2.3 Deploy model

2.3.1 Setting up data source

Trong thư mục Voz_neww/Voz_neww/spiders. File demospiders.py sẽ làm nhiệm vụ trọng yếu là cào hết tất cả các dữ liệu trong khoảng 10 phút so với thời gian thực từ trang web voz.vn. Các dữ liệu thu vào được bao gồm: mã số id, chủ đề thread, ngày đăng thread, người comment mới nhất, thời gian đăng comment mới nhất, tin nhắn và đường dẫn tới tin nhắn đó

```
import scrapy
import asyncio
from scrapy.spiders import Spider
from urllib.parse import urljoin, urlparse
from datetime import datetime, timedelta, timezone
import hashlib
import time
import re
class DemospiderSpider(Spider):
    name = "demospider"
    allowed_domains = ["voz.vn"]
    start_urls = ["https://voz.vn/whats-new"]

    processed_posts = set()
    last_scraped_timestamp = None

    custom_settings = {
        'CONCURRENT_REQUESTS': 1,
        'DOWNLOAD_DELAY': 1,
        'COOKIES_ENABLED': True,
        'ROBOTSTXT_OBEY': True,
        'DUPEFILTER_CLASS': 'scrapy.dupefilters.BaseDupeFilter'
    }

    def extract_thread_id(self, url):
        parsed = urlparse(url)
```

```
        path_parts = parsed.path.split('.')
        if len(path_parts) > 1:
            return path_parts[-1]
        return None

    def generate_item_id(self, thread_url, timestamp):
        thread_id = self.extract_thread_id(thread_url)
        if thread_id and timestamp:
            id_string = f"{thread_id}_{timestamp}"
            return hashlib.md5(id_string.encode()).hexdigest()
        return None

    def start_requests(self):
        while True:
            yield scrapy.Request(
                self.start_urls[0],
                callback=self.parse,
                dont_filter=True,
                meta={'dont_cache': True}
            )
            time.sleep(10)

    def parse(self, response):
        thread_containers = response.xpath("//div[contains(@class, 'structItem structItem--thread')]")

        threads = []
        for thread in thread_containers:
            latest_link = thread.xpath(".//div[@class='structItem-cell
structItem-cell--latest']//a[contains(@href, '/latest')]/@href").get()
            if latest_link:
                thread_url = urljoin(response.url, latest_link)
                thread_date_str = thread.xpath(".//div[@class='structItem-cell
structItem-cell--main']//time/@datetime").get()

                thread_info = {
                    'url': thread_url,
                    'thread_title': thread.xpath(".//div[@class='structItem-title']//a/text()").get(),
                    'thread_date': thread_date_str,
                    'timestamp': datetime.fromisoformat(thread_date_str.replace('Z', '+00:00')) if
thread_date_str else None
```

```
            }
            threads.append(thread_info)

    sorted_threads = sorted(threads, key=lambda x: x['timestamp'] if x['timestamp'] else
datetime.min)

    for thread in sorted_threads:
        yield scrapy.Request(
            thread['url'],
            callback=self.parse_latest_message,
            meta={'thread_info': thread},
            dont_filter=True
        )

  async def parse_latest_message(self, response):
    thread_info = response.meta['thread_info']

    message_containers = response.xpath("//article[contains(@class, 'message message--post')]")
    sorted_messages = sorted(
        message_containers,
        key=lambda m: m.xpath(".//time[@class='u-dt']/@datetime").get()
    )

    # Calculate the cutoff time: current time (UTC) - 10 minutes
    time_threshold = datetime.utcnow().replace(tzinfo=timezone.utc) - timedelta(minutes=10)

    new_messages = []
    for message_container in sorted_messages:
        message_content = message_container.xpath(".//div[contains(@class,
'message-userContent')]//div[@class='bbWrapper']//text()[not(ancestor::blockquote)]").getall()
        message_content = ' '.join([text.strip() for text in message_content if text.strip()])
        pattern = r'\{\n\t+\"lightbox_.*?\"Toggle sidebar\"\n\t+\}'

        message_content = re.sub(pattern, '', message_content, flags=re.DOTALL).strip()
        username =
message_container.xpath(".//h4[@class='message-name']//span[@itemprop='name']/text()").get()
        timestamp_str = message_container.xpath(".//time[@class='u-dt']/@datetime").get()
        timestamp = datetime.fromisoformat(timestamp_str.replace('Z', '+00:00')) if timestamp_str
else None
```

```
            if timestamp and timestamp >= time_threshold:
                item_id = self.generate_item_id(response.url, timestamp_str)
                if item_id and item_id not in self.processed_posts:
                    new_messages.append({
                        'id': item_id,
                        'thread_title': thread_info['thread_title'],
                        'thread_date': thread_info['thread_date'],
                        'latest_poster': username,
                        'latest_post_time': timestamp_str,
                        'message_content': message_content,
                        'thread_url': response.url
                    })

        # Yield each message with a delay between them
        for message in new_messages:
            self.processed_posts.add(message['id'])
            self.logger.info(f"Yielding post: {message['thread_title']} from user
{message['latest_poster']}")
            yield message
            await asyncio.sleep(1)  # Set a 1-second delay between each message

        if not new_messages:
            self.logger.info("No new messages found. Waiting 10 seconds before checking again.")
            await asyncio.sleep(10)  # Wait 10 seconds if no new messages are found

        # Resume from the last scraped timestamp in future requests
        if self.last_scraped_timestamp:
            self.logger.info(f"Resuming from last timestamp: {self.last_scraped_timestamp}")
```

Sau khi extract được, từng data đã extract sẽ tiến hành đi vào phần xử lý chính là pipeline.py

2.3.2 Transform and load

2.3.2.1 Transform

Sau khi đã extract. Pipeline.py sẽ có nhiệm vụ biến đổi những data cần thiết. Class FetchMessagePipeline dưới đây sẽ sử dụng các model đã được train trước đó để làm nhiệm vụ dự đoán cảm xúc cho từng "message content" và cập nhật data. Các hàm preprocess_raw_input , inference_model, prediction dùng để gọi model và dự đoán cảm xúc

```
import psycopg2
import logging
from datetime import datetime
import pytz
from tensorflow.keras.models import load_model
import pickle
import tensorflow as tf
from pyvi import ViTokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import model_from_json
tz = pytz.timezone('Asia/Ho_Chi_Minh')
# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

def preprocess_raw_input(raw_input, tokenizer):
    input_text_pre = list(tf.keras.preprocessing.text.text_to_word_sequence(raw_input))
    input_text_pre = " ".join(input_text_pre)
    input_text_pre_accent = ViTokenizer.tokenize(input_text_pre)
    tokenized_data_text = tokenizer.texts_to_sequences([input_text_pre_accent])
    vec_data = pad_sequences(tokenized_data_text, padding='post', maxlen=335)
    return vec_data

def inference_model(input_feature, model):
    output = model(input_feature).numpy()[0]
    result = output.argmax()
    label_dict = {'tiêu cực': 0, 'trung lập': 1, 'tích cực': 2}
    label = list(label_dict.keys())
    return label[int(result)]
```

```
def prediction(raw_input, tokenizer, model):
    input_model = preprocess_raw_input(raw_input, tokenizer)
    result= inference_model(input_model, model)
    return result
model_path = "./models/model.pkl"
tokenizer_data_path = "./models/tokenizer_data.pkl"
with open(model_path, "rb") as model_file:
    model = pickle.load(model_file)

with open(tokenizer_data_path, "rb") as tokenizer_file:
    my_tokenizer = pickle.load(tokenizer_file)

class FetchMessagePipeline:

    def analyze_sentiment(self, text):
        try:
            # Get sentiment label from underthesea
            sentiment_label = prediction(text,my_tokenizer,model)

            # Convert sentiment labels to counts
            sentiment_counts = {
                'positive': 0,
                'negative': 0,
                'neutral': 0
            }

            # Increment the appropriate counter based on sentiment
            if sentiment_label == 'tích cực':
                sentiment_counts['positive'] = 1
            elif sentiment_label == 'tiêu cực':
                sentiment_counts['negative'] = 1
            else:
                sentiment_counts['neutral'] = 1

            return sentiment_counts

        except Exception as e:
            logger.error(f"Error in sentiment analysis: {str(e)}")
            return {'positive': 0, 'negative': 0, 'neutral': 0}
```

```
def process_item(self, item, spider):
    """Process each scraped item"""
    try:
        # Get the message content
        message_text = item['message_content']

        # Analyze sentiment
        sentiment_counts = self.analyze_sentiment(message_text)

        # Update the item with sentiment counts
        item.update({
            **sentiment_counts,
            'processed_at': datetime.now(tz).isoformat()
        })

        logger.info(f"Processed item {item['id']}")
        return item

    except Exception as e:
        logger.error(f"Error processing item: {str(e)}")
        return item
```

2.3.2.2 Load

Sau khi đã xong transform xong data sẽ có class tiến SentimentAnalysisPipeline tiến hành gọi database và load dữ liệu vào đóng quá trình scrap cho mỗi data đầu vào

```
class SentimentAnalysisPipeline:
    def __init__(self):
        try:
            self.conn = psycopg2.connect(
                dbname="vozdb",
                user="postgres",
                password="postgres",
                host="db",
            )
            self.cur = self.conn.cursor()
```

```
        except Exception as e:
            logger.error(f"Error connecting to database: {str(e)}")

    def process_item(self, item, spider):
        try:
            # Store in database with sentiment counts
            self.cur.execute("""
                INSERT INTO voz_messages (
                    id, thread_title, thread_date, latest_poster,
                    latest_post_time, message_content, thread_url,
                    positive_count, negative_count, neutral_count,
                    analyzed_at
                ) VALUES (
                    %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
                )
                ON CONFLICT (id) DO UPDATE SET
                    positive_count = EXCLUDED.positive_count,
                    negative_count = EXCLUDED.negative_count,
                    neutral_count = EXCLUDED.neutral_count,
                    analyzed_at = EXCLUDED.analyzed_at
            """, (
                item['id'], item['thread_title'], item['thread_date'],
                item['latest_poster'], item['latest_post_time'],
                item['message_content'], item['thread_url'],
                item['positive'], item['negative'], item['neutral'],
                item['processed_at']
            ))

            self.conn.commit()
            logger.info(f"Successfully stored item {item['id']} in database")

        except Exception as e:
            logger.error(f"Error storing item {item['id']} in database: {str(e)}")
            self.conn.rollback()

        return item

    def close_spider(self, spider):
        self.cur.close()
        self.conn.close()
```

2.3.3 Backend

Sau khi đã load xong dữ liệu vào database tiến hành tạo API bằng FastAPI.Các chức năng của Hàm sau đây:

- wait_for_db() : Gọi database, nếu không có database sẽ chờ 2 giây và gọi lại, cho tới lần thứ 30 sẽ trả về lỗi nếu không kết nối được
- get_db_connection(): Tiến hành kết nối với database dựa trên config
- get_sentiment_stats(conn, limit=15): Lấy thống kê cảm xúc trong khoảng thời gian nhất định (số giây gần nhất) từ cơ sở dữ liệu.
- get_sentiment_summary(conn): Lấy tóm tắt tổng thể về cảm xúc trong vòng 24 giờ qua
- get_messages_with_sentiment: Lấy full tổng thể của dữ liệu đầu ra
- Cac api endpoint

```python
# api/main.py
from datetime import datetime, timedelta
from fastapi import FastAPI, WebSocket, WebSocketDisconnect, Depends, Query, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import psycopg2
from psycopg2.extras import RealDictCursor
import os
import logging
import time
from contextlib import contextmanager
from typing import Optional, List

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
```

```
logger = logging.getLogger(__name__)

# Initialize FastAPI app
app = FastAPI(title="VOZ Analytics API")
origins = ["*"]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Database configuration
DB_CONFIG = {
    "dbname": "vozdb",
    "user": "postgres",
    "password": "postgres",
    "host": "db",
    "port": "5432"
}

def wait_for_db(max_retries=30, delay_seconds=2):
    """Wait for database to be ready"""
    retries = 0
    while retries < max_retries:
        try:
            conn = psycopg2.connect(**DB_CONFIG)
            conn.close()
            logger.info("Successfully connected to the database")
            return True
        except psycopg2.Error as e:
            retries += 1
            logger.warning(f"Attempt {retries}/{max_retries} to connect to database failed: {str(e)}")
            logger.warning("Retrying in %s seconds...", delay_seconds)
            time.sleep(delay_seconds)

    raise Exception("Could not connect to the database after multiple attempts")
```

```python
@contextmanager
def get_db_connection():
    """Context manager for database connections"""
    conn = None
    try:
        conn = psycopg2.connect(**DB_CONFIG, cursor_factory=RealDictCursor)
        yield conn
    except psycopg2.Error as e:
        logger.error(f"Database connection error: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database connection error: {str(e)}")
    finally:
        if conn:
            conn.close()
            logger.debug("Database connection closed")

def get_db():
    """Database dependency for FastAPI"""
    with get_db_connection() as conn:
        yield conn

# Analytics queries
def get_sentiment_stats(conn, limit=15):
    """Get the last `limit` seconds of sentiment statistics"""
    try:
        with conn.cursor() as cur:
            query = f"""
            WITH RECURSIVE time_series AS (
                SELECT NOW() - INTERVAL '1 second' * generate_series(0, {limit - 1}) AS check_time
            )
            SELECT
                ts.check_time AS check_time,
                (SELECT SUM(positive_count) FROM voz_messages) AS total_positive_count,
                (SELECT SUM(negative_count) FROM voz_messages) AS total_negative_count,
                (SELECT SUM(neutral_count) FROM voz_messages) AS total_neutral_count,
                (SELECT COUNT(*) FROM voz_messages) AS total_messages
            FROM time_series ts
            ORDER BY ts.check_time DESC;
            """
            cur.execute(query)
            results = cur.fetchall()
```

```
        return results
    except psycopg2.Error as e:
        logger.error(f"Error fetching sentiment stats list: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database query error: {str(e)}")


def get_sentiment_summary(conn):
    """Get overall sentiment summary for the last 24 hours"""
    try:
        with conn.cursor() as cur:
            query = """
                SELECT
                    SUM(positive_count) as total_positive,
                    SUM(negative_count) as total_negative,
                    SUM(neutral_count) as total_neutral,
                    COUNT(*) as total_messages
                FROM voz_messages
                WHERE analyzed_at >= NOW() - INTERVAL '24 hours'
            """
            cur.execute(query)
            result = cur.fetchone()
            return result
    except psycopg2.Error as e:
        logger.error(f"Error fetching sentiment summary: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database query error: {str(e)}")

def get_messages_with_sentiment(conn, limit: int = 10, offset: int = 0, thread_id: Optional[str] =
None):
    """Get messages with their sentiment analysis"""
    try:
        with conn.cursor() as cur:
            query = """
                SELECT
                    id,
                    thread_title,
                    thread_date,
                    message_content,
                    latest_poster,
                    latest_post_time,
                    thread_url,
```

```
            CASE
                WHEN positive_count = 1 THEN 'positive'
                WHEN negative_count = 1 THEN 'negative'
                ELSE 'neutral'
            END as sentiment,
            analyzed_at
        FROM voz_messages
        WHERE 1=1
    """
    params = []

    if thread_id:
        query += " AND thread_id = %s"
        params.append(thread_id)

    query += """
        ORDER BY analyzed_at DESC
        LIMIT %s OFFSET %s
    """
    params.extend([limit, offset])

    cur.execute(query, params)
    messages = cur.fetchall()

    # Get total count for pagination
    count_query = """
        SELECT COUNT(*) as total
        FROM voz_messages
        WHERE 1=1
    """
    if thread_id:
        count_query += " AND thread_id = %s"
        cur.execute(count_query, [thread_id] if thread_id else None)
    else:
        cur.execute(count_query)

    total_count = cur.fetchone()['total']

    return {
        "messages": messages,
```

```
            "total": total_count,
            "limit": limit,
            "offset": offset
        }
    except psycopg2.Error as e:
        logger.error(f"Error fetching messages with sentiment: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database query error: {str(e)}")


# API endpoints
@app.get("/stats/sentiment/iter")
@app.get("/stats/sentiment/iter")
def sentiment_stats(
    conn = Depends(get_db),
    limit: int = Query(15, ge=1, le=60)  # Limit can be adjusted (default 15, max 60)
):
    """Get the last `limit` seconds of sentiment statistics"""
    return get_sentiment_stats(conn, limit=limit)

@app.get("/stats/sentiment/summary")
def sentiment_summary(conn = Depends(get_db)):
    """Get overall sentiment summary"""
    return get_sentiment_summary(conn)

@app.get("/messages/sentiment")
def get_messages(
    conn = Depends(get_db),
    limit: int = Query(5, ge=1, le=100),
    offset: int = Query(0, ge=0),
    thread_id: Optional[str] = None
):
    """Get messages with their sentiment analysis"""
    return get_messages_with_sentiment(conn, limit, offset, thread_id)

# Health check endpoint
@app.get("/health")
async def health_check():
    """Health check endpoint that also verifies database connection"""
    try:
        with get_db_connection() as conn:
```

```
        with conn.cursor() as cur:
            cur.execute("SELECT 1")
            return {
                "status": "healthy",
                "database": "connected",
                "timestamp": datetime.now().isoformat()
            }
    except Exception as e:
        return {
            "status": "unhealthy",
            "database": "disconnected",
            "error": str(e),
            "timestamp": datetime.now().isoformat()
        }

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True, log_level="info")
```

2.3.4 Frontend

Ở phần này chỉ chú ý đến phần trong file src của vite-project. Trong src sẽ có phần chính là App.jsx để tiến hành chạy frontend. Tụi em sẽ sử dụng thư viện của MaterialUI (MUI) để hình hoá dữ liệu.

```
import { useState, useEffect } from 'react';
import './App.css';
import { PieChart } from '@mui/x-charts/PieChart';
import { LineChart } from '@mui/x-charts/LineChart';
import {
 Typography,
 List,
 ListItem,
 Card,
 CardContent,
 CardHeader,
 Divider,
 Box,
```

```
 Paper,
 Link,
} from '@mui/material';

const API_ENDPOINTS = {
 sentimentSummary: 'http://localhost:8000/stats/sentiment/summary',
 sentimentMessages: 'http://localhost:8000/messages/sentiment',
 sentimentByTime: 'http://localhost:8000/stats/sentiment/iter',
};
function formatSentimentData(data) {
 const totalMessages = data.total_messages ?? 0;
 const positive = totalMessages ? ((data.total_positive / totalMessages) * 100).toFixed(2) : 0;
 const negative = totalMessages ? ((data.total_negative / totalMessages) * 100).toFixed(2) : 0;
 const neutral = totalMessages ? ((data.total_neutral / totalMessages) * 100).toFixed(2) : 0;
 return {
  chartData: [
   { id: 0, value: data.total_positive ?? 0, label: 'Positive' },
   { id: 1, value: data.total_negative ?? 0, label: 'Negative' },
   { id: 2, value: data.total_neutral ?? 0, label: 'Neutral' },
  ],
  summary: {
   totalMessages,
   positive,
   negative,
   neutral,
  },
 };
}
function normalizeSentimentData(dataArray) {
 return dataArray.map(data => {
  const hourDate = new Date(data.check_time);
  const total = data.total_messages;
  return {
   Hour: hourDate,
   positive: total ? Number((data.total_positive_count / total * 100).toFixed(2)) : 0,
   negative: total ? Number((data.total_negative_count / total * 100).toFixed(2)) : 0,
   neutral: total ? Number((data.total_neutral_count / total * 100).toFixed(2)) : 0,
  };
 });
}
function App() {
 const [sentiment, setSentiment] = useState([]);
 const [sentimentSummary, setSentimentSummary] = useState({});
```

```
const [messages, setMessages] = useState([]);
const [analysisByTime, setAnalysisByTime] = useState([]);

const fetchData = async (endpoint, processData) => {
  try {
    const response = await fetch(endpoint);
    if (!response.ok) {
      throw new Error(`Failed to fetch: ${endpoint}`);
    }
    const data = await response.json();
    processData(data);
  } catch (error) {
    console.error(`Error fetching data from ${endpoint}:`, error);
  }
};

useEffect(() => {
  const fetchAllData = () => {
    fetchData(API_ENDPOINTS.sentimentSummary, data => {
      const formatted = formatSentimentData(data);
      setSentiment(formatted.chartData);
      setSentimentSummary(formatted.summary);
    });
    fetchData(API_ENDPOINTS.sentimentMessages, data => setMessages(data.messages));
    fetchData(API_ENDPOINTS.sentimentByTime, data => {
      const normalizedData = normalizeSentimentData(data);
      setAnalysisByTime(normalizedData);
    });
  };

  fetchAllData();
  const intervalId = setInterval(fetchAllData, 1000);

  return () => clearInterval(intervalId);
}, []);

return (
  <Box sx={{ padding: '2rem', backgroundColor: '#f9f9f9', minHeight: '100vh' }}>
    {/* Summary Boxes */}
    <Box
      sx={{
        display: 'flex',
        gap: '1.5rem',
```

```
    marginBottom: '2rem',
    justifyContent: 'space-between',
  }}
>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Total Messages</Typography>
    <Typography variant="h4">{sentimentSummary.totalMessages || 0}</Typography>
  </Paper>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Positive (%)</Typography>
    <Typography variant="h4">{sentimentSummary.positive || 0}%</Typography>
  </Paper>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Negative (%)</Typography>
    <Typography variant="h4">{sentimentSummary.negative || 0}%</Typography>
  </Paper>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Neutral (%)</Typography>
    <Typography variant="h4">{sentimentSummary.neutral || 0}%</Typography>
  </Paper>
</Box>

{/* Main Content */}
<Box sx={{ display: 'flex', gap: '2rem' }}>
  {/* Left Section */}
  <Box sx={{ flex: 7 }}>
    <Card elevation={3} sx={{ marginBottom: '2rem' }}>
      <CardHeader title="Sentiment Distribution" />
      <Divider />
      <CardContent>
        {sentiment.length > 0 ? (
          <PieChart
            series={[{ data: sentiment }]}
            width={600}
            height={400}
          />
        ) : (
          <Typography>Loading sentiment data...</Typography>
        )}
      </CardContent>
    </Card>

    <Card elevation={3}>
```

```
    <CardHeader title="Sentiment Over Time" />
    <Divider />
    <CardContent>
     {analysisByTime.length > 0 ? (
      <LineChart
       height={400}
       series={[
        {
         data: analysisByTime.map(item => item.positive),
         label: 'Positive',
         area: true,
         stack: 'total',
         showMark: false,
        },
        {
         data: analysisByTime.map(item => item.negative),
         label: 'Negative',
         area: true,
         stack: 'total',
         showMark: false,
        },
        {
         data: analysisByTime.map(item => item.neutral),
         label: 'Neutral',
         area: true,
         stack: 'total',
         showMark: false,
        },
       ]}
       xAxis={[
        {
         data: analysisByTime.map(item => item.Hour),
         scaleType: 'time',
         valueFormatter: item => item.toLocaleTimeString(),
        },
       ]}
      />
     ) : (
      <Typography>Loading time-series data...</Typography>
     )}
    </CardContent>
   </Card>
  </Box>
```

```
    {/* Right Section */}
    <Box sx={{ flex: 3 }}>
      <Card elevation={3}>
        <CardHeader title="Recent Messages" />
        <Divider />
        <CardContent>
          <List>
            {messages.length > 0 ? (
              messages.map(item => (
                <ListItem key={item.id} sx={{ marginBottom: '1rem' }}>
                  <Paper elevation={1} sx={{ padding: '1rem', width: '100%' }}>
                    <Typography variant="body1">
                      <strong>Time:</strong> {new Date(item.latest_post_time).toLocaleString()}
                    </Typography>
                    <Typography variant="body1">
                      <strong>Message:</strong> {item.message_content}
                    </Typography>
                    <Typography variant="body1">
                      <strong>Thread:</strong>{' '}
                      <Link href={item.thread_url} target="_blank" rel="noopener noreferrer">
                        {item.thread_url}
                      </Link>
                    </Typography>
                    <Typography variant="body1">
                      <strong>Sentiment:</strong> {item.sentiment}
                    </Typography>
                  </Paper>
                </ListItem>
              ))
            ) : (
              <Typography>No messages to display.</Typography>
            )}
          </List>
        </CardContent>
      </Card>
    </Box>
  </Box>
);
}
export default App;
```

2.3.5 Docker build file

Để dự án này có thể chạy đơn giản không cần phải tải môi trường hay cài bất cứ phần mềm chạy code nào, chúng em sẽ sử dụng Docker làm việc đó. Và sau đây là các file cần thiết để build cho dự án này :

- Docker-compose.yaml: Dùng để xây dựng các image cần thiết cho dự án. Trong đó có các image như sau :
  - db: Xây dựng image cho csdl postgresql và thực thi lệnh sql trong file init.sql
  - api: Xây dựng image cho backend và khởi tạo chạy FastAPI dựa trên file Dockerfile trong Voz_neww
  - scrapy: Dựa vào image của backend và bắt đầu scrap data về
  - frontend: Xây dựng image cho frontend dựa trên Dockerfile trong vite_project

```yaml
services:
 db:
  image: postgres:13
  volumes:
   - ./VOZ_neww/postgres/init.sql:/docker-entrypoint-initdb.d/init.sql
  environment:
   POSTGRES_USER: postgres
   POSTGRES_PASSWORD: postgres
   POSTGRES_DB: vozdb
  ports:
   - "5433:5432"
  healthcheck:
   test: ["CMD-SHELL", "pg_isready -U postgres"]
   interval: 5s
   timeout: 5s
   retries: 5
  networks:
   - voz_dash

 api:
  build: ./VOZ_neww
```

```
  command: ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
  ports:
    - "8000:8000"
  volumes:
    - ./VOZ_neww:/app
  depends_on:
    db:
      condition: service_healthy
  environment:
    DATABASE_URL: postgresql://postgres:postgres@db:5432/vozdb
  networks:
    - voz_dash

scrapy:
  build: ./VOZ_neww
  command: ["scrapy", "crawl", "demospider"]
  volumes:
    - ./VOZ_neww:/app
  depends_on:
    db:
      condition: service_healthy
  networks:
    - voz_dash

frontend:
  build: ./vite-project
  ports:
    - "5173:5173"
  networks:
    - voz_dash


networks:
voz_dash:
  driver: bridge
```

Vite-project/Dockerfile:

```
FROM node:18.17.1
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0"]
```

Voz_new/Dockerfile

```
FROM python:3.11

WORKDIR /app

RUN apt-get update && apt-get install -y \
  postgresql-client \
  cron \
  && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY models/ /app/models/

COPY start.sh /app/start.sh
RUN chmod +x /app/start.sh
EXPOSE 8000
```

SETUP A DEMO

1. Ở bước này, để có thể khởi tạo project, trước hết hãy tải docker về máy tính.
2. Sau đó vào đường link github https://github.com/Amature123/new_new_sentiment và clone project về
3. Bật màn command promt hoặc terminal và đi tới địa chỉ của project
4. Gõ Docker compose build để tiến hành tải các image cần thiết
5. Sau khi đã tải các project cần thiết thì tiến hành gõ docker compose up để chạy code
6. Sau khi đã hoàn tất hãy vào localhost:5183 và sẽ hiện hết các dashboard.

Tài liệu tham khảo

- **Docker:** https://www.docker.com/
- **Frontend:** https://vite.dev/
- **Backend:** https://fastapi.tiangolo.com/
- Model: https://www.tensorflow.org/
- Scrapy: https://scrapy.org/
  Project được lấy ví dụ từ
- https://github.com/nama1arpit/reddit-streaming-pipeline
- https://github.com/jalvin99/RedditSentimentAnalyzer
- https://github.com/undertheseanlp/underthesea