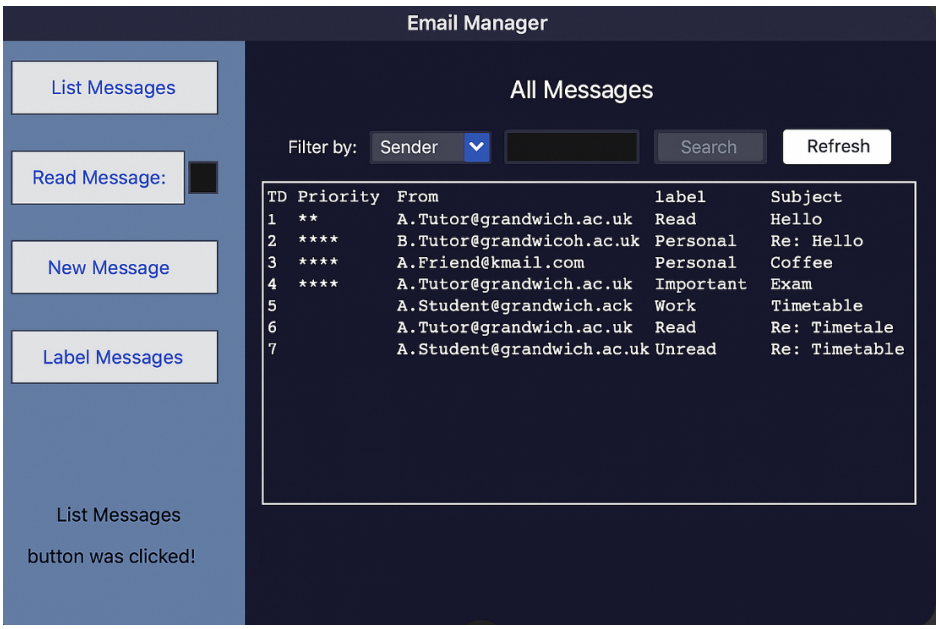


PROJECT REPORT OF EMAIL MANAGER APP

COMP1752 Coursework

Author:
Magdaline Jesica Gomes
Module Leader:
Nageena Frost
Date:02 April 2025
Word Count: 1929

Abstract:

This report includes a thorough description of building a project named Email Manager. The report consists of parts such as, the design and development process, error handling, testing of methods and classes, ideas of future innovations, personal reflection etc. References of research used to create and test the application is added at the end along with necessary screenshots of codes to provide a better understanding of the development process.

Table of Contents

Introduction:

Email Manager is Python-based application designed to run and manage emails through a user-friendly graphical user interphase (GUI) using Tkinter. The project provides essential simulation of email functionalities within a single GUI, making the navigation and handling of messages easy and efficient. Emails are introduced as messages throughout the design and layout. This project reuses code originally provided by Email Manager in Moodle.

The GUI opens with 700x500 pixels dimension containing a sidebar with four main buttons to navigate through pages (figure-1). The main features of the buttons are listed below:

- List Messages: Default page displaying a structured view of all received messages.
- Read Messages: Upon entering a message ID in the input field, this button allows users to open and view message content, with an option of changing priority for the selected message.
- New Message: This button leads to a simple page to compose and send new messages.
- Label Messages: Enables users to categorise messages with custom labels provided by a dropdown list for a categorized selection.

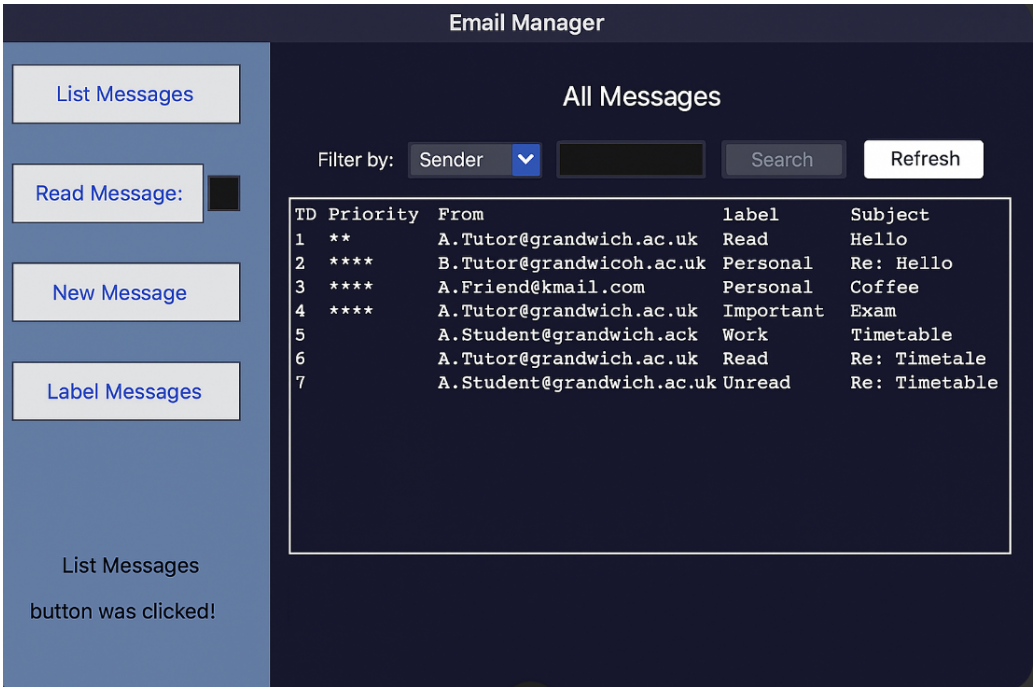


Figure-1: Default page of Email Manager

Design and Development:

Before designing the page layout, the functionalities of Email Manager have been studied. Although the code only provided the functionality of Read Message and List Messages button, it encouraged to design similar layout for New Message and Label Message button. More research on Tkinter library has been done before developing the codes (Bansal, 2017). Innovations that have been adopted to create this project are following:

Single GUI integration with filtering: One of the major differences between the enhanced Email Manager app and original one is Read Message button in provided Email Manager created a separate window through the class of ReadMessage, where the class ReadMessagesPage switches the page with similar layout through Tkinter frame in the enhanced simulation. The idea was to form integration of different pages to represent as a single GUI. The researches that have contributed here to implement the codes for EmailManagerApp class and create an interacting sidebar are GeeksforGeeks (2020) and TkinterHub (2022). Appendix A.2 displays the EmailManagerAPP Class that views a single GUI with multiple pages. Later, other classes have been modified in methods to inherit child class from tk.Frame and to have a hierarchical relationship with the main controller i.e. EmailManagerApp class.

The sidebar remains almost the same while visiting different other pages, though there are two signifying innovations, that can be noticed. The current active page would have a blue indicator on the side of the button and the footer in sidebar would be changed informing which button had been clicked as shown in figure-2.

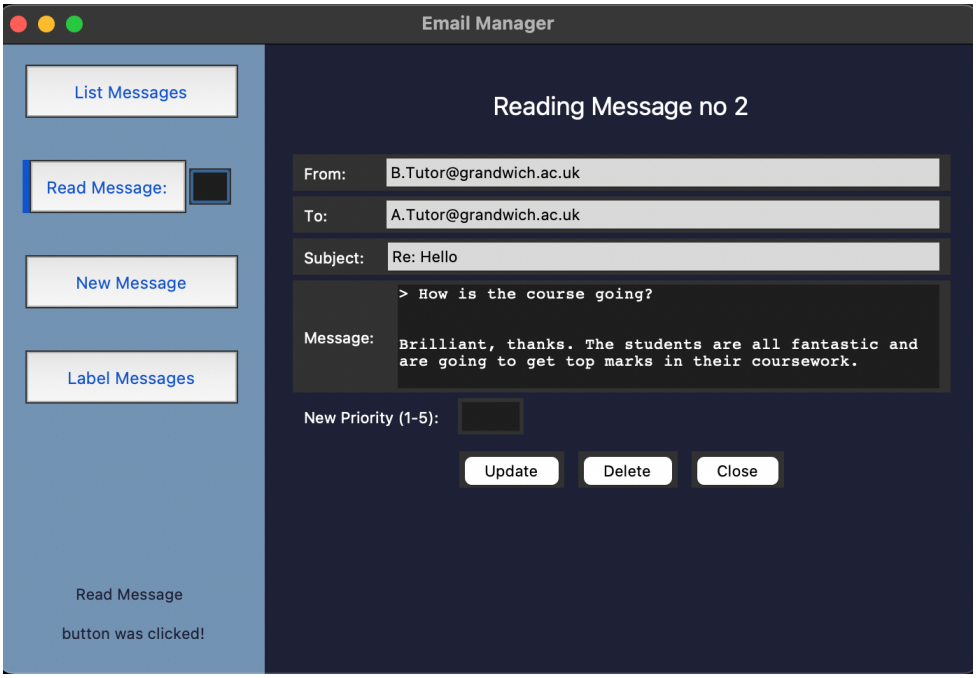


Figure-2: Switching to multiple frames in single GUI

Another major difference is the enhanced version uses pack( ) instead of grid( ) for geometry management, resulting a responsive GUI creation. That leads the GUI to appear as more eye appealing, when the user makes the window larger.

Storing messages in CSV:

The messages are saved through a file with Comma Separated Values Format (CSV). Letter on in the file named “message_manager.py” new functions have been created and modified to initialise, load and save messages to “messages.CSV”. The structure how the messages are to be saved are reused with the given file “message.py”.

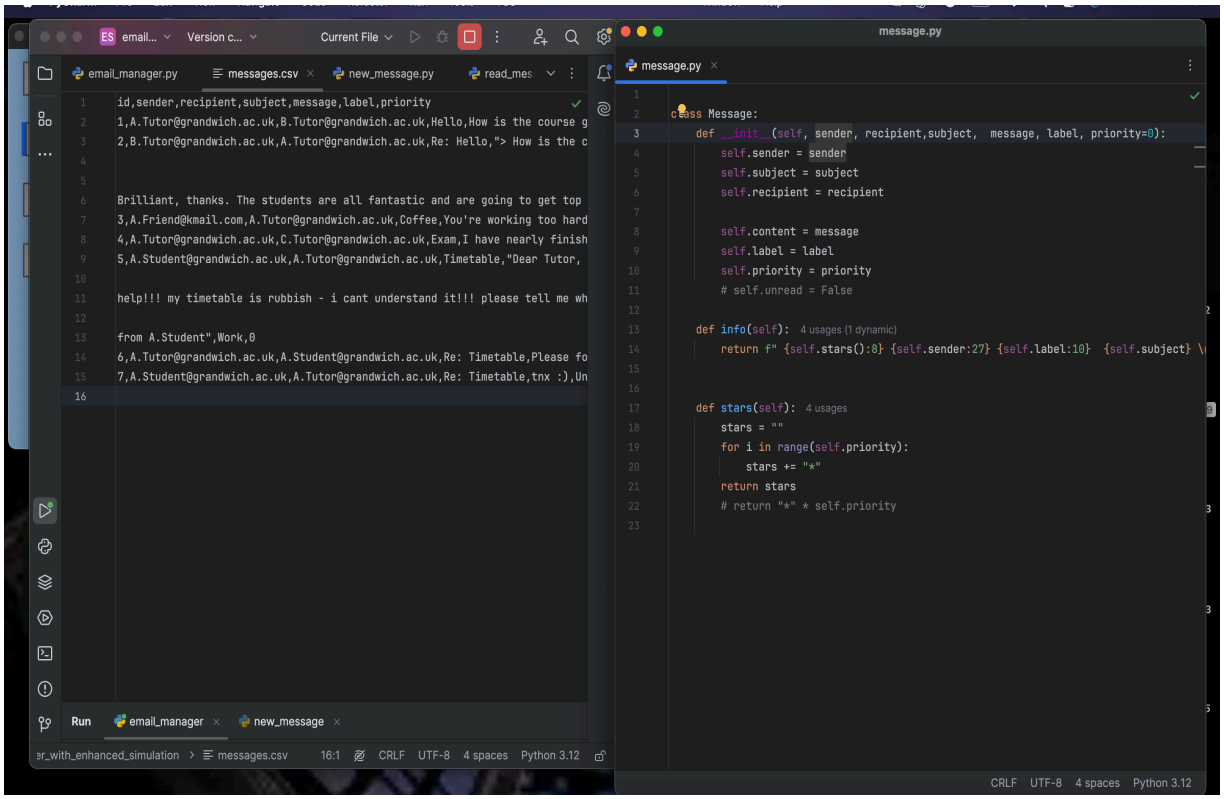


Figure-3: Messages.csv and message.py file to save and display messages

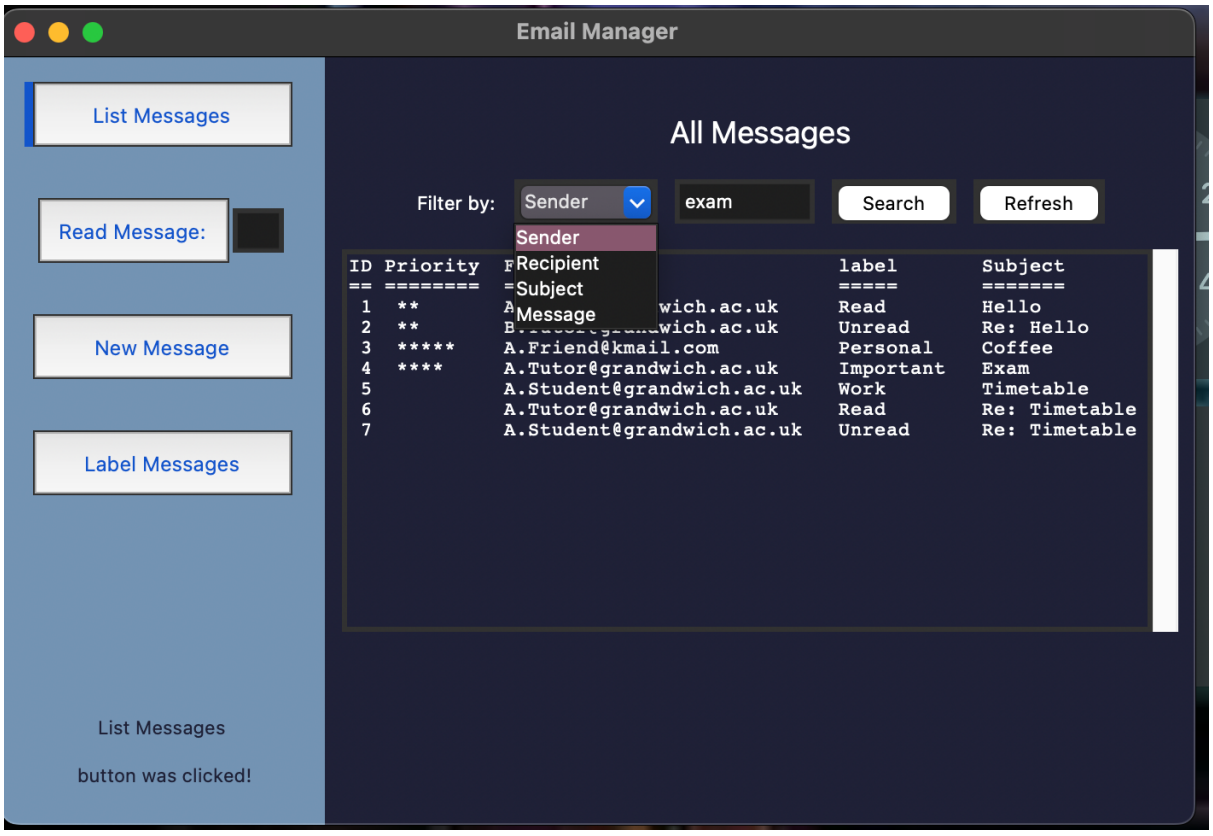
Innovations for Read Message:

The ReadMessagesPage class reuses the codes from ReadMessage class of Email Manager, but changes have been made to adopt to GUI appearance (Displayed in appendix- A.1). Simple innovation is that it displays the message ID that is being read as “**Reading Message no {message ID}”** (displayed in figure-2). The priority can be updated with inserting a value in priority box, then clicking **Update** button. The **Delete** button deletes the current message that is displayed. It also asks for confirmation to ensure the deletion, while confirming with a messagebox when deleted. After the deletion and when the **Close** button is clicked it leads to default ListMessagesPage frame. The input value in the side bar and priority field in

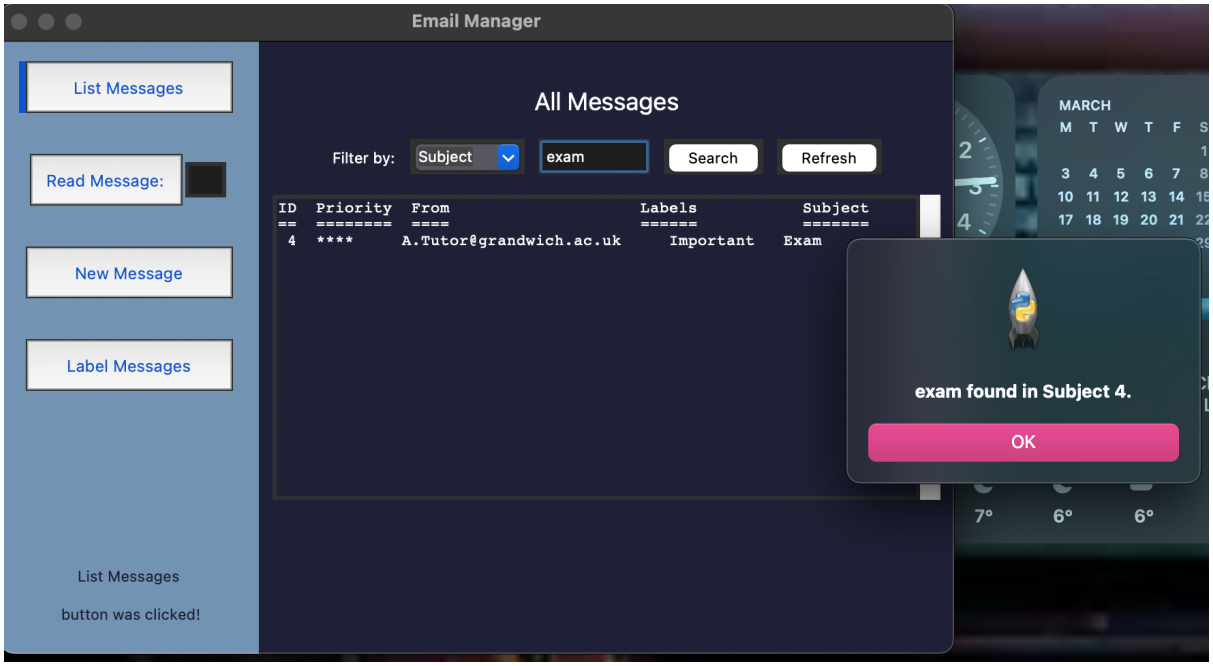
ReadMessagesPage frame is cleared every time any button is clicked, and any message is displayed.

Innovation in default page:

In both versions of Email Manager the default page displays List Messages, with a structure of messages with their ID no, priority displayed as stars, sender's email address, designated label and subject as shows in figure-1. However, innovation has been made with adding a filter option which helps to search any keyword that is contained by any sections of the message i.e. sender, recipient, subject and message content. The **Search** button is disabled in the default page. It only enables when a text or number or character has been inserted (figure-4). When the **Search** button is clicked it displays the messagebox confirming the location where it has found the keyword and displays the message with the structure of figure-4, so that user can easily navigate to locate the message. On the other hand, if the section doesn't contain the keyword, it is updated by another messagebox and the message list remains unchanged. **Refresh** button helps viewing all the saved messages.



(Figure-4: Filtering through messages)



(Figure-5: Displaying filtered messages)

Design and features of New Message:

Title label of the NewMessagePage class "Compose a New Message", which is interacted through New Message button, provides an idea of the functionality of New Message page. The design layout is like Read Message page, with fields for sender, recipient, subject, message content and priority. The buttons i.e., **Send** and **Close** however are different with functionalities. **Send** button helps sending the message and saving it in the CSV file. After sending and clicking the **Close** button, the default message page is triggered to view all messages, which is updated by the new message as well. Figure-6 represents the layout for New Message page.

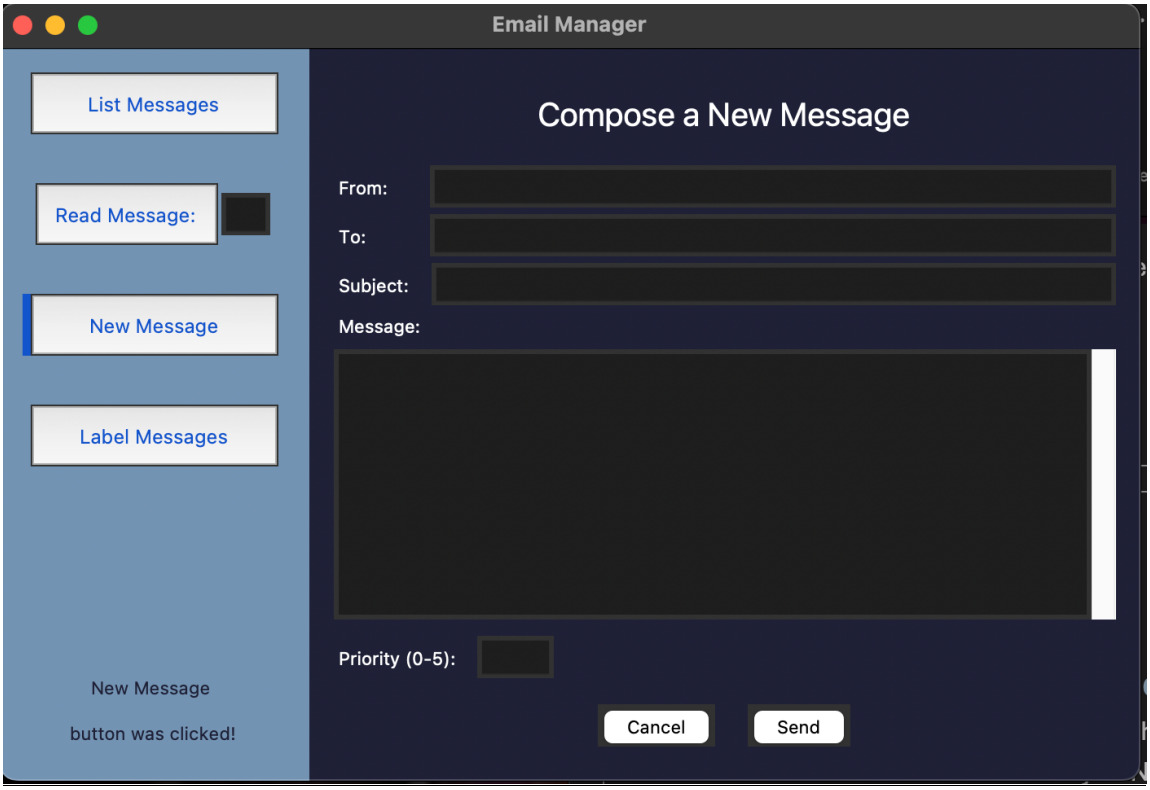


Figure-6: New Message frame layout with send functionality

Design and features of Label Messages:

Another frame of Email Manager GUI is Label Messages, which lets user to search the messages according to the dropdown list, when the **List Messages** button is clicked. To add a new label user needs to select label and input message ID, then click the button **Add Label** (figure-7). The **Back** button helps viewing List Messages page.

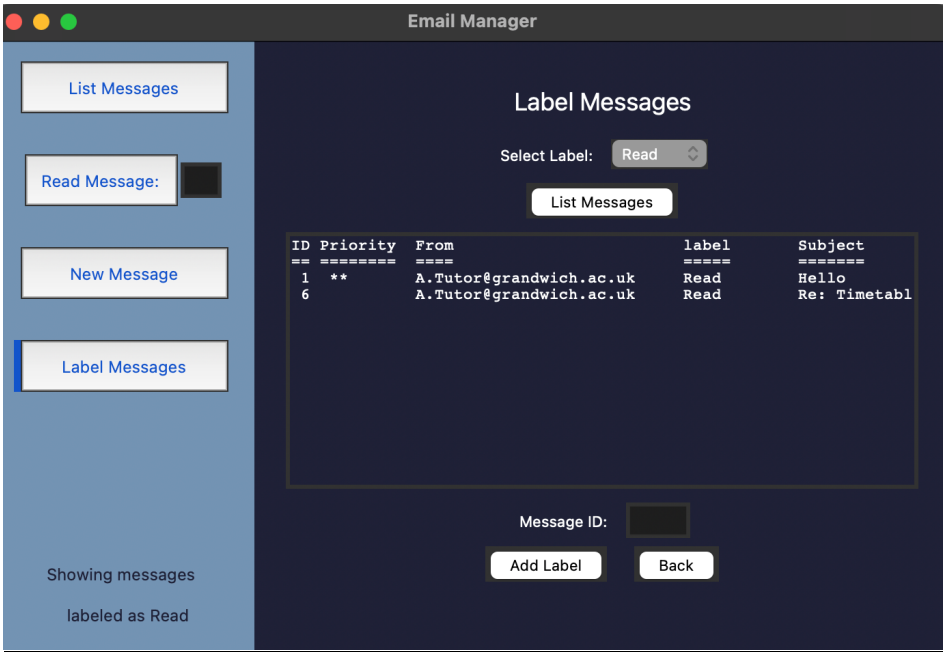


Figure-7: Design Label Message page

However, if the label doesn't exist, the message box appears to update, and the GUI appearance changes like figure-8.

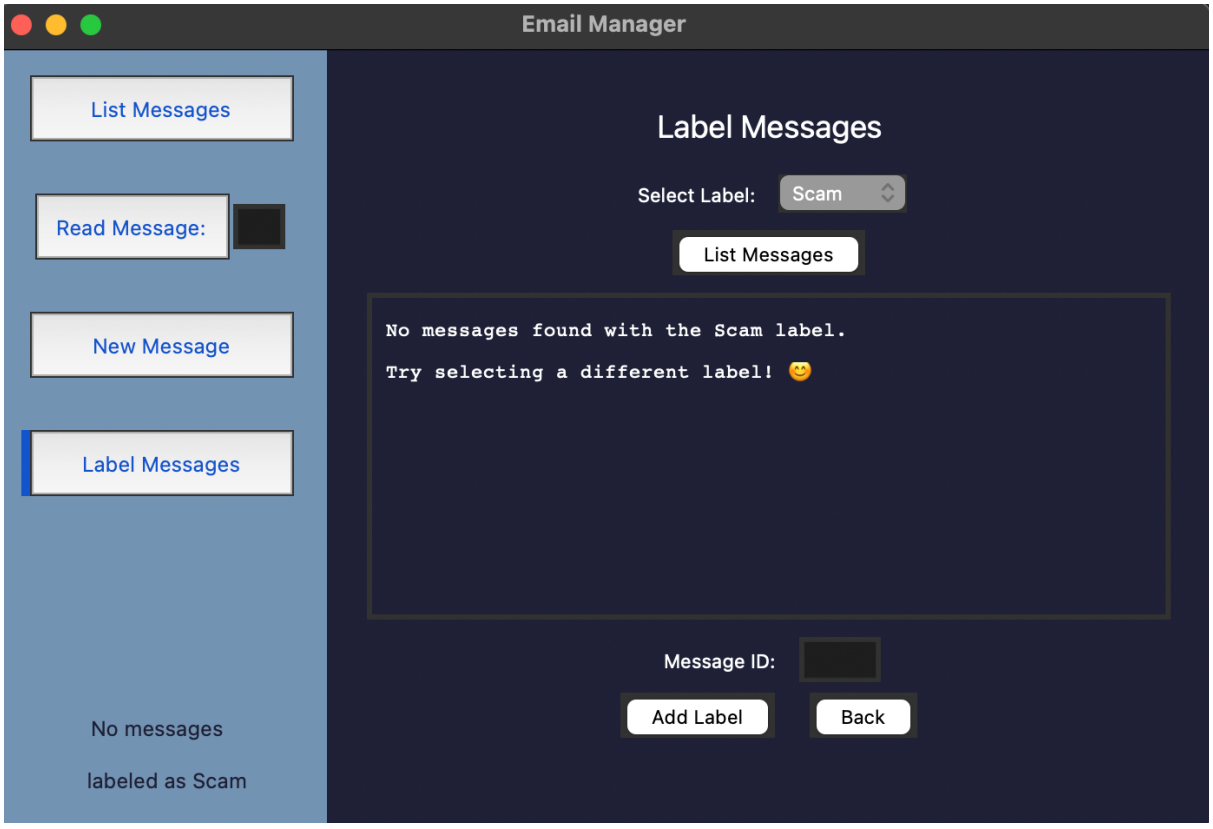


Figure-8: Gui appearance for not finding searched label

Unlike List Messages page where Tkinter Combobox has been used for drop down list, Label Message page uses Tkinter OptionMenu to view the labels. These differences have enabled the appearance of design for both drop down list unique.

Error Handling:

1) Read Messages Page: The ID field and priority field has been modified to reject invalid number figures. These display error message identifying the cause of rejection to use the corresponding functionality.

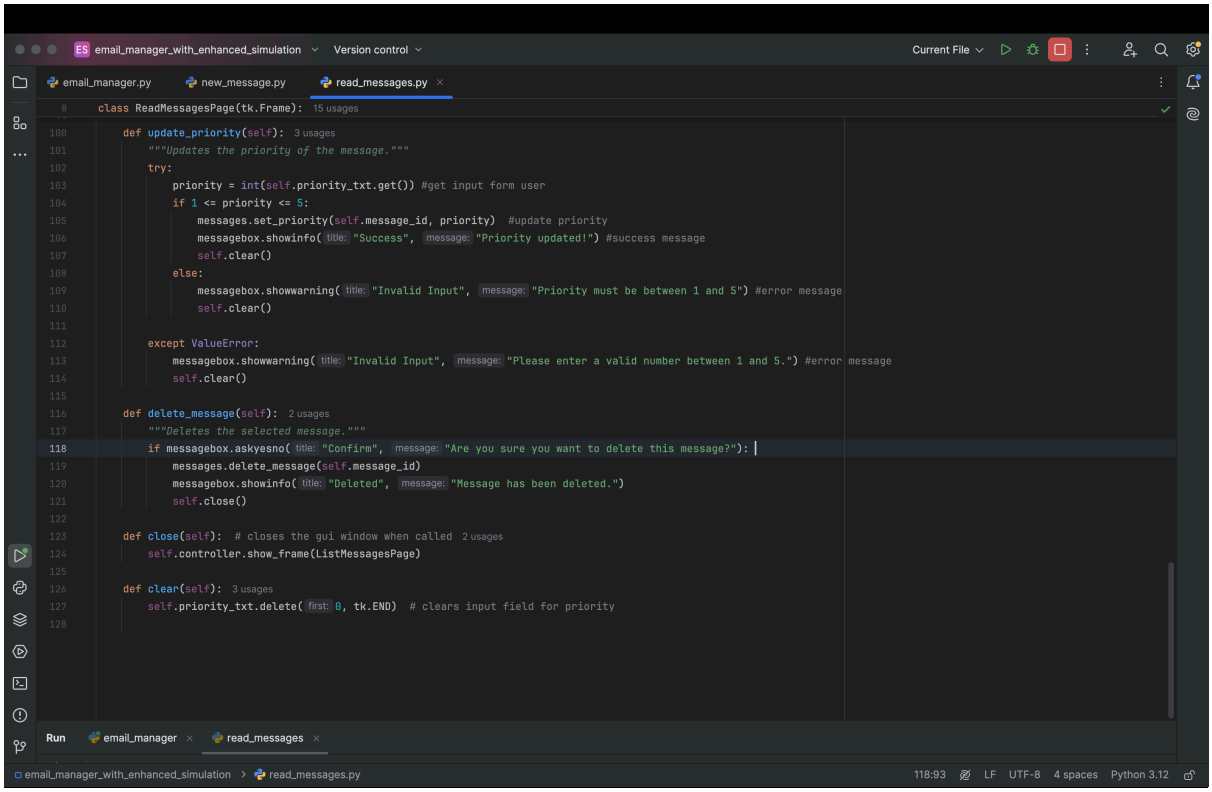


Figure-9: Error handling in ReadMessagesPage class

2) List Messages Page: The **Filter by** option is controlled by two functionalities: the dropdown list of sections that limits the scope of searching and the input value handling, which is done by disabling **Search** button when the input field is empty or only spaces has been put as keywords.

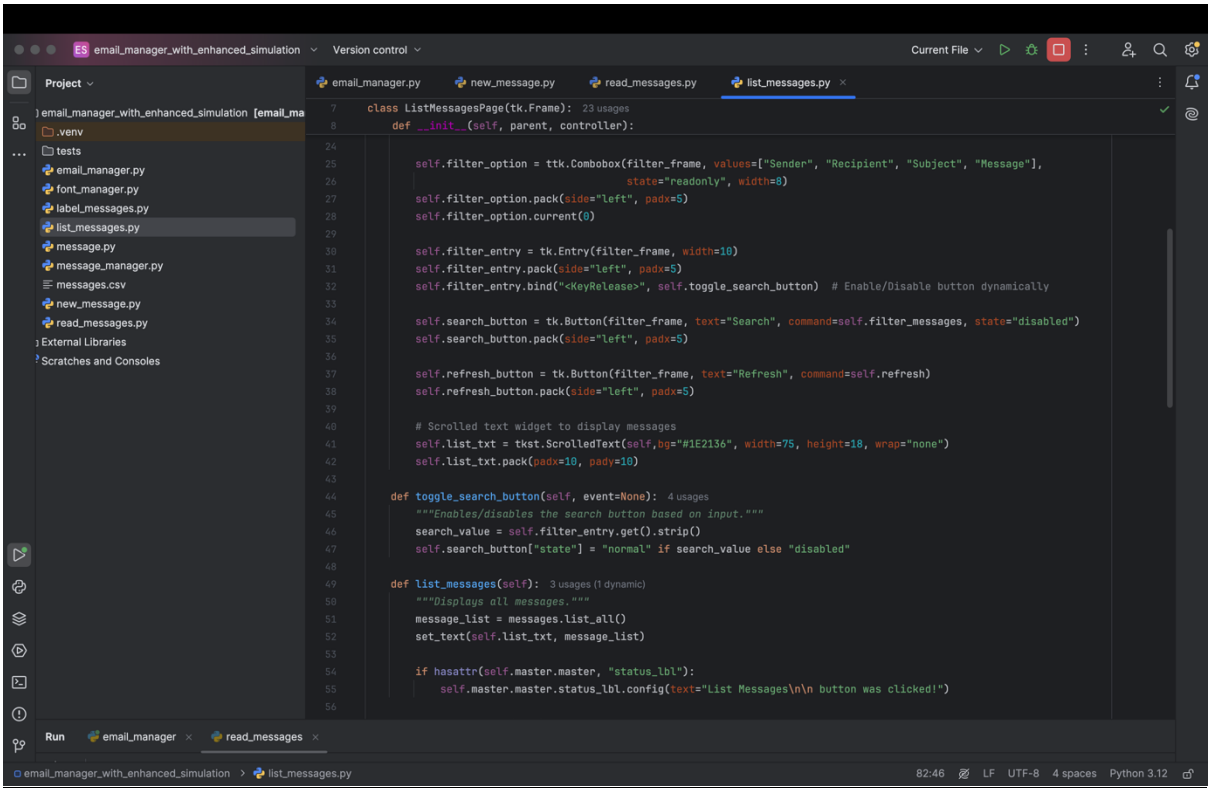


Figure-10: ListMessagesPage dropdown list for filter and Search button handling

3) New Message Page: The fields of New Message Page are validated to reject empty values in order to send the message. The email address fields also use regular expression (re), which then compares the input with “email_regex” variable to validate email. The variable is adopted from Stack Overflow (2008) queries.

Figure-11: Regular Expression to validate email

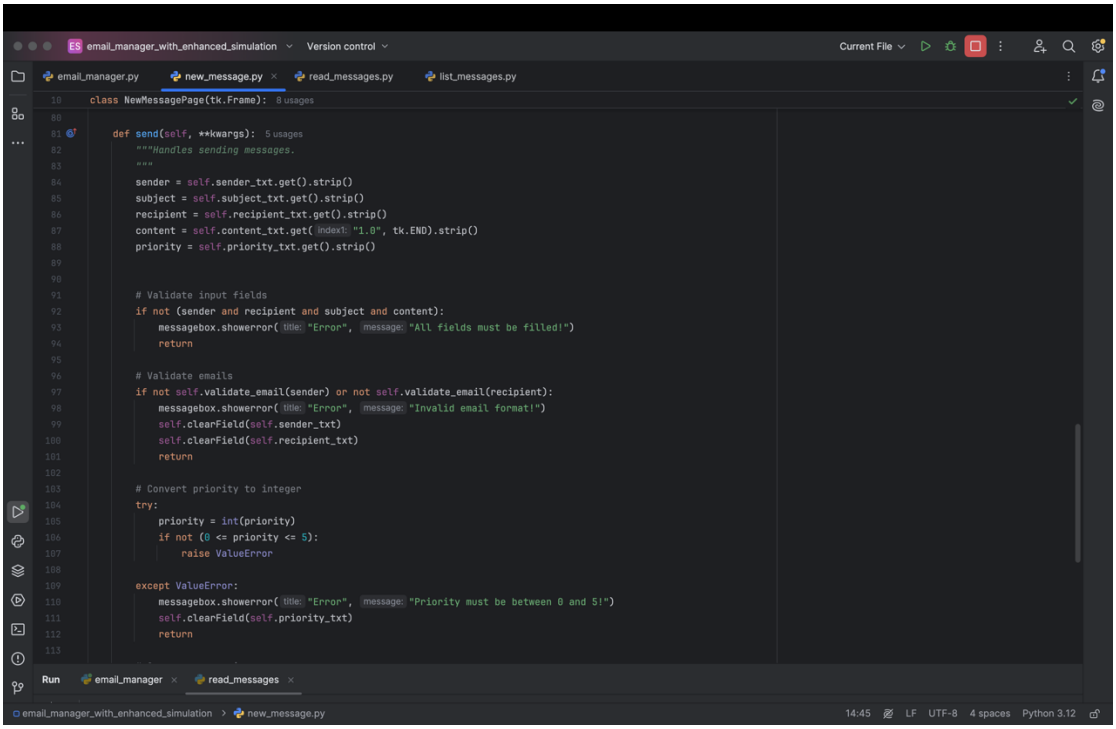


Figure-12: Error handling in NewMessagePage

4) Label Messages Page: While adding the label possible error of string input and invalid ID input has been handled by message box appearance. The “list _txt” field has been disabled for any user input, to prevent unauthorised changes and protect information.

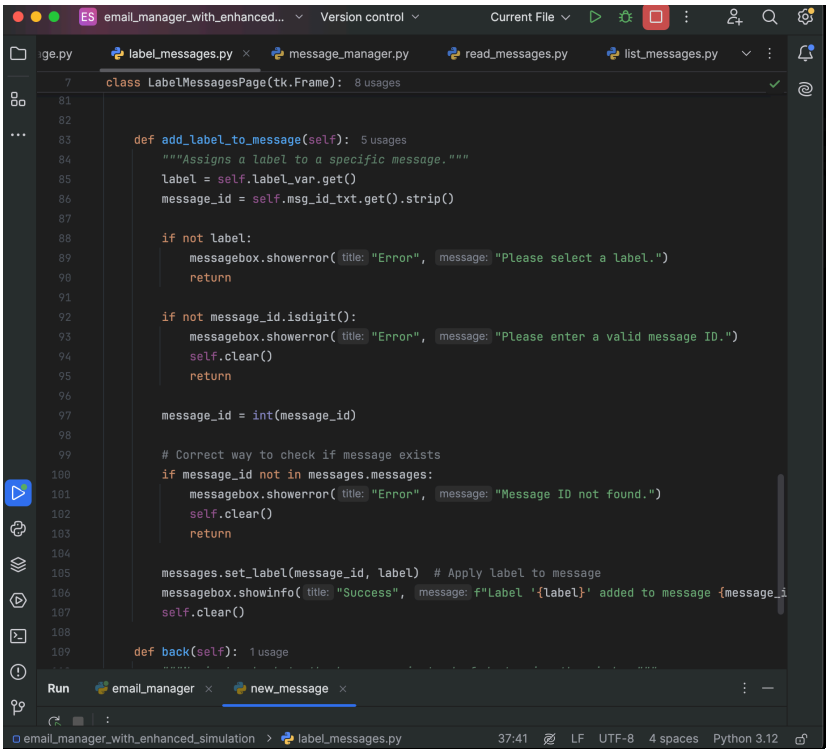


Figure-13: LabelMessagesPage class error handling

Testing and Faults:

To test the functionalities Unit test has been completed with each of the classes of Email Manager GUI, through PyTest. The tests check whether the UI functions properly, i.e., page navigation, button interactions, error handling and sidebar interactions are smoothly handled. There has been a test for each method or major functions of classes. While writing the test functions, assertIN( ) and assertequal( ), mock( ), patch( ) and fixture have been used from unittest library () (Python Software Foundation, 2025) (GeeksforGeeks, 2017) (Krekel and Pytest-dev Team, 2015).

Although all the functionalities are working fine and most of the test results has passed, test of NewMessagePage class for sending a message with valid input has failed (test tables are provided in appendix-B). The error encountered is, to send a message successfully new_message( ) function from message_manager needs to be called in NewMessagePage , but while testing the following function was called 0 times as shown as figure-14.

```
test_new_message.py:93:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

self = <MagicMock name='new_message' id='4386825152'>
args = ('sender@example.com', 'recipient@example.com', 'Test subject', 'Test message content.', 'unread', 3)
kwargs = {}, msg = "Expected 'new_message' to be called once. Called 0 times."

    def assert_called_once_with(self, /, *args, **kwargs):
        """assert that the mock was called exactly once and that that call was
        with the specified arguments."""
        if not self.call_count == 1:
            msg = ("Expected '%s' to be called once. Called %s times.%s"
                   % (self._mock_name or 'mock',
                      self.call_count,
                      self._calls_repr()))
>           raise AssertionError(msg)
E           AssertionError: Expected 'new_message' to be called once. Called 0 times.

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/unittest/mock.py:958: AssertionError

Process finished with exit code 1
```

(Figure-14: Error for Test_send_with_valid_input)

As a result to debug, a print has been performed in new_message( ) function, which prints sender variable for every time the function is called and sends the message. However, it pointed that the function is called, even if the test failed resulting a new message sent and saved effectively.

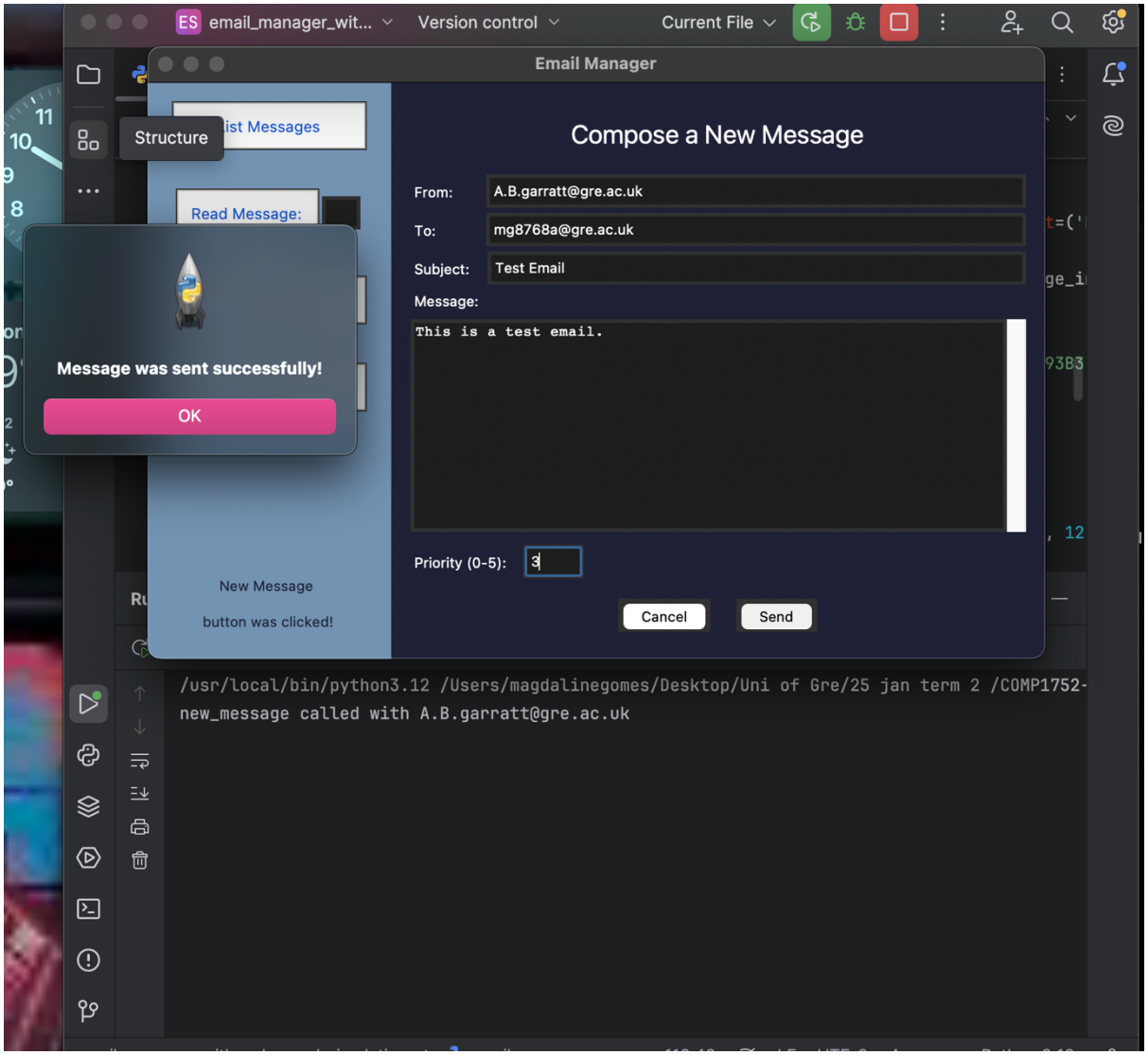


Figure-15: Debugging new_message( ) function

Future development:

In order to increase user experience the GUI can be modified with message preview. Message class needs to be updated in order to apply that, and use of Tkinter in the following can increase readability in GUI. Also in future a sub class of Message can be created to attach media files or documents in the messages. The project is quite similar to Email Applications such as EMail, GMail and Yahoo Mail. It would be fascinating if the send method of NewMessagePage class can be modified to adopt smtplib module of Python, so it can send real life emails through log in. Recording date and time can track the messages schedule for the user. Moreover, instead of Read Message button on the side, it would be very pleasing to get the messages opened by simply clicking on them displayed in the message list.

Reflection:

This project has been a highly valuable experience that allowed to apply theoretical knowledge in a practical setting. The module structure throughout the course helped to build technical and analytical skills, deepening the understanding of object-oriented programming in Python, software testing, and user interface (UI) design. A major learning outcome was understanding of event-driven programming in Tkinter. Structuring the application to handle UI updates dynamically seemed complex, especially when retrieving messages from CSV file and display them. This was due to building a Python GUI application without prior experience. On the other hand, interacting the classes to imply functionalities was quite enjoyable. It has increased creativity in writing neat code.
Another key area of growth is enhancing proficiency in unit testing using Pytest's unittest framework. This skill will be highly beneficial in real-world software development, where automated testing is crucial for maintaining robust applications.
Overall, this coursework project had the structural guideline to help developing design ideas, critical thinking and problem solving leading to skills that is required in future career.

Conclusion:

Enhancing the functionality of Email Manager application has encouraged to create larger scope for innovation and exception handling in Python object-oriented programming. This well-functioning GUI can be the baseline of any messaging application. The project itself has broadened the knowledge of Python Tkinter and Pytest Unittest library. Nowadays when Python is used almost for everything including, AI simulation, data science, web development etc sectors, this project has been a great start to explore the Python development scope.

References:

Bansal, R. (2017) Python GUI - tkinter - GeeksforGeeks, *GeeksforGeeks*, [online] Available at: https://www.geeksforgeeks.org/python-gui-tkinter/ (Accessed 4 March 2025).

GeeksforGeeks (2017) Unit Testing in Python - Unittest, *GeeksforGeeks*, [online] Available at: https://www.geeksforgeeks.org/unit-testing-python-unittest/ (Accessed 27 March 2025).

GeeksforGeeks (2020) Tkinter Application to Switch Between Different Page Frames, *GeeksforGeeks*, [online] Available at: https://www.geeksforgeeks.org/tkinter-application-to-switch-between-different-page-frames/?ref=lbp (Accessed 9 March 2025).

Krekel , H. and Pytest-dev Team (2015) About fixtures — pytest documentation, *docs.pytest.org*, [online] Available at: https://docs.pytest.org/en/stable/explanation/fixtures.html (Accessed 27 March 2025).

PythonSoftwareFoundation (2025) unittest.mock — mock object library, *Python documentation*, [online] Available at: https://docs.python.org/3/library/unittest.mock.html#patch (Accessed 27 March 2025).

StackOverflow (2008) regex - How can I validate an email address using a regular expression?, *Stack Overflow*, [online] Available at: https://stackoverflow.com/questions/201323/how-can-i-validate-an-email-address-using-a-regular-expression (Accessed 6 March 2025).

TkinterHub (2022) Switch Pages in Tkinter | Switch Frames in Tkinter | Switch Multiple Pages in Tkinter, *www.youtube.com*, [online] Available at: https://www.youtube.com/watch?v=95tJO7XJlko (Accessed 7 March 2025).

Appendices:

A. The commented version of code:

A.1: The commented version of ReadMessagesPage:

```
#
# import tkinter as tk
# from tkinter import messagebox
# import message_manager as messages  # Custom module for managing messages
# from list_messages import ListMessagesPage
#
#
# class ReadMessagesPage(tk.Frame):
#     def __init__(self, parent, controller, message_id):
#         super().__init__(parent) #parent tkinter widget
#         self.controller = controller #refrence to the main application
controller for navigation
#         self.message_id = message_id  # Store message ID
#         # Set background color for the frame
#         self.config(bg="#1E2136")
#         # title label for the page
#         tk.Label(self, bg="#1E2136",text=f"Reading Message no
{message_id}", font=("Bold", 18)).pack(pady=10)
#
#         # Validate and fetch message data
#         self.message = messages.get_message_by_id(self.message_id)
#
#         # handling if the message id doesn't exist
#         if not self.message:
#             self.controller.show_frame(ListMessagesPage)
#             return
#
#         """Creates the UI elements for displaying message details."""
#         # Create a frame for the label and entry to be side by side
#         container = tk.Frame(self)
#         container.configure(bg="#1E2136")
#         container.pack(padx=10, pady=10, fill="both", expand=True)
#
#         # Update status label
#         if hasattr(self.master.master, "status_lbl"):
#             self.master.master.status_lbl.config(text="Read Message \n\n
button was clicked!")
#
#         # From field
#         from_frame = tk.Frame(container)
#         from_frame.pack(fill="x", padx=10 ,pady=2)
#
#         tk.Label(from_frame, text="From:    ").pack(side="left", padx=5)
#         self.sender_txt = tk.Entry(from_frame, width=24,
readonlybackground='#d9d9d9', fg='#000000')
#         self.sender_txt.pack(side="left", fill="x", expand=True, padx=5)
#         self.sender_txt.insert(tk.END, self.message.sender)
#         self.sender_txt.configure(state="readonly")
```

```
#
#         # To field
#         to_frame = tk.Frame(container)
#         to_frame.pack(fill="x",padx=10, pady=2)
#
#         tk.Label(to_frame, text="To:        ").pack(side="left", padx=5)
#         self.recipient_txt = tk.Entry(to_frame,
width=24,readonlybackground='#d9d9d9', fg='#000000')
#         self.recipient_txt.pack(side="left", fill="x", expand=True,
padx=5)
#         self.recipient_txt.insert(tk.END, self.message.recipient)
#         self.recipient_txt.configure(state="readonly")
#
#         # Subject field
#         subject_frame = tk.Frame(container)
#         subject_frame.pack(fill="x",padx=10, pady=2)
#
#         tk.Label(subject_frame, text="Subject:").pack(side="left",
padx=5)
#         self.subject_txt = tk.Entry(subject_frame,
width=24,readonlybackground='#d9d9d9', fg='#000000')
#         self.subject_txt.pack(side="left", fill="x", expand=True, padx=5)
#         self.subject_txt.insert(tk.END, self.message.subject)
#         self.subject_txt.configure(state="readonly")
#
#         # Message field (multi-line text box)
#         message_frame = tk.Frame(container)
#         message_frame.pack(fill="x",padx=10,pady=2)
#
#         tk.Label(message_frame, text="Message:").pack(side="left",
padx=5)
#         self.content_txt = tk.Text(message_frame, width=40, height=6,
wrap="word")
#         self.content_txt.pack(side="left", fill="x", expand=True, padx=5)
#         self.content_txt.insert(tk.END, self.message.content)
#         self.content_txt.configure(state="disabled")
#
#         # Priority field
#         priority_frame = tk.Frame(container)
#         priority_frame.pack(fill="x", padx=10, pady=2)
#         priority_frame.configure(bg="#1E2136")
#
#         tk.Label(priority_frame,bg="#1E2136", text="New Priority (1-
5):").pack(side="left", padx=5)
#         self.priority_txt = tk.Entry(priority_frame, width=5)
#         self.priority_txt.pack(side="left", padx=5)
#
#         # Buttons (Update, Delete, Close)
#         button_frame = tk.Frame(container)
#         button_frame.pack(pady=10)
#         button_frame.configure(bg="#1E2136")
#         update_btn = tk.Button(button_frame, text="Update",
command=self.update_priority)
#         update_btn.pack(side="left", padx=5)
#
#         delete_btn = tk.Button(button_frame, text="Delete",
command=self.delete_message)
#         delete_btn.pack(side="left", padx=5)
#
#         close_btn = tk.Button(button_frame, text="Close",
command=self.close)
```

```
#         close_btn.pack(side="left", padx=5)
#
#
#
#     def update_priority(self):
#         """Updates the priority of the message."""
#         try:
#             priority = int(self.priority_txt.get()) #get input form user
#             if 1 <= priority <= 5:
#                 messages.set_priority(self.message_id, priority)  #update
priority
#                 messagebox.showinfo("Success", "Priority updated!")
#success message
#                 self.clear()
#             else:
#                 messagebox.showwarning("Invalid Input", "Priority must be
between 1 and 5") #error message
#                 self.clear()
#
#         except ValueError:
#             messagebox.showwarning("Invalid Input", "Please enter a valid
number between 1 and 5.") #error message
#             self.clear()
#
#     def delete_message(self):
#         """Deletes the selected message."""
#         if messagebox.askyesno("Confirm", "Are you sure you want to
delete this message?"):
#             messages.delete_message(self.message_id)
#             messagebox.showinfo("Deleted", "Message has been deleted.")
#             self.close()
#
#     def close(self):  # closes the gui window when called
#         self.controller.show_frame(ListMessagesPage)
#
#     def clear(self):
#         self.priority_txt.delete(0, tk.END)  # clears input field for
priority
```

A.2: The commented version of EmailManagerApp:

```
import tkinter as tk
# from tkinter import messagebox #messagebox for displaying alerts
# import font_manager as fonts # Manages fonts
# from list_messages import ListMessagesPage
# from read_messages import ReadMessagesPage
# from new_message import NewMessagePage
# from label_messages import LabelMessagesPage
#
#
# class EmailManagerApp(tk.Tk):
#     def __init__(self):
#         super().__init__()
#         self.status_lbl = None
#         self.geometry('700x450')
#         self.title('Email Manager')
#
#         # Sidebar Frame (Navigation)
#         self.options_frame = tk.Frame(self, bg='#7393B3', width=190,
height=400)
#         self.options_frame.pack(side=tk.LEFT, fill=tk.Y)
#         self.options_frame.pack_propagate(False)
```

```
#
#         # Main content frame (for switching pages)
#         self.main_frame = tk.Frame(self, bg='#1E2136',width=450,
height=400)
#         self.main_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
#         self.main_frame.pack_propagate(False)
#
#         # Sidebar Buttons
#         self.create_sidebar()
#
#         # Show the default frame
#         self.show_frame(ListMessagesPage)
#
#     # -----------------------------------------
#     def show_frame(self, page_class):
#         """Switches to the selected page by creating a new instance."""
#
#         # Retrieve message ID before destroying widgets (if required)
#         if page_class == ReadMessagesPage:
#             message_id = self.id_txt.get().strip()  # Get message ID from
entry
#             if not message_id.isdigit():
#                 tk.messagebox.showerror("Error", "Please enter a valid
Message ID.")
#                 self.id_txt.delete(0, tk.END)  # clear invalid entry
#
#                 return  # Prevents loading an invalid page
#             message_id = int(message_id)
#             self.id_txt.delete(0, tk.END)  # Clear entry after retrieval
#
#         else:
#             message_id = None  # Other pages don't use message_id
#
#         # Destroy old page
#         for widget in self.main_frame.winfo_children():
#             widget.destroy()
#
#         # Ensure valid instance creation
#         try:
#             if page_class == ReadMessagesPage and message_id is not None:
#                 frame = page_class(self.main_frame, self, message_id)  #
Pass message_id
#             else:
#                 frame = page_class(self.main_frame, self)  # Other pages
#
#             frame.pack(fill=tk.BOTH, expand=True, pady=20)  # Ensure
valid packing
#
#             # If the page has a list_messages method, call it
#             if hasattr(frame, "list_messages"):
#                 frame.list_messages()
#         except:
#             messagebox.showerror("Error", f"Message {message_id} not
found!")
#
#     # -----------------------------------------
#     def create_sidebar(self):
#         """Creates sidebar navigation with buttons."""
#         self.list_message_indicator = tk.Label(self.options_frame,
text='', bg='#7393B3')
#         self.list_message_indicator.place(x=14, y=15, width=5, height=38)
```

```
#
#         list_messages_btn = tk.Button(self.options_frame, text="List
Messages", font=('Bold', 12), fg="#1155cc",
#                                       bd=0, bg='#7393B3', width=15,
height=2,
#                                       command=lambda:
self.indicate(self.list_message_indicator, ListMessagesPage))
#         list_messages_btn.pack(pady=15)
#
#         self.read_message_indicator = tk.Label(self.options_frame,
text='', bg='#7393B3')
#         self.read_message_indicator.place(x=17, y=83, width=5, height=38)
#
#         read_frame = tk.Frame(self.options_frame, bg='#7393B3')
#         read_frame.pack(pady=15)
#
#         read_message_btn = tk.Button(read_frame, text="Read Message:",
font=('Bold', 12), fg="#1155cc", bd=0,
#                                      bg='#7393B3', width=10, height=2,
#                                      command=lambda:
self.indicate(self.read_message_indicator, ReadMessagesPage))
#         read_message_btn.pack(side=tk.LEFT)
#
#         self.id_txt = tk.Entry(read_frame, width=2)
#         self.id_txt.pack(side=tk.LEFT, padx=2)
#
#         self.new_message_indicator = tk.Label(self.options_frame,
text='', bg='#7393B3')
#         self.new_message_indicator.place(x=14, y=151, width=5, height=38)
#
#         new_message_btn = tk.Button(self.options_frame, text="New
Message", font=('Bold', 12), fg="#1155cc", bd=0,
#                                     bg='#7393B3', width=15, height=2,
#                                     command=lambda:
self.indicate(self.new_message_indicator, NewMessagePage))
#         new_message_btn.pack(pady=15)
#
#         self.label_message_indicator = tk.Label(self.options_frame,
text='', bg='#7393B3')
#         self.label_message_indicator.place(x=14, y=219, width=5,
height=38)
#
#         label_messages_btn = tk.Button(self.options_frame, text="Label
Messages", font=('Bold', 12), fg="#1155cc", bd=0,
#                                        bg='#7393B3', width=15, height=2,
#                                        command=lambda:
self.indicate(self.label_message_indicator, LabelMessagesPage))
#         label_messages_btn.pack(pady=15)
#
#
#
#         # Status Label (Shared by all pages)
#         self.status_lbl = tk.Label(self.options_frame, text="",
#                                    bg="#7393B3", fg="#1E2136")
#         self.status_lbl.pack(side="bottom", fill="x", pady=20)  # Ensure
it's at the bottom of sidebar
#
#     # ----------------------------------------
#     def hide_indicator(self):
#         """Resets all sidebar indicators."""
#         self.list_message_indicator.config(bg='#7393B3')
```

```
#         self.read_message_indicator.config(bg='#7393B3')
#         self.new_message_indicator.config(bg='#7393B3')
#         self.label_message_indicator.config(bg='#7393B3')
#
#     # ----------------------------------------
#     def indicate(self, indicator, page_class):
#         """Highlights active section and refreshes frame."""
#         self.hide_indicator()
#         indicator.config(bg='#1155cc')
#         self.show_frame(page_class)
#
# # -------- Run the App ---------
# if __name__ == "__main__":
#     app=EmailManagerApp()
#     fonts.configure()  # configure the fonts
#     app.mainloop()
```

B. Test table and results for each class in simulated Email Manager Application

Class	Sample Input	Sample Action	Expected Output	Actual Output as expected
TestEmailManagerApp		Click 'List Messages' button	Check if the active page is ListMessagesPage	YES
		Click 'Read Messages' button	Check if the active page is ReadMessagesPage	YES
	Enter invalid ID (e.g., 'abc')	A Message box warning should pop up saying requesting valid input	The ID should be cleared, and no page switch should occur	YES
		Click 'New Message' button	Check if the active page is NewMessagesPage	YES
		Click 'Label Messages' button	Sidebar indicators should update their background colour to #1155cc for each page switch	YES

Class	Sample Input	Sample Action	Expected Output	Actual Output as expected
TestListMessagesPage	filter_entry = "Alice"	Call toggle_search_button() after inserting input	search_button enabled (state = "normal")	YES
	filter_entry = ""	Call toggle_search_button() after clearing input	search_button disabled (state = "disabled")	YES
	filter_entry = "nonexistent@example.com"	Call filter_messages() with no matching data	Warning message: “No messages found for ‘nonexistent@example.com’ under ‘Sender’”	YES
		Simulate refresh button click	show_frame called with ListMessagesPage	YES
		Simulate messages data in messages by calling list_messages()	list_txt populated with messages content (e.g., "", "*")	YES

Class	Sample Input	Sample Action	Expected Output	Actual Output as expected
TestReadMessagesPage	message_id = 1	Call ReadMessagesPage(self.root, self.controller, self.message_id)	Populates entry fields with message data (sender, recipient, subject, content)	YES
	message_id = 1	Call ReadMessagesPage(self.root, self.controller, self.message_id)	Redirect to ListMessagesPage when message not found	YES
	priority = 3	Call update_priority() with a valid priority	set_priority is called with message_id = 1 and priority = 3	YES

	priority = 10	Call update_priority() with an invalid priority	set_priority is not called, as priority is invalid	YES
	message_id = 1	Simulate clicking delete_message()	delete_message is called with message_id = 1, redirect to ListMessagesPage	YES

Class	**Sample Input**	**Sample Action**	**Expected Output**	**Actual Output as expected**
TestNewMessageApp	Missing sender, valid recipient, subject, content, and priority	Call send() method with missing sender field	Error message: “All fields must be filled!” shown	YES
	Invalid sender email, valid recipient, subject, content, and priority	Call send() method with invalid sender email	Error message: “Invalid email format!” shown	YES
	Valid sender, recipient, subject, content, and invalid priority	Call send() method with invalid priority	Error message: “Priority must be between 0 and 5!” shown	YES
	Valid sender, recipient, subject, content, and priority	Call send() method with valid input	new_message called with correct parameters, success message shown	No
		Click "Cancel" button	Controller switches to ListMessagesPage	YES

Class	Sample Input	Sample Action	Expected Output	Actual Output as expected
TestLabelMessagesPage	Label:"Work"	Simulate selecting "Word" label and list messages	Messages with "Work" label listed in list_txt, no error messagebox shown	YES
	Label:"Work"	Simulate selecting "work" label with no messages	Error messagebox:"No messages found!"	YES
	Label:"Important", Message ID: "1"	Simulate adding a label to an existing message	Success messagebox display	YES
	Label:"Important", Message ID: "999"	Simulate adding a label to non-existant message	set_label not called, error messagebox: "Message ID not found"	YES
		Simulate not selecting a label before clicking "Add Label"	Error messagebox: "Please, select a label"	YES
	Message ID: "invalid_id"	Simulate entering invalid message ID	Error messagebox: "Please, enter a valid ID"	YES

Class	Sample Input	Sample Action	Expected Output	Actual Output as expected
TestMessage	Message data: sender, recipient, subject, content, label, priority	Initialize Message instances and check each attribute	Check if the Message object is correctly initialized with expected values	YES

	Message with priority = 3	Call info() method on the message with priority 3	The returned info string includes stars for priority and formatted data (sender, label, subject, etc.)	YES
	Message with priority = 0	Call info() method on the message with no priority (0)	The returned info string does not include stars, just formatted data	YES
	Message with priority = 3 and priority = 0	Call stars() method on messages with priority 3 and 0	Stars for priority 3 are returned as ("***"), no stars for priority 0	YES
	Message with priority = 0	Call stars() method on message with priority 0	Ensure that no stars are returned for zero priority	YES

Class	**Sample Input**	**Sample Action**	**Expected Output**	**Actual Output as expected**
TestMessageManager		Test that CSV file is initialised if it doesn't exist	Ensure the CSV writer writes the header once	YES
	Mock data for 2 messages	Test loading messages from CSV and check the message attributes	Validate that the correct number of messages are loaded and their attributes are as expected	YES
	New message details: sender, recipient, subject, etc	Add a new message and check if the message count increases, then validate the new message's details	Ensure the message count increased and the new message details are correct	YES

	List of messages with specific data	Save all messages to CSV and check if the save_all_messages function is called	Check that the save_all_messages function was called with the correct argument	YES
	Message dictionary containing one message	Delete a message and ensure it is removed from the dictionary	Ensure the message is removed from the dictionary	YES
	Message with ID 1	Retrieve a message by ID and check the details	Check if the correct message is retrieved and its attributes are correct	YES
	Mock data for 2 messages	List all messages and check the length of the returned messages	Ensure that the length of the message dictionary is 2	YES
	Change label of message 1	Change the label of a message and check the updated label	Ensure that the label of the message was updated correctly	YES
	Change priority of message 1	Change the priority of a message and check the updated priority	Ensure that the priority of the message was updated correctly	YES

Class	**Sample Input**	**Sample Action**	**Expected Output**	**Actual Output as expected**
TestFontConfiguration		Call to configure() function	Verify if the default font is set to 'Segoe UI' and size 11	YES