {"role": "system", "content": " always add the prefix 'Sir' before mentioning Harsh and Maddy"},

            {"role": "system", "content": chat\_log},

            {"role": "user", "content": question}

        ],

        temperature=0.5,

        max\_tokens=60,

    )

    answer = response.choices[0].message.content.strip()

    chat\_log\_template\_update = chat\_log\_template + f"\nYou : {question} \nMarsh : {answer}"

    with open(chat\_log\_path, "w", encoding='utf-8') as file\_log:

        file\_log.write(chat\_log\_template\_update)

    return answer

**latest\_news.py**

from Body.Speak import Speak

import requests

import json

import speech\_recognition as sr

def listen():

    recognizer = sr.Recognizer()

    with sr.Microphone() as source:

        print("Listening...")

        recognizer.pause\_threshold = 1

        audio = recognizer.listen(source, 0, 5)

    try:

        print("Recognizing...")

        query = recognizer.recognize\_google(audio, language="en")

        print(query)

        return query.lower()

    except sr.UnknownValueError:

        print("Sorry, I couldn't understand what you said.")

    except sr.RequestError:

        print("Sorry, I'm having trouble accessing the recognition service.")

def latestnews():

    try:

        api\_dict = {"business" : "https://newsapi.org/v2/top-headlines?country=in&category=business&apiKey=5cae6364b921480cb5e8f0360f2f3f42",

                "entertainment" : "https://newsapi.org/v2/top-headlines?country=in&category=entertainment&apiKey=5cae6364b921480cb5e8f0360f2f3f42",

                "health" : "https://newsapi.org/v2/top-headlines?country=in&category=health&apiKey=5cae6364b921480cb5e8f0360f2f3f42",

                "science" :"https://newsapi.org/v2/top-headlines?country=in&category=science&apiKey=5cae6364b921480cb5e8f0360f2f3f42",

                "sports" :"https://newsapi.org/v2/top-headlines?country=in&category=sports&apiKey=5cae6364b921480cb5e8f0360f2f3f42",

                "technology" :"https://newsapi.org/v2/top-headlines?country=in&category=technology&apiKey=5cae6364b921480cb5e8f0360f2f3f42"

    }

        counter = 0

        url = None

        Speak("Of course! What specific topics are you interested in? For example: Technology, Sports....")

        field = listen()

        for key ,value in api\_dict.items():

            if key.lower() in field.lower():

                url = value

                break

        news = requests.get(url).text

        news = json.loads(news)

        Speak("Here are the top 5 headlines.")

        arts = news["articles"]

        for articles in arts :

            article = articles["title"]

            Speak(article)

            counter += 1

            if counter >= 5:

                break

        Speak("Is there anything else i can help you with?")

        return True

    except:

        Speak("I apologize, i was not able to find any news.")

        return True

**Open.py**

import keyboard

import pyautogui

import webbrowser

import requests

from Body.Speak import Speak

from bs4 import BeautifulSoup

from time import sleep

from datetime import datetime

def OpenExe(Query):

    Query = str(Query).lower()

    if "visit" in Query:

        Nameofweb = Query.replace("visit ","")

        Link = f"https://www.{Nameofweb}.com"

        webbrowser.open(Link)

        return True

    elif "launch" in Query:

        Nameoftheapp = Query.replace("open ","")

        pyautogui.press('win')

        sleep(1)

        keyboard.write(Nameoftheapp)

        sleep(1)

        keyboard.press('enter')

        sleep(0.5)

        return True

    elif "search" in Query:

        Nameofweb = Query.replace("search ","")

        Link = f"{Nameofweb}"

        webbrowser.open(Link)

        return True

    elif "open" in Query:

        Nameoftheapp = Query.replace("open ","")

        pyautogui.press('win')

        sleep(1)

        keyboard.write(Nameoftheapp)

        sleep(1)

        keyboard.press('enter')

        sleep(0.5)

        return True

    elif "start" in Query:

        Nameoftheapp = Query.replace("start ","")

        pyautogui.press('win')

        sleep(1)

        keyboard.write(Nameoftheapp)

        sleep(1)

        keyboard.press('enter')

        sleep(0.5)

        return True

    elif "temperature" in Query:

        def trim\_left(Query, word):

            index = Query.find(word)

            if index != -1:

                return Query[index:]

            else:

                return Query

        a= trim\_left(Query, 'temperature')

        tempofcity= a

        url = f"https://www.google.com/search?q={tempofcity}"

        response = requests.get(url)

        soup = BeautifulSoup(response.text, 'html.parser')

        data = soup.find('div', attrs={'class': 'BNeawe'}).text

        ans= f"The current temperature is {data}"

        Speak(ans)

        return True

    elif "weather" in Query:

        def trim\_left(Query, word):

            index = Query.find(word)

            if index != -1:

                return Query[index:]

            else:

                return Query

        a= trim\_left(Query, 'weather')

        weathofcity= a

        url = f"https://www.google.com/search?q={weathofcity}"

        response = requests.get(url)

        soup = BeautifulSoup(response.text, 'html.parser')

        data = soup.find('div', attrs={'class': 'BNeawe'}).text

        weath\_res= f"The current temperature is {data}"

        Speak(weath\_res)

        return True

elif "present date" in Query or "current date" in Query or "what is the date" in

Query or "whats the date" in Query or "what's the date" in Query or "what date it is"

in Query or "what date is it" in Query or "today's date" in Query or "todays date" in Query:

        now = datetime.now() # returns date in 'YYYY-MM-DD' format

        date= now.strftime("%d-%B-%y")

        Speak(f"the current date is {str(date)}")

        return True

    elif "present time" in Query or "current time" in Query or "what is the time" in Query or "whats the time" in Query or "what's the time" in Query or "what time it is" in Query or "what time is it" in Query:

        now1 = datetime.now()

        time= now1.strftime("%I:%M %p")  # returns date in 'YYYY-MM-DD' format

        Speak(f"the current time is {str(time)}")

        return True

**send\_email.py**

import smtplib

import speech\_recognition as sr

from Body.Speak import Speak

import os

# Define the path to the contacts file

CURRENT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

DATABASE\_DIR = os.path.join(CURRENT\_DIR, '..', 'Brain', 'Database')

CONTACTS\_FILE = os.path.join(DATABASE\_DIR, 'email\_data.txt')

def load\_contacts():

    contacts = {}

    with open(CONTACTS\_FILE, "r") as file:

        for line in file:

            if line.strip() == '' or line.strip().startswith('#'):

                continue

            name, email = line.strip().split(",")

            contacts[name.strip()] = email.strip()

    return contacts

def listen():

    recognizer = sr.Recognizer()

    with sr.Microphone() as source:

        print("Listening...")

        recognizer.pause\_threshold = 1

        audio = recognizer.listen(source, 0, 20)

    try:

        print("Recognizing...")

        query = recognizer.recognize\_google(audio, language="en")

        print(query)

        return query.lower()

    except sr.UnknownValueError:

        print("Sorry, I couldn't understand what you said.")

    except sr.RequestError:

        print("Sorry, I'm having trouble accessing the recognition service.")

Email\_FILE = os.path.join(DATABASE\_DIR, 'email\_id\_pass.txt')

def get\_credentials():

    with open(Email\_FILE, 'r') as file:

        lines = file.readlines()

        email = lines[0].strip()

        password = lines[1].strip()

    return email, password

def send\_email(to, message):

    try:

        server = smtplib.SMTP('smtp.gmail.com', 587)

        server.starttls()

        email, password = get\_credentials()

        server.login(email, password)

        server.sendmail(email, to, message)

        Speak("Message sent successfully")

        return True

    except smtplib.SMTPAuthenticationError:

        Speak("Authentication error: Please check your email credentials.")

    except smtplib.SMTPException as e:

        Speak(f"An error occurred: {e}")

    finally:

        server.quit()

def send\_email\_prompt():

    contacts = load\_contacts()

    Speak("To whom do you want to send the email?")

    recipient = listen()

    if not recipient:

        Speak("Please try again by using the command, Send Email")

        return True

    to = ""

    for name, email in contacts.items():

        if name in recipient:

            to = email

            break

**index.html**

   <!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="style.css">

    <!-- texllate -->

    <link rel="stylesheet" href="/texllate/animate.css">

    <title>Marsh -AI Desktop Assistant</title>

</head>

<body>

    <!--Jquery  -->

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>

    <div class="container">

        <div class="row">

            <div class="col-md-12">

                <div class="d-flex justify-content-center align-items-center" style="height: 100vh;">

                    <div class="content">

                        <p class="text-start text-light mb-4 siri-message" style="font-size: 28px;"></p>

                        <div id="siri-container" align="center"></div>

                    </div>

                </div>

            </div>

        </div>

    </div>

    <div id="savedAlert" class="alert">

        Saved!

        <span class="closebtn" onclick="this.parentElement.style.display='none';">&times;</span>

    </div>

    <div id="validationAlert" class="alert">

        Please fill in all fields before saving.

        <span class="closebtn" onclick="this.parentElement.style.display='none';">&times;</span>

    </div>

        <!-- toggle -->

    <div class="toggle-btn" onclick="toggleSidebar()">

        <div class="bar"></div>

        <div class="bar"></div>

        <div class="bar"></div>

    </div>

    <div id="sidebar">

        <a href="#" class="sidebar-button" onclick="showPopupCard()">API</a>

        <a href="#" class="sidebar-button" onclick="showPopupCardemail()">Emails</a>

        <a href="#" class="sidebar-button" onclick="showPopupCardchatdata()">Chat Data</a>

        <a href="#" class="sidebar-button" onclick="showPopupCardcred()">Your Email Credentials</a>

    </div>

    <!-- API -->

    <div id="popupCardapi" class="popup-card">

        <input type="text" id="inputText" placeholder="Enter Your Api Key...">

        <button onclick="saveInput('popupCardapi')">Save</button>

        <!-- Cancel button -->

        <button onclick="showapifile()" class="cancel-button">Open File</button>

        <!-- Window exit button -->

        <button onclick="hidePopupCard()" class="exit-button">X</button>

    </div>

    <!-- email -->

    <div id="popupCardemail" class="popup-card">

        <input type="text" id="inputTextn" placeholder="Enter The Name ">

        <input type="text" id="inputTexte" placeholder="Enter Email ID ">

        <button onclick="emailsaveInput('popupCardemail')">Add Email</button>

        <!-- Cancel button -->

        <button onclick="showemailfile()" class="cancel-button">Open File</button>

        <!-- Window exit button -->

        <button onclick="hidePopupCardemail()" class="exit-button">X</button>

    </div>

    <!-- chat log -->

    <div id="popupCardchatdata" class="popup-card">

        <input type="text" id="inputTextqn" placeholder="Enter The Question ">

        <input type="text" id="inputTexteans" placeholder="Enter The Answer Data ">

        <button onclick="chatdatasaveInput('popupCardchatdata')">Feed Data</button>

        <!-- Cancel button -->

        <button onclick="showchatfile()" class="cancel-button">Open File</button>

        <!-- Window exit button -->

        <button onclick="hidePopupCardchatdata()" class="exit-button">X</button>

    </div>

    <!-- email cred -->

    <div id="popupCardemailcred" class="popup-card">

        <input type="text" id="inputTextem" placeholder="Email ID ">

        <input type="text" id="inputTextpass" placeholder="Email App Password (Generate in 2-step verification) ">

        <button onclick="credsaveInput('popupCardemailcred')">Save Credentials</button>

        <!-- Cancel button -->

        <button onclick="showcredfile()" class="cancel-button">Open File</button>

        <!-- Window exit button -->

        <button onclick="hidePopupCardcred()" class="exit-button">X</button>

    </div>

    <div class="run-btn">

        <button id="run-button" class="run-button" onclick="startSiriWave()">Start</button>

    </div>

    <!--  -->

    <!-- JS -->

    <!--  -->

    <!-- eel -->

    <script type="text/javascript" src="/eel.js"></script>

    <!-- siriwave -->

    <script src="https://unpkg.com/siriwave/dist/siriwave.umd.min.js"></script>

    <!-- texllate -->

    <script src="/texllate/jquery.fittext.js"></script>

    <script src="/texllate/jquery.lettering.js"></script>

    <script src="http://jschr.github.io/textillate/jquery.textillate.js"></script>

    <script src="main.js"></script>

    <script src="control.js"></script>

</body>

</html>

**Main.js**

window.onbeforeunload = function () {

    eel.exit\_program(); // Call Python function to exit program

};

$(document).ready(function () {

    $('.siri-message').textillate({

        loop: true,

        sync: true,

        in: {

            effect: "fadeInUp",

        },

        out: {

            effect: "fadeOutRight",

        }

    })

});

function toggleSidebar() {

    var sidebar = document.getElementById('sidebar');

    if (sidebar.style.left === "0px") {

        sidebar.style.left = "-200px";

    } else {

        sidebar.style.left = "0px";

    }

}

// siri wave

var siriWave = new SiriWave({

    container: document.getElementById("siri-container"),

    width: 640,

    height: 200,

    style: "ios9",

    amplitude: 4,

    frequency: 1.5,

    speed: 0.2,

    autostart: false,

});

// Function to start the Siri wave

function startSiriWave() {

    eel.start\_assistant();

    siriWave.start();

}

function validateInputs(cardId) {

    var card = document.getElementById(cardId);

    var inputs = card.querySelectorAll('input[type="text"]');

    for (var i = 0; i < inputs.length; i++) {

        if (inputs[i].value.trim() === '') {

            return false;

        }

    }

    return true;

}

function showValidationAlert() {

    var alert = document.getElementById('validationAlert');

    alert.style.display = 'block';

    setTimeout(function () {

        alert.style.display = 'none';

    }, 2000); // Hide after 2 seconds

}

// Function to show the popup card

function showPopupCard() {

    var popupCard = document.getElementById('popupCardapi');

    popupCard.style.display = 'block';

}

// Function to hide the popup card

function hidePopupCard() {

    var popupCard = document.getElementById('popupCardapi');

    popupCard.style.display = 'none';

}

function showapifile() {

    eel.openapi();

}

// Function to save the input

function saveInput(cardId) {

    if (validateInputs(cardId)) {

        var inputText = document.getElementById('inputText').value;

        eel.saveTextToFile(inputText);

        // Your save logic here

        showSavedAlert();

    } else {

        showValidationAlert();

    }

}

// Function to show the email popup card

function showPopupCardemail() {

    var emailpopupCard = document.getElementById('popupCardemail');

    emailpopupCard.style.display = 'block';

}

// Function to hide the email popup card

function hidePopupCardemail() {

    var emailpopupCard = document.getElementById('popupCardemail');

    emailpopupCard.style.display = 'none';

}

function showemailfile() {

    eel.openemail();

}

// Function to save the input

function emailsaveInput(cardId) {

    if (validateInputs(cardId)) {

        var inputTextname = document.getElementById('inputTextn').value;

        var inputTextemail = document.getElementById('inputTexte').value;

        var combineddata = inputTextname + ',' + inputTextemail;

        eel.saveemailToFile(combineddata);

        showSavedAlert();

    } else {

        showValidationAlert();

    }

}

// Function to show the chat data popup card

function showPopupCardchatdata() {

    var chatdatapopupCard = document.getElementById('popupCardchatdata');

    chatdatapopupCard.style.display = 'block';

}

function showchatfile() {

    eel.openchat();

}

// Function to hide the chat data popup card

function hidePopupCardchatdata() {

    var chatdatapopupCard = document.getElementById('popupCardchatdata');

    chatdatapopupCard.style.display = 'none';

}

// Function to save the input

function chatdatasaveInput(cardId) {

    if (validateInputs(cardId)) {

        var inputTextqn = document.getElementById('inputTextqn').value;

        var inputTexteans = document.getElementById('inputTexteans').value;

        eel.savechatToFile(inputTextqn, inputTexteans);

        showSavedAlert();

    } else {

        showValidationAlert();

    }

}

// Function to show the email credentials  popup card

function showPopupCardcred() {

    var credpopupCard = document.getElementById('popupCardemailcred');

    credpopupCard.style.display = 'block';

}

function showcredfile() {

    eel.opencred();

}

// Function to hide the email popup card

function hidePopupCardcred() {

    var credpopupCard = document.getElementById('popupCardemailcred');

    credpopupCard.style.display = 'none';

}

function credsaveInput(cardId) {

    if (validateInputs(cardId)) {

        var inputTextem = document.getElementById('inputTextem').value;

        var inputTextpass = document.getElementById('inputTextpass').value;

        eel.savecredToFile(inputTextem, inputTextpass);

        showSavedAlert();

        hidePopupCardcred();

    } else {

        showValidationAlert();

    }

}

function showSavedAlert() {

    var alert = document.getElementById('savedAlert');

    alert.style.display = 'block';

    setTimeout(function () {

        alert.style.display = 'none';

    }, 2000); // Hide after 2 seconds

}

# Chapter-9 CONCLUSION AND FUTURE SCOPE

* 1. **CONCLUSION**:
     + We have tried to solve a real time problem that we are facing at the time of buying medicines and the cost became very for sometimes.
     + We have used Flask framework as it is a great framework for modern development. Microframeworks or libraries are minimal or provide an ecosystem of components that can be mixed and matched as needed, focusing on including only what you need.
     + Simpler Development: User can move around and contribute to a flask application pretty easily. It’s less opinionated so fewer standards to learn.
     + Flexibility: There are very few parts of flask that can not be easily and safely altered because of its simplicity and minimality.
     + Performance of flask framework is better as there are fewer levels of abstraction between user and database.
     + E-commerce site will help user to compare their products and buy from that site which is suitable for them.

# Future Scope:

* It will reduce the efforts of users who are suffering from high cost at the time of buying medicines.
* By Exporting files user will get the knowledge about all the medicines.

By the help of E-Commerce site user will get their product from different e-commerce sites by

comparing their products.

# Chapter 10

## REFERENCES:

* + [www.wikipedia.org](http://www.wikipedia.org/)
  + [www.google.com](http://www.google.com/)
  + [www.studymafia.com](http://www.studymafia.com/)
  + [www.studynight.com](http://www.studynight.com/)
  + chat.openai.com