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

<u>CERTIFICATE</u>

*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 careers.

Abstract

The Smart Email Client is an android application with a custom build AI to categorize and prioritize emails via easily customizable colored 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, a reminder system with custom notifications according to priority, text-to-speech, and speech-to-text integration.

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

Contents

List of Figures

Chapter 1

Introduction

The Smail - Smart Email Client is an android application with a custom build AI to categorize and prioritize emails via easily customizable colored 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 in itself has been a journey that is now in its final stage, where we have successfully developed a full-fledged working app. During the second part of phase II, we integrated the priority algorithm and features in the application as well as tested them for our IIT Pkd smail accounts. Not only this a configuration page was also created for the users to allow them to configure the respective tags as per their needs.

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 the bombardment of mails from all subdivisions 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 handy. Thus the project Smail emerged.

1.2 Finalization of Project

The already set goals for the project have been accomplished i.e. the priority algorithm has been integrated along with the pre-processor in the back end. With Regard to the extended features such as Text-to-speech for user convenience, calendar event setup, alarm setup to avoid any deadline, and date detection for the essential details of an event, have been implemented. It goes without saying that the app can be even further refined as per needs with the addition of various advanced features etc. In fact, we plan to lay out an improvement plan in the possible future modification chapter.

1.3 Overview of The Report

This chapter dealt with the introduction of the Smail client along with the motivation behind it and also, goes into the finalization of the project. In the next chapter i.e. chapter 2, we give a brief overview of previous works and talk about the progress accomplished in the project up-till date in phase II. In chapters 3 and 4, we give a summary of the previous algorithm and the finalized one. In chapter 5, we discuss possible modifications in accordance with our vision. In chapter 6, we showcase the results of the work done. We will close our report with the conclusion and contribution of each member toward the project. And Lastly will include all the references that we have used for this project so far.

Chapter 2

Review of Prior/Current Works

This chapter provides a detailed analysis of the progress made in phase II and a brief overview of the tasks accomplished in phase I.

2.1 Phase I overview

Phase I started with careful thorough research and analyzing various components such as installing Android Studio on Windows and flutter. We also fixed dart language for designing and displaying it on the device emulator. Due to this, we were able to design the Interface of Smail - Smart Email Client. Made important decisions on the architecture. Also, we deal with the Google API's authentication to establish the credentials for use of google API(done in this phase). Last but not least we focused on the backbone of the project i.e. the priority Algorithm, and after multiple refinements, we made an acceptable algorithm for the priority handler of the app along with working demos.

2.2 Phase II Midsem overview

1. **Gmail API Study:**

   This was the research part about Gmail APIs[1] where we also created a map of

GmailApi instance (Figure 2.1) for better clarity. Using the map we were able to get the message instance which stores all information regarding a particular mail. Thereafter, we focus on 3 properties namely:- MIMEtype[2](for the layout of mails), parts(for the body of mail), headers(for the subject, time, sender, etc.)

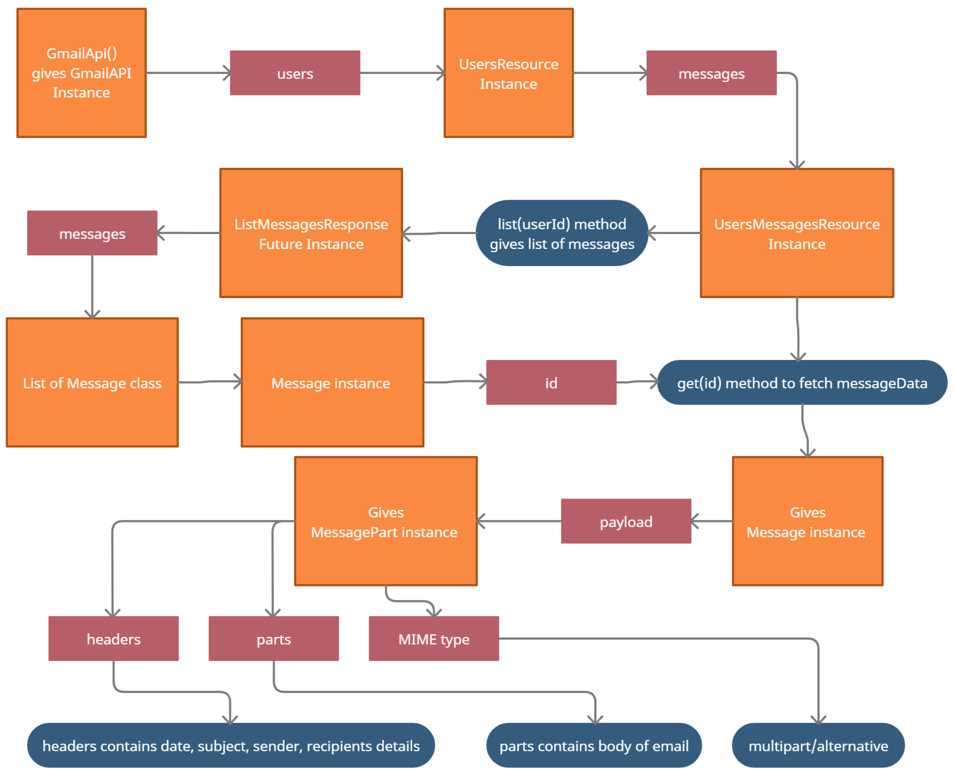

GmailApi() gives GmailAPI Instance
users
UsersResource Instance
messages
UsersMessagesResource Instance
list(userId) method gives list of messages
ListMessagesResponse Future Instance
messages
List of Message class
Message instance
id
get(id) method to fetch messageData
Gives Message instance
payload
Gives MessagePart instance
headers
parts
MIME type
headers contains date, subject, sender, recipients details
parts contains body of email
multipart/alternative

Fig. 2.1 GmailApi Instance.

2. **OAUTH Integration:**

   In order to get authorization, we have to mention in scopes of class constructor that we need permission regarding read-only of gmail data of user. Once added, and run, on opening of application when user sign in to app, it will ask permissions to user whether to allow or decline to view their gmail data. Hence, we can initialize OAUTH with scope[3] for readonly gmailApi.

3. **Integration Of Gmail API:**

Using the flowchart in Fig 2.1, we fetch the list of messages using list method. After this, we iterate over the results to get id of each message. Once done, now we extract the body of mail using messageData, and iterate each header to get data such as sender, date, time, recepients etc. And finally store it in AppMessage to present emails to the front end.

4. **Priority Algorithm Finalisation/Modification:**
   In the phase II Mid Semester, we had finalised the priority algorithm. The final algorithm could support multiple tags, plus the keyword list of a tag is automatically updated using the LFU replacement policy to get the new keywords for a particular tag. Not to forget that these additions and changes to the algorithm came at minimal cost in regards to complexity of algorithm.

5. **Front End Modification:**
   Throughout the project, with the main goal of achieving a working app in mind, we did consider aesthetic appeal. Front end has been modified a lot to appease the user. These modifications ranges from making a logging page to adding a configuration page to edit tags to signing out. More changes/tweaks have been made to the app using flutter widgets[4] which will be thoroughly discussed in this latest report.

2.3 Phase II Final progress

2.3.1 Designing of Text Pre-processor

In order to reduce computational burden on TextParser algorithm that is mentioned in chapter 4, we have designed some Preprocessor steps which does not affect overall time complexity of algorithm but reduce computational power from TextParser which takes most of time in calculation of priority. In Text Preprocessor, following will be done:

- **Removal of Stop Words:** Stop words are those word which get used in grammar of language but on their own they does not contribute towards extraction of conclusion. Some examples of Stop words are a, an, the, when, where, is, are, am, etc. These are useful in term of grammar but in our case to determine whether mail belong to Academics tag or CDC tag, they won't be helpful, hence they will be get removed using Trie data structure which will store stop words in it, and then we will pass each word of subject/body of email one by one to it and in case stop word detected, we will drop it from list of words of email.

- **Removal of Punctuation marks:** As punctuation marks are also come along with words and sometimes count as a word with single character, all which sometimes confuses TextParser algorithm. To avoid this, we will remove all punctuation marks before passing text to TextParser.

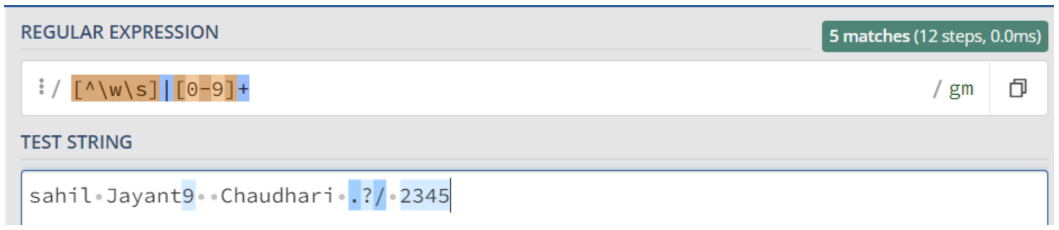
- **Removal of Numbers**: We are using keywords of type string to detect tags for emails which does not consider number in it and numbers also get considered as separate word which burdens TextParser on further extent, hence will be removed along with Punctuation marks.

- **Removal of extra spaces:** Similar to Punctuation Marks, sometimes extra spaces occur in text which wastes calling of TextParser instance to process and hence will also be removed.

In order to remove numbers and punctuation marks, we will use following regular expression.

```
r'[^\w\s]|[0-9]+'
```

In above regular expression, `[^\w\s]` means that we will leave all words and spaces, `|` stands for or, and `[0-9]+` means that all digits from 0 to 9 and + means that multiple occurrences. So if we use this regular expression with replace method with replacement with `''` then all digits and punctuation marks will be removed.


REGULAR EXPRESSION

5 matches (12 steps, 0.0ms)

```
/ [^\w\s]|[0-9]+ / gm
```

TEST STRING

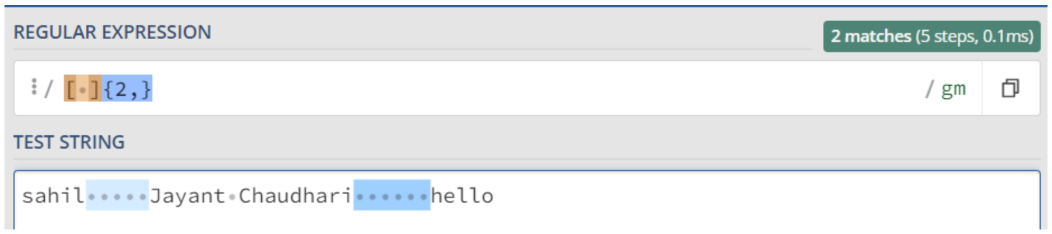
```
sahil Jayant9  Chaudhari .?/ 2345
```

Fig. 2.2 Regular Expression to remove numbers and punctuation marks.

In above figure we can see that, our regular expression is correctly detecting all numbers and punctuation marks by highlighting them in test string. We have used regex101.com[5] website to test our regular expressions.

```
r'[ ]{2,}'
```

This regular expression will detect consecutive occurrences of more than or equal to two spaces and similarly using replace method, extra spaces will be removed.


REGULAR EXPRESSION

2 matches (5 steps, 0.1ms)

```
/ [ ]{2,} / gm
```

TEST STRING

```
sahil     Jayant Chaudhari      hello
```

Fig. 2.3 Regular Expression to remove multiple spaces.

In above figure we can see that, our regular expression is correctly detecting all multiple spaces i.e more than one space by highlighting them in test string. We have used regex101.com[5] website to test our regular expressions.

Above steps will be integrated with GetPriority Algorithm in chapter 4 with revised pseudo algorithm and time complexity. These steps will reduce total words that TextParser will need to process by larger extent as stop words, punctuation marks, number and additional spaces almost conquered more than 50 percentage of words in text body of emails.

2.3.2 Priority Algorithm Integration

To integrate priority algorithm to application there were four files were created under models folder. These files will be our back-end which will communicate with front-end which is stored in tab folder and entire application will be executed using main.dart file. Back-end files are as follows:

- **user_model.dart:** This file contains UserData class which stores name of user, emailID, unique id that algorithm assigns and color URL for background of user avatar. This class will be used by AppMessage class stored in message_model.dart.
- **message_model.dart:** This file contains AppMessage class which stores all information about email includes UserData object to store sender's information, email ID, unique ID given by Gmail API, subject, body text, date, time and priority number which will be used by front end to display email with proper priority color. It also contains gmailMessage class which fetches email data from GmailAPI and builds AppMessage class and store it in list and at end call priority algorithm to find tags for emails which is stored in priority_handler.dart file.
- **tag_model.dart:** This file contains tag class that we have defined in priority algorithm.
- **priority_handler.dart:** This file contains priority algorithm class which have methods that we will use to set tag for each emails includes setTag, TextParser, GetPriority, editTag, sortMails etc functions which will be discussed in following subsection and in chapter 4.

Once all of these implemented, we created static instance of gmailMessage class in front-end in file frontpage.dart. To access priority algorithm for it's map for email, tag and priority color, we will access it by calling gmailMessage.priorityHandler.mapName. This way all of the back-end values were integrated with front end.

2.3.3 Configuration of tags

In SMail application, we are initializing some tags by default using initTag function defined in priorityHandler class. But user might want to change, delete or edit tag or its properties. User even might want to add some extra tags according to their comfort. Hence we have implemented config bottom navigation bar tab in our front-end. Which have following options:

- **Add Tag Drop-down:** In this option, user will be asked to click on button which will redirect user to form which will ask for tag details to fill such as tag name, keywords, color, threshold and priority value.
- **Edit Tag Drop-down:** In this option, when user will click on drop-down, list of available tags will be shown. On clicking one of them will redirect user to form where it will show existing values of tag class property and will ask user to change some of it or add additional keywords to it. Once submitted, changes will be added. Remove option is also present to delete tag.
- **Refresh button:** Once made some changes i.e add tag, edit tag or remove tag, and click on refresh button changes will be reflected in priority algorithm and hence to front end.

For addition of Tag, Front-end will provide values entered by user to back-end where back-end will simply call setTag method which takes $O(t)$ time complexity, where t is number of tags present in tag list. For removal of Tag, we simple erase tag class from tag list and for edition of Tag, we will fetch old values, reflect user entered values with it, delete older class and will add new class which will overall takes $O(t)$ time complexity.

2.3.4 Integration of features

The uniqueness of our app partially depends upon the priority algorithm and partially on the features that it supports to make the user experience good. With the guidance from our Mentor Dr. Albert, we were able to add various features in our application. Although, it wasn't an easy task. We faced various difficulties among which time constraint was the biggest one, but we crossed such hurdles and implemented the following features in our application:-

- **Mail Sorting according to priority:**
    - **Ideology and Motivation:** Mail Sorting is a simple yet amazing feature of our application. It simply sorts the mail according to the priority which is simply convenient for the user. The motivation was to highlight important mails on the top so that the user don't miss out on them.
    - **Implementation:** To implement this feature, we have created helper function and some changes in priorityHandler class. Those changes are, we have created additional map that stores AppMessage class of email as value under key of unique id of mail given by GmailAPI[1]. We also created sortedMessageList list to store sorted ids of mail according to priority number that is stored in emailMap map. Once priority handler finishes calculation of tags for mails we will use those value to sort emails which takes time complexity of $O(nlog(n))$. We will also make sure that date and time is also sorted where newest email will be at top and oldest email will be at bottom withing same priority number.

Algorithm 1 sortMessages()

```
1: $emailMap.sort(by\ mailMap[emailMap.id].dateTime)$
2: $emailMap.sort(by\ value)$
3: $sortedList = emailMap.keys.toList()$
4: return $sortedList$
```

In above algorithm 1, emailMap is the map which stores key value pair of id of mail and priority number associated with it. At line 1, we are sorting emailMap according to dateTime of email which we are retrieving from mailMap which stores AppMessage object w.r.t id of mail. Then at line 2, we are sorting emailMap by priority number which is it's value and then we are fetching sorted id of mails by using keys attribute of map and returning it. The time complexity of it will be $O(nlog(n))$.

- **Text-to-speech:**
  - **Ideology and Motivation:** As the name suggest Text to speech converts the text i.e. the mail body to speech. The feature was suggested by our mentor. Also, can come in handy as a part of audible mails.
  - **Implementation:** To implement Text-to-speech, we used flutter package flutter_tts[6] which reads string of words from mail body and convert it into vocal. We added bottom navigation tab in mail front end where on clicking button of text-to-speech, user will be redirected to the page where user will have various options such as changing pitch, volume and speed of reader as well as changing of language in case mail is having another language other than English. When user will click play, body of mail will be passed to flutter_tts engine which will start reading according to settings.
- **Speech-to-text:**
  - **Ideology and Motivation:** Speech to text is a handy feature for the user as it allows them to compose the mail body simply by narrating the mail contents. This feature may be extremely helpful for certain users who may have disabilities.
  - **Implementation:** To implement speech-to-text, we used flutter package called speech_to_text[7] which detected user voice and tries to detect word from it. We added this feature inside compose option, where user have to type body of mail, so Instead of typing, user can click on microphone button and speak, engine of speech_to_text will try to detect it and will display in text field of mail section.

User can edit false detected words.

- **Calendar Event Setup:**
  - **Ideology and Motivation:** Calendar Event Setup feature is a routing feature providing easy access to the user in case they want to setup reminders or so.
  - **Implementation:** To implement calendar event setup, we have used flutter package called external_app _launcher[8] which opens external app using android play store URL and IOS store URL, if installed then simply opens application, else take user to installation page on respecting app store. We have added bottom navigation bar tab in mail page where user can click on event tab to create event in google calendar application. Since google calendar application is present in every android phone by default and event of google calendar is accessible from every device, we redirect user to google calendar, as creating new calendar inside application will be resource consuming and will also increase application size and parallely downgrade performance. User will read mail body and if user wanted, they can create event. Once event is created on google calendar, clicking back button will bring user back to SMail application, to mail page from where user clicked button.

    Appstore link for google calendar is `'https:`/play.google.com/store/apps/details?/ `id=com.google.android.calendar'`

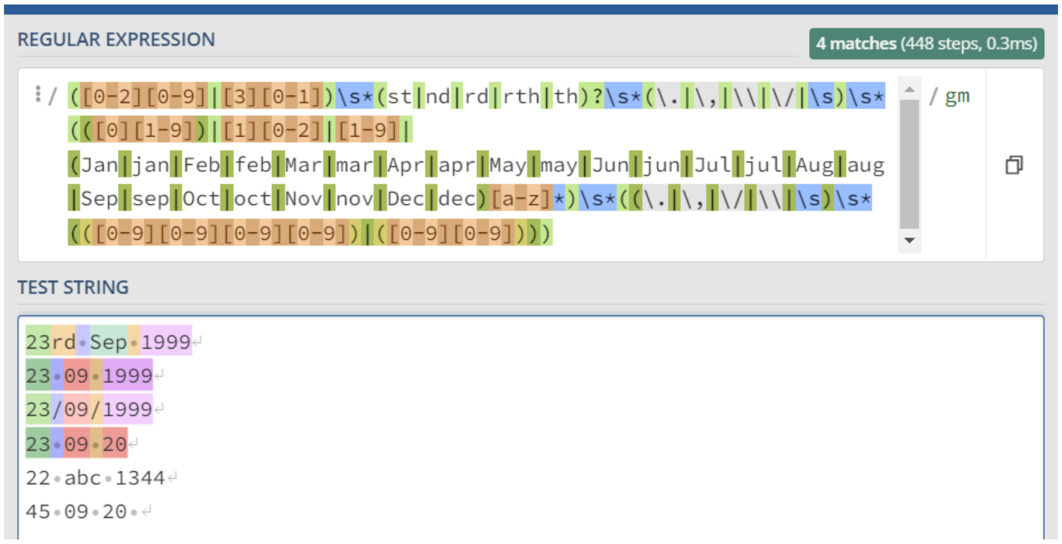
- **Alarm Setup:**
  - **Ideology and Motivation:** Alarm Setup is another routing feature of the app that allows the user to easily navigate in order to setup alarm as per their convenience.
  - **Implementation:** To implement alarm setup, similar to calendar event setup, we are using external_app_launcher[8] package to redirect user to system app

of clock as it is easier to config alarm from their for user. We have added bottom navigation bar tab in mail page where user can click on alarm tab to create alarm in system clock application. If user want to set alarm after reading mail, user will click on tab and clock application will open, once alarm is set, on clicking back button of android, user will come back to SMail application to point where user left off i.e mail page. Appstore link for clock is 'https:play.google.com/store/apps/details?/
id=com.google.android.deskclock&hl=en_IN&gl=US'

- **Date Detection:**

  - **Ideology and Motivation:** The idea behind date detection is to find out dates in the mail which has deadline as a subset. This feature helps us to target the dates. This is helpful in itself as a tag and is necessary highlights the dates of important events, deadlines, and exams etc.

  - **Implementation:** To implement date detection, we will use regular expression to detect various date pattern. This regular expression will be passed to subject and body text of email before text preprocessor in TextParser method which will have boolean flag which will be true if regular expression finds desired pattern else false. In case it is true, then we will append "has date" tag with email which have highest priority i.e 11.0. We have added red color for has date tag so that user can easily understand that particular email have date. Following is the regular expression that we have used to detect date pattern:
    ```
    r'([0-2][0-9]|[3][0-1])\s*(st|nd|rd|rth|th)?\s*(\.|\,|\\|\s)\s*
    (([0][1-9])|[1][0-2]|[1-9]|(Jan|jan|Feb|feb|Mar|mar|Apr|apr|May|
    may|Jun|jun|Jul|jul|Aug|aug|Sep|sep|Oct|oct|Nov|nov|Dec|dec)[a-z]*)
    \s*((\.|\,|\\|\s)\s*(([0-9][0-9][0-9][0-9])|([0-9][0-9])))'
    ```
    In figure 2.4, we can see that, our regular expression is correctly detecting various



```
/ ([0-2][0-9]|[3][0-1])\s*(st|nd|rd|rth|th)?\s*(\.|\,|\\|\/|\s)\s*
((([0][1-9])|[1][0-2]|[1-9]|
(Jan|jan|Feb|feb|Mar|mar|Apr|apr|May|may|Jun|jun|Jul|jul|Aug|aug
|Sep|sep|Oct|oct|Nov|nov|Dec|dec)[a-z]*)\s*((\.|\,|\/|\\|\s)\s*
(([0-9][0-9][0-9][0-9])|([0-9][0-9]))) / gm
```

```
23rd Sep 1999
23 09 1999
23/09/1999
23 09 20
22 abc 1344
45 09 20 
```

Fig. 2.4 Regular Expression to detect dates.

types of formats of date by highlighting them in test string. We successfully detected dates with separator, ordinal, alphabetical month, short year and long year. In figure 2.4 it says, that 4 matches it found in test string i.e first four date patterns. Then we also gave invalid date format such as wrong month, or date more than 31 and pattern does not detect it and hence our regular expression is working properly. We have used regex101.com[5] website to test our regular expressions.

Let us understand meaning of this regular expression.

1. `([0-2][0-9]|[3][0-1])`: In it we are trying to cover all days i.e from 01 to 31.

2. `\s*(st|nd|rd|rth|th)?\s*`: In it `\s*` means that if there are multiple spaces, also considered them in pattern. It will be multiple places in regular expression. Then we are considering ordinals i.e st, nd, rd, and th in 1st, 2nd, 3rd, 4rth, 5th, etc.

3. `(\.|\,|\\|\|\s)`: These are for separator between day, month and year. i.e dd/mm/yyyy or dd-mm-yyyy or dd mm yyyy or dd,mm,yyyy, etc.

4. `([0][1-9])|[1][0-2]|[1-9]`: This is for month in mm format. i.e 1-12 or 01-12.

5. `(Jan|jan|Feb|feb|Mar|mar|Apr|apr|May|may|Jun|jun|Jul| jul| Aug|aug|Sep|sep|Oct|oct|Nov|nov|Dec|dec)[a-z]*)`: This consider all type of alphabetical month representation such as dec, Dec, december and December. `[a-z]*` will take care of long form of months such as ember in December.

6. `(([0-9][0-9][0-9][0-9])|([0-9][0-9]))`: This will take care of year in yy or yyyy format.

This way it will be easier to detect dates in efficient way as regex function in dart language uses turing machine DFA to traverse states mentioned in regular expression.

Chapter 3

Phase I Algorithm

In this chapter, we will dive into the algorithm constructed in phase I. To note that the algorithm of phase I was constructed mainly in 2 steps. We shall give a brief overview of the final Phase I algorithm.

The first algorithm proposed in phase I had 3 components namely, TextParser, GetPriority and SetTag. 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.

Later on with the help of data structures the algorithm was refined further. Tries were used to parse the content of the email where each word was inserted in a trie tree which in turn makes it faster to search a specific Keyword. We also used unordered maps in order to make search, insert and traversing a bit faster.

- **TextParser(DataTree):**

  As the name suggests, the job of TextParser is simple i.e. to parse the text. It requires a DataTree(a trie tree) as input. Then for tags in the taglist according to high to low priority its checks whether the email has a certain set of keywords. If the email does indeed cross the threshold to belong to that tag, it was then assigned the Priority Number.

- **GetPriority(Emails):** GetPriority method takes care of organizing and assigning the priority numbers of all emails. It takes an array of emails and for each email create 2 data trees, one for the subject and other for the body. Then it calls the TextParser on both of these trees separately and assigns the Priority number of subject and body to the respective email ID.

Algorithm 2 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:** As one of the main objective of our project is to provide user friendly interface where it can change or set its own tags. Also for us to have a predefined set of tags. To kill these 2 birds with one stone, we created the SetTag method to set a tag, it takes in the Tagname, an array of keywords, Tag's desired priority and colour associated with it to create a tag for us which is appended into the Taglist.

3.1 Conclusion

In this chapter, we gave a recap of the algorithm of Phase I which essentially had 3 parts:-

- **TextParser** which currently has a time complexity of $O(ntk)$ where n = length of word , t = number of tags, k = number of keywords.
- **GetPriority** which also has a time complexity of $O(ntk)$.
- **SetTag** which have the time complexity of $O(t)$

The algorithm is further optimized in the next chapter with due consideration so as to not increase its computation otherwise it may make the functioning a bit laggy. The optimization bring out the best in the algorithm where complexity sure does increase only to make the priority handler better.

Chapter 4

Phase II Algorithm

In previous chapter, we have seen proposed priority algorithm for our smart email client for phase 1 which have time complexity bottled by the GetPriority algortihm and TextParser algorithm by using trie and unordered map i.e. of the $O(n^4)$ to $O(n^3)$. This algorithm only supported one tag per email. In this chapter, we will propose priority algorithm of same maintained time complexity i.e of $O(n^3)$ with enhanced feature optimally and also bring fairness. We have supported multiple tags per email, also used Least frequently used cache method to find top K most frequent word in $O(nlog(K))$ where n is number of words in text and K is Constant and hence in $O(n)$. These will help to enhance Tag's keywords. We also used early stoppage when conditions are met rather than completing entire loop to save several computational power.

4.1 Construction

In this section we will see, how our new proposed modification over previous algorithm is supporting multiple tags and also bringing fairness in finding appropriate tags and keywords for them in same time complexity.

Our first modification is, we have keeping track of frequency of word in particular email and also keeping track of top K most frequent word. We are achieving this by using unordered map as finding frequency of word in hashmap takes $O(1)$ which will keep track of freqecy of all words in email body as well as in subject and by using least frequent used cache method to find top K most frequent words.

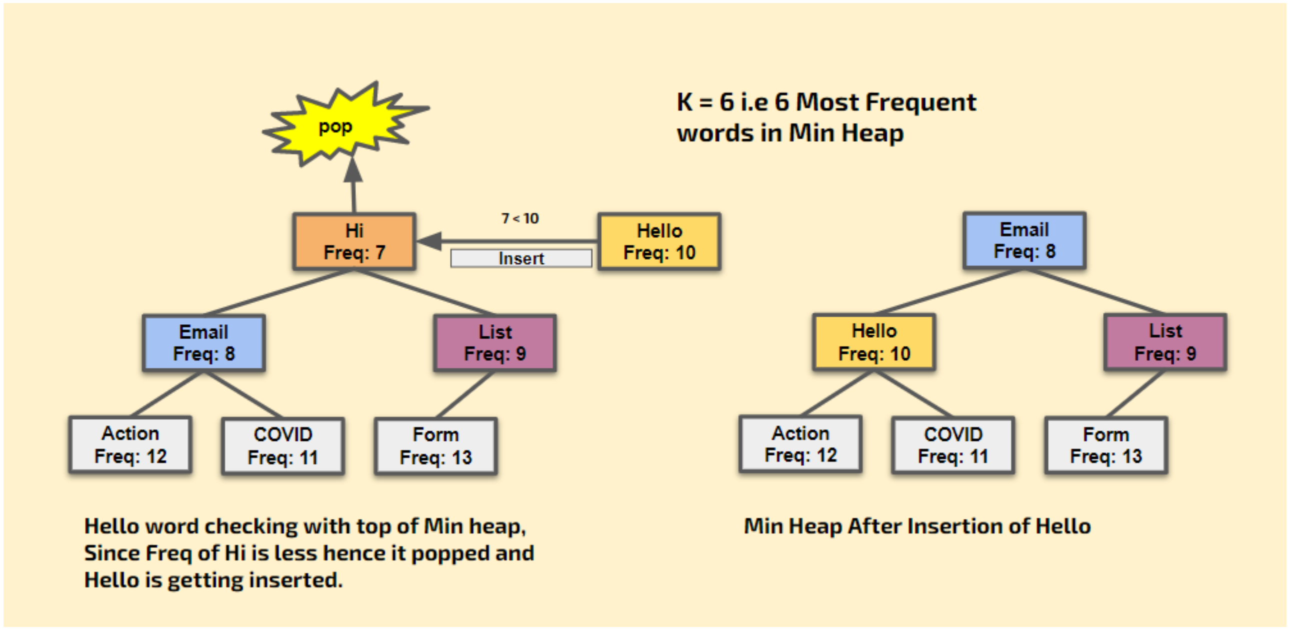

pop
K = 6 i.e 6 Most Frequent words in Min Heap
Hi Freq: 7
7 < 10
Insert
Hello Freq: 10
Email Freq: 8
List Freq: 9
Action Freq: 12
COVID Freq: 11
Form Freq: 13
Hello word checking with top of Min heap, Since Freq of Hi is less hence it popped and Hello is getting inserted.
Email Freq: 8
Hello Freq: 10
List Freq: 9
Action Freq: 12
COVID Freq: 11
Form Freq: 13
Min Heap After Insertion of Hello

Fig. 4.1 LFU MinHeap Demonstration for K Most frequent words.

Let us understand how we are using LFU cache replacement policy method [9] to find top K most frequent, as in fig 4.1 shown, we are using min-heap to keep track of top K most frequent words at any instance where top node will always have word with least frequency in K most frequent words, let say new word hello is coming then we will check frequency of hello word in frequency hashmap and will check whether frequency of top of minheap is less than frequncy of new word i.e hello, in this case it is smaller and hence we will pop top element and insert hello node with frequency value in minheap. Insertion and deletion of minheap both takes $O(log(n))$ but in our case n is K i.e constant hence maintenance of K most frequent word will be done in constant time for each new word arrival and hence to check for all unique words of email we will require $O(n)$ time where n is number of words.

Now question arises, why we need K most frequent words, answer is to modify list of keywords for tag. As shown is fig 4.2, let us consider that our K is 3 and we got three most

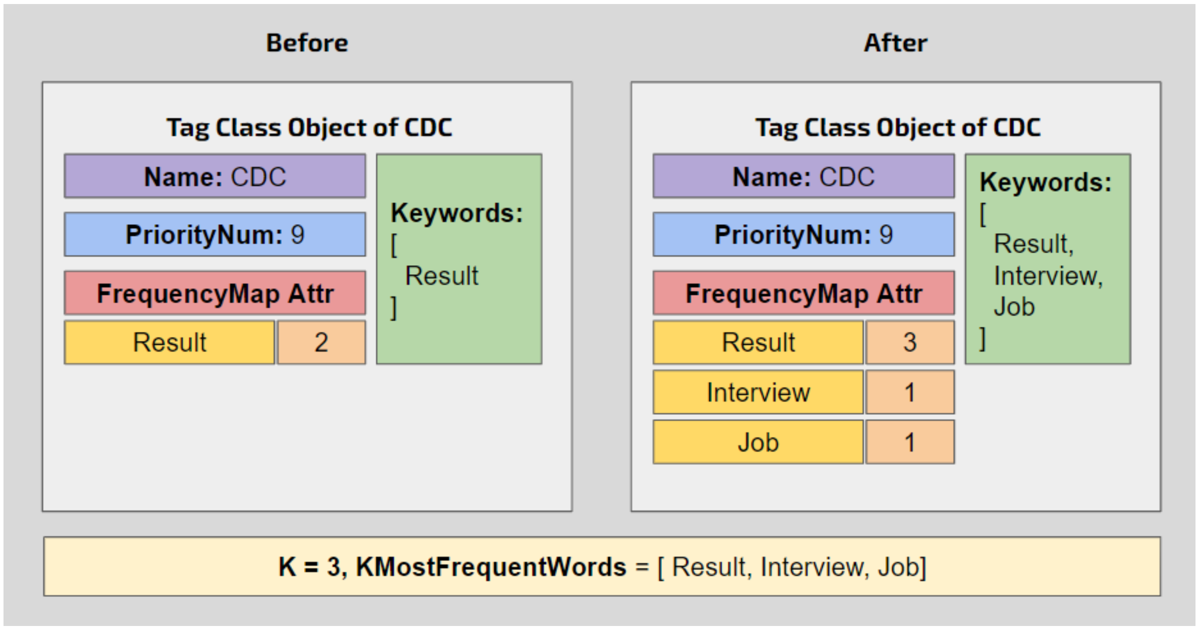


Fig. 4.2 Modification of Tag object using K Most Frequent words.

frequent words from email's words from both subject and body, now one word i.e result is already in our tag CDC's keyword list but words interview and job are not in it. So we will insert these two as well in keyword list, but if we consider this expanded list of keywords to find optimal tags, every time computation is increasing and hence we have to restrict it.

Restriction is done by frequencyMap attribute of Tag class which have described in coming section. This attribute keep track of number of time each keyword in modified list in tag has appeared in K most frequent words of every email, if it is in those words then we will increament frequency of that word by 1 and if new word is inserted in keyword list then it will be initialized as 1. So in above fig, result was there for 2 time before and hence after evaluation of current mail, it's frequency went to 3 and frequency of job and interview initialized to 1. Now for upcoming email, will choose top K' most frequent keywords from keyword list according to hashmap, and those will participate in evaluation of tags. This way for each email, for each tag we will only consider top K' most frequent keywords from list and hence time complexity will not be affected and remain same for every email's evaluation.

To find multiple tags for each emails, Instead of returning one tag when threshold satisfied, we will keep finding tags that are satisfying threshold and store them in list and when there are tags of desired value, let's say two i.e max two tags per email will be calculated and function will return that list. Since in previous chapter's algorithm, we anyway traversing through each tag for evaluation, time complexity wont be affected, just we will require additional list to keep track of those tags. We will also break loops when following conditions are met:

- If keyword threshold satisfied then we will break keyword loop in TextParser algorithm.
- If tag counter i.e number of tags suited for emails reach limit max allowed tags per email then return list of tags.
- If keyword count reach limit i.e top K element from keyword list of tag undergoes evaluation then break keyword loop.

4.2 Improved Method

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 according to explanation given in section Construction.The SetTag method will remain unchanged.

Algorithm 3 Class Tag

```
1: string name,
2: string[] keywords,
3: float PriorityNum,
4: Unordered_map frequencyMap,
5: float Threshold,
6: Tag(name) {
7:         self.name = name
8:         self.keywords = []
9:         self.frequencyMap = new Unordered_map(string,int)
10:        self.Threshold = 0.2
11:        self.PriorityNum = 0
   }
```

Above is a Tag class, which has been changed due to various requirement by other methods. This class will have following attributes:

- **name**: This will be name of tag given by default or by user
- **keywords**: This will be list of words that will be used to identify appropriate tags for email
- **frequencyMap**: This will be hashtable i.e unordered map where key will be keywords and value will be number of times they are appeared in unique emails.
- **PriorityNum**: This will be float value in range 0 to 10, depending on importance of tag where 10 is highest importance and 0 is least importance.
- **Threshold**: This will be float value from 0 to 1.0, which will decide minimum fraction of keywords should be present in email text to assign tag to that email.

Constructor of Tag class will take name of tag given as parameter and will assign it to name attribute, priorityNum attribute will be initialized as 0 and Threshold attribute as 0.2 and both keywords and frequencyMap will be initialized as empty.

Algorithm 4 TextPreProcessor()

```
 1: hasDatePattern = false
 2: if regex.have(DateRegex, Email.Subject.words && Email.body.words) then
 3:    hasDatePattern = true;
 4:    TopKTags.add("has date")
 5: end if
 6: filterRegex = r'[^\w\s]|[0-9]+|[ ]{2,}'
 7: Email.Subject.words.replaceAll(filterRegex, ")
 8: Email.body.words.replaceAll(filterRegex, ")
 9: TreeNode StopWordTree = TreeNode();
10: for word in stopWords do
11:    StopWordTree.add(word)
12: end for
13: for word in Email.Subject.words do
14:    if StopWordTree.contains(word) then
15:       Email.Subject.words.drop(word)
16:    end if
17: end for
18: for word in Email.body.words do
19:    if StopWordTree.contains(word) then
20:       Email.body.words.drop(word)
21:    end if
22: end for
```

Above TextPreProcessor will be in between GetPriority method. line 2-5: We are checking whether subject or text of email have date pattern using regular expression that we had discussed in chapter 2. Then we are removing all unwanted numbers, punctuation marks and extra spaces using filter regex which also we had discussed in chapter 2 for both subject and body from line 6 to line 8.

From line 9-12: we are creating trie tree with trie node which is storing stopwords that we had discussed in chapter 2. At line 13-17 and 18-22, we are checking each word of preprocessed subject and body, whether it have stopword using trie tree. Since finding pattern using regex takes $O(m * n)$ where m is length of regex expression and n is number of words, and since m is constant as we have constant amount of stopwords, overall time complexity for both regex pattern i.e have and replaceAll method will take $O(n)$ time complexity. Creation of stopWordTree and adding words in it will be of $O(a)$ where a is length of stop word. Searching in stopWordTree will take $O(a * n)$ where a is length of word i.e subject and body words and n is number of words. Hence overall time complexity of Text Preprocessor is $O(n^2)$.

Algorithm 5 TextParser(DataTree)

```
1: TagCounter = 0
2: TagMatched = []
3: for Tag in taglist do
4:    presentCount = 0
5:    keywordCount = 0
6:    for Keyword in Tag.keywords do
7:       if DataTree.contains(keyword) then
8:          presentCount++
9:       end if
10:      keywordCount++
         // break loop if Threshold satisfied
11:      if presentCount $\geq$ Tag.Threshold*(Size(DataTree.keywords) then
12:         TagMatched.add(Tag)
13:         TagCounter++
14:         if TagCounter $\geq$ MAXTAGSALLOWED then
15:            return TagMatched
16:         end if
17:         break Keyword loop
18:      end if
         // if top K Keywords are tested then break keyword loop
19:      if KeywordCount $\geq$ topKKeyword then
20:         break keyword loop
21:      end if
22:   end for
23: end for
```

In above algorithm 5, We have following variables:

- **TagCounter**: This counts number of tags satisfied for email i.e fraction of its keywords crossed threshold of tag.
- **TagMatched**: This is list which stores tags satisfied for email
- **presentCount**: This counter tracks number of keywords of tag are in dataTree
- **keywordCount**: This counter tracks number of keywords of tag undergo evaluation

We traversing tags and in tags we are traversing every keyword of Tag.keywords list and in it we are checking whether that keyword present in DataTree of trie datastructure, if present we are incrementing presentCount and for each keyword we are incrementing keywordCount, then we are checking conditions as mentioned above in Construction section, i.e if presentCount passed threshold then we will add that tag to TagMatched list and also increment TagCounter, we also check whether TagCounter crossed MAXTAGSALLOWED threshold which is global constant if crossed then return TagMatched list. We also checked whether KeywordCount crossed topKKeyword constant, which represent top K most frequent keywords from keyword list of that tag using frequencyMap attribute.

At line 3, we are traversing all tags lets say that loop run at max t times, then at line 6 we are running keywords loop at max k times and checking whether keyword contains in $O(n)$ where n is length of word. Condition checking and list insertion takes constant time hence TextParser will have time complexity of $O(ntk)$.

Algorithm 6 ModifyTags(KMostFrequent[], tags[])

```
1: for tag in tags do
2:     for word in KMostFrequent do
3:         tag.keywords.add(word)
           // keyword present in email counter
4:         tag.frequencyMap[word]++
5:     end for
6:     tag.keywords.sort(tag.frequencyMap)
7: end for
```

This algorithm adds KMostFrequent words we have obtained from LFU method to tag's keywords list and increment value in frequencyMap with key of word. Atlast we sort keywords list according to frquencyMap's value. Adding of word in list and accessing value in hashmap in constant time. We are doing it for all tags i.e t times and for K times internally but since K is fixed and constant overall insertion of word in list and managing hashmap take $O(t)$. Sorting of list takes $O(log(k))$ where K is number of keywords in list. Hence overall time complexity of ModifyTags is $O(t.log(k))$.

Algorithm 7 LFU(PQ, frequency, word)

```
1: if PQ.size < MAXMOSTFREQUENTWORD then
2:     PQ.add(word, frequency)
3: else
4:     if PQ.top.frequency < frequency then
5:         PQ.pop
6:         PQ.add(word, frequency)
7:     end if
8: end if
9: return PQ
```

In LFU method, we are passing PQ parameter i.e priority queue having min heap internally and managing K most frequent words, we are also passing word and frequency as parameter then we are comparing top of heap with frequency and if it is less than frequency of word we are popping it and then adding word with frequency, both of this operation takes $O(log(K))$ time complexity but since K is constant entire method takes constant time i.e $O(1)$.

Algorithm 8 GetPriority(EmailData)

```
 1: TreeNode DataTree = TreeNode();
 2: Unordered_map frequency = Unordered_map();
 3: KMostFrequent = []
 4: TextPreProcessor();
 5: PQ = PriorityQueue();
 6: for word in Email.Subject.words do
 7:     DataTree.add(word)
 8:     frequency[word]++
 9: end for
10: SubjectPNum[] = TextParser(DataTree)
11: DataTree.clear
12: for word in Email.body.words do
13:     DataTree.add(word)
14:     frequency(word)++
15: end for
16: for word in frequency do
17:     PQ = LFU(PQ, frequency[word], word)
18: end for
19: BodyPNum[] = TextParser(DataTree)
20: while PQ.size > 0 do
21:     KMostFrequency.add(PQ.pop(), word)
22: end while
23: PriorityNums[], TopKTags[] = topKTags(SubjectPNum, BodyPNum)
24: EmailMap.add(Email.ID, PriorityNums)
25: EmailTag.add(Email.ID, TopKTags)
26: ModifyTags(KMostFrequent, TopKTags)
27: return True
```

In GetPriority method, we takes EmailData as parameter that have subject, body of emails with other information. At line 1, we are initializing TreeNode in constant time. At line 2, we are initializing frequency hashmap in constant time that stores frequency of each word in emailData. We also initialize KMostFrequent list and PQ priority queue by default minheap, both to find K most frequent word using LFU method in constant time. TextPreProcessor method discussed earlier takes $O(n^2)$ complexity. Then we traverse words in subject and word in body in line 5 to 9 and line 12 to 16 respectively, where we are adding word to DataTree which takes $O(n)$ where n is length of word, we incrementing frequecy of word in constant time hence for each loop we are taking $O(n)$ time. Let say for subject we are running loop for a times and for body, b times hence both loop will take $O(a.n)$ and $O(b.n)$ time respectively. Once DataTree is prepared at the end of loop we are passing it to TextParser method which return list of at max K tags in $O(ntk)$ time, where t is number of tags and k is number of keywords in it. At line 11, we are clearing DataTree to be used by body loop thats takes constant time. We also calculating most frequent words by iterating over all words in frequency map and passing them to LFU, which takes constant time hence overall time complexity will be $O((a + b).n)$.

Once, tag list is received by calling TextParser method we are storing it in SubjectPNum and BodyPNum list respectively. At lines 18 to 20, we are fetching all words from priority queue to KMostFrequent list by popping nodes that takes $O(K')$ where K' is constant and hence in constant time for K' time and hence overall popping of PQ happens in constant time. Then we will find top K tags among SubjectPNum and BodyPNum by calling topKNums which will use merge technique that get used in mergesort to create list of topKTags and PriorityNums of these topKTags in ordered manner. Merge tachnique of mergesort takes $O(K)$ time where K is max tags allowed i.e length of topKtags list which is constant and hence time is also constant. Then we will add those list to respective lists i.e EmailMap and EmailTag in constant time and will call ModifyTags function that takes $O(t.log(k))$ where t is number of tags and k is number of keywords.

Hence overall time complexity of GetPriority method is $O(max(an, bn, O(n^2), (a + b)n, ntk, tlog(k)))$ i.e $O(ntk)$ which is equivalent to $O(n^3)$. Hence, overall time complexity of algorithm remains unharmed, with multiple optimizations.

4.3 Conclusion

We have seen construction and algorithms above. We have managed to calculate priority in $O(ntk)$ which is equivalent to $O(n^3)$ time complexity i.e unchanged from prior algorithm. Because of multiple optimizations we discussed and shown above, every word gets fair chance to be participate in tag assignment as we always keeping track of K most frequent words from email and adding them to keywords list where also top K' most frequent words will participate in process. And since we are supporting multiple tags, not only one tag but other tags also have fair chance to be part of tag assignment without compromising computational power. We further even reduced burden on TextParser function by introducing TextPreProcessor in GetPriority, which removes unwanted words and characters without compromising overall time complexity.

Chapter 5

Possible Future Modifications for Production

Even though we have a full fledged working application, our vision doesn't stop just yet. We want our app to be successfully deployed so that people can freely use it for their convenience. This chapter talks about possible modifications that will make the app more practical. In terms of production, the application is currently in its β version. So without further Adieu, let's jump into these modification.

- **Deployment:** One of main step in production of any application is to make it available for all. This modification refers to the deployment of the application on playstore and appstore. This will ensure the application availability to all, also ensures that the app is compatible with devices through rigorous testing by playstore. Only limitation to this is that it costs a money to do so.

- **Threading:** As of now the app support only a single account. This is so because our app was designed for IIT Pkd keeping the smail accounts in mind. But it's always better to provide flexibility and we know the importance of it. Hence, we wanted the app to support multiple accounts so that user can easily access all their accounts. This can be achieved by adding threading in our application.

- **Cloud Integration:** To reiterate currently the application supports single account on a single device which works but is not entirely practical in today's world. Hence we want the app to be cloud integrated to store algorithm on cloud instance and run it on cloud so that multiple instance of single user can be accessed from multiple device. This will not only solve the problem of multiple devices but will come in handy as the algorithm is run on cloud making it easier for user's device hardware.

- **Web Application:** Many applications today are also run on browsers as it allows the user to access the features from anywhere. We can use the cloud API discussed above to support on browser. The only thing it requires is a front end web application for the same.

Last but not the least our application was designed for IIT Pkd but in reality these features and algorithm is flexible and can be used in multiple organizations. The only thing needed is to know the structure of the organization for creation of tags and their respective keywords. This can be achieved by creating a simple UI(User Interface) where the management of the organization can customize the application for their needs.

Not to forget that along with future modifications, updating the app for unknown bugs or certain discrepancies will also be a part of possible future modifications. This can be achieved via testing and user feedback.

Chapter 6

Results

The progress made in phase II has already been discussed in the above chapters but in order to dive into more depth of the works done, we have included certain images in this chapter. This way it will be easier to judge the output of the work input in the project. So let's jump right into it.

6.1 Sneak Peak at Smail's Interface

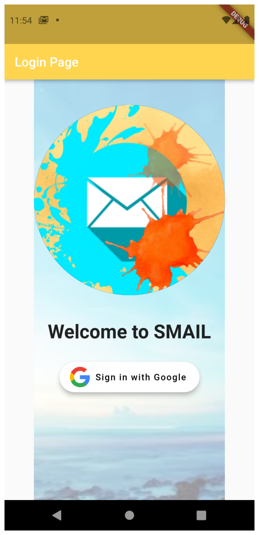
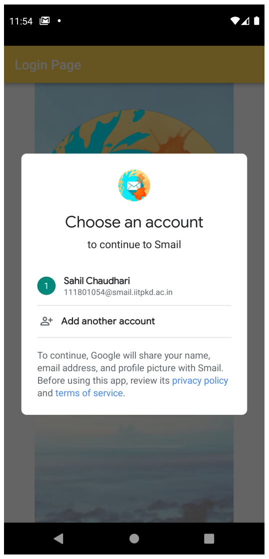


Fig. 6.1 User's Main Page



Choose an account

to continue to Smail

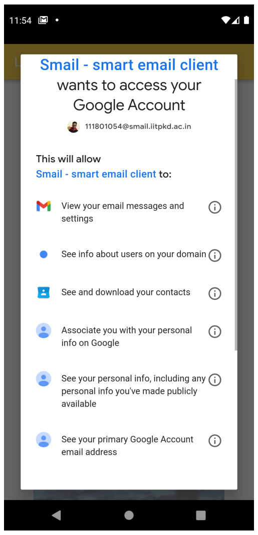
Sahil Chaudhari
111801054@smail.iitpkd.ac.in

Add another account

To continue, Google will share your name, email address, and profile picture with Smail. Before using this app, review its privacy policy and terms of service.

Fig. 6.2 Login Page

Figure 6.1 shows the User's Main Page, The User first is greeted by the main page where our custom designed logo can be seen as welcoming the user with an option to sign in to the respective email id. On the other side, Figure 6.2 shows the Login Page where the user can toggle, choose or add the email account which they wanna use for logging into the app. This interface is somewhat similar to one offered by various google apps, hence is convenient for the user as it might be familiar.


11:54

Smail - smart email client
wants to access your
Google Account
111801054@smail.iitpkd.ac.in

This will allow
Smail - smart email client to:

View your email messages and settings

See info about users on your domain

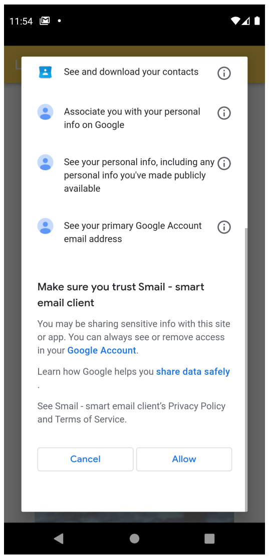
See and download your contacts

Associate you with your personal info on Google

See your personal info, including any personal info you've made publicly available

See your primary Google Account email address

Fig. 6.3 Permission 1


11:54

See and download your contacts

Associate you with your personal info on Google

See your personal info, including any personal info you've made publicly available

See your primary Google Account email address

Make sure you trust Smail - smart email client

You may be sharing sensitive info with this site or app. You can always see or remove access in your Google Account.

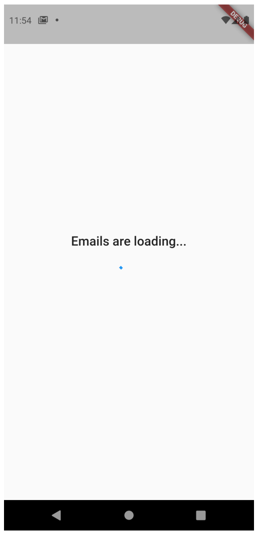
Learn how Google helps you share data safely
.

See Smail - smart email client's Privacy Policy and Terms of Service.

Fig. 6.4 Permission 2

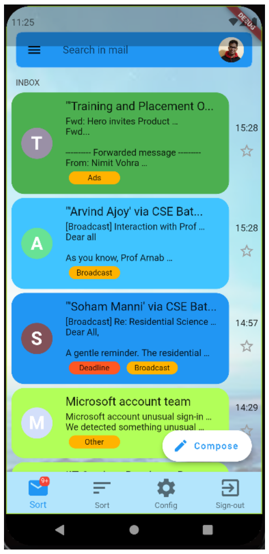
Figure 6.3 and Figure 6.4, shows the permission page, it nothing but the necessary permission that one gives while working with an app. Basically this is what allows us to view mails from gmail or use the other privileges that are associated with google account.

After we go through the permissions, we are all set to fetch the mails, Figure 6.5 shows us the loading page where in the background the app is fetching the mails for the user. Once done we can see the Final Front Page (in Figure 6.6), where we can see the colour coded mails with multiple tags associated with them. Moreover, at the bottom, 4 toggle options can be seen for sort, config and signout which will be covered shortly.



Emails are loading...

Fig. 6.5 Loading Page



"Training and Placement O...
Fwd: Hero invites Product ...
Fwd...

--------- Forwarded message ---------
From: Nimit Vohra ...
Ads
15:28

"Arvind Ajoy' via CSE Bat...
[Broadcast] Interaction with Prof ...
Dear all

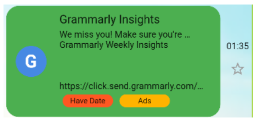
As you know, Prof Arnab ...
Broadcast
15:28

"Soham Manni' via CSE Bat...
[Broadcast] Re: Residential Science ...
Dear All,

A gentle reminder. The residential ...
Deadline Broadcast
14:57

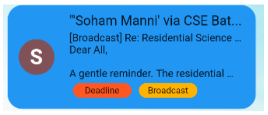
Microsoft account team
Microsoft account unusual sign-in ...
We detected something unusual ...
Other
14:29

Fig. 6.6 Front Page


Grammarly Insights
We miss you! Make sure you're ...
Grammarly Weekly Insights

https://click.send.grammarly.com/...
Have Date Ads
01:35

Fig. 6.7 Date Detection


"Soham Manni' via CSE Bat...
[Broadcast] Re: Residential Science ...
Dear All,

A gentle reminder. The residential ...
Deadline Broadcast

Fig. 6.8 Deadline

Figure 6.7 shows the date detection feature which using the regular expression detects the date as explained earlier and display it's tag in red. Same goes with deadline detection in Figure 6.8, where the red deadline tag can be seen. Note: These tags have been specially assigned different colours from other tags(amber) to highlight there importance.

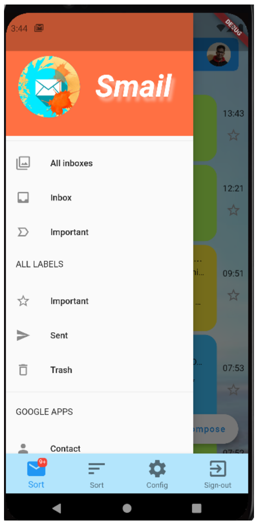


Fig. 6.9 Drawer

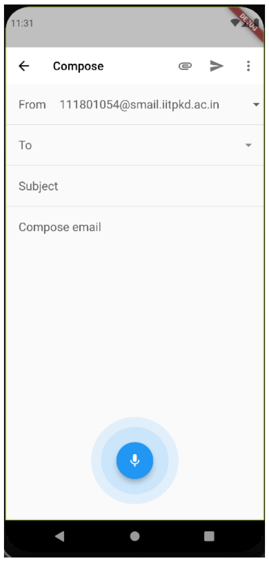
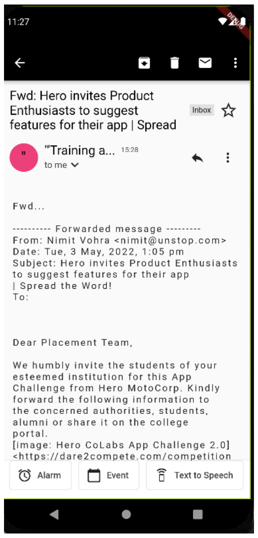


Fig. 6.10 Speech-to-text

Figure 6.9 shows the drawer of the app with its various option. Figure 6.10 shows the compose page where the blue mic at the bottom is there for the speech to text feature i.e. to be able to speak the mail body.



Fwd: Hero invites Product Enthusiasts to suggest features for their app | Spread

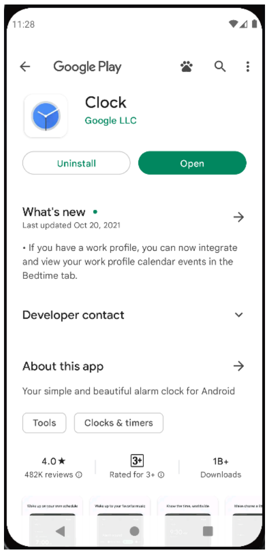
```
Fwd...

---------- Forwarded message ---------
From: Nimit Vohra <nimit@unstop.com>
Date: Tue, 3 May, 2022, 1:05 pm
Subject: Hero invites Product Enthusiasts
to suggest features for their app
| Spread the Word!
To:


Dear Placement Team,

We humbly invite the students of your
esteemed institution for this App
Challenge from Hero MotoCorp. Kindly
forward the following information to
the concerned authorities, students,
alumni or share it on the college
portal.
[image: Hero CoLabs App Challenge 2.0]
<https://dare2compete.com/competition
```

Fig. 6.11 mailpage

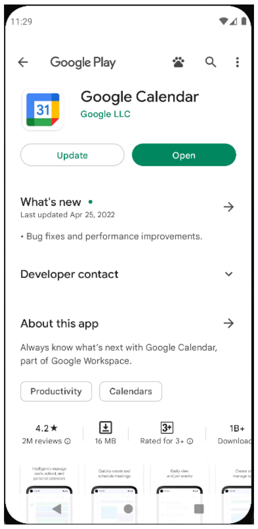


• If you have a work profile, you can now integrate and view your work profile calendar events in the Bedtime tab.

Your simple and beautiful alarm clock for Android

Fig. 6.12 clock

The mail Page i.e. where a specific mail can be seen is shown in Figure 6.11, if looked closely, it can see at the bottom 3 buttons namely, Alarm which when clicked will route the user to the clock app (Figure 6.12) in order to set up any alarms, next is Event which on clicking will redirect the user to the Google Calendar (Figure 6.13) to setup any reminders or events. Lastly, the text-to-speech feature of our app. The button on clicking will open up a simple page which was designed keeping user convenience in mind.


11:29

Google Play

Google Calendar
Google LLC

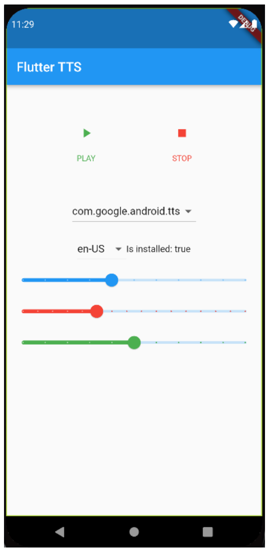
What's new
Last updated Apr 25, 2022
• Bug fixes and performance improvements.

Developer contact

About this app
Always know what's next with Google Calendar, part of Google Workspace.

4.2★ 2M reviews | 16 MB | Rated for 3+ | 1B+ Download

Fig. 6.13 calendar


11:29

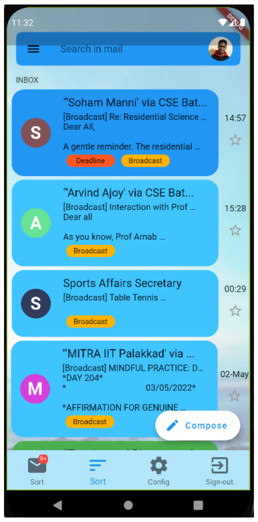
Flutter TTS

com.google.android.tts

en-US Is installed: true

Fig. 6.14 text-to-speech

It can be clearly seen in Figure 6.14, which has a simple play and stop button, also has an option to choose the voice engine and language as per user requirements. Also, the three shown slider when hovered over shows their respective functions i.e. pitch(blue), volume(red), speed(green).



"Soham Manni' via CSE Bat...
[Broadcast] Re: Residential Science ...
Dear All,

A gentle reminder. The residential ...
Deadline Broadcast
14:57

"Arvind Ajoy' via CSE Bat...
[Broadcast] Interaction with Prof ...
Dear all

As you know, Prof Arnab ...
Broadcast
15:28

Sports Affairs Secretary
[Broadcast] Table Tennis ...
Broadcast
00:29

"MITRA IIT Palakkad' via ...
[Broadcast] MINDFUL PRACTICE: D...
DAY 204
* 03/05/2022*

*AFFIRMATION FOR GENUINE ...
Broadcast
02-May

Fig. 6.15 Sort

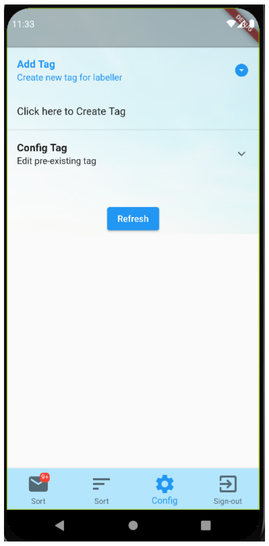


Fig. 6.16 AddTag

In order to save some time to the user as well as to make sure that important mails are not missed, the app provides a sorting feature which will show important mails on top, can be seen in Figure 6.15.

The app also provides the user with the option to configure/add/remove the tags as per their own needs. The configuration page is made for the same. In figure 6.16 we can see the option of addition of a tag, which on clicking will look like a form (Figure 6.17). These fields can be filled in order to create a new tag. Once done a Confirmation message also appears(Figure 6.18).

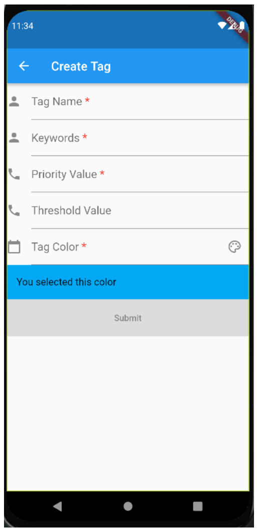
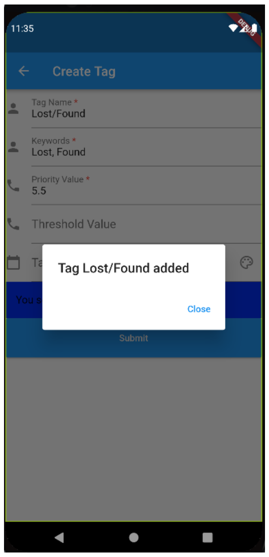


Fig. 6.17 AddTag Form



Tag Name *
Lost/Found

Keywords *
Lost, Found

Priority Value *
5.5

Tag Lost/Found added

Fig. 6.18 AddTag Confirmation

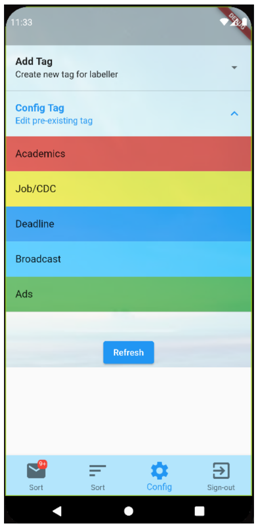
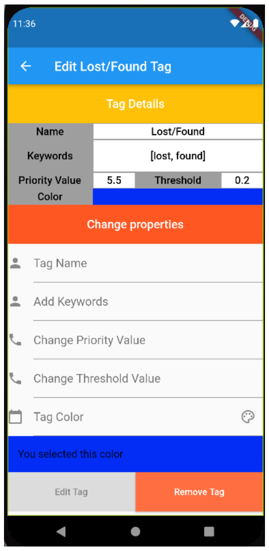


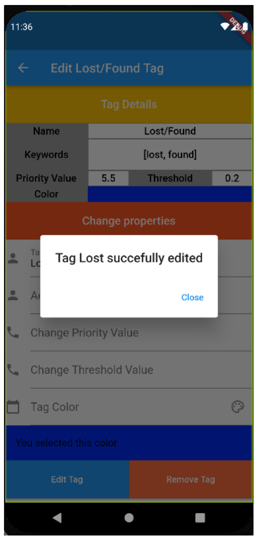
Fig. 6.19 Config Tag



Name	Lost/Found		
Keywords	[lost, found]		
Priority Value	5.5	Threshold	0.2
Color			

Fig. 6.20 Edit Tag

For the configuration of tag we can select the respective tag (Figure 6.19) and then can fill a similar form structure to edit the tag(Figure 6.20). Once done we get either a confirmation for editing(Figure 6.21) or removing(6.22) the tag.


11:36

Edit Lost/Found Tag

Tag Details

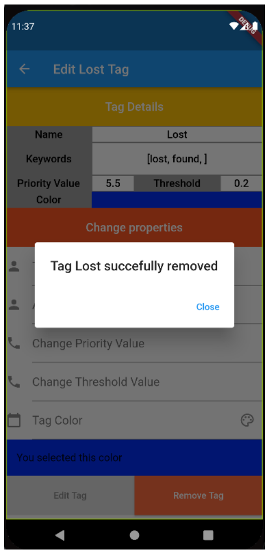
Name	Lost/Found		
Keywords	[lost, found]		
Priority Value	5.5	Threshold	0.2
Color			

Change properties

Tag Lost succefully edited

Close

Fig. 6.21 Edit Confirmation


11:37

Edit Lost Tag

Tag Details

Name	Lost		
Keywords	[lost, found,]		
Priority Value	5.5	Threshold	0.2
Color			

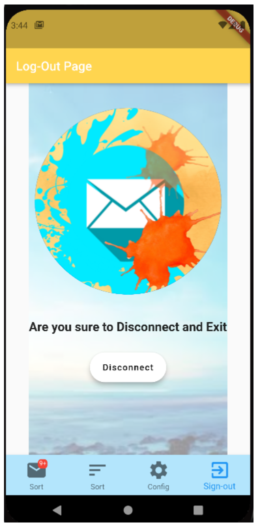
Change properties

Tag Lost succefully removed

Close

Fig. 6.22 Remove Confirmation

The signout page, shown in Figure 6.23, has also been changed which now has a disconnect button which will sign the user out and exit the app. FYI, Figure 6.24 shows the Smail App which will be view once we exit the app.


3:44

Log-Out Page

Are you sure to Disconnect and Exit

Fig. 6.23 SignOut

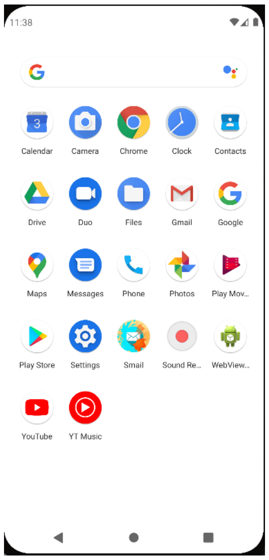


Fig. 6.24 Smail App

Chapter 7

Conclusion

Along with the final phase came the responsibility to create a proper working app. In phase II part 2 i.e. after the Midsem, we had several challenges in front of us, which were - to integrate the finalized priority algorithm, create and integrate the text pre-processor, creating the configuration page, as well as creating the promised features in the constrained time. During the journey to overcome these hurdles, we faced some more challenges such as testing and resolving bugs. Through the joint effort of both the team members, we were able to progress and achieve our goal/vision(The Smail App). The uniqueness of our app can be defined by the features (suggested by our mentor) namely - the sorting feature(to display the important mails on top), the speech-to-text and text-to-speech feature(For User convenience), lastly the date detection(to avoid missing out on important dates). This was the end of a great journey, one that we will never forget for the rest of our life. The Smail App now belongs not only to us but to the whole IIT Pkd.

Needless to say, our vision doesn't stop here, we have suggested future modifications(in Chapter 5) that can make the app more amazing than it already is. So If the opportunity shall arise in the future, we will surely progress more. The budding app is now bloomed into a properly working application and will always be on its way to improving itself and being more than its previous versions.

7.1 Contribution

Smail App is a part of both the teammates, our work, time, and effort was dedicated to building it. So it is safe to say, it was a team effort. To get down to the specifics of the Phase II part 2(after MidSem), Sahil handled the integration of the priority algorithm and text pre-processor along with the addition of speech-to-text and Date Detection whereas Vishesh created the configuration page, as well as added features such as text-to-speech, routing and sorting. We helped each other during any difficulty faced.

References

[1] “Gmailapi gmail.v1 library documentation.” [Online]. Available: https://pub.dev/documentation/googleapis/latest/gmail.v1/gmail.v1-library.html

[2] “Mime protocol’s multipart content type documentation.” [Online]. Available: https://www.w3.org/Protocols/rfc1341/7_2_Multipart.html

[3] “Googleapis authorization scopes information.” [Online]. Available: https://developers.google.com/identity/protocols/oauth2/scopes

[4] “Flutter widget documentation.” [Online]. Available: https://docs.flutter.dev/development/ui/widgets-intro

[5] “Regular expression tester.” [Online]. Available: https://regex101.com/

[6] “Flutter text-to-speech package.” [Online]. Available: https://pub.dev/packages/flutter_tts

[7] “Flutter speech-to-text package.” [Online]. Available: https://pub.dev/packages/speech_to_text

[8] “Flutter external_app_launcher package.” [Online]. Available: https://pub.dev/packages/external_app_launcher

[9] “Least frequently used replacement policy.” [Online]. Available: https://www.geeksforgeeks.org/least-frequently-used-lfu-cache-implementation/