Smail - Smart Email Client

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Sahil J. Chaudhari and Vishesh Munjal
(111801054 and 111801055)

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CERTIFICATE

*This is to certify that the work contained in the project entitled "**Smail - Smart Email Client**" is a bonafide work of **Sahil J. Chaudhari and Vishesh Munjal** (**Roll No. 111801054 and 111801055**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my guidance and that it has not been submitted elsewhere for a degree.*

Dr. Albert Sunny

Assistant/Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

The BTP project assigned to us is a great opportunity for learning and personal growth. The project started as a vague idea to counter some problems that were generated due to the communication gap in the recent pandemic. Therefore, We are grateful for this chance and are grateful to our professors who have motivated, contributed and modified this vague idea and gave it the shape of a budding project which is yet to bloom.

First of all, we would like to thank our college, IIT Palakkad, for making it possible for us to have such an opportunity exist in the first place. We really appreciate the hard work our professors, along with the academic section put in to tutor us and make us the best version of ourselves.

Second of all, I would really like to thank Dr. Albert Sunny for being the project mentor. Due to Dr. Albert only, we were able to extend the project - smail client with additional features which are unique to the project and may come in handy to anyone using the client. His insight and helpful nature are in some sense the backbone of our project.

We would like to thank Prof. Unnikrishnan C, the BTP project coordinator for his helpful nature and understanding. We really appreciate the effort he puts in to make us understand the relevance of the project as well as the norms to be followed. Also, we are thankful for the regular updates and his patience with the deadlines.

Due to all of the above-mentioned people, directly or indirectly, we hope this experience becomes an unforgettable golden opportunity that will help us be successful in our career.

Abstract

The Smart Email Client is an android application with a custom build AI to categorise and prioritize emails via easily customizable coloured classification according to users organization needs. The Client will be packed with a custom UI and exciting features such as auto google calendar integration , google assistant integration, reminder system with custom notification according to priority, text-to-speech and speech-to-text integration and tagging emails through mentions to friends.

The client even extends to more features but essentially to focus on the main feature is the custom priority that a user/organisation experiences with the focus to prioritize important mails and display them accordingly to their priority class and colour code. A follow up feature of this is to have calendar integration to handle notifications for deadlines or events that will be automatically handled by the client. The extended features aim to make the client as user friendly as possible with an advanced option to use google assistant to convert voice to text mails.

Contents

List of Figures **v**

1 Introduction **1**

1.1 Motivation 2

1.2 Extension of project 2

1.3 Overview of report 2

2 Progress **3**

2.1 Setup 3

2.1.1 Need For Credentials 3

2.1.2 Process For Establishing Credentials for API 4

2.2 Designing Architecture 4

2.3 Building Front-End 6

2.4 Priority Algorithm 6

3 Algorithm I **7**

3.1 Idea behind algorithm 7

3.2 Construction 8

3.3 Algorithm analysis and Conclusion 12

4 Algorithm II **13**

4.1 Improved Method 13

4.2 Construction of improvised algorithm 15

4.3 Conclusion 17

5 Future Work **19**

5.1 Improvisation of priority handler 19

5.2 Addition of features 20

6 Results **21**

6.1 Smail Front End 21

6.2 Priority Algorithm Results 24

6.3 Google API Credentials 25

7 Conclusion **27**

7.1 Contribution 28

References **29**

List of Figures

2.1 Overview of Smail client application 5
2.2 Smail client application model flow diagram 5

3.1 Tag keywords example with priority 8

4.1 Trie example forAPPLE, ARE, ORDER and ORE. 14

6.1 User's Main Page 21
6.2 User's Tool Bar 21
6.3 Meet Main Page 22
6.4 Meet Schedule Page 22
6.5 Mail Page 23
6.6 User Configuration Option's 23
6.7 Reply Page 23
6.8 User Accounts toggle 23
6.9 Sample Emails 24
6.10 Defined Tags and Priority 25
6.11 Calculated Priority for Respective Mails 25
6.12 Google API credential for application 26

Chapter 1

Introduction

The Smail - Smart Email Client is an android application with a custom build AI to categorise and prioritize emails via easily customizable coloured classification according to users organization needs. This will allow the user to give importance to the mails they are interested in and will make sure that such mails are never misplaced or forgotten like a needle in a haystack. The basic implementation behind this feature is to use labels or in layman's terms certain keywords and a collection of such keywords will generate the priority index for the respective mail which can then be further classified into various categories and color coded. This implementation can then be refined further and further using the mail context to create a smart client as the name suggests. The project being a year long project is divided into goals that are to achieve the first and foremost goal is to create the app environment and develop the basic functionalities and the priority feature as mentioned above. Flutter which is an open source UI development kit by google will be used for the same and the target is to complete the basic setup and functionalities by this semester(SEM 7 august - december 2021).

1.1 Motivation

The motivation behind the project came from a vague idea to counter some problems such as missing deadlines conveyed through mails or unread mails due to bombardment of mails from all sub divisions of college namely clubs, academics, TPO etc. that were generated due to the communication gap in the recent pandemic. All the above mentioned problems lead to a need for a custom priority system for emails which further with some insight from our mentor was extended to other features as well which makes the client useful and handy. Thus the project Smail emerged.

1.2 Extension of project

The second and later goals is to extend the features of the client to support automated google calendar integration to set up notifications for deadlines and events with the basic idea behind is to never miss a deadline. Also provide the feature so that friends can tag another friend in regards to a particular mail through the means of mentions. Another feature is the google assistance integration with speech to text integration and vice-versa for user convenience. These will be the end goals and will be refined as much as possible in the upcoming semester(SEM 8 January - May).

1.3 Overview of report

In this chapter we have introduce our Smail-smart email client with motivation behind it, possible extension of it. In next chapter i.e Progress, we will explain the current progress of our project in this semester. In later chapters, we discuss about the algorithms for priority classification and colour coding as well as display some progress results(i.e. screenshots of app interface) and we will close our report with conclusion and contribution of each member toward project. And Lastly include all the references that we have used for this project so far.

Chapter 2

Progress

In this chapter, we will discuss and analyse the progress made so far in order to tackle the problem statement related to our email client.

2.1 Setup

After thoroughly researching and analysing various technologies and platforms, we established the setup:- We installed Android Studio on Windows and used flutter to program our app's interface. By using the dart programming language(i.e. the one supported by flutter), we were able to design the app's front end interface and display it on the Android device emulator(in our case, it is the inbuilt emulator of Android Studio). Moreover, since our application is a smart email client, it is important to integrate it with the Gmail API. Hence we established the credentials for using the Google API.

2.1.1 Need For Credentials

Since the Google API, for example Gmail API, allows us to access a user data, it becomes important to establish credentials as per google norms that act as a licence making the application a trusted one.

2.1.2 Process For Establishing Credentials for API

The process includes the following various steps[1][2]:-

- The first and the obvious step is to select the desired API as per the requirement.
- The Second Step is to enable the respective API. Note: In order to use a Google API, a google account and project is required.
- The Third Step deals with the authentication of the user and providing the desired scopes for the respective API.
- Then we can get a HTTP client consisting of the necessary credentials for the API invoking.
- Lastly we need to create and use the API class.

2.2 Designing Architecture

Since the Setup has been configured we will move on to the architecture design of our email client. This section also highlights the design model flow of the smail client application.

In our application there will be three main parts[refer figure 2.1]:- user interface handler which focus on the graphical presentation of the client , client controller which process user requests and API handler since our application will use Google APIs to integrate some of the functionalities namely fetching of emails etc.

The client controller is broken further into various parts[refer figure 2.2]:- Login Handler to handle user login, Account Handler to manage various accounts, Priority Handler to compute priority of emails, User configuration Handler to adjust the priorities according to user, Email Handler to fetch the emails using API, Labeler will take care of labeling the emails accordingly.

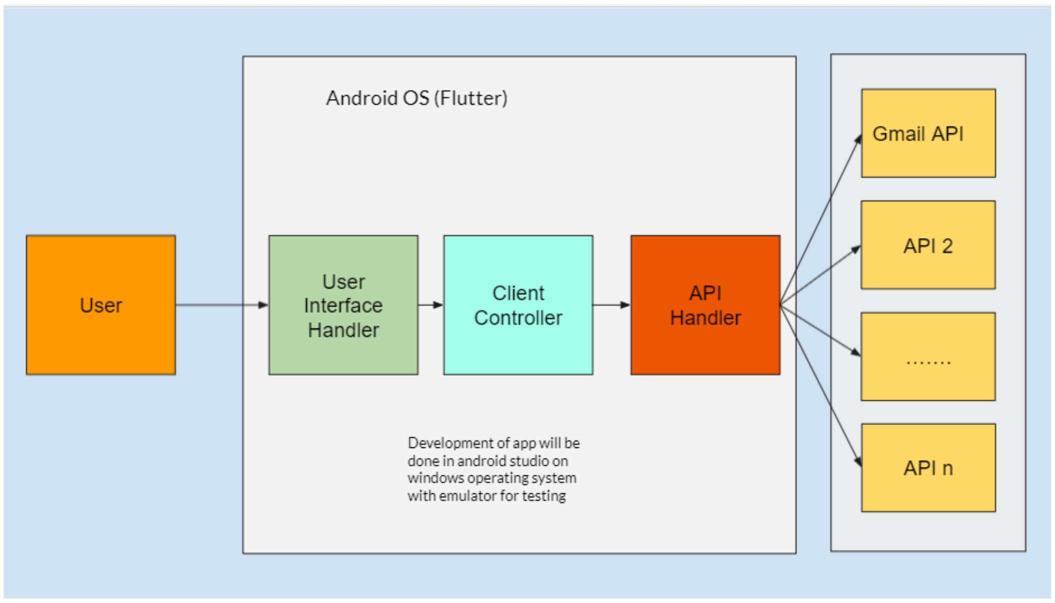

Android OS (Flutter)
User
User Interface Handler
Client Controller
API Handler
Gmail API
API 2
....
API n
Development of app will be done in android studio on windows operating system with emulator for testing

Fig. 2.1 Overview of Smail client application

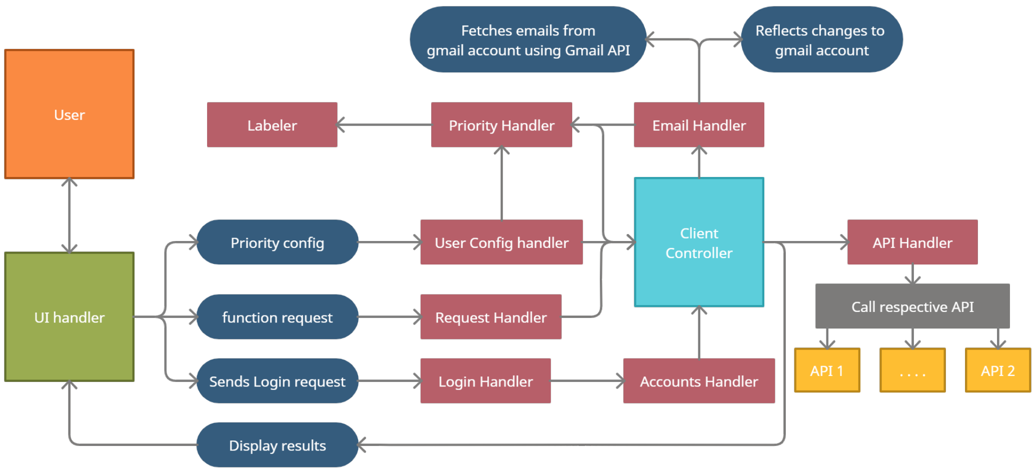


Fig. 2.2 Smail client application model flow diagram

2.3 Building Front-End

The front end has been designed[3], coded and debugged in the Android studio using dart programming language on the flutter platform. We have successfully completed the front end interface for our application. The interface showcase the exciting features of the application, namely, the main outlook, the drawer features and labels, integrated google meet page and the schedule meeting page, reply as well as user account page. All the pages have been coded and tested on the Android device emulator. The resulting screenshot of all the above mentioned pages as well as a details explanation is given in the Chapter 6 [Results].

In addition to the above mentioned pages we have also included some pages that are related to future work i.e. later phases of our project, namely, the configuration or various options/features that the full fledged application promises its users. Once the back end of the following is completed, it shall be integrated with these pages. **Note:** These also have been shown and explained in brief in Chapter 6 [Results].

2.4 Priority Algorithm

We have also completed the algorithm for the priority calculation of the emails, this is calculated by identifying certain keywords and then classifying the respective mails into various classes, namely, tags. which then are assigned a particular colour as per the label. The algorithm went under various modification during this phase to achieve better efficiency and now consists of 3 parts:- TextParser, GetPriority and SetTag. In order to test the validity of the algorithm, the code has been written in python (colab notebook) and tested out with a simple email data set. The working images of the above has also be included in the later chapter 6[Results]. The detailed explanation and analysis of various component of the algorithm has been covered in the immediate chapters.

Chapter 3

Algorithm I

In this chapter, we are going to build algorithm that will define and set various types of tags in system, assign them to email that will be most suited to email by parsing subject and body of email into array of string. This string will be used as lookup for keywords that define various labels for email. Based on that, we will assign tag that will have highest priority.

3.1 Idea behind algorithm

To explain our approach, let us consider the fig 3.1, where there are three tags named as Academics, LMS, CDC. In Academics tag, we have multiple keywords set by user or application i.e Fee, Exam, Scholarship, and Academic, using which our algorithm can understand that if email have these keywords then it belongs to Academics Tag.

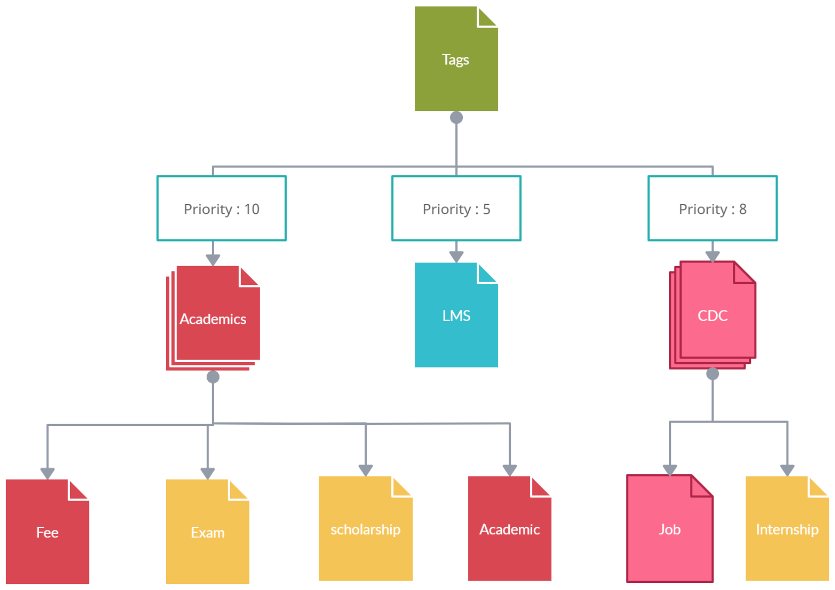


Fig. 3.1 Tag keywords example with priority.

Let us consider that, email also have Job keyword that is under CDC tag, and also satisfying Fee and Academics keyword of Academics tag, in this case we will consider tag with highest priority, In the fig 3.1, we have priority 10 for Academics tag where 8 for CDC tag and hence email will get tag Academic, with priority number 10 and color set to tag, here it is red.

3.2 Construction

Based on above mentioned idea, let us look at pseudo code of our algorithm which will have three main components i.e TextParser, GetPriority and SetTag methods.

Algorithm 1 TextParser(StringList)

```
1: for $Tag\ in\ taglist.Sorted(max(PriorityNum))$ do
2:     for $keyword\ in\ Tag.keywords$ do
3:         $bool\ contains\ =\ false$
4:         for $words\ in\ StringList:$ do
5:             $contains\ =\ String.Compare(keyword,\ words)$
6:         end for
7:         if $contains$ then
8:             $count++;$
9:         end if
10:    end for
11:    if $(count\ >=\ (0.8) * Size(StringList.keywords))$ then
12:        $return\ Tag.PriorityNum,\ Tag.Name;$
13:    end if
14: end for
```

In TextParser(StringList) method we are taking StringList as input parameter which had array of string that represent words of subject or message of email. At line 1, we considering tag from taglist which is sorted according to priority number associated with that tag in High to Low manner. This tag will have label, priority number, and keywords as shown in Fig 3.1. At line 2, we considering each keyword from keywords field of tag, then at line 3 we are keeping contains Boolean that checks whether keyword is in StringList in lines 4-5. If contains is true then we will increment counter count at line 8 that signifies number keyword that are present in StringList. At line 11-13, we checking whether count is greater than product of 0.8 and total number of keywords present in tag and if it holds true then we returning tag name and priority number. Here 0.8 is threshold that signifies more than 80% of keywords are present.

Algorithm 2 GetPriority(EmailData[] PooledEmail)

```
1:  for Email in PooledEmail : do
2:      SubjectStringList = [];
3:      for word in Email.Subject.split(" ") do
4:          SubjectStringList.append(word);
5:      end for
6:      SubjectPNum = TextParser(SubjectStringList);
7:      BodyStringList = [];
8:      for word in Email.body.split(" ") do
9:          BodyStringList.append(word);
10:     end for
11:     BodyPNum = TextParser(BodyStringList);
12:     EmailPNumPairList.append(makePair(Email.ID, Max(SubjectPNum,
        BodyPNum))
13:     EmailTagPairList.append(makePair(Email.ID, Argmax(SubjectPNum,
        BodyPNum).Totag))
14: end for
15: return True;
```

In GetPriority(EmailData[] PooledEmail) method, we are taking PooledEmail which is array emailData where emailData will have email ID, subject, sender and message field. At line 1, we are fetching email from array of emails. At line 2, we are initializing empty array to store words of subject from fetched email that we are storing at line 3-5. Then we are sending that array of word to TextParser Method to calculate appropriate tag and priority value that we are storing in variable SubjectPNum at line 6. Similarly at line 6-11, we are storing word of message i.e body of email in list and passing it to TextParser method and storing tag and priority value in variable BodyPNum. At line 12 and 13, we are appending maximum value of priority among SubjectPNum and BodyPNum and tag of that max value in EmailPNumPairList and EmailTagPairList array respectively in form of pair where first value is Email ID and second value is priority value and tag respectively.

Algorithm 3 SetTag(TagName, str[] keywords, PriorityNum=0, PriorityColor="Purple")

```
1: if $not\ exist$ : then
2:     $t = CreateTag(TagName);$
3:     $t.keywords = keywords;$
4:     $t.PriorityNum = PriorityNum;$
5:     $PNumPairList.append(makePair(PriorityNum,\ PriorityColor));$
6:     $TagPairList.append(makePair(TagName,\ t));$
7:     $TagList.append(t);$
8: end if
```

In SetTag(TagName, str[] keywords, PriorityNum=0, PriorityColor="Purple", Threshold = 0.8) method, we are taking five input parameters that are TagName that is name of user given tag, array of keywords that will be used to identify appropriate tag for email, priorityNum that is user given priority value from 0 to 10 where higher means high priority and by default it is 0, and PriorityColor is user given color for each priority which is by default purple for 0 priority value. At line 1, we are checking whether there is existing tag or not, if not then at line 2-4, we are creating tag object which will have name, keywords and priorityNum field and we are assigning them parameter values. At line 5, PNumPairList is a global array of pair where first value is priorityNum and second is color associated with it. At line 6, TagPairList is a global array of pair where first value is tag name and second value is tag object associated with it. These global arrays will be use by front end. At line 7, we are appending tag object in TagList global array of tag object which TextParser algorithm will use.

3.3 Algorithm analysis and Conclusion

In this chapter, we proposed a priority handler algorithm which has three components, that are TextParser which calculates priority and returns priority value along with suitable tag, GetPriority which take batch of emails and parse those email in array of string for subject and body and calls TextParser method and assign most suitable priority value and tag among those received for subject and body, and last is SetTag which creates new tag according to user given values and store it in various global arrays for front end and above methods.

For construction of TextParser algorithm. The complexity of sorting taglist is $O(t \log t)$ where t is number of tags in list, then for each tag it will take run for t times, then for each keyword it will run for k times where k is number of keywords in tag, for each word loop it will run for n time where n is number of words in stringList where internally it is comparing two string i.e keyword and word which will take $O(m)$ where m is length of word. Rest will be of constant time. Hence overall time complexity for TextParser algorithm is $O(t \log t + tknm)$ i.e $O(tknm)$ which is equivalent to $O(n^4)$.

For construction of GetPriority algorithm, for e emails we are running algorithm for e times, then for subject or body, we are storing word in list, that will take $O(n)$ where n is number of words. Then we are calling TextParser method over it which takes $O(tknm)$ time. Atlast, we appending value and tag to lists with max comparator which all take constant time hence overall time complexity will be $O(etknm)$ or to say $O(tknm)$ for each email which is equivalent to $O(n^4)$ per email. Construction of SetTag is of constant time except while checking tag is present in taglist which take $O(t)$ time where t is number of tags in taglist and hence will take $O(t)$ time overall.

Overall, our backend algorithm will run at $O(n^4)$ complexity which can be improved by using trie datastructure and unorder map instead of using arrays or lists.

Chapter 4

Algorithm II

Since the previous algorithm is of order $O(n^4)$ and there is room for improvement. We propose another algorithm for priority.

4.1 Improved Method

With the help of data structures we can further refine the algorithm. One such data structure which we can use here is using Tries[4]. It is an efficient data structure which uses BST to store keys and retrieve information in a faster order. In our project, Tries are used to parse subject and message body of emails where each word is getting inserted in trie tree in $O(n)$ where n is length of word i.e len(string). This trie tree will be then used to search specific keywords that are set in tags inside this tree in $O(n)$ where n is length of keyword. In this manner we are reducing time complexity of searching string in array of string that takes O(mnk) for each tag to O(nk) which is reducing power of 4 to 3.

where,

- m is size of string array storing words of message body or subject.
- n is length of word we are searching
- k is size of string array storing keywords of tag.

Let us look at a sample Trie tree for words :- APPLE, ARE, ORDER and ORE.

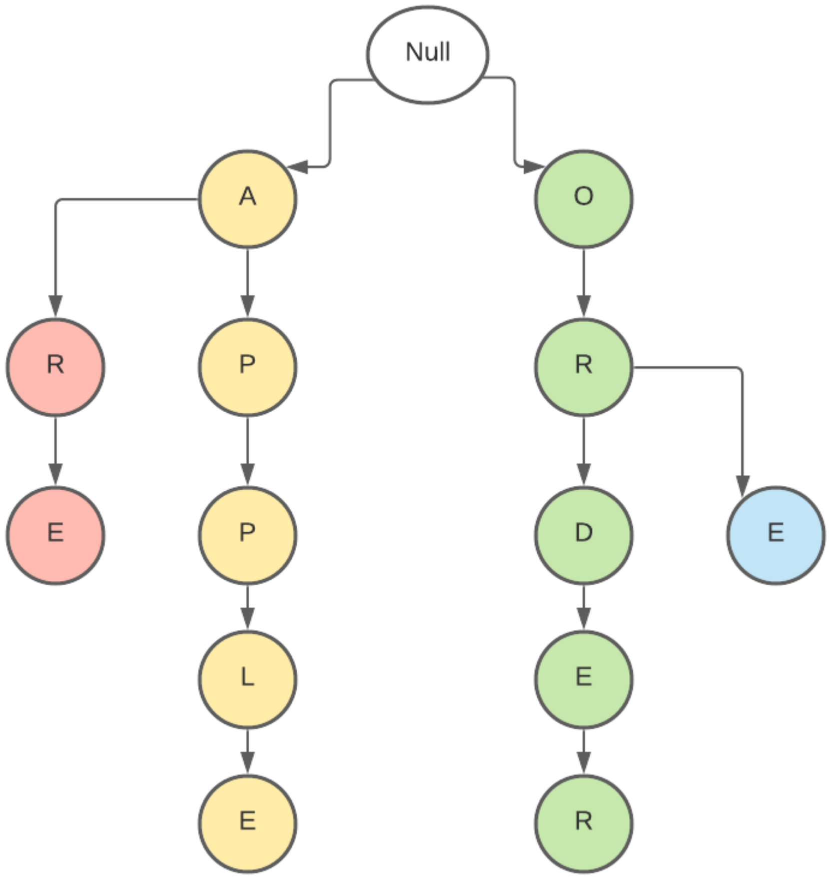


Fig. 4.1 Trie example forAPPLE, ARE, ORDER and ORE.

Another datastructure that we are using is unordered map, as search, insert in unordered map in constant time i.e $O(1)$ using key value pair while list of pair takes $O(n)$. Traversal in both cases to find if key exist takes $O(n)$.

Also in previous algorithm, we kept threshold at value 0.8, which means for each tag, 80% of keyword must present in message or subject, but let us consider fig 3.1 again, where there are two keywords for CDC, that are Job and Internship, but let us assume that CDC is sending mail for internship, but in that case only one of two keywords will be satisfied and hence it will only have threshold value of 50% which is less than 80% which will make our algorithm to not consider CDC tag. In order to resolve this, we came up with dynamic threshold value which will be different for various tags according to their type and priority value.

Understanding that Trie and unorder hashmap can improve our algorithm and dynamic threshold will bring fairness to algorithm, let us have a look at the new modified algorithm.

4.2 Construction of improvised algorithm

As most of the structure and parameter names are similar to that of algorithm proposed in chapter 3, we will only discuss changes in following algorithms.

Algorithm 4 TextParser(DataTree)

```
 1: for $Tag\ in\ taglist.Sorted(High\ to\ Low:\ PriorityNum)$ do
 2:     for $keyword\ in\ Tag.keywords$ do
 3:         if $(DataTree.contains(keyword))$ then
 4:             $count++;$
 5:         end if
 6:     end for
 7:     if $(count >= Tag.Threshold * Size(DataTree.keywords))$ then
 8:         $return\ Tag.PriorityNum,\ Tag.Name;$
 9:     end if
10: end for
```

Given that:- n = length of word , t = number of tags, k = number of keywords, m = number of words in array of string in Algorithm 1

As instead of searching in array of string, now we are searching keywords in DataTree which is trie [4] which have all words in form of BST as shown in Fig 4.1, which takes

$O(n)$ to search word comparing to array which takes $O(mn)$. TextParser will have a time complexity of $O(ntk)$.

Another change here is that we using Tag.Threshold at line 46 instead of 0.8 value which will bring fairness as discussed earlier.

Algorithm 5 GetPriority(EmailData[] PooledEmail)

```
1: for $Email\ in\ PooledEmail$ : do
2:   $TreeNode\ DataTree = TreeNode();$
3:   for $word\ in\ Email.Subject.words$ do
4:     $DataTree.add(word);$
5:   end for
6:   $SubjectPNum = TextParser(DataTree);$
7:   $DataTree.clear();$
8:   for $word\ in\ Email.body.words$ do
9:     $DataTree.add(word);$
10:  end for
11:  $BodyPNum = TextParser(DataTree);$
12:  $EmailMap.add(Email.ID,\ Max(SubjectPNum,\ BodyPNum))$
13:  $EmailTag.add(Email.ID,\ Argmax(SubjectPNum,\ BodyPNum).Totag)$
14: end for
15: $return\ True;$
```

Also Given that:- e = number of email, a = number of words in subject b = number of words in body. At line 2, we are initializing empty trie[4] names as DataTree instead of list in previous chapter. Also we using same DataTree for subject as well as for body at line 4 and 9, just at line 7, we are erasing previous data from it which happens in $O(1)$. As insertion of word in trie takes $O(n)$, time complexity for data insertion will be $O(an)$ and $O(bn)$ for subject and body respectively. TextParser now will take $O(ntk)$ as discussed above for both cases. Another change is that we have replaced array of pairs with unordered map for storing priority value and tag and hence reduced time complexity for lookup for front-end from $O(n)$ to $O(1)$ which will indeed make UI response faster.

Hence, GetPriority will have a time complexity of $O(e.max[O(an), O(bn), O(ntk)]) \approx O(entk)$ or in other words $O(ntk)$ per email which has been reduced from $O(mntk)$ per email. To simplify, reduced from $O(n^4)$ to $O(n^3)$.

Algorithm 6 SetTag(TagName, str[] keywords, PriorityNum, PriorityColor, Threshold = 0.8)

```
1: if $not\ exist$ : then
2:     $t = CreateTag(TagName);$
3:     $t.keywords = keywords;$
4:     $t.PriorityNum = PriorityNum;$
5:     $t.Threshold = Threshold;$
6:     $PNumMap.add(PriorityNum, PriorityColor);$
7:     $TagMap.add(TagName, t);$
8:     $TagList.add(t);$
9: end if
```

SetTag will have the time complexity of $O(t)$, as same with that of chapter 3 but, key change here is that we have replaced array of pairs with unordered map for storing priority value with color and tagname with tag and hence reduced time complexity for lookup for front-end from $O(n)$ to $O(1)$ which will indeed make UI response faster.

4.3 Conclusion

In this chapter, we proposed priority algorithm for our smart email client. This algorithm has time complexity bottled by the GetPriority algortihm and TextParser algorithm by using trie and unordered map i.e. of the $O(n^4)$ to $O(n^3)$ as discussed above.

This algorithm can be further optimized in terms of accuracy by using natural language processing (NLP) which have lemmatizer, stemmar and so on, which reduces redundant words as well as reduces length of word which will reduce time for calculation at some instinct and with the help of Machine Learning and data processing technique such as regression, classifier, clustering, PCA, etc., better heuristic search can be built which also offer multiple tags e.g one mail can be from Academics but also have Deadline with it and hence can have two tags of Academics and Deadline. This further optimization will be part of Phase 2 and also subject to availability of resources for app development.

In next chapter, we will discuss about future work that will be part of Phase 2.

Chapter 5

Future Work

After the end of phase I we have a client interface, as per the future work we will integrate the already created priority algorithm. As a final result we will have descent email client that will fetch, send, categorize emails with labels and tags according to customizable priority handler. Once we achieve this functionality we shall move on to adding additional features for the application. We will try to provide more flexible configuration setting to user to make user experience great.

5.1 Improvisation of priority handler

Our primary goal, will be to enhance our priority algorithm, to provide advanced categorisation of emails. e.g When there is deadline due, that mail will have highest priority but once deadline passed then that mail will have lower priority according to importance of email set by priority handler based on experience. Moreover, we shall improvise our heuristic for the priority algorithm and in the likely scenario, if possible we will integrate some NLP into the algorithm for better efficiency.

5.2 Addition of features

We will expand our email client by adding some additional features which will be keep adding on as time passes and availability of resources i.e time, system requirements, availability of APIs, documentation, articles, etc.

Some of the features are following:

- Application will have google assistance integration for hands free operation using voice command.
- Application will have speech-to-text and text-to-speech functionality to support flexible read and write experience to users.
- Application will have google calendar integration with reminder system to alert user about deadlines and events. It will also have customizable notifications that will have various tones according to priorities and category.
- Application will have chatting functionality where user can chat with other users similar to whatsapp, discord, etc.

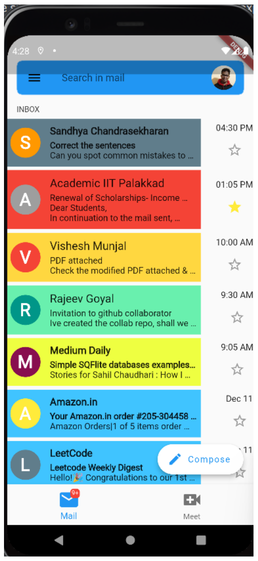
In next chapter, we will see some results of progress of project so far.

Chapter 6

Results

In this chapter, we showcase the results of our project's phase I and explain briefly about them. Without further ado let us jump right into the first set of images. We have referred design documentation of flutter.[3]

6.1 Smail Front End



Sandhya Chandrasekharan 04:30 PM
Correct the sentences
Can you spot common mistakes to ...

Academic IIT Palakkad 01:05 PM
Renewal of Scholarships- Income ...
Dear Students,
In continuation to the mail sent, ...

Vishesh Munjal 10:00 AM
PDF attached
Check the modified PDF attached & ...

Rajeev Goyal 9:30 AM
Invitation to github collaborator
Ive created the collab repo, shall we ...

Medium Daily 9:05 AM
Simple SQFlite databases examples...
Stories for Sahil Chaudhari : How I ...

Amazon.in Dec 11
Your Amazon.in order #205-304458 ...
Amazon Orders|1 of 5 items order ...

LeetCode
Leetcode Weekly Digest
Hello! 🎉 Congratulations to our 1st ...

Fig. 6.1 User's Main Page

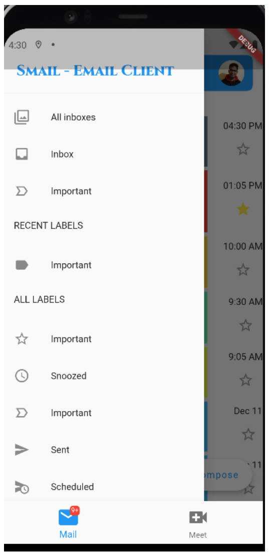


Fig. 6.2 User's Tool Bar

Figure 6.1 shows the User's main interaction page. In the screenshot we can see that the UI is similar to that of the official Gmail app with all the basic features such as user pictures, sender's name and subject along with the gist of the email body. The page also handles the minute details such as display of the time and star option for respective emails. Also, provides a toggle between meet and mails. The screenshot displays the sample user Sahil's emails and It can be clearly seen that the emails are classified according to priority and assigned a specific color to them. On the other hand, the neighbouring figure(Fig 6.2) shows the drawer tool bar for the user that has to be integrated with their respective functionalities in the next phase.

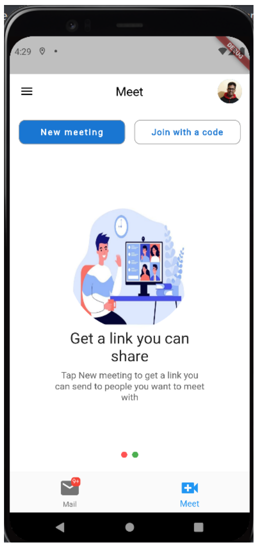

4:29
Meet
New meeting
Join with a code
Get a link you can share
Tap New meeting to get a link you can send to people you want to meet with
Mail
Meet

Fig. 6.3 Meet Main Page

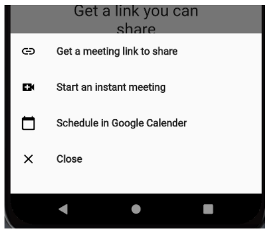
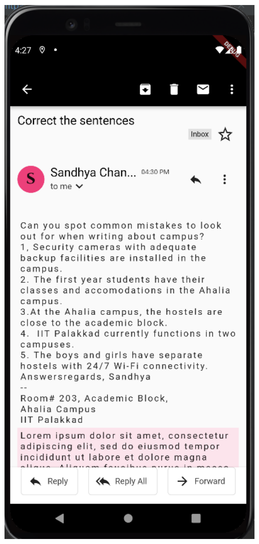


Fig. 6.4 Meet Schedule Page

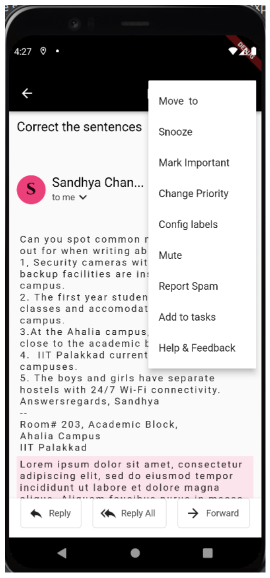
Figure 6.3 and Figure 6.4 show's the user's meet page which can be used by the user to join or host a meeting. In order to make convenience for the user similar to the Gmail app we have added the meet options in our app to make sure that the user can join meeting, start a meeting or schedule a meeting for the future.

Figure 6.5 display's the Mail page i.e. the page that opens and displays the body of the mail. In the screenshot it can be clearly seen that we are able to read the random text of the email body as well as the features such as star, time, and reply feature as well. On the other hand, the Figure 6.6 showcase the user's options for that particular email. The options include the basic options such as muting, reporting spam, marking important and



Can you spot common mistakes to look out for when writing about campus?
1, Security cameras with adequate backup facilities are installed in the campus.
2. The first year students have their classes and accomodations in the Ahalia campus.
3.At the Ahalia campus, the hostels are close to the academic block.
4. IIT Palakkad currently functions in two campuses.
5. The boys and girls have separate hostels with 24/7 Wi-Fi connectivity.
Answersregards, Sandhya
--
Room# 203, Academic Block,
Ahalia Campus
IIT Palakkad
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna

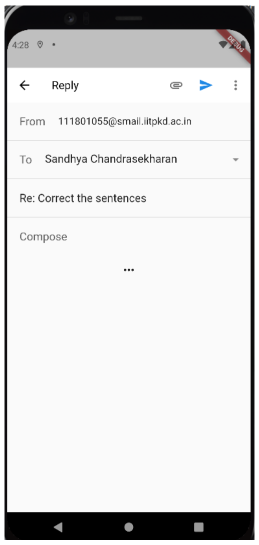
Fig. 6.5 Mail Page



Can you spot common r
out for when writing ab
1, Security cameras wit
backup facilities are in
campus.
2. The first year studen
classes and accomodat
campus.
3.At the Ahalia campus,
close to the academic b
4. IIT Palakkad current
campuses.
5. The boys and girls have separate hostels with 24/7 Wi-Fi connectivity.
Answersregards, Sandhya
--
Room# 203, Academic Block,
Ahalia Campus
IIT Palakkad
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna

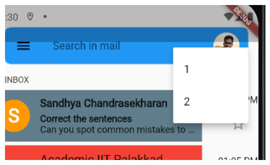
Fig. 6.6 User Configuration Option's

the added option i.e. the changing priority option (to be integrated in phase 2).



From 111801055@smail.iitpkd.ac.in
To Sandhya Chandrasekharan
Re: Correct the sentences

Fig. 6.7 Reply Page



Sandhya Chandrasekharan
Correct the sentences
Can you spot common mistakes to ...
Academic IIT Palakkad

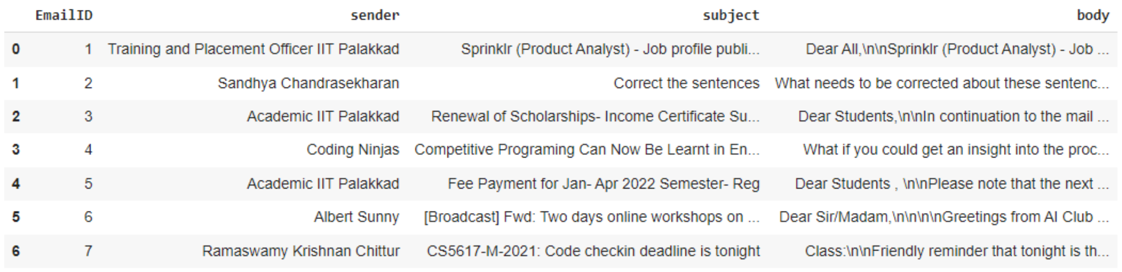
Fig. 6.8 User Accounts toggle

Figure 6.7 shows the reply page for a respective email, let's say as shown in the image we want to reply the "correct the sentence" email, after clicking on the reply option in the Figure 6.5 shown page we are redirected to our current page allowing us to compose a reply email for that particular "correct the sentences" mail.

Figure 6.8 shows the account handler basic setup, as per the architecture, a user can allow various account and figure 6.8 highlights this feature by allowing the user to toggle between accounts.

6.2 Priority Algorithm Results

As mentioned earlier in the Chapter 2[Progress], in order to validate the working of the designed priority algorithm, we worked out the algorithm separately in a colab notebook. In this validity test we first created an email data set containing emails of various types, namely, from academics, deadline mails from professors, mails from TPO etc. Now each of these sample mails have different context which may possibly contains certain important keywords that will help identify the tag for respective emails.

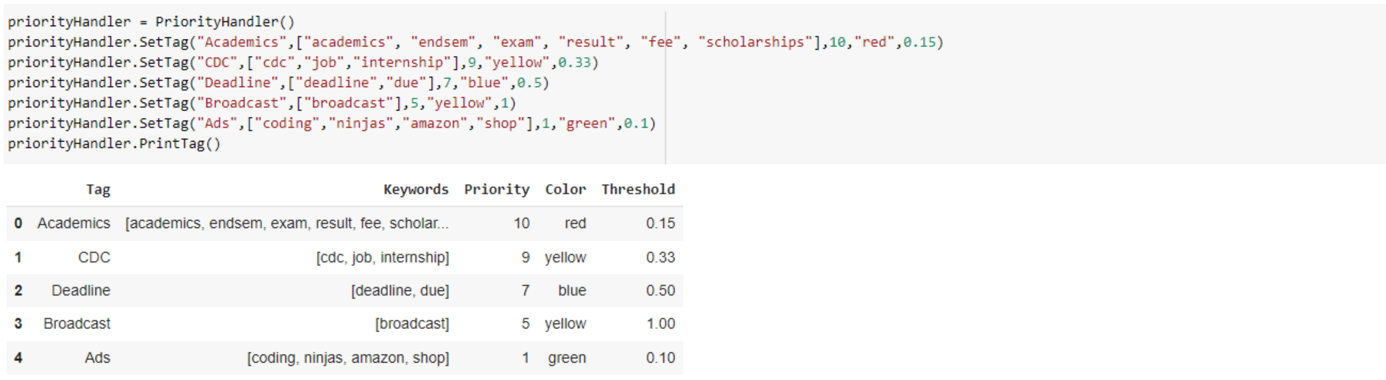


	EmailID	sender	subject	body
0	1	Training and Placement Officer IIT Palakkad	Sprinklr (Product Analyst) - Job profile publi...	Dear All,\n\nSprinklr (Product Analyst) - Job ...
1	2	Sandhya Chandrasekharan	Correct the sentences	What needs to be corrected about these sentenc...
2	3	Academic IIT Palakkad	Renewal of Scholarships- Income Certificate Su...	Dear Students,\n\nIn continuation to the mail ...
3	4	Coding Ninjas	Competitive Programing Can Now Be Learnt in En...	What if you could get an insight into the proc...
4	5	Academic IIT Palakkad	Fee Payment for Jan- Apr 2022 Semester- Reg	Dear Students , \n\nPlease note that the next ...
5	6	Albert Sunny	[Broadcast] Fwd: Two days online workshops on ...	Dear Sir/Madam,\n\n\n\nGreetings from AI Club ...
6	7	Ramaswamy Krishnan Chittur	CS5617-M-2021: Code checkin deadline is tonight	Class:\n\nFriendly reminder that tonight is th...

Fig. 6.9 Sample Emails

Figure 6.9 display the email data set in an orderly manner so that we can get the gist of the type of emails used in the data set.

Now in order to determine the priority or to even classify the set of emails, we need to know the tag list and the pre-set keywords for that particular tag. The keywords can be identified in the mail in order to map it to the respective tag, hence classifying it. Figure 6.10 displays the sample tags and their respective keyword set. One more thing is shown in Figure 6.10 which is the threshold of a particular tag. Now one may ask why is there a threshold involved. The answer to that question is that each tag has a criteria that the email has to pass in order to be classified for that tag. In layman terms, the email should



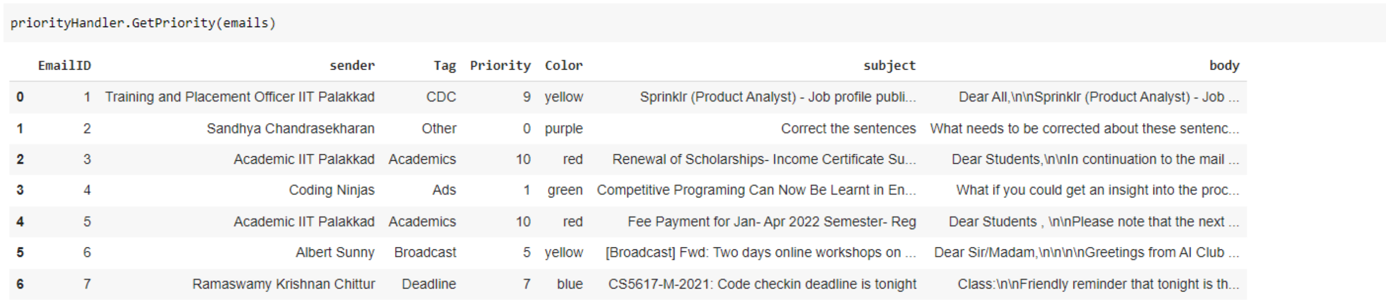
```
priorityHandler = PriorityHandler()
priorityHandler.SetTag("Academics",["academics", "endsem", "exam", "result", "fee", "scholarships"],10,"red",0.15)
priorityHandler.SetTag("CDC",["cdc","job","internship"],9,"yellow",0.33)
priorityHandler.SetTag("Deadline",["deadline","due"],7,"blue",0.5)
priorityHandler.SetTag("Broadcast",["broadcast"],5,"yellow",1)
priorityHandler.SetTag("Ads",["coding","ninjas","amazon","shop"],1,"green",0.1)
priorityHandler.PrintTag()
```

	Tag	Keywords	Priority	Color	Threshold
0	Academics	[academics, endsem, exam, result, fee, scholar...	10	red	0.15
1	CDC	[cdc, job, internship]	9	yellow	0.33
2	Deadline	[deadline, due]	7	blue	0.50
3	Broadcast	[broadcast]	5	yellow	1.00
4	Ads	[coding, ninjas, amazon, shop]	1	green	0.10

Fig. 6.10 Defined Tags and Priority

contain a minimum amount of keywords from a particular tag to belong to it.

Once our algorithm explained above in chapter 5[Algorithm II] works its magic, we can see that all the emails are assigned to a particular tag and given a respective priority which then can be mapped to a color to assign the colour. Or a set of emails can also be sorted according to such priority. Figure 6.11 shows the final output in an orderly manner that we get from applying our algorithm for the given email data set and tag list.



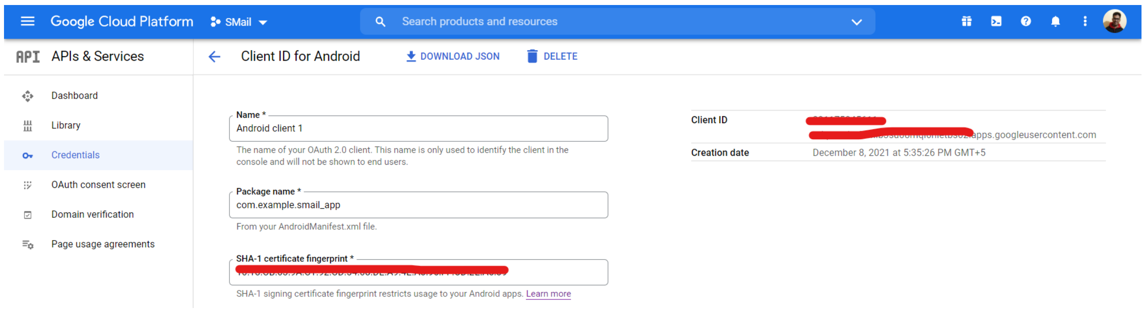
```
priorityHandler.GetPriority(emails)
```

	EmailID	sender	Tag	Priority	Color	subject	body
0	1	Training and Placement Officer IIT Palakkad	CDC	9	yellow	Sprinklr (Product Analyst) - Job profile publi...	Dear All,\n\nSprinklr (Product Analyst) - Job ...
1	2	Sandhya Chandrasekharan	Other	0	purple	Correct the sentences	What needs to be corrected about these sentenc...
2	3	Academic IIT Palakkad	Academics	10	red	Renewal of Scholarships- Income Certificate Su...	Dear Students,\n\nIn continuation to the mail ...
3	4	Coding Ninjas	Ads	1	green	Competitive Programing Can Now Be Learnt in En...	What if you could get an insight into the proc...
4	5	Academic IIT Palakkad	Academics	10	red	Fee Payment for Jan- Apr 2022 Semester- Reg	Dear Students , \n\nPlease note that the next ...
5	6	Albert Sunny	Broadcast	5	yellow	[Broadcast] Fwd: Two days online workshops on ...	Dear Sir/Madam,\n\n\n\nGreetings from AI Club ...
6	7	Ramaswamy Krishnan Chittur	Deadline	7	blue	CS5617-M-2021: Code checkin deadline is tonight	Class:\n\nFriendly reminder that tonight is th...

Fig. 6.11 Calculated Priority for Respective Mails

6.3 Google API Credentials

As mentioned in chapter 2, Progress, we have enabled GMAIL API from google developer console[5] and also created credentials for it to use in our application.



Client ID for Android

Name *
Android client 1

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

Package name *
com.example.smail_app

From your AndroidManifest.xml file.

SHA-1 certificate fingerprint *
[redacted]

SHA-1 signing certificate fingerprint restricts usage to your Android apps. Learn more

Client ID	[redacted]pps.googleusercontent.com
Creation date	December 8, 2021 at 5:35:26 PM GMT+5

Fig. 6.12 Google API credential for application

In Fig 6.12, In credentials tab, we have given Android client 1 name for our credential client ID, then we have provided package name of application which one can find manifest.xml file inside project work space, then we have also provide SHA1 key, which is unique for each work space. After giving all information to google, they will take some time to validate those information, and will provide us unique ClientID using which we can access OAUTH of google which is responsible to handle google sign-in for application and will also provide security. Since SHA1 key and ClientID is sensitive data, hence hidden in Fig 6.12.

Chapter 7

Conclusion

In conclusion, it is safe to say that after thorough research and analysing various components, we were able to design the Interface for the Smail - Smart Email Client. Taking baby steps towards the aim of the full fledged application. In layman terms, we have handled the front end, i.e. the outlook of the application, made decisions over its architecture. Also, we handled the authentication to establish the credentials for the use of the Google API's. Apart from all this, we have also focused on the back bone of our project namely, the priority algorithm, after passing it through various filters we refined it more and more to improve its time complexity substantially. For now the algorithm is a simple heuristic depending upon certain keywords to determine priority in regards to a certain tag but later we plan to improve the heuristic. Moreover, in order to test and verify the validity of algorithm we have done a demo test as well.

In regards to the future steps of the Smail client, we have set certain goals named as phase II which we shall achieve in the upcoming times. As of now we have worked upon the above mentioned things and are debugging and analysing our code for the demo purposes.

7.1 Contribution

In regard to contribution, it has been a duo effort of the team members to bring the progress of the project to where it stands today. We together designed and composed the priority algorithm. After critical thinking and figuring out efficient ways we designed a heuristic based algorithm. Moreover, both of us have contributed to interface of the application. In regards to front end, many aspects of it has been designed and coded by Sahil while Vishesh main role included designing some aspect and debugging the errors as well. In short, both of us have worked towards the advancement of the project and will continue to do so in the near future as well.

References

[1] “Googleapis in flutter documentation.” [Online]. Available: https://docs.flutter.dev/development/data-and-backend/google-apis

[2] “Googleapis in dart documentation.” [Online]. Available: https://pub.dev/packages/googleapis

[3] “Flutter design documentation.” [Online]. Available: https://docs.flutter.dev/resources/design-docs

[4] “Trie datastructure.” [Online]. Available: https://www.geeksforgeeks.org/trie-insert-and-search/

[5] “Google developement console.” [Online]. Available: https://console.cloud.google.com