Automatic Text Detection in Images using Google Cloud Vision API

By

Sumbal Samad

2015-NUST-MS-IT

00000117522

Supervisor

Dr. Osman Hassan

Department of Electrical Engineering

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

In

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(June 2019)

Approval

It is certified that the contents and form of the thesis entitled "**Automatic Text Detection in Images using Google Cloud Vision API**" submitted by **Sumbal Samad** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Osman Hassan**
Signature: ____________________

Date: ____________________

Committee Member 1: **Dr. Sharifullah Khan**

Signature: ____________________
Date: ____________________

Committee Member 2: **Dr. Sohail Iqbal**

Signature: ____________________
Date: ____________________

Committee Member 3: **Asad Shah**

Signature: ____________________
Date: ____________________

Abstract

There has been significant growth in daily emails which is estimated to be over 293 billion by the end of 2019 [11]. About 90% emails are machine-generated sent from businesses to consumers such as confirmation, reservation, online purchase receipt etc. While a large fraction of them are commercial emails promoting an offer such as 40% off on entire store, buy 1 get 1 free. The sheer volume imposes a burden and users are unable to read and manage all of these emails. Consequently, people hesitate while subscribing to an online store. To fill this gap between companies and consumers third party apps have been introduced by many developers. These apps assist businesses to reach out consumers by notifying their promotional emails to app users at earliest. These apps have been manually processing promotional emails to extract key information, which will eventually be sent as notifications. Due to Manual processing third-party app's performance has been affected as it costs more resources, time and prone to errors.

To solve aforementioned problems, this thesis presents a solution aimed at performing the above-mentioned manual process of information extraction from promotional emails with minimal human assistance. Automatic extraction of key information unlocks the value of promotional emails and assists third-party apps to make timely updates without human intervention. In order to automate, we have developed a web app which can retrieve incoming mails, extract images from machine-generated emails and preprocess them, perform OCR, extract required information and save information on back-end server periodically. In addition the proposed app, fetches emails using Gmail API after user authorize app through OAuth 2.0 protocol, extracts images using lxml parser and performs OCR using Google cloud Vision API [5].

Dedication

I dedicate this thesis to my Parents and Siblings.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Sumbal Samad**
Signature: ________________

Acknowledgment

I would like to thank my Advisor and friends.

Table of Contents

List of Figures

Chapter 1

Introduction

Objective of this thesis is to assist third party apps by extracting key information from commercial emails. These days promotional emails are machine-generated and consists of heavy-images. Processing these image-rich emails using machine learning and applying it to each email would be very costly in terms of processing. In addition, applying model to varying emails can be very difficult as compared to static emails where accuracy can be achieved using template induction. We have given a cost-effective method and used off-the-shelf techniques. Implementation details are given in Chapter 4.

1.1 Motivation

Needless to say that harnessing of email marketing is an influential channel adopted by digital marketers. Digital marketing encapsulates multiple methods like social media campaigns, Google AdWords and email marketing. Email marketing is important channel for businesses to directly reach out their customers in an effective way. In addition, email marketing is a cost effective method as compared to traditional ways of advertisements. Due to above-mentioned advantages, there is rapid growth in utilization of commercial emails for promotions by businesses. In turn, users feel it difficult to find out which emails are worth reading. Third-party apps solve their problems by sending notifications without having them to navigate through emails and freeing them from hassle of subscription. Some apps also have coupon and rewards systems. So, we need to facilitate third-party apps in automation of their manual process, so businesses can target larger audience through indirect channel.

1.2 Prior State of Art

A lot of work has been done to pipeline services from different platforms and making them useful based on triggers and actions. Such platforms get data from one source application. Process and save data to the targeted application automatically e.g. sending a webhook to pushover etc. Zapier is one of those platforms which provide orchestration between multiple platforms like Facebook, Gmail, and twitter [12]. User can define different sources of data and then data is saved to their targeted applications based on different triggers. It also provides the facility to fetch emails from source application like Gmail and give multiple options to trigger events like new email, new attachment, new label, and email from specific sender than save emails to sink application such as firebase.

IFTTT(IF-THIS-THAN-THAT) [8] also works for the integration of different services. IFTTT focuses on automation and connectivity of mobile devices and accessories. Microsoft workflow is another platform which offers automation based on event-to-action model [10]. It offers multiple services as connectors for the automation of repetitive task. However, these services do not cater to the need of users who want to extract data from their HTML emails. All of the above-mentioned platforms unable to extract information from machine-generated mails as they only get data from one source and save data to another platform. However, mailparser is such a platform which provides extraction of data from email's attachment [9]. Email parser asks users to configure their cron job like sniping the attributes from pdf file such that next time when an email is triggered parser can automatically snip data according to previous snip positions and save data for the user. This also lacks in the processing of HTML email like extraction of images embedded in HTML content-type of email.

As we exploit google cloud vision API for text detection similarly Shin and YiHui [15] make use of the same API for for accessing images using the text they have insight. Content-based image retrieval was done by annotating images using API. As API does not consider the synonyms, word net was used for synonyms and it improved accuracy by 10%. Orekondy et al. (2018) also used vision API for the automatic redaction of private information from images [24]. GSV has been exploited to find recognize text, face and landmark's bounding boxes for redaction. Hosseini et al. evaluate API on ImageNet dataset by adding noise to images [19]. Results are completely different by adding 14.25% of noise. This paper [22] extract useful information from camera-images and able to achieve 98% after preprocessing images. Camera-images have noise, contrast and light issues, therefore, preprocessing helped in achieving better results.

Recently, efforts have also been made to classify non-spam machine-generated emails into different categories using embedded contents. Gmail used to classify emails into different tabs such as primary, social and promotional by analyzing embedded contents in emails [25]. Clustering has been done using template induction technique [14]. Template induction technique has been widely used for purposes like email threading [14] and hierarchical classification [29]. Template induction technique can be applied when emails have static layout and structured information whereas promotional emails have unstructured information. Template induction has also been applied over well-known Enron email dataset which is unstructured in nature [26]. Gmail also suggests auto-response which is known as smart-reply. Smart-reply helps users to send a small response as a reply, these responses have been generated using LSTM [20].

1.3 Problem Description

Manual processing has been an obstacle to scale especially for small businesses where resources are limited. In addition, manual processing is not time efficient while third-party apps are time-bound because promotions should be sent to user before they expire. If apps would unable to give latest deals to its users then they might get bored and may not continue to use apps. Due to manual processing app's performance has been affected as it cost more resources, time and prone to errors.

1.4 Proposed Solution

In this thesis, we propose to overcome above-mentioned limitations by automating the hand-operated process, aimed at performing with minimum human-intervention. A very similar approach have been formally applied for automation extraction of information from business cards [22]. However, to the best of our knowledge no automation have been made for information extraction from images of HTML emails specifically in regards of thirdparty apps.

Proposed solution is based on web app, which fulfill essential requirements of our client. First requirement is to process emails while respecting user's privacy concerns, which we achieved through authorization server [7]. Another requirement is to extract and process images from machine-generated emails, accomplished through beautiful soup. Foremost requirement is to perform OCR while making system efficient in terms of processing, which is done by using off-the-shelf component. Implementation details has been given in chapter 4

1.5 Huntasale Application as Real Client

Huntasale is Pakistan's first sale hunting mobile app [6]. It assists businesses to reach out consumers by notifying their promotional emails to app users at earliest. Huntasale is our real client. We will implement our solution to facilitate huntasale in automating their manual process. Our solution will help them in achieving their business goals efficiently.

Chapter 2

Literature Review

As Google cloud vision API was launched in 2015 therefore literature have very few examples. However a very similar approach is used in a paper, in which an android app can extract information from business cards, newspaper using vision API and able to achieve 98% accuracy on business cards [22].Following section contains related work done in frameworks based on trigger-action model and technologies used in solution.

2.1 Trigger-Action Model

Zapier and IFTT are based on trigger-action model with the aim of providing customizability and usability. IFTT focuses on consumer services and devices while Zapier focuses on making complex task easy such as project management and marketing automation. This paper [27] produced and analyzed 6406 triggers and 1051 channels from both platforms to analyze user's ease of use to create and manage zaps/recipes. For example Zapier allow user to define a zap which can automatically save attachment to Google drive whenever defined sender sends and email. But none of zaps allow user for text recognition from images embedded in html email which is requirement of our client. Email-parser [13] by zappier provides a visual interface for defining rules on email data. This app allows extracting information from attachments by snipping at time of configuration which means that one zap can handle structural information whereas our data is highly unstructured and artistic in nature.

2.2 OCR (Opticle Character Recognition)

History of OCR in field of pattern recognition is quiet old. There was a time where everyone took subject of OCR, because characters were much easy to deal with and new problem which needs to be solved. However, after paper published by C.Y. Suen's [23] modify their interest and explore new dimensions of pattern recognition, like 3-D object recognition and image processing for best results. OCR solves many problems which are common to other pattern recognition problems.

OCR have been used in many fields like in banking for check processing without involvement of human, in healthcare for forms processing, and digital pen [16] and smart glasses [21] is latest application of OCR. Optical character recognition not only used for scanning but also help blind person. There are always some major problems for OCRs. For example it's difficult to differentiate and recognize text in cluttered images, also letters like O can be consider zero and vice versa. Researchers have worked a lot for hand written text recognition and converting scanned documents into editable form. Detecting text in images has been done for spam filtering. As to break spam algorithm, they send emails with images with embedded text. So they use OCR to detect text in attachments for spam filtering [18]. This paper [17] shows accuracy of 87% using Naive Bayes algorithm to identify spam in images.

To improve the accuracy there are many customized solutions available like freeOCR which use the Tesseract as OCR and its own algorithm for image pre-processing and post-processing. Algorithm developed for that purpose was imgHOG [28], where HOG was histogram of gradient. This program converts the image into a form on which OCR gives best result. HOG extract region of interest by detecting features of interest. Features of interests can be found by using PCA. When there are a lot of features in an image, so to find principle component which gives high variance or information, researchers used some technique such as PCA, Binning etc. They can also do some normalization by binning. Mostly OCR do so by taking 4X4 or 8X8 blocks or image and each block have number of gradients within itself. Then they decide one gradient orientation bin based on frequency of gradients. It can detect any shape but it is slow. Another way of implementing histogram of gradient is by sampling regions of interest using kernel trick. It can be linear kernel or polynomial kernel.

2.3 Comparison between Tesseract and Google Cloud Vision

We have perform experiments on tesseract OCR .Studies shows that tesseract gives better result when images are converted to bilevel. So we binarize the images for better results. Because of artistic style of fonts tesseract could not perform well. Actual and preprocessed images are shown in figure 2.1 Result of tesseract and Google vision api is shown in figure 2.2

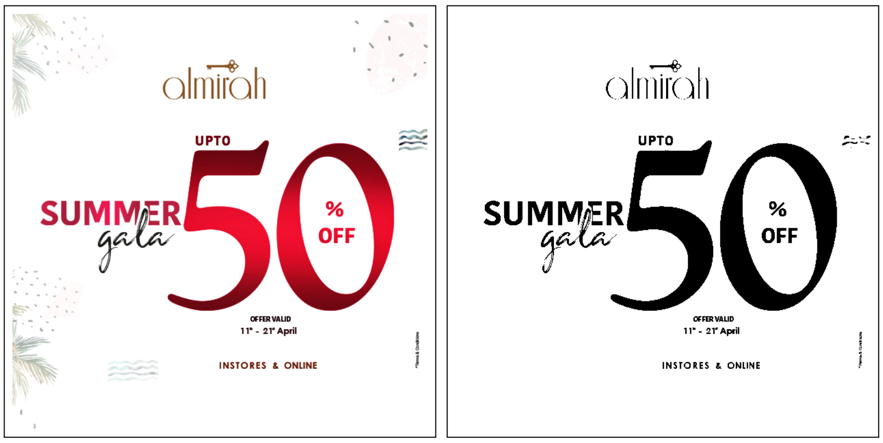
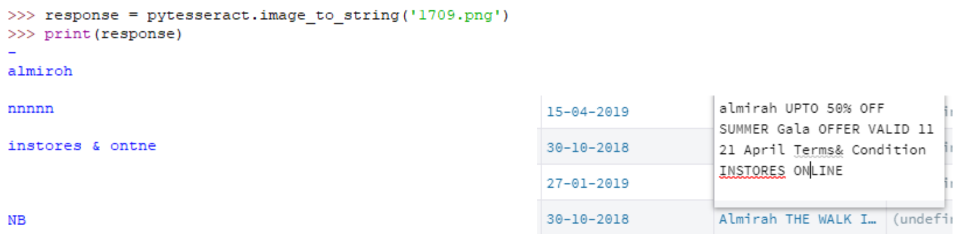


Figure 2.1: Colored & Bi-level image



```
>>> response = pytesseract.image_to_string('1709.png')
>>> print(response)
-
almiroh

nnnnn

instores & ontne



NB
```

```
15-04-2019   almirah UPTO 50% OFF
             SUMMER Gala OFFER VALID 11
30-10-2018   21 April Terms& Condition
             INSTORES ONLINE
27-01-2019

30-10-2018   Almirah THE WALK I…  (undefi
```

Figure 2.2: Tesseract vs Vision API

Chapter 3

Framework and APIs

3.1 Django

Django is a Python-based framework for web development [2]. Django 1.11.7 version is used in this application. It promotes modular programming which makes it possible for us to create separate applications for each purpose, such as registration application, authentication application, and Data source application.

3.2 OAuth 2.0

OAuth 2.0 protocol is used to access user-owned resources [7]. This protocol is used to access user's mailbox on behalf of user by using credentials granted by authorization server. We have used authorization code access grant for accessing user's mailbox in this application. Implicit grant is also used for creating service of google-cloud vision by using ouath2client library.

3.3 Gmail API

Google provides Gmail API [4] for accessing mailboxes. It allows developers to get access to user's mailbox by using different scope. Scope defines what application can do in user's mailbox on behalf of user. First application needs user consent and grant. There are different types of grants implicit code, authorization code depending upon application and scope type. We have created project on google console and enabled Gmail API.

3.4 Google Cloud Vision API

Google Cloud Vision Api is used in this project for text detection [5]. Document Text Detection feature is used in this application.

3.5 Email and MIME Package

This package is used to process emails [3]. This package is used to extract html content type part of email. This package creates object model of email which makes easy to traverse each part of email.

3.6 bs4

Beautiful Soup is python library, which has been widely used for extraction of information from webpages.We also have used in our system for extraction of URL from machine-generated emails.

Chapter 4

Implementation

4.1 Methodology

Following section describes key challenges and their corresponding solutions in order to achieve defined objective. Methodology diagram is shown in figure 4.2 and following is description of each step

1. While accessing emails privacy is main concern, we ensure to respect user's privacy by using authorization protocol. In addition, our application will have access_token rather than actual passwords and user can revoke access at any time. As shown in diagram users grant access for his protected resource/mailbox to authorization server (Oauth 2.0).

2. Authorization server issue credentials to our app, only then our app can make requests to protected resource using Gmail API on behalf of user.

3. App will make GET requests to Gmail API for mails, corresponding to each configured job defined by user. In order to minimize network load we exploit user's defined query (from:xyz,OR subject: 40% off, OR search:'sale'). Query works same as people do search in Gmail mailbox.

4. In previous step, Ids of all messages are returned so now application will make request to Gmail API for each message received previously.

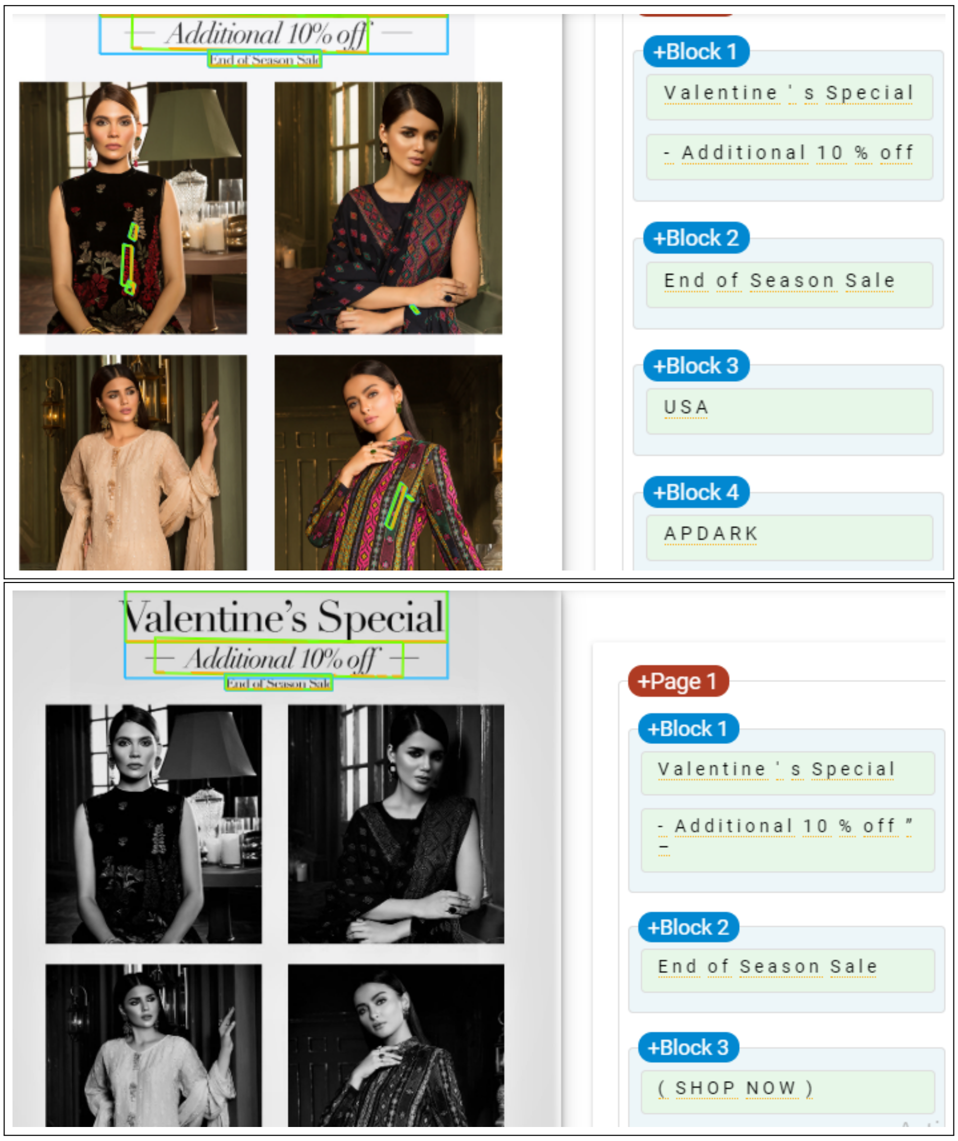
5. Each message is sent to email parser, where each part is traversed and html content_type is saved.

6. Images are extracted by getting src tag of all ¡img¿ tags. This step creates more hurdles as there are multiple img tags inside machine generated emails, so it returns a large number of images of all sizes. Filtering them after performing OCR would increase cost, time and network usage. We minimize number of images by applying threshold on size attribute based on our observation of templates.

7. It is more difficult to recognize text on synthetic images rather than natural images. As images used in promotional emails have background variations and artistic fonts. Effects of background are shown in figure 4.1, where GCV marks embroidery as text blocks. We overcame this issue by binerizing images and increasing their contrast and it is also shown in figure 4.1. After preprocessing, images are then sent to cloud for text detection.

8. Response received from cloud will be processed for information extraction by using regex.

9. Processed text will be sent to parse server and saved in application's database as well.



+Block 1
Valentine ' s Special
- Additional 10 % off

+Block 2
End of Season Sale

+Block 3
USA

+Block 4
APDARK

+Page 1

+Block 1
Valentine ' s Special
- Additional 10 % off "
–

+Block 2
End of Season Sale

+Block 3
(SHOP NOW)

Figure 4.1: Effect of Background

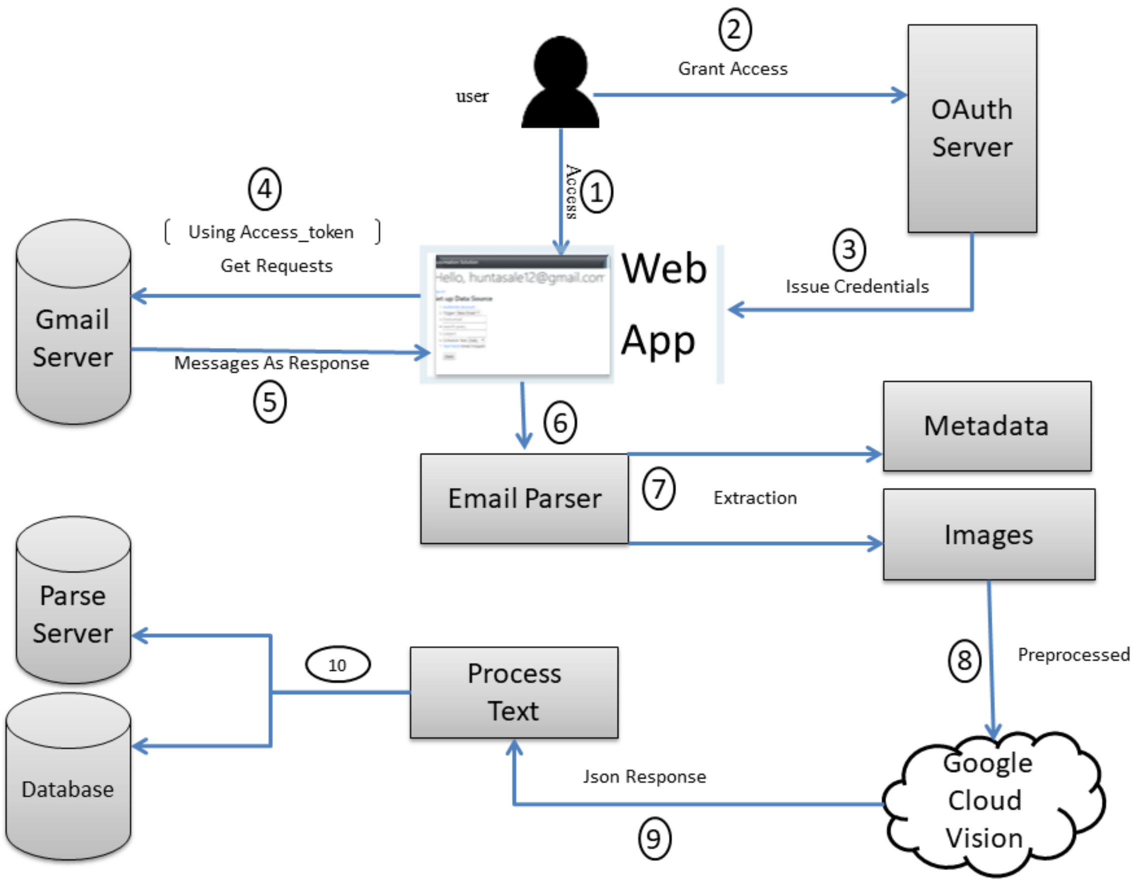


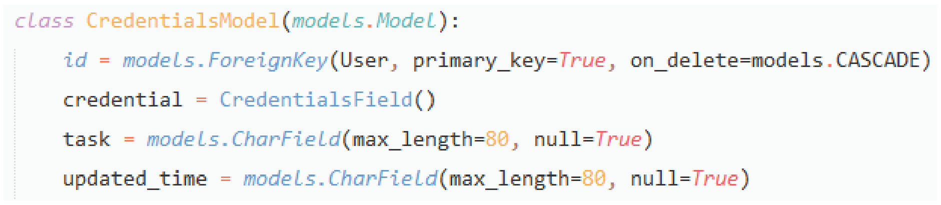
Figure 4.2: Methodology

4.1.1 MVT

This application is developed using model-view-Template software design pattern. However Django is based on MVC(Model-view-controller) but controller is handled by framework itself. In MVT model is responsible for carrying data and view carries business logic and then results are rendered on template.

Models

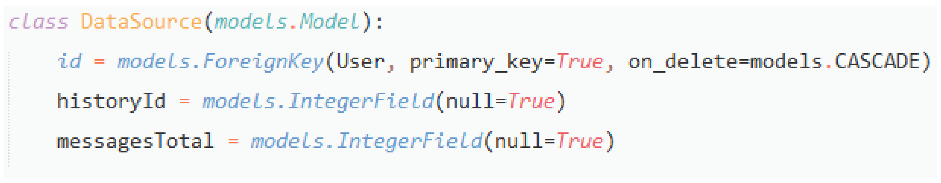
Models used in this application are User, CredentialsModel, CredentialsAdmin, DataSource, Task and TaskMessages. User model is used for user signup, sign in. User model is imported from django.contrib.auth.models. Credentials Model is used to carry data retrieved from authorization server. Access token and refresh_token are saved in credentailsfield which is supported by oauth2client library. Id from User model is used to create a one-to-one relationship. Model is shown in following figure.



```
class CredentialsModel(models.Model):
    id = models.ForeignKey(User, primary_key=True, on_delete=models.CASCADE)
    credential = CredentialsField()
    task = models.CharField(max_length=80, null=True)
    updated_time = models.CharField(max_length=80, null=True)
```

Figure 4.3: Credential_Model

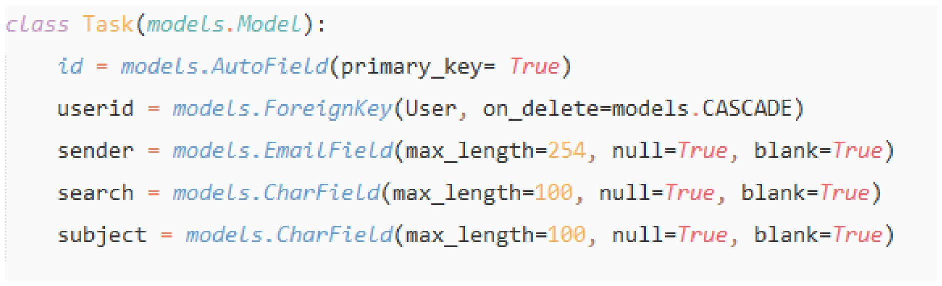
Datasource model is used to store information of given mailbox. History Id and total messages are used to save and then look for changes of user's mailbox. If there is change in history Id then cron job is executed and mails are fetched using access token. Model is shown in following figure.
Task model is carrying parameters used to fetch mails from user's mailbox.



```
class DataSource(models.Model):
    id = models.ForeignKey(User, primary_key=True, on_delete=models.CASCADE)
    historyId = models.IntegerField(null=True)
    messagesTotal = models.IntegerField(null=True)
```

Figure 4.4: DataSource_Model

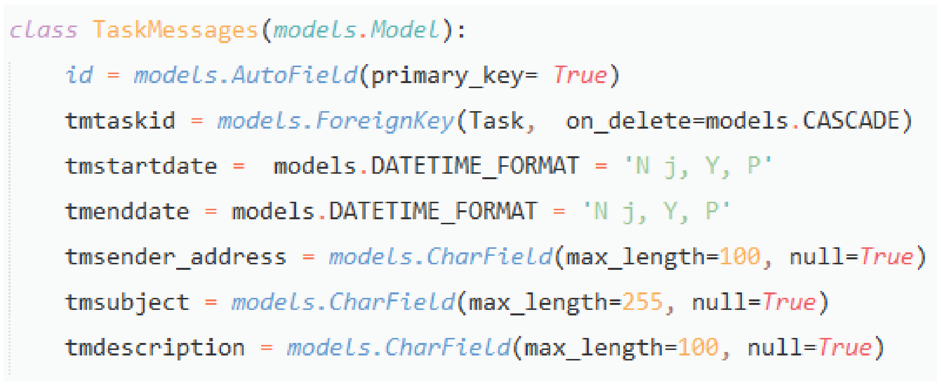
Fields sender, search and subject are used to create query which will be sent in request. User can create query as much as specific he wants by supplying values to all three variables or can create only one filter (sender, subject, search query). Model is shown in following figure.



```
class Task(models.Model):
    id = models.AutoField(primary_key= True)
    userid = models.ForeignKey(User, on_delete=models.CASCADE)
    sender = models.EmailField(max_length=254, null=True, blank=True)
    search = models.CharField(max_length=100, null=True, blank=True)
    subject = models.CharField(max_length=100, null=True, blank=True)
```

Figure 4.5: Task_Model

TaskMessages is model where results are saved after processing. Currently start date end date sender and description are being saved as per requirements. Model is shown in following figure.



```
class TaskMessages(models.Model):
    id = models.AutoField(primary_key= True)
    tmtaskid = models.ForeignKey(Task,  on_delete=models.CASCADE)
    tmstartdate =  models.DATETIME_FORMAT = 'N j, Y, P'
    tmenddate = models.DATETIME_FORMAT = 'N j, Y, P'
    tmsender_address = models.CharField(max_length=100, null=True)
    tmsubject = models.CharField(max_length=255, null=True)
    tmdescription = models.CharField(max_length=100, null=True)
```

Figure 4.6: Messages_Model

Views

Django offers function-based and class-based views. Function-based views have been used in this application. There are two important views one is responsible for handling Http Requests for OAuth and other is responsible for processing emails.

Authentication view starts from creating Flow from Client secrets downloaded from Google console with parameters of scope, redirect_uri and prompt. Scope defines level of grant application needs from user to access its resources. After defining scope it check Credentials model if user has already been authorized our application with credentials, if credentials have been found then it creates a http object for sending request to user resource and then build service object by defining Api, version and http. For new users it generates a URL for authorization by using flow created in first step. If user has granted access then it redirects to redirect URI defined in flow. Authentication view's Flowchart is shown in following Figure 4.6.

Second View will be run as cron job and flowchart is shown in figure 4.7. It is dedicated for getting all task_ids and make list.messages request to Gmail API.

```
response = service.users().messages().list(userId='me', q=query
   ↪ ).execute()
```

Response may have one or multiple messages, however Get request is made for each id in response.

```
message = service.users().messages().get(userId='me',id='12159
   ↪ b98531c9853c5', format='raw', alt = 'json').execute()
```

After getting message in raw format it is decoded by using following code.

```
decoded_message = base64.urlsafe_b64decode(message[key].encode
   ↪ ('UTF8'))
```

Once message is decoded it is traversed using email [3] and html conten_types will be saved. Then these images are extracted using beautiful soup [1]. Images are then filtered out to find useful images so that vision requests could be minimized. Images are then sent to cloud and response is returned as json format. Preprocessing is done to extract information and align text.

```
vision_client = vision.ImageAnnotatorClient()
with io.open(image_path, ''rb') as image:
vision_content = image.read()
image_type = vision.types.Image(content=vision_content)
ocr_response = vision_client.document_text_detection(image=
   ↪ image_type)
```

Templates

Templates are pages where requests and responses have been shown. Templates are coded in django templating language.Templates will be further explained in Section 4.1.2.

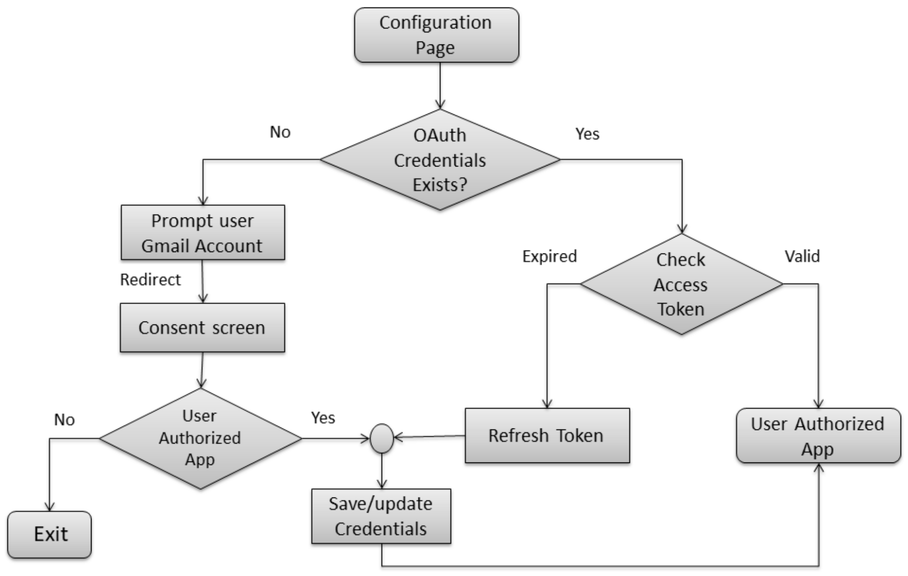


Figure 4.7: Application Authorization Flow

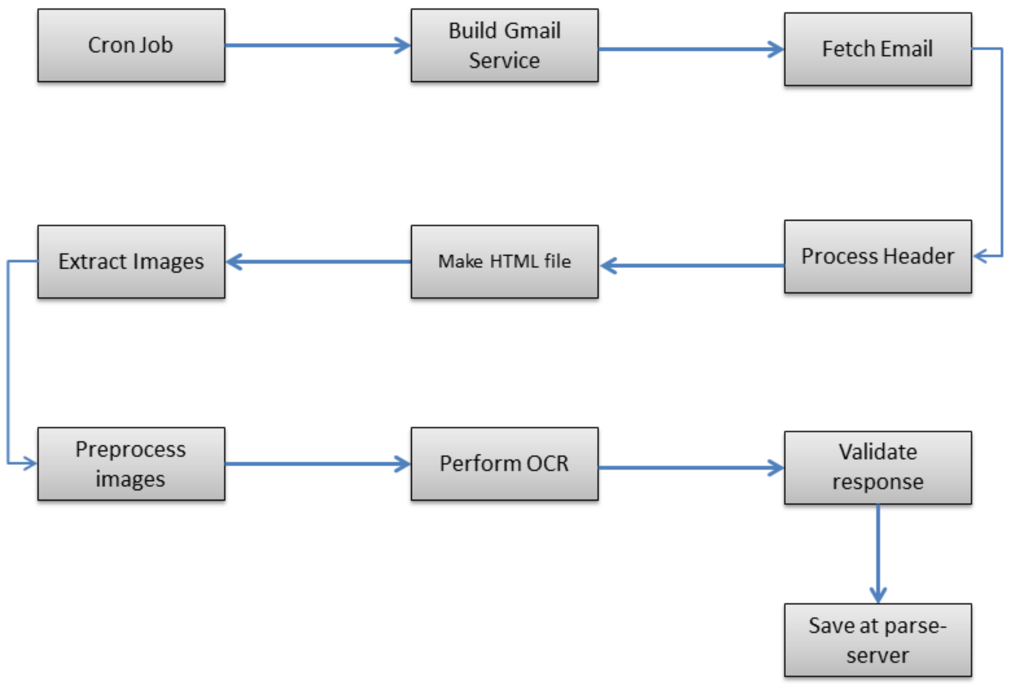


Figure 4.8: Cron View

4.1.2 Interface

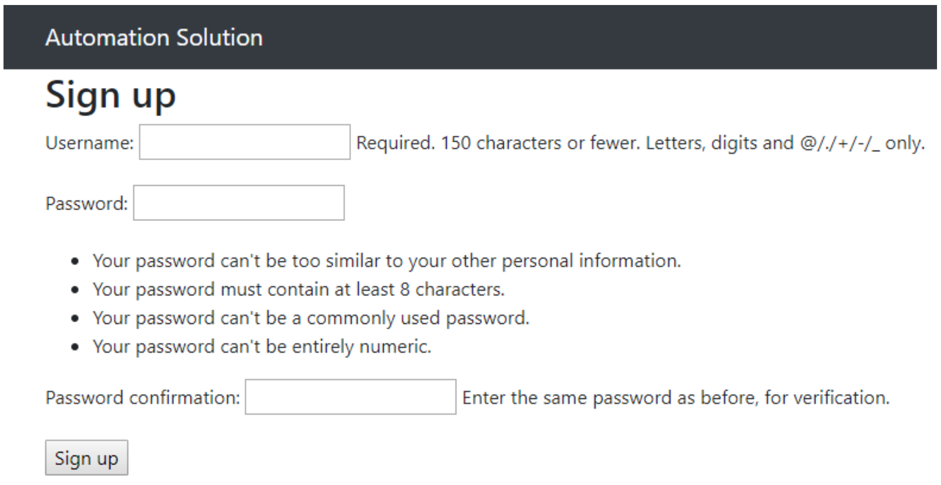
This section contains description of front-end of application.

User module

This module have registration module has signup and login pages. Signup view is created using Django generic views.

Signup Page

Signup page is used for creating users other than admin while admin is created by using this command python manage.py createsuperuser. Sign up page is shown in following.


Automation Solution

Sign up

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

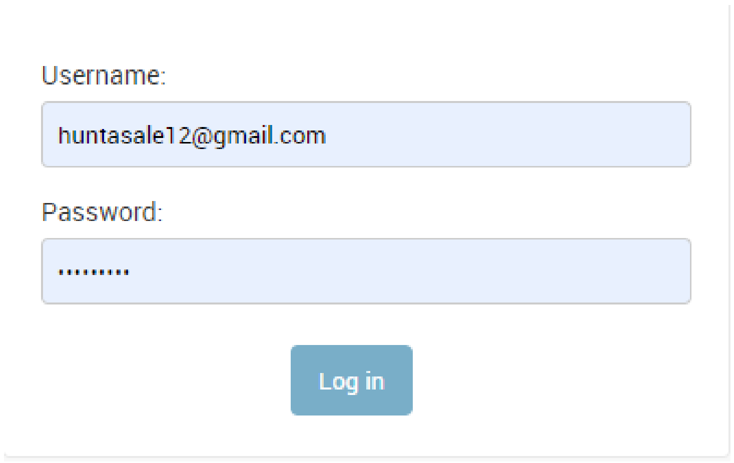
Password confirmation: Enter the same password as before, for verification.

Sign up

Figure 4.9: Sign up Page

Login Page

Login page for admin and user is shown following figure.


Username:

huntasale12@gmail.com

Password:

•••••••••

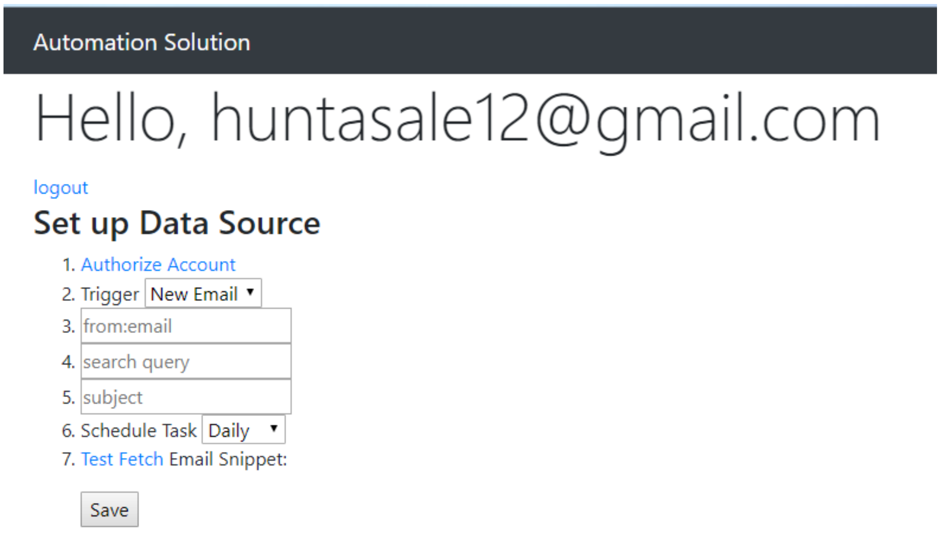
Figure 4.10: Login Page

Configuration page

This page is used to configure Source of data. Only authorized users can access this page. Authorize account is link for authorizing this application with Gmail credentials. Users can verify settings by clicking on Test Fetch after filling input boxes. For example a user can input address of sender and click on test fetch then snippet of first email from that sender will be shown on this page. Authorization and testing requests are sent through ajax, authorization steps are shown in figure 4.10. Once user grant access than application redirected to redirect URI.

4.2 Admin Module

This module consists of administrative site of application. Admin have dashboard where he can add, delete and update models. Dashboard page is shown in figure 4.11 Admin can also see number of deals which have been created


Automation Solution

Hello, huntasale12@gmail.com

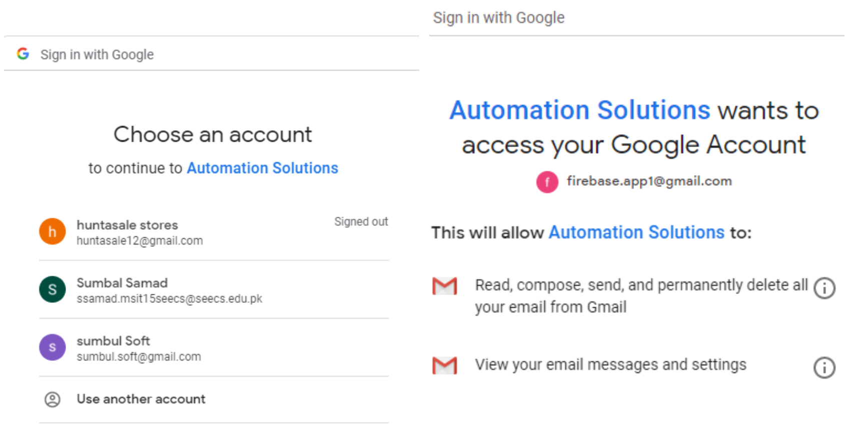
logout

Set up Data Source

1. Authorize Account
2. Trigger New Email
3. from:email
4. search query
5. subject
6. Schedule Task Daily
7. Test Fetch Email Snippet:

Save

Figure 4.11: Configuration Page


Sign in with Google

Choose an account

to continue to Automation Solutions

huntasale stores huntasale12@gmail.com Signed out

Sumbal Samad ssamad.msit15seecs@seecs.edu.pk

sumbul Soft sumbul.soft@gmail.com

Use another account

Sign in with Google

Automation Solutions wants to access your Google Account

firebase.app1@gmail.com

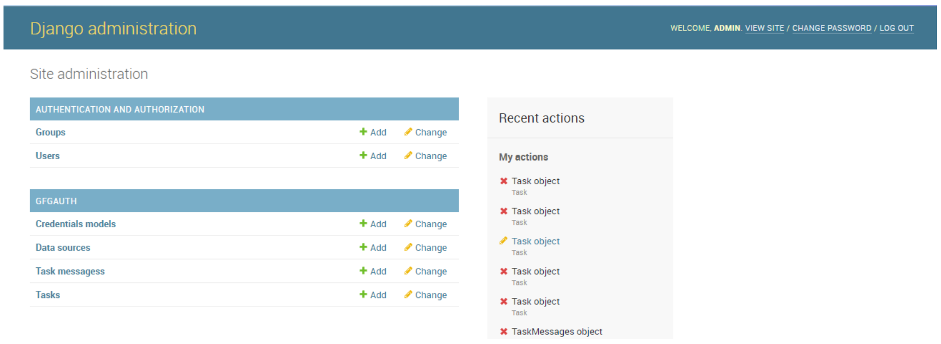
This will allow Automation Solutions to:

Read, compose, send, and permanently delete all your email from Gmail

View your email messages and settings

Figure 4.12: Authorization steps

till date. Deals are being saved in both databases one is website's database and other is parse-server. Admin can change and edit text in case of OCR errors form following page figure 4.12.


Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

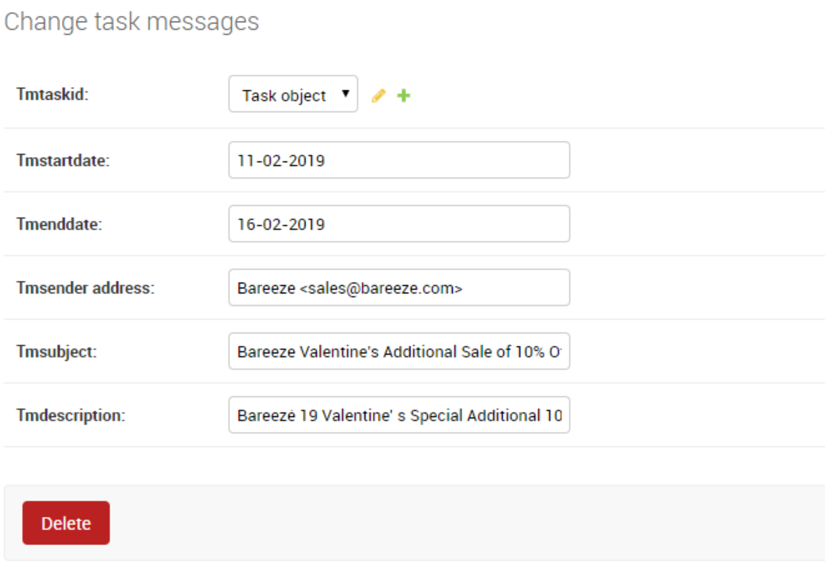
GFGAUTH		
Credentials models	+ Add	Change
Data sources	+ Add	Change
Task messagess	+ Add	Change
Tasks	+ Add	Change

Recent actions

My actions

- Task object Task
- Task object Task
- Task object Task
- Task object Task
- Task object Task
- TaskMessages object

Figure 4.13: Administrative Site


Change task messages

Tmtaskid:	Task object
Tmstartdate:	11-02-2019
Tmenddate:	16-02-2019
Tmsender address:	Bareeze <sales@bareeze.com>
Tmsubject:	Bareeze Valentine's Additional Sale of 10% O
Tmdescription:	Bareeze 19 Valentine' s Special Additional 10

Delete

Figure 4.14:

4.3 Database Design

This section shows that database structure for the website. E-R Diagram of website is given below figure 4.13

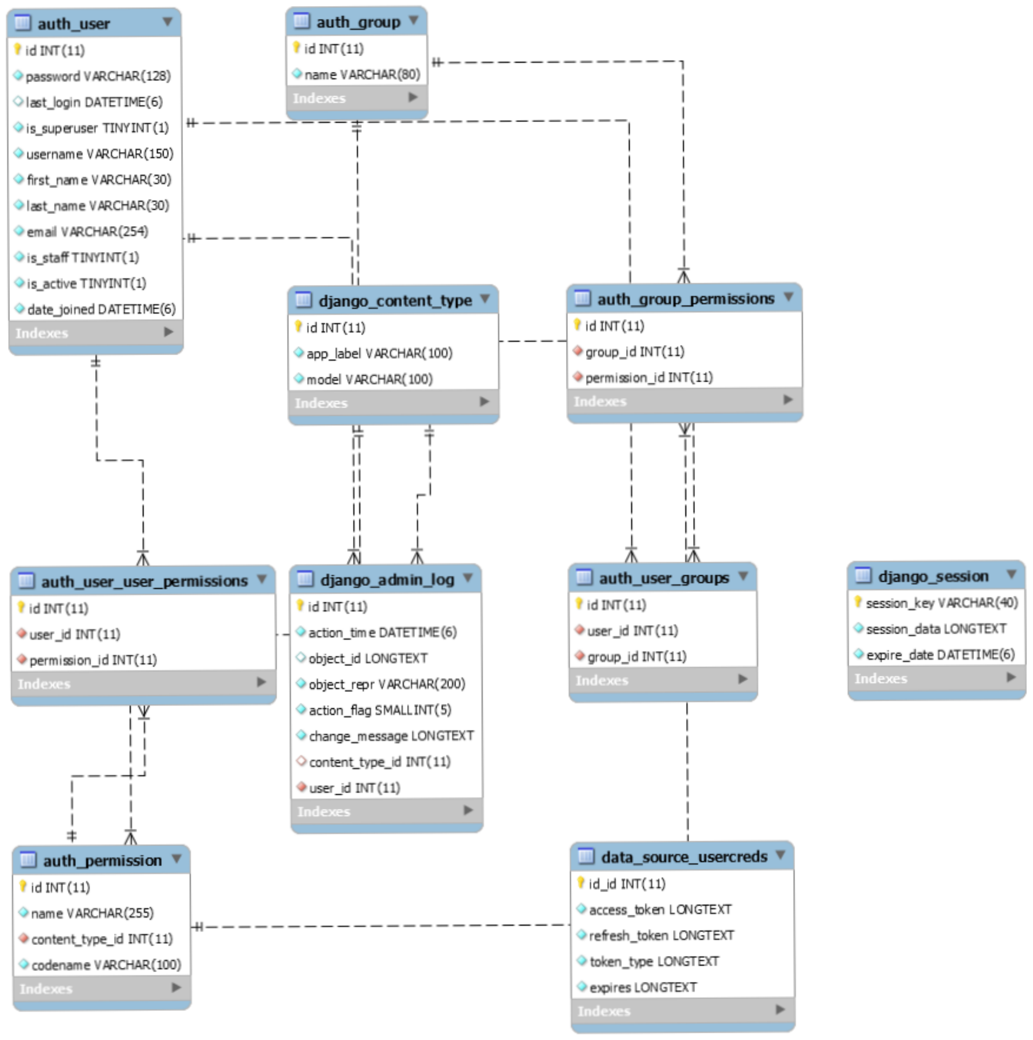


Figure 4.15: E-R Diagram

4.4 Pricing and Quotes

Pricing includes charges incurred for document text detection feature of google cloud vision API. Google cloud vision API is providing free OCR service for 11000 images per month but if our images cross this limit then Google will charge 1.50$ for 5 million images per month. Google Cloud vision gives free trial of 300$ and 6.93$ of free trial have been used in development. Gmail API defines quota for each user. It allows 250 quota units per user per seconds.

Chapter 5

Conclusions

5.1 Summary

In this research we have implemented a web application which will automate manual process of Huntasale. This application will make Huntasale app efficient in terms of timely deals updates. This automation creates compounding effects for Huntasale because cost has become an efficiency, which frees up people, resources and time.

Moreover, we showed that Google cloud vision API can be easily used for processing image-rich emails without having to run exorbitant machine learning models on emails. In essence, we also found that google cloud vision API can extract large fonts if we upscale images. We also showed that background issues can be resolved by applying vision API on binary images rather than original images.

5.2 Future Directions

We plan to extend our system beyond extraction of information from promotional emails for extraction from scanned documents received in emails such as reports and receipts. However, algorithm for text extraction will be required for relevant information extraction according to domain of documents.

Bibliography

[1] *Beautiful Soup, https://www.crummy.com/software/BeautifulSoup/bs4/doc/, 2019.*

[2] *Django, https://www.djangoproject.com, 2019.*

[3] *Email Library, https://docs.python.org/3/library/email.html, 2019.*

[4] *Gmail, https://developers.google.com/gmail/api/, 2019.*

[5] *Google Cloud Vision, https://cloud.google.com/vision/, 2019.*

[6] *Huntasale, http://huntasale.com/, 2019.*

[7] *IETF, https://tools.ietf.org/html/rfc6749/section-1.3.4, 2019.*

[8] *IFTTT, https://ifttt.com/, 2019.*

[9] *MailParser, https://mailparser.io/, 2019.*

[10] *Microsoft WorkFlow, https://flow.microsoft.com/en-us/, 2019.*

[11] *Then Radicati Group.2019.Email Statistics report, 2019-2023. https://www.radicati.com/?p=16037.*

[12] *Zapier, https://zapier.com/,2019.*

[13] *Zapier Parser, https://parser.zapier.com, 2019.*

[14] Nir Ailon, Zohar S Karnin, Edo Liberty, and Yoelle Maarek. Threading machine generated email. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 405–414. ACM, 2013.

[15] Shih-Hsin Chen and Yi-Hui Chen. A content-based image retrieval method based on the google cloud vision api and wordnet. In *Asian Conference on Intelligent Information and Database Systems*, pages 651–662. Springer, 2017.

[16] Stanford CoreNLP. a suite of core nlp tools. *URL http://nlp. stanford. edu/software/corenlp. shtml (Last accessed: 2013-09-06)*, page 3, 2016.

[17] Meghali Das and Vijay Prasad. Analysis of an image spam in email based on content analysis. In *Proc. Int. Conf. On Natural Language Processing And Cognitive Computing*, volume 201, 2014.

[18] Giorgio Fumera, Ignazio Pillai, and Fabio Roli. Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research*, 7(Dec):2699–2720, 2006.

[19] Hossein Hosseini, Baicen Xiao, and Radha Poovendran. Google's cloud vision api is not robust to noise. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 101–105. IEEE, 2017.

[20] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, et al. Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 955–964. ACM, 2016.

[21] Bernard Kress. Optics for smart glasses, smart eyewear, augmented reality and virtual reality headsets. *Fundamentals of Wearable Computing and Augmented Reality*, pages 85–124, 2015.

[22] Rafsanjany Kushol, Imamul Ahsan, and Md Nishat Raihan. An android-based useful text extraction framework using image and natural lan-

guage processing. *International Journal of Computer Theory and Engineering*, 10(3), 2018.

[23] Shunji Mori, Ching Y Suen, and Kazuhiko Yamamoto. Historical review of ocr research and development. *Proceedings of the IEEE*, 80(7):1029–1058, 1992.

[24] Tribhuvanesh Orekondy, Mario Fritz, and Bernt Schiele. Connecting pixels to privacy and utility: Automatic redaction of private information in images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8466–8475, 2018.

[25] Navneet Potti, James B Wendt, Qi Zhao, Sandeep Tata, and Marc Najork. Hidden in plain sight: Classifying emails using embedded image contents. 2018.

[26] Julia Proskurnia, Marc-Allen Cartright, Lluís Garcia-Pueyo, Ivo Krka, James B Wendt, Tobias Kaufmann, and Balint Miklos. Template induction over unstructured email corpora. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1521–1530. International World Wide Web Conferences Steering Committee, 2017.

[27] Amir Rahmati, Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Ifttt vs. zapier: A comparative study of trigger-action programming frameworks. *arXiv preprint arXiv:1709.02788*, 2017.

[28] Olarik Surinta, Mahir F Karaaba, Lambert RB Schomaker, and Marco A Wiering. Recognition of handwritten characters using local gradient feature descriptors. *Engineering Applications of Artificial Intelligence*, 45:405–414, 2015.

[29] James B Wendt, Michael Bendersky, Lluis Garcia-Pueyo, Vanja Josifovski, Balint Miklos, Ivo Krka, Amitabh Saikia, Jie Yang, Marc-Allen Cartright, and Sujith Ravi. Hierarchical label propagation and discovery

for machine generated email. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 317–326. ACM, 2016.