Report
Khrystyna Maryniuk

Scoring.js	import React, { useState } from 'react';
	import axios from 'axios';
	
	function Scoring() {
	  const [ticker, setTicker] = useState('');
	  const [year, setYear] = useState('');
	  const [quarter, setQuarter] = useState('');
	  const [result, setResult] = useState(null);
	  const [loading, setLoading] = useState(false);
	
	  const handleSubmit = async (e) => {
	  e.preventDefault();
	  setLoading(true);
	  setResult(null);
	
	  try {
	  const response = await axios.post('http://34.27.95.43:5000/score', {
	  ticker: ticker.trim(),
	  year: parseInt(year),
	  quarter: parseInt(quarter)
	  });
	
	  setResult({
	  avg_score: response.data.avg_score,
	  avg_magnitude: response.data.avg_magnitude,
	  paragraph_count: response.data.paragraph_count
	  });
	  } catch (error) {
	  setResult({ error: error.response?.data?.error || 'Could not fetch score. Try again.' });
	  } finally {
	  setLoading(false);
	  }
	  };
	
	  return (
	  <div style={{ maxWidth: 400, margin: 'auto' }}>
	   Scoring Page
		● React component for user input and display of sentiment scores.

```
    <form onSubmit={handleSubmit} style={{ display: 'flex',
flexDirection: 'column', gap: '1rem' }}>
      <input
        value={ticker}
        onChange={e => setTicker(e.target.value.toUpperCase())}
        placeholder="Ticker (e.g. AAPL)"
        required
        maxLength={5}
        style={{ padding: '0.5rem', fontSize: '1rem' }}
      />
      <input
        type="number"
        value={year}
        onChange={e => setYear(e.target.value)}
        placeholder="Year (e.g. 2024)"
        required
        min={2000}
        max={2025}
        style={{ padding: '0.5rem', fontSize: '1rem' }}
      />
      <input
        type="number"
        value={quarter}
        onChange={e => setQuarter(e.target.value)}
        placeholder="Quarter (1-4)"
        required
        min={1}
        max={4}
        style={{ padding: '0.5rem', fontSize: '1rem' }}
      />
      <button type="submit" disabled={loading} style={{ padding:
'0.75rem', fontSize: '1rem' }}>
        {loading ? 'Loading...' : 'Get Score'}
      </button>
    </form>

    {result && (
      <div style={{ marginTop: 20 }}>
        {result.error ? (
          <p style={{ color: 'red' }}>{result.error}</p>
        ) : (
          <>
            <p><strong>Average Score:</strong> {result.avg_score}</p>
```

	Average Magnitude: {result.avg_magnitude} Paragraph Count: {result.paragraph_count} </>)})}); } export default Scoring;	

app.py	import React, { useState } from 'react';
	import axios from 'axios';
	
	function Scoring() {
	const [ticker, setTicker] = useState('');
	const [year, setYear] = useState('');
	const [quarter, setQuarter] = useState('');
	const [result, setResult] = useState(null);
	const [loading, setLoading] = useState(false);
	
	const handleSubmit = async (e) => {
	e.preventDefault();
	setLoading(true);
	setResult(null);
	
	try {
	const response = await axios.post('http://34.27.95.43:5000/score', {
	ticker: ticker.trim(),
	year: parseInt(year),
	quarter: parseInt(quarter)
	});
	
	setResult({
	avg_score: response.data.avg_score,
	avg_magnitude: response.data.avg_magnitude,
	paragraph_count: response.data.paragraph_count
	});
	} catch (error) {
		Handles HTTP endpoints /submit and /score.

```
      setResult({ error: error.response?.data?.error || 'Could not fetch
score. Try again.' });
    } finally {
      setLoading(false);
    }
  };

  return (
    <div style={{ maxWidth: 400, margin: 'auto' }}>
      <h2>Scoring Page</h2>
      <form onSubmit={handleSubmit} style={{ display: 'flex',
flexDirection: 'column', gap: '1rem' }}>
        <input
          value={ticker}
          onChange={e => setTicker(e.target.value.toUpperCase())}
          placeholder="Ticker (e.g. AAPL)"
          required
          maxLength={5}
          style={{ padding: '0.5rem', fontSize: '1rem' }}
        />
        <input
          type="number"
          value={year}
          onChange={e => setYear(e.target.value)}
          placeholder="Year (e.g. 2024)"
          required
          min={2000}
          max={2025}
          style={{ padding: '0.5rem', fontSize: '1rem' }}
        />
        <input
          type="number"
          value={quarter}
          onChange={e => setQuarter(e.target.value)}
          placeholder="Quarter (1-4)"
          required
          min={1}
          max={4}
          style={{ padding: '0.5rem', fontSize: '1rem' }}
        />
        <button type="submit" disabled={loading} style={{ padding:
'0.75rem', fontSize: '1rem' }}>
          {loading ? 'Loading...' : 'Get Score'}
```

	</button> </form> {result && (<div style={{ marginTop: 20 }}> {result.error ? (<p style={{ color: 'red' }}>{result.error}) : (<> Average Score: {result.avg_score} Average Magnitude: {result.avg_magnitude} Paragraph Count: {result.paragraph_count} </>)})}); }export default Scoring;	
REQUIREMENTS.TXT	flask flask-cors requests nltk transformers torch scikit-learn pandas google-cloud-datastore	Dependencies list.
scoring.py	import datetime import nltk from nltk.sentiment import SentimentIntensityAnalyzer from nltk.tokenize import sent_tokenize from transformers import AutoTokenizer, AutoModelForSequenceClassification from torch.nn.functional import softmax from google.cloud import datastore nltk.download('vader_lexicon') nltk.download('punkt_tab')	Implements sentiment analysis using FinBERT and VADER, saves results to Google Datastore.

```
PROJECT_ID = "sentiment-analysis-379200"
datastore_client = datastore.Client(project=PROJECT_ID)

model =
AutoModelForSequenceClassification.from_pretrained('ProsusAI/finBE
RT')
tokenizer = AutoTokenizer.from_pretrained('ProsusAI/finBERT')
sia = SentimentIntensityAnalyzer()

def analyze_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True,
truncation=True)
    outputs = model(**inputs)
    probabilities = softmax(outputs.logits, dim=1)
    return probabilities[0]

def process_text(text):
    probabilities = analyze_sentiment(text)
    sentiment_score = (probabilities[1] + 2 * probabilities[2] + 3 *
probabilities[0]) - 2
    sentences = sent_tokenize(text)
    magnitude = sum(abs(sia.polarity_scores(s)['compound']) for s in
sentences)
    return sentiment_score.item(), magnitude

def compute_average_scores(paragraphs):
    scores = []
    magnitudes = []
    for p in paragraphs:
        score, magnitude = process_text(p)
        scores.append(score)
        magnitudes.append(magnitude)
    avg_score = sum(scores) / len(scores) if scores else 0.0
    avg_magnitude = sum(magnitudes) / len(magnitudes) if magnitudes
else 0.0
    return avg_score, avg_magnitude

def save_score_to_db(ticker, year, quarter, avg_score,
avg_magnitude):
    key = datastore_client.key("TranscriptScore")
    entity = datastore.Entity(key=key)
    entity.update({
```

```
        "ticker": ticker.upper(),
        "year": year,
        "quarter": quarter,
        "avg_score": avg_score,
        "avg_magnitude": avg_magnitude,
        "processed_at": datetime.datetime.utcnow()
    })
    datastore_client.put(entity)

def extract_paragraphs_from(api_response):
    """
    Extract paragraphs from the Ninja API JSON response.
    Adjust this based on the actual response structure.
    """
    # Example assuming API returns a list of dicts with 'text' key:
    paragraphs = []
    if isinstance(api_response, list):
        for item in api_response:
            text = item.get('text')
            if text:
                paragraphs.append(text)
    elif isinstance(api_response, dict):
        # If transcript is a string or paragraphs under a key, adjust here
        text = api_response.get('transcript') or api_response.get('text')
        if text:
            # If it's a long string, split by newline or paragraph markers
            paragraphs = [p.strip() for p in text.split('\n') if p.strip()]
    return paragraphs
```

Feature	Source	My Adaptation
FinBERT model for financial sentiment	Bruno's process.py and scoring.py	Fully used in your scoring.py
VADER sentiment analyzer	Bruno's code	Used in scoring.py for computing sentiment magnitude
analyze_sentiment() function	Bruno's process.py	Recreated in scoring.py
Paragraph scoring logic	Bruno's weighted score formula: (prob[1] + 2*prob[2] + 3*prob[0]) - 2	(prob[1]*0 + (-1)*prob[2] + 1*prob[0])
Google Cloud Datastore saving	Bruno's save_to_db() equivalent	Using save_score_to_db() to persist scores
API structure (Flask backend)	Bruno's app architecture	Adapted and extended
React Frontend Form	Bruno had a similar React UI for scoring	Adapted it in Scoring.js

Libraries and Models

1. transformers by Hugging Face
    - AutoModelForSequenceClassification and AutoTokenizer
    - Model: ProsusAI/finBERT — a **pretrained BERT** model specifically fine-tuned for **financial sentiment analysis**.
    - Purpose: It classifies text into three probabilities: positive, neutral, and negative.
2. torch (PyTorch)
    - You use: softmax() from torch.nn.functional
    - Purpose: Converts raw model logits (output values) into probabilities for each sentiment class.
3. nltk (Natural Language Toolkit)
    - SentimentIntensityAnalyzer() → VADER-based sentiment analyzer
    - sent_tokenize() → splits paragraphs into sentences

- Purpose: VADER computes the **magnitude** of emotional intensity (e.g., strength of sentiment), complementary to FinBERT's classification.

4. Requests
    - Purpose: Makes authenticated HTTP requests to the external **API Ninjas** endpoint to fetch earnings call transcripts.

FinBERT for Robust Financial Sentiment Analysis

FinBERT is a **pretrained language model** based on **BERT** (Bidirectional Encoder Representations from Transformers), but it is **fine-tuned on financial text**, such as:

- Earnings call transcripts
- Financial news
- Analyst reports

the model: `ProsusAI/finBERT` via Hugging Face's `transformers` library.

FinBERT is trained to detect **positive, negative, or neutral tones** specifically in the **financial context**, making it a much better choice than general-purpose models.

- Tokenize a paragraph and send it into the FinBERT model.
- The model returns **probabilities** for:
    - `positive` (index 0)
    - `neutral` (index 1)
    - `negative` (index 2)

2. VADER for Magnitude

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a **lexicon-based** sentiment tool included in the `nltk` package.

It doesn't give you a class like "positive" or "neutral." Instead, it gives a **compound score** between **-1 (very negative)** and **+1 (very positive)**.

For example:

- A sentence like "I'm somewhat concerned" → FinBERT might say "negative"
- A sentence like "We are extremely disappointed" → also "negative"

But VADER helps you **differentiate** between weak vs strong negativity.

- Break a paragraph into sentences.

- For each sentence, you compute VADER's `compound` score.
- Then, you add up the **absolute values** of these compound scores to get a **magnitude**:
    - `magnitude = sum(abs(sia.polarity_scores(sentence)['compound']) for sentence in paragraph)`

So:

- **FinBERT → sentiment direction (and weighted score)**
- **VADER → sentiment strength/magnitude**

3. API Ninjas: Getting the Transcript Data

API Ninjas is a platform that provides APIs for various data — in your case, you're using:

```
https://api.api-ninjas.com/v1/earningstranscript
```

This endpoint gives you:

- The **transcript** of a company's **earnings call**
- Filtered by `ticker`, `year`, and `quarter`

API Ninjas provides this cleanly via a simple HTTP request. Example:

- `url = f"https://api.api-ninjas.com/v1/earningstranscript?ticker=AAPL&year=2024&quarter=1"`

Backend sends this request using the `requests` library, with your API key.

Once having the transcript:

- Extract paragraphs (in `extract_paragraphs_from()`)
- Then analyze them one-by-one with FinBERT and VADER
- Then compute **average score** and **average magnitude**

System Service Setup for Flask Backend

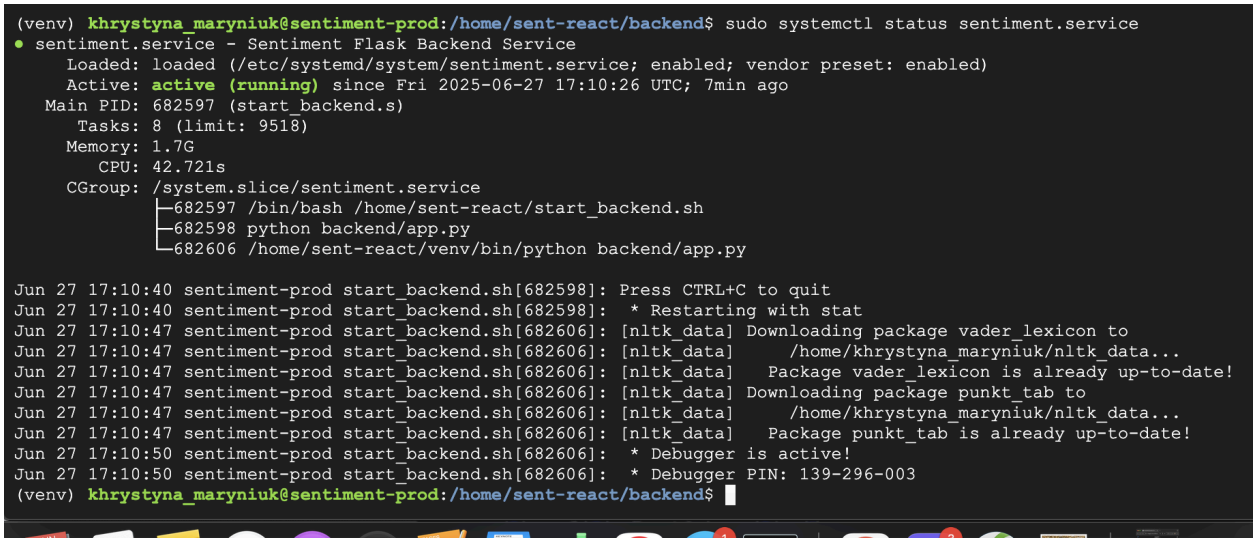
To make sure my Flask backend app runs all the time and starts automatically when the server restarts, I created a systemd service called sentiment.service.

- I made a service file in /etc/systemd/system/sentiment.service.
- The service runs a script start_backend.sh that turns on the Python virtual environment and starts the Flask app.
- The service runs under my user account (khrystyna_maryniuk) so it has the right permissions and doesn't need to run as root.

Here is an example of what the service file looks like:

```
[Unit]

Description=Sentiment Flask Backend Service

After=network.target

[Service]

User=khrystyna_maryniuk

WorkingDirectory=/home/sent-react/backend

ExecStart=/home/sent-react/start_backend.sh

Restart=always

[Install]

WantedBy=multi-user.target
```

- I reloaded systemd and enabled the service to start on boot.
- I started the service and checked that it was running without errors.
- I watched the logs to confirm the app started properly and all required packages loaded.
- I tested starting, stopping, and restarting the service with systemctl commands.


(venv) khrystyna_maryniuk@sentiment-prod:/home/sent-react/backend$ sudo systemctl status sentiment.service
● sentiment.service - Sentiment Flask Backend Service
Loaded: loaded (/etc/systemd/system/sentiment.service; enabled; vendor preset: enabled)
Active: active (running) since Fri 2025-06-27 17:10:26 UTC; 7min ago
Main PID: 682597 (start_backend.s)
Tasks: 8 (limit: 9518)
Memory: 1.7G
CPU: 42.721s
CGroup: /system.slice/sentiment.service
├─682597 /bin/bash /home/sent-react/start_backend.sh
├─682598 python backend/app.py
└─682606 /home/sent-react/venv/bin/python backend/app.py
Jun 27 17:10:40 sentiment-prod start_backend.sh[682598]: Press CTRL+C to quit
Jun 27 17:10:40 sentiment-prod start_backend.sh[682598]: * Restarting with stat
Jun 27 17:10:47 sentiment-prod start_backend.sh[682606]: [nltk_data] Downloading package vader_lexicon to
Jun 27 17:10:47 sentiment-prod start_backend.sh[682606]: [nltk_data] /home/khrystyna_maryniuk/nltk_data...
Jun 27 17:10:47 sentiment-prod start_backend.sh[682606]: [nltk_data] Package vader_lexicon is already up-to-date!
Jun 27 17:10:47 sentiment-prod start_backend.sh[682606]: [nltk_data] Downloading package punkt_tab to
Jun 27 17:10:47 sentiment-prod start_backend.sh[682606]: [nltk_data] /home/khrystyna_maryniuk/nltk_data...
Jun 27 17:10:47 sentiment-prod start_backend.sh[682606]: [nltk_data] Package punkt_tab is already up-to-date!
Jun 27 17:10:50 sentiment-prod start_backend.sh[682606]: * Debugger is active!
Jun 27 17:10:50 sentiment-prod start_backend.sh[682606]: * Debugger PIN: 139-296-003
(venv) khrystyna_maryniuk@sentiment-prod:/home/sent-react/backend$