**Київський політехнічний інститут імені Ігоря Сікорського**

**Фізико-технічний інститут**

**Проектування розподілених систем**

**Лабораторна робота №1**

Варіант 1

**Виконала:**

Студентка групи ФБ-42мп

Алькова Аліна

**Task 1 - Базова архітектура мікросервисів**

Це завдання є основою для наступних завдань, у яких будуть додаватись додаткові програмні засоби та функціональність.

У завданні необхідно реалізувати три мікросервіси, взаємодія між якими відбувається на основі HTTP-протоколу, бажано на основі парадігми REST API (можливо також використання gRPC-протоколу).

Архітектура складається з трьох мікросервисів:

* ***facade-service*** - приймає POST/GET запити від клієнта
* ***logging-service*** - зберігає у пам’яті всі повідомлення які йому надходять, та може повертати їх
* ***messages-service*** - поки виступає у ролі заглушки, при звернені до нього повертає статичне повідомлення

**Базова функціональність системи (5 балів)**

Клієнт взаємодіє з ***facade-service*** через HTTP POST та GET запити. У якості клієнта може бути *curl*, *Postman*, браузер у Dev mode та інше

Код доступний за посилання на гітхаб.

**Facade-service — приймає POST/GET запити від клієнта**. У коді FacadeServer є маршрут @server.route("/entry", methods=["POST", "GET"]), який обробляє POST і GET запити від клієнта.

**Logging-service — зберігає у пам’яті всі повідомлення, які йому надходять, та може повертати їх**. Він зберігає повідомлення у словнику records (хеш-таблиця в пам’яті) та повертає їх через ", ".join(records.values()) при GET-запиті.

**Messages-service — поки виступає у ролі заглушки, при зверненні повертає статичне повідомлення** "Placeholder active, no features yet" при GET-запиті.

**1. FacadeService** підтримує стандартні HTTP POST і GET запити через Flask, що сумісно з curl, Postman, браузером тощо. У FacadeService обробка request.form.get("data") отримує текстове повідомлення (content) від клієнта. Використовується unique\_key = str(uuid.uuid4()) для генерації унікального ідентифікатора. FacadeService відправляє POST-запит через requests.post(tracker\_url, data=packet), де packet = {"key": unique\_key, "data": content}. У FacadeService повертається f"Tracker says: {tracker\_data.text} | Placeholder says: {placeholder\_data.text}", що є конкатенацією відповідей у вигляді одного рядка.

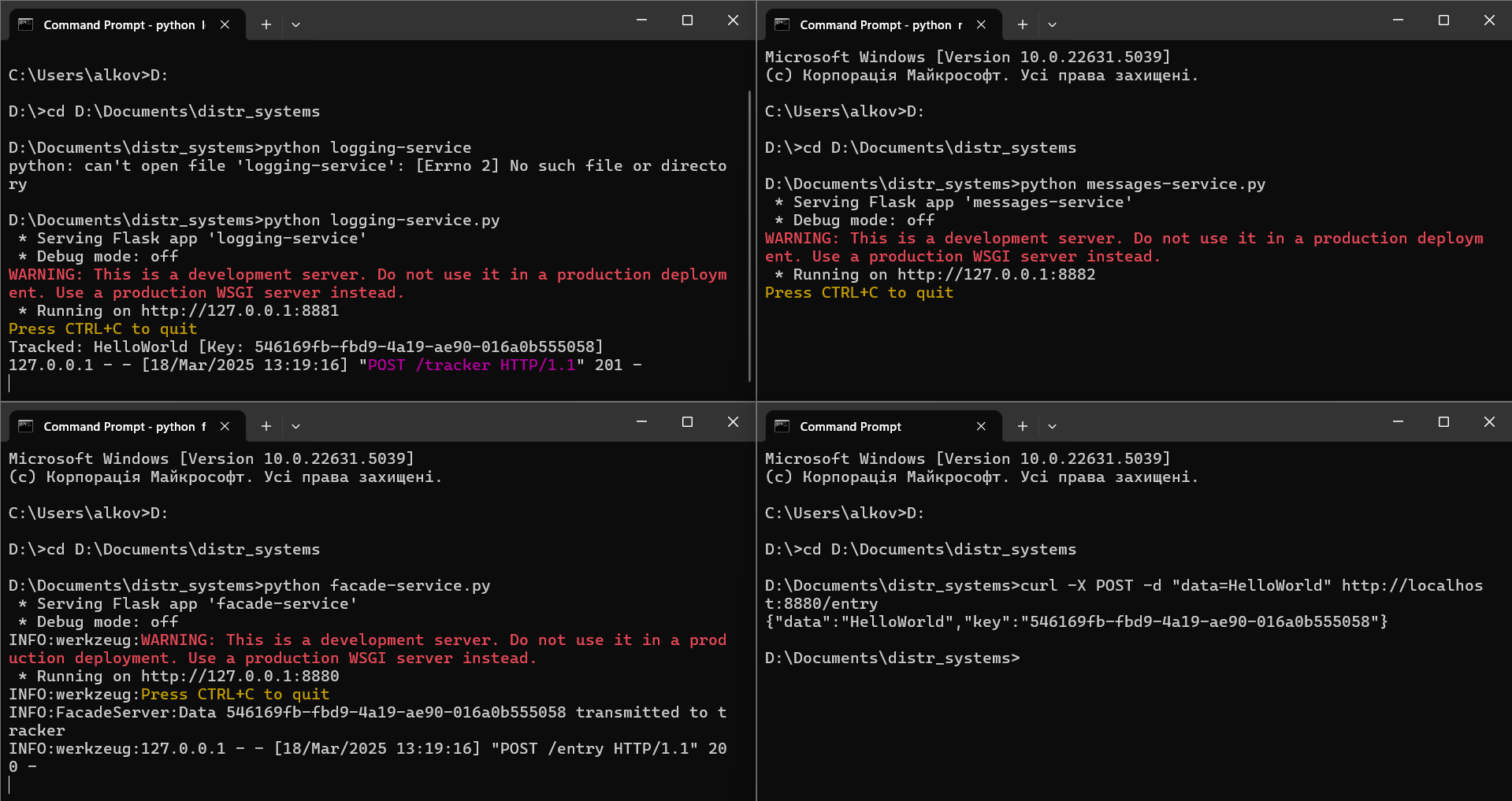
from flask import Flask, request, jsonify  
import requests  
import uuid  
import time  
import logging  
  
server = Flask(\_\_name\_\_)  
  
# Налаштування логів  
logging.basicConfig(level=logging.INFO)  
log = logging.getLogger("FacadeServer")  
  
tracker\_url = "http://localhost:8881/tracker"  
placeholder\_url = "http://localhost:8882/placeholder"  
  
# Функція для повторних спроб  
def push\_to\_tracker(payload, max\_attempts=3, wait\_time=1):  
 for try\_num in range(max\_attempts):  
 try:  
 result = requests.post(tracker\_url, data=payload, timeout=4)  
 if result.status\_code == 201:  
 log.info(f"Data {payload['key']} transmitted to tracker")  
 return result  
 except (requests.exceptions.ConnectionError, requests.exceptions.Timeout) as error:  
 log.warning(f"Try {try\_num + 1}/{max\_attempts} failed for {payload['key']}: {error}")  
 if try\_num < max\_attempts - 1:  
 time.sleep(wait\_time)  
 continue  
 log.error(f"Transmission failed for {payload['key']} after {max\_attempts} attempts")  
 return None  
  
@server.route("/entry", methods=["POST", "GET"])  
def process():  
 if request.method == "POST":  
 content = request.form.get("data")  
 if not content:  
 return jsonify({"error": "Empty data"}), 400  
  
 unique\_key = str(uuid.uuid4()) #зміна на фіксоване значення для тесту дедуплікації  
 packet = {"key": unique\_key, "data": content}  
  
 # Використовуємо retry  
 outcome = push\_to\_tracker(packet)  
 if outcome is None:  
 return jsonify({"error": "Tracker unreachable"}), 503  
  
 return jsonify({"key": unique\_key, "data": content}), 200  
  
 elif request.method == "GET":  
 tracker\_data = requests.get(tracker\_url)  
 placeholder\_data = requests.get(placeholder\_url)  
 combined = f"Tracker says: {tracker\_data.text} | Placeholder says: {placeholder\_data.text}"  
 return combined, 200  
  
 else:  
 return jsonify({"error": "Invalid method"}), 405  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 server.run(port=8880)

2. **У LoggingService** пара зберігається як records[identifier] = info, де identifier — ключ, а info — повідомлення. Вивід у консоль: print(f"Tracked: {info} [Key: {identifier}]"). LoggingService повертає ", ".join(records.values()), що є рядком із усіма повідомленнями (без ключів).

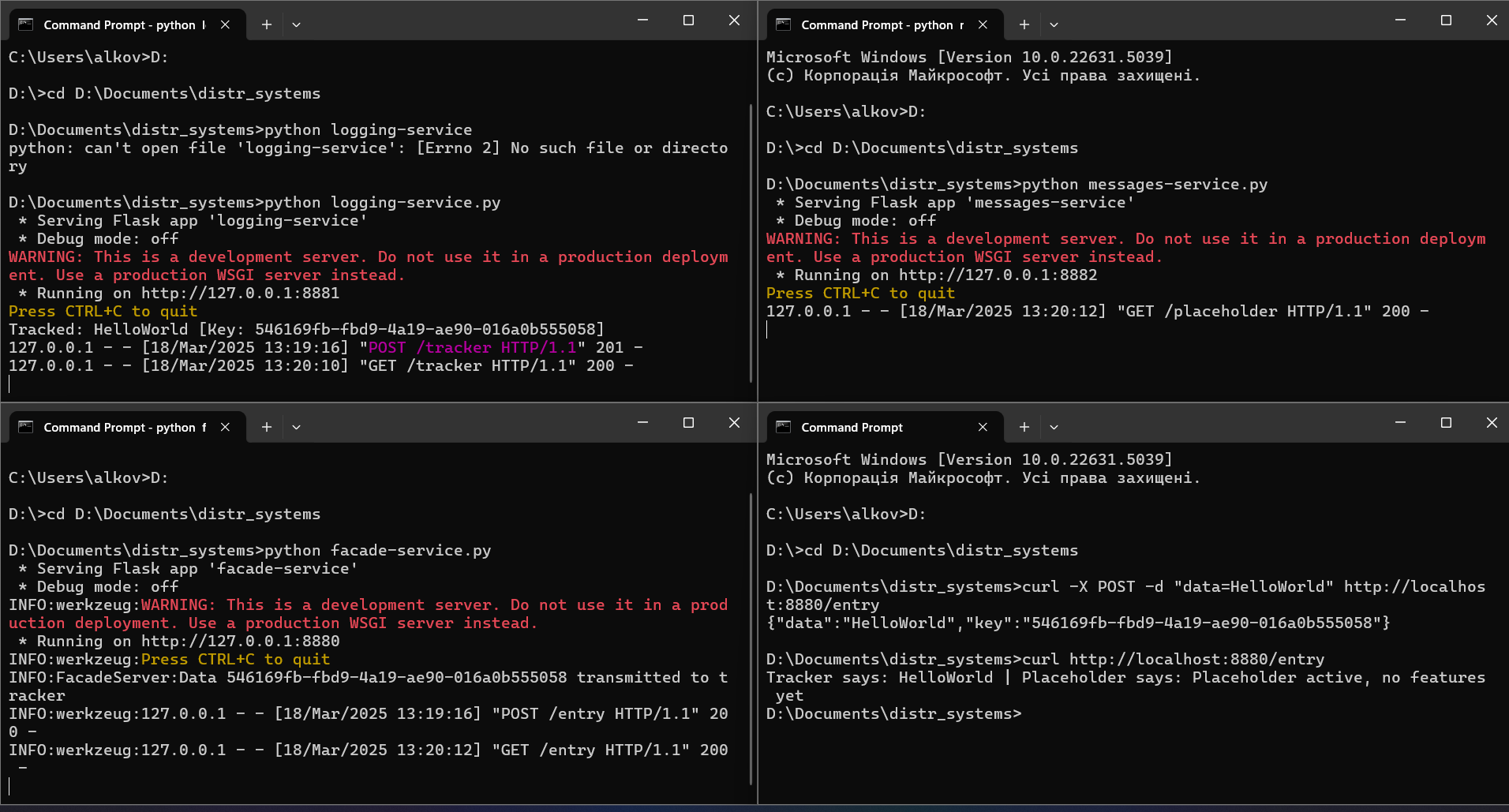
from flask import Flask, request, jsonify  
  
tracker = Flask(\_\_name\_\_)  
  
# Словник для зберігання даних  
records = {}  
  
@tracker.route("/tracker", methods=["POST", "GET"])  
def handle():  
 if request.method == "POST":  
 identifier = request.form.get("key")  
 info = request.form.get("data")  
  
 if not (identifier and info):  
 return jsonify({"error": "Missing key or data"}), 400  
  
 # Дедуплікація  
 if identifier in records:  
 print(f"Repeated entry skipped: {identifier}")  
 return jsonify({"status": "Skipped repeat"}), 201  
  
 records[identifier] = info  
 print(f"Tracked: {info} [Key: {identifier}]")  
 return jsonify({"status": "Entry recorded"}), 201  
  
 elif request.method == "GET":  
 return ", ".join(records.values()) # Об'єднуємо дані комами  
  
 else:  
 return jsonify({"error": "Method unsupported"}), 405  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 tracker.run(port=8881)

3. **MessagesService** повертає "Placeholder active, no features yet", що відповідає вимозі статичного тексту.

from flask import Flask, request, jsonify  
  
stub = Flask(\_\_name\_\_)  
  
@stub.route("/placeholder", methods=["GET"])  
def respond():  
 if request.method == "GET":  
 return "Placeholder active, no features yet"  
 else:  
 return jsonify({"error": "Only GET allowed"}), 405  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 stub.run(port=8882)

**POST-запити:**

* Клієнт відправив POST-запит із повідомленням (data=HelloWorld).
* FacadeService згенерував UUID (<UUID>).
* Пара {key: <UUID>, data: HelloWorld} передана до LoggingService.
* LoggingService зберіг повідомлення в records і вивев його в консоль.

**GET-запити:**

* Клієнт відправив GET-запит до FacadeService.
* FacadeService надіслав GET-запити до LoggingService і MessagesService.
* LoggingService повернув рядок із повідомленнями (HelloWorld).
* MessagesService повернув статичний текст (Placeholder active, no features yet).
* FacadeService сконкатенував відповіді та повернув їх клієнту.

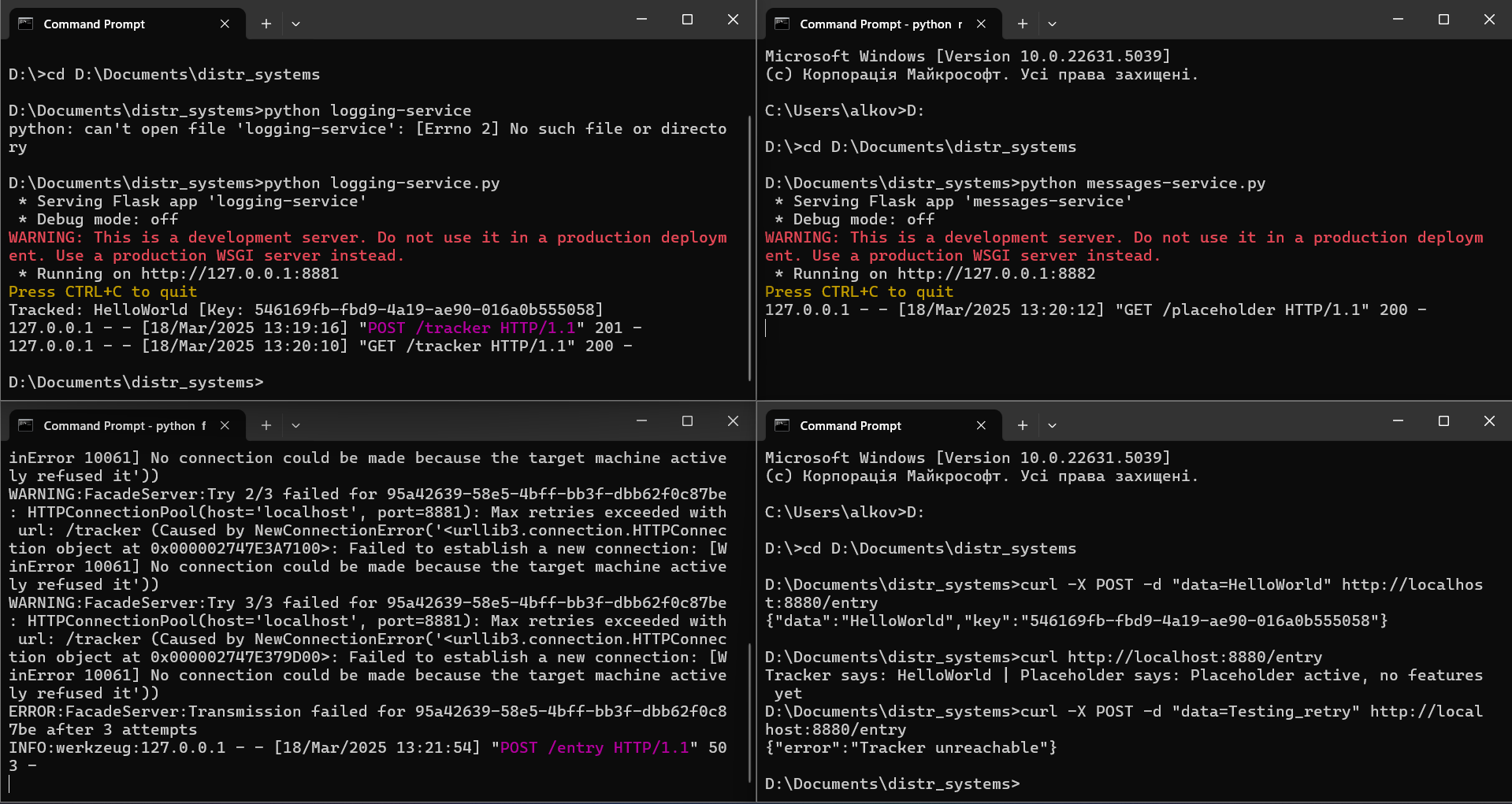
**Додатковий функціонал системи (+5 балів)**

* Наявність механізму retry (+2 бали):
  + У випадку відсутності звʼязку, затримки або відстуності відповіді від ***logging-service***, має застосовуватись механізм *retry*, який пробує повторити цю саму операцію по передачі повідомлення
  + Одночасно з застосуванням механізму *retry*, на ***logging-service*** має застосовуватись deduplication для повідомлень (*exactly once delivery*), для перевірки того що одне й те саме повідомлення не повторилось декілька разів. З протоколу має бути видно, що механізм *retry* був протестований

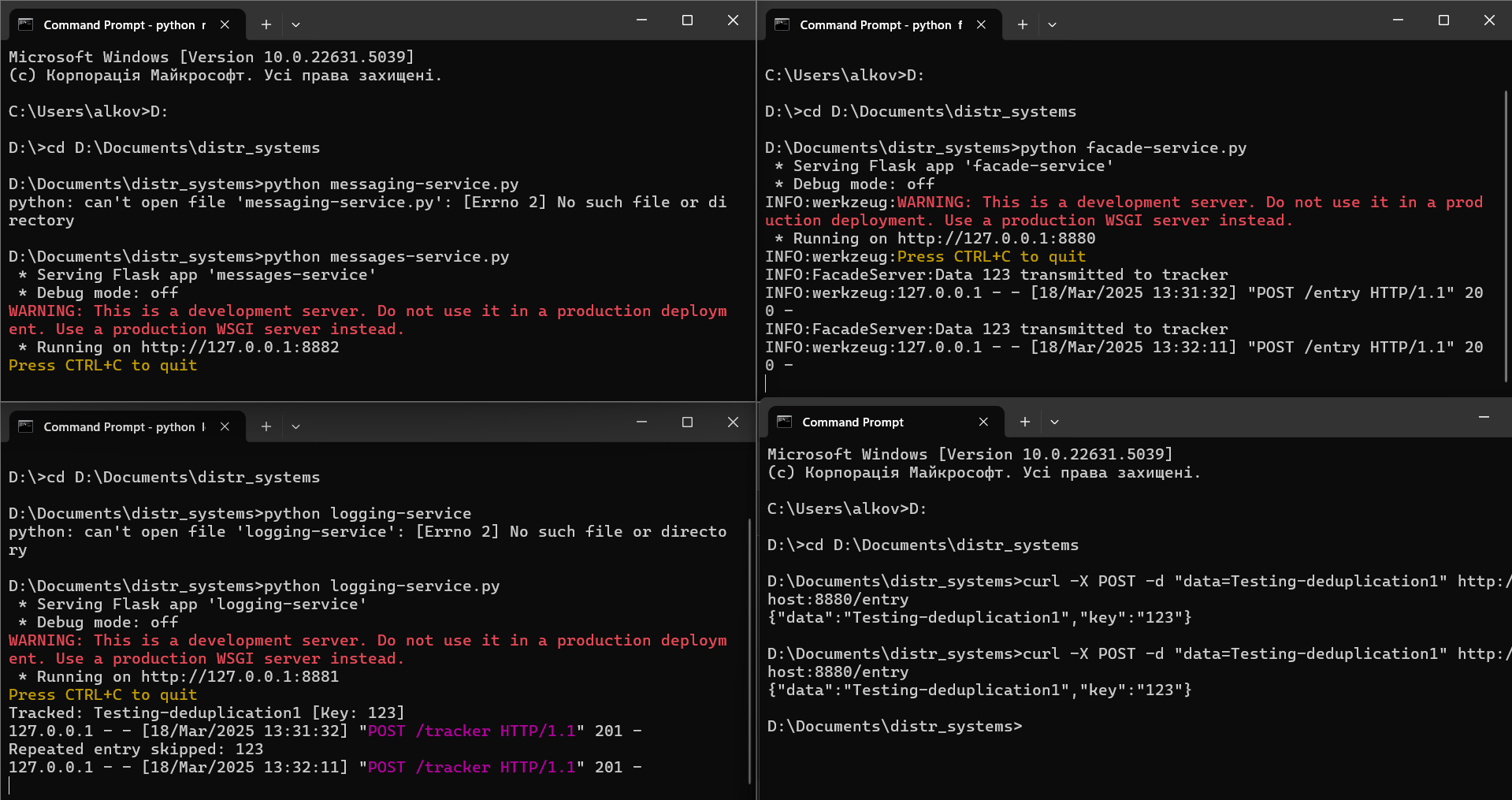
Вже реалізовано вище у коді.

Протестуємо:

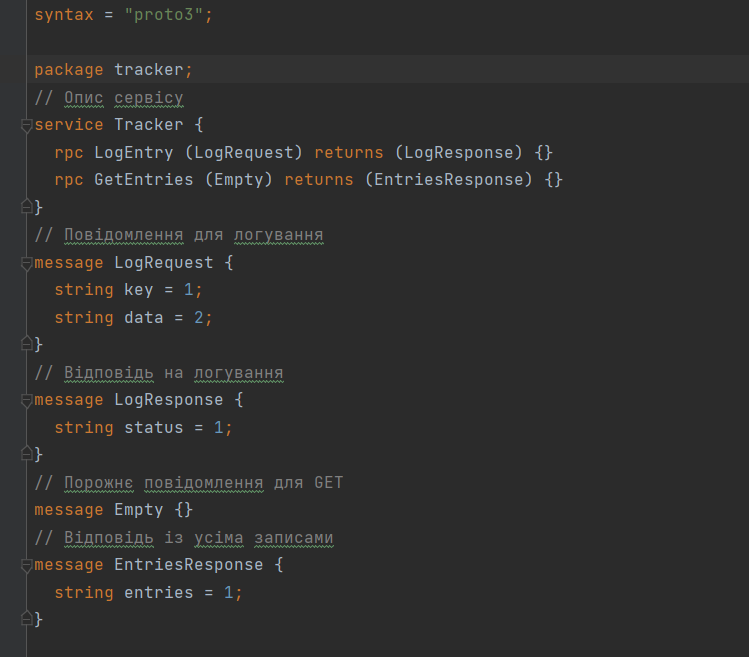
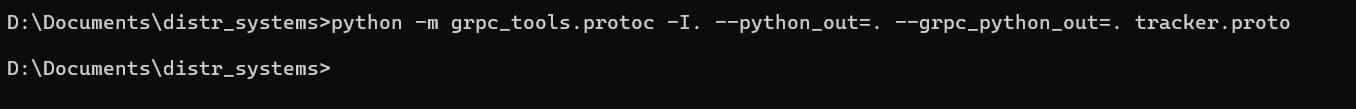
* LoggingService недоступний, тому FacadeService застосував механізм retry.
* Виконано 3 спроби з інтервалом 1 секунда.
* Логи показують, що retry був протестований (видно всі спроби та помилку).



**Дедуплікація**

* Перше повідомлення збережено в records.
* Друге повідомлення з тим самим key (fixed-key-123) проігноровано через дедуплікацію.
* У консолі видно, що повторне повідомлення пропущено.

**Використання GRPC для взаємодії між сервісами (+3 бали)**

* + Для взаємодій між ***facade-service*** та ***logging-service***, замість протоколу HTTP (та POST/GET запитів) використати протокол GRPC

Тепер Для POST (LogEntry) між FacadeService і LoggingService використовується gRPC через tracker\_stub.LogEntry.

**FacadeService**. POST: Як і раніше, використовує gRPC через tracker\_stub.LogEntry. GET: Замість requests.get("http://localhost:8881/tracker") використовується gRPC-запит tracker\_stub.GetEntries.

rom flask import Flask, request, jsonify  
import requests  
import uuid  
import time  
import logging  
import grpc  
import tracker\_pb2  
import tracker\_pb2\_grpc  
  
server = Flask(\_\_name\_\_)  
  
logging.basicConfig(level=logging.INFO)  
log = logging.getLogger("FacadeServer")  
  
# gRPC канал до logging  
channel = grpc.insecure\_channel('localhost:50051')  
tracker\_stub = tracker\_pb2\_grpc.TrackerStub(channel)  
  
placeholder\_url = "http://localhost:8882/placeholder"  
  
# Функція з retry для gRPC LogEntry  
def push\_to\_tracker\_grpc(key, data, max\_attempts=3, wait\_time=1):  
 payload = tracker\_pb2.LogRequest(key=key, data=data)  
 for try\_num in range(max\_attempts):  
 try:  
 response = tracker\_stub.LogEntry(payload, timeout=4)  
 log.info(f"Data {key} transmitted to tracker via gRPC: {response.status}")  
 return response  
 except grpc.RpcError as error:  
 log.warning(f"Try {try\_num + 1}/{max\_attempts} failed for {key}: {error.details()}")  
 if try\_num < max\_attempts - 1:  
 time.sleep(wait\_time)  
 continue  
 log.error(f"Transmission failed for {key} after {max\_attempts} attempts")  
 return None  
  
# Функція для gRPC GetEntries  
def fetch\_tracker\_entries():  
 try:  
 response = tracker\_stub.GetEntries(tracker\_pb2.Empty(), timeout=4)  
 return response.entries  
 except grpc.RpcError as error:  
 log.error(f"Failed to fetch tracker entries: {error.details()}")  
 return "Tracker unavailable"  
  
@server.route("/entry", methods=["POST", "GET"])  
def process():  
 if request.method == "POST":  
 content = request.form.get("data")  
 if not content:  
 return jsonify({"error": "Empty data"}), 400  
  
 unique\_key = str(uuid.uuid4())  
  
 outcome = push\_to\_tracker\_grpc(unique\_key, content)  
 if outcome is None:  
 return jsonify({"error": "Tracker unreachable"}), 503  
  
 return jsonify({"key": unique\_key, "data": content}), 200  
  
 elif request.method == "GET":  
 tracker\_data = fetch\_tracker\_entries() # gRPC-запит до TrackerService  
 placeholder\_data = requests.get(placeholder\_url) # HTTP до PlaceholderService  
 combined = f"Tracker says: {tracker\_data} | Placeholder says: {placeholder\_data.text}"  
 return combined, 200  
  
 else:  
 return jsonify({"error": "Invalid method"}), 405  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 server.run(port=8880)

**У LoggingService** видалено Flask, тепер це чистий gRPC-сервер на порту 50051. Підтримує два методи: LogEntry (POST) і GetEntries (GET).

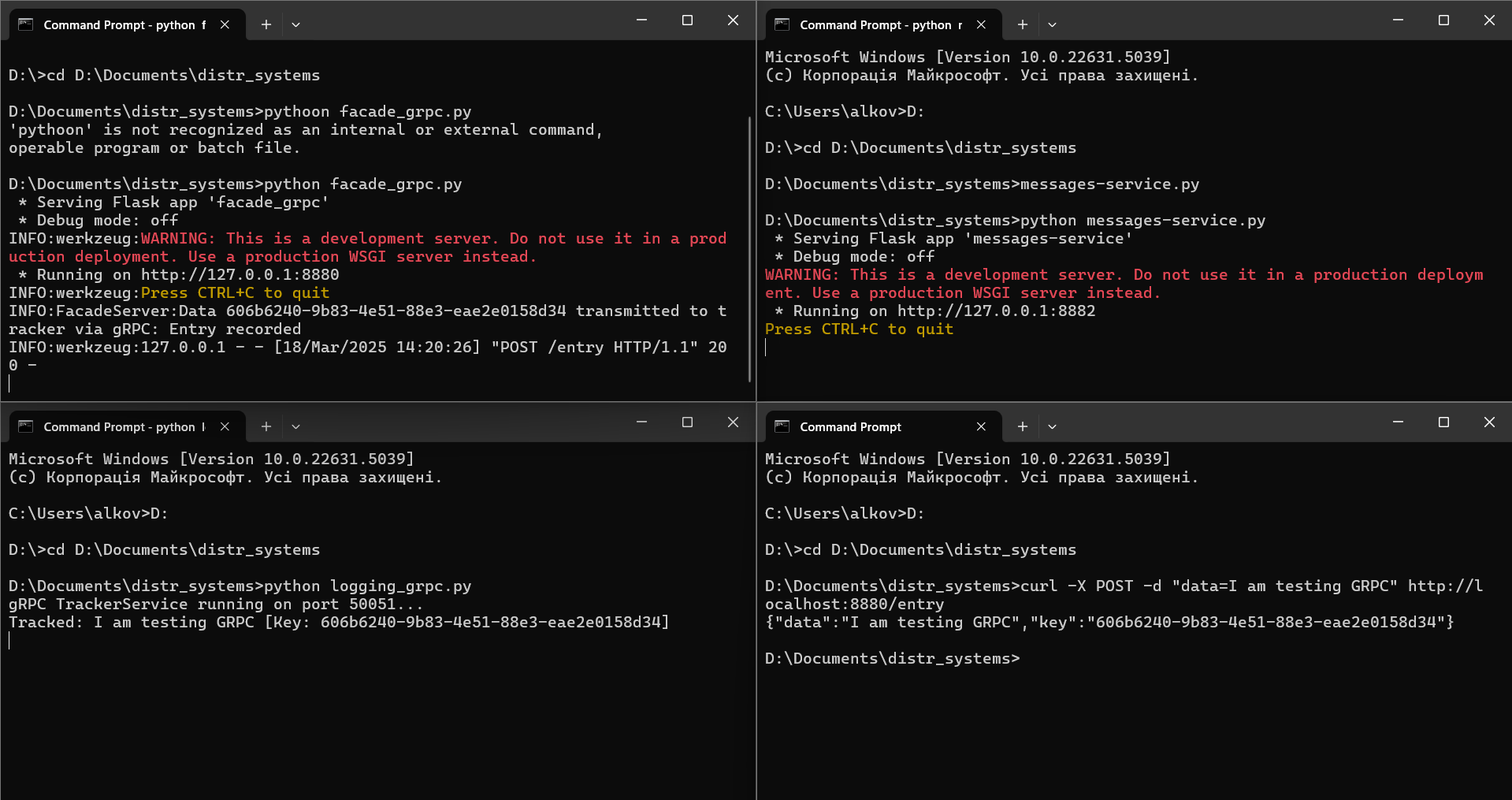
import grpc  
from concurrent import futures  
import tracker\_pb2  
import tracker\_pb2\_grpc  
  
# Словник для зберігання даних  
records = {}  
  
class TrackerServicer(tracker\_pb2\_grpc.TrackerServicer):  
 def LogEntry(self, request, context):  
 identifier = request.key  
 info = request.data  
  
 if not (identifier and info):  
 context.set\_code(grpc.StatusCode.INVALID\_ARGUMENT)  
 context.set\_details("Missing key or data")  
 return tracker\_pb2.LogResponse(status="Error")  
  
 # Дедуплікація  
 if identifier in records:  
 print(f"Repeated entry skipped: {identifier}")  
 return tracker\_pb2.LogResponse(status="Skipped repeat")  
  
 records[identifier] = info  
 print(f"Tracked: {info} [Key: {identifier}]")  
 return tracker\_pb2.LogResponse(status="Entry recorded")  
  
 def GetEntries(self, request, context):  
 return tracker\_pb2.EntriesResponse(entries=", ".join(records.values()))  
  
def serve():  
 server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))  
 tracker\_pb2\_grpc.add\_TrackerServicer\_to\_server(TrackerServicer(), server)  
 server.add\_insecure\_port('[::]:50051')  
 print("gRPC TrackerService running on port 50051...")  
 server.start()  
 server.wait\_for\_termination()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 serve()

**MessagesService** залишився на HTTP, оскільки вимога стосується лише FacadeServer ↔ LoggingService.

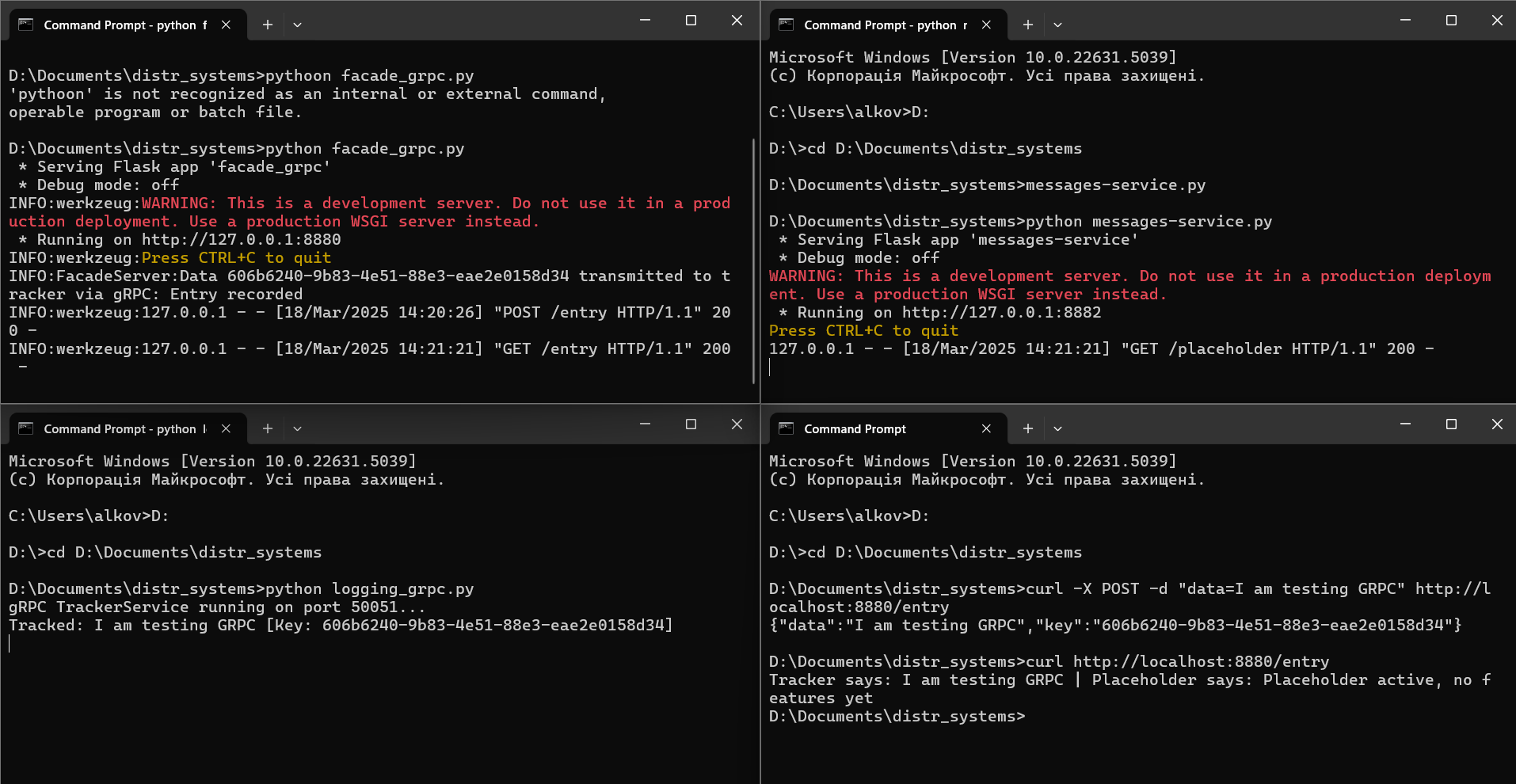
**Retry і Deduplication**: Retry перенесено в gRPC-контекст із обробкою grpc.RpcError. Дедуплікація залишилася в LoggingService.

LoggingService запускає gRPC-сервер на порту 50051 і Flask на 8881 одночасно. FacadeService підключається до gRPC через localhost:50051.

**POST:**



**GET:**



**Retry:**

