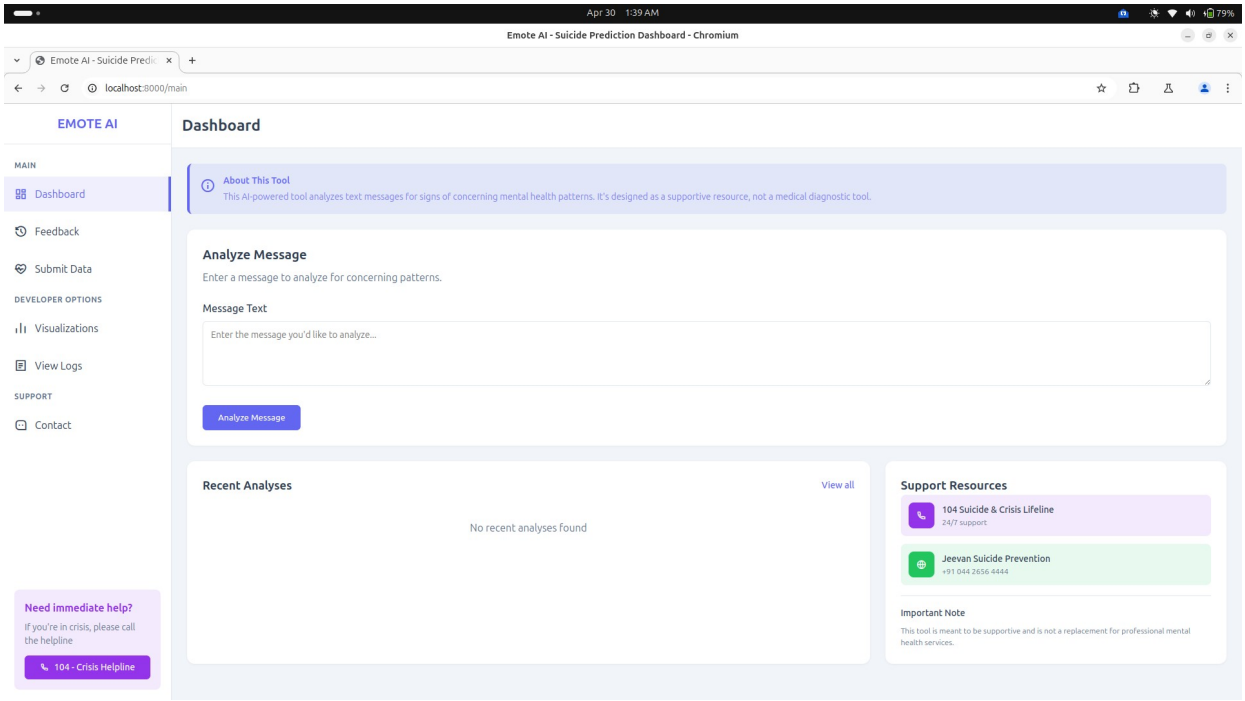
DA5402 – Machine Learning Operations Lab (AI APP)

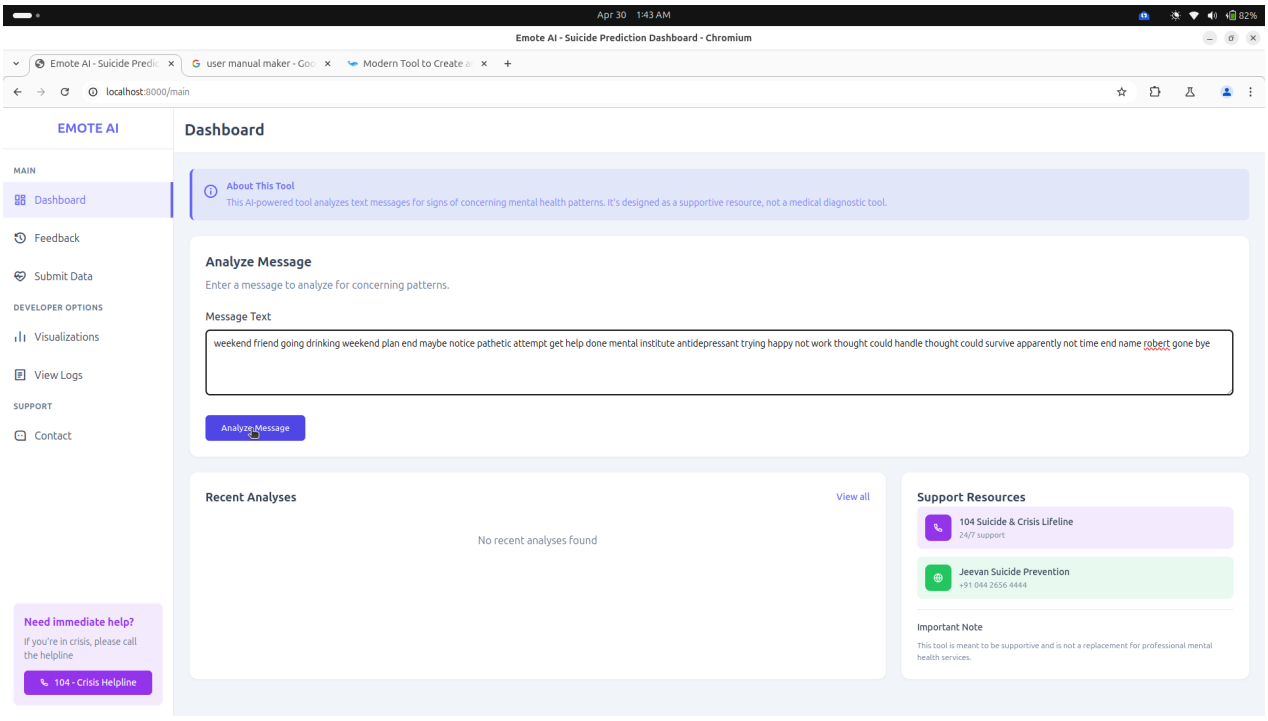
EMOTE AI (User Manual)

Main Dashboard

1) Navigate to the url that takes to the web application of the app. Here it is http://localhost:8000/ as 8000 is the port exposed in the host laptop (hosting the application). From other devices navigate to the url http://<host_laptop_ip>:8000/ to have a glance at the main dashboard of the app.



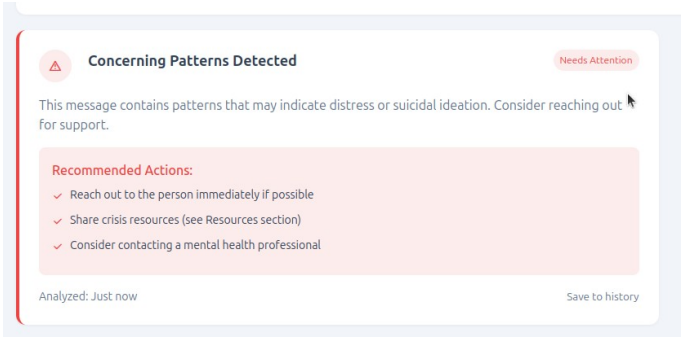
This how the dashboard looks like, now to analyze a text message (suicidal or not), you will have to type/copy-paste a message into the box named as “Message Text” and click on the “Analyze Message” button. An example of which is attached below,

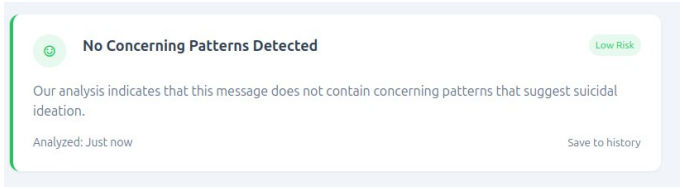


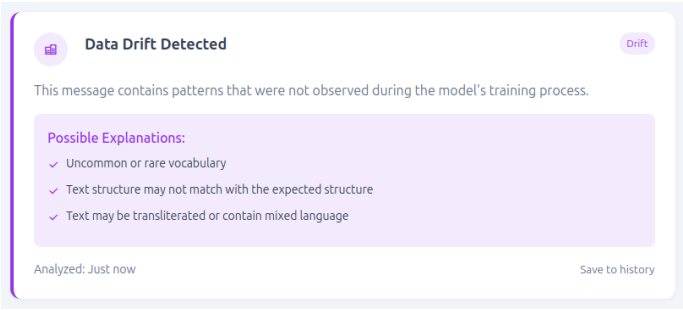
Once you click “Analyze Message” there are four outputs you might get based upon the message,

- Concerning patterns detected (High risk of suicide)
- Depressing patterns detected (Medium risk of suicide)
- No concerning patterns detected (Low risk of suicide)
- Data Drift detected (When the message is not similar to the ones that it is trained for)

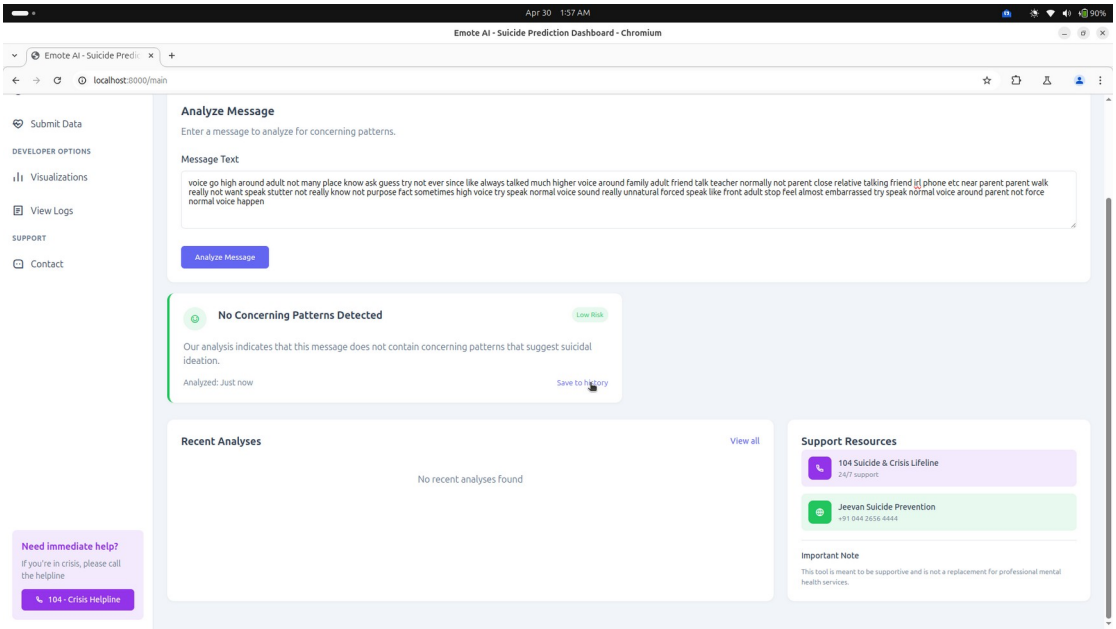
The different type of output screenshots are attached below



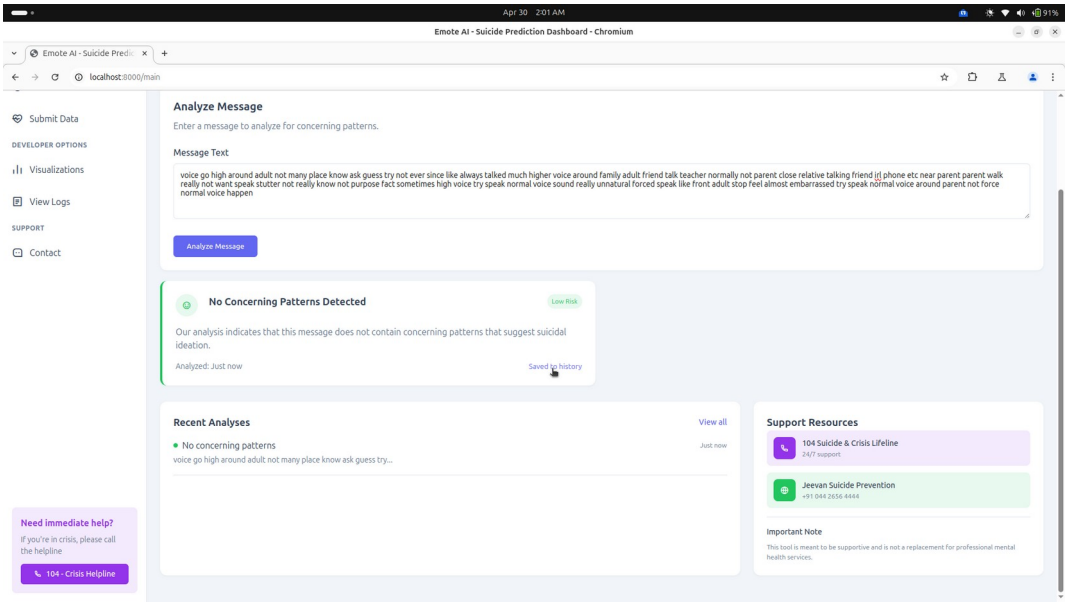




Now there is a save to history button that simply saves as a list which is lost when the page is refreshed. It is only for reference for the user.



the output when the save to history is clicked is given below,



API Endpoints

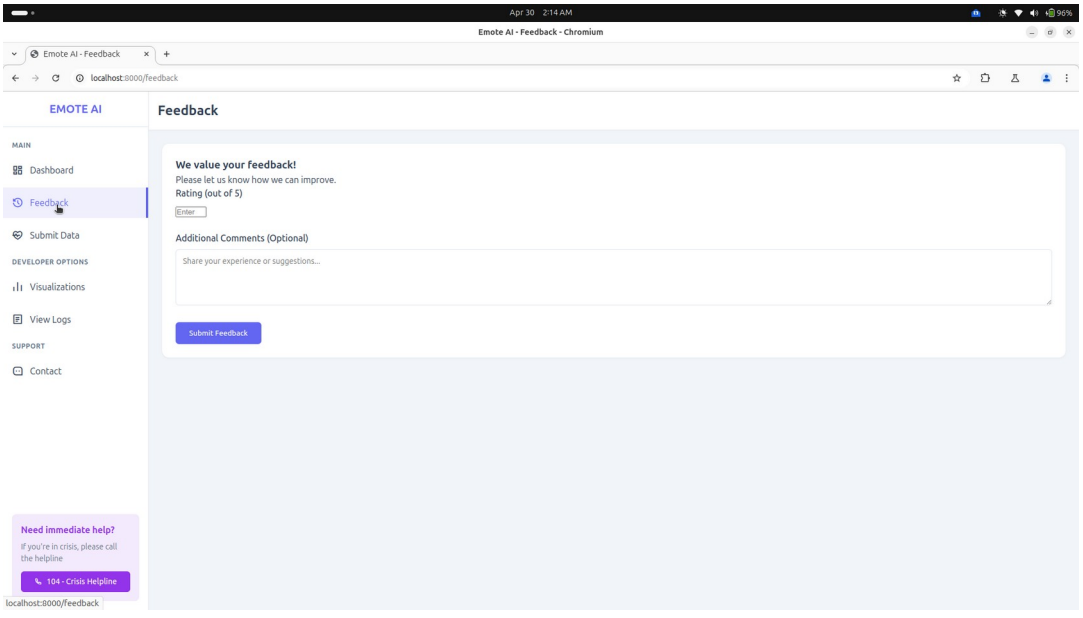
Note that only the front-end port is exposed to the localhost of the hosting machine and the ip of the localhost can be used to access the web application. The back-end is only accessible through the docker networks. So for this I have used API routing, i.e. when the “Analyze Message” button is clicked it fetches the http://localhost:8000/backend/predict URL with the post request containing the text message in the front-end Fast-API. This API request is router back to the back-end API http://backend:4000/predict where back-end is the name of the docker container for back-end functional and 4000 is the port exposed to the docker container networks.

This post request is processed by the back-end and the response is sent to the front-end where the front-end forwards it to the JavaScript where fetch was called and the corresponding output is displayed.

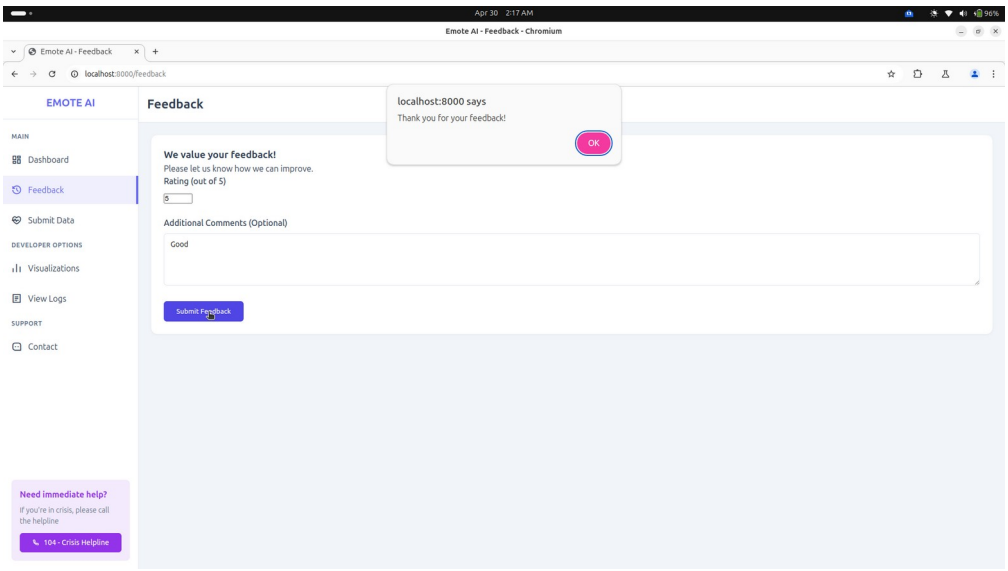
Request body : { “text” : message }

Feedback page

To access the feedback page, click on the “Feedback” sidebar.



In this page you may give feedback to improve the services provided by our app. Fill in a number from 1-5 whole numbers and an optional text message and click on submit feedback. On successful submission you will see a prompt like this



On clicking “OK” the page is refreshed and you may give as many feedback as you wish !!

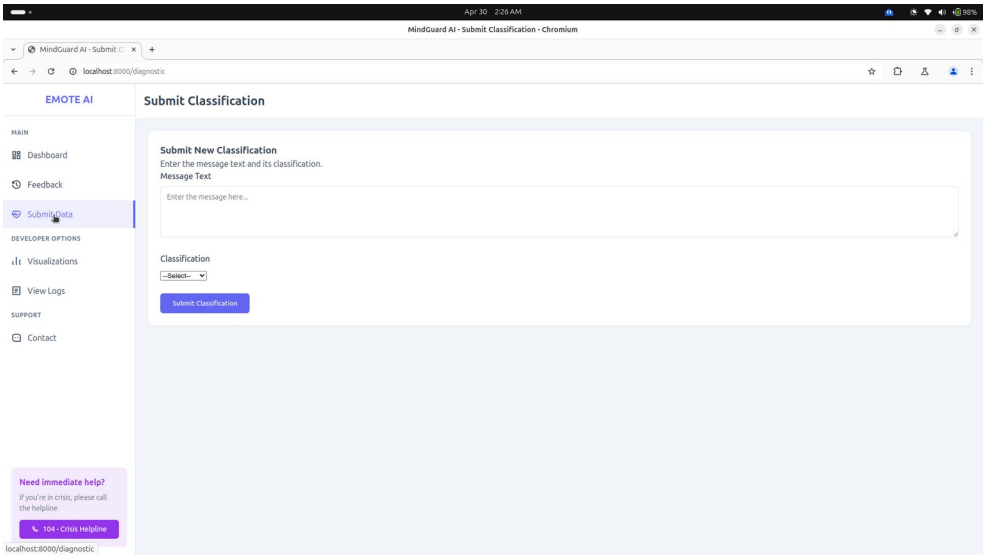
API Usage

Similar to the previous section the clicking of the button triggers a JavaScript fetch call http://localhost:8000/backend/feedback which routes the API request to the back-end via docker networks by forwarding the same request to http://backend:4000/feedback where backend is the name of the docker container where back-end services are run. Similar to the previous section the request is processed by the back-end and the response is forwarded to the front-end to the JavaScript fetch call.

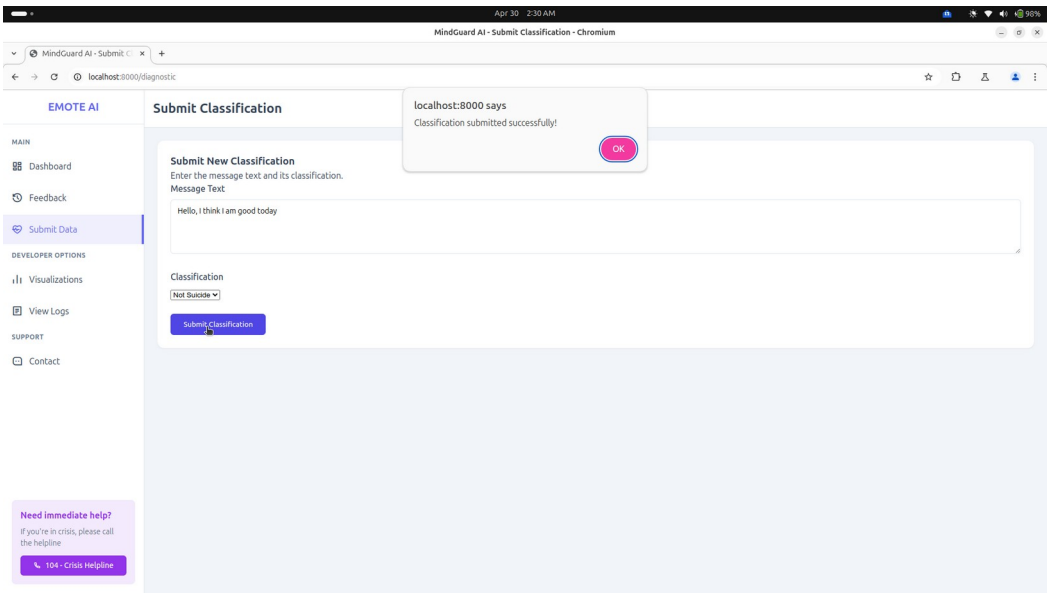
The back-end processing is as follows, it connects to the Postgres database and makes entries of the feedback rating and the optional feedback comments into the database to be looked upon by the developer. Request body : {“rating” : rating , “comments” : message }

Submit Data

To submit new data to the developer to be looked into you can feed the data with the label for that which is suicide and not suicide. To access the page click on the “Submit Data” page.



You will have to just type the message in the “Message Text” box and choose the label “Classification” which is one of the “suicide” or “not suicide” and then click “Submit Classification” to submit the data. Upon successful submission you see a prompt like this.



API Usage

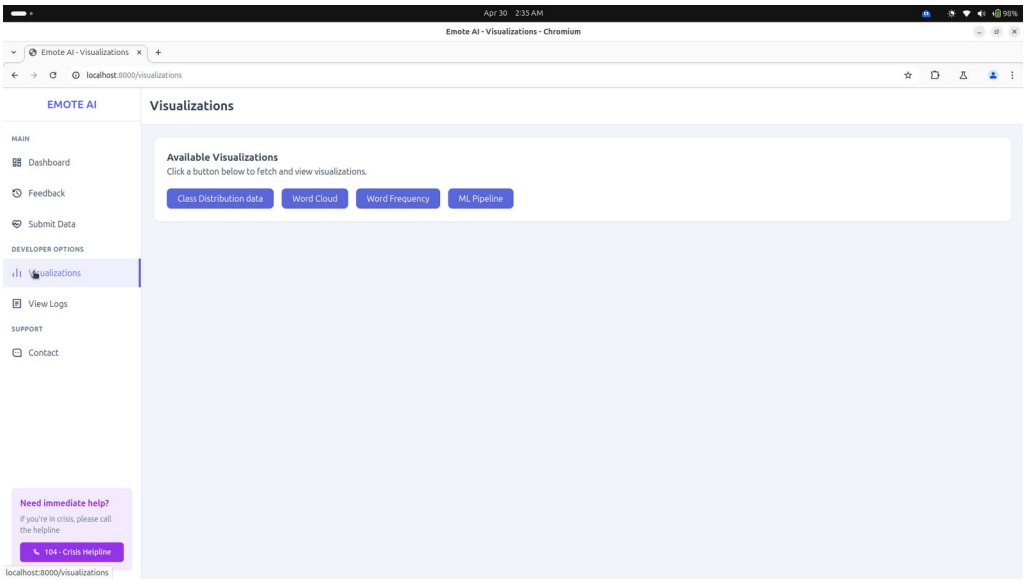
Similar to the previous section the clicking of the button triggers a JavaScript fetch call http://localhost:8000/backend/diagnostic/ which routes the API request to the backed via docker networks by forwarding the same request to http://backend:4000/diagnostic/ where backend is the name of the docker container where back-end services are run. Similar to the previous section the request is processed by the back-end and the response is forwarded to the front-end to the JavaScript fetch call.

The back end processing is as follows, it connects to the PostgreSQL database and makes entries of the feedback rating and the optional feedback comments into the database to be looked upon by the developer.

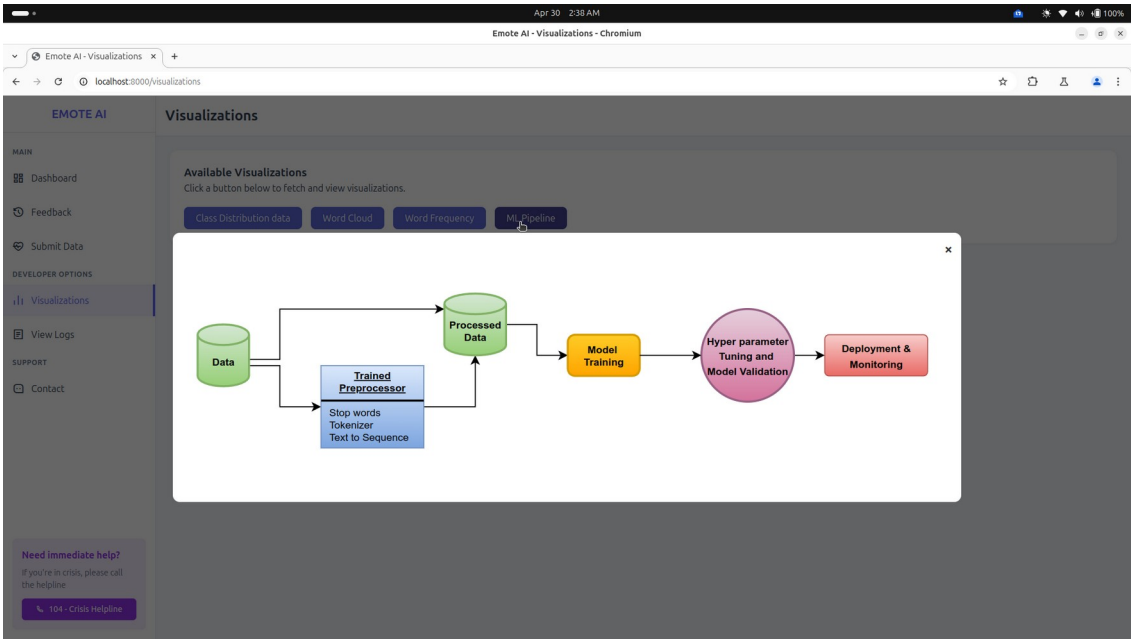
Request body : {“text” : message , “label” : label/classification }

Visualizations

I have added some visualizations which help to look at some EDA on the training data and the ML pipeline structure used to train the model and deployment. To access this click on the “Visualizations” on the left sidebar.



I have added 4 plots “Class distribution”, “Word Cloud”, “Word Frequency” and “ML Pipeline” as the name suggest. Just click on them to visualize them. For a demonstration I will attach the screenshot for the ML pipeline.

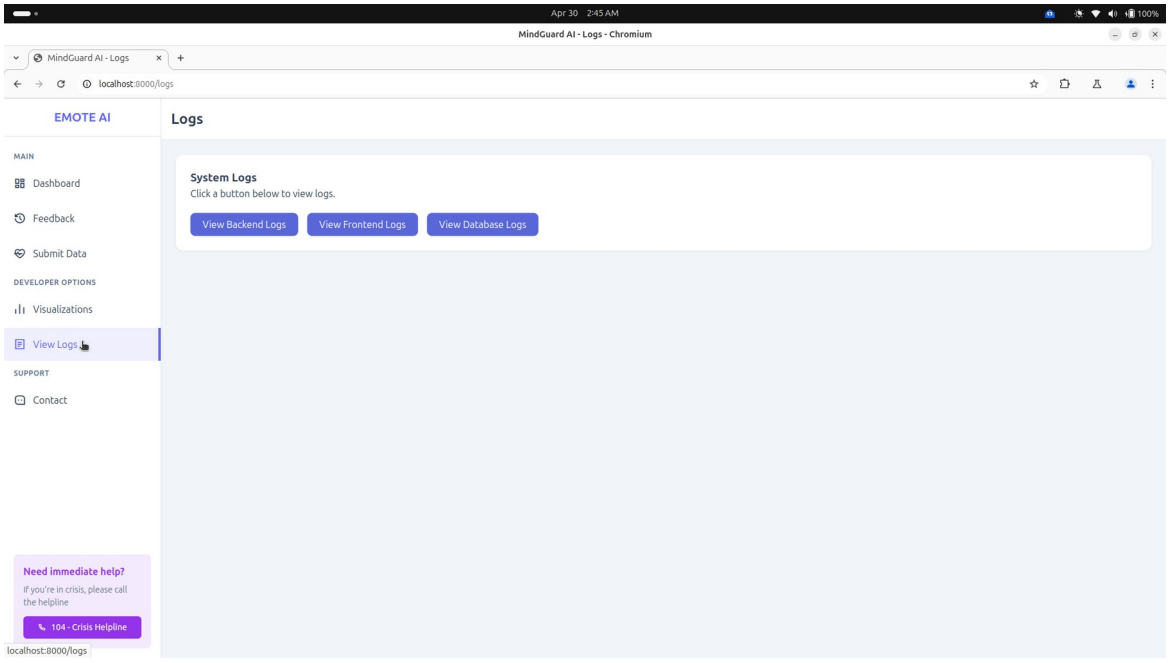


API Usage

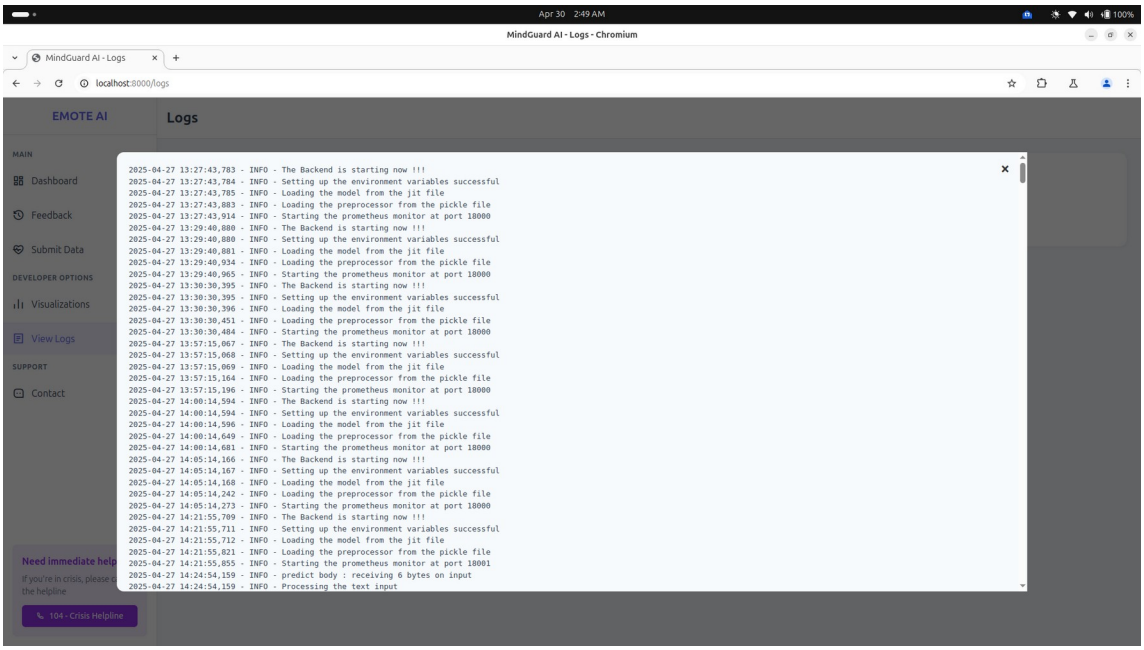
The API usage is same like the previous sections so I am not going to repeat. The JavaScript fetches the request http://localhost:8000/backend/visualizations/{plotName}/ which will route the call to the back-end (note that this is a get request) http://backend:4000/visualizations/{plotName}/ which will return the base64 encoded image stored. This is forwarded to the JavaScript and the image is displayed. Request body is not present as this is a get request.

View Logs

To view the logs generated by the containers, click the “View Logs” part of the sidebar.



Similar to the previous sections the log files shown are as the name suggests “View Backend Logs”, “View Frontend Logs” and “View Database Logs”. For demonstration I will attach the screenshot for the one log file.

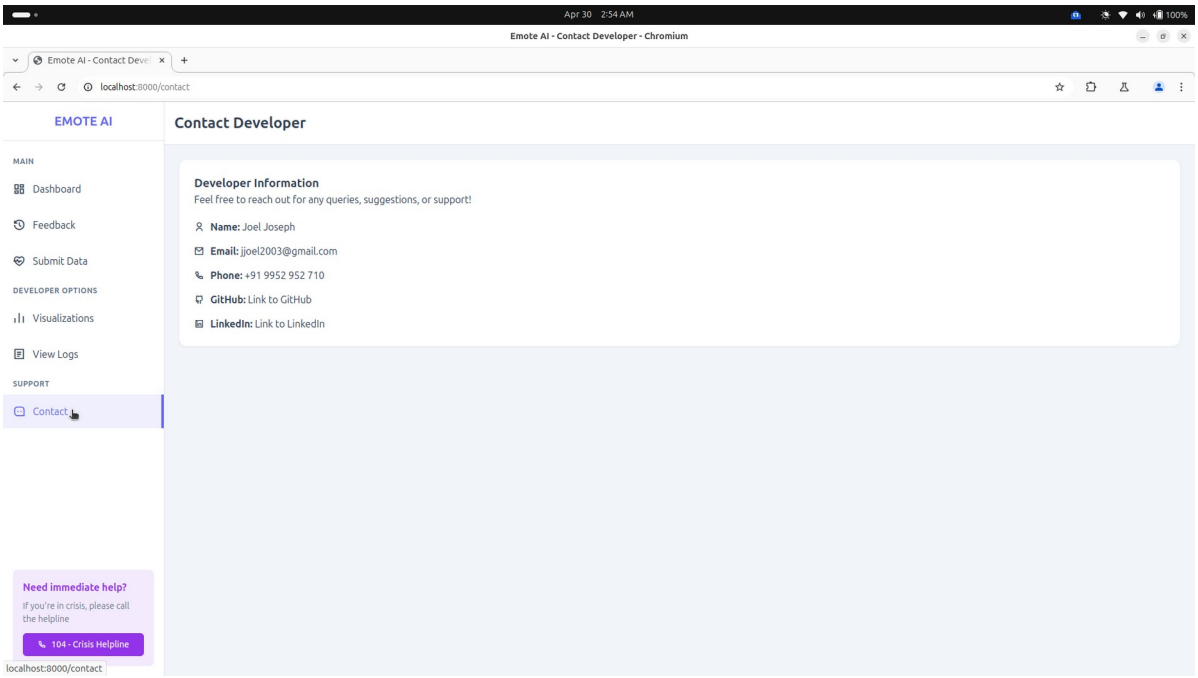


API Usage

Similar to the previous section, The JavaScript fetches the request http://localhost:8000/backend/logs/{container_name}/ which will route the call to the back-end (note that this is a get request) http://backend:4000/logs/{container_name}/ which will return the file encoded in string. This is forwarded to the JavaScript and the logs as a scroll type is shown. Request body is not present as this is a get request.

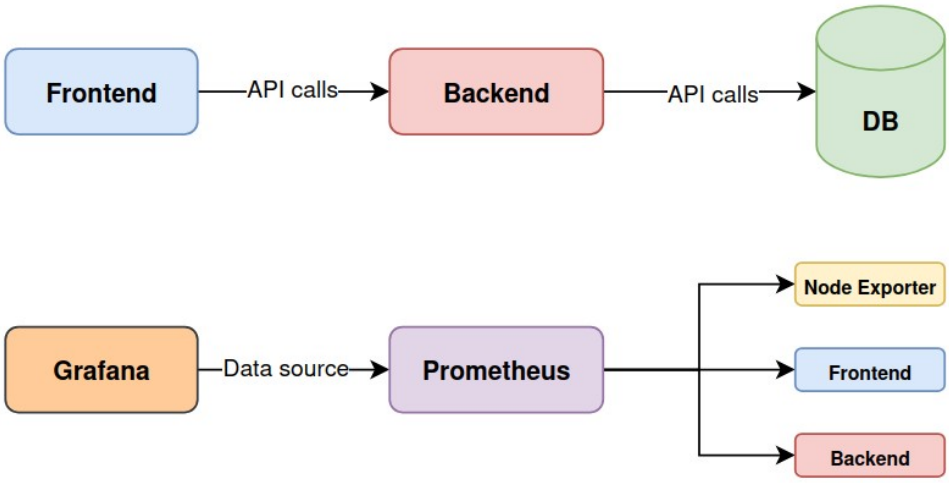
Contact page

To look at the contacts page and the developers details click on the “Contact” sidebar.



High Level Diagram

The high level diagram of the app is attached below



Low Level API requests

The low level API request body and response returned is given below.

Predict

Request body : { “text” : message }

Response : {“risk” : risk}

Feedback

Request body : {“rating” : rating , “comments” : message }

Response : None

Submit Classification

Request body : {“text” : message , “label” : label/classification }

Response : None

Visualizations

Request : Get

Response : {“image_base64” : base64 encoded image string}

View Logs

Request : Get

Response : {“log_text” : text file string}