# IMPLEMENTING SMTP & IMAP USING PYTHON

*Report submitted to the SASTRA Deemed to be University as the requirement for the course*

## CSE302: COMPUTER NETWORKS

*Submitted by*

## AKASH S

**Reg. No.: 124004393**

**B. Tech. Electronics and Communication Engineering**

**December 2022**



# SCHOOL OF ELECTRIC AND ELECTRONICS ENGINEERING.

## THANJAVUR, TAMIL NADU, INDIA – 613 401



**SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING.**

**THANJAVUR – 613 401**

**Bonafide Certificate**

This is to certify that the report titled “**Implementing SMTP & IMAP using Python”** submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B. Tech. is a bonafide record of the work done by **Shri. AKASH S (Reg. No. 124004393, B. Tech. Electronics and Communication Engineering)** during the academic year 2022-23, in the School of Electric and Electronics Engineering.

Project Based Work *Viva voce* held on

## Examiner 1 Examiner 2

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| 5.1 | Connecting to hMailServer software | 26 |
| 5.2 | Dashboard of hMailServer | 26 |
| 5.3 | Creating a user account | 27 |
| 5.4 | Adding Domain name in the hosts file | 27 |
| 5.5 | Splash/Login screen of the Application | 28 |
| 5.6 | Home Screen of Application | 28 |
| 5.7 | Compose the mail screen in the Application | 29 |
| 5.8 | Add an attachment Dialog Box in the Application | 29 |
| 5.9 | Accessing the Inbox folder in the Application | 30 |
| 5.10 | Select the Directory Dialog Box for downloading the Attachment in the Application | 30 |
| 5.11 | Accessing the Sent folder in the Application | 31 |

.

## Abbreviations

|  |  |
| --- | --- |
| IP | Internet Protocol |
| GUI | Graphical User Interface |
| SMTP | Simple Mail Transfer Protocol |
| IMAP | Internet Message Access Protocol |
| POP3 | Post Office Protocol 3 |
| TCP | Transmission Control Protocol |
| MTA | Mail Transfer Agent |
| MUA | Mail User Agent |
| DNS | Domain Name Server |
| MX | Mail eXchanger |
| TLS | Transport Layer Security |
| SSL | Secure Sockets Layer |
| MIME | Multipurpose Internet Mail Extensions |
| ASCII | American Standard Code for Information Interchange |
| RFC | Requests for Comments |

**Abstract**

Email is an effective method of business communication that is fast, accessible, and cheap. Using email can be greatly beneficial even for small-scale communications such as conversations between friends or a teacher sending documents & notes to his/her students. One of the main advantages of email is that you can send digital files such as Text Documents, Images, etc. quickly and simultaneously. Many business companies use Email to send in their invoices and receipts to their customers.

A Mail Server is set locally to mimic the large-scale public mail servers in warehouses. User accounts are created in the mail server software to receive emails from other mail addresses of the same domain. Currently, we can only receive from or send to the user with the same domain name. In the future, we require a machine with a static public IP address to host our mail server. By doing so, we can send or receive emails from other domains too.

A GUI application has been created using Python to access our Mailboxes and send emails. This application uses SMTP and IMAP methods to send & retrieve mail respectively. This is a small-scale application that can send emails, and read & delete mails from Inbox, Sent & Trash mailboxes. Also, attachments of any type & size can be sent or downloaded to your local directory.

**KEYWORDS:** SMTP, IMAP, Mail Server, Domain, Mailbox

**Table of Contents**

|  |  |
| --- | --- |
| **Title** | **Page No.** |
| Bona Fide Certificate | ii |
| List of Figures | iii |
| Abbreviations | iv |
| Abstract | v |
| 1. Introduction | 1 |
| 1.1. System Requirements | 1 |
| 2. Protocols | 2 |
| 2.1. SMTP | 2 |
| 2.2. IMAP | 2 |
| 2.3. POP3 | 3 |
| 3. Implementation | 4 |
| 3.1. Setting up a Mail Server locally | 4 |
| 3.2. Working of Application | 4 |
| 4. Source Code | 6 |
| 5. Snapshots | 18 |
| 6. Conclusion | 23 |
| 7. References | 24 |

# INTRODUCTION

A server is a computer that sends and receives email over a network known as Mail Server. In many cases, web servers and mail servers are combined into a single machine. However, large companies like Gmail and Hotmail have dedicated Mail servers to reduce traffic on their servers. Every mail server must include mail server software for the machine to act as a mail server. Mail Server Software allows the system to administer, create and manage email accounts for any domains hosted on the server.

Standard email protocols are used by the mail server for sending and receiving emails. SMTP sends messages and handles outgoing mail requests. IMAP and POP3 receive mail and are used to process incoming mail. When a user logs on to the mail server using an email client, these protocols handle all the connections behind the scenes. These protocols use TCP for establishing a connection between the user and the mail server.

Firstly, emails are composed using Mail Transfer Agent (MTA) by the user. The email client assembles the email by joining the content of the mail with the recipient, subject, date, etc.

The email client uses SMTP to send mail to the outgoing mail server. The outgoing mail server with the help DNS server finds the IP address of the recipient. This is done with the help of the server’s MX records After getting the necessary details, SMTP sends the email to the recipient’s incoming mail server. The receiving server will store the mail in its database for the access of the user.

The mail server of the recipient doesn’t straight away accept mail from anyone. It always checks for the existence of the sender of the mail. The DNS server makes this possible for the server with the help of MX records. Accessing the stored emails from the server is done with the help of IMAP or POP3. Mail User Agent (MUA) makes it possible for the user who can read the emails from his/her mailbox.

## SYSTEM REQUIREMENTS:

* OS: Windows 11, 10, 8.1, 8, 7, Vista, XP Professional
* Python 3.6 or above
* Tkinter module

# PROTOCOLS

## SMTP:

SMTP stands for “Simple Mail Transfer Protocol”. It is part of the Application Layer of the TCP/IP protocol. It instructs how the mail must move from one MTA to another MTA. Mails might move to many mail servers before reaching the recipient’s server. SMTP uses the “Store and Forward” feature to send tail emails TP and provides a set of codes that simplify the communication of email between servers. It breaks the mail into chunks for easy transmission. SMTP works on port 25 (standard for mail transfer), port 465 (not compliant with RFC), and port 587 (TLS encrypted).

Advantages:

* SMTP provides the simplest form of email communication.
* Emails can be sent quickly and effectively.
* Offers reliability in terms of outgoing mail.

Disadvantages:

* SMTP can only send plain text messages, not fancy items like attachments or fonts. This part is done by the MIME. It converts non-text elements to plain text.
* SMTP is limited to 7-bit ASCII characters.
* Some firewalls may block common ports required by SMTP.

## IMAP:

IMAP stands for “Internet Message Access Protocol”. It is part of the Application Layer of the TCP/IP protocol. Emails are stored on the servers. IMAP is used to access the mail irrespective of your geographical location. Unlike POP, emails stay in the servers for future access even after retrieving them. IMAP can be used from multiple devices since the mail is only read when it is first accessed not deleted from the server. IMAP works on port 143 (non-encrypted), and port 993 (TLS/SSL encrypted).

Advantages:

* Emails can be accessed from any place, via as many devices as you want.
* Attachments aren’t downloaded by default. This makes reading the body of the email quicker.
* Even after reading, the mail stays in the server for future reading, unlike POP which deletes the mail after downloading.

Disadvantages:

* It is mandatory for an internet connection whenever using IMAP.
* Accessing the mailbox is slower than POP as all folders need to get synchronized every time.

## POP3:

POP3 stands for “Post Office Protocol 3”. It is part of the Application Layer of the TCP/IP protocol. POP3 allows users to download emails from the server and read them offline. Using POP3 is simple and easy to use. This protocol was designed keeping in mind the users having only a temporary internet connection. POP3 works on port 110 (non-encrypted), and port 995 (TLS/SSL encrypted).

Advantages:

* No internet connection is needed to read emails once the emails are downloaded.
* Big storage is not needed for the server, since after downloading, emails are deleted from the server.

Disadvantages:

* If the device where the emails are stored is lost, emails are lost too.
* Storage on the user side is required for storing mail.

# IMPLEMENTATION

## SETTING UP A MAIL SERVER LOCALLY:

* Download hMailSever from the official website
* Install hMailServer on your local machine
* Connect to the hostname “localhost”
* Under Domain, click the “Add Domain” button
* Click “Add”, enter your domain name (e.g., test.com), and click “Save”
* Under Settings -> Protocols -> SMTP, click the “Delivery of e-mail” tab. Enter “localhost” in the Local host name entry text field, click “Save”
* Under Domains -> “Your Domain name” -> Accounts, click “Add”
* Enter your Username in the Address text field, Password in the Password text field, and click “Save”
* Continue the previous step with as many users as required.
* Minimize the application
* Open C:\Windows\System32\drivers\etc\hosts using Notepad as Administrator.
* Enter “127.0.0.1 your domain.XYZ” in the file and save

## WORKING OF APPLICATION:

This application uses smtplib and imaplib Python modules. The user is prompted to enter his/her email credentials to connect to his/her mailbox using SMTP and IMAP. The credentials can either be from the previously created mail server or Gmail login credentials. After entering the credentials, if the credentials are valid, the user is taken to the next screen, or else the login page persists.

In the next screen, the user is shown the basic mail client look. The screen contains buttons like “Compose”, “Inbox”, “Sent”, and “Logout”. The user can select any of those buttons. Log out button is available so that the user can log out of his/her mailbox. The “Compose” button lets the user compose a MIME-enabled email and send it to any address of any domain if the user is logged in using his/her Gmail credentials or only sends it to the users within the same domain. “Inbox”, “Send t”, and “Trash” buttons let the user view his/her respective mailbox. The “Delete” button lets the user delete unwanted mail present in Inbox and send it to the

Trash folder. If a mail contains attachments, the user is able is download those to his local machine’s F:\Mail Attachments folder.

The basic underlying working code is borrowed from the smtplib and imaplib modules. SMTP\_SSL and SMTP classes with mail server addresses as constructors inside smtplib create an SMTP object with secured and unencrypted connection respectively. Log in to the respective mailbox is done by the login() function. sendmail() inside the SMTP class lets the user send MIME-enabled emails.

IMAP4\_SSL and IMAP4 classes with mail server addresses as constructors inside imaplib create IMAP objects with secured and unencrypted connections respectively. Log in to the respective mailbox is done by the login() function. Because of its high functionality, IMAP is complex at the code level. After object creation, the selection of mailboxes is required. Then, searching for emails is required. After searching, fetching the selected mail is done. The mail is general in the byte string formal. After formatting the string, we get the mail in a readable format.

The application when the “Send” button is clicked after composing the mail, gets all the input data and converts them to a byte string, and sent as a parameter to the sendmail() function. Also, while retrieving the mail, the application converts them into a human-readable format and displays the mail.

When a mail is deleted, a copy of the mail is made in the Trash folder and marks the mail as deleted with the help of the \\Deleted flag. expunge() function is called, which permanently removes emails that are marked as \\Deleted. When a mail is deleted from the Trash folder, no copy is made but directly deleted from the mailbox.

Attachments are downloaded to the local drive using Python’s open() function. Generally, the attachments are sent after encoding them as text. Here, the application decodes the text and writes the bytes to F:\Mail Attachments folder. Inside the Mail Attachments folder, the corresponding folder is created and files are written into the folder.

Additionally, the email module is used to convert the attachment part to text format. An object of class EmailMessage() is used to convert any type of file to a text file with base64 encoding. While reading the emails, the message\_from\_bytes() function is used to convert the email messages to a human-readable format.

# SOURCE CODE

# # Importing necessary modules

# from tkinter import \*

# from tkinter import messagebox

# from tkinter import filedialog

# from tkinter.scrolledtext import ScrolledText

# from email.message import EmailMessage

# import smtplib

# import imaplib

# import email

# import os

# import time

# # Attachment Directory

# attachment\_dir = 'F:/Mail Attachments'

# # Creating SMTP & IMAP objects

# smtp = None

# imap = None

# Gmail\_Login = True

# def objectCreation(email\_var):

# global smtp

# global imap

# global Gmail\_Login

# # Clearing previous user contents

# mail\_list.clear()

# scrl.delete(1.0, END)

# listbox.delete(0)

# attach\_label.config(text="")

# no\_of\_mails.config(text="")

# if email\_var.get().split('@')[-1] == 'gmail.com':

# smtp = smtplib.SMTP\_SSL('smtp.gmail.com', 465)

# imap = imaplib.IMAP4\_SSL('imap.gmail.com')

# else:

# smtp = smtplib.SMTP('localhost', 587)

# imap = imaplib.IMAP4('localhost', 143)

# Gmail\_Login = False

# # Creating a Toggle function

# def toggle(togg=[0]):

# togg[0] = not togg[0]

# if togg[0]:

# top.deiconify()

# top.state('zoomed')

# root.withdraw()

# else:

# top.withdraw()

# print(smtp.quit())

# print(imap.logout())

# root.deiconify()

# # Creating Login Check function (SMTP)

# def login\_check\_smtp(email\_var, password\_var):

# try:

# print(smtp.login(email\_var.get(),

# password\_var.get()))

# return 1

# except:

# return 0

# # Creating Login Check function (IMAP)

# def login\_check\_imap(email\_var, password\_var):

# try:

# print(imap.login(email\_var.get(),

# password\_var.get()))

# return 1

# except:

# return 0

# # Creating Toggle & Login function

# def toggle\_and\_login\_check(email\_var, password\_var):

# if email\_var.get() == '' or password\_var.get() == '':

# print('Either empty Username or Password')

# root\_info\_label.config(text='Username or Password is Empty')

# else:

# objectCreation(email\_var)

# if login\_check\_smtp(email\_var, password\_var) == 1 and login\_check\_imap(email\_var, password\_var) == 1:

# toggle()

# else:

# root\_info\_label.config(text='Login Unsuccessful')

# # Creating Send Mail function

# def send\_mail(from\_addr, to\_var, cc\_var, bcc\_var, subject\_var, body\_var, attach, attachments=[]):

# to = to\_var.get()

# cc = cc\_var.get()

# bcc = bcc\_var.get()

# subject = subject\_var.get()

# body = body\_var.get(1.0, END).strip()

# print('To addr', to)

# print('Subject', subject)

# print('Body', body)

# if to == '' or subject == '' or body == '':

# messagebox.askokcancel('Error!!', 'Either Empty To address, Subject or Body')

# else:

# msg = EmailMessage()

# msg['To'] = str(to)

# msg['Subject'] = subject

# msg['From'] = from\_addr

# msg['Cc'] = cc

# msg['Bcc'] = bcc

# msg['Date'] = time.ctime(time.time())

# msg.set\_content(body)

# print('Before:', attachments)

# if attachments != []:

# print('After:', attachments)

# for value in attachments:

# with open(value, 'rb') as f:

# file\_name = f.name.split('/')[-1]

# file\_data = f.read()

# print('Filename', file\_name)

# msg.add\_attachment(file\_data, maintype='application', subtype='octet-stream', filename=file\_name)

# try:

# smtp.send\_message(msg)

# print('Main sent')

# if Gmail\_Login == False:

# imap.append('Sent', '\\Seen', imaplib.Time2Internaldate(time.time()), msg.as\_string().encode('utf8'))

# messagebox.askokcancel('Success!!', 'Mail sent successfully!!')

# to\_var.delete(0, END)

# cc\_var.delete(0, END)

# bcc\_var.delete(0, END)

# subject\_var.delete(0, END)

# body\_var.delete(1.0, END)

# attach.config(text='')

# attachments = None

# except:

# messagebox.askokcancel('Error!!', 'Cannot send mail, check Inputs!!')

# attachment\_list = []

# attachment\_label = None

# mail\_list = []

# mb = ""

# dir\_name = ""

# # Creating Put Content in ListBox function

# def put\_content(l):

# listbox.delete(0, END)

# for index, value in enumerate(l[::-1]):

# str\_value = 'Date: {}, Subject: {},To: {}, From: {}'.format(value['Date'], value['Subject'], value['To'],

# value['From'])

# listbox.insert(index, str\_value)

# no\_of\_mails.config(text='Total no. of mails: {}'.format(len(l)))

# # Creating Print Content in ScrolledText function

# def print\_content(event, l):

# scrl.delete(1.0, END)

# print(listbox.curselection()[0])

# scrl.insert(1.0, l[::-1][listbox.curselection()[0]]['Body'])

# if l[::-1][listbox.curselection()[0]]['Attachment']:

# attach\_label.config(text='Attachment(s) available!!')

# else:

# attach\_label.config(text='')

# # Creating Get Body function

# def get\_body(msg):

# if msg.is\_multipart():

# return get\_body(msg.get\_payload(0))

# else:

# return msg.get\_payload(None, True)

# # Creating Get Attachment(s) function

# def get\_attachments(msg):

# for part in msg.walk():

# if part.get\_content\_maintype() == 'multipart':

# continue

# if part.get('Content-Disposition') is None:

# continue

# filename = part.get\_filename()

# if bool(filename):

# global dir\_name

# dir\_name = filedialog.askdirectory(initialdir='D:')

# filepath = os.path.join(dir\_name, filename)

# if os.path.isfile(filepath) == False:

# with open(filepath, 'wb') as f:

# f.write(part.get\_payload(decode=True))

# else:

# a = 1

# while True:

# temp = filename.split('.')

# alt\_filename = temp[0] + ' ({})'.format(a) + '.' + temp[1]

# alt\_filepath = os.path.join(dir\_name, alt\_filename)

# if os.path.isfile(alt\_filepath) == False:

# with open(alt\_filepath, 'wb') as f:

# f.write(part.get\_payload(decode=True))

# break

# a += 1

# # Creating Reverse List Mail order function

# def download\_attachment():

# \_, search = imap.search(None, 'ALL')

# \_, data = imap.fetch(search[0].split()[::-1][listbox.curselection()[0]], '(RFC822)')

# msg = email.message\_from\_bytes(data[0][1])

# if check\_attachment(msg):

# get\_attachments(msg)

# messagebox.askokcancel('Success!!', 'Attachment(s) downloaded!! in ' + dir\_name)

# else:

# messagebox.askokcancel('Missing!!', "Attachment for this mail doesn't exist!!")

# # Creating Check Attachment(s) function

# def check\_attachment(msg):

# for part in msg.walk():

# if part.get\_content\_maintype() == 'multipart':

# continue

# if part.get('Content-Disposition') is None:

# continue

# return bool(part.get\_filename())

# # Creating Delete Mail function

# def delete\_mail():

# global mail\_list

# \_, search = imap.search(None, 'ALL')

# ####imap.store(search[0].split()[::-1][listbox.curselection()[0]], '+FLAGS', '"[Gmail]/Trash"')####

# if Gmail\_Login == True:

# imap.store(search[0].split()[::-1][listbox.curselection()[0]], '+X-GM-LABELS', '\\Trash')

# else:

# \_, s\_d = imap.search(None, 'ALL')

# try:

# if imap.state == 'SELECTED':

# \_, da = imap.fetch(s\_d[0].split()[::-1][listbox.curselection()[0]], '(UID)')

# else:

# messagebox.askokcancel('Wait!!', 'No mail has been selected!!')

# except:

# messagebox.askokcancel('Wait!!', 'No mail has been selected!!')

# if mb != 'Trash':

# res = imap.uid('COPY', da[0].decode().split()[-1].split(')')[0], 'Trash')

# if res[0] == 'OK':

# \_, da = imap.uid('STORE', da[0].decode().split()[-1].split(')')[0], '+FLAGS', '\\Deleted')

# imap.expunge()

# elif mb == 'Trash':

# \_, da = imap.uid('STORE', da[0].decode().split()[-1].split(')')[0], '+FLAGS', '\\Deleted')

# imap.expunge()

# print(mail\_list.pop(len(mail\_list) - 1 - listbox.curselection()[0]))

# listbox.delete(listbox.curselection()[0])

# scrl.delete(1.0, END)

# no\_of\_mails.config(text='Total no. of mails: {}'.format(len(mail\_list)))

# # Creating Receive Mail function

# def recv\_mail(mailbox='INBOX'):

# if mailbox == 'Sent' and Gmail\_Login == True:

# mailbox = '"[Gmail]/Sent Mail"'

# elif mailbox == 'Trash' and Gmail\_Login == True:

# mailbox = '"[Gmail]/Bin"'

# mail\_list.clear()

# scrl.delete(1.0, END)

# listbox.delete(0)

# if attach\_label != None:

# attach\_label.config(text="")

# imap.select(mailbox)

# global mb

# mb = mailbox

# \_, search\_data = imap.search(None, 'ALL')

# for num in search\_data[0].split():

# email\_data = {}

# \_, data = imap.fetch(num, '(RFC822)')

# email\_msg = email.message\_from\_bytes(data[0][1])

# for header in ['Subject', 'To', 'From', 'Date']:

# email\_data[header] = email\_msg[header]

# email\_data['Body'] = email.message\_from\_bytes(get\_body(email\_msg))

# email\_data['Attachment'] = check\_attachment(email\_msg)

# mail\_list.append(email\_data)

# put\_content(mail\_list)

# # Creating Add Attachment function

# def put\_attachment():

# global attachment\_list

# temp = filedialog.askopenfilenames(initialdir='D:\\', title='Select Attachment(s)',

# filetypes=(('All files', '\*.\*'), ('JPEG files', '\*.jpeg')))

# for k in temp:

# attachment\_list.append(k)

# print('Inside function', attachment\_list)

# attachment\_str = []

# for value in attachment\_list:

# attachment\_str.append(value.split('/')[-1])

# attachment\_label.config(text=','.join(attachment\_str))

# # Creating Compose Mail function

# def compose\_mail():

# # Creating Compose Mail Window

# com = Toplevel()

# com.title('Compose Mail')

# com.iconbitmap('./email.ico')

# # Creating Entry Variables

# to\_entry\_variable = StringVar()

# cc\_entry\_variable = StringVar()

# bcc\_entry\_variable = StringVar()

# subject\_entry\_variable = StringVar()

# scrolled\_text\_variable = StringVar()

# print('Outside function before calling', attachment\_list)

# # Creating To Label

# to\_label = Label(com, text='To', font=('', 10))

# to\_label.grid(row=0, column=0, padx=10, pady=10)

# # Creating CC Label

# cc\_label = Label(com, text='CC', font=('', 10))

# cc\_label.grid(row=1, column=0, padx=10, pady=10)

# # Creating BCC Label

# bcc\_label = Label(com, text='BCC', font=('', 10))

# bcc\_label.grid(row=2, column=0, padx=10, pady=10)

# # Creating Subject Label

# subject\_label = Label(com, text='Subject', font=('', 10))

# subject\_label.grid(row=3, column=0, padx=10, pady=10)

# # Creating Body Label

# body\_label = Label(com, text='Body', font=('', 10))

# body\_label.grid(row=4, column=0, padx=10, pady=10, sticky=N)

# # Creating To Entry

# to\_entry = Entry(com, font=('', 10), width=60)

# to\_entry.grid(row=0, column=1, padx=10, pady=10, sticky=W)

# # Creating CC Entry

# cc\_entry = Entry(com, font=('', 10), width=60)

# cc\_entry.grid(row=1, column=1, padx=10, pady=10, sticky=W)

# # Creating BCC Entry

# bcc\_entry = Entry(com, font=('', 10), width=60)

# bcc\_entry.grid(row=2, column=1, padx=10, pady=10, sticky=W)

# # Creating Subject Entry

# subject\_entry = Entry(com, font=('', 10), width=60)

# subject\_entry.grid(row=3, column=1, padx=10, pady=10, sticky=W)

# # Creating Body ScrolledText

# scrolled\_text = ScrolledText(com, wrap=WORD, font=('Verdana', 12))

# scrolled\_text.grid(row=4, column=1, padx=10, pady=10)

# # Creating Attach button

# attachment\_btn = Button(com, text='Attach', fg='white', bg='blue', font=('', 10), command=put\_attachment)

# attachment\_btn.grid(row=5, column=1, sticky=E, padx=10, pady=10)

# # Creating Send Button

# send\_btn = Button(com, text='Send', bg='blue', fg='white', font=('', 12),

# command=lambda: send\_mail(root\_email\_entry.get(),

# to\_entry, cc\_entry, bcc\_entry, subject\_entry, scrolled\_text,

# attachment\_label, attachment\_list))

# send\_btn.grid(row=5, column=0, padx=10, pady=10)

# # Creating Attachment Label

# global attachment\_label

# attachment\_label = Label(com, font=('', 10))

# attachment\_label.grid(row=5, column=1, sticky=W, padx=10, pady=10)

# # Creating Root Window Destroy function

# def on\_closing():

# if messagebox.askyesno('Quit', 'Do you want to quit?'):

# root.destroy()

# # Creating Login Window Destroy function

# def on\_closing\_top(x):

# if messagebox.askyesno('Quit', 'Do you want to quit?'):

# x.destroy()

# root.deiconify()

# # Creating Login Window (Main)

# root = Tk()

# root.title('SMTP & IMAP')

# root.iconbitmap('./email.ico')

# root.config(bg='black')

# # Screen Details

# app\_width = 550

# app\_height = 400

# screen\_width = root.winfo\_screenwidth()

# screen\_height = root.winfo\_screenheight()

# x = (screen\_width // 2) - (app\_width // 2)

# y = (screen\_height // 2) - (app\_height // 2)

# root.geometry(f'{app\_width}x{app\_height}+{x}+{y}')

# # Creating Login Frame

# root\_frame = Frame(root, bg='yellow')

# root\_frame.place(in\_=root, anchor='c', relx=.5, rely=.5)

# # Creating Login page variables

# root\_email\_variable = StringVar()

# root\_password\_variable = StringVar()

# # Creating Email Label

# root\_email\_label = Label(root\_frame, text='Email', font=('', 16), bg='white')

# root\_email\_label.grid(row=0, column=0, padx=5, pady=5)

# # Creating Email Entry

# root\_email\_entry = Entry(root\_frame, font=('Verdana', 16), textvariable=

# root\_email\_variable, width=30)

# root\_email\_entry.focus\_set()

# root\_email\_entry.grid(row=0, column=1, padx=5, pady=5)

# # Creating Password Label

# root\_password\_label = Label(root\_frame, text='Password', font=('', 16), bg='white')

# root\_password\_label.grid(row=1, column=0, padx=5, pady=5)

# # Creating Password Entry

# root\_password\_entry = Entry(root\_frame, font=('Verdana', 16), show='\*', textvariable=root\_password\_variable, width=30)

# root\_password\_entry.grid(row=1, column=1, padx=5, pady=5)

# # Creating Login Button

# root\_login\_button = Button(root\_frame, text='Login', font=('', 16), command=lambda:

# toggle\_and\_login\_check(root\_email\_variable, root\_password\_variable))

# root\_login\_button.grid(columnspan=2, padx=5, pady=5)

# # Creating Info Label

# root\_info\_label = Label(root\_frame, text='', fg='red', font=('', 16), bg='yellow')

# root\_info\_label.grid(columnspan=2, padx=5, pady=5)

# # Creating Mail sending Window

# top = Toplevel()

# top.withdraw()

# top.iconbitmap('./email.ico')

# # Creating Mail Window frames

# left\_frame = Frame(top, borderwidth=5, highlightthickness=3)

# left\_frame.pack(side=LEFT, expand=False, fill=Y)

# right\_frame = Frame(top)

# right\_frame.pack(side=LEFT, expand=True, fill='both')

# mails\_frame = Frame(right\_frame)

# mails\_frame.pack(side=TOP, expand=True, fill='both')

# view\_frame = Frame(right\_frame)

# view\_frame.pack(side=TOP, expand=True, fill='both')

# # Creating Compose Button

# compose\_btn = Button(left\_frame, text='Compose', font=('', 16), command=

# compose\_mail)

# compose\_btn.pack(side=TOP, padx=30, pady=20, fill='both')

# # Creating Inbox Button

# inbox\_btn = Button(left\_frame, text='Inbox', font=('', 16), command=lambda:

# recv\_mail('INBOX'))

# inbox\_btn.pack(side=TOP, padx=30, pady=20, fill='both')

# # Creating Sent Mail Button

# sent\_mail\_button = Button(left\_frame, text='Sent', font=('', 16), command=lambda:

# recv\_mail('Sent'))

# sent\_mail\_button.pack(side=TOP, padx=30, pady=20, fill='both')

# # Creating Trash Button

# trash\_button = Button(left\_frame, text='Trash', font=('', 16), command=lambda:

# recv\_mail('Trash'))

# trash\_button.pack(side=TOP, padx=30, pady=20, fill='both')

# # Creating Delete Mail Button

# delete\_button = Button(left\_frame, text='Delete', font=('', 16), command=delete\_mail)

# delete\_button.pack(side=TOP, padx=30, pady=20, fill='both')

# # Creating Number of mails Label

# no\_of\_mails = Label(left\_frame, font=('', 16))

# no\_of\_mails.pack(side=TOP, padx=30, pady=20, fill='both')

# # Creating Attachment Label

# attach\_label = Label(left\_frame, font=('', 16))

# attach\_label.pack(side=TOP, padx=30, pady=20, fill='both')

# # Creating Logout Button

# logout\_btn = Button(left\_frame, text='Logout', font=('', 16), command=toggle)

# logout\_btn.pack(side=BOTTOM, padx=30, pady=10, fill='both')

# # Creating Download Attachment Button

# download\_attach = Button(left\_frame, text='Download', font=('', 16), command=download\_attachment)

# download\_attach.pack(side=BOTTOM, padx=30, pady=20, fill='both')

# # Creating ListBox for Mails view (Scrollable)

# scrl\_x = Scrollbar(mails\_frame, orient='horizontal')

# scrl\_x.pack(side=BOTTOM, fill='both', padx=5)

# listbox = Listbox(mails\_frame, font=('Verdana', 12))

# listbox.pack(side=LEFT, padx=5, pady=5, fill='both', expand=True)

# listbox.bind('<Double-1>', lambda event: print\_content(event, mail\_list))

# scrl\_y = Scrollbar(mails\_frame)

# scrl\_y.pack(side=RIGHT, fill='both', pady=5)

# listbox.config(xscrollcommand=scrl\_x.set, yscrollcommand=scrl\_y.set)

# scrl\_x.config(command=listbox.xview)

# scrl\_y.config(command=listbox.yview)

# # Creating a Viewing area for Mail

# scrl = ScrolledText(view\_frame, width=40, height=10, font=('Verdana', 14), wrap=WORD)

# scrl.pack(side=LEFT, padx=5, pady=5, fill='both', expand=True)

# # Closing Windows protocols

# root.protocol("WM\_DELETE\_WINDOW", on\_closing)

# top.protocol("WM\_DELETE\_WINDOW", lambda: on\_closing\_top(top))

# root.mainloop()

# SNAPSHOTS

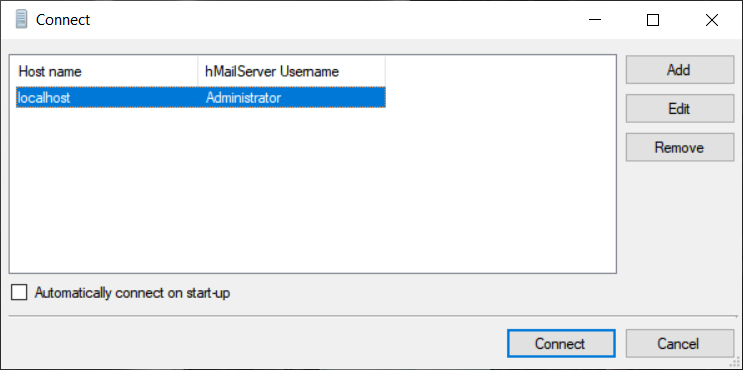


Fig. 5.1. Connecting to hMailServer software

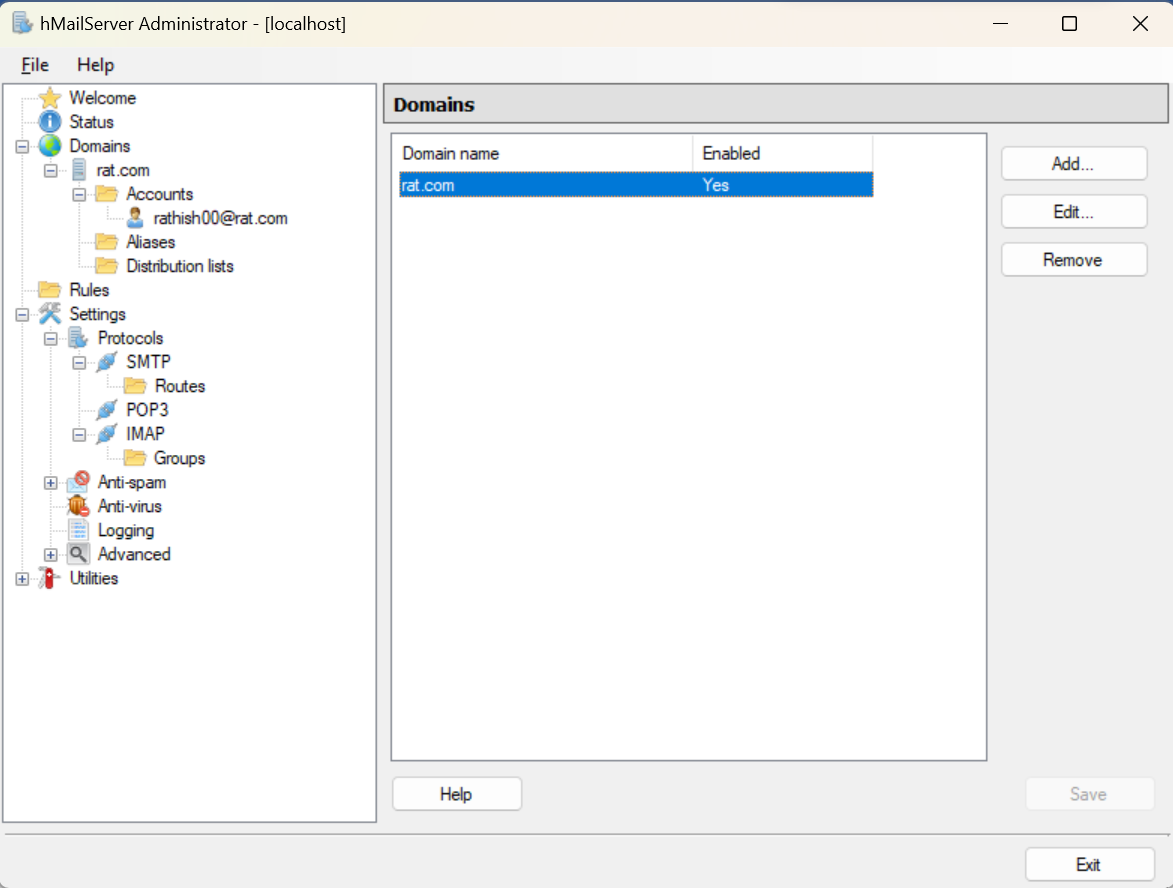


Fig.5.2.Creating Domain

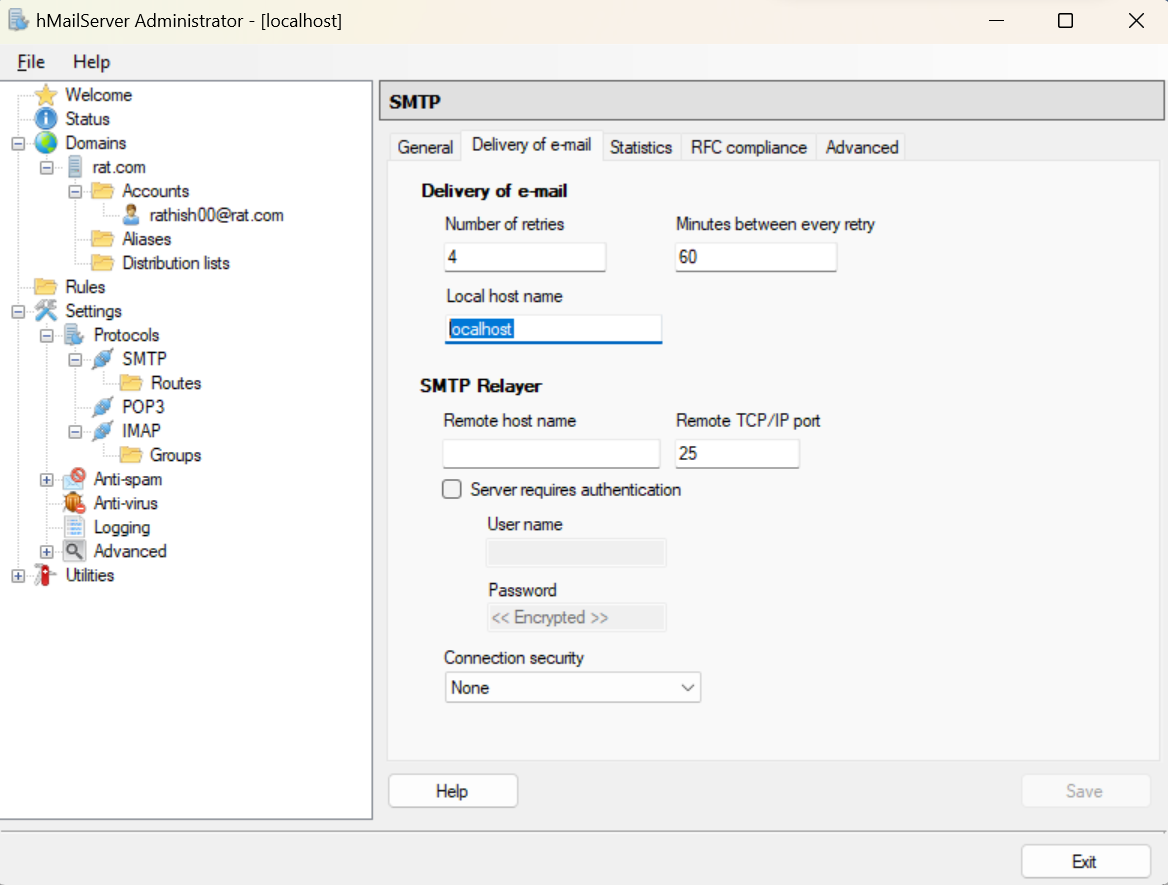


Fig. 5.3. Naming Local Host Name in SMTP

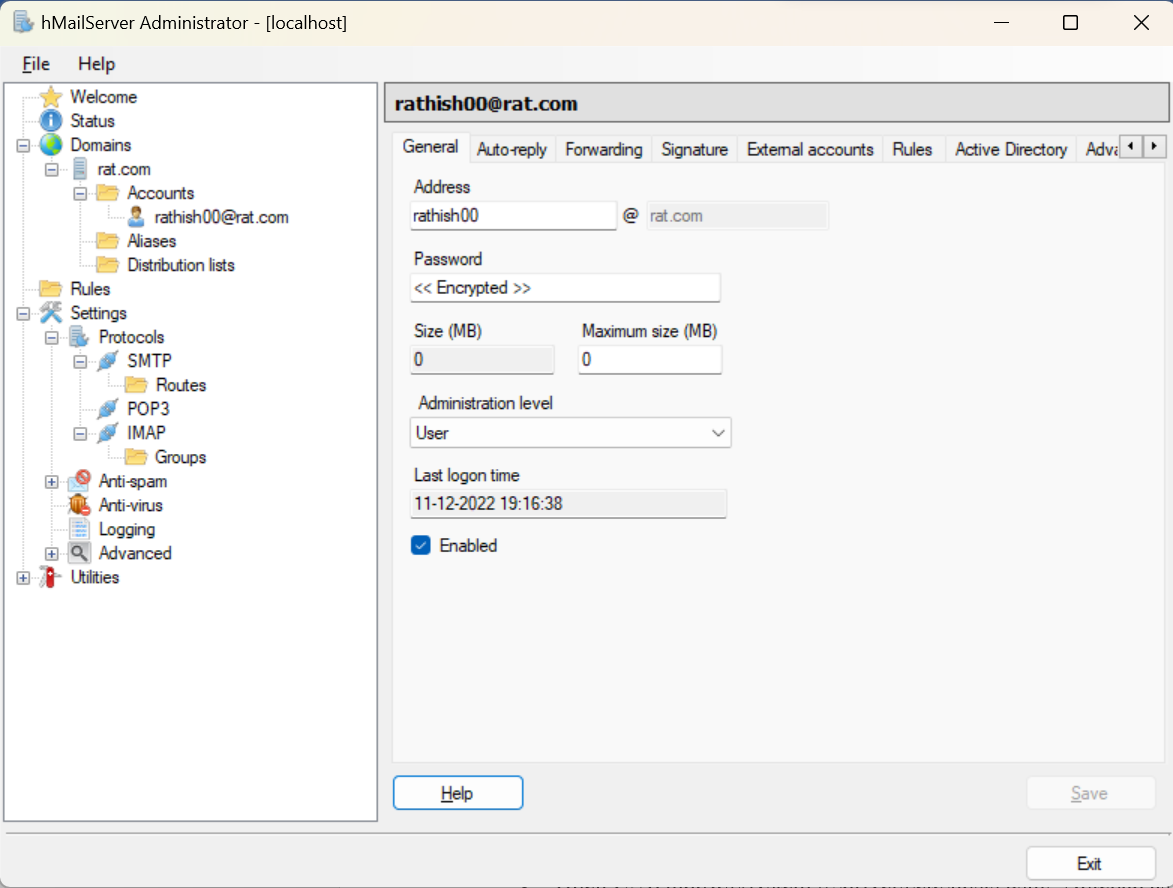


Fig. 5.4. Creating Accounts with Address and Password

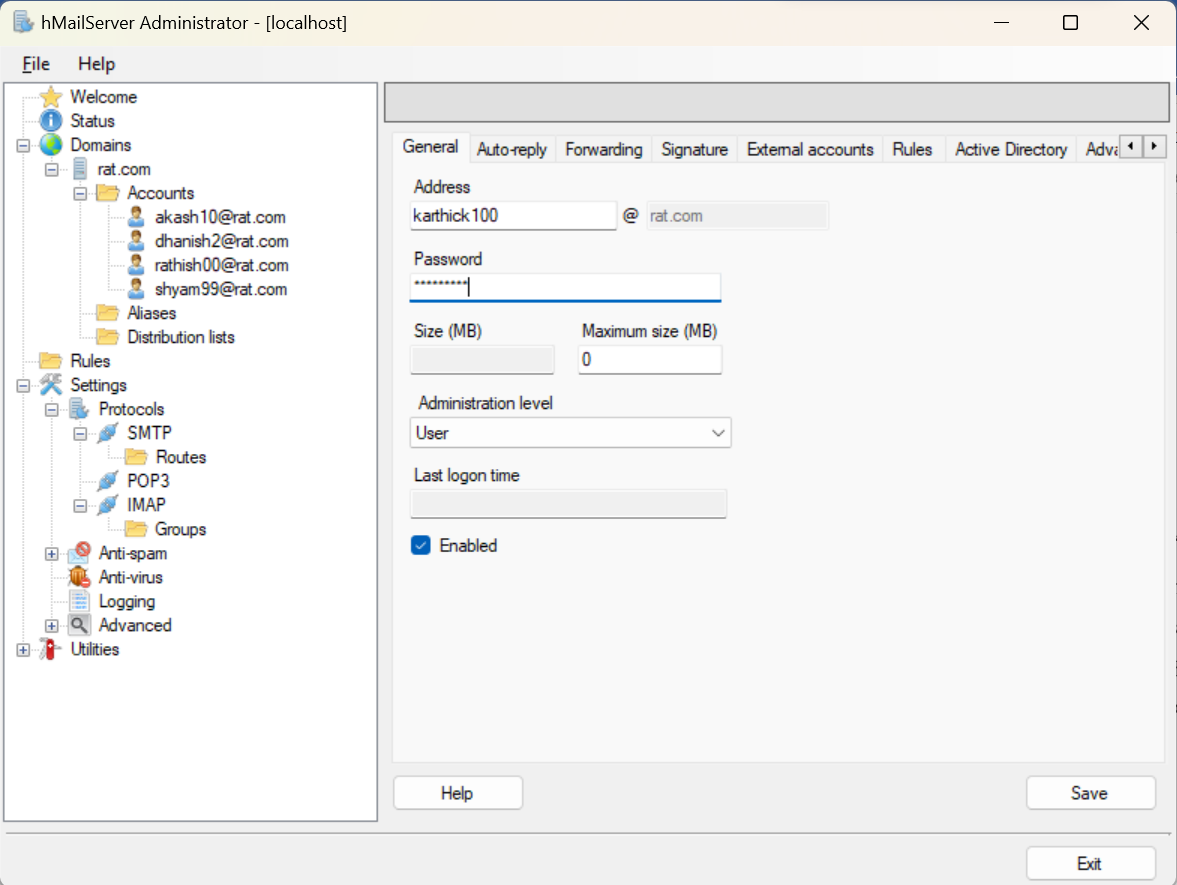


Fig.5.5. Creating Multiple Accounts

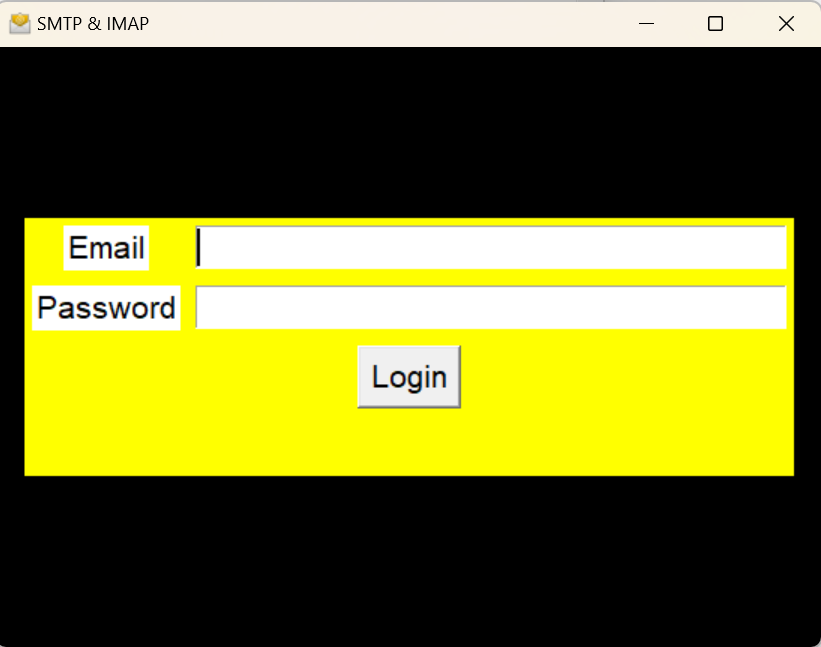


Fig. 5.6. Home Screen of Application

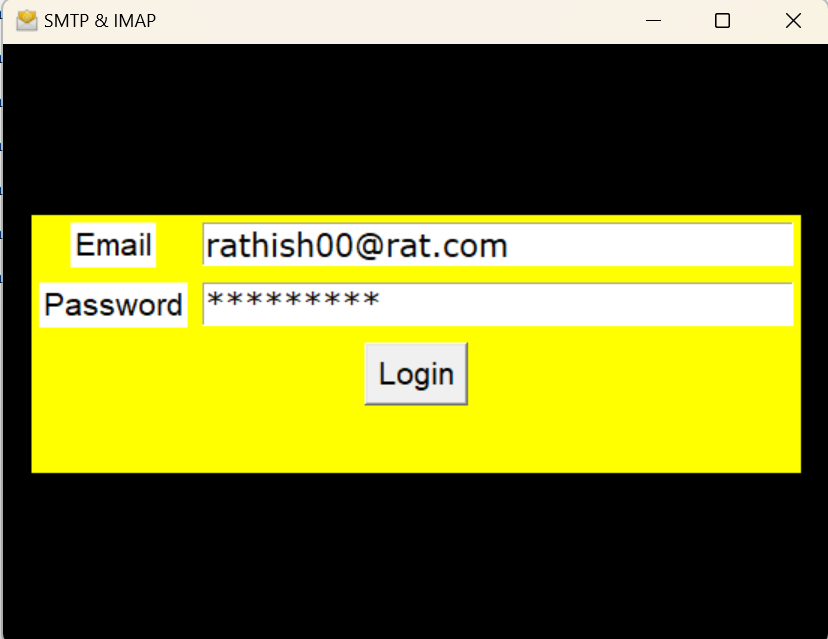


Fig.5.7. Login into the application using the Address and password entered before

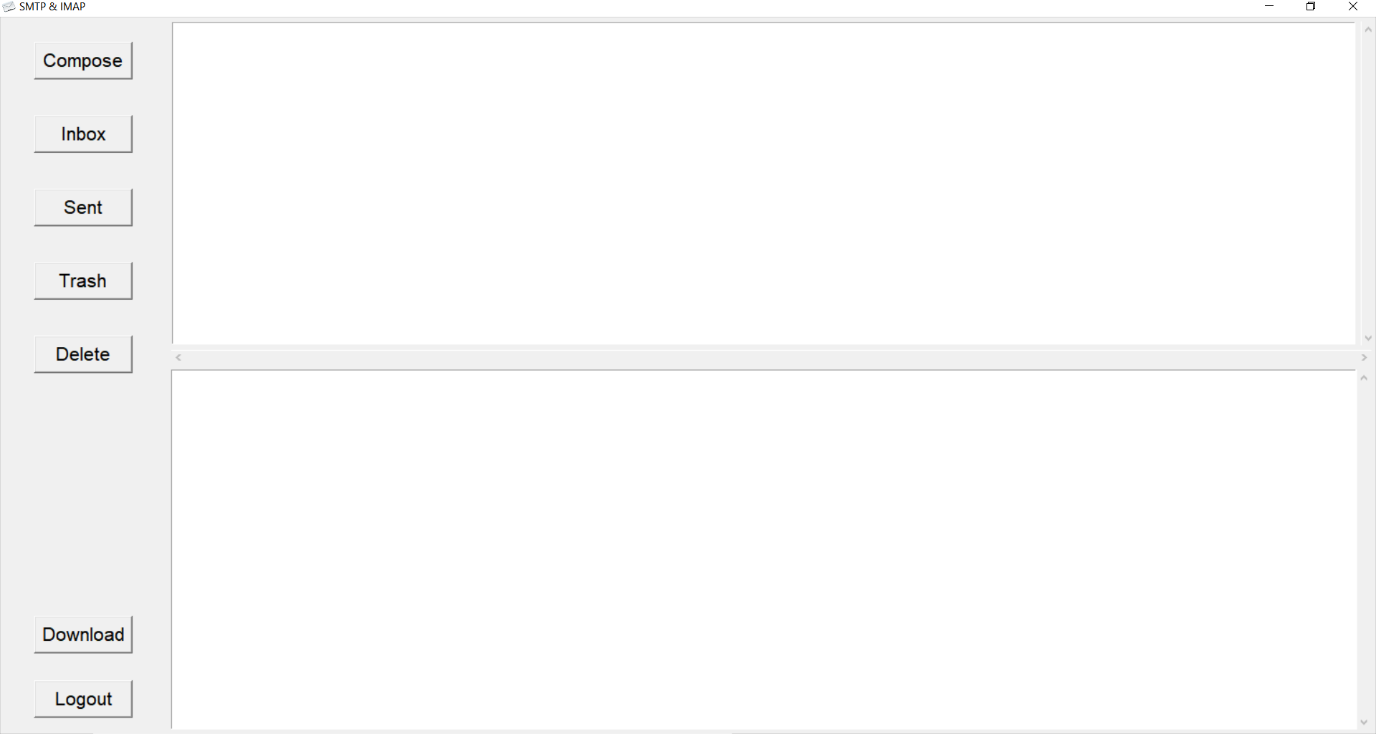


Fig.5.8. Application’s Home Screen

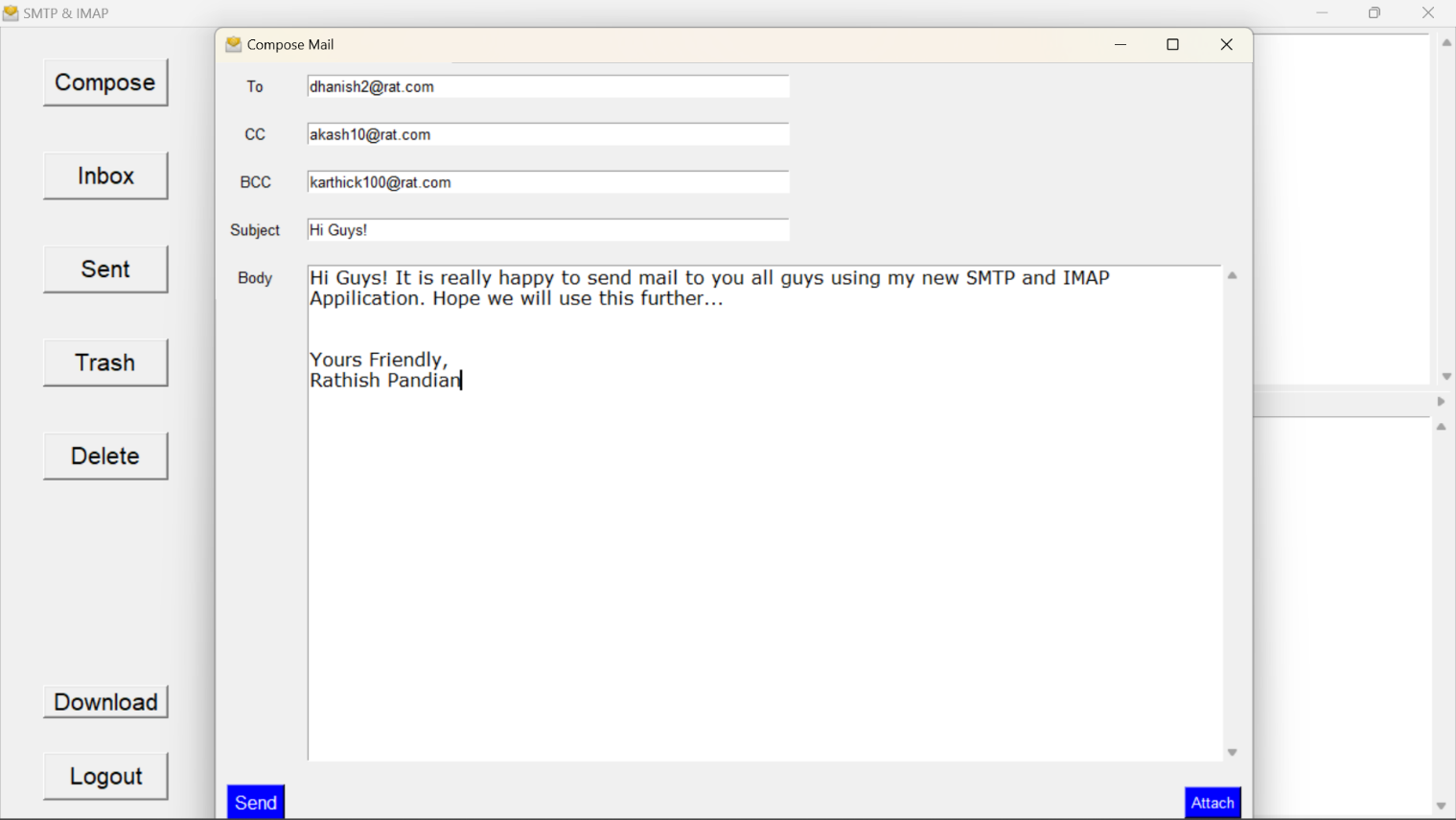


Fig. 5.7. Compose the mail screen in the Application

**CONCLUSION**

A basic Mail server and a domain has been created. This enabled us to visualize how a mail server works. The same method can be implemented on a large scale for a full-fledged application. SMTP & IMAP were used to send and retrieve emails respectively.

SMTP was used because of its monopoly over sending emails to MTAs. IMAP was used because of its features and since it preserves mail in the server too. Since a part of the TCP/IP protocol was used, we can ensure the security & reliability of packets that have been transmitted.

Future upgrades to the code will be done to enhance the experience of the user. While the application provides basic functions like sending, and reading emails, unique features like adding Labels, and adding Important folders are lacking. Also, from the custom domain mail address, emails can only be sent to or received from the same domain address. In the future, a dedicated VPS will be installed to ensure that the domain will be available publicly and emails can be sent to other domains too.

# REFERENCES

1. What is Simple Mail Transfer Protocol (SMTP)? –

<https://www.javatpoint.com/simple-mail-transfer-protocol>

1. What is an IMAP?

<https://www.techtarget.com/whatis/definition/IMAP-Internet-Message-Access-Protocol>

1. IMAP Vs POP3: Advantages & Disadvantages-

<https://www.reliconstech.com/blog/imap-vs-pop3-advantages-disadvantages/>

1. hMailServer Step-by-Step Installation Guide-Free SMTP server for Microsoft Windows-

<http://www.codingfusion.com/Post/hMailServer-Step-by-Step-Installation-Guide-Free-S>

1. Internet Mail Architecture, D. Crocker-

<https://www.rfc-editor.org/rfc/rfc5598>