A Mini Project Synopsis on

Web Content Sanitization

T.E. - I.T Engineering

Submitted By

Parth Bhoir	**19104050**
Pranav Mayekar	**19104040**
Anjali Singh	**20204006**

Under The Guidance Of

Prof. Nahid Shaikh

DEPARTMENT OF INFORMATION TECHNOLOGY

A.P. SHAH INSTITUTE OF TECHNOLOGY

G.B. Road, Kasarvadavali, Thane (W), Mumbai-400615

UNIVERSITY OF MUMBAI

Academic year: 2021-22

ACKNOWLEDGEMENT

First and foremost, we would like to thank our guide **Prof. Nahid Shaikh** for her invaluable expertise, guidance and support throughout our project. A debt of gratitude is also owed to our HOD **Prof. Kiran Deshpande**, and the Department of Information Technology for providing us with the opportunity as well as the support required to pursue this project. We would also like to thank our peers for their helpful suggestions.

CERTIFICATE

This to certify that the Mini Project report on **Web Content Sanitization** has been submitted by **Parth Bhoir (**19104050), **Anjali Singh (**20204006) and **Pranav Mayekar (**19104040) who are a Bonafede students of A. P. Shah Institute of Technology, Thane, Mumbai, as a partial fulfilment of the requirement for the degree in **Information Technology**, during the academic year **2021-2022** in the satisfactory manner as per the curriculum laid down by University of Mumbai.

Prof. Nahid Shaikh
Project Guide

Prof. Kiran Deshpande	Dr. Uttam D. Kolekar
Head of Department of Information Technology	Principal

External Examiner(s)
1.
2.

Place: A.P. Shah Institute of Technology, Thane
Date:

TABLE OF CONTENTS

1. Introduction:

There has been a steep increase in the use of social media in our everyday lives in recent years. Along with this, there has been an increase in hate speech disseminated on these platforms, due to the anonymity of the users as well as the ease of use.
Social media platforms need to filter and prevent the spread of hate speech to protect their users and society. Due to the high traffic, automatic detection of hate speech is necessary.

Hate speech detection is one of the most difficult classification challenges in text mining Hate speech is an attack towards a specific person or group based on characteristics such as race, ethnicity, religion, gender, age, disability. In recent years, the number of people using social media platforms has increased dramatically. These platforms have become a suitable place for people to express their feelings and anger toward each other Most of the social media platforms still face challenges regarding hate speech spreading on their platforms.

In most cases, these platforms rely on user reports or blacklists of offensive or hate-related words to identify and remove hate speech from their platform. Nevertheless, since in these cases, the posts containing hate speech appears on the platform before it can be deleted, hate speech may start spreading and harming society. Furthermore, these approaches are far from “automatically” detecting hate speech and they require expert teams or have poor performance since the language used in social media platforms is colloquial and is evolving continuously.

1.1. Purpose:

The purpose of this project is to help survey owners and people who visits our website and attempt those surveys, to provide a menace free environment for all the visitors who visits and fill the feedback.

1.2. Objectives :

• To create a menace free space for users.
• To detect any inappropriate content.
• To reduce the distribution difference between source and target domains.
• To develop a user-friendly website.

1.3. Scope:

• It will reduce online hatred.
• It provides menace free platform.
• It will help reduce toxic content from being broadcasted

2. Problem Definition:

Many companies while starting their new projects or products find it difficult to gather data & spend huge capital to test their products.

So, coin planet provides a platform for new entrepreneurs & new business establishments to test their product on this platform.

On this platform through our website namely coin planet they can gather unbiased opinions & information through surveys & develop their resources accordingly.

So basically, we will be gathering huge loads of data from the customers who are sharing their thoughts and their ideas about that particular product/services and some users might use some inappropriate text or data while sharing those feedback.

The content we are receiving for our website should not contain any inappropriate text/content because of that our website engagement will be less so it is necessary for us to protect the other user’s views and to provide user a secured user space.

3. Proposed System:

This is where our project comes into picture by providing a menace free platform to every individual who visits and perform certain actions that can be survey owners who create surveys for their respective business/service or they can be user users who kindly fill the form.

There might be people who can misuse this and gets blocked by the system, their actions/feedback will not get further access.

3.1. Features & Functionality:

1. The ML model used here is created using Logistic Regression which helps us predict the probability as to how much extent is the text toxic

2. We have 6 categories here namely Toxic, Obscene, Threat, Insult, Severe Toxic, Identity

3. We are using the Model for these 6 categories separately & if any of the category results in a probability higher than 0.6 or 60% then that survey will not be published unless & until the text is changed & the user publishes the survey again.

4. What normally happens in Request-Response pattern is that the client waits for appropriate response from the server withing milliseconds. If it exceeds beyond that then a timeout occurs & an error is thrown.

5. As the Machine Learning models take time to predict & give response as well as we need something that can queue the requests & process them later without dropping that request.

6. For that we are going to use Message Queue namely RabbitMQ. This will help in queuing the messages & the top messages will be popped off the queue as soon as any one consumer among the N consumers becomes free & will consume that message.

7. The consumer is the one that contains the Machine Learning package & the Consumer package uses a function from that Machine Learning package to predict the results.

8. The consumer connects to the database and after authentication it can access the data related to that "surveyId" (which is contained in the message).

9. Using that "survey Id" it gathers all the text data related to that Survey from various tables like Surveys table itself, Questions table, Options table by performing joins & aggregates all the data.

10. Then using the ML function imported from the Machine Learning package is used to analyse the data

Logistic Regression :-

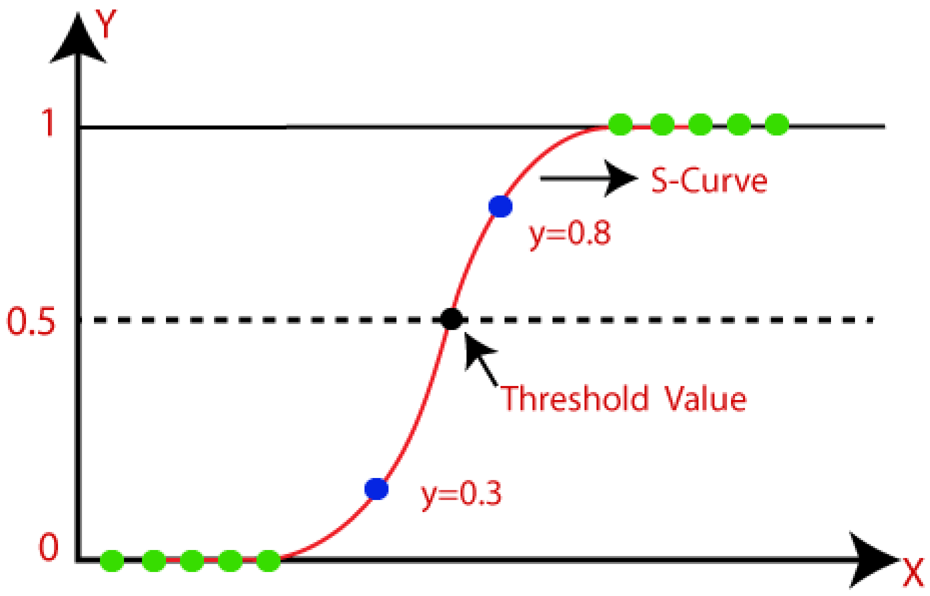


Fig 1 :- Logistic Regression

Logistic Regression is the process of modelling the probability of a discrete outcome given an input variable

The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on

Multinomial logistic regression can model scenarios where there are more than two possible discrete outcomes

Logistic regression is a useful analysis method for classification problems, where you are trying to determine if a new sample fits best into a category

We have 6 categories of data that are not allowed they are Toxic, Obscene, Threat, Insult, Severe Toxic, Identity

For each one of them we use logistic regression to see the probability that if some sample text has probability greater than 0.6 that is 60%

If yes then that text is marked as harmful & the survey is given a status of un-sanitized. Unless & until the survey is un-sanitized the survey cannot be published.

The user needs to publish the survey again after making changes to the survey's text content

A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures

Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer

Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads

There are various message queues available but one of the most prominent & promising message queues among them is RabbitMQ

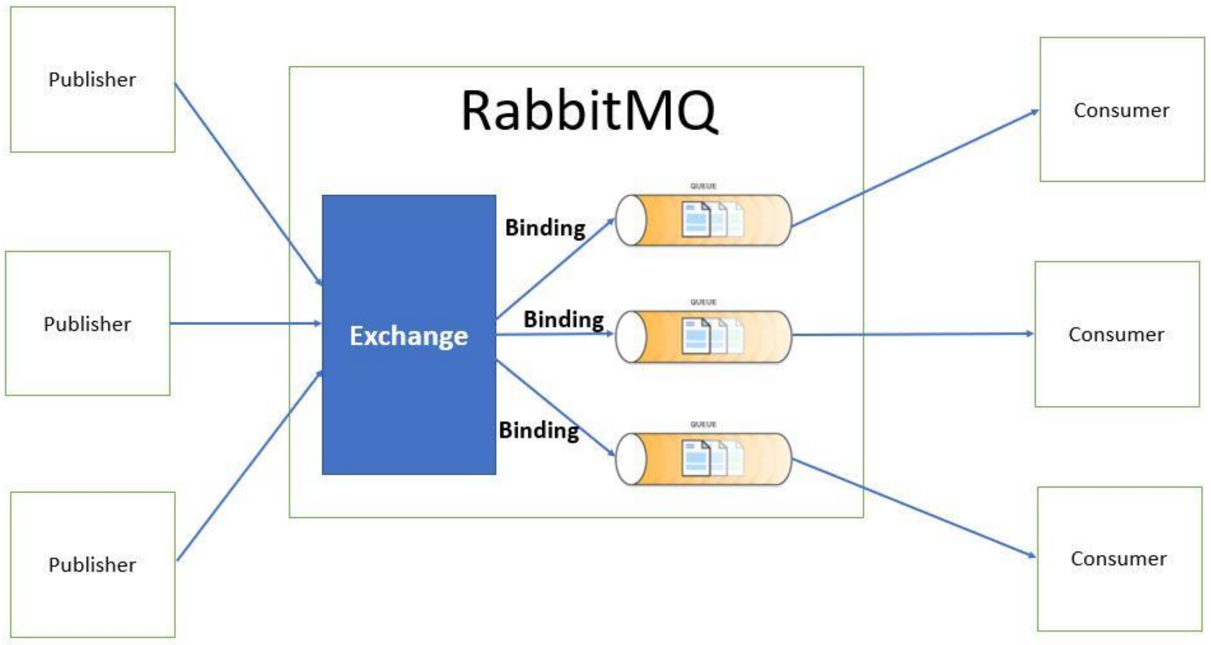


Fig 2 :- RabbitMQ Working

RabbitMQ is one of the most popular open-source message brokers. From T-Mobile to Runtastic, RabbitMQ is used worldwide at small start-ups and large enterprises.

RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements.

RabbitMQ runs on many operating systems and cloud environments, and provides a wide range of developer tools for most popular languages.

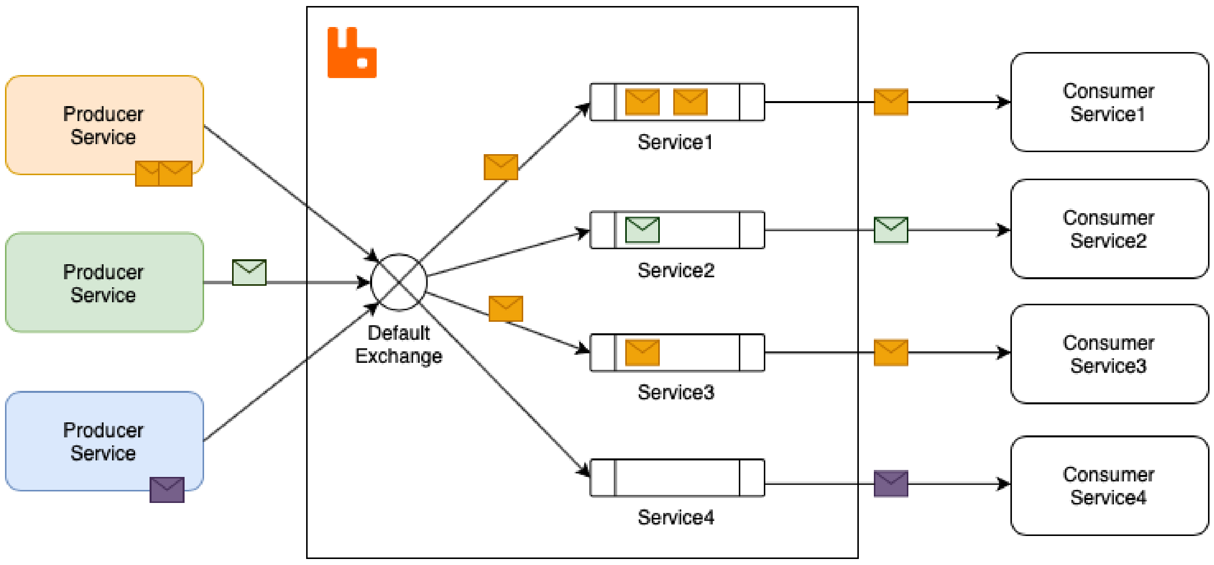
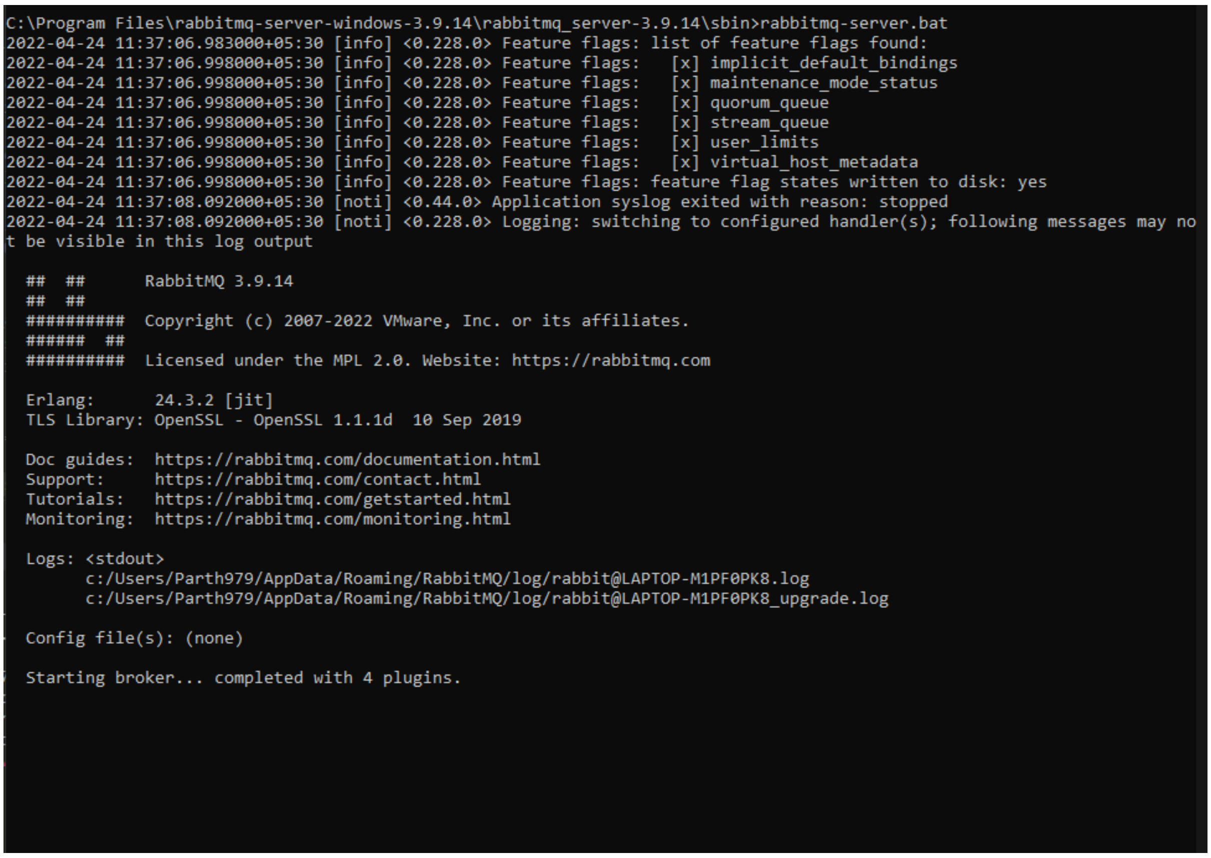


Fig 3 :- RabbitMQ Architecture

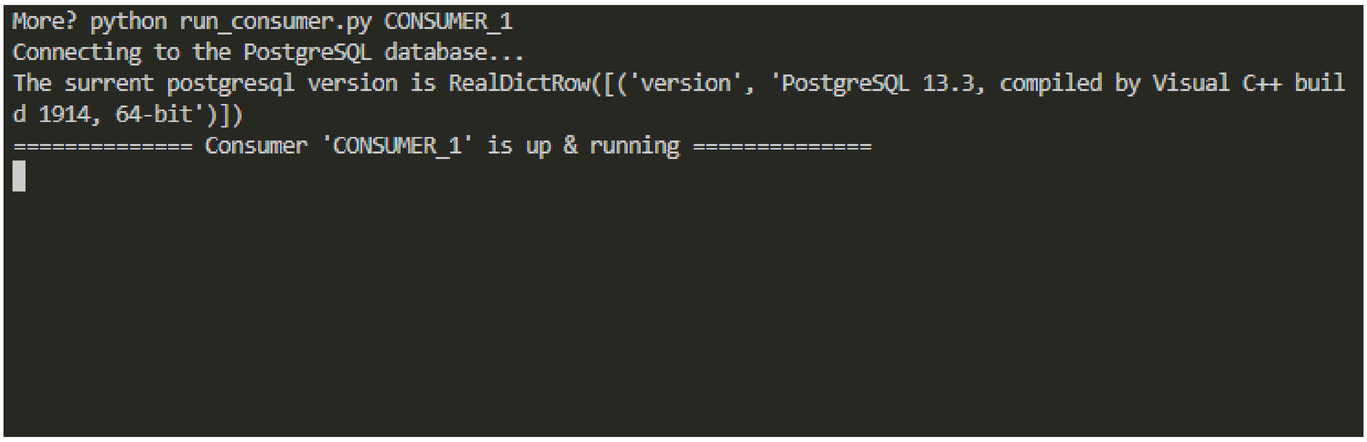
4. Project Outcomes



```
C:\Program Files\rabbitmq-server-windows-3.9.14\rabbitmq_server-3.9.14\sbin>rabbitmq-server.bat
2022-04-24 11:37:06.983000+05:30 [info] <0.228.0> Feature flags: list of feature flags found:
2022-04-24 11:37:06.998000+05:30 [info] <0.228.0> Feature flags:   [x] implicit_default_bindings
2022-04-24 11:37:06.998000+05:30 [info] <0.228.0> Feature flags:   [x] maintenance_mode_status
2022-04-24 11:37:06.998000+05:30 [info] <0.228.0> Feature flags:   [x] quorum_queue
2022-04-24 11:37:06.998000+05:30 [info] <0.228.0> Feature flags:   [x] stream_queue
2022-04-24 11:37:06.998000+05:30 [info] <0.228.0> Feature flags:   [x] user_limits
2022-04-24 11:37:06.998000+05:30 [info] <0.228.0> Feature flags:   [x] virtual_host_metadata
2022-04-24 11:37:06.998000+05:30 [info] <0.228.0> Feature flags: feature flag states written to disk: yes
2022-04-24 11:37:08.092000+05:30 [noti] <0.44.0> Application syslog exited with reason: stopped
2022-04-24 11:37:08.092000+05:30 [noti] <0.228.0> Logging: switching to configured handler(s); following messages may no
t be visible in this log output

  ##  ##      RabbitMQ 3.9.14
  ##  ##
  ##########  Copyright (c) 2007-2022 VMware, Inc. or its affiliates.
  ######  ##
  ##########  Licensed under the MPL 2.0. Website: https://rabbitmq.com

  Erlang:      24.3.2 [jit]
  TLS Library: OpenSSL - OpenSSL 1.1.1d  10 Sep 2019

  Doc guides:  https://rabbitmq.com/documentation.html
  Support:     https://rabbitmq.com/contact.html
  Tutorials:   https://rabbitmq.com/getstarted.html
  Monitoring:  https://rabbitmq.com/monitoring.html

  Logs: <stdout>
        c:/Users/Parth979/AppData/Roaming/RabbitMQ/log/rabbit@LAPTOP-M1PF0PK8.log
        c:/Users/Parth979/AppData/Roaming/RabbitMQ/log/rabbit@LAPTOP-M1PF0PK8_upgrade.log

  Config file(s): (none)

  Starting broker... completed with 4 plugins.
```

Fig 4 :- Starting the RabbitMQ Service

This is done in windows by going to the folder RabbitMQ is installed & then going to the sbin folder & then running the batch file

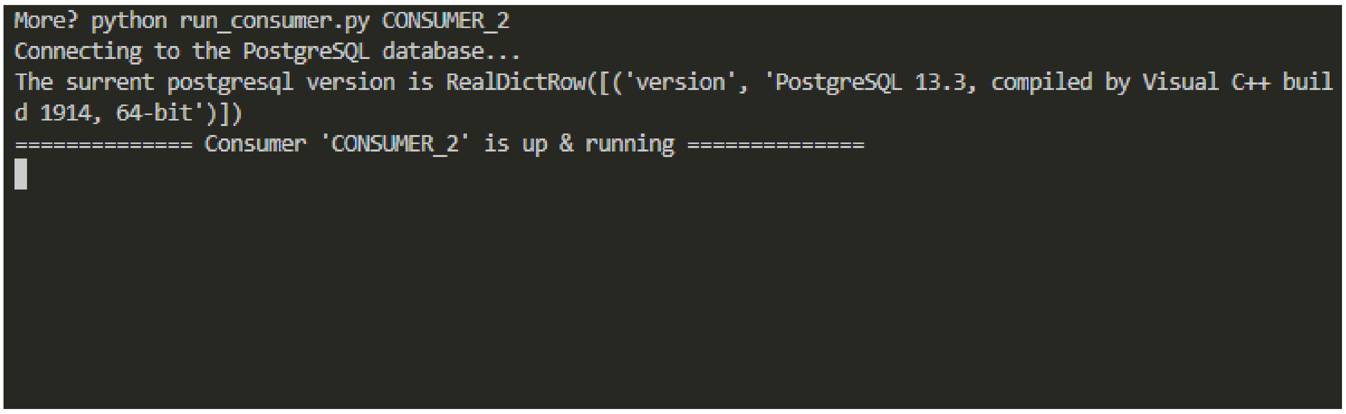


```
More? python run_consumer.py CONSUMER_1
Connecting to the PostgreSQL database...
The surrent postgresql version is RealDictRow([('version', 'PostgreSQL 13.3, compiled by Visual C++ buil
d 1914, 64-bit')])
============= Consumer 'CONSUMER_1' is up & running =============
```

Fig 5 :- Running the first Consumer

This is done by going to the consumer package written in python & then running the consumer from there.

We are giving it a name "CONSUMER_1" to identify the different consumers running. These consumer consume message from the message queue named "sanitize_survey_queue"



```
More? python run_consumer.py CONSUMER_2
Connecting to the PostgreSQL database...
The surrent postgresql version is RealDictRow([('version', 'PostgreSQL 13.3, compiled by Visual C++ buil
d 1914, 64-bit')])
============= Consumer 'CONSUMER_2' is up & running =============
```

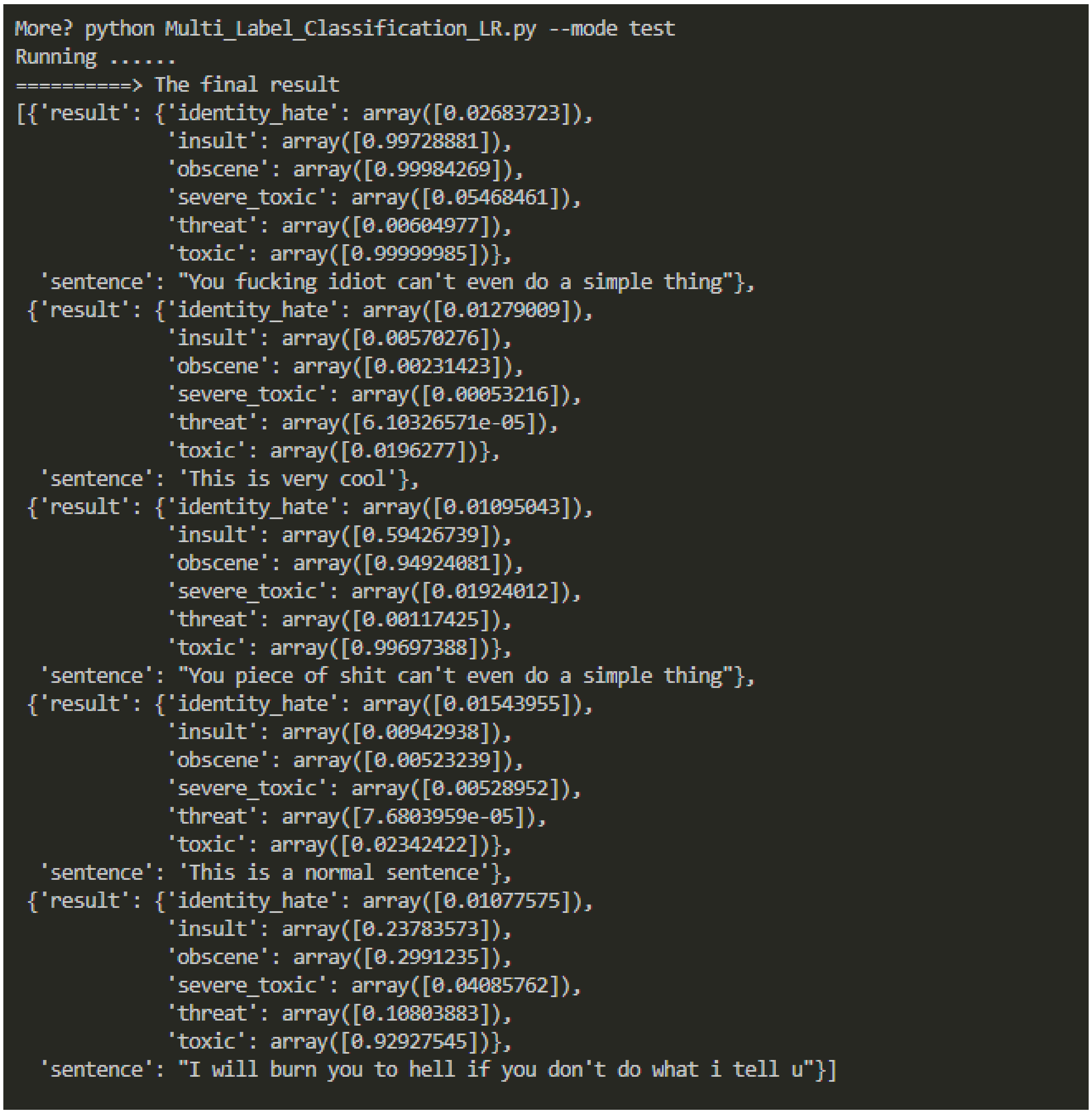
Fig 6 :- Running the second Consumer

This is done by going to the consumer package written in python & then running the consumer from there.

We are giving it a name "CONSUMER_2" to identify the different consumers running. These consumer consume message from the message queue named "sanitize_survey_queue"
The important thing why we are using 2 consumers is that so that more requests can be processed from the queue.

The more the number of consumers we can run the more number of messages from the consumed from the queue & processed.

They work in a competing consumer pattern where every consumer compete for the message in a Competing Consumer Fashion

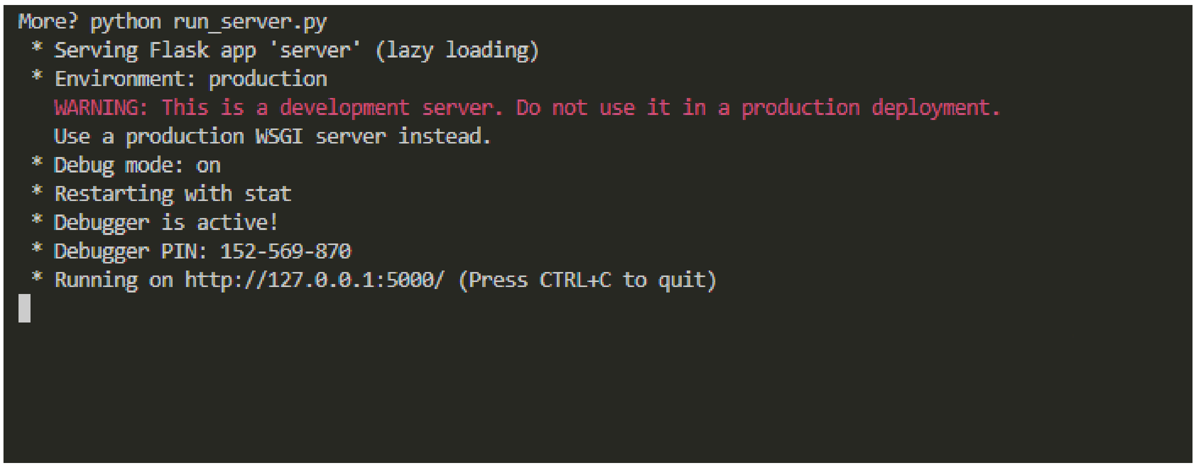


```
More? python Multi_Label_Classification_LR.py --mode test
Running ......
==========> The final result
[{'result': {'identity_hate': array([0.02683723]),
             'insult': array([0.99728881]),
             'obscene': array([0.99984269]),
             'severe_toxic': array([0.05468461]),
             'threat': array([0.00604977]),
             'toxic': array([0.99999985])},
  'sentence': "You fucking idiot can't even do a simple thing"},
 {'result': {'identity_hate': array([0.01279009]),
             'insult': array([0.00570276]),
             'obscene': array([0.00231423]),
             'severe_toxic': array([0.00053216]),
             'threat': array([6.10326571e-05]),
             'toxic': array([0.0196277])},
  'sentence': 'This is very cool'},
 {'result': {'identity_hate': array([0.01095043]),
             'insult': array([0.59426739]),
             'obscene': array([0.94924081]),
             'severe_toxic': array([0.01924012]),
             'threat': array([0.00117425]),
             'toxic': array([0.99697388])},
  'sentence': "You piece of shit can't even do a simple thing"},
 {'result': {'identity_hate': array([0.01543955]),
             'insult': array([0.00942938]),
             'obscene': array([0.00523239]),
             'severe_toxic': array([0.00528952]),
             'threat': array([7.6803959e-05]),
             'toxic': array([0.02342422])},
  'sentence': 'This is a normal sentence'},
 {'result': {'identity_hate': array([0.01077575]),
             'insult': array([0.23783573]),
             'obscene': array([0.2991235]),
             'severe_toxic': array([0.04085762]),
             'threat': array([0.10803883]),
             'toxic': array([0.92927545])},
  'sentence': "I will burn you to hell if you don't do what i tell u"}]
```

Fig 7 :- Testing the Model Seperately

As we can see above that 5 sample sentences are given to the model to predict the category.

We can give the mode option as “test” or “train” to it so that it runs in a standalone manner & we can train or test the model accordingly



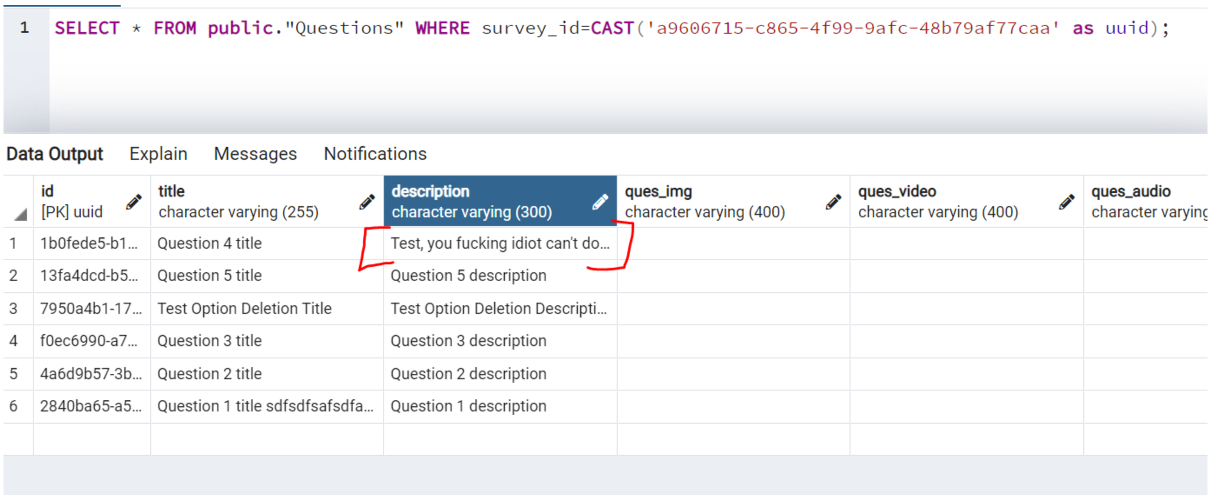
```
More? python run_server.py
 * Serving Flask app 'server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 152-569-870
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Fig 8 :-_Running the Flask Server

The flask server contains the Producer code which can be seen in the Project Design Diagram as well.

When a request is sent to the flask server that request is then forwarded to the exchange by the flask server.

That particular route contains the producer code which is responsible to sent the message (request) to the exchange



```
SELECT * FROM public."Questions" WHERE survey_id=CAST('a9606715-c865-4f99-9afc-48b79af77caa' as uuid);
```

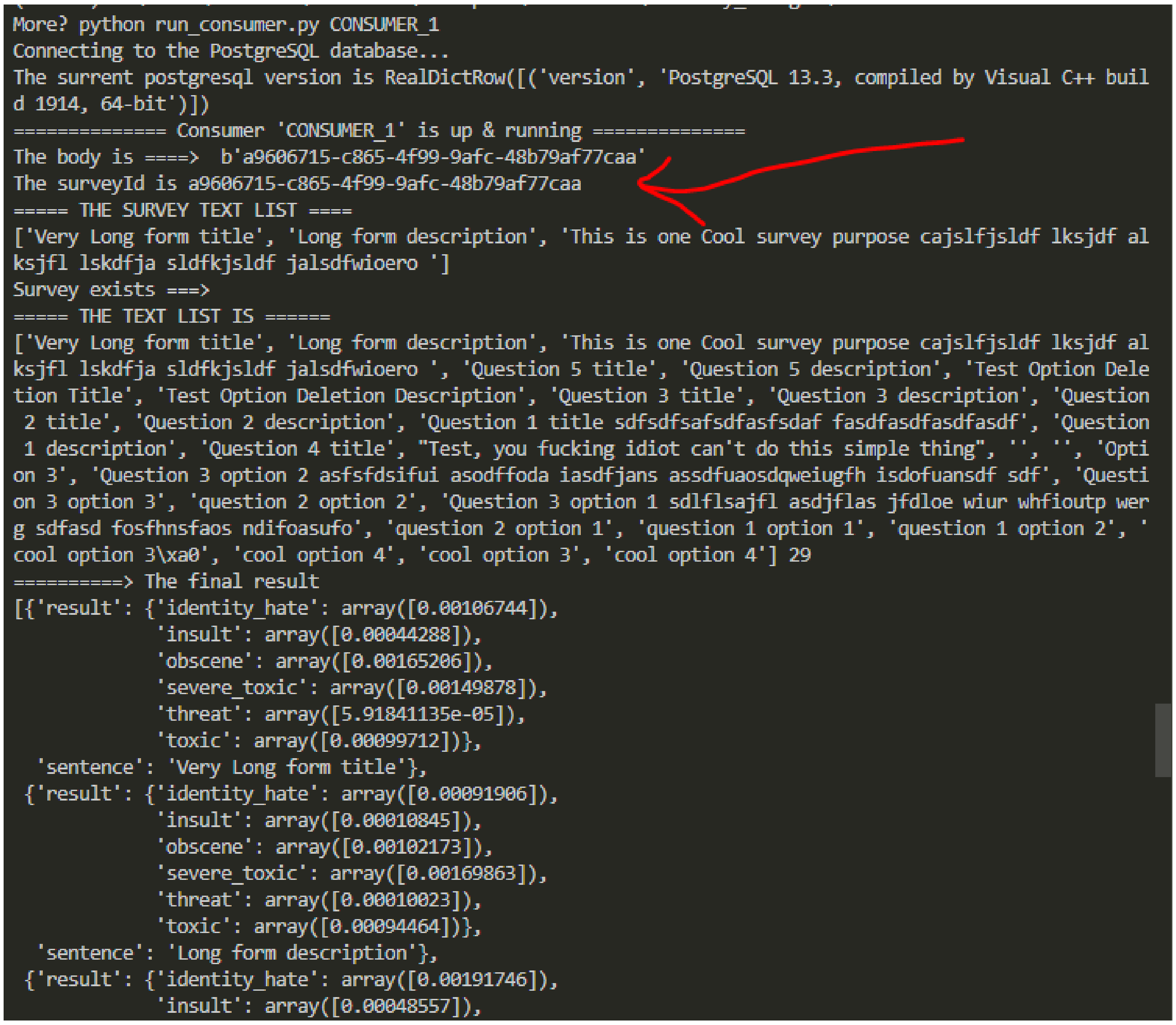
	id [PK] uuid	title character varying (255)	description character varying (300)	ques_img character varying (400)	ques_video character varying (400)	ques_audio character varying
1	1b0fede5-b1...	Question 4 title	Test, you fucking idiot can't do...			
2	13fa4dcd-b5...	Question 5 title	Question 5 description			
3	7950a4b1-17...	Test Option Deletion Title	Test Option Deletion Descripti...			
4	f0ec6990-a7...	Question 3 title	Question 3 description			
5	4a6d9b57-3b...	Question 2 title	Question 2 description			
6	2840ba65-a5...	Question 1 title sdfsdfsafsdfa...	Question 1 description			

Fig 9 :-_Toxic data in the database.

The following toxic data can be seen in the database for the survey table with id

This id is sent to the flask server. The flask server then forwards this id as a message to the exchange with the help of producer functionality present in it

Let's see if the model can identify this toxic text



```
More? python run_consumer.py CONSUMER_1
Connecting to the PostgreSQL database...
The surrent postgresql version is RealDictRow([('version', 'PostgreSQL 13.3, compiled by Visual C++ buil
d 1914, 64-bit')])
============= Consumer 'CONSUMER_1' is up & running =============
The body is ===>  b'a9606715-c865-4f99-9afc-48b79af77caa'
The surveyId is a9606715-c865-4f99-9afc-48b79af77caa
===== THE SURVEY TEXT LIST ====
['Very Long form title', 'Long form description', 'This is one Cool survey purpose cajslfjsldf lksjdf al
ksjfl lskdfja sldfkjsldf jalsdfwioero ']
Survey exists ===>
===== THE TEXT LIST IS ======
['Very Long form title', 'Long form description', 'This is one Cool survey purpose cajslfjsldf lksjdf al
ksjfl lskdfja sldfkjsldf jalsdfwioero ', 'Question 5 title', 'Question 5 description', 'Test Option Dele
tion Title', 'Test Option Deletion Description', 'Question 3 title', 'Question 3 description', 'Question
 2 title', 'Question 2 description', 'Question 1 title sdfsdfsafsdfasfsdaf fasdfasdfasdfasdf', 'Question
 1 description', 'Question 4 title', "Test, you fucking idiot can't do this simple thing", '', '', 'Opti
on 3', 'Question 3 option 2 asfsfdsifui asodffoda iasdfjans assdfuaosdqweiugfh isdofuansdf sdf', 'Questi
on 3 option 3', 'question 2 option 2', 'Question 3 option 1 sdlflsajfl asdjflas jfdloe wiur whfioutp wer
g sdfasd fosfhnsfaos ndifoasufo', 'question 2 option 1', 'question 1 option 1', 'question 1 option 2', '
cool option 3\xa0', 'cool option 4', 'cool option 3', 'cool option 4'] 29
==========> The final result
[{'result': {'identity_hate': array([0.00106744]),
             'insult': array([0.00044288]),
             'obscene': array([0.00165206]),
             'severe_toxic': array([0.00149878]),
             'threat': array([5.91841135e-05]),
             'toxic': array([0.00099712])},
  'sentence': 'Very Long form title'},
 {'result': {'identity_hate': array([0.00091906]),
             'insult': array([0.00010845]),
             'obscene': array([0.00102173]),
             'severe_toxic': array([0.00169863]),
             'threat': array([0.00010023]),
             'toxic': array([0.00094464])},
  'sentence': 'Long form description'},
 {'result': {'identity_hate': array([0.00191746]),
             'insult': array([0.00048557]),
```

Fig 10 :-Request is received by the consumer and it starts processing

The message is consumed by the consumer named "CONSUMER_1" we can see the surveyId which is the message sent to the exchange & eventually to the queue by the producer.

This is processed by the consumer accordingly running various operations

The functionality of the Consumer :-

The consumer performs various important tasks they are the following

1. Getting the data from the database by running queries against it for that specific surveyId

2. It's second important task is to fetch all the text content from the data for analysis purposes

```
cur.execute("SELECT * FROM public.\"Questions\" WHERE survey_id=CAST(%s as uuid);", 
(surveyId,))


      questions = cur.fetchall()
      # get the "id", "title" & "description"


        query = f'SELECT * FROM public.\"Options\" WHERE ({" OR 
".join(list(map(string_from_qid, question_ids)))})'
```

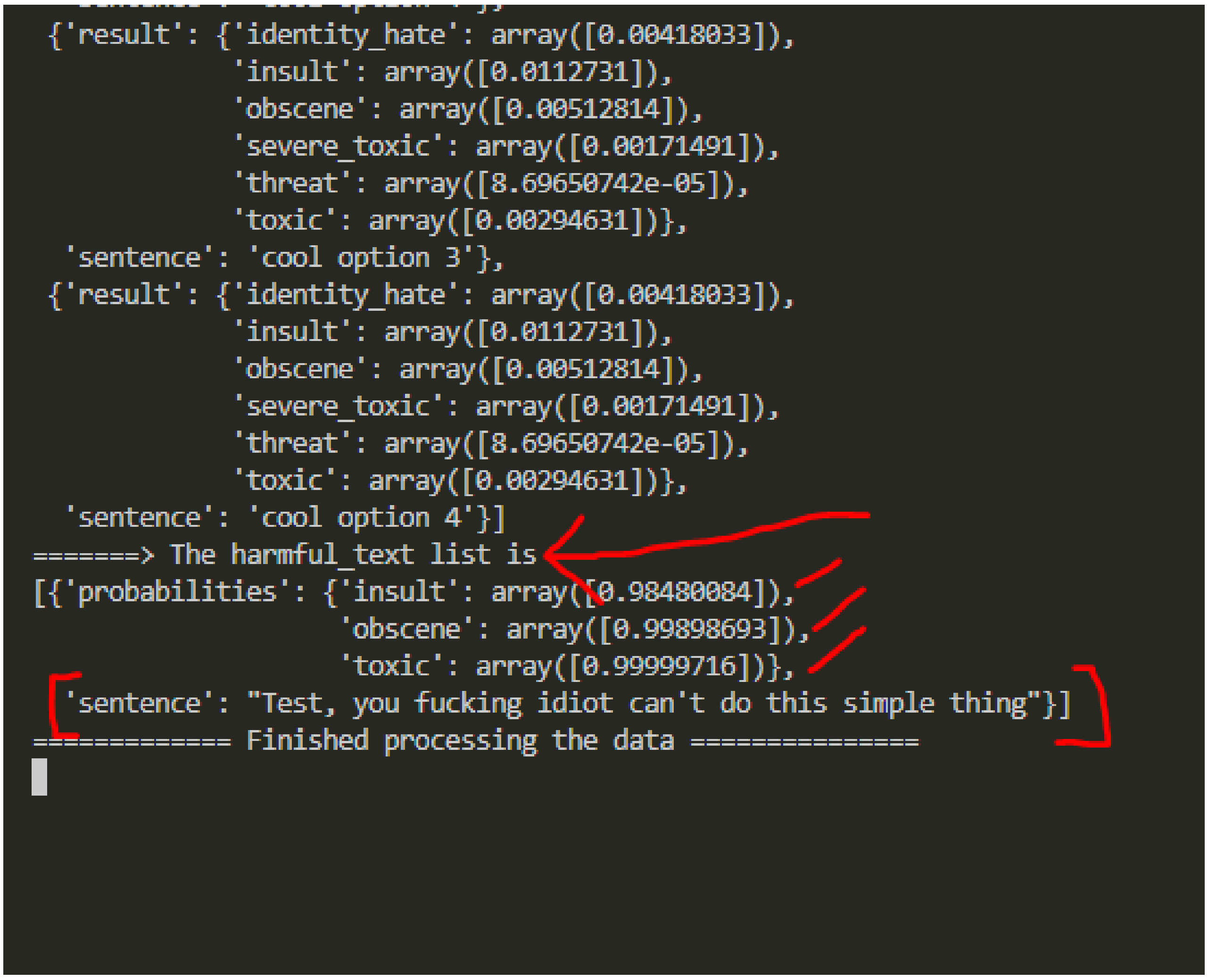
3. After extracting all the text content it then groups them for further use

```
if survey is not None:
      print("Survey exists ===>")

      cur.execute("SELECT * FROM public.\"Questions\" WHERE survey_id=CAST(%s as 
uuid);", (surveyId,))
      # SELECT * FROM public."Questions" WHERE survey_id=CAST('a9606715-c865-4f99-
9afc-48b79af77caa' as uuid);

      questions = cur.fetchall()
      # get the "id", "title" & "description"

      question_ids = []

      if len(questions) > 0:
        for question in questions:

          question_ids.append(question['id'])

          text_list.append(question['title'])
          text_list.append(question['description'])
        query = f'SELECT * FROM public.\"Options\" WHERE ({" OR
".join(list(map(string_from_qid, question_ids)))})'

        cur.execute(query)
        # SELECT * FROM public."Options" WHERE (question_id=CAST('1b0fede5-b1ab-
427e-9a98-d135c5c6a4db' as uuid) OR question_id=CAST('13fa4dcd-b514-483e-b699-
ba9efb17a836' as uuid) OR question_id=CAST('7950a4b1-175f-466f-a826-81bf51dd781e' as
uuid) OR question_id=CAST('f0ec6990-a71e-425b-8690-cc621f072293' as uuid) OR
question_id=CAST('4a6d9b57-3bf1-4267-bedc-99537fd677ff' as uuid) OR
question_id=CAST('2840ba65-a591-46c0-8558-908c823b4bd7' as uuid));

        all_options = cur.fetchall()
        # get the "option_text"

        if len(all_options) > 0:
          for option in all_options:
            text_list.append(option['option_text'])
        else:
          print("CONSUMER ERROR :- Cannot add survey to the Sanitization Queue bcz
Question doesn't have any options")

        print("===== THE TEXT LIST IS ======")

        print(text_list, len(text_list))
```

4. Finally the Machine Learning Models are used & then the analysis is done as to how extent the text is toxic

```
data_sanitization_mapping = []

        # RUN THE ML CODE HERE
        for i in range(len(text_list)):
            # analyzer = LR_analyze_data()
            result = analyze_data(text_list[i])
            data_sanitization_mapping.append({
                'sentence' : text_list[i],
                'result': result
            })

        print("==========> The final result")
        pprint(data_sanitization_mapping)

        # HARMFUL TEXT
        harmful_text = []
        for sent_dict in data_sanitization_mapping:
          harmful = False
          harmful_list = {}
          for key in sent_dict["result"]:
            if sent_dict["result"][key] > 0.6:
              harmful_list[key] = sent_dict["result"][key]
              harmful = True
          if harmful:
            harmful_text.append({"sentence": sent_dict["sentence"], "probabilities":
harmful_list})

        print("=======> The harmful_text list is")
        pprint(harmful_text)
```

After processing we get the result the sentence that is toxic & the result :-



```
 {'result': {'identity_hate': array([0.00418033]),
             'insult': array([0.0112731]),
             'obscene': array([0.00512814]),
             'severe_toxic': array([0.00171491]),
             'threat': array([8.69650742e-05]),
             'toxic': array([0.00294631])},
  'sentence': 'cool option 3'},
 {'result': {'identity_hate': array([0.00418033]),
             'insult': array([0.0112731]),
             'obscene': array([0.00512814]),
             'severe_toxic': array([0.00171491]),
             'threat': array([8.69650742e-05]),
             'toxic': array([0.00294631])},
  'sentence': 'cool option 4'}]
=======> The harmful_text list is
[{'probabilities': {'insult': array([0.98480084]),
                    'obscene': array([0.99898693]),
                    'toxic': array([0.99999716])},
  'sentence': "Test, you fucking idiot can't do this simple thing"}]
============= Finished processing the data ===============
```

Fig 12 :- The final result

As it can be seen in the above database that the toxic text is “Test, you fucking idiot can’t do this simple thing”

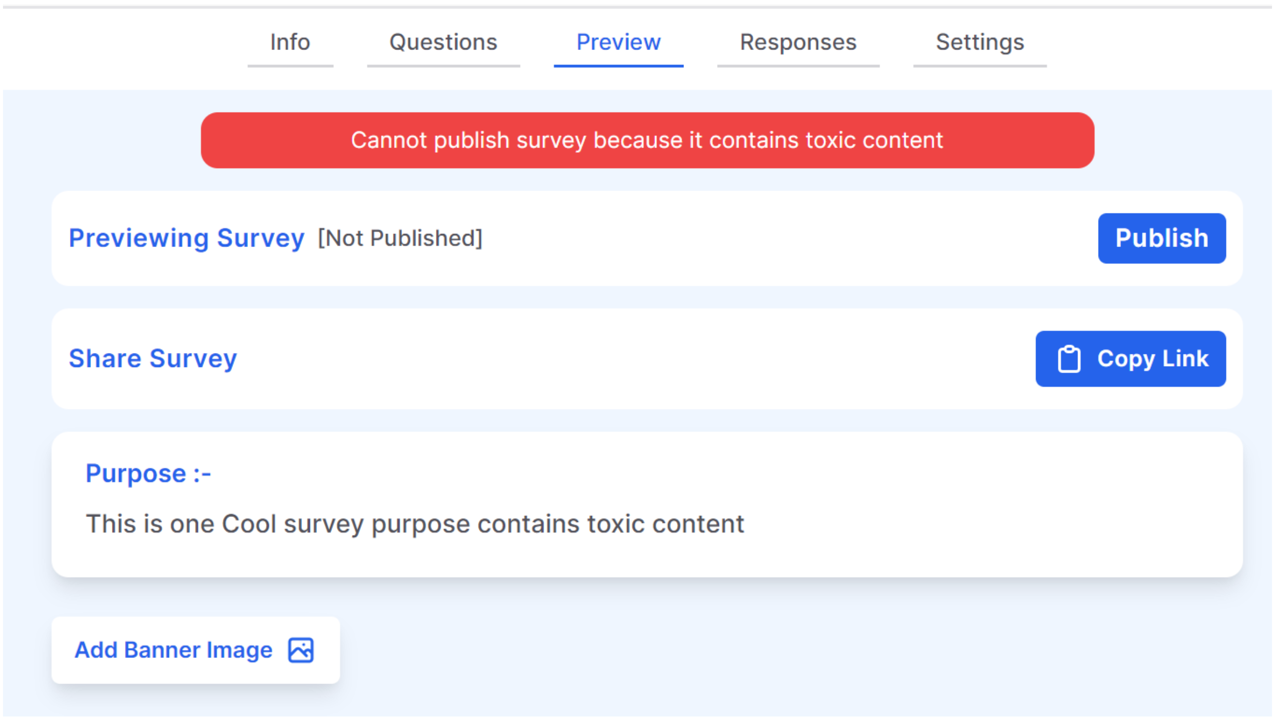
The predicted toxic categories to which it belong can be seen below. The categories are “insult”, “obscene” and “toxic” with the following probabilities.

The request for processing the text :-

The request for checking toxic content is sent when the user publishes the survey by clicking on the "Publish" button.

When the user clicks on the Publish button the request is sent to the Flask server which is responsible for forwarding the "surveyId" which is sent by the Publish request

In our case when we publish the survey we get the following



Cannot publish survey because it contains toxic content

Previewing Survey [Not Published]

Share Survey

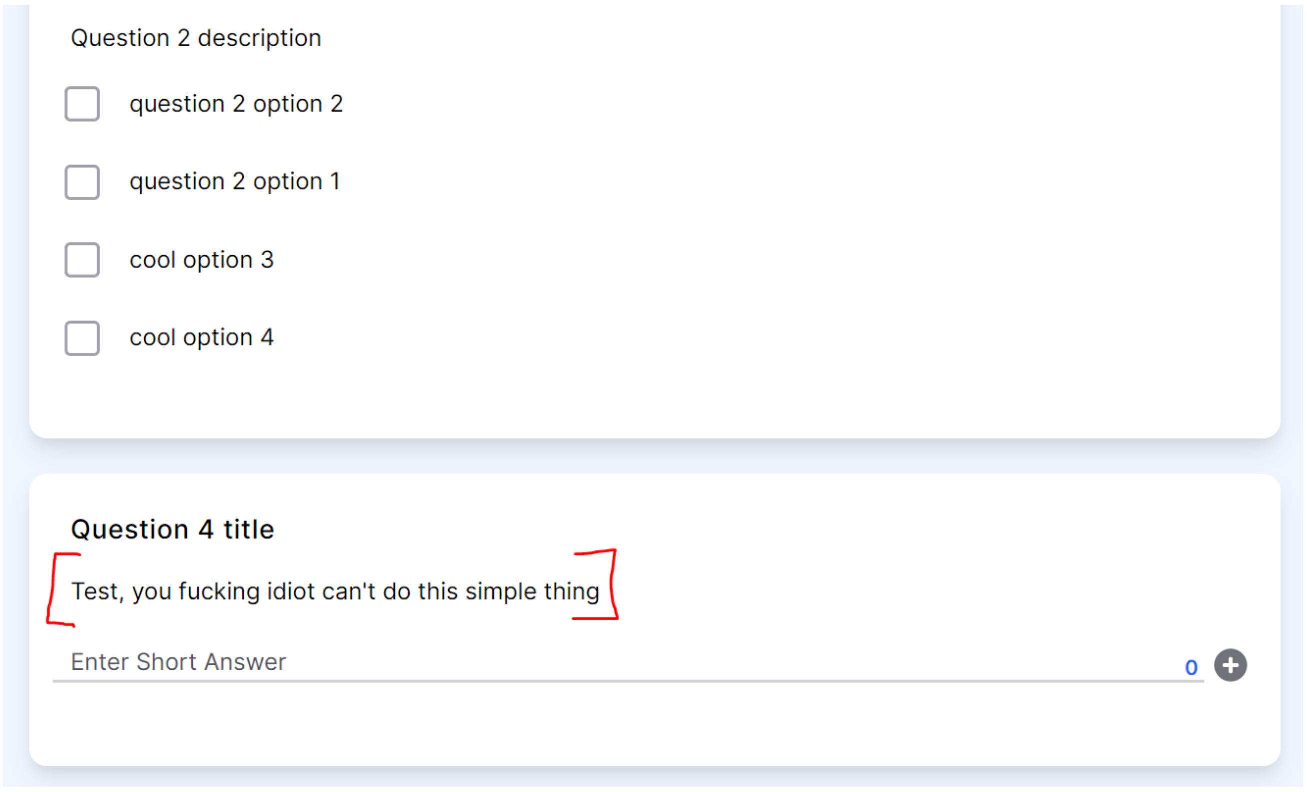
Purpose :-

This is one Cool survey purpose contains toxic content

Fig 13 :- Showing the message on survey page

The following is shown because there is toxic content in this survey which can be seen in the database above

Location of the actual toxic content can be seen in the following image where the toxic content is located



Question 2 description

- [ ] question 2 option 2
- [ ] question 2 option 1
- [ ] cool option 3
- [ ] cool option 4

Question 4 title

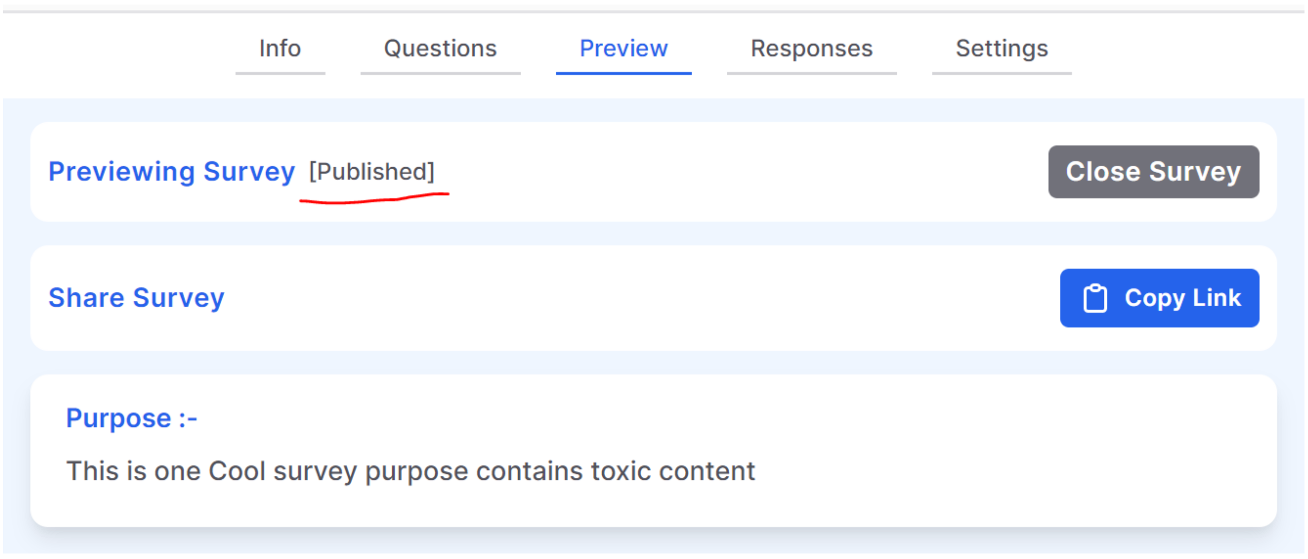
Test, you fucking idiot can't do this simple thing

Enter Short Answer 0

Fig 14 :- Location of the actual toxic content

We can refer to the Figure 12 & Figure 14 to see that the survey is marked as toxic when it contains toxic content

If the survey doesn’t contain any toxic content when publishing then the message which can be seen in Red would not be visible & the survey can be published. Otherwise the user is not allowed to publish the survey



Previewing Survey [Published]

Share Survey

Purpose :-

This is one Cool survey purpose contains toxic content

Fig 17 :- Normal interface

It can be seen in the above interface when the survey is sanitized the survey can be published successfully without getting the Red Toxic Content Message.

The message for Closing the survey is also present which indicates message has been published successfully & a field a named “toxic” is set to False because the toxic content has been removed & it is safe to publish

5. Software Requirements

React :- For creating the frontend

Nodejs :- Runtime environment for the server

Express :- Creating lightweight servers using Nodejs

Postgresql DB :- Database for storing the operational data

RabbitMQ :- Message Queues for managing the AI requests smoothly, in an ordered manner without dropping them

Flask :- Server written in python for handling AI Requests & producing messages that are sent to message queues

6. Project Design

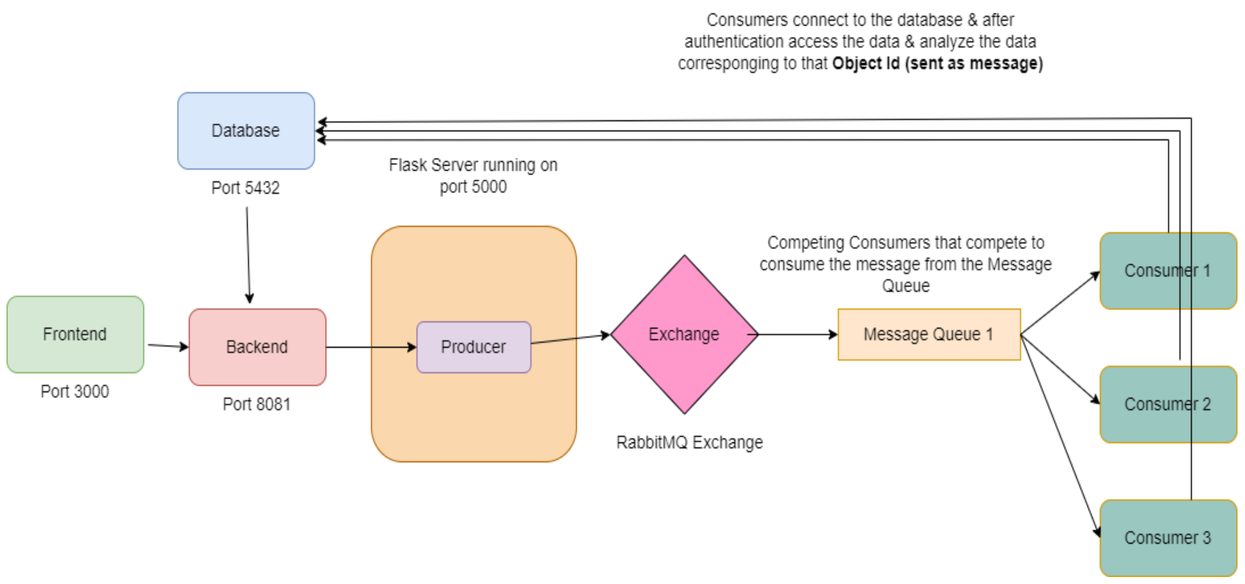

Consumers connect to the database & after authentication access the data & analyze the data corresponging to that **Object Id (sent as message)**
Database
Port 5432
Flask Server running on port 5000
Competing Consumers that compete to consume the message from the Message Queue
Frontend
Port 3000
Backend
Port 8081
Producer
Exchange
RabbitMQ Exchange
Message Queue 1
Consumer 1
Consumer 2
Consumer 3

Fig 18 :-_Project Working Design

The flow starts from the Frontend like any normal request-response model.

The Manage Survey Page has a section called Preview where one can see the state of the survey if is published or not. If it is to be published then the User needs to click on Publish Button on the UI. On clicking the Publish button the request is sent to the backend

The backend processes the request to see if it is valid or not using the Operation data from the database. If it not valid then it is rejected & the flow stops there itself.

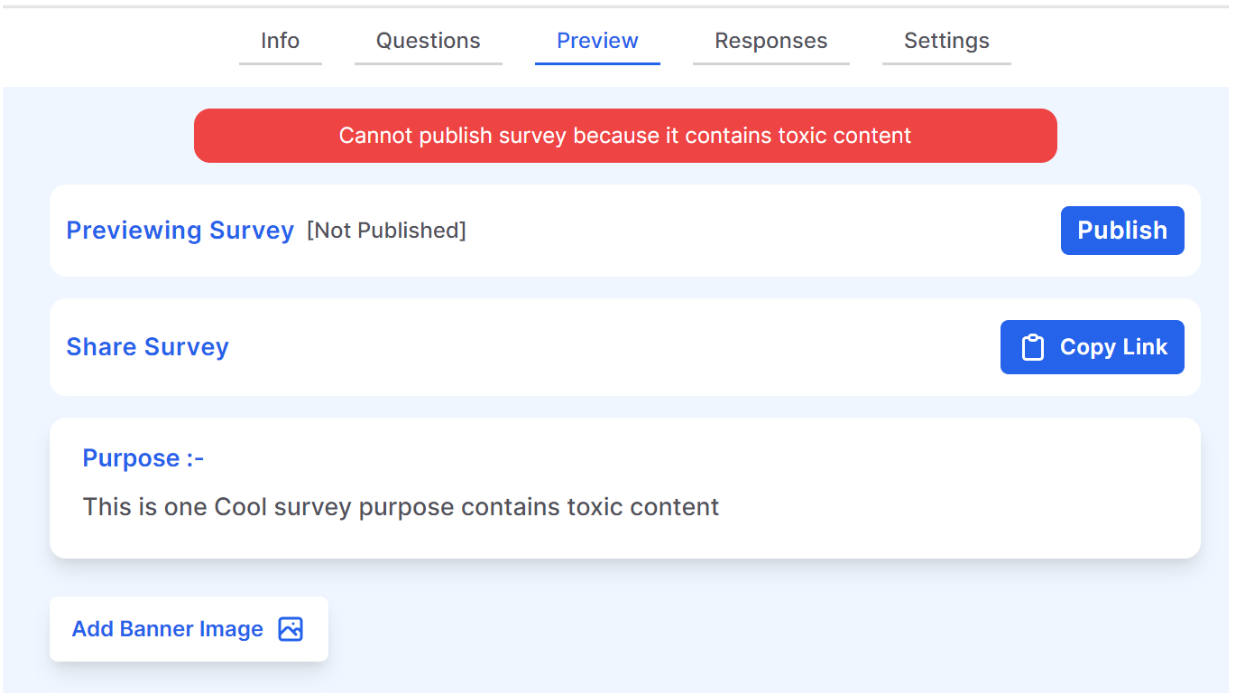
However if it is valid then the request is sent to the Flask Server running on Port 5000 where the Producer code resides. On hitting that particular url with the surveyId as the body of the message the message is the forwarded by the Producer to the Exchange

The Exchange is the main box where the routing of messages is present as defined by the user. It decides to which message queue the message should be sent. Here only 1 message is present so that message is sent to that queue

Here 3 consumers are present which behave in a competing consumer pattern to pop out the message from the message queue.

The consumers contain the actual code for running the ML Models & do the required prediction to see if the text data is toxic or not. If some of the text is toxic then the survey is marked as toxic & therefore the survey cannot be published

7. Screenshots of the Application



Cannot publish survey because it contains toxic content

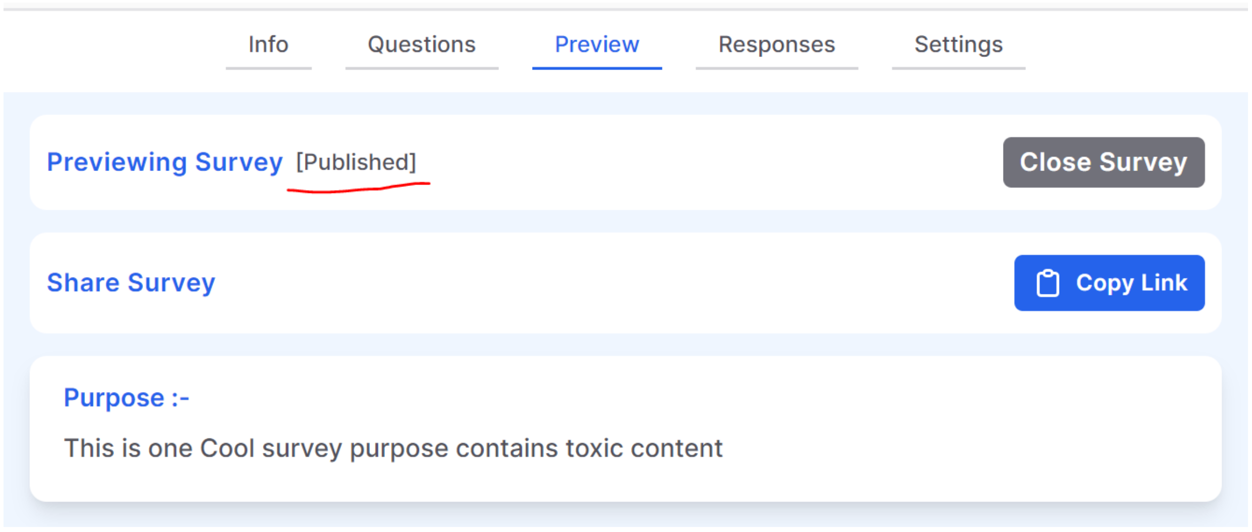
Previewing Survey [Not Published]

Share Survey

Purpose :-

This is one Cool survey purpose contains toxic content

Fig 19 :- Screenshot of GUI



Previewing Survey [Published]

Share Survey

Purpose :-

This is one Cool survey purpose contains toxic content

Fig 20 :- Screenshot of GUI

8. Project Scheduling

Sr No	Project Member	Time Required	Work Done
1	Parth Bhoir	Jan 1 – 15	Create the AI Model
		Jan 7-15 & Jan 15-20	Functionality to use the model In an optimal manner
		Jan 20 – Feb 15	Create the Consumer for Message Queues
		Feb 1 – Feb 17	Functionality to fetch data from the Database & use AI Models to predict
		Feb 20 – Feb 30	Manually test the flow
2	Pranav Mayekar	Mar 1 – 15	Read article & papers for Integration
		Mar 15 – 20	Plan the integration
		Mar 15 – 17	Plan the UI changes
		Mar 17 – 30	Integrate AI processing with the platform
3	Anjali Singh	Apr 1 – 10	Work on the frontend
		Apr 10 – 20	Work on the message queue for integration
		Apr 20 – 30	Create a Flask Server which encapsulates Message Queue's producer

9. Conclusion :-

During the course of this project, we were able to achieve text content sanitization & exclude inappropriate text & data for our website environment by using logistic regression. So, we hope that in the future this will help to reduce toxic & inappropriate content for text on English Language for now

10. References

[1] Robertson C, Mele C, Tavernise S. 11 Killed in Synagogue Massacre; Suspect Charged With 29 Counts. 2018;.

[2] Hate Speech ABA Legal Fact Check-American Bar Association;. Available from: https://abalegalfactcheck.com/articles/hate-speech.html.

[3] Ross B, Rist M, Carbonell G, Cabrera B, Kurowsky N, Wojatzki M. Measuring the Reliability of Hate Speech Annotations: The Case of the European Refugee Crisis. In: The 3rd Workshop on Natural Language Processing for Computer-Mediated Communication @ Conference on Natural Language Processing; 2016

[4] Pedregosa F, Varoquaux G, Gramfort A. Michel V, Thirion B, Grisel O, et al. Scikit- learn: Machine Learning in Python. JMLR. 2011:12:2825-2830.

[S] Opitz D, Maclin R. Popular ensemble methods: An empirical study. Journal of artificial intelligence research. 1999;11:169-198.