MASS EMAIL DETECTOR

Group 3

Published November 20th

OUR PROBLEM

Mass Emails affect nearly every profession. People find it hard to distinguish between an email made just for them or one that's been slightly altered and sent to numerous other people. This is particularly a problem with recruiting and with “one-time” offers.

Our solution is to provide a way for the recipient of an email to know how many other people have received a similar or slightly altered version of the same email, by providing the user with a clean visual interface displaying this information.

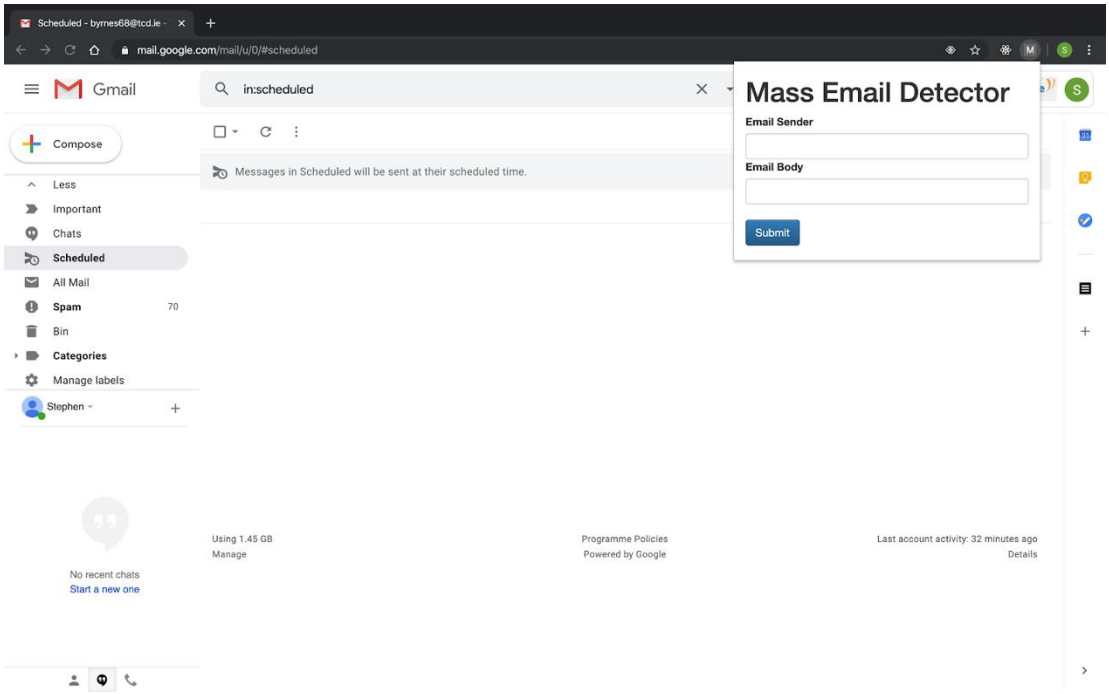
ITERATION ONE – THE CHROME PLUGIN

Our initial goal was to develop a browser bar plugin that could be toggled on or off, and when it was on it would detect mass emails and alert the user. Our backend system was built using Django and SQLite (detail included later).

Non-Technical Overview

We initially went for a minimum viable product (MVP) approach that simply allowed users to copy and paste the body of an email into a form located in the browser bar. We chose this as it could be used universally and not only with one mail client.

This was parsed and ran through an algorithm to develop a hash of similar emails. This value was then updated in our database and the number of similar mails with the same hash value returned. Then, an alert was displayed to inform the user if it was likely a mass email.

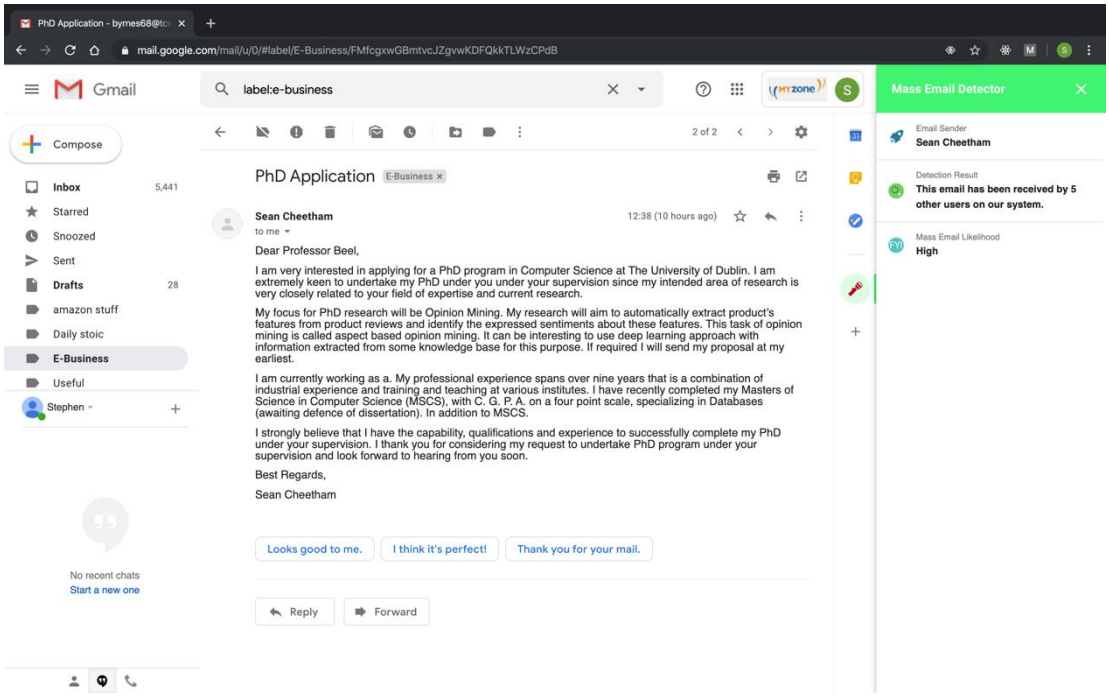


This solution worked, providing a solid base to move forward from, however the user experience was grim at best.

We decided to improve on this by having the email read automatically, however we ran into issues with different mail clients as well as difficulties isolating the body from the entire HTML page.

This led us to switching our front-end to using Googles Appscript and building a plugin tailored for Gmail as well as hosting our backend in the cloud.

ITERATION TWO – THE GMAIL ADDON



…. Success! (Video demonstration included below)

Front-End

Google Appscript is a development platform specifically tailored for G-suite apps. It is essentially a JavaScript platform in the cloud.

Appscript provides a host of services that enable developers to add functionality and compatibility across various G-suite apps. We utilised the "Card" service which allowed us to build a uniform (and hopefully user-friendly) interface.

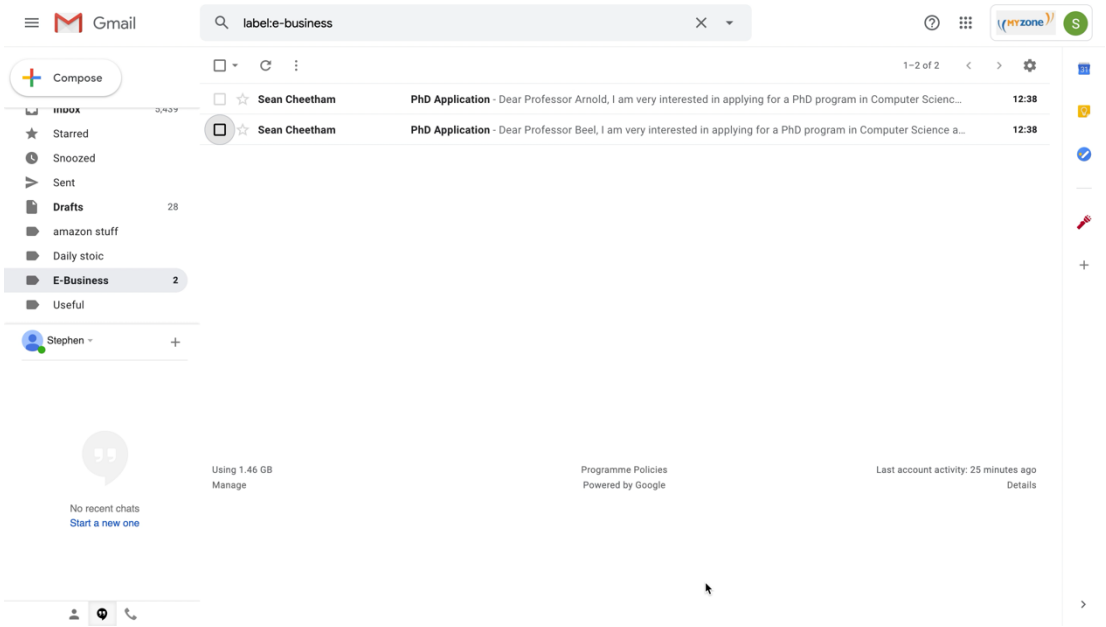
The UI built with card services automatically works across other devices (mobile) , so developers don't need to create separate UIs for each platform, resulting in a uniform experience across the board. This was perfect for our solution.

The getContextualAddon function solved our issue of not being able to automatically parse the email body as it returns the contextual data for the current email thread.

Using getContextualAddon to retrieve the body of the email, it is then parsed through our generalisation hashing algorithm and sent to our back-end via HTTP requests.

The cards displayed to the user are then generated based on the response from our REST API.

Video Demo



(Right-click to play)

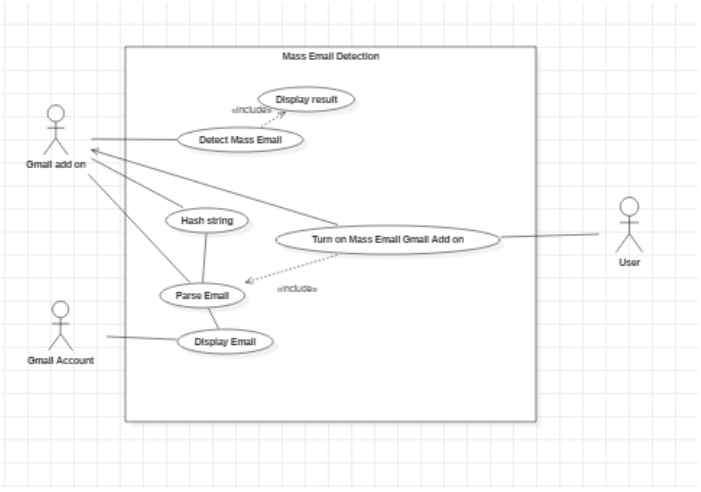
Generalisation and Hashing Algorithm

We developed an algorithm that can detect to some level of accuracy, similar emails. If several similar emails are detected in our database, it is likely the email is a mass email. This is sufficient for our MVP.

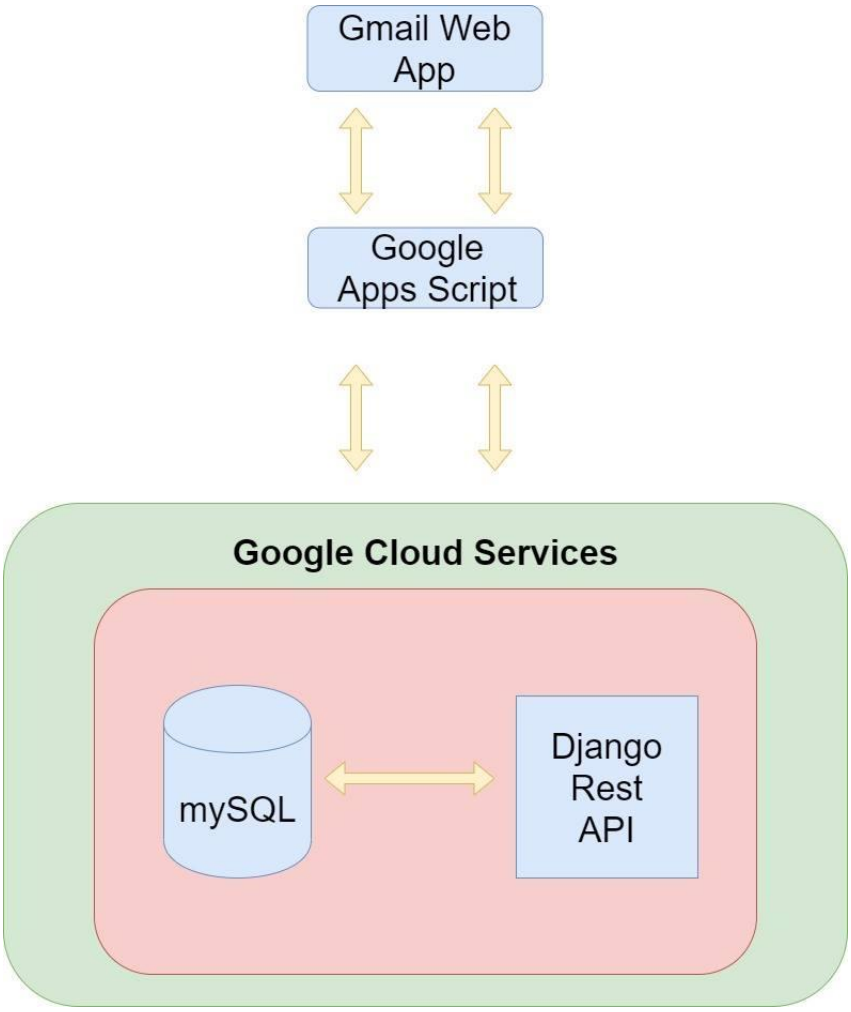
- Parse Email body as string.
- Remove interchangeable nouns (*)
- Remove white space.
- Use MD5 hash to hash string.

*Nouns not at the start of a sentence which are capitalised.

UML Use Case



System Architecture



Back-End

Our backend is built with a Django Restful API, which is used to interface with our MySQL database.

When the application moved from a Chrome plugin to a Gmail addon, the whole solution had to be moved to the cloud. To deploy our Django app online we decided to use Google Cloud Services.

Google's cloud services conveniently allowed us to create an online MySQL instance to host our database, that could seamlessly interface with our Django application. To host the Django application online we simply had to upload our Django app into a storage bucket, then our application could be deployed.

Our API is currently hosted on Google's Appspot domain, and will be scalable for future development.

Below we will discuss the structure of our API.

RESTful API Structure:

Address:
***http://host*/api/hashes/**

Call	Return object
GET	Returns a list of all hash objects - hashValue - Count Always returns an HTTP code 200
POST	Payload must contain a new hash value to be added to the SQL database A well-formed post call returns an HTTP code 201 CREATED.

Sample GET response

/api/hashes


```
[
  {
     "hashValue": "02e4bd27b4107fd8c6562d1c27fc50dd",
     "count": 9
   },
   {
     "hashValue": "04c7ed138dac7e2e863496ed7c0b6e79",
     "count": 2
   },
   {
     "hashValue": "0ce83c3f17286e4b7cdaeddb05dc04be",
     "count": 4
   },
   {
     "hashValue": "1b5d14aebb52459136bc3d232614f329",
     "count": 10
   },
   {
     "hashValue": "260f9d3cfd23beca9a667fe12b8563e7",
     "count": 2
   },
]
```

Sample POST payload

/api/hashes

```
{ "hashValue": "260f9d3cfd23beca9a667fe12b8563e7"}
```

This will initialise a new hashValue object in the table above.

http://host/api/hashes/<hashValue>

Call	Return object
GET	Returns a single hash value object defined in <hashValue> - hashValue - count If the value defined in <hashValue> exists an HTTP 200 OK code will be returned, otherwise an HTTP 404 NOT FOUND code will be returned
PUT	Update a record that currently exists, a successful update will return an HTTP 200 OK code.

Sample GET response

/api/hashes/02e4bd27b4107fd8c562d1c27fc50dd

```
{
    "hashValue": "02e4bd27b4107fd8c6562d1c27fc50dd",
    "count": 9
}
```

Sample PUT payload

/api/hashes/02e4bd27b4107fd8c562d1c27fc50dd

```
{
    "hashValue": "02e4bd27b4107fd8c6562d1c27fc50dd",
    "count": 10
}
```

Updating the value in the put request is incrementing the count by 10, this is how it will work inside the application.

Further Proposed Features

Our current version acts as a proof of concept for our problem, so there are many further additions that we would have liked to implement, but were unable to due to time constraints.

Add cookies to prevent the same email hash being sent to the server by the same user in a period of time.

- *Time required:* 1 week
- *Implementation:* Create a server side system that keeps track of which email address submitted which hash and when. This can be implemented securely by storing the receiver email as a hashed combination of the email hash and receiver email which will expire after an extended period of time (-1 month). This will prevent the same user from artificially bumping up the hash count values.

Expand from Gmail to other mail clients.

- *Time required*: 1 month
- *How to implement:* Create new front end clients to work with new email clients.

Limitations and Outlook

Our finished product is based on the assumption that numerous users will download and use the app. As the user base grows, it would get better at detecting mass emails as more users would be submitting hashes. With a low number of users it is ineffective.

However, eventually this would reach a tipping point where every possible email is hashed and the value exists. In this case we would need to improve our detection method by advancing our algorithm, possibly using machine learning techniques to detect mass emails, rather than our naive algorithm.

We are also aware of a bug where the email will be updated a second time if the page is refreshed. To fix this we would need to also store a hash of the users email along with the email and not update two submissions from the same user.

INSTALLATION

The application is currently in development so can only be tried out through a Gmail developer addon, to perform installation follow these steps:

1. Open the Gmail add-on settings tab. --
   > https://mail.google.com/mail/u/0/#settings/addons
2. In the Add-ons tab, make sure that you have selected the Enable developer add-ons for my account checkbox.
3. Paste your add-on's deployment ID into the Developer add-on textbox and click Install.

- Current app ID - AKfycbxueVIsTgZRR094YASCMWH9L49P_-8sbqAixvo387Q

Posted by: User4 at 12:00