**Different Layers in the Sentiment Analysis project:**

1. **Presentation Layer:**

API endpoint (/sentiment)

Basic request/response models (SentimentRequest, SentimentResponse)

Input validation

1. **Service Layer:**

Business logic

ML service communication

1. **Infrastructure Layer:**

Python ML service integration

Error handling

Logging

Configuration

**Architecture**

Client Request

↓

[C# Gateway API]

├── Presentation Layer

│ ├── API Endpoints

│ ├── Request/Response Models

│ └── Input Validation

│

├── Service Layer

│ ├── Interface: ISentimentService

│ ├── Implementation: SentimentService

│ └── Business Logic

│

├── Infrastructure Layer

│ ├── HTTP Client

│ ├── Configuration

│ └── Error Handling

↓

[Python ML Service]

├── FastAPI

├── ML Model

└── Prediction Logic

**Let me explain how the layers interact in our sentiment analysis system:**

**1.Flow of Request:**

Client → Presentation Layer

- Client sends text for analysis

- Presentation Layer validates input

**Between Layers:**

Presentation → Service Layer

- Passes validated request

- Service layer processes business logic

Service → Infrastructure Layer

- Makes HTTP calls to Python service

- Handles timeouts/retries

**2. Response Flow:**

Python ML Service → Infrastructure Layer

- Returns sentiment prediction

- Infrastructure handles errors

Infrastructure → Service → Presentation → Client

- Each layer adds its processing

- Final response returned to client

**Let's test service implementation using Swagger:**

**Run your API:**

1. dotnet run

**Test steps:**

Open: http://localhost:5067/swagger/index.html

Click on POST /sentiment

Click "Try it out"

**Test these scenarios:**

Valid input:

{

"text": "I love this new service architecture!"

}

Invalid input (empty):

{

"text": ""

}

**You should see:**

Valid input: 200 OK with sentiment analysis

Invalid input: 400 Bad Request with error message

**Details of the Infrastructure Layer**

1. Configuration:

We're preparing to connect our C# API to a Python ML service

The Python service will run on a different port (8000)

We need to store these connection settings somewhere

MLServiceOptions Class:

public class MLServiceOptions

{

public const string Section = "MLService"; *// Name in config file*

public string BaseUrl { get; set; } = string.Empty; *// URL of Python service*

public int TimeoutSeconds { get; set; } = 30; *// How long to wait for response*

}

Add settings to appsettings.json:Configuration file that stores these settings

{

"MLService": {

"BaseUrl": "http://localhost:8000",

"TimeoutSeconds": 30

}

}

**Now, Lets add HTTP client to communicate with our Python ML service.**

1. What is an HTTP Client?

* It's a tool that makes HTTP requests to other services
* Like a browser making requests to websites
* In our case, it will call our Python ML service

1. Why do we need it?

* Our C# API needs to send text to Python service
* Python service will analyze sentiment
* HTTP client handles this communication
* Example Flow:
* C# API (sends) → HTTP Client → Python Service
* C# API ← HTTP Client (receives) ← Python Service Response

Python Service

* Project Setup:

PythonMLService/

├── app/

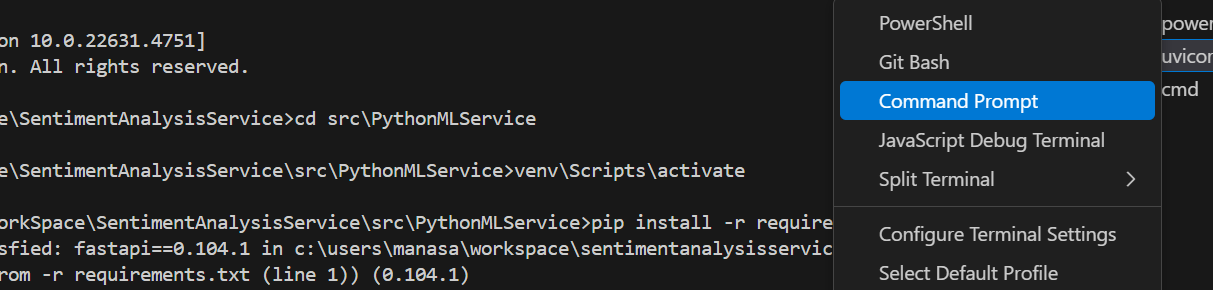
│ ├── \_\_init\_\_.py

│ ├── main.py

│ └── ml\_model.py

└── requirements.txt

Let's run the FastAPI service. Here are the steps:



1. First, create a Python virtual environment and install dependencies:

*# Create and activate virtual environment*

python -m venv venv

*# On Windows:*

venv\Scripts\activate

*# On Mac/Linux:*

source venv/bin/activate

*# Install dependencies*

pip install -r requirements.txt

1. Run the service:

*# From the PythonMLService directory*

uvicorn app.main:app --reload --port 8000

1. Test the service:

* Open browser: http://localhost:8000/docs
* You'll see the Swagger UI for FastAPI

Let me explain how the sentiment analysis model works:

1.The Pipeline:

self.analyzer = pipeline("sentiment-analysis")

Uses Hugging Face's transformers library

Loads a pre-trained BERT model

Optimized for sentiment classification

Automatically downloads model on first use

2.Analysis Process:

result = self.analyzer(text)[0]

Takes input text

Tokenizes (splits into words/subwords)

Processes through neural network

Returns prediction scores

3.Output Format:

{

"sentiment": "Positive" or "Negative",

"score": 0.0 to 1.0 *# Confidence score*

}

Let's connect our C# API with the Python service:

1.First, ensure both services are running:

*# Terminal 1 (Python Service)*

uvicorn app.main:app --reload --port 8000

*# Terminal 2 (C# API)*

dotnet run

2.Verify your C# appsettings.json has the correct Python service URL:

{

"MLService": {

"BaseUrl": "http://localhost:8000",

"TimeoutSeconds": 30

}

}

3.Test the integration:

Open C# Swagger UI: http://localhost:5067/swagger

Send a request to the sentiment endpoint

The C# API should forward it to Python service

