**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 2

**З дисципліни:** *“Архітектура та проєктування програмного забезпечення”*

**На тему:** *“Використання брокерів повідомлень для організації міжкомпонентної взаємодії”*

**Лектор:**

асист. каф. ПЗ

Луцик І.І.

**Виконав:**

ст. гр. ПЗ-45

Хруставчук М.Л.

**Прийняв:**

асист. каф. ПЗ

МалийР.М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

∑= \_\_\_\_\_ .

Львів – 2025

**Тема роботи:** Використання брокерів повідомлень для організації міжкомпонентної взаємодії.

**Мета роботи:** Ознайомитися з принципами роботи брокерів повідомлень для реалізації міжкомпонентної взаємодії, вивчити основи роботи з RabbitMQ та реалізувати приклад взаємодії компонентів програмного забезпечення через черги повідомлень з використанням технології RabbitMQ.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

RabbitMQ – це брокер повідомлень, який забезпечує асинхронну взаємодію між компонентами програмного забезпечення. Він працює за принципом обміну повідомленнями між Producer та Consumer через черги, що дозволяє сервісам не чекати завершення обробки даних і виконувати роботу паралельно. Такий підхід підвищує масштабованість, надійність та швидкодію системи.

Передача даних у RabbitMQ здійснюється через обмінники (exchange), які розподіляють повідомлення у відповідні черги відповідно до заданих ключів маршрутизації (routing keys). Існують різні типи обмінників: Direct, Fanout, Headers та Topic. Direct використовується для точного збігу ключів, тоді як Topic – для маршрутизації за шаблоном, що забезпечує більшу гнучкість у складних системах.

У межах лабораторної роботи реалізовано два сервіси: ExerciseService (Producer) та SpeechAIService (Consumer), які обмінюються повідомленнями через RabbitMQ із використанням Topic Exchange. Додано обробку винятків і логування за допомогою бібліотеки Serilog, що забезпечує контроль виконання та зручність налагодження програми.

**ЗАВДАННЯ**

1. Розгорнути RabbitMQ за допомогою Docker або локальної інсталяції.
2. У відповідності до варіанту вибраного у лабораторній роботі №1 реалізувати два сервіси:

* Producer: приймає вхідні дані від користувача та відправляє їх у RabbitMQ.
* Consumer: отримує повідомлення, обробляє їх (наприклад, конвертує текст у верхній регістр) та зберігає у файл.

1. Модифікувати приклад, використавши Topic Exchange.
2. Додати обробку винятків і логування у сервіси.

Предметна область – “Мобільний застосунок для логопедичних занять”.

**ХІД ВИКОНАННЯ**

**1. Розгорнув RabbitMQ за допомогою Docker або локальної інсталяції.**

RabbitMQ було розгорнуто за допомогою офіційного Docker-образу rabbitmq:3-management, який містить у собі сервер та веб-інтерфейс адміністрування. Для запуску використано команду:

docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management

Після запуску брокер став доступний за адресою <http://localhost:15672>, а підключення до нього здійснювалось з використанням стандартного логіну та пароля guest/guest. У веб-інтерфейсі можна переглядати черги, повідомлення та активність сервісів, що дозволяє контролювати процес обміну даними між компонентами системи.

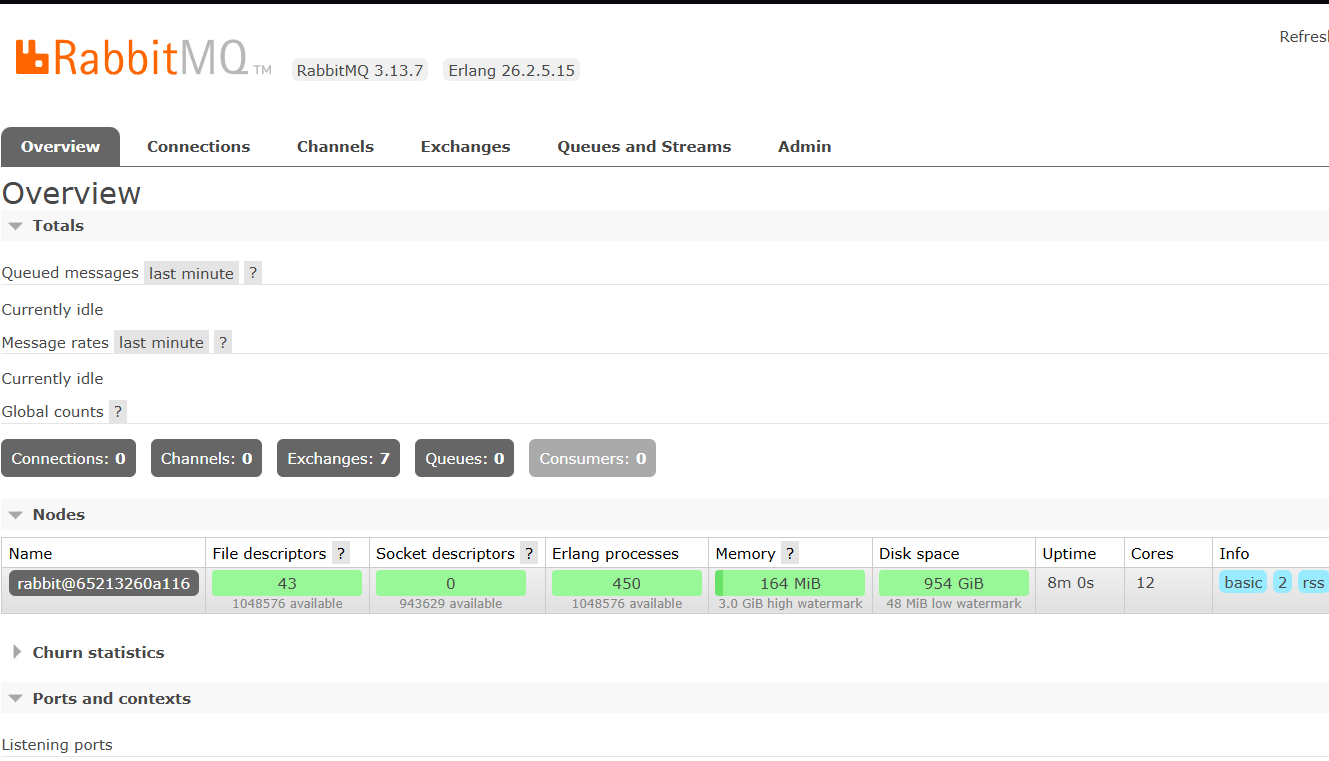


Рис. 1. Запущений контейнер та панель керування RabbitMQ

**2. Визначити архітектурний стиль та спроєктувати рівні взаємодії.**

Для реалізації обміну повідомленнями було створено два окремі сервіси – **Producer (ExerciseService)** та **Consumer (SpeechAIService)**. Сервіс **ExerciseService** приймає вхідні дані від користувача (ідентифікатор вправи), формує повідомлення у форматі JSON та відправляє його до брокера RabbitMQ. Сервіс **SpeechAIService** отримує ці повідомлення, імітує процес обробки (аналіз вимови), формує результат із полями *Accuracy* та *Feedback* і надсилає його назад у чергу з результатами.

Таким чином, було реалізовано двосторонню асинхронну взаємодію між сервісами через RabbitMQ. Передача та обробка даних відбуваються незалежно, без блокування основного потоку виконання, що відповідає принципам мікросервісної архітектури.

Файл ExerciseService.cs

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using System;

using System.Text;

using System.Text.Json;

using System.Threading.Tasks;

namespace ExerciseService;

class ExerciseService

{

static async Task Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

await using var connection = await factory.CreateConnectionAsync();

await using var channel = await connection.CreateChannelAsync();

// ===== Відправник аудіо =====

await channel.QueueDeclareAsync("exercise.audio", false, false, false, null);

// ===== Отримувач результатів =====

await channel.QueueDeclareAsync("speech.result", false, false, false, null);

var consumer = new AsyncEventingBasicConsumer(channel);

consumer.ReceivedAsync += async (sender, ea) =>

{

var json = Encoding.UTF8.GetString(ea.Body.ToArray());

var result = JsonSerializer.Deserialize<JsonElement>(json);

var exerciseId = result.GetProperty("ExerciseId").GetString();

var accuracy = result.GetProperty("Accuracy").GetDouble();

var feedback = result.GetProperty("Feedback").GetString();

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine($"\n📥 Отримано результат для вправи #{exerciseId}");

Console.ResetColor();

Console.WriteLine($"→ Точність: {accuracy:P0}");

Console.WriteLine($"→ Коментар: {feedback}");

await Task.Delay(1000);

Console.WriteLine($"[ExerciseService] Результат збережено у ProgressService ✅");

// 🟡 Після обробки — повторно показуємо запрошення

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("\nВведи ідентифікатор вправи (або 'exit' для виходу):");

Console.ResetColor();

};

await channel.BasicConsumeAsync("speech.result", autoAck: true, consumer);

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("\nВведи ідентифікатор вправи (або 'exit' для виходу):");

Console.ResetColor();

while (true)

{

var exerciseId = Console.ReadLine();

if (exerciseId == "exit") break;

var message = new

{

ExerciseId = exerciseId,

UserId = 7,

AudioUrl = $"audio\_{exerciseId}.wav",

Timestamp = DateTime.UtcNow

};

var body = Encoding.UTF8.GetBytes(JsonSerializer.Serialize(message));

var props = new BasicProperties();

await channel.BasicPublishAsync<BasicProperties>(

exchange: "",

routingKey: "exercise.audio",

mandatory: false,

basicProperties: props,

body: body);

Console.WriteLine($"[ExerciseService] Надіслано повідомлення для вправи {exerciseId}");

}

}

}

Файл SpeechAIService.cs

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using System;

using System.Text;

using System.Text.Json;

using System.Threading.Tasks;

namespace SpeechAIService;

class SpeechAIService

{

static async Task Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

await using var connection = await factory.CreateConnectionAsync();

await using var channel = await connection.CreateChannelAsync();

// Черга для прийому аудіо

await channel.QueueDeclareAsync(

queue: "exercise.audio",

durable: false,

exclusive: false,

autoDelete: false,

arguments: null);

// Черга для відправки результатів

await channel.QueueDeclareAsync(

queue: "speech.result",

durable: false,

exclusive: false,

autoDelete: false,

arguments: null);

Console.WriteLine("[SpeechAIService] Очікування повідомлень у 'exercise.audio'...");

var consumer = new AsyncEventingBasicConsumer(channel);

// ⚙️ НОВА СИГНАТУРА: два параметри — sender, eventArgs

consumer.ReceivedAsync += async (sender, ea) =>

{

try

{

var json = Encoding.UTF8.GetString(ea.Body.ToArray());

var message = JsonSerializer.Deserialize<JsonElement>(json);

var exerciseId = message.GetProperty("ExerciseId").GetString();

Console.WriteLine($"\n🎧 Отримано вправу #{exerciseId}");

Console.WriteLine("→ Аналіз вимови (імітація 2 сек)...");

await Task.Delay(2000); // імітація аналізу

var result = new

{

ExerciseId = exerciseId,

Accuracy = 0.91,

Feedback = "Вимова правильна, артикуляція гарна!",

Timestamp = DateTime.UtcNow

};

var body = Encoding.UTF8.GetBytes(JsonSerializer.Serialize(result));

var props = new BasicProperties();

await channel.BasicPublishAsync<BasicProperties>(

exchange: "",

routingKey: "speech.result",

mandatory: false,

basicProperties: props,

body: body);

Console.WriteLine($"✅ Результат для вправи #{exerciseId} відправлено у 'speech.result'");

}

catch (Exception ex)

{

Console.WriteLine($"❌ Помилка обробки повідомлення: {ex.Message}");

}

};

await channel.BasicConsumeAsync(

queue: "exercise.audio",

autoAck: true,

consumer: consumer);

Console.WriteLine("Натисни Enter для виходу...");

Console.ReadLine();

}

}

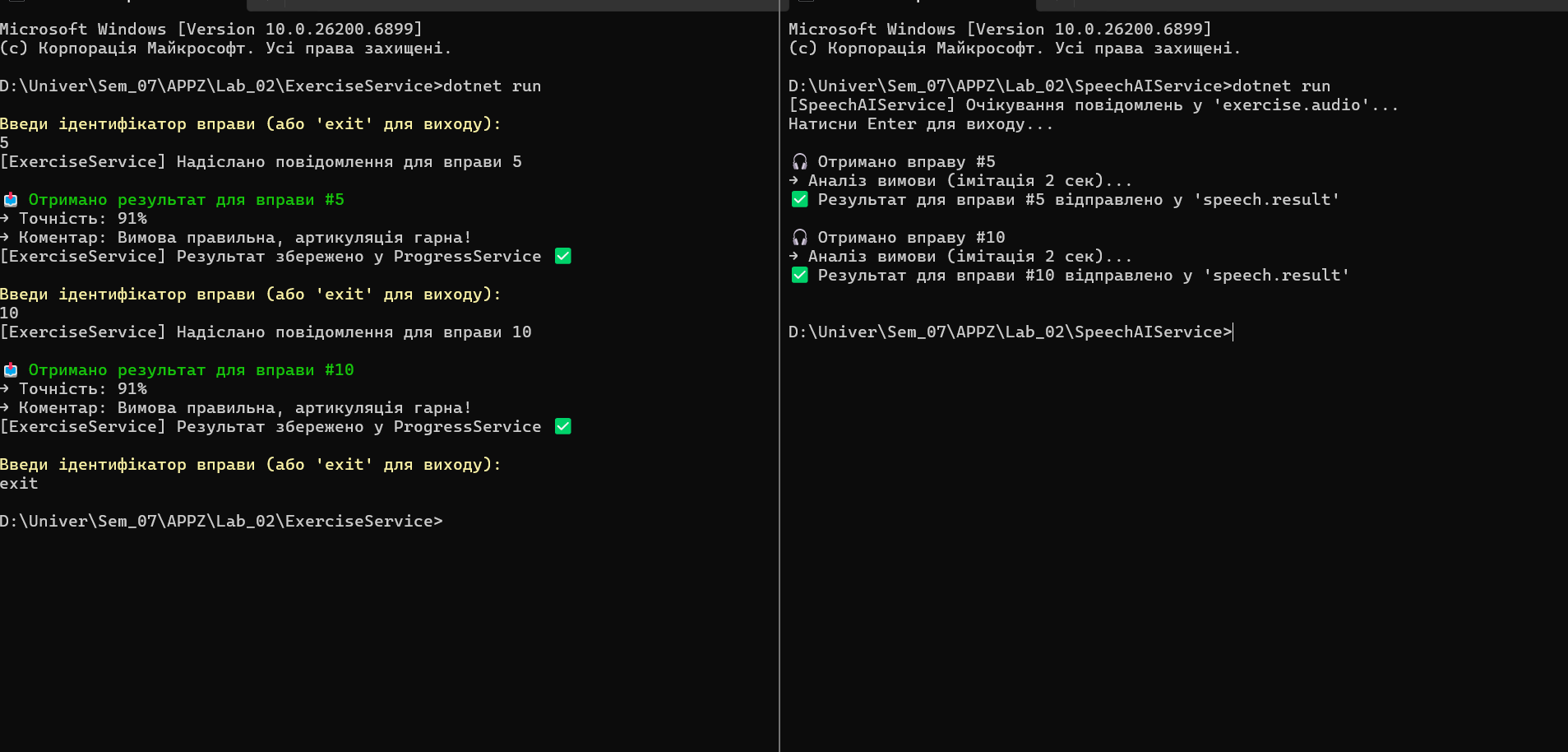


Рис. 2. Робота програми з використанням Direct Exchange

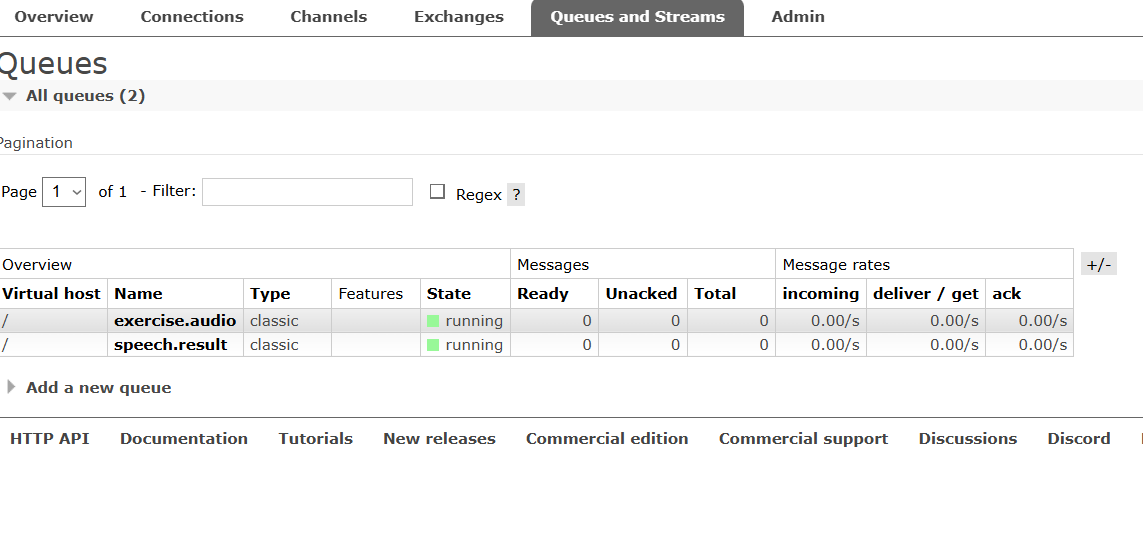


Рис. 3. Черги RabbitMQ під час роботи системи з використанням Direct Exchange

**3. Модифікував програму, використавши Topic Exchange. Додати обробку винятків і логування у сервіси.**

Програма була модифікована для використання Topic Exchange, що дозволило реалізувати гнучку маршрутизацію повідомлень між сервісами за допомогою шаблонів ключів. Було створено обмінник speech\_exchange типу *topic* та дві черги: exercise.audio і speech.result. Повідомлення з ключем exercise.audio.new відправляються від ExerciseService, а SpeechAIService підписується на шаблон exercise.audio.\*, отримує повідомлення, обробляє їх та публікує результат у чергу з ключем speech.result.done. ExerciseService у свою чергу підписується на шаблон speech.result.\*, отримує оцінку та імітує збереження даних у ProgressService.

Додатково в обидва сервіси було інтегровано логування за допомогою Serilog для запису подій у консоль та файл, а також реалізовано обробку винятків через конструкції try/catch. Це дозволило забезпечити стабільність роботи, відстеження подій і спростити діагностику під час виконання програми.

Файл ExerciseService.cs

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using Serilog;

using Serilog.Events;

using System;

using System.Text;

using System.Text.Json;

using System.Threading.Tasks;

namespace ExerciseService;

class ExerciseService

{

static async Task Main(string[] args)

{

// 🪵 Логування

Log.Logger = new LoggerConfiguration()

.WriteTo.Console(

outputTemplate: "[{Timestamp:yyyy-MM-dd HH:mm:ss}] [{Level:u3}] {Message:lj}{NewLine}")

.WriteTo.File(

path: "exercise\_log.txt",

rollingInterval: RollingInterval.Day,

outputTemplate: "[{Timestamp:yyyy-MM-dd HH:mm:ss}] [{Level:u3}] {Message:lj}{NewLine}",

restrictedToMinimumLevel: LogEventLevel.Information)

.CreateLogger();

try

{

var factory = new ConnectionFactory() { HostName = "localhost" };

await using var connection = await factory.CreateConnectionAsync();

await using var channel = await connection.CreateChannelAsync();

const string exchangeName = "speech\_exchange";

// 🧩 Обмінник Topic

await channel.ExchangeDeclareAsync(exchange: exchangeName, type: ExchangeType.Topic);

// ====== Queue для аудіо

await channel.QueueDeclareAsync("exercise.audio", false, false, false, null);

await channel.QueueBindAsync("exercise.audio", exchangeName, "exercise.audio.new");

// ====== Queue для результатів

await channel.QueueDeclareAsync("speech.result", false, false, false, null);

await channel.QueueBindAsync("speech.result", exchangeName, "speech.result.\*");

// ====== Consumer для результатів ======

var consumer = new AsyncEventingBasicConsumer(channel);

consumer.ReceivedAsync += async (sender, ea) =>

{

try

{

var json = Encoding.UTF8.GetString(ea.Body.ToArray());

var result = JsonSerializer.Deserialize<JsonElement>(json);

var exerciseId = result.GetProperty("ExerciseId").GetString();

var accuracy = result.GetProperty("Accuracy").GetDouble();

var feedback = result.GetProperty("Feedback").GetString();

Log.Information("📥 Отримано результат для вправи #{Id}: {Accuracy:P0} | {Feedback}", exerciseId, accuracy, feedback);

await Task.Delay(500);

Log.Information("[ExerciseService] Результат збережено у ProgressService ✅");

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("\nВведи ідентифікатор вправи (або 'exit' для виходу):");

Console.ResetColor();

}

catch (Exception ex)

{

Log.Error(ex, "❌ Помилка при обробці результату");

}

};

await channel.BasicConsumeAsync("speech.result", autoAck: true, consumer);

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Введи ідентифікатор вправи (або 'exit' для виходу):");

Console.ResetColor();

while (true)

{

var exerciseId = Console.ReadLine();

if (exerciseId == "exit") break;

try

{

var message = new

{

ExerciseId = exerciseId,

UserId = 7,

AudioUrl = $"audio\_{exerciseId}.wav",

Timestamp = DateTime.UtcNow

};

var body = Encoding.UTF8.GetBytes(JsonSerializer.Serialize(message));

var props = new BasicProperties();

await channel.BasicPublishAsync<BasicProperties>(

exchange: exchangeName,

routingKey: "exercise.audio.new",

mandatory: false,

basicProperties: props,

body: body);

Log.Information("[ExerciseService] Надіслано аудіо для вправи {Id}", exerciseId);

}

catch (Exception ex)

{

Log.Error(ex, "❌ Помилка при надсиланні повідомлення");

}

}

}

catch (Exception ex)

{

Log.Fatal(ex, "❌ Помилка при запуску ExerciseService");

}

finally

{

Log.CloseAndFlush();

}

}

}

Файл SpeechAIService.cs

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using Serilog;

using System;

using System.Text;

using System.Text.Json;

using System.Threading.Tasks;

namespace SpeechAIService;

class SpeechAIService

{

static async Task Main(string[] args)

{

Log.Logger = new LoggerConfiguration()

.WriteTo.Console(

outputTemplate: "[{Timestamp:yyyy-MM-dd HH:mm:ss}] [{Level:u3}] {Message:lj}{NewLine}")

.WriteTo.File(

path: "speechai\_log.txt",

rollingInterval: RollingInterval.Day,

outputTemplate: "[{Timestamp:yyyy-MM-dd HH:mm:ss}] [{Level:u3}] {Message:lj}{NewLine}")

.CreateLogger();

try

{

var factory = new ConnectionFactory() { HostName = "localhost" };

await using var connection = await factory.CreateConnectionAsync();

await using var channel = await connection.CreateChannelAsync();

const string exchangeName = "speech\_exchange";

// 🧩 Обмінник Topic

await channel.ExchangeDeclareAsync(exchange: exchangeName, type: ExchangeType.Topic);

// Queue для прийому аудіо

await channel.QueueDeclareAsync("exercise.audio", false, false, false, null);

await channel.QueueBindAsync("exercise.audio", exchangeName, "exercise.audio.\*");

Log.Information("[SpeechAIService] Очікування повідомлень...");

var consumer = new AsyncEventingBasicConsumer(channel);

consumer.ReceivedAsync += async (sender, ea) =>

{

try

{

var json = Encoding.UTF8.GetString(ea.Body.ToArray());

var message = JsonSerializer.Deserialize<JsonElement>(json);

var exerciseId = message.GetProperty("ExerciseId").GetString();

Log.Information("🎧 Отримано вправу #{Id}, аналіз вимови...", exerciseId);

await Task.Delay(2000); // Імітація аналізу

var result = new

{

ExerciseId = exerciseId,

Accuracy = 0.91,

Feedback = "Вимова правильна, артикуляція гарна!",

Timestamp = DateTime.UtcNow

};

var body = Encoding.UTF8.GetBytes(JsonSerializer.Serialize(result));

var props = new BasicProperties();

await channel.BasicPublishAsync<BasicProperties>(

exchange: exchangeName,

routingKey: "speech.result.done",

mandatory: false,

basicProperties: props,

body: body);

Log.Information("✅ Результат для вправи #{Id} опубліковано у 'speech.result.done'", exerciseId);

}

catch (Exception ex)

{

Log.Error(ex, "❌ Помилка при обробці повідомлення");

}

};

await channel.BasicConsumeAsync("exercise.audio", autoAck: true, consumer);

Console.WriteLine("Натисни Enter для виходу...");

Console.ReadLine();

}

catch (Exception ex)

{

Log.Fatal(ex, "❌ Помилка при запуску SpeechAIService");

}

finally

{

Log.CloseAndFlush();

}

}

}

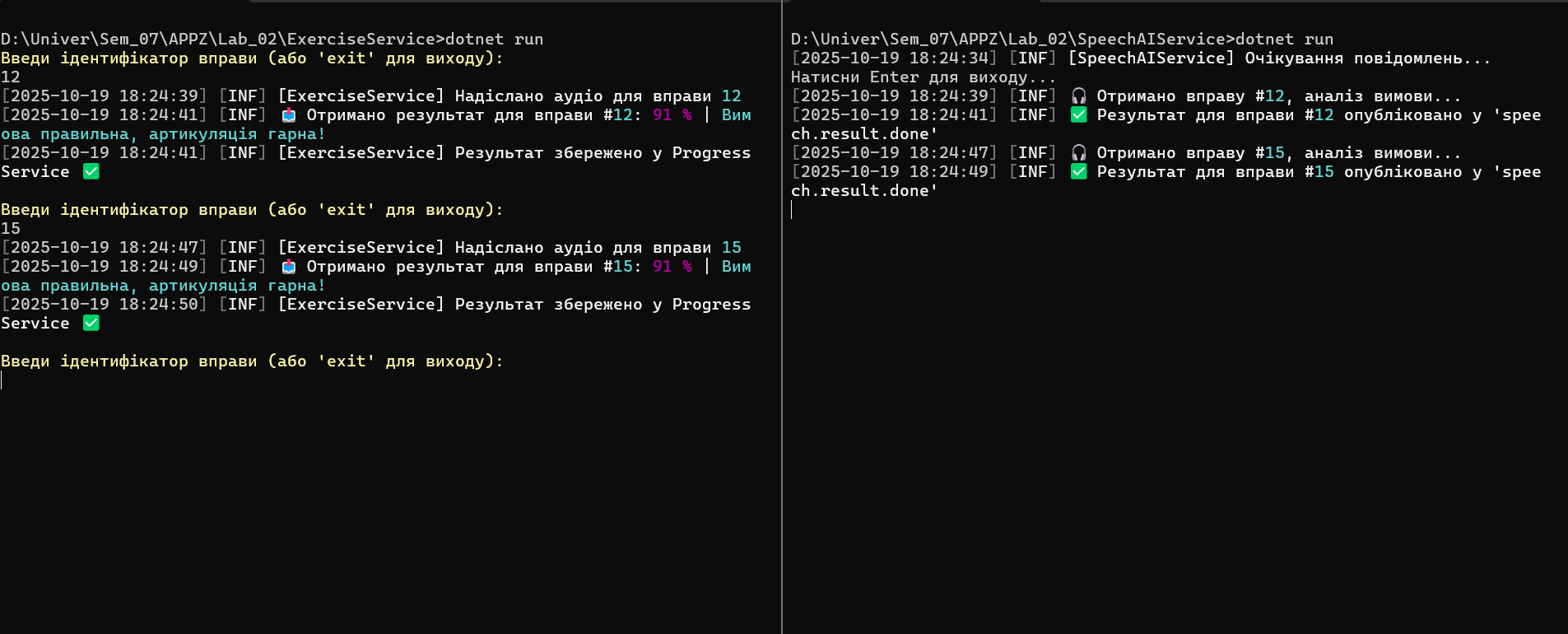


Рис. 4. Робота програми після модифікації з використанням Topic Exchange

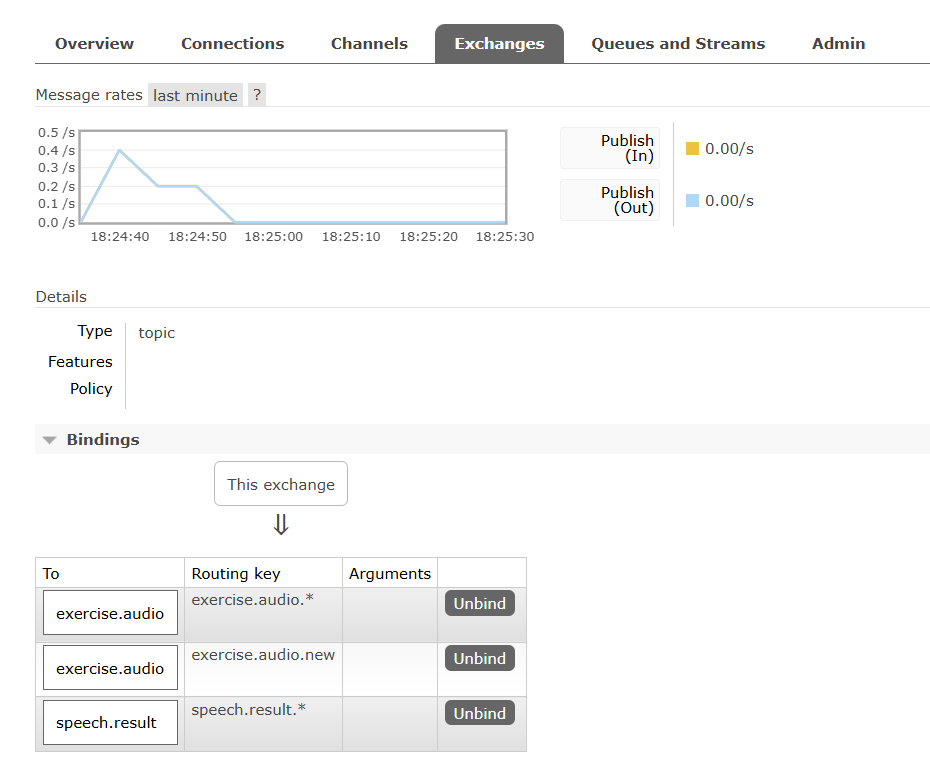
****

Рис. 5. Черги RabbitMQ під час роботи системи з використанням Topic Exchange

**ВИСНОВКИ**

У результаті виконання лабораторної роботи було розгорнуто брокер повідомлень RabbitMQ та реалізовано два сервіси, що здійснюють асинхронний обмін даними через черги. Завдяки використанню Topic Exchange вдалося забезпечити гнучку маршрутизацію повідомлень між компонентами системи.  
Додане логування та обробка винятків підвищили надійність і контроль роботи сервісів. Отриманий результат демонструє переваги асинхронної взаємодії в мікросервісній архітектурі – масштабованість, незалежність та стабільність системи.