

<b>Software Engineering: Project</b>
--

<b>Names And SRNs:</b>
Shreya Varma: PES1UG20CS403
Shria Guntunur: PES1UG20CS411
Shruthi Pai: PES1UG20CS416
Shubangi Saxena: PES1UG20CS418
<b>Course Code:</b> UE20CS303
<b>Course Title:</b> Software Engineering
<b>Project:</b> Automatic Email Attachment Extractor
<b>Semester:</b> V

### **TABLE OF CONTENTS:**

<b><u>SL.No</u></b>	<b><u>TOPIC</u></b>	<b><u>PAGE NUMBER</u></b>
<b>1.</b>	<b>PROPOSAL OF THE PROJECT</b>	<b>1</b>
<b>2.</b>	<b>SOFTWARE REQUIREMENTS SPECIFICATION</b>	<b>1</b>
<b>3.</b>	<b>PROJECT PLAN</b>	<b>7</b>
<b>4.</b>	<b>DESIGN DIAGRAMS</b>	<b>8</b>
<b>5.</b>	<b>TEST CASES</b>	<b>9</b>
<b>6.</b>	<b>SCREENSHOTS OF OUTPUT</b>	<b>13</b>
<b>7.</b>	<b>CONCLUSION</b>	<b>17</b>

## **Abstract of the project**

Our project is to automate the process of downloading the attachment from an email into the required directory. We have taken this project topic from the list of titles that had been shared with us. This software will mainly be used by students and individuals working in the industry as people find it hard to keep track of all the downloaded and partially downloaded files because of the sheer volume of emails received. Along with this, we will provide the functionality of downloading the email attachment in the appropriate directory.

The user will be able to perform the following functions-

- Download the email attachment having a specific subject
- Attachments can be saved in different folders based on the requirement of the user

We have used imap protocol in python for accessing the emails.

For the UI (front end), we have used streamlit to accept username and password of the user's email account.

## **Software Requirements Specification**

### **Purpose:**

Everyone has once been in a position where they search over their mailbox to download all the attachments needed. The Automated Email Attachment extractor will download attachments from emails sent to a particular directory. This is a generic product that can be used that can be used by everyone.

### **Intended Audience:**

This software can mainly be used by students and individuals working in the industry as people find it hard to keep track of all the downloaded and partially downloaded files because of the sheer volume of emails received.

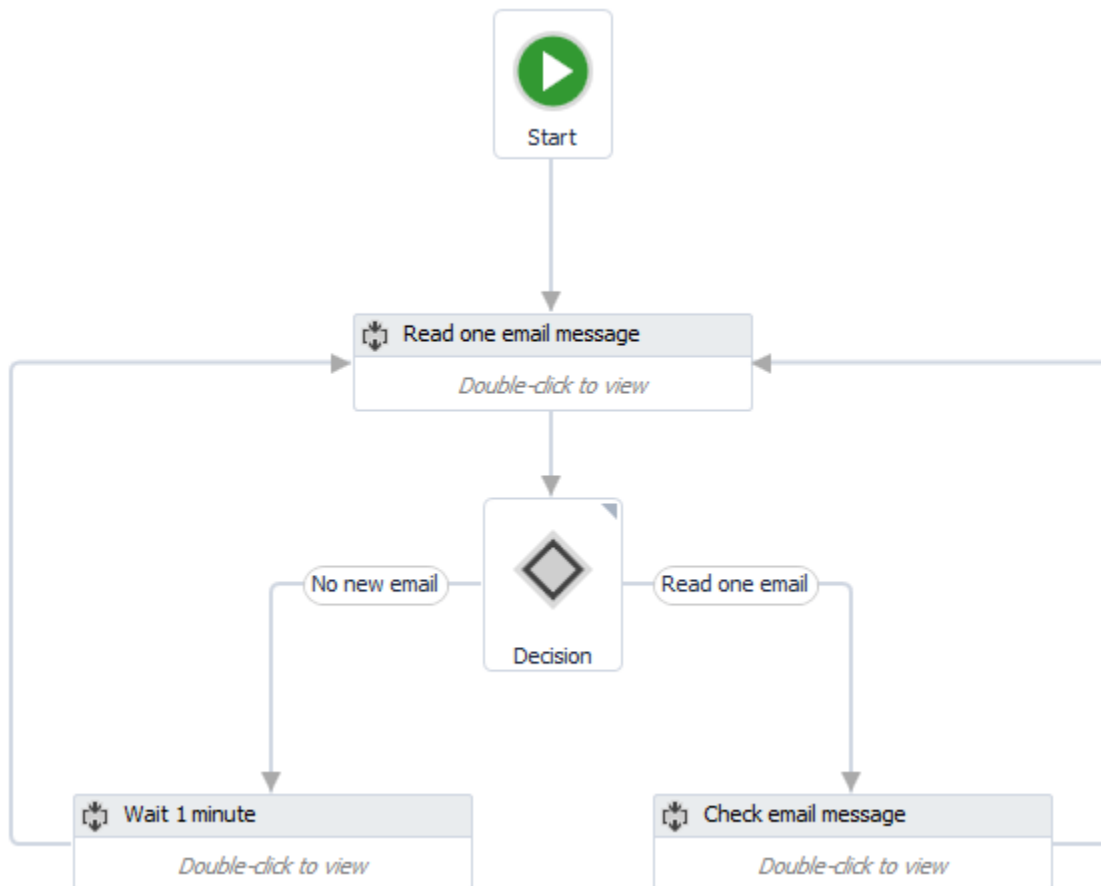
### **Product Scope:**

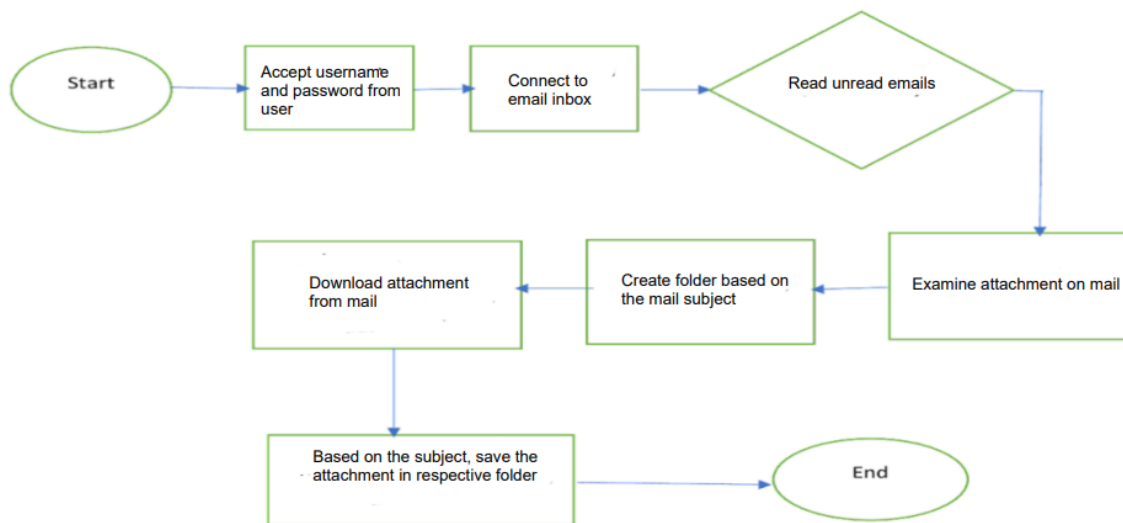
The functionality of downloading the email attachment in the appropriate directory will be provided. As this process is being automated, the manual work involved will be reduced drastically. As a result, this will save a lot of time especially when it's being used by big corporations considering the sheer volume of emails they receive every minute.

## Overall description:

### Product Perspective:

The product perspective is aimed to be user-friendly, straightforward that facilitates downloads of email attachments as an automated process. From a user perspective, with just one click, they can download the attachments desired.





The first diagram explains the decision process once an email is read. In this product, once the decision is taken, the attachment will be downloaded.

The second diagram explains the entire process of the product from reading emails to downloading attachments in their respective directories based on the extensions.

### Product functions:

The user will have the following functionalities:

- Download the email attachment having a specific subject
- Attachments can be saved in different folders based on the the subject

### User Classes and Characteristics:

- Student: Students are constantly getting mails from their universities regarding course details and assignments. Each student has their own university email ID which they can use.
- Corporations/Companies: It's a known fact that companies get tons of emails in a minute and this product would help them sort through the emails and their respective attachments.

### Operating Environment:

The client using the product can use the software on any operating system be it Linux or Windows as long as it has a valid Python IDE with the latest version of python installed.

### Design and Implementation Constraints:

The product will work as long as it is being used for the specified email service provider, for instance, Gmail.

The setup and maintenance of the product will be the responsibility of the customer's organization.

### Assumptions and Dependencies:

The main assumption of our product is that the attachment extractor is being used on a valid email id.

### External Interface Requirements:

#### User Interfaces:

We are going with a simple User Interface using streamlit, where the user can provide their username and password. The automatic email attachment extractor does not require a full-fledged UI design but we have decided to go with this to enhance the user experience.

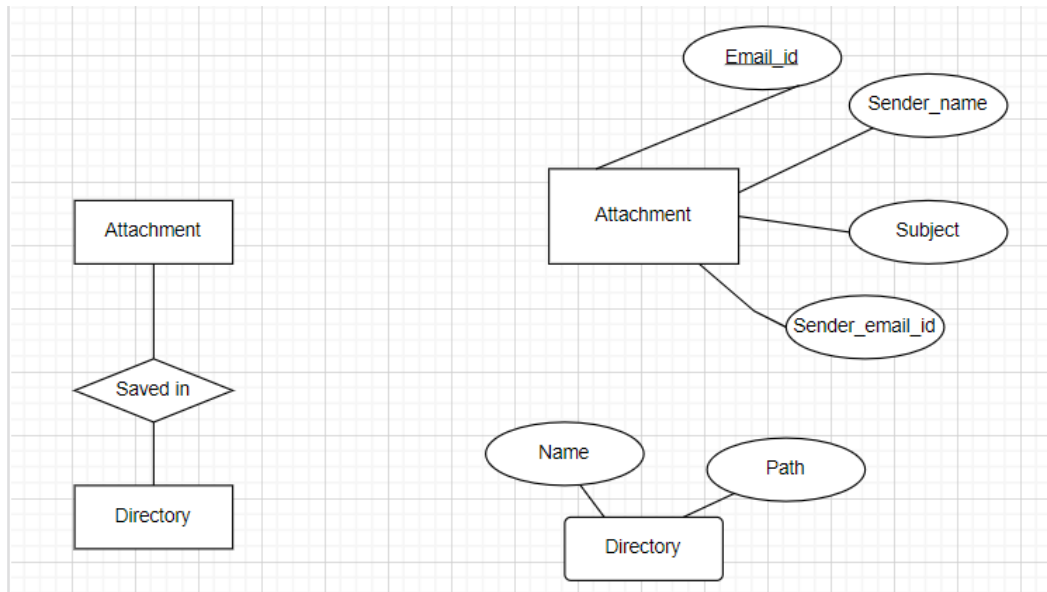
#### Software Interfaces:

Internally, our product makes use of python and streamlit.

#### Communication Interfaces:

Our product makes use of IMAP protocol to access emails. No specific web browser requirements.

## Analysis Models:



## System Features:

### System Feature 1: Downloading Interface

- Download email attachment having a specific subject

### System Feature 2: Saving Interface

- Save the attachment in the specified folder

## Other non-functional requirements:

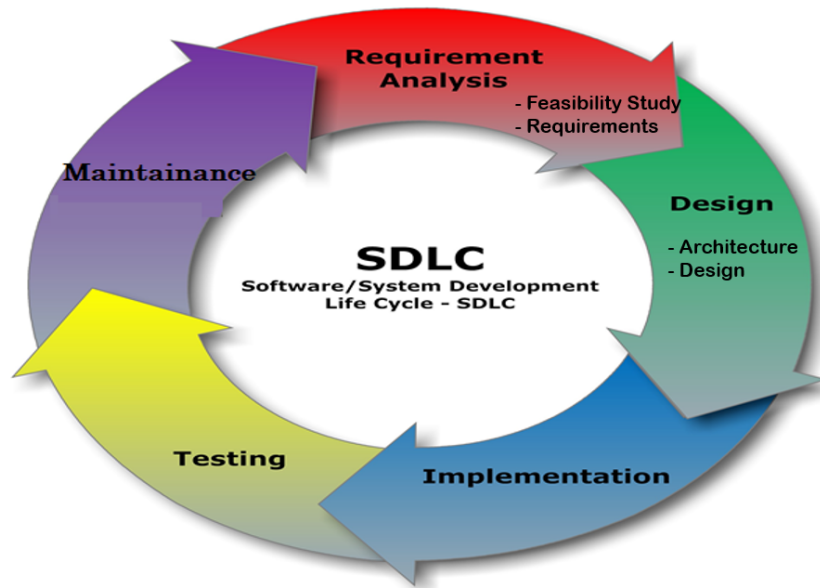
### Performance Requirements:

We are trying to make an efficient automatic email attachment extractor which is easy to use for the user. We also look to keep storage capacity flexible, aiming at lesser storage, to retain an efficient space-time ratio. We look to achieve a quick speed of response and execution time by our product, with maximum throughput.

### Security Requirements:

The final product will ensure security especially regarding user details.

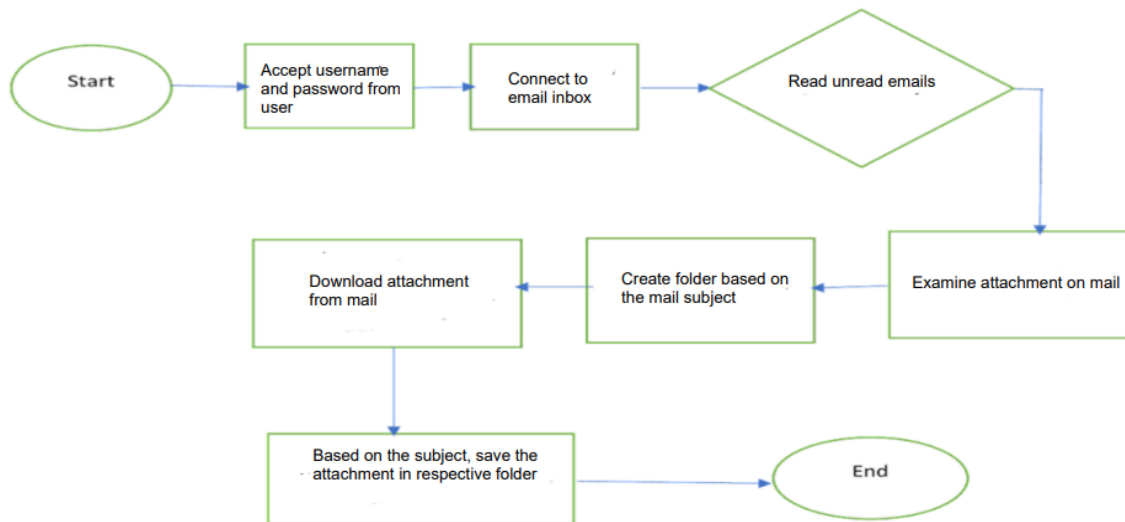
# Project Plan



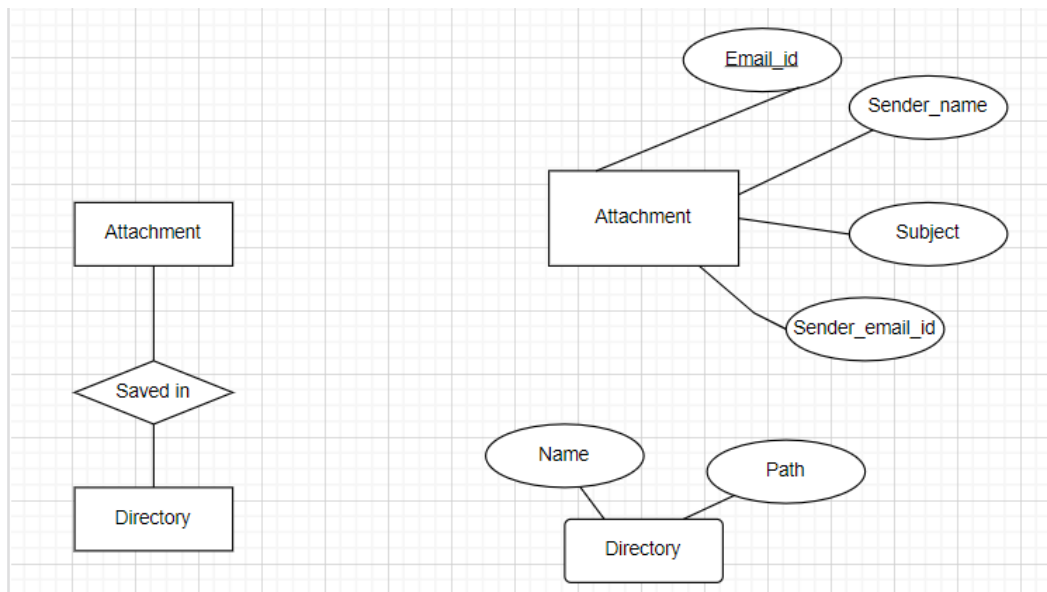
The above image displays the various stages for Software Development Life Cycle. We have followed this to plan and complete our project in a timely manner.

1. **Requirement Analysis-** For the first step, we did a feasibility study. We observed that our project is a very practical one. As of 2022, Google removed the option to select “Allow less secure apps” which meant that we weren’t able to connect our Python code with gmail using IMAP, etc. As a result, we switched to App passwords instead, where the user will have to generate one and input it so that we could connect to gmail on their behalf.
2. **Design-** For the next step, we created various high-level architectures and analysis diagrams for us to understand how we should proceed with this project. The design diagrams have been shown as well.
3. **Implementation-** For the third step, we divided the implementation among backend and frontend. In the frontend, using streamlit, we accept the username and password to their gmail account(with an option to delete it later). In the backend, we store these details in a database and also run our main python code to download attachments to their respective folders.
4. **Testing-** For the fourth step, we tested our code against various usernames and passwords which all ran successfully. We also had some fail cases in the initial stages but eventually we solved all the errors.
5. **Maintenance-** In this last step, we have a fully working product. We made some minute changes and made sure everything works without a glitch.

# Design Diagrams



The above image explains the flowchart of our model and project. Once we connect to the gmail inbox, the emails with attachments will be read. The attachments are then downloaded to the respective folders based on subject name.



The above image shows important components/entities that will be used in our project.



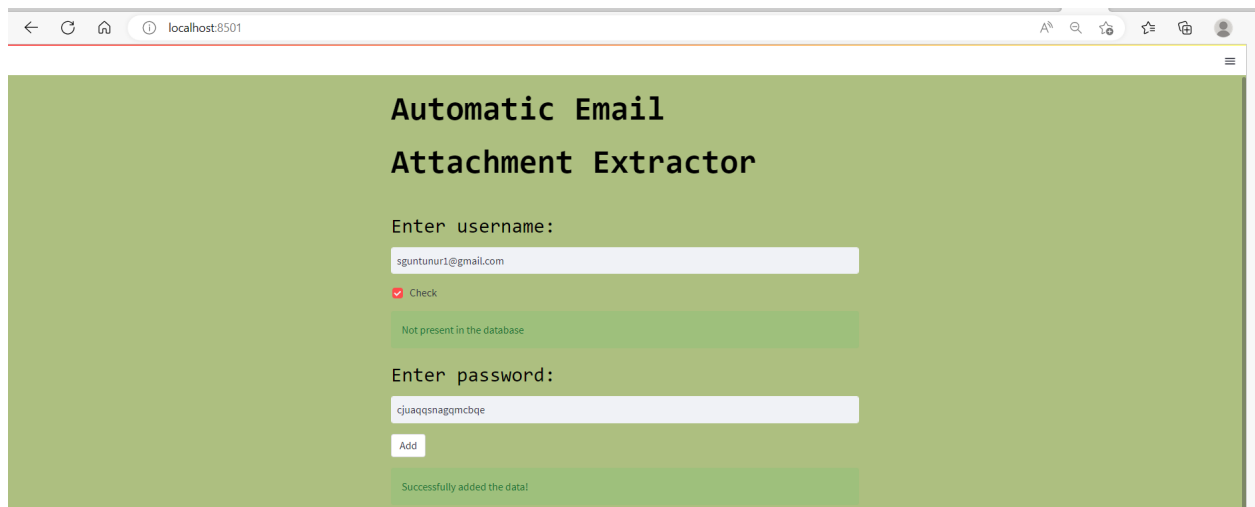
# Test cases

## Unit testing

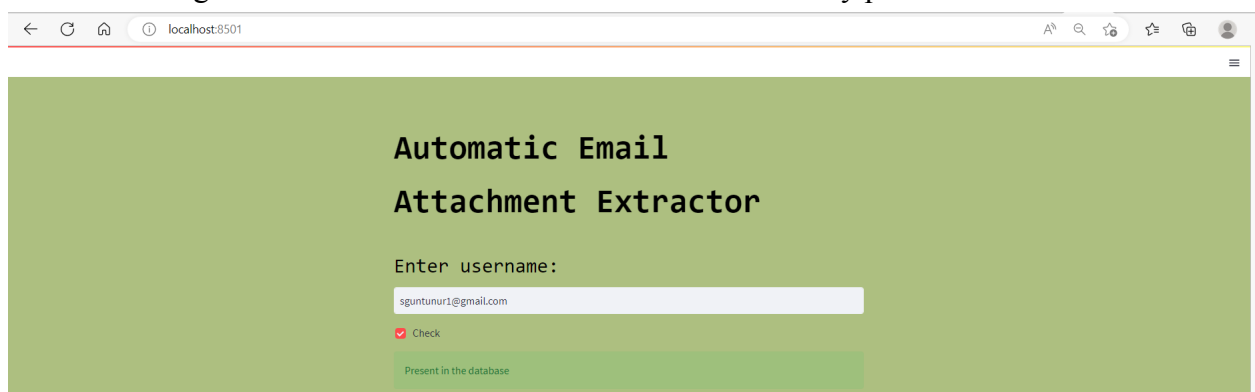
### Unit 1:

```
database.py > Jupyter > runp
24
25 def check(username):
26     c.execute('select username,password from details where username="{0}"'.format(username))
27     data=c.fetchall()
28     return data
29
```

The check function verifies if the username and password are already present in the database.



The above image shows the case when the username is not already present in the database.



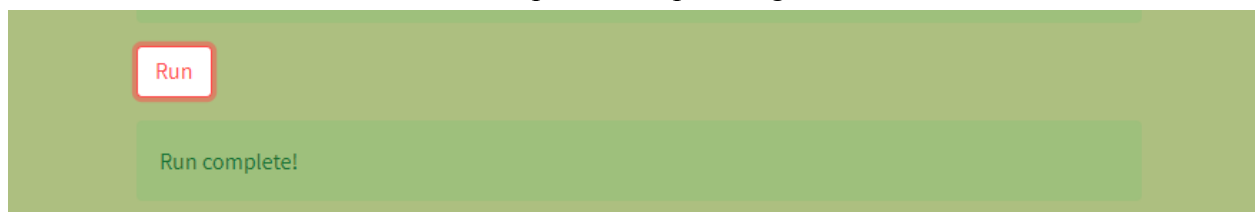
The above image shows the case where the username is already present in the database.

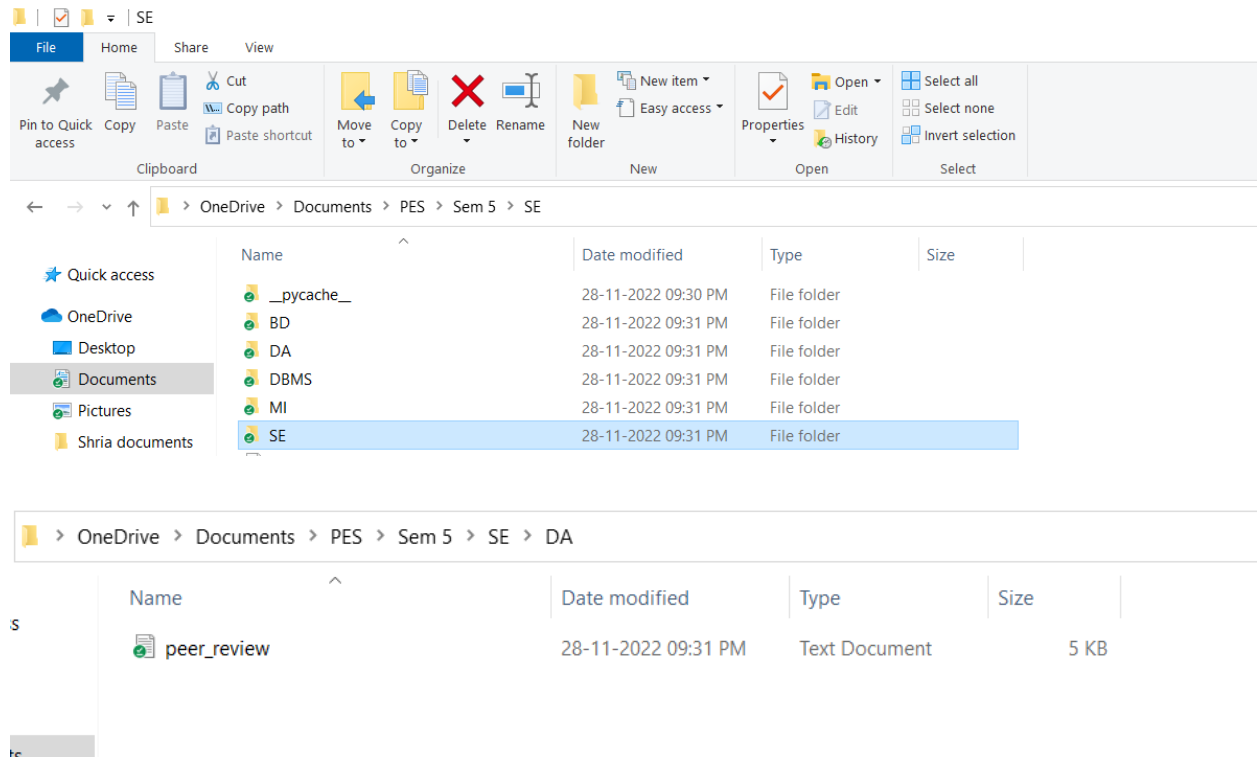
### Unit 2:

```

database.py > Jupyter > runp
30 def runp(username,password):
31     userName=username
32     passwd=password[0][1]
33     print(userName)
34     print(passwd)
35     try:
36         imapSession = imaplib.IMAP4_SSL('imap.gmail.com')
37         typ, accountDetails = imapSession.login(userName, passwd)
38         print('done')
39
40         print(typ,accountDetails)
41         imapSession.select('INBOX')
42         print('break')
43         sub_list=['DBMS','SE','MI','DA','BD']
44         for i in range(5):
45             typ, data = imapSession.search(None,'HEADER Subject "{}"'.format(sub_list[i]))
46             detach_dir = '.'
47             if sub_list[i] not in os.listdir(detach_dir):
48                 os.mkdir(sub_list[i])
49             print('done1')
50             # Iterating over all emails
51             for msgId in data[0].split():
52                 typ, messageParts = imapSession.fetch(msgId, '(RFC822)')
53                 emailBody = messageParts[0][1]
54                 raw_email_string = emailBody.decode('utf-8')
55                 mail = email.message_from_string(raw_email_string)
56                 print('emailbody complete ...')
57                 for part in mail.walk():
58                     print(part)
59
60                     fileName = part.get_filename()
61                     print(fileName)
62                     if bool(fileName):
63                         filePath = os.path.join(detach_dir,sub_list[i], fileName)
64                         if not os.path.isfile(filePath) :
65                             print(fileName)
66                             fp = open(filePath, 'wb')
67                             fp.write(part.get_payload(decode=True))
68                             fp.close()
69
70                     imapSession.close()
71                     imapSession.logout()
72                     st.success('Run complete!')
73
74     except :
75         print('Not able to download all attachments.')
```

A successful run is shown below with respective output images-





If the run button is clicked without entering username and password, it throws an error and prints an appropriate message as shown below-

### Unit 3:

```

database.py > Jupyter > runp
75 def delete_data(username):
76     c.execute('DELETE FROM details WHERE username="{}".format(username))
77     mydb.commit()

```

When the delete\_data function is called, the function tries to delete the entered username and password from the database. If the username and password is not inputted, it throws an error, as shown below.

Enter username:

☐ Check

Run

Delete information

Cannot delete without entering username and password!

Enter username:

☒ Check

Present in the database

Run

Delete information

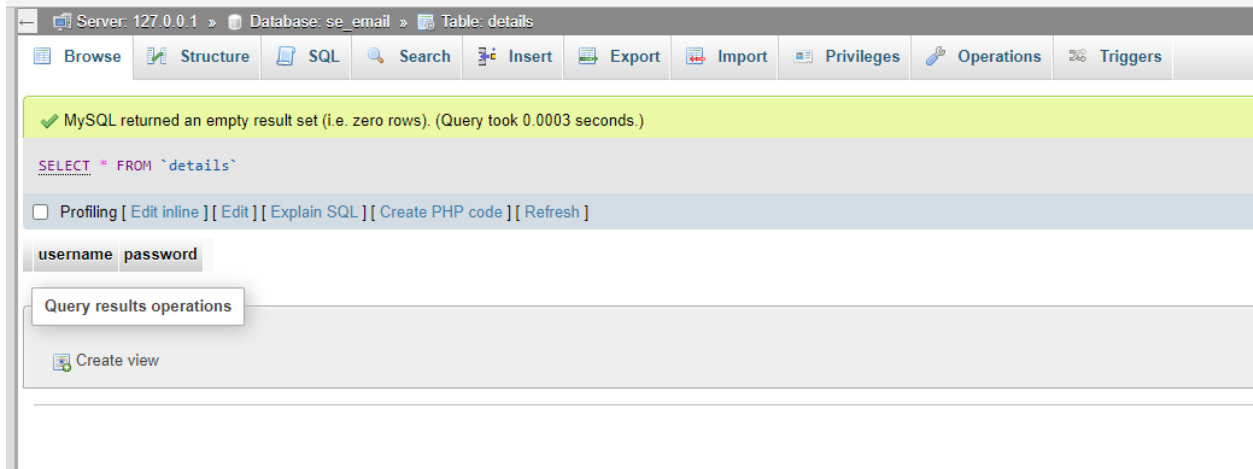
Successful delete

The above image shows a successful delete.

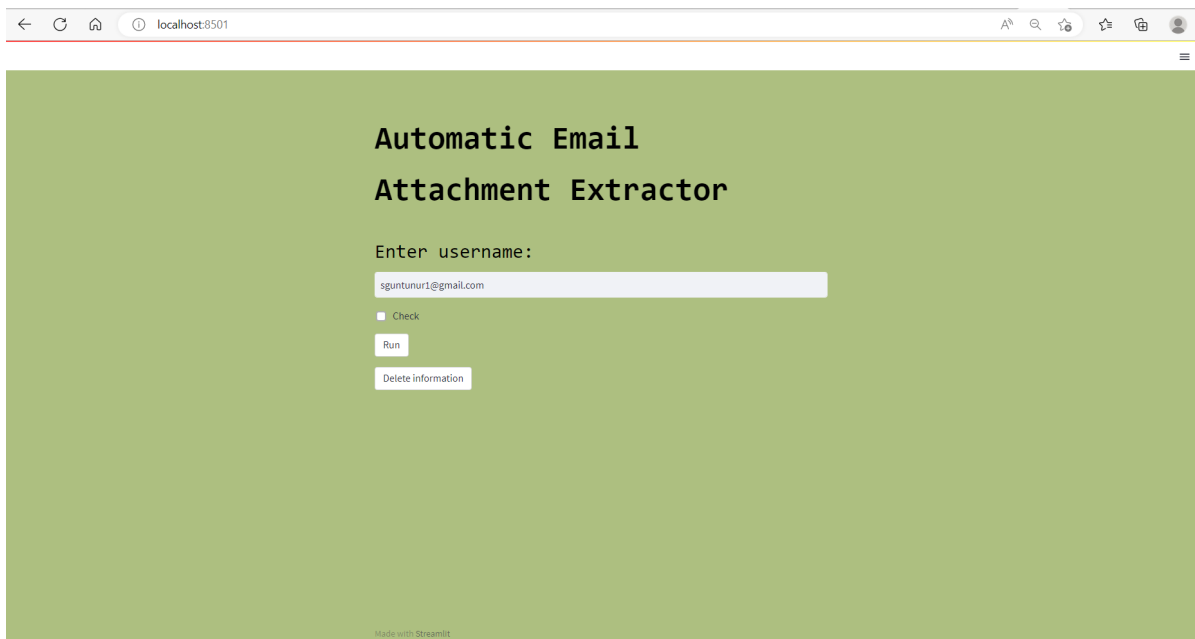
When writing larger blocks of code, they can be split into various files. Once that has been completed, we can import these files to the main input file.

# Output screenshots

## 1.Empty details table



## 2. UI



### 3. Add data

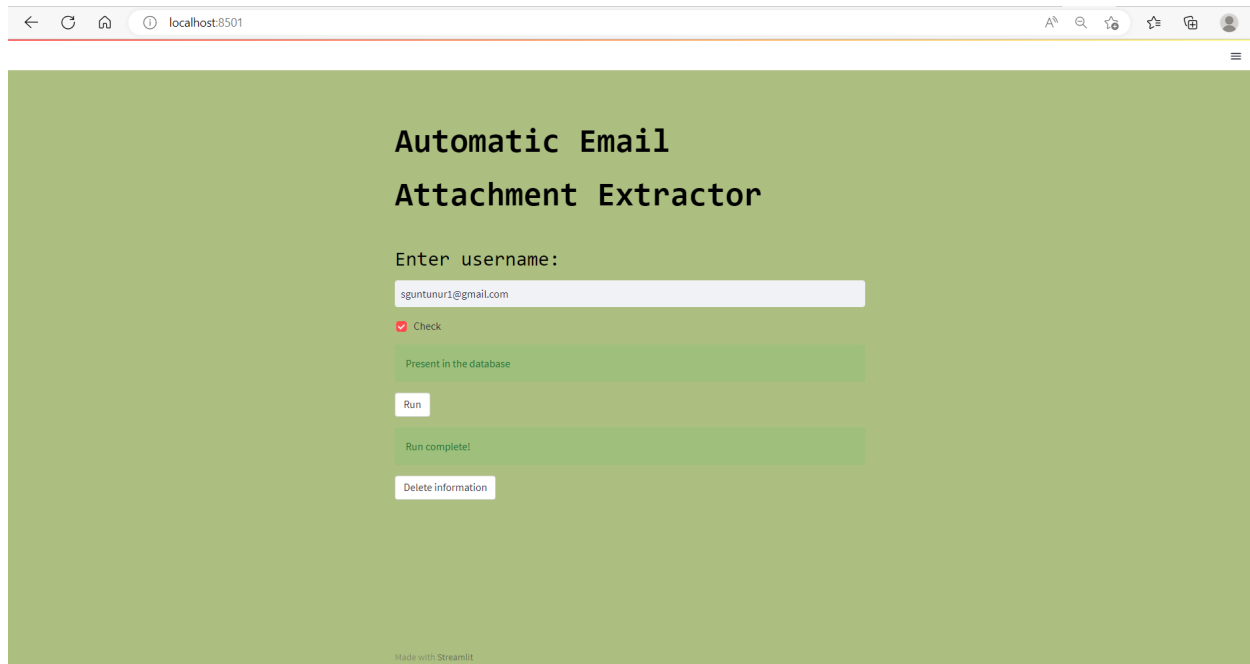
The screenshot shows a web browser at localhost:8501 displaying the 'Automatic Email Attachment Extractor' application. The interface has a green background. It contains two main input sections. The first section, 'Enter username:', has a text input field containing 'sguntunur1@gmail.com', a red checkmark icon, and a green message box stating 'Not present in the database'. The second section, 'Enter password:', has a text input field containing 'cjuqqsnagqmcqbq', an 'Add' button, a green message box stating 'Successfully added the data!', a 'Run' button, and a 'Delete information' button.

### Updated database

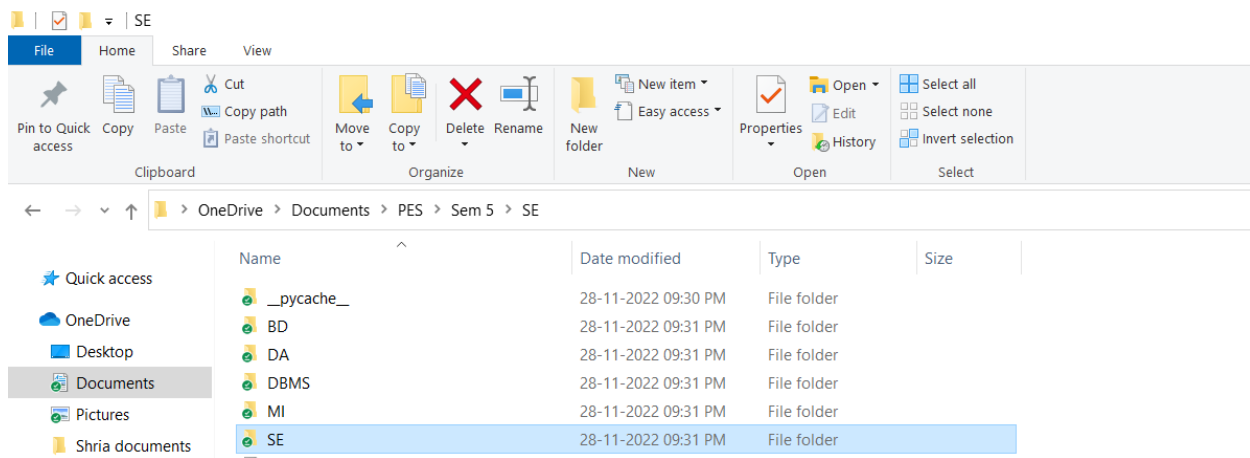
The screenshot shows the phpMyAdmin interface for a database named 'se\_email'. The 'Table: details' tab is selected. The table structure is shown with two columns: 'username' and 'password'. The table contains one row of data: 'sguntunur1@gmail.com' and 'cjuqqsnagqmcqbq'. The interface includes various navigation and query tools at the top, a status bar indicating 'Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)', and a 'Query results operations' section at the bottom with options like 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

username	password
sguntunur1@gmail.com	cjuqqsnagqmcqbq


#### 4. Run complete



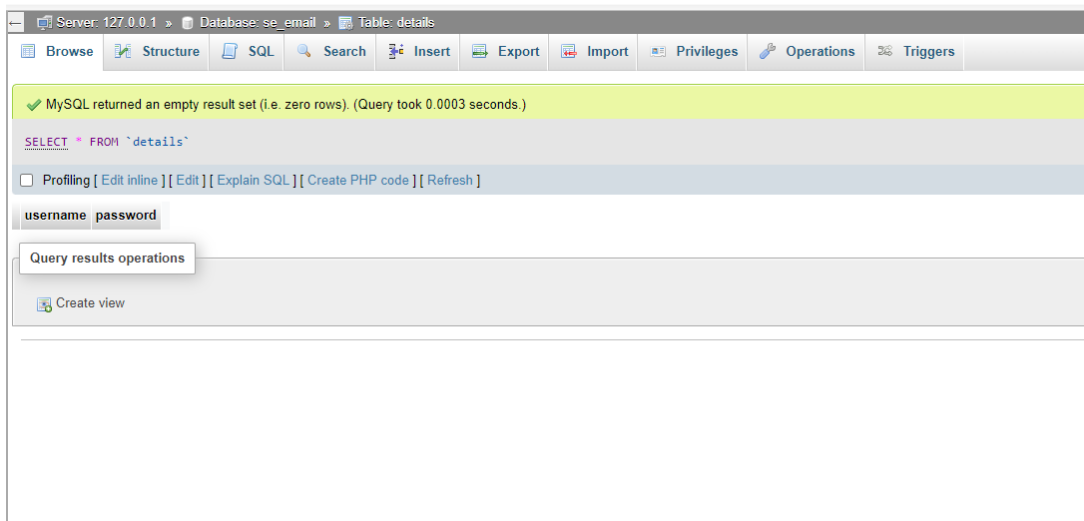
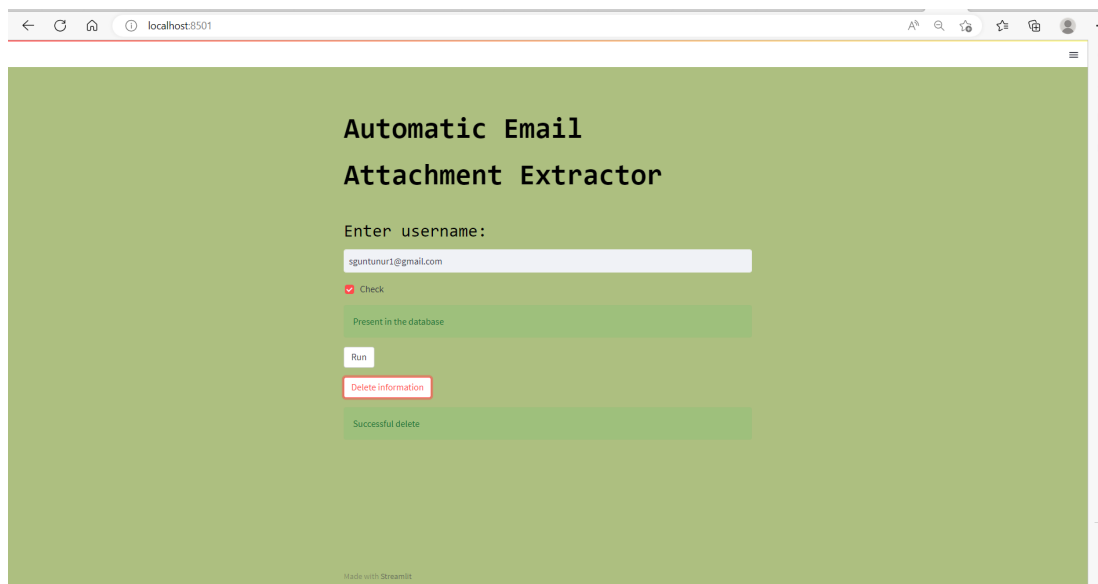
#### Folders created for each subject-



Attachment present in DA folder-

OneDrive > Documents > PES > Sem 5 > SE > DA				
	Name	Date modified	Type	Size
s	 peer_review	28-11-2022 09:31 PM	Text Document	5 KB

## 5. Data deleted from the database





## **CONCLUSION**

This project- an automatic email attachment extractor, works in a way that is beneficial to students, teachers as well as paper setters. In our project, the attachment extraction occurs on the basis of the subject. The UI is simple and easy to understand, hence can be expanded into multiple sectors.