**Technical Document**

1. **Introduction**

The **Technical Document** outlines the technical aspects and design of the **Email Ticketing System**, including the architecture, technology stack, and implementation details.

1. **Architecture**

The system follows a **client-server architecture**:

* **Email Client (IMAP)**: Connects to the IMAP server to fetch incoming customer emails.
* **Backend (Python Script)**: Handles email parsing, keyword detection, ticket creation, and logging.
* **Ticketing System**: Receives data from the backend to create support tickets. This could be **Jira**, **Zendesk**, or any other compatible system.

1. **Technology Stack**

* **Language**: Python 3.x
* **Libraries**:
  + imaplib: For IMAP connection and email fetching.
  + email: For parsing email content and attachments.
  + msal (if using **Microsoft Graph API** for accessing Outlook emails).
  + PyPDF2: For handling and parsing PDF attachments.
  + python-docx: For handling **.docx** file attachments.
  + API
* **Server**: IMAP server (e.g., **Outlook IMAP**).

1. **System Workflow**
2. **Email Fetching**:
   * The system connects to the **IMAP server** using credentials (email and password or OAuth token).
   * Fetches emails from the **Inbox** or any predefined folder.
3. **Email Parsing**:
   * The email content is parsed using the **email** library.
   * If the email contains attachments (PDF or DOCX), they are parsed and processed accordingly.
4. **Ticket Creation**:
   * Based on keywords or issue descriptions, a ticket is automatically created in the designated **ticketing system** (e.g., **Jira**, **Zendesk**).
   * The ticket will include details from the email such as:
     + Subject of the email.
     + Body text of the email.
     + Attachments (if any).
5. **Error Handling**:
   * The system includes basic error handling for scenarios like:
     + Invalid login credentials.
     + IMAP server connectivity issues.
     + Ticketing system API errors.
6. **Logging**:
   * The system logs each step of the process, including:
     + Successful email retrieval.
     + Ticket creation.
     + Any errors that occur during the process.
7. **Code Design**

* **Main Python Script**:
  + **Email Connection Module**: Responsible for connecting to the IMAP server and fetching emails.
  + **Email Parsing Module**: Handles the parsing of email content and attachments.
  + **Ticket Creation Module**: Responsible for creating tickets using the ticketing system's API.
  + **Logging Module**: For detailed logging and error handling.

import imaplib

import email

from email.header import decode\_header

import json

def connect\_to\_imap():

try:

print("Connecting to IMAP server...")

mail = imaplib.IMAP4\_SSL("outlook.office365.com")

mail.login("your\_email@domain.com", "your\_app\_password") # Use app password if 2FA enabled

print("Logged in successfully!")

return mail

except Exception as e:

print(f"Failed to connect: {e}")

return None

def fetch\_emails(mail):

status, messages = mail.search(None, "ALL")

if status != 'OK':

print("Failed to fetch emails.")

return []

email\_ids = messages[0].split()

emails = []

for email\_id in email\_ids:

status, msg\_data = mail.fetch(email\_id, "(RFC822)")

for response\_part in msg\_data:

if isinstance(response\_part, tuple):

msg = email.message\_from\_bytes(response\_part[1])

subject = decode\_header(msg["Subject"])[0][0]

body = msg.get\_payload(decode=True).decode()

emails.append({"subject": subject, "body": body})

return emails

def create\_ticket(email\_data):

# Example ticket creation function

ticket = {

"summary": email\_data["subject"],

"description": email\_data["body"]

}

print(f"Creating ticket: {ticket}")

# Send this to Jira API or Zendesk API

return ticket

# Main function

def main():

mail = connect\_to\_imap()

if mail:

emails = fetch\_emails(mail)

for email\_data in emails:

create\_ticket(email\_data)

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. **Error Handling and Logging**

* Errors will be caught and logged at each step (connection, fetching emails, ticket creation).
* The system will log:
  + **Successful actions**: Email fetched, ticket created.
  + **Error messages**: IMAP login failed, ticket creation failed.

1. **Deployment**

* **Server**: The script can run on any server with **Python** installed.
* **Schedule**: You can schedule the script to run periodically (e.g., every hour)