

ICS 2309: Commercial Programming

Feeler: Micro - Service and Micro - Service architecture

SCT211-0535/2022

AKECH DAU ATEM

FEELER

MICRO-SERVICES

1 CONTEXT

A microservice is a service with one, and only one, very narrowly focused capability that can be exposed to the rest of the system. I am identifying the individual components that can be thought of as microservices from Feeler – our sentiment analysis platform using NLP.

2 OUR MICROSERVICE

ARCHITECTURE:

Microservices in Feeler, basically decomposes our application into small independent services that communicate via defined interfaces. I have listed out below microservices; these components are decoupled from parts of the NLP platform and can be seen as independent components except for when they need to communicate with the others.

Microservice	What It Does
1. Auth Service	User authentication, JWT token generation, role-based access control (RBAC)
2. Text Processing Service	Handles text submission, preprocessing, and routing to appropriate NLP models
3. Sentiment Analysis Service	Runs Sentient73 (CNN) and Twitter-roBERTa models for sentiment prediction
4. Analytics Service	Generates visualizations (word clouds, trends), stores historical data, and provides insights

feeler table 1: microservices

3 API DEFINITIONS

The examples below are sample API endpoints in the Feeler platform and we have given sample requests that would be made in an environment of doing **sentiment analysis**.

AUTH SERVICE

1. **Endpoint:** `/api/register/` (POST) - registers a new user (Admin/Analyst/Viewer).

Request Body: { "username": "feelrsu", "password": "mood73", "email": "muo.byte@gmail.com", "role": "analyst" }

Response: { "token": "JWT_TOKEN", "user_id": 123 }

2. **Endpoint:** `/api/token/` (POST) - authenticates users and returns a JWT.

Request Body: { "username": "user1", "password": "pass123" }

Response: { "token": "JWT_TOKEN" }

TEXT PROCESSING SERVICE

Endpoint: `/api/analyze` (POST) - accepts text input and returns a sentiment prediction.

Request Body: { "text": "I love this product!", "model_type": "roberta" }

Response: { "sentiment": "positive", "confidence": 0.92 }

SENTIMENT ANALYSIS SERVICE

Internal Endpoint (REST): `api/predict/` - receives preprocessed text from Text Processing Service and returns sentiment.

Request Body: { "text": "I am in awe of what machine learning is capable of my guy; love it!" }

Response: { "label": "positive", "score": 0.87 }

ANALYTICS SERVICE

Endpoint: `api/history/` (GET) - retrieves past sentiment analyses for a user.

Query Params: `?user_id=123&limit=5`

Response:

```
{
  "history": [
    { "text": "Great service!", "sentiment": "positive", "timestamp": "2025-04-10" },
    { "text": "Slow delivery.", "sentiment": "negative", "timestamp": "2025-04-09" }
  ]
}
```

3 DATA OWNERSHIP

The **Auth** Service owns *user credentials* and *JWT blacklist* for token generation and expiration. The **Text Processing** Service is a temporary *text/data cache* for frequent repetitive requests. The **Sentiment Analysis** service hosts *model weights and predictions* data. And the **Analytics** Microservice has *sentiment and visualization* data.

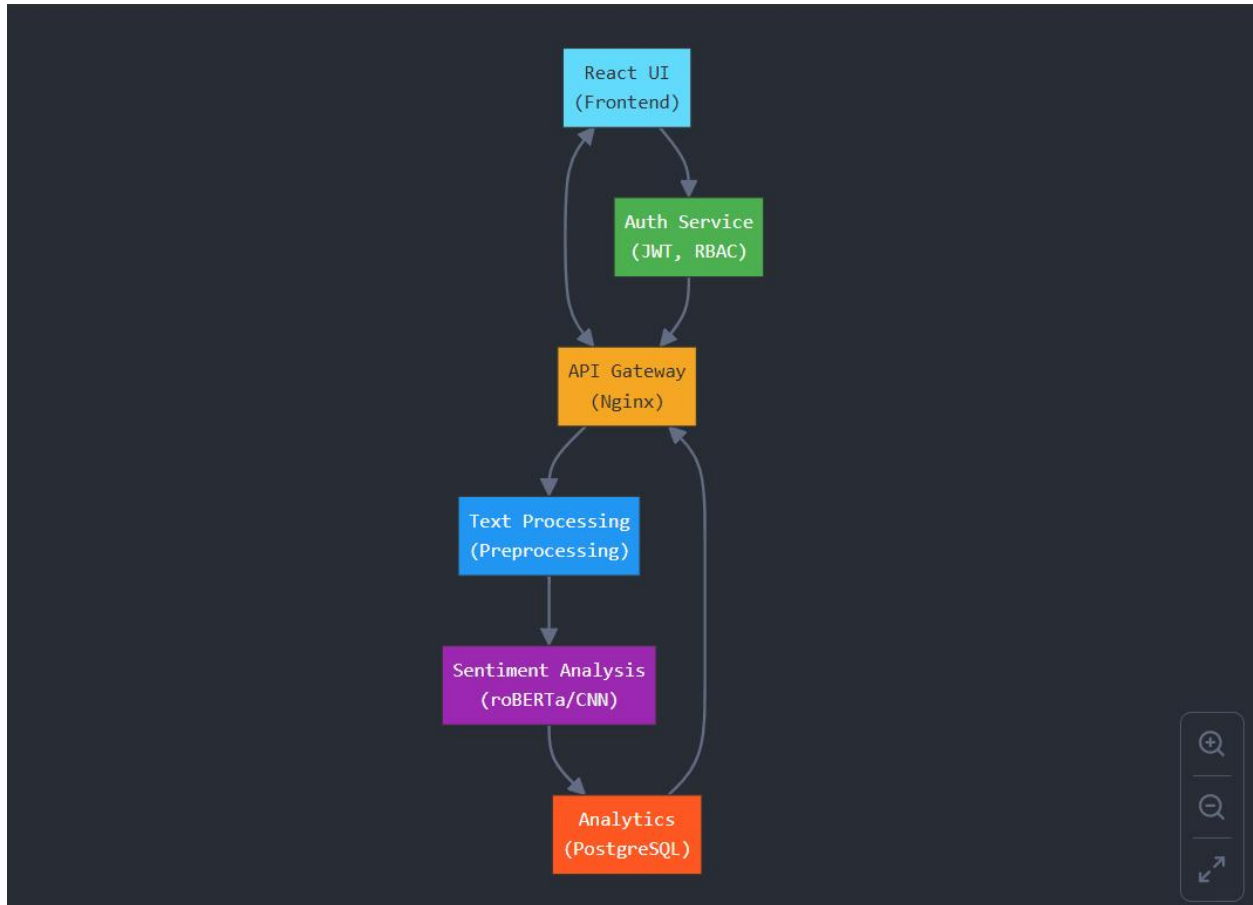
4 COMMUNICATION PATTERNS

The table below has brief descriptions of the communication patterns of Feeler. This has to do with the data flow in the platform.

Interaction	Protocol	Sync/Async
Auth → Text Processing	REST (JWT validation)	Synchronous
Text Processing → Sentiment Analysis	REST (low-latency)	Synchronous
Analytics → Database	PostgreSQL (via Django ORM)	Synchronous
Notification Service	RabbitMQ/Kafka	Asynchronous

feeler table 2: comms-patterns

5 FEELER MICROSERVIVCE ARCHITECTURE (DIAGRAM)



feeler figure (3): microservices arch

Benefits of This Design:

There is merit to the thinking that this design has benefits, namely:

- **Scalability:** each service (e.g., sentiment analysis) can scale independently.
- **Fault Isolation:** If auth Service fails, text processing still works.
- **Tech Flexibility:** use fast APIs for auth, TensorFlow for models.
- **CI/CD Friendly:** smaller services = faster deployments.

3 POSTMAN – API CALLS TO INFERENCE ENDPOINT

We generate a JWT token first since this is a protected endpoint. This is passed in to postman as a bearer token as part of the request header.

This is the endpoint that was tested in the Postman client app and this is what was done:

1. Create a new Postman Request of Type POST
2. Add a Bearer Token(Valid) under Authorization Tab.
3. Add the Header Content-Type: application/json
4. Request Body: {"text": "I am in awe of what machine learning is capable of my guy; love it!"}

Endpoint: POST /api/predict

Headers: Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2t1bI90eXBlljoiYWNjZXNzliwiZXhwIjoxNzQ0NzA0NTkxLCJpYXQiOiE3NDQ3MDI3OTEslmp0aSI6ImU0MDRIYTJmZGMwMjQ1YzVhZDY1MGM0MDk1OGZhYzkwliwidXNlci9pZCI6M30.XzKKqYVpiRWmDG_udtnzzII33LcjhScFLFqg2cikVqU

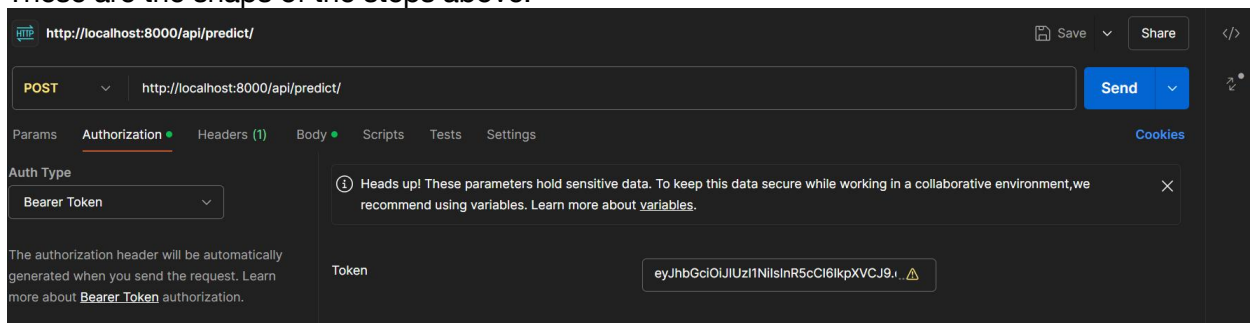
Body:


{"text": "I am in awe of what machine learning is capable of my guy; love it!"}


Response:


```
{
  "sentiment": "positive",
  "confidence": 0.7860926985740662
}
```

These are the snaps of the steps above:



 http://localhost:8000/api/predict/

 Save

 Share

POST

http://localhost:8000/api/predict/

Send

Params

Authorization

Headers (1)

Body

Scripts

Tests

Settings

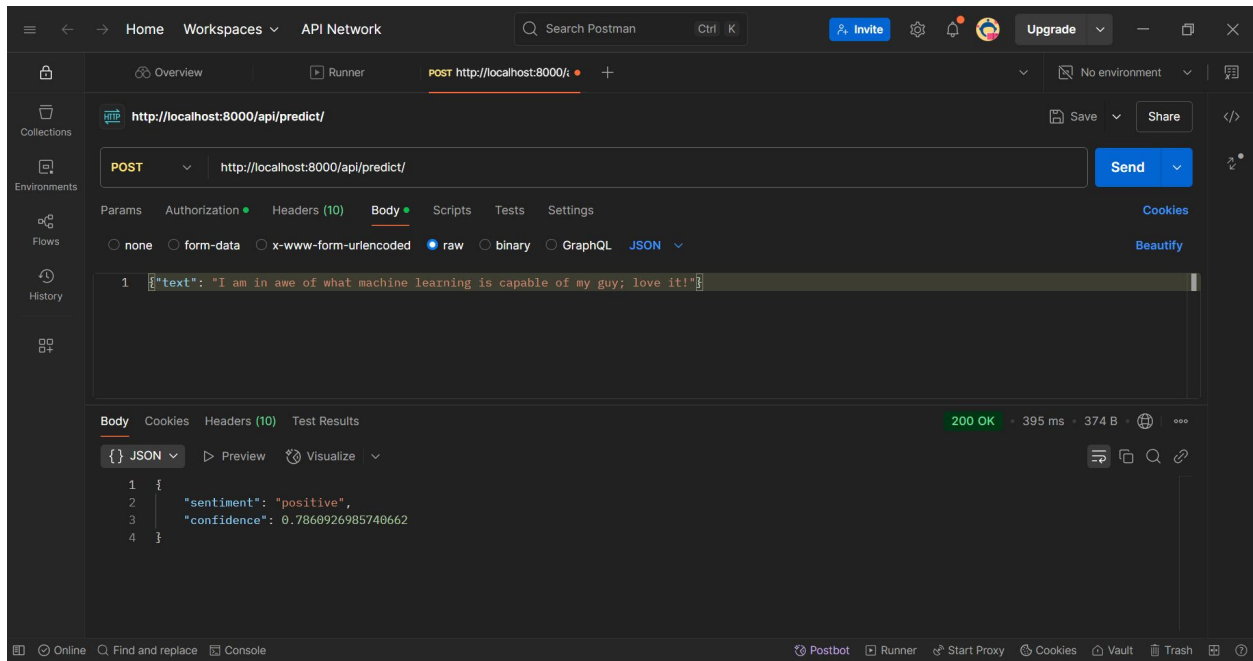
Cookies

Headers

	Key	Value	Description		Bulk Edit	Presets
<input checked="" type="checkbox"/>	Content-Type	application/json				
	Key	Value	Description			



RESPONSE:



END