DESKTOP-ASSISTANT

A Project Report

Submitted in Partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF SCIENCE (COMPUTER SCIENCE)

By ALBAR KHAN

Seat No.________________

Under the esteemed guidance of

Prof. Javed Pathan

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Affiliated to University of Mumbai)

MUMBAI-400050

MAHARASHTRA

2023-2024

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Affiliated to University of Mumbai)

MUMBAI-MAHARASHTRA-400050

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled, **"Desktop Assistant[PYTHON]"**, is benefited work of **ALBAR KHAN** bearing **Seat** No.: ___________ **Roll No.** 11 submitted **in partial fulfilment of the requirements for the award of degree of BACHELOR OF SCIENCE in COMPUTER SCIENCE from University of Mumbai.**

Project Guide **HOD**

External Examiner

Date: / / 2023 **College Seal**

ACKNOWLEDGEMENT

I owe special thanks to the Department of Computer Science of **Rizvi College of Arts Science and Commerce** for giving me a chance to prepare this project dissertation. I thank the Principal, **Professor Khan Ashfaq Ahmad** for his leadership and management. I thank the Coordinator and Head of the Department **Professor Arif Patel** for providing us the required facilities and guidance throughout the course which culminated into this thesis. Last and not the least to the project guide this semester**- Professor Javed Pathan**. Deep gratitude to the staff and faculty of Rizvi College for their help and support. And also, my beloved **Parents and Classmates** for their infinite support and love.

DESKTOP ASSISTANT USING PYTHON

INTRODUCTION:

The very first voice activated product was released in 1922 as Radio Rex. This toy was very simple, wherein a toy dog would stay inside a dog house until the user exclaimed its name, “Rex” at which point it would jump out of the house. This was all done by an electromagnet tuned to the frequency similar to the vowel found in the word Rex, and predated modern computers by over 20 years.[2]

In the 21st century, human interaction is being replaced by automation very quickly. One of the main reasons for this change is performance. There’s a drastic change in technology rather than advancement. In today’s world, we train our machines to do their tasks by themselves or to think like humans using technologies like Machine Learning, Neural Networks, etc. Now in the current era, we can talk to our machines with the help of virtual assistants.[2]

Virtual desktop assistant is an awesome thing. If you want your machine to run on yourcommand like Jarvis did for Tony. Yes it is possible. It is possible using Python. Pythonoffers a good major library so that we can use it for making a virtual assistant. We have so many virtual assistants, such as Apple’s Siri, Amazon’s Alexa and Microsoft’s Cortana and Amazon's Alexa and this has been an inspiration for us to do this as a project. Windows has Sapi5 and Linux has Espeak which can help us in having the voice from our machine. It is a weak A.I.[1]

Voice searches have dominated over text search. Web searches conducted via mobile devices have only just overtaken those carried out using a computer. And the analyst are already predicting that 50% of searches will be via voice by 2024. Virtual assistants are turning out to be smarter than ever. Allow your intelligent assistant

Desktop Assistant is software program that help you ease your day-to-day tasks, such asshowing weather report , creating reminders , making shopping list etc. This system will be designed to be used efficiently on desktops, Personal assistant software improves userproductivity by managing routine tasks of the user and by providing information from information from online sources to the user. It is effortless to use.

OBJECTIVES:

Main objective of building Desktop assistant software (a virtual voice assistant) is using semantic data sources available on the web, user generated content and providing knowledge from the knowledge databases. The main purpose of an intelligent virtual assistant is to answer questions that users may have.

Provide a topic for research and continue with your tasks while the assistant does the research. Another difficult task is to remember test dates, birthdates or anniversaries. It comes with a surprise when you enter the class and realize it is class test today. Just tell assistant in advance about your tests and she reminds you well in advance so you can prepare for the test.

One of the main advantages of voice searches is their rapidity. In fact, voice is reputed to be four times faster than a written search: whereas we can write about 40 words per minute we are capable of speaking around 150 during the same period of time.the ability of desktop assistants to accurately recognize spoken words are a prerequisite for them to be adopted by consumer.[1]

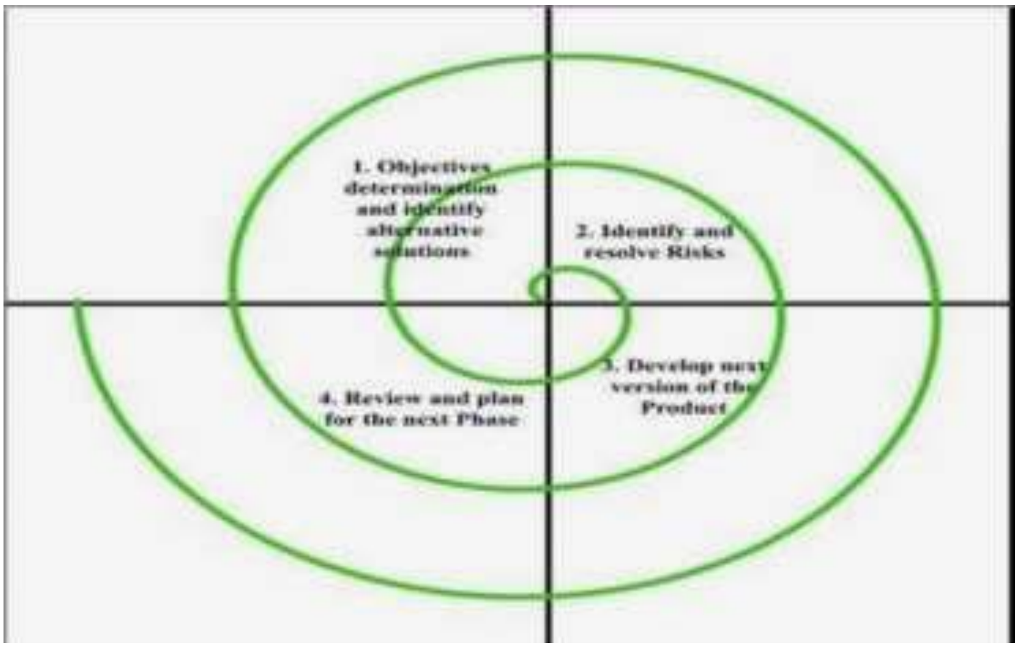
SCOPE:

Desktop assistants (virtual voice assistant) will continue to offer more individualized experiences as they get better at differentiating between voices. However, it’s not just developers that need to address the complexity of developing for voice as brands also need to understand the capabilities of each device and integration and if it makes sense for them specific brand. They will also need to focus on maintaining a user experience that is consistent within the coming years as complexity becomes more of a concern.[1]

METHODOLOGY:

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models. Each process model follows a Series of steps unique to its type to ensure successin the process of software

Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.[1]


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

SPIRAL MODEL [1]

TOOLS AND TECHNOLOGY:

Tools and technologies that are going to be used in this project are Visual Studio Code IDE. Visual Studio Code is an excellent Python editor that works on any operating system with a variety of Python interpreters. It provides auto complete and IntelliSnse, linting, debugging, and unit testing, along with the ability to easily switch between enviroments. Visual Studio Code is an open-source, extensible,, light-weight editor available on all platforms, making it a great platform for Python development.[1]

And I will create all .py files in Visual Studio Code. Along with this I Will use following modules and libraries in my project. pyttsx, SpeeechRecognnition, Datetime, Wikipedia, pywhatkit etc. I will also create a live GUI for interacting with the Assistant as it gives a design and interesting look while having the conversation[2]

Developer Configuration:

OS-Windows 11 processor-Ryzen

5,
RAM- 8gb SSD-

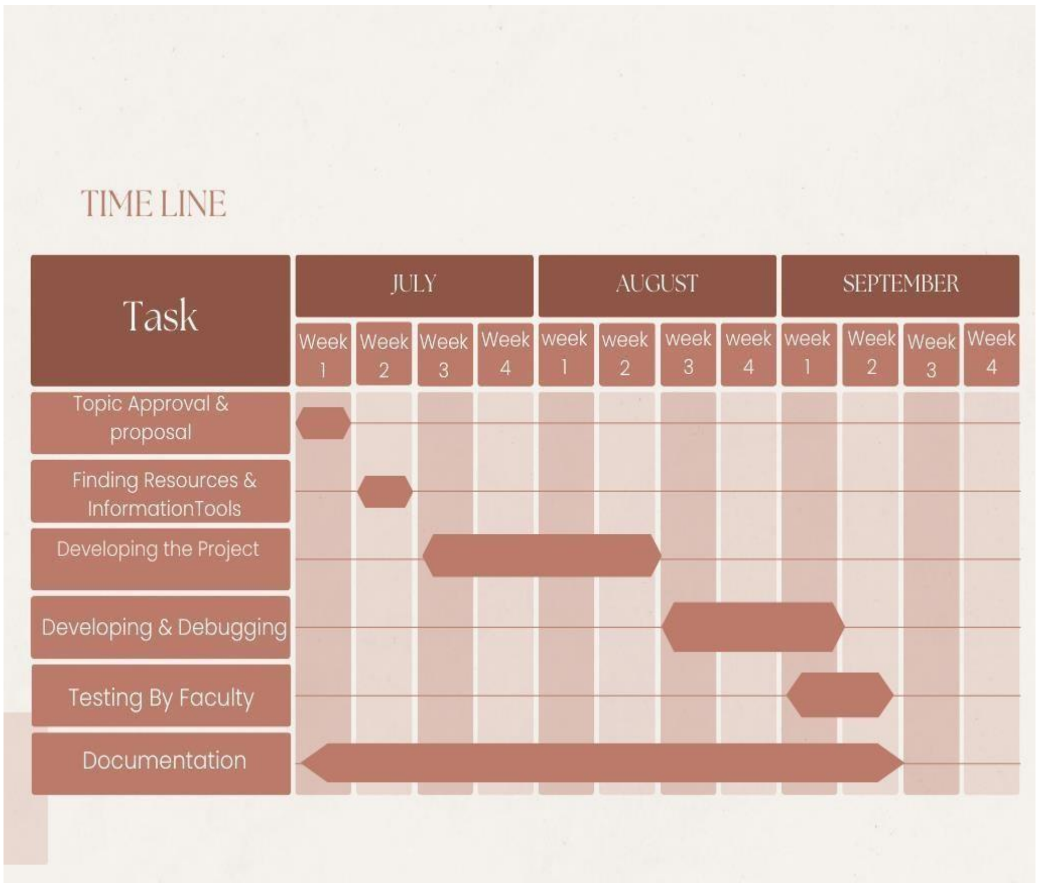
512 microphone

User Configuration:

OS – Windows 7 and above processor

– intel core i3 or Ryzen 3
Ram – 4gb
SSD-128 or HDD-258
Microphone

TIMELINE:


TIME LINE
Task
JULY
AUGUST
SEPTEMBER
Week 1
Week 2
Week 3
Week 4
week 1
week 2
week 3
week 4
week 1
Week 2
Week 3
Week 4
Topic Approval & proposal
Finding Resources & InformationTools
Developing the Project
Developing & Debugging
Testing By Faculty
Documentation

EXPECTED OUTCOME:

I expect That the desktop assistant will be capable oof voice interaction and perform the tasks through voice command such as playing music, making to-do list, setting alarms, playing audiobook, and providing weather info, sports, and the other real-time information, sending messages on what'sapp through voice command ,opening and closing apps and shutting down the system itself through voice command.Desktop assistant sense able users to speak natural language voice commands in order to operate the device and its apps. There is an increased overall ever-evolving digital world where speed, efficiency, and convience are constantly being optimized, it's clear that we are moving towards less screen interaction.[1]

DECLARATION

I hereby declare that the project entitled, **"Desktop Assistant"** done at **Rizvi College of Arts, Science and Commerce,** has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (COMPUTER SCIENCE)** to be submitted as final semester project as part of our curriculum.

ALBAR KHAN

ABSTRACT

In an era marked by an ever-expanding digital landscape and the relentless influx of information, the demand for a versatile, user-friendly, and highly efficient desktop assistant has surged dramatically. This project represents our unwavering commitment to the development of a sophisticated desktop assistant, meticulously crafted using the versatile Python programming language. This desktop assistant, underpinned by advanced Natural Language Processing (NLP) techniques, emerges as a highly adaptable virtual companion, finely tuned to cater to the distinct needs of its users.

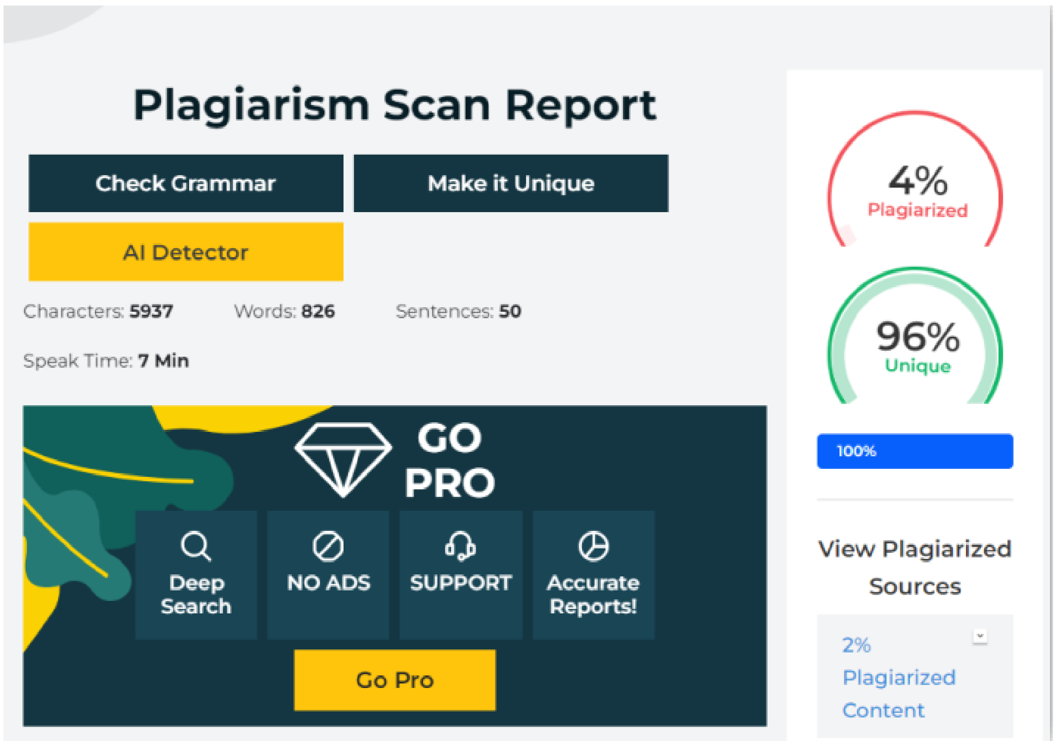
At its core, this multifunctional desktop assistant is a technological marvel. It empowers users to seamlessly accomplish a diverse array of tasks, seamlessly transitioning from answering queries to setting timely reminders, delivering real-time weather forecasts, managing schedules and calendars, composing and dispatching emails, and seamlessly orchestrating control over a multitude of desktop applications and processes. This remarkable range of capabilities is at the fingertips of users, accessible through both voice and text commands. What truly sets this assistant apart is its remarkable capacity to evolve and adapt based on user interactions, culminating in a personalized and intuitive user experience that leaves no digital stone unturned.

The foundation of this project rests firmly on Python's rich ecosystem of libraries and Application Programming Interfaces (APIs). These libraries offer a versatile toolkit, allowing us to harness a wide array of functionalities that make our desktop assistant robust and versatile. Key components encompass cutting-edge speech recognition, text-to-speech synthesis, web scraping for real-time information retrieval, and seamless integration with external APIs to enable tasks such as automation and data acquisition. The result is a formidable, extensible, and highly customizable desktop assistant, designed to serve with distinction in both personal and professional computing environments.

In summary, this desktop assistant project isn't just another application; it represents a culmination of Python's immense potential and sophisticated NLP techniques. Its overarching goal is to streamline tasks, enhance productivity, and elevate the overall computing experience for users across an extensive spectrum of applications and use cases.

PLAGIARISM REPORT

A plagiarism report is a document or output generated by plagiarism detection software or tools that highlights instances of potential plagiarism in a given piece of content. It typically compares the submitted content, such as an essay, article, or research paper, to a vast database of existing academic papers, websites, and published works to identify similarities or matches. The plagiarism report usually includes the following information: • Similarity Score: This is a percentage that represents the amount of content in the document that matches existing sources. A higher similarity score indicates a higher potential for plagiarism. • Highlighted Text: The report will highlight specific sentences or phrases within the document that are similar to other sources, making it easy for the user to identify the problematic areas. • Source References: Plagiarism detection tools often provide links or references to the sources that match the content, allowing users to verify the originality of the material and the source of the match. • Overall Assessment: Some reports provide an overall assessment of the document's originality, flagging it as potentially plagiarized or not, based on the similarity score and the highlighted text



Plagiarism Scan Report

Characters: **5937** Words: **826** Sentences: **50**

Speak Time: **7 Min**

4% Plagiarized

96% Unique

100%

View Plagiarized Sources

2% Plagiarized Content

TABLE OF CONTENTS

Chapter 1. Introduction

1.1 Introduction to the System

Artificial Intelligence, when integrated with machines, empowers them to mimic human-like thinking and interaction. In this realm, computer systems are meticulously designed to operate with a level of autonomy that necessitates minimal human intervention. Python, an ever-evolving language, is the ideal canvas for crafting a script to create a Voice Assistant. The instructions for this digital assistant can be tailored to meet the specific needs and preferences of users. Notably, Python boasts a dedicated API called Speech Recognition, which enables the transformation of spoken language into text. The journey of crafting my very own digital assistant was nothing short of fascinating.[1]

The capability to send emails without the need for manual typing, initiate Google searches without launching a web browser, and effortlessly carry out numerous daily tasks like music playback or launching a preferred Integrated Development Environment (IDE) through a single voice command was a revelation. In the current technological landscape, the advancements are so profound that they can perform tasks as effectively, if not more so, than humans. This project underscored the idea that Artificial Intelligence, pervasive in virtually every field, serves as a catalyst for reducing human effort and optimizing time management.[1]

As the Voice Assistant harnesses the power of Artificial Intelligence, the results it delivers are characterized by extraordinary accuracy and efficiency. The assistant's role is to streamline tasks, sparing human effort and time. It effectively eliminates the need for typing, akin to conversing with another individual, issuing tasks through verbal commands. This assistant transcends the capabilities of a human counterpart and excels in terms of task execution and efficiency. The libraries and packages utilized in the creation of this assistant are meticulously chosen for their capacity to optimize time efficiency

The functionalities embedded within this Voice Assistant project are diverse and comprehensive. It can execute tasks such as sending emails, reading PDF documents, sending text messages via WhatsApp, opening the command prompt, launching your preferred IDE, or notepad, playing music, conducting Wikipedia searches, and opening websites like Google, YouTube, and others in a web browser. It also offers weather forecasts and the ability to set desktop reminders as per the user's preference. Furthermore, it is even equipped to engage in basic conversation, adding a personalized touch to interactions.

I created all .py files in Visual Studio code. Along with this I used following modules and libraries in my project. pyttsx3, SpeechRecognition, Datetime, Wikipedia, Smtplib, pywhatkit, pyjokes, pyPDF2, pyautogui, pyQt etc. I have created a live GUI for interacting with the JARVIS as it gives a design and interesting look while having the conversation[1]

In summary, this Voice Assistant project exemplifies the fusion of cutting-edge technology with the ease of Python programming, showcasing the transformative capabilities of Artificial Intelligence. It not only simplifies tasks but also embodies a new era of efficient and personalized digital interactions, epitomizing the ongoing trend of AI-driven solutions in modern society.

1.2 Problem Definition

In the ever-evolving contemporary digital landscape, individuals and professionals are confronted by an array of formidable challenges linked to information management, task automation, and the efficient utilization of computer resources. These challenges have become emblematic of our times, touching every facet of our daily lives and work. They include:

- **Information Overload:**
  The age of information abundance is both a blessing and a curse. The internet and myriad applications offer a cornucopia of data, but this abundance can often lead to a sense of drowning in a sea of information. Users find themselves on a quest for needles in digital haystacks, seeking to pinpoint and retrieve specific, meaningful information from the vast sea of data available at their fingertips.

- **Task Complexity:**
  The modern workstation or personal computer resembles a digital Swiss Army knife with a plethora of applications and processes. Navigating, managing, and harnessing these diverse tools can become an intricate dance, one that poses a particular challenge to users who navigate a labyrinth of diverse and demanding tasks, each with its own unique set of tools and demands.

- **Time Management:**
  In a world where work, personal commitments, and leisure activities often coexist in the digital realm, effective time management and task scheduling have taken center stage. Users are, more than ever, in

pursuit of ways to optimize their schedules and maintain a balance between productivity and leisure, all while managing their digital lives.

- **Accessibility:**
  Digital inclusion is a moral imperative. It's important to recognize that not all users have equal access to digital resources. Physical disabilities can significantly limit one's ability to interact with traditional computer interfaces. This necessitates innovative solutions to ensure that everyone, regardless of their abilities, has access to the digital world.

- **Personalization:**
  In the realm of computing, 'one size fits all' no longer suffices. Every user brings their unique preferences, needs, and expectations to the digital table. The demand for a personalized computing experience is not just a preference; it's a necessity to ensure user satisfaction, efficiency, and the best possible user experience.

- **Integration:**
  As digital ecosystems expand, there is a growing demand for systems that can seamlessly integrate with a wide array of applications, services, and data sources.
  The problem we aim to address with this desktop assistant project is to provide a solution that alleviates these challenges. By creating an intelligent and adaptable desktop assistant, we seek to streamline information retrieval, task execution, and personalization, thereby enhancing the overall computing experience and making it more accessible to users of all backgrounds and abilities.

  This project aspires to offer a solution that goes beyond traditional digital assistants, focusing specifically on desktop computing and customization. It aims to be a valuable tool for individuals seeking to manage their digital lives more effectively, whether for work, personal use, or both.

1.3 Aim

The primary aim of this project is to design, develop, and deploy a versatile and user-centric desktop assistant, implemented in Python, that effectively addresses the challenges associated with modern computing environments. The project seeks to achieve the following key objectives:

- **Enhance User Productivity:** The desktop assistant aims to streamline tasks, automate common actions, and provide quick access to information, ultimately boosting user productivity and efficiency in various computing tasks.[

- **Simplify User Interactions:** By offering both voice and text-based commands, the assistant aims to simplify user interactions with desktop applications, making computing more accessible and intuitive.

- **Provide Personalization:** The project endeavours to create an assistant capable of learning from user interactions and adapting to individual preferences, thereby offering a personalized computing experience.

- **Offer Utility Across Diverse Domains:** The assistant is designed to be versatile, offering functionalities ranging from answering questions and managing calendars to controlling desktop applications and fetching real-time information, making it useful in various domains.

- **Leverage Python Ecosystem:** Leveraging the power of Python and its rich ecosystem of libraries, the project aims to create a robust and extensible solution that can be easily customized and expanded upon.

- **Bridge the Gap Between Users and Technology:** Ultimately, the project's aim is to bridge the gap between users and technology, making complex computing tasks more accessible to a wide range of individuals, from tech-savvy users to those with limited technical expertise.
  The successful completion of this project will result in a valuable desktop assistant that not only simplifies daily computing tasks but also contributes to a more inclusive and user-friendly digital ecosystem. It will be a tool that empowers users to harness the full potential of their computers while adapting to their unique needs and preferences

1.4 Objective

In this section, we delve into the fundamental objectives that underpin the development of our Functional Desktop Assistant. These objectives serve as guiding principles, illuminating the path to a more intuitive and user-centric digital experience.

- **Functional Desktop Assistant:**

  At the heart of this project lies the objective of crafting a fully operational desktop assistant, a digital entity with the remarkable capacity to understand and execute both voice and text commands. Our aim is to bridge the gap between users and their computers, rendering interactions seamless and intuitive.[1]

- **Feature Implementation:**

  The comprehensive implementation of core features stands as a cornerstone of our mission. These features are meticulously designed to enhance the overall user experience, and they encompass, but are not limited to:

  Provision of precise responses to user queries, tapping into the vast pool of knowledge on the internet to furnish accurate information.
  Efficient management of tasks, entailing the setting, tracking, and timely notification of scheduled events, resulting in heightened organization and punctuality.
  Real-time weather updates tailored to the user's specific location or queries, ensuring users remain well-informed.
  Streamlined organization and management of calendars and schedules, simplifying the management of personal and professional commitments.
  Seamless handling of emails through both voice and text-based commands, simplifying digital correspondence.
  Robust control and automation of desktop applications and processes, allowing users to customize and optimize their digital workspace.[1]

- **Thorough Testing and Optimization:**
  To ensure the desktop assistant is not only functional but also highly reliable, precision and performance are paramount. Extensive testing is conducted to guarantee the utmost reliability, and a commitment to continuous refinement and enhancement is part of our core philosophy. The goal is an assistant that evolves and improves over time.
- **Accessibility Integration:**

  Our commitment to inclusivity extends to individuals with disabilities. The desktop assistant is designed with accessibility features, ensuring that every user, regardless of their abilities, can utilize it with ease and inclusivity, thereby fostering a more equitable digital experience.

- **Security and Privacy Measures:**

  The trust and privacy of our users are of utmost importance. We implement stringent security protocols to protect user data and privacy, adhering to the highest standards of data security and privacy compliance. Our users can interact with the assistant with the confidence that their information is safeguarded.

- **Feedback and Enhancement:**

  User input and recommendations are integral to our development cycle. Effective feedback channels are established to collect valuable insights and suggestions. This user feedback serves as the bedrock for ongoing improvements to the assistant, enhancing its capabilities and ensuring that user satisfaction remains at the forefront.

  In conclusion, these objectives not only drive the development of our Functional Desktop Assistant but also underscore our commitment to an exceptional and user-centric digital experience. We aspire to revolutionize the way users interact with technology, making it more accessible, secure, and efficient while fostering inclusivity and continuous improvement.

1.5 Goal

In this section, we delve into the overarching objectives and ambitions that underpin the creation of this Functional Desktop Assistant project. Our aim is to outline not only what we intend to achieve but also the profound impact we anticipate it will have on users and their interactions with technology.

- **Develop a Functional Desktop Assistant:**

  At the core of this project is the development of a fully functional desktop assistant that transcends conventional paradigms. Our objective is to engineer an intelligent digital companion capable of comprehending and responding to a diverse range of voice and text commands. In essence, we aspire to create a seamless intermediary between users and their computers, simplifying digital interactions and rendering them more intuitive.[2]

- **Implement Core Features:**
  Central to our mission is the implementation of a suite of core features, each carefully designed to enhance the user experience and streamline

daily digital interactions. These features encompass, but are by no means limited to:

Answering questions by harnessing the vast expanse of knowledge available on the web.

Setting, managing, and delivering reminders for tasks and events, ensuring users stay organized and punctual.

Providing real-time weather updates, effortlessly keeping users informed about current conditions and forecasts.

Efficiently managing calendars and schedules, facilitating the juggling of personal and professional commitments.

Sending, receiving, and managing emails via voice or text commands, simplifying digital correspondence.

Exerting control over a spectrum of desktop applications and processes, offering automation and customization tailored to user preferences[2].

- **Adaptability and Personalization:**

  In line with modern AI principles, our goal is to design an assistant that transcends mere functionality. We aspire to create a digital companion that evolves with the user, adapting to their preferences and needs over time. The ability to learn from user interactions and offer a personalized experience is paramount, ensuring that the assistant becomes a true ally, one that understands and caters to the user's unique requirements.[1]

- **Utilize Python Libraries:**

  The strength of Python, with its rich ecosystem of libraries and APIs, forms the backbone of this project. We are committed to leveraging this vast toolkit to construct a desktop assistant that is robust, extensible, and versatile. Our utilization of these libraries includes integrating cutting-edge speech recognition, text-to-speech synthesis, web scraping for real-time information retrieval, and seamless interactions with external APIs. By doing so, we endeavor to endow our assistant with a wide range of functionalities that cater to diverse user needs.[1]

- **User-Friendly Interface**:

The user experience is at the forefront of our design principles. We aim to create an intuitive interface that empowers users to interact with the assistant effortlessly, irrespective of their technical expertise. This user-friendly approach ensures that the power of our assistant is accessible to all, democratizing technology and simplifying complex digital tasks.

In summary, our goals extend beyond mere technological achievement. We aspire to redefine the way individuals interact with their computers and technology, introducing a dynamic and intelligent assistant that eases the burdens of daily digital life. Our commitment is to make technology more accessible, personalized, and user-centric, ultimately forging a new era in digital assistance.

1.6 Need of System

- **Minimum System Requirements:**
  Operating System: Windows, macOS, or Linux (Ubuntu, CentOS, or other distributions).
  Processor: Dual-core or higher processor (e.g., Intel Core i3 or AMD equivalent).
  RAM: 4 GB or more for smoother performance.
  Storage: 20 GB of free disk space for the operating system and project files.
  Python: Python 3.6 or later (preferably Python 3.7+).[2]

- **Recommended System Configuration:**

  Operating System: A modern 64-bit version of Windows, macOS, or a Linux distribution like Ubuntu.
  Processor: Quad-core or higher processor (e.g., Intel Core i5 or AMD equivalent).
  RAM: 8 GB or more for improved multitasking and responsiveness.
  Storage: 50 GB or more of free disk space for project files and data storage.
  Python: Python 3.7 or later (with essential libraries like NumPy, pandas, and others, depending on your project).

  Graphics Processing Unit (GPU): While not strictly necessary for most desktop assistant projects, having a dedicated GPU may be beneficial if your project involves intensive machine learning tasks. GPUs can significantly accelerate certain computations.

Internet Connectivity: An active internet connection may be required for features that rely on fetching data from online sources, such as weather updates or web searches.

Microphone and Speakers: If your assistant has voice recognition and synthesis capabilities, you'll need a working microphone and speakers or headphones.

External Hardware: Depending on your project's features, you might need additional hardware components like a webcam or external sensors.

Virtual Environment: Consider using a virtual environment management tool like virtualenv or conda to isolate your project's dependencies and prevent conflicts with other Python installations.

Dependency Management: Use a package manager like pip or conda to install and manage the required Python packages and libraries.

Keep in mind that the system requirements may vary based on the specific functionalities and libraries you integrate into your desktop assistant. Always refer to the documentation of the libraries and tools you use for their specific system requirements and recommendations.[2]

Chapter 2. Requirement Specification

2.1 Introduction

The Desktop Assistant project is a Python-based application with the primary goal of providing users with a versatile, interactive, and highly functional assistant capable of executing a wide range of tasks through voice and text commands. To embark on this project successfully, it is imperative to have a clear understanding of the essential tools and libraries required for its development, as well as the setup procedures. This section not only outlines the prerequisites but also elucidates the development environment and the necessary libraries to ensure a seamless start to your journey with this project.[3]

- **Python:**
  Python stands as the bedrock of this project, serving as the primary programming language for crafting our Desktop Assistant. To set up Python on your system, follow these steps:

- **Download Python:**
  Visit the official Python website (https://www.python.org/downloads/) and acquire the latest version of Python suitable for your operating system, be it Windows, macOS, or Linux.

  Installation: Execute the Python installer that you downloaded and adhere to the installation instructions. During installation, ensure that you select the option to add Python to your system's PATH, which facilitates seamless command-line access.

- **Verification:**
  To confirm the successful installation of Python, open a terminal or command prompt and type 'python --version'. You should witness the installed Python version displayed as a confirmation of a job well done.

- **Visual Studio Code (VS Code):**
  Visual Studio Code serves as our integrated development environment (IDE) for tasks encompassing coding, debugging, and the overall management of the Desktop Assistant project. Here's how to set up Visual Studio Code:

- **Download VS Code**:
  Visit the official Visual Studio Code website (https://code.visualstudio.com/download) and secure the installer tailored to your operating system.
  Installation: Execute the installer you downloaded for VS Code and adhere to the installation instructions, ensuring that this powerful IDE is seamlessly integrated into your development workflow.[3]

- **Extensions:**
  To enhance your development experience, consider installing the following VS Code extensions:

- **Python Extension:**
  This indispensable extension equips you with comprehensive Python language support, debugging capabilities, and code formatting features, making it an essential companion for our project.

  By adopting these tools and adhering to these installation procedures, you're equipped with a robust development environment that facilitates a productive and efficient journey through the Desktop Assistant project. This ensures that you have the requisite foundation to create an intelligent and versatile assistant, capable of seamlessly bridging the gap between users and their digital experiences.[3]

- **Required Libraries**
  The Desktop Assistant project relies on an array of Python libraries and modules, each serving a distinct purpose to enable the functionality of this dynamic assistant. It is essential to ensure that these libraries are installed in your Python environment to facilitate a seamless development experience. Here's an overview of these indispensable libraries and modules:

- **pyttsx3**:
  This library plays a pivotal role in the project by enabling text-to-speech conversion. It empowers the assistant to communicate audibly, making the interaction more engaging and accessible.[3]

- **datetime:**

Handling date and time-related operations is made effortless with this library. It equips the assistant to manage time, schedule tasks, and perform actions that involve precise time management.[3]

- **wikipedia:**
  In the quest for knowledge, this library acts as a powerful tool. It allows the assistant to fetch information directly from the vast repository of knowledge that is Wikipedia, providing accurate and timely responses to user queries.[3]

- **webbrowser:**
  An essential component for web interactions, this library enables the assistant to open and interact with web browsers seamlessly. It's the key to accessing and utilizing web-based information and services.[3]

- **os:**
  System-level operations such as opening and closing applications are streamlined through this library. It empowers the assistant to control and manage desktop applications effectively, ensuring a smooth user experience.[3]

- **requests:**
  For fetching data from external sources, this library is indispensable. It allows the assistant to make HTTP requests, facilitating the acquisition of real-time data from diverse online platforms.[3]

- **time:**
  Time is a crucial dimension for scheduling tasks and actions. This library provides the necessary time-related functionality, ensuring that tasks are executed punctually and seamlessly.[3]

- **pyautogui:**
  Visual interactions are made possible by this library. It plays a significant role in capturing and saving screenshots, an essential feature for various tasks and user interactions[3].

- **PyQt5:**
  In case you intend to incorporate a graphical user interface (GUI) component into your assistant, PyQt5 stands as a versatile library for creating sophisticated GUIs. Its integration offers a visual

dimension to the interaction between the assistant and the user, making it more intuitive and engaging[3]
.

- **Testing and Deployment:**
  Test your application within Visual Studio code to ensure it works as expected. Once you are satisfied, you can deploy your application as a standalone Python application or package it for distribution.

Conclusion
This introduction section outlines the prerequisites and essential tools required to kickstart the Desktop Assistant project. With Python installed, Visual Studio Code set up, and the necessary libraries in place, you're well-prepared to embark on the development journey of your interactive assistant. The subsequent sections of this specification requirement document will delve into the detailed functional and non-functional requirements, data handling, and development methodologies to guide you through the project's implementation.[2]

2.2 System Environment

In the development and deployment of the Desktop Assistant project, specific software components play a pivotal role. We've outlined the essential software requirements for both the development environment and the target deployment environment to ensure the successful implementation and optimal performance of this Python-based project.

Development Environment Requirements
For the development and testing of the Desktop Assistant, you will need a computer equipped with the following software components:

- **Operating System Versatility:**
  Your choice of operating system for development can be Windows, macOS, or Linux. However, it's imperative to ensure that your selected operating system is fully compatible with the development tools and libraries utilized throughout the project. This compatibility is crucial for seamless development and testing processes.

- **Python - The Powerhouse:**

Python, the programming language at the heart of this project, is the backbone. It's essential to install the latest version of Python that is compatible with your operating system. Moreover, during installation, be sure to add Python to your system's PATH. This simple step will facilitate a smooth and hassle-free development experience.

- **Visual Studio Code (VS Code):**
  Visual Studio Code stands as the integrated development environment (IDE) of choice for coding and managing the Desktop Assistant project. To get started, download and install VS Code. This powerful IDE provides an array of features that enhance your development workflow and make managing the project a breeze.

- **VS Code Extensions for Python:**
  Enhance your Python development experience further by installing the Python extension for VS Code. This extension equips you with language support, debugging capabilities, and code formatting features, making your development process more efficient and error-free.

- **Target Deployment Environment Requirements:**
  When preparing to deploy the Desktop Assistant for end-users, it's vital to consider the following software requirements to ensure that it runs seamlessly on their computers:

- **Operating System Compatibility:**
  The Desktop Assistant should seamlessly integrate with various operating systems, including different versions of Windows (7, 8, 10), macOS, and diverse Linux distributions. Ensuring compatibility with these systems is paramount to guarantee a broad user base.

- **Python Runtime Prerequisite:**
  To guarantee a smooth user experience, it's essential to ensure that the user's system has Python installed. This Python version must be compatible with the specific Desktop Assistant version you plan to distribute.

General Recommendations for Success

In addition to the software requirements, here are some general recommendations to further ensure the Desktop Assistant's success:

- **Internet Connectivity:** Make sure the target deployment environment maintains an active internet connection. This connection is vital for accessing online resources and services that may be integral to the functionality of the Desktop Assistant.

- **Regular Updates:** Regularly update and maintain the operating system and installed software on both the development and deployment environments. This practice enhances security and performance while keeping the project up-to-date.

- **Clear Installation Instructions:** Consider providing clear and concise installation instructions and system requirement guidelines to end-users. This thoughtful gesture can significantly contribute to a hassle-free setup and user satisfaction.

By diligently adhering to these software requirements and general recommendations, you can be confident that the Desktop Assistant will operate efficiently in both the development and target deployment environments, offering a seamless and satisfying user experience.

2.3 Software Requirements

Python
Visual Studio Code

2.4 Hardware Requirements

The Desktop Assistant project, meticulously crafted using Python, relies on a combination of hardware components to deliver its array of functionalities. Here, we delineate the critical hardware requirements for both the development environment and the target deployment environment. These requirements are the cornerstone of ensuring that the Desktop Assistant runs seamlessly and offers a delightful user experience.

Development Environment Hardware Requirements

To embark on the journey of developing and testing the Desktop Assistant, your development computer should be equipped with the following hardware components:

- **Processor (CPU) Power:**
  A multi-core processor, with a clock speed of at least 1.5 GHz or faster, is strongly recommended for a development environment that operates

smoothly. The CPU is the heartbeat of your system during development and testing, and a robust one is invaluable.

- **Memory (RAM) Matters:**
  For essential development operations involving running development tools, the Python interpreter, and the Desktop Assistant application, a minimum of 2 GB of RAM is required. However, for a more responsive and efficient development experience, it's advisable to have 4 GB or more of RAM at your disposal.

- **Storage Space: The Repository**
  Allocating at least 2.5 GB of free disk space on your development computer is essential. This space is instrumental for accommodating development tools, project files, and the Python environment. Adequate storage ensures a clutter-free environment for your project to thrive.

Target Deployment Environment Hardware Requirements

Transitioning from development to deployment, certain hardware prerequisites must be met for the target computer to deliver responsive performance to end-users:

- **Processor (CPU) Performance**
  The target computer should boast a CPU with a clock speed of at least 1 GHz or faster. This ensures that the Desktop Assistant operates responsively, providing users with the swift performance they expect.

- **Memory (RAM) Recommendations:**
  For running the Desktop Assistant application, a minimum of 512 MB of RAM is recommended. However, depending on the complexity of your assistant and its functionalities, additional RAM may be required to guarantee a smooth and satisfying user experience. Be mindful of the memory needs of your specific project.

- **Adaptive Screen Resolution:**
  The Desktop Assistant's user interface should be adaptive, catering to different screen resolutions. This adaptability guarantees a consistent user experience across an array of devices, ensuring that your assistant feels at home on any screen.

General Recommendations for Success

In addition to these hardware requirements, here are some general recommendations to further guarantee the success of the Desktop Assistant:

- **Well-Maintained Environment:** Ensure that the target deployment environment is well-maintained, with all hardware components in good working condition. A well-kept system is more likely to provide consistent and reliable performance.

- **Resource Monitoring**: Monitor system resources diligently to ensure that the Desktop Assistant operates without hiccups. This practice helps in identifying and mitigating potential issues, preventing system slowdowns or crashes.

- **Clear System Requirement Guidelines:** Consider providing clear and concise system requirement guidelines to end-users, especially if your assistant targets a specific user base with varying hardware configurations. This thoughtful gesture can make the installation and setup process a breeze for your users.

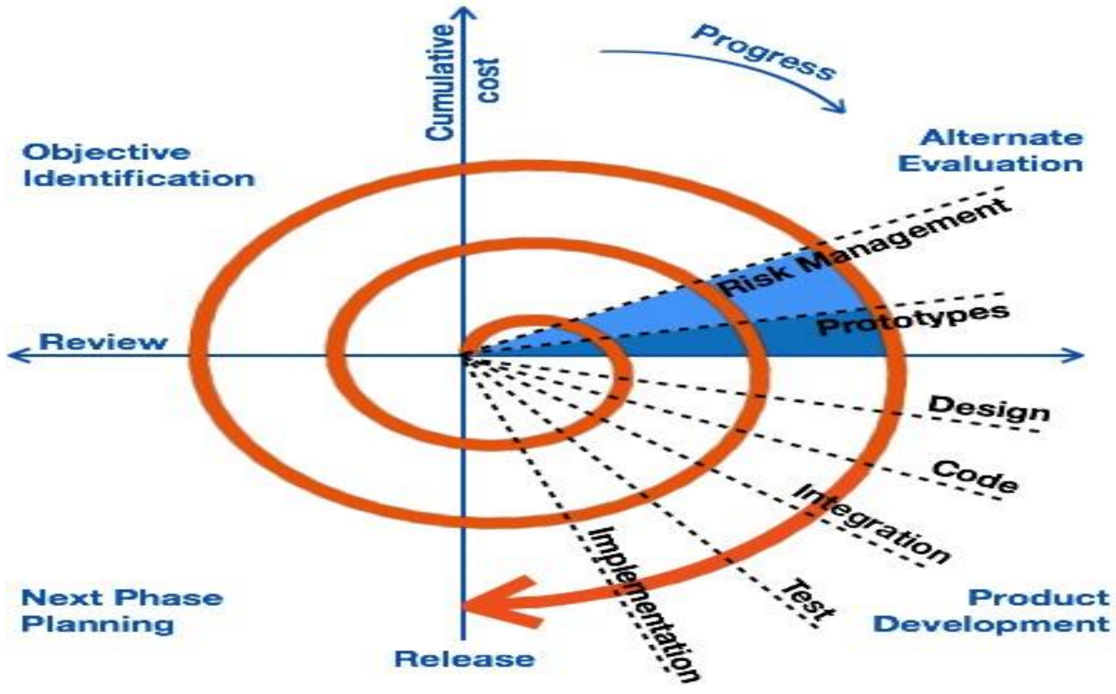
By meticulously adhering to these hardware requirements and embracing the general recommendations, you can ensure that the Desktop Assistant operates efficiently and provides a satisfactory user experience in both the development and target deployment environments. This attention to detail and performance optimization is key to the success of your project.

2.5 Methodology

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models. Each process model follows a Series of steps unique to its type to ensure success in the process of software development. Following are the most important and popular[4]

Spiral Model:

Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.[4]



Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1. **Objectives determination and identify alternative solutions:**
   - Requirements are gathered from the customers and the objectives are identified, elaborated, and analysed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

2. **Identify and resolve Risks:**
   - During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy.

3. **Develop next version of the Product:**
   - During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

4. **Review and plan for the next Phase:**

- In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Why Spiral Model?

The Spiral model is called a Meta-Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the Iterative Waterfall Model. The spiral model incorporates the stepwise approach of the Classical Waterfall Model. The spiral model uses the approach of the Prototyping Model by building a prototype at the start of each phase as a risk-handling technique. Also, the spiral model can be considered as supporting the Evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built. [4]

Spiral Model Application

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e., learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model –

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback. Significant canges are expected in the product during the development cycle[4].

Chapter 3. System Analysis

3.1 Introduction

The approach of systems analysis in the context of the Desktop Assistant project is fundamentally different from traditional forms of analysis. Traditional analysis often focuses on breaking down complex systems into individual components. In contrast, systems analysis, in this project, emphasizes how various components and functionalities of the Desktop Assistant interact to deliver a seamless user experience.

Rather than isolating smaller parts of the system, systems thinking expands its view to consider larger and larger numbers of interactions within the assistant. This holistic approach is particularly important when dealing with a multifaceted project like the Desktop Assistant, as it allows us to better understand the intricate relationships between its components and functionalities

3.2 System Analysis

System analysis is the comprehensive study of sets of interacting entities within the context of the Desktop Assistant project. It encompasses various aspects, including understanding user requirements, assessing existing functionalities, and identifying areas for improvement. System analysis serves as a crucial foundation for designing and building an efficient and user-friendly assistant.

3.2.1 Analysis of Existing System

Before delving into the design of the proposed Desktop Assistant, it is essential to analyze the shortcomings of the existing system. This analysis not only helps us understand the limitations of the current system but also guides us in developing a solution that addresses these issues effectively.

Some of the issues associated with the existing system include:

Lack of Monitoring: The current system lacks a monitoring mechanism, which can lead to various challenges, especially when it comes to user interactions. Without adequate monitoring, it becomes challenging to ensure a seamless user experience.

Safety Concerns: The absence of a safety mechanism within the existing system raises concerns about user security and protection. Implementing robust safety features is essential to enhance user confidence.

Manual Alerts: In the absence of automated alert mechanisms, users are required to perform manual actions to trigger alerts or responses. This manual process can be cumbersome and inefficient.

Through systematic analysis, we intend to identify areas for improvement and optimization, ensuring that the Desktop Assistant not only meets but exceeds user expectations. This analysis phase is a critical step toward building a robust and user-centric assistant.

3.3 Analysis of Proposed System

The primary objective of our Desktop Assistant project is to create a versatile digital assistant designed to assist users efficiently. In the early stages of system planning, we strategized to evolve our ideas and gain a comprehensive understanding of how our system would provide essential functionalities without compromising on user experience.

3.3.1 Core Objectives

Our main goal in developing this Desktop Assistant is to provide users with a valuable digital companion that enhances their daily tasks and interactions. We aim to implement the following key functionalities:

3.3.1.1 Voice Interaction

The assistant should be able to understand and respond to user voice commands effectively. This includes tasks such as opening applications, searching the web, and performing system operations.

3.3.1.2 Information Retrieval

Users should be able to access information quickly and effortlessly. The assistant will be equipped with features to provide information, answer questions, and fetch data from reliable sources like Wikipedia.

3.3.1.3 Web Browsing

The assistant should have the capability to browse the web, open websites, and retrieve information as per user requests. This feature enhances the user's ability to access online content conveniently.

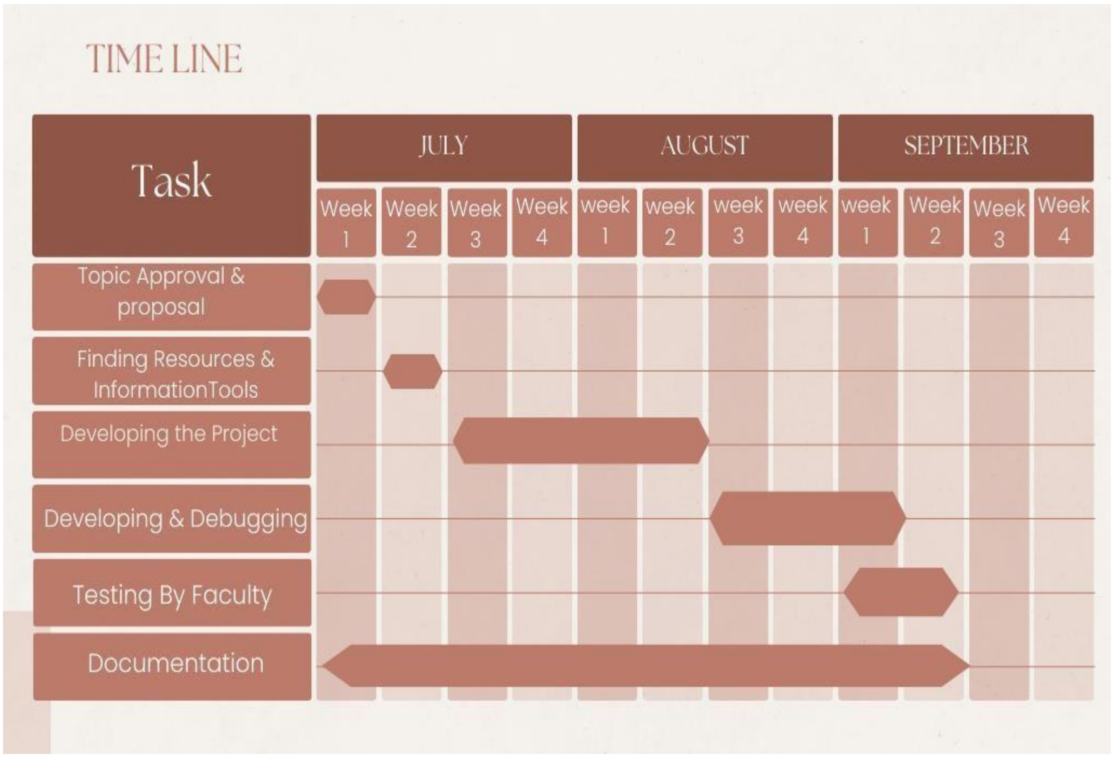
3.3.1.4 Application Management

Users should be able to open and close various applications on their desktop through voice commands. This functionality aims to simplify the user's interaction with their computer.

3.3.1.5 Communication

The assistant will facilitate communication by allowing users to send messages through platforms like WhatsApp. It will also provide features for sending SMS and making calls.

3.3 Timeline Chart:



Chapter 4. Survey of Technology

4.1 Python

Python, a high-level and interpreted programming language, stands as the cornerstone of the Desktop Assistant project. Its reputation for simplicity and readability has made it a preferred choice for developers across the globe. Python is engineered with minimal implementation dependencies, granting it unparalleled versatility to cater to a wide array of applications. What sets Python apart is its "write once, run anywhere" philosophy, enabling code to execute seamlessly on different platforms without the need for recompilation. While it shares certain similarities with languages like C and C++, Python offers higher-level abstractions, making it exceedingly user-friendly.[3]

Python's Popularity and Ubiquity:
Python has soared in popularity, amassing millions of dedicated developers worldwide. Its adoption spans diverse domains, including desktop applications, web development, data analysis, machine learning, and scientific computing. The language's ubiquity can be attributed to its accessible syntax, extensive library support, and a vibrant community that continually enriches its ecosystem..

Why Python?

Python was chosen as the primary programming language for the Desktop Assistant project for several compelling reasons:

Simplicity and Readability: Python's simplicity and readability align perfectly with the goal of creating a user-friendly assistant. Its clean and intuitive syntax makes it approachable for developers of all levels.

Versatility: Python's versatility empowers us to implement a wide range of features seamlessly. From voice recognition to web browsing and system control, Python's adaptability ensures that the Desktop Assistant is well-equipped to meet diverse user needs.[5]

Key Python Libraries Empowering the Desktop Assistant:

pyttsx3 Library: The pyttsx3 library plays a pivotal role in the project by facilitating text-to-speech conversion. This enables the Desktop Assistant to convert text into spoken words, utilizing various speech engines to provide natural-sounding voice output. It's a fundamental component for enabling the assistant to communicate with users through speech.[3]

datetime Module: In the world of time-related functionalities, Python's datetime module shines. It equips the Desktop Assistant with the ability to manage dates and times effectively. This is instrumental in tasks such as retrieving the current time and scheduling actions, ensuring that time-related responses are accurate.[3]

Web Browser Module: The web browser module in Python empowers the Desktop Assistant to interact with web browsers, opening the door to actions like launching websites and conducting web searches. This module is fundamental for providing web-related functionalities, enabling users to access online resources and information with ease.[3]

os Module: Python's os module is a valuable asset, providing functions for interacting with the operating system. It's instrumental in tasks such as opening and closing applications and executing system commands. The os module facilitates seamless interaction with the underlying operating system, enabling the assistant to perform system-related actions and manage applications effectively.[3]

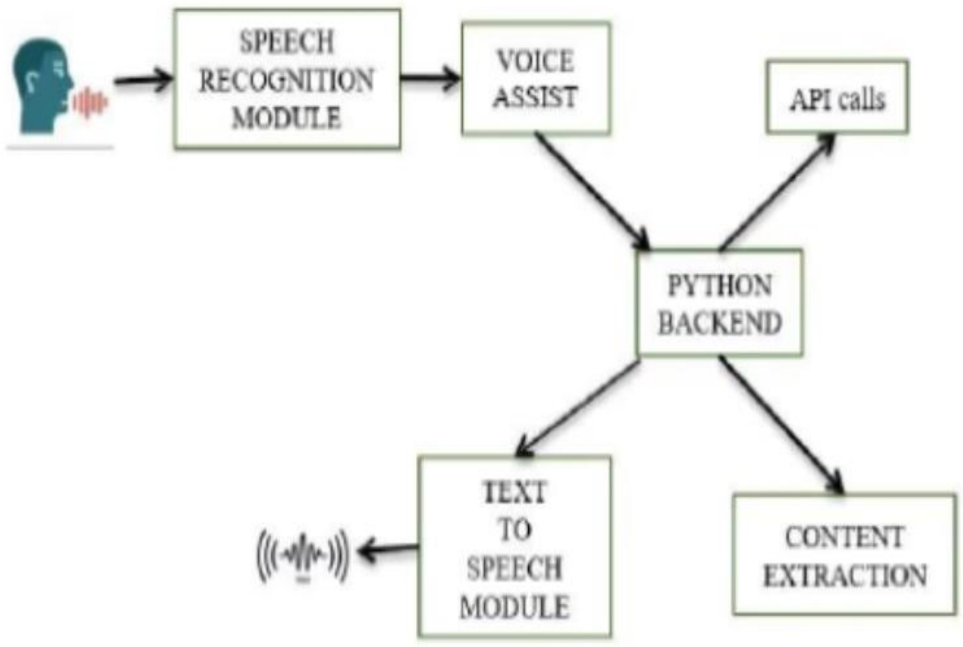
Requests Library: The requests library in Python is a vital component for making HTTP requests to external resources. It serves essential functions such as retrieving the user's IP address from a web service. This library enriches the assistant's capabilities, enhancing its ability to fetch data from external sources.[3]

In the forthcoming chapters, we will delve deeper into the integration of these modules and libraries within the Desktop Assistant, showcasing how they collectively contribute to creating a feature-rich and user-friendly application that caters to a multitude of user needs and preferences. Python, with its elegance and robust ecosystem, serves as the bedrock upon which this innovative assistant t

Chapter 5. System Design

1. Introduction

The Desktop Assistant is a voice-activated virtual assistant designed to provide users with a seamless and interactive experience while performing various tasks on their desktop computers. This system design document outlines the architecture, modules, data flow, user interface, security measures, performance considerations, error handling, integration with external services, and potential future enhancements of the Desktop Assistant.

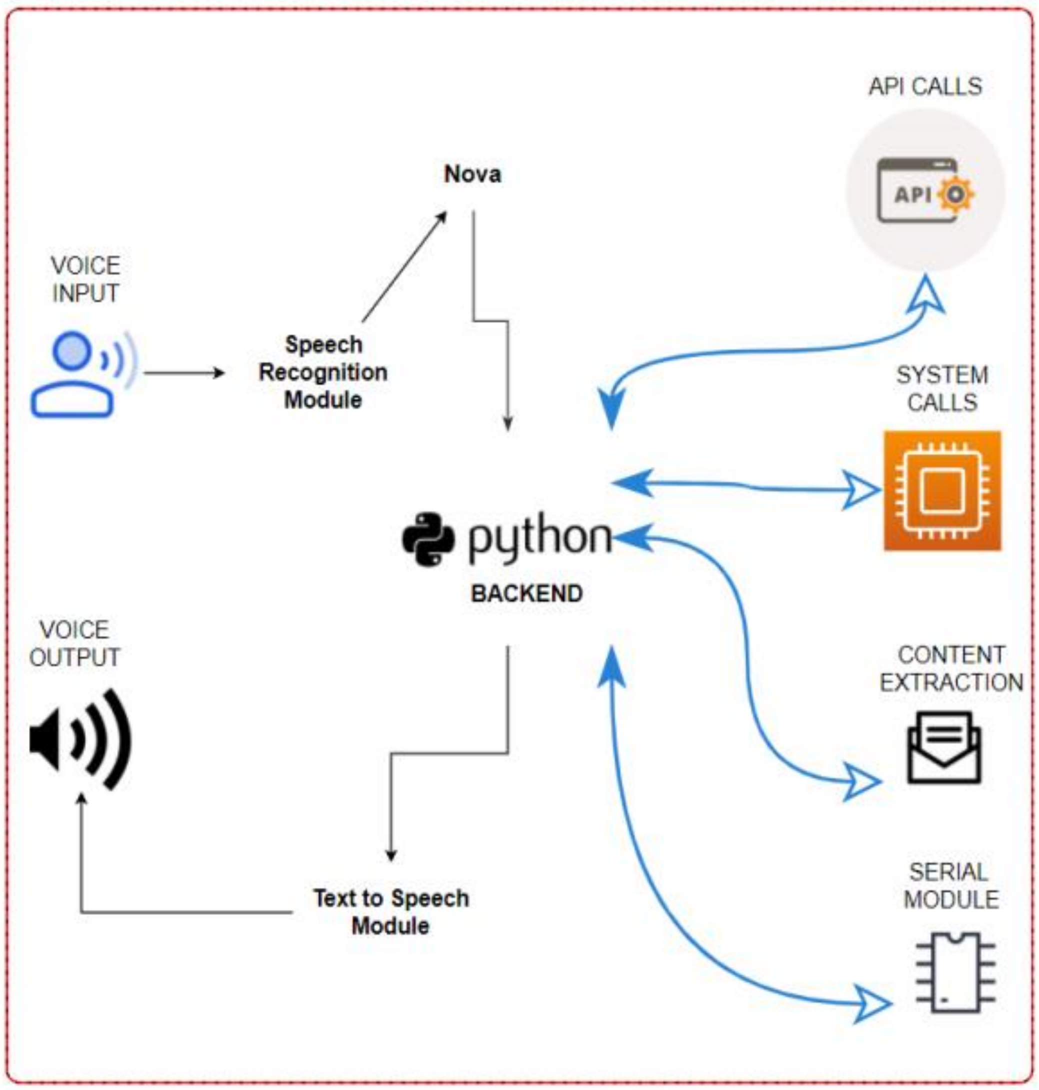


5.2 System Architecture Diagram

The Desktop Assistant is built using a client-server architecture, where the client component runs on the user's desktop computer, and the server component manages tasks, processes voice commands, and interacts with external services. The architecture consists of the following key components:

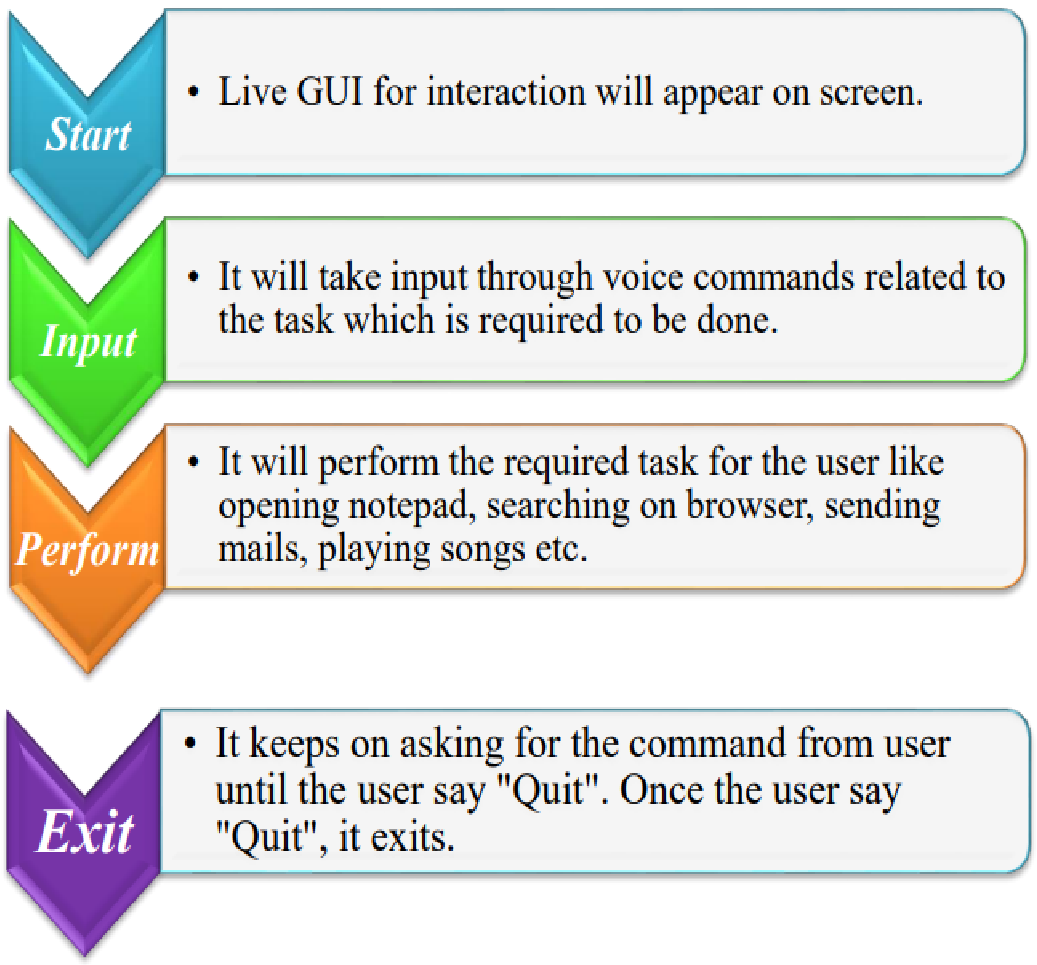
Client Component: This component resides on the user's desktop computer and is responsible for capturing voice commands, processing user input, and displaying responses through the user interface.[2]

Server Component: The server component manages voice recognition, natural language processing, and interaction with external services. It communicates with the client component through API calls and serves as the brain of the Desktop Assistant.[2]



5.3 Data Flow

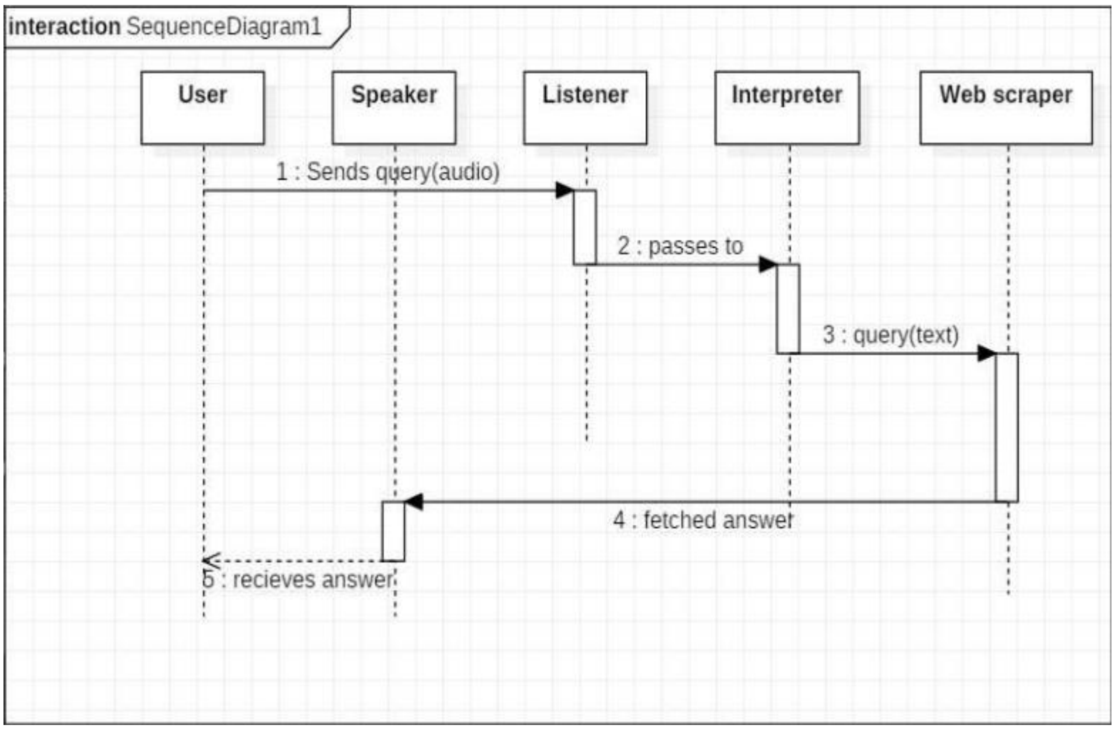
The data flow within the Desktop Assistant follows these steps:
Voice commands are captured by the client component using a microphone.
The voice recognition module converts spoken words into text.
The NLP module analyzes the text to determine user intent and extract relevant details.
The appropriate command execution module is triggered based on user intent.
The client component displays responses and information through the user interface.
If required, external services are accessed for data retrieval or actions like web browsing and online searches.[1]


Start
• Live GUI for interaction will appear on screen.
Input
• It will take input through voice commands related to the task which is required to be done.
Perform
• It will perform the required task for the user like opening notepad, searching on browser, sending mails, playing songs etc.
Exit
• It keeps on asking for the command from user until the user say "Quit". Once the user say "Quit", it exits.

DATA FLOW DAIGRAM[1]

The data in this project is nothing but user input, whatever the user says, the assistant performs the task accordingly. The user input is nothing specific but the list of tasks which a user wants to get performed in human language i.e. English.[1]

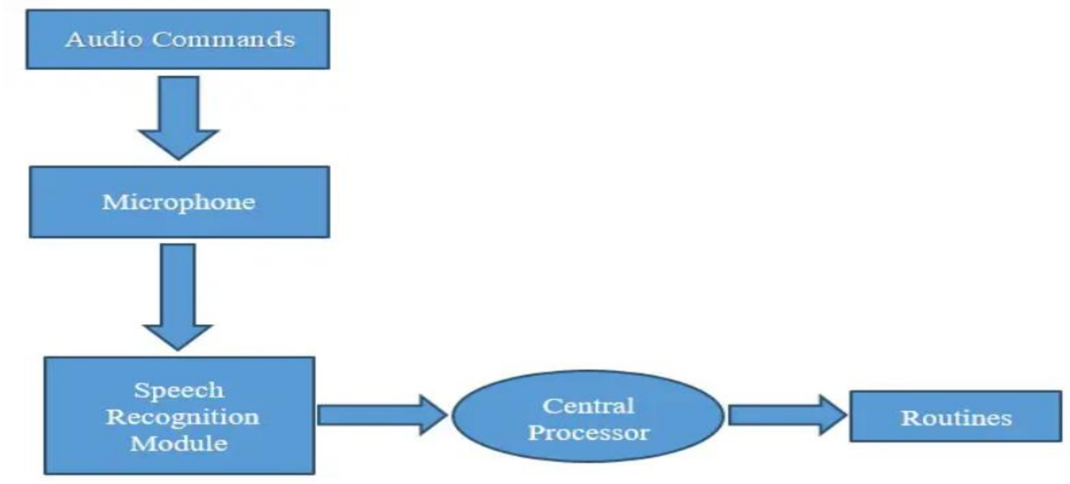
5.4 sequence diagram:



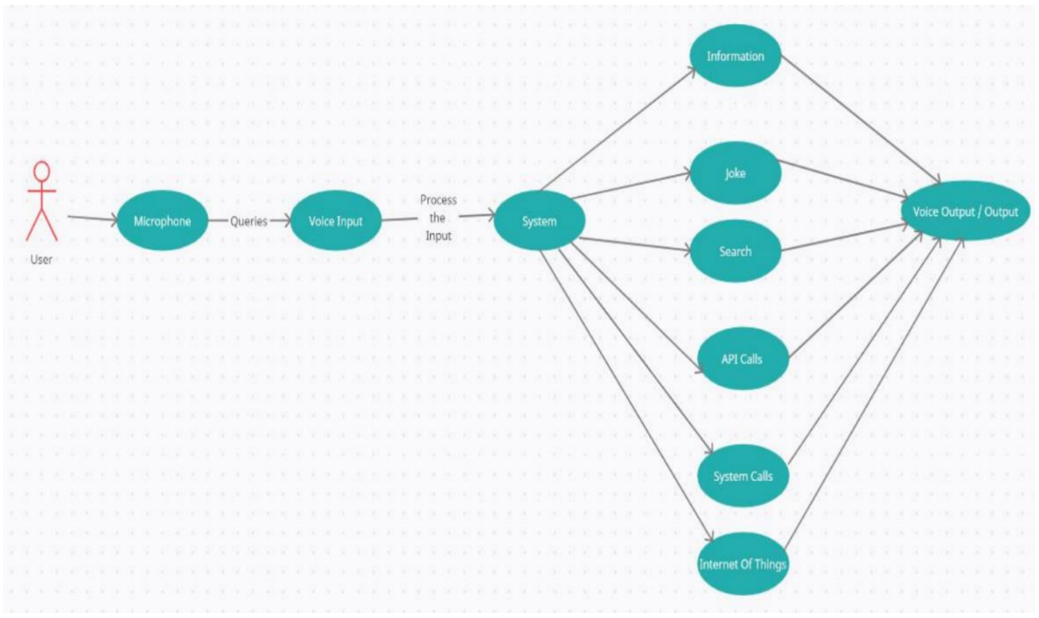
Sequence Daigram[2]

The above sequence diagram shows how an answer asked by the user is being fetched from internet. The audio query is interpreted and sent to Web scraper. The web scraper searches and finds the answer. It is then sent back to speaker, where it speaks the answer to user.

5.5 ER Daigram:



5.5 USE CASE DAIGRAM:



USE CASE DAIGRAM[2]

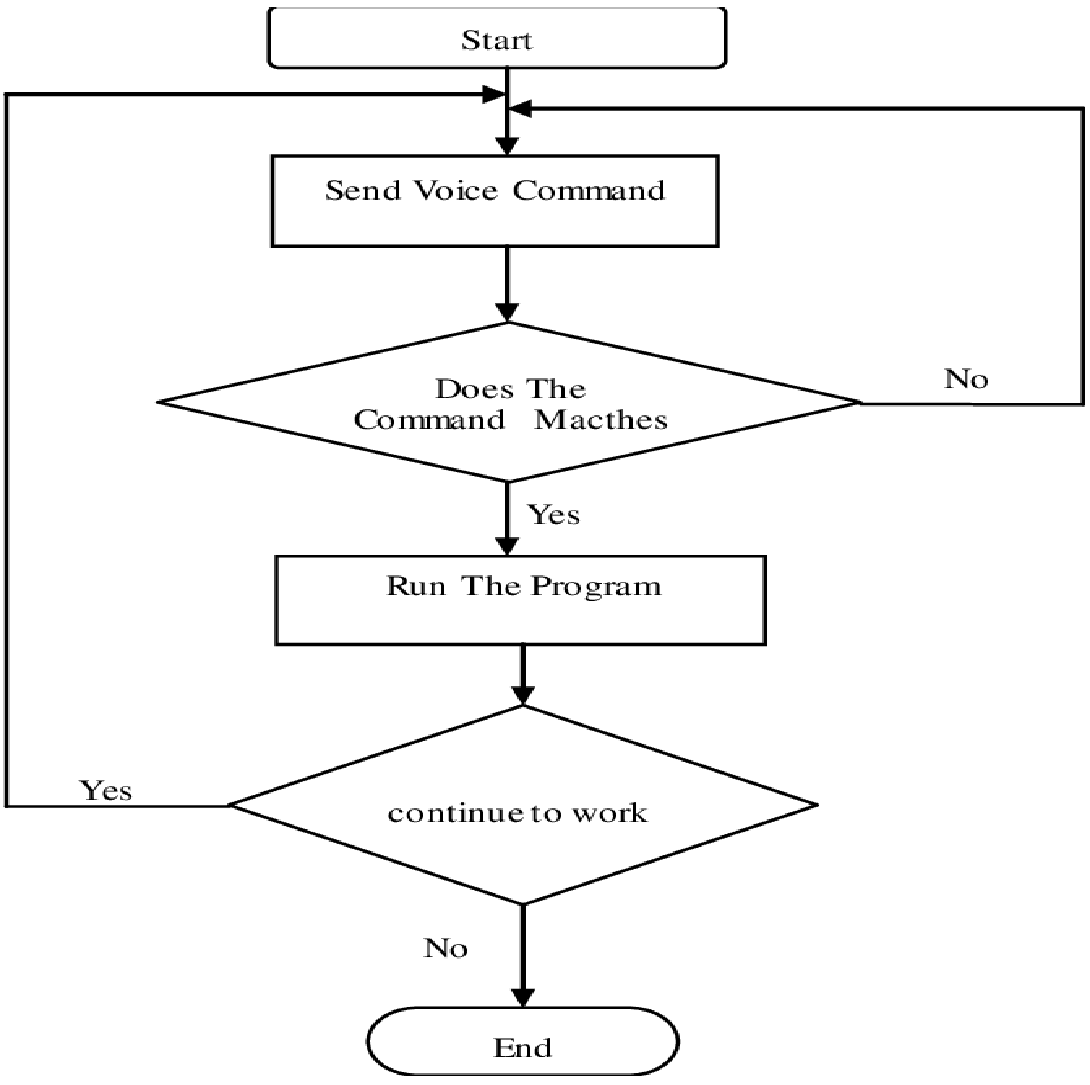
In this project there is only one user. The user queries command to the system. System then interprets it and fetches answer. The response is sent back to the user[2]

Chapter 6. System Implementation

6.1 Introduction

Project implementation is the process of putting a project plan into action to produce the deliverables, otherwise known as the products or services, for clients or stakeholders. It takes place after the planning phase, during which a team determines the key objectives for the project, as well as the timeline and budget. Implementation involves coordinating resources and measuring performance to ensure the project remains within its expected scope and budget. It also involves handling any unforeseen issues in a way that keeps a project running smoothly.

6.2 Flowchart



6.3 Coding

Main.py:

```
from asyncio import QueueEmpty
import imp
from logging import shutdown
from msvcrt import kbhit
from turtle import speed
from email.message import EmailMessage
from tkinter import Scale
import typing
from PyQt5.QtWidgets import QWidget
from numpy import safe_eval
import pyttsx3
import speech_recognition as sr
import datetime
import wikipedia
import cv2
import json
import numpy as np
import pyautogui as p
import pygame
import pywhatkit as kit
import requests
import sys
import pyjokes
import os.path
import smtplib
import webbrowser
import os
import smtplib

from PyQt5 import QtWidgets, QtCore ,QtGui
from PyQt5.QtCore import QTimer ,QTime , QDate,Qt
from PyQt5.QtGui import QMovie
from PyQt5.QtCore import *
from PyQt5.QtGui  import *
from PyQt5.QtWidgets import *
from PyQt5.uic import loadUiType
from albargui import Ui_albargui
```

```
hour_now = int(datetime.datetime.now().strftime("%H"))
minute_now =int(datetime.datetime.now().strftime("%M"))



'''recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainner/trainner.yml')
cacadePath ="haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cacadePath)

font = cv2.FONT_HERSHEY_SIMPLEX

id =2
names =['','sufiyan']
cam = cv2.VideoCapture(0,cv2.CAP_DSHOW)
cam.set(3, 640)
cam.set(4, 480)

minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)

while True:
    ret ,img = cam.read()

    converted_image = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        converted_image,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(minW),int(minH)),
    )
    for(x,y,w,h) in faces:
        cv2.rectangle(img,(x,y), (x+w,y+h), (0,255,0),2)
        id,accuracy = recognizer.predict(converted_image[y:y+h,x:x+w])

        if (accuracy < 100):
            id = names[id]
            accuracy = "{0}%".format(round(100 - accuracy))
            #TaskExecution()
        else:
            id = "unknow"
            accuracy="{0}%".format(round(100 - accuracy))

        cv2.putText(img, str(id), (x+5,y-5), font, 1, (255,255,255), 2)
        cv2.putText(img, str(accuracy), (x+5,y+h-5), font, 1, (255,255,0), 1)

    cv2.imshow('camera',img)
    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break


print("Thanks for using this program.")
cam.release()
cv2.destroyAllWindows()'''


def TaskExecution():
    wishMe()
    if 1:

        p.press('esc')
        Speak("verification sucessful")
        Speak("Welcome to Sufiyan sir")


engine = pyttsx3.init('sapi5')
voices = engine.getProperty('voices')
# print(voices[0].id)
engine.setProperty('voice', voices[1].id)


def Speak(audio):
    engine.say(audio)
    engine.runAndWait()


def wishMe():
    hour = int(datetime.datetime.now().hour)
    if hour >= 0 and hour < 12:
        Speak("Good Morning!")

    elif hour >= 12 and hour < 18:
        Speak("Good Afternoon")

    else:
        Speak("Good Evening!")


    #Speak("verification sucessful")
    Speak("Welcome to Sufiyan sir")
    Speak("Iam Javris maam. Please tell me how may help you")


class MainThread(QThread):
    def __init__(self):
        super(MainThread,self).__init__()

    def run(self):
        self.TaskExecution()


    def takeCommand(self):
        # It Take microphone input from the user and return string output

        r = sr.Recognizer()
        with sr.Microphone() as source:
            print("Listening...")
            r.pause_threshold = 0.8
            audio = r.listen(source)

        try:
            print("Recognizing...")
            query = r.recognize_google(audio, language='en-in')
            print(f"User said:{query}\n")

        except Exception as e:
            # print(e)

            print("Say that again please...")
            return "None"
        return query


    def sendEmail(to, content):
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.ehlo()
        server.starttls()
        server.login('shaikhsufiyan0045@gmail.com', 'Sufiyan@123')
        server.sendmail('shaikhsufiyan0045@gmail.com', to, content)
        server.close()


    def TaskExecution(self):
        wishMe()
        while True:

          self. query = self.takeCommand().lower()
        # login for executed taks
          if 'wikipedia' in self.query:
              Speak('Searching wikipedia...')
              self.query = self.query.replace("wikipedia", "")
              results = wikipedia.summary(self.query, sentences=2)
              Speak("According to wikipedia")
              print(results)
              Speak(results)

          elif  'how are you' in self.query:
              Speak("I am Fine sir,How can i help you")

          elif  'hello' in self.query:
              Speak("Hello sir")

          elif  "What are you doing" in self.query:
              Speak("Iam a waiting for command")

          elif "open command" in self.query:
              os.system("start cmd")

          elif 'open youtube' in self.query:
              webbrowser.open("youtube.com")
              Speak("opening youtube")
```

```
        elif 'close youtube' in self.query:
            Speak('yes sir.closing youtube')
            os.system('taskkill /f /im msedge.exe')

        elif 'tell me joke' in self.query:
            joke = pyjokes.get_joke()
            Speak(joke)

         # elif 'open google' in query:
         #     webbrowser.open("google.com")

        elif 'open google' in self.query:
            Speak("sir .what should i open in google")
            cm=self.takeCommand().lower()
            webbrowser.open(cm)

        elif 'close google' in self.query:
            Speak('yes sir.closing google')
            os.system('taskkill /f /im msedge.exe')


         #below code is for sending message from whatsapp by using
'pywhatkit' and 'web browser module'


        elif 'send message shadab bhai' in self.query:
            Speak("what should i send")
            minute=minute_now+2
            speaking=self.takeCommand().lower()

kit.sendwhatmsg("+919820038805",f"{speaking}",hour_now,minute)

        elif 'open stackoverflow' in self.query:
            webbrowser.open("open stackoverflow.com")

        elif 'open firefox' in self.query:
            webbrowser.open("open firefox")

        elif 'open visual studio' in self.query:
```

```
codePath="C:\\ProgramFiles\MicrosoftVisualStudio\\2022\Community\\Co
mmon7\IDE\\devenv.exe"
          os.startfile(codePath)

      elif 'close visual studio' in self.query:
          Speak("okay sir . closing visual studio")
          os.system("taskkill /f /im visual studio.exe")

      elif 'open data' in self.query:
          codePath      =      "C:\\ProgramData\\Microsoft\\Windows\\Start
Menu\\Programs\\MySQL"
          os.startfile(codePath)

      elif "camera" in self.query or "take a photo" in self.query:
          os.capture(0, "Jarvis Camera ", "img.jpg")

      elif 'open idea' in self.query:
          idcode = "C:\\Users\\ADMIN\\Downloads"
          os.startfile(idcode)

      elif 'open Database' in self.query:
          ddpath        =        "C:\\ProgramData\\Microsoft\\Windows\\Start
Menu\\Programs"
          os.startfile(ddpath)

       #elif "restart" in query:
          #subprocess.call(["shutdown", "/r"])

      elif 'open powerpoint' in self.query:
          pptpath       =       "C:\\ProgramData\\Microsoft\\Windows\\Start
Menu\\Programs"
          os.startfile(pptpath)

      elif 'open microsoft' in self.query:
          wordpath      =      "C:\\ProgramData\\Microsoft\\Windows\\Start
Menu\\Programs"
          os.startfile(wordpath)

      elif 'play music' in self.query:
          music_dir = 'E:\\Aiassitant'
          songs = os.listdir(music_dir)
```

```
            print(songs)
            os.startfile(os.path.join(music_dir,songs[0]))

        elif 'what is time' in self.query:
            strTime = datetime.datetime.now().strftime("%H:%M:%S")
            Speak(f"sir The time is:{strTime}")

        elif 'open studio' in self.query:
            codePath                                                    =
"C:\\Users\\ADMIN\\AppData\\Local\\Programs\\Microsoft                    VS
Code\\Code.exe"
            os.startfile(codePath)

        elif 'close studio' in self.query:
            Speak("okay sir Closing code")
            os.system("taskkill /f /im code.exe")

        elif 'set alarm' in self.query:
            nn = int(datetime.datetime.now().hour)
            if nn==22:
                music_dir = "E:\\music"
                songs = os.listdir(music_dir)
                os.startfile(os.path.join(music_dir,songs[0]))

        elif 'open Excel' in self.query:
            codePath       =       "C:\\ProgramData\\Microsoft\\Windows\\Start
Menu\\Programs"
            os.startfile(codePath)

        elif 'shut down the system' in self.query:
            os.system("shutdown /s /t 5")

        elif 'restart the system' in self.query:
            os.system("shutdown /s /t 5")

        elif 'sleep the system' in self.query:
            os.system("rundl132.exe powrprof.dil,SetSuspendState 0,1,0")

        elif 'close microsoft' in self.query:
            Speak("Thanks for using me sir, have a good day")
            os.system("taskkill /f /im microsoft.exe")
        elif 'you can sleep ' in self.query:
            Speak("Thank for using me sir, have good day")
            sys.exit()

        elif "shutdown system " in self.query:
            Speak("Are You sure want to shutdown")
            shutdown = input("Do you wish to shutdown your computer?
(yes/no}")
            if shutdown == "yes":
                os.system("shutdown /s /t 1")

        elif 'Search on google' in self.query:
                Speak('What do you want to search on Google, sir?')

        elif "send whatsapp message" in self.query:
                Speak('On what number should I send the message sir? Please
enter in the console: ')
                number = input("Enter the number: ")
                Speak("What is the message sir?")
                # message = take_user_input().lower()
                # send_whatsapp_message(number, message)
                Speak("I've sent the message sir.")

        # elif "play a game" in query:cls
        #     from game import game_play
        #     game_play()

        elif 'ip address' in self.query:
            from requests import get
            ip = get('https://api.ipify.org').text
            Speak(f"your IP address is {ip}")

        # elif 'open' in query:
        #     query = query.replace("open","")
        #     query = query.replace("javis","")
        #     #pyautogui.press("super")
        #     pyautogui.typewrite(query)
        #     pyautogui.sleep(2)
        #     pyautogui.press("enter")

        elif 'screenshot' in self.query:
            import pyautogui
            im = pyautogui.screenshot()
            im.save("tt.jpg")
        elif 'click my photo' in self.query:
            pyautogui.press("enter")
            #pyautogui.press("super")
            pyautogui.typewrite("camera")
            pyautogui.press("enter")
            pyautogui.sleep(2)
            Speak("SMILE")
            pyautogui.press("enter")

        elif 'send to email' in self.query:
            try:
                Speak("what should I say")
                content = self.takeCommand()
                to = "shaikhsufiyan0045@gmail.com"
                self.sendEmail(to, content)
                Speak("Email has been sent!")
            except Exception as e:
                print(e)
                Speak("sorry my friend sufiyan bhai.Iam not able to send this
email")



startExecution = MainThread()
class Main(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_albargui()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.startTask)
        self.ui.pushButton_2.clicked.connect(self.close)

    def startTask(self):
        self.ui.movie                                          =
QtGui.QMovie("C:/Users/ADMIN/Downloads/7LP8.gif")
        self.ui.label.setMovie(self.ui.movie)
        self.ui.movie.start()
        startExecution.start()
```

```
app = QApplication(sys.argv)
jarvis = Main()
jarvis.show()
exit(app.exec_())
# akakakakakakaka
```

albargui.py:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'albargui.ui'
#
# Created by: PyQt5 UI code generator 5.15.9
#
# WARNING: Any manual changes made to this file will be lost when pyuic5
is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_albargui(object):
    def setupUi(self, albargui):
        albargui.setObjectName("albargui")
        albargui.resize(836, 595)
        self.centralwidget = QtWidgets.QWidget(albargui)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(10, 0, 811, 561))
        self.label.setText("")

self.label.setPixmap(QtGui.QPixmap("C:/Users/ADMIN/Downloads/7LP8.
gif"))
        self.label.setScaledContents(True)
        self.label.setObjectName("label")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(530, 450, 75, 23))
        self.pushButton.setStyleSheet("\n"
"background-color: rgb(0, 85, 127);")
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(660, 450, 75, 23))
        self.pushButton_2.setStyleSheet("\n"
"background-color: rgb(0, 85, 127);")
        self.pushButton_2.setObjectName("pushButton_2")
        albargui.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(albargui)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 836, 21))
        self.menubar.setObjectName("menubar")
        albargui.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(albargui)
        self.statusbar.setObjectName("statusbar")
        albargui.setStatusBar(self.statusbar)

        self.retranslateUi(albargui)
        QtCore.QMetaObject.connectSlotsByName(albargui)

    def retranslateUi(self, albargui):
        _translate = QtCore.QCoreApplication.translate
        albargui.setWindowTitle(_translate("albargui", "MainWindow"))
        self.pushButton.setText(_translate("albargui", "RUN"))
        self.pushButton_2.setText(_translate("albargui", "TERMINATE"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    albargui = QtWidgets.QMainWindow()
    ui = Ui_albargui()
    ui.setupUi(albargui)
    albargui.show()
    sys.exit(app.exec_())
```

6.4 Testing Approach

We used different testing approach to test our application like unit testing. Usability testing and Security testing.

The system testing is done on fully integrated system to check whether the requirements are matching or not. The system testing for Desktop assistant focuses on the following four parameters:

6.5 Test Cases:

- **FUNCTIONALITY:**
  In this we check the functionality of the system whether the system performs the task which it was intended to do. To check the functionality each function was checked and run, if it is able to execute the required task correctly then the system passes in that particular functionality test. For example to check whether Desktop assistant can search on Google or not, user said "Open Google", the assistant asked, "What should I search on Google?" then user said, "What is Python",Desktop assistant open Google and searched for the required input.[2]

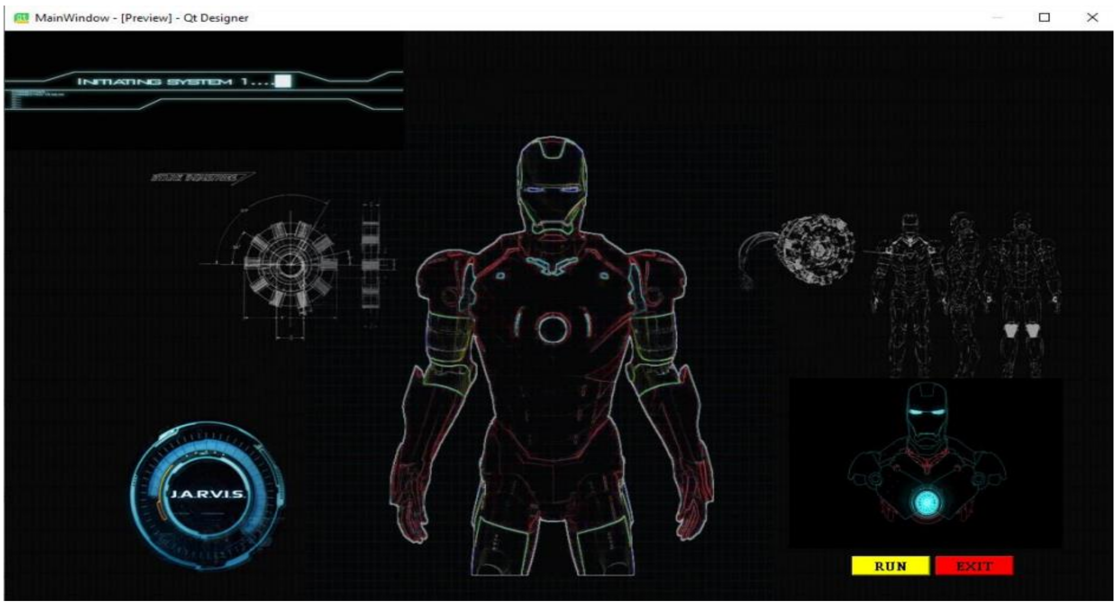
- **USABILITY:**
  Usability of a system is checked by measuring the easiness of the software and how user friendly it is for the user to use, how it responses to each query that is being asked by the user. It makes it easier to complete any task as it automatically do it by using the essential module or libraries of Python, in a conversational interaction way. Hence any user when instruct any task to it, they feel like giving task to a human assistant because of the conversational interaction for giving input and getting the desired output in the form of task done. The desktop assistant is reactive which means it know human language very well and understand the context that is provided by the user and gives response in the same way, i.e. human understandable language, English. So user finds its reaction in an informed and smart way. The main application of it can be its multitasking ability. It can ask for continuous instruction one after other until the user "QUIT" it. It asks for the instruction and listen the response that is given by user without needing any trigger phase and then only executes the task.[2]

- **SECURITY:**
  The security testing mainly focuses on vulnerabilities and risks. As Desktop Assistant is a local desktop application, hence there is no risk of data breaching through remote access. The software is dedicated to a specific system so when the user logs in, it will be activated.[2]

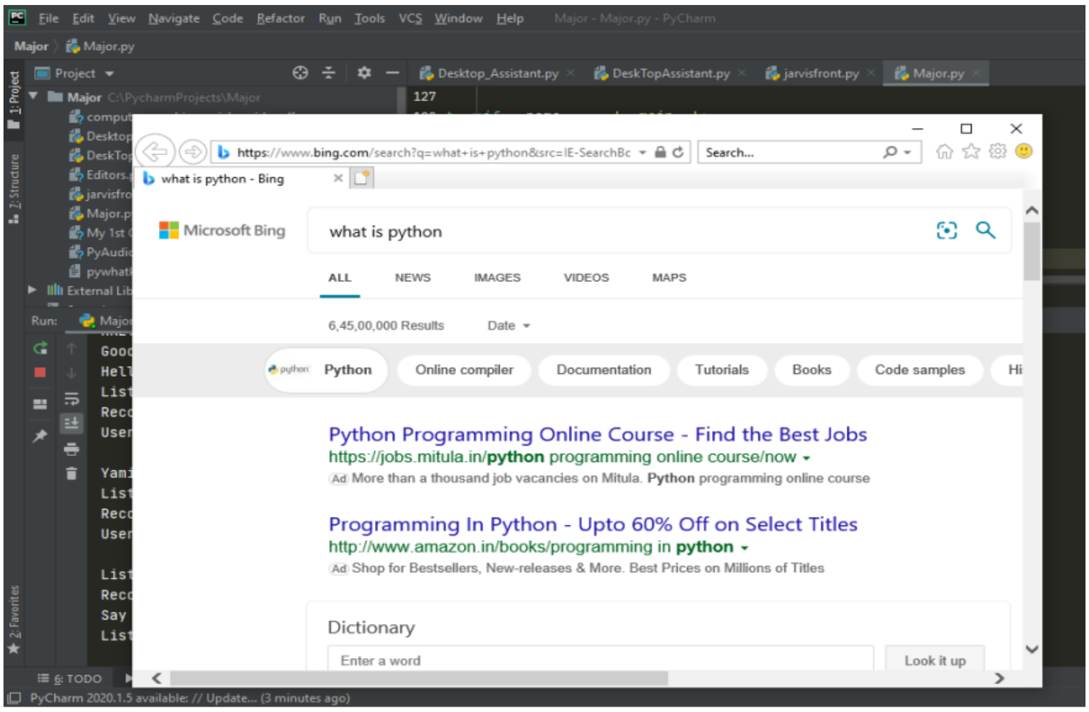
- **STABILITY:**
  stability of a system depends upon the output of the system, if the output is bounded and specific to the bounded input then the system is said to be stable. If the system works on all the poles of functionality then it is stable[2]

Chapter 7. Results

7.1 Live GUI of Desktop Assistant



7.2 Output of Google Search:



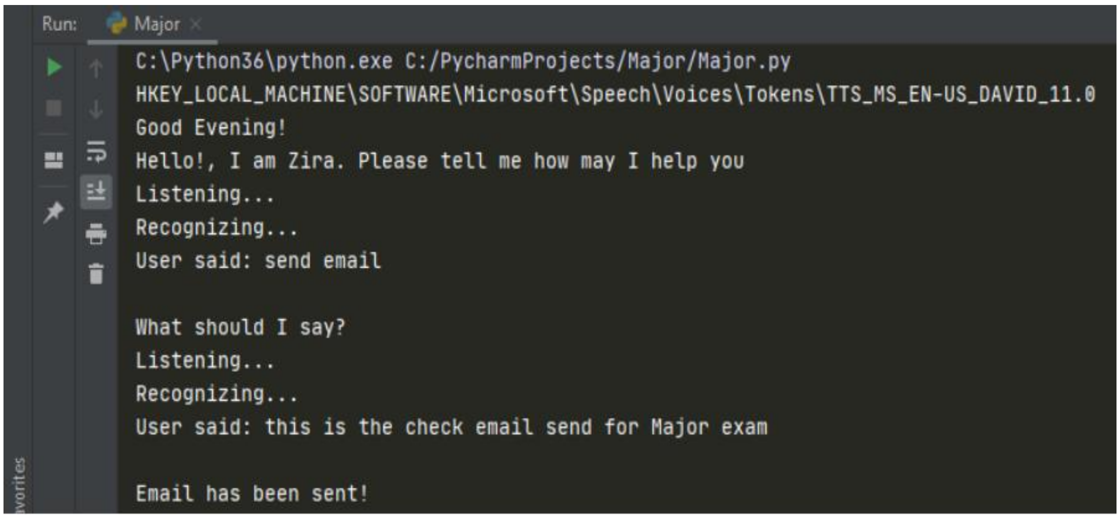
6,45,00,000 Results Date

Python Programming Online Course - Find the Best Jobs
https://jobs.mitula.in/**python** programming online course/now
Ad More than a thousand job vacancies on Mitula. **Python** programming online course

Programming In Python - Upto 60% Off on Select Titles
http://www.amazon.in/books/programming in **python**
Ad Shop for Bestsellers, New-releases & More. Best Prices on Millions of Titles

Dictionary
Enter a word

7.3 Input to send Email

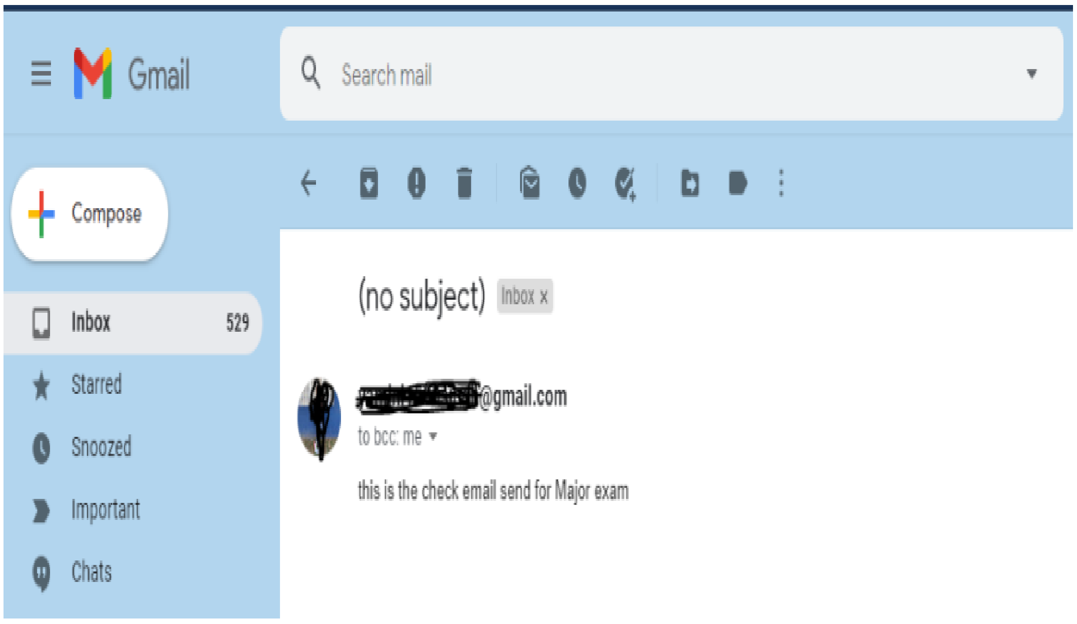


```
C:\Python36\python.exe C:/PycharmProjects/Major/Major.py
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS_MS_EN-US_DAVID_11.0
Good Evening!
Hello!, I am Zira. Please tell me how may I help you
Listening...
Recognizing...
User said: send email

What should I say?
Listening...
Recognizing...
User said: this is the check email send for Major exam

Email has been sent!
```

7.4 output to send Email

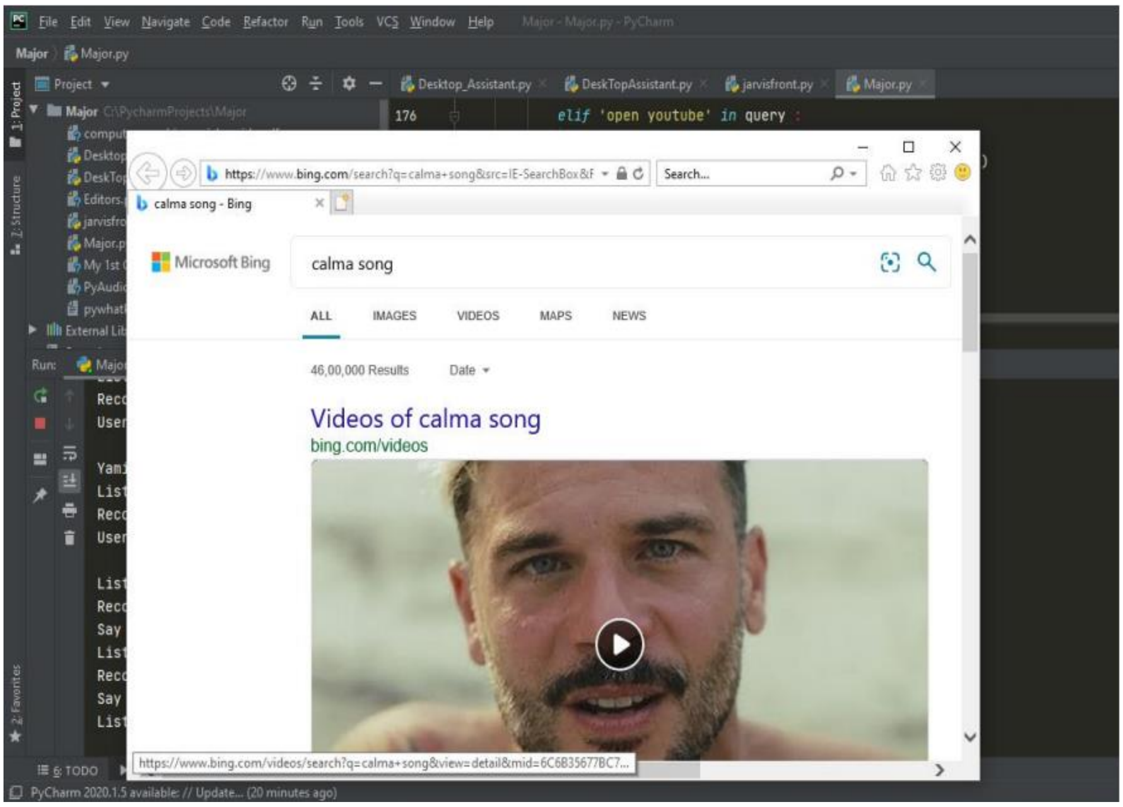


(no subject) Inbox x

[redacted]@gmail.com
to bcc: me

this is the check email send for Major exam

7.5 Output for YouTube search

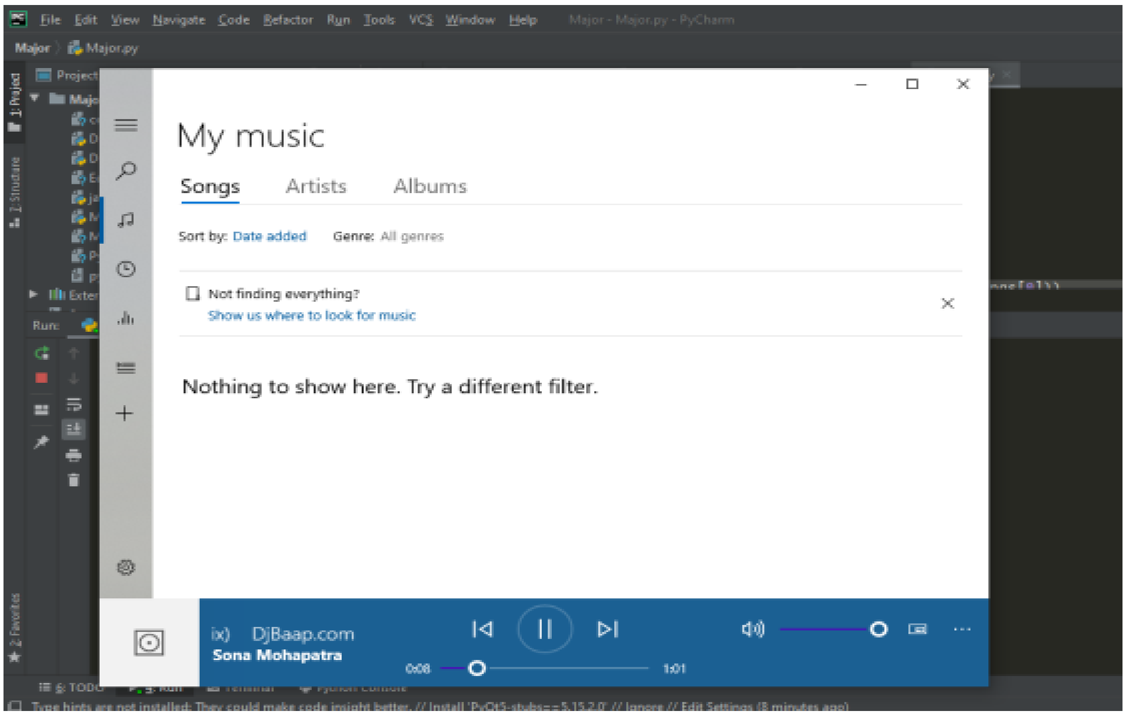


```
176    elif 'open youtube' in query :
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Major - Major.py - PyCharm
Desktop_Assistant.py
DeskTopAssistant.py
jarvisfront.py
Major.py
https://www.bing.com/search?q=calma+song&src=IE-SearchBox&F
calma song - Bing
Microsoft Bing
calma song
ALL
IMAGES
VIDEOS
MAPS
NEWS
46,00,000 Results
Date
Videos of calma song
bing.com/videos
https://www.bing.com/videos/search?q=calma+song&view=detail&mid=6C6B35677BC7...

PyCharm 2020.1.5 available: // Update... (20 minutes ago)

7.6 output to music play



Not finding everything?
Show us where to look for music

Nothing to show here. Try a different filter.

Type hints are not installed: They could make code insight better. // Install 'PyQt5-stubs==5.15.2.0' // Ignore // Edit Settings (8 minutes ago)

Chapter 8. Conclusion and Future Scope

Desktop Assistant is a very helpful voice assistant without any doubt as it saves time of the user by conversational interactions, its effectiveness and efficiency. But while working on this project, there were some limitations encountered and also realized some scope of enhancement in the future.

The Desktop Assistant project, with its foundation built on Python and a versatile range of functionalities, offers a multitude of exciting prospects for the future. As technology evolves and user demands continue to advance, the project can expand and adapt to provide even more value and enhance user experiences. Here, we briefly outline some of the potential future directions and scope for this project:

- **LIMITATIONS:**
  Security is somewhere an issue, there is no voice command encryption in this project.
  Background voice can interfere
  Misinterpretation because of accents and may cause inaccurate results.
  Desktop Assistant cannot be called externally anytime like other traditional assistants like Google Assistant can be called just by saying, "Ok Google!"[1]

- **SCOPE FOR FUTURE WORK:**
  As technology progresses, the Desktop Assistant can incorporate more advanced features. These might include improved natural language processing capabilities, expanded support for third-party applications, and the ability to learn and adapt to users' preferences and habits. Enhancing its voice recognition, web browsing, and system control features could further elevate its utility[1]

  Expanding the Desktop Assistant's compatibility to run on a broader range of operating systems and devices would increase its accessibility. This might involve developing mobile applications or web-based versions of the assistant, making it available to a wider user base.[1]

  In an ever-evolving digital landscape, security and privacy are paramount. Future developments may focus on implementing robust security measures to protect user data and privacy. Incorporating advanced authentication methods and encryption can enhance user confidence.[1]

Allowing users to customize and personalize their Desktop Assistant experience can be a compelling future feature. Users could define their preferences for voice, appearance, and specific functionalities, tailoring the assistant to meet their unique needs.

Leveraging artificial intelligence (AI) and machine learning (ML) technologies can enhance the Desktop Assistant's ability to understand user requests and provide more accurate responses. ML models can be trained to recognize and adapt to user behavior, making interactions more intuitive.

Chapter 9. References

References and Bibliography:

1	https://www.studocu.com/in/document/panjab-university/project-work/final-report-word/38446901
2	https://www.slideshare.net/PRASUNCHAKRABORTY21/desktop-assistant
3	Build a Virtual Assistant Using Python - GeeksforGeeks
4	Software Engineering | Spiral Model - GeeksforGeeks
5	Why is Python So Popular in 2023? | GoSkills