GetawayGo

Service Bus Implementation

Fontys University of Applied Sciences

Anna Kadurina
15/01/2025

Table of Contents

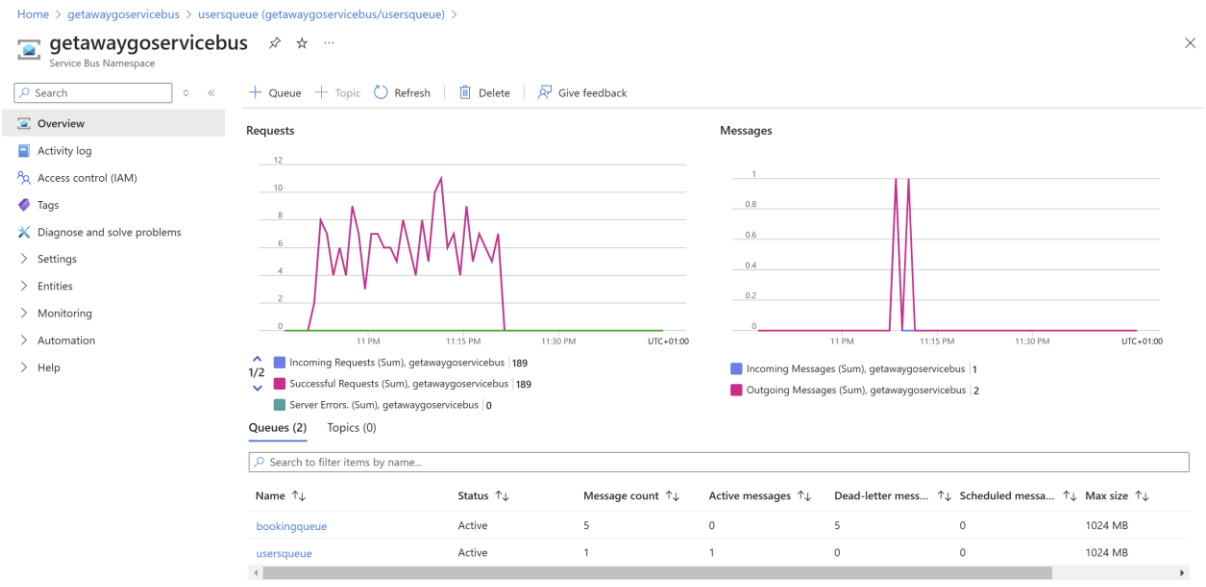
Introduction 1
Service Bus in Azure 1
Implementation examples 2
Creation of a booking 2
Deletion of user 5
Conclusion 7

Introduction

This document outlines the implementation of Azure Service Bus in the GetawayGo system, designed to facilitate reliable and scalable messaging between different services. Azure Service Bus is used to decouple microservices, allowing them to communicate asynchronously without direct dependencies. By sending messages between services, I enable event-driven workflows, such as user deletion events triggering data removal actions in other services.

Service Bus in Azure

I have created a service bus resource in Azure with 2 queues – bookings and users.



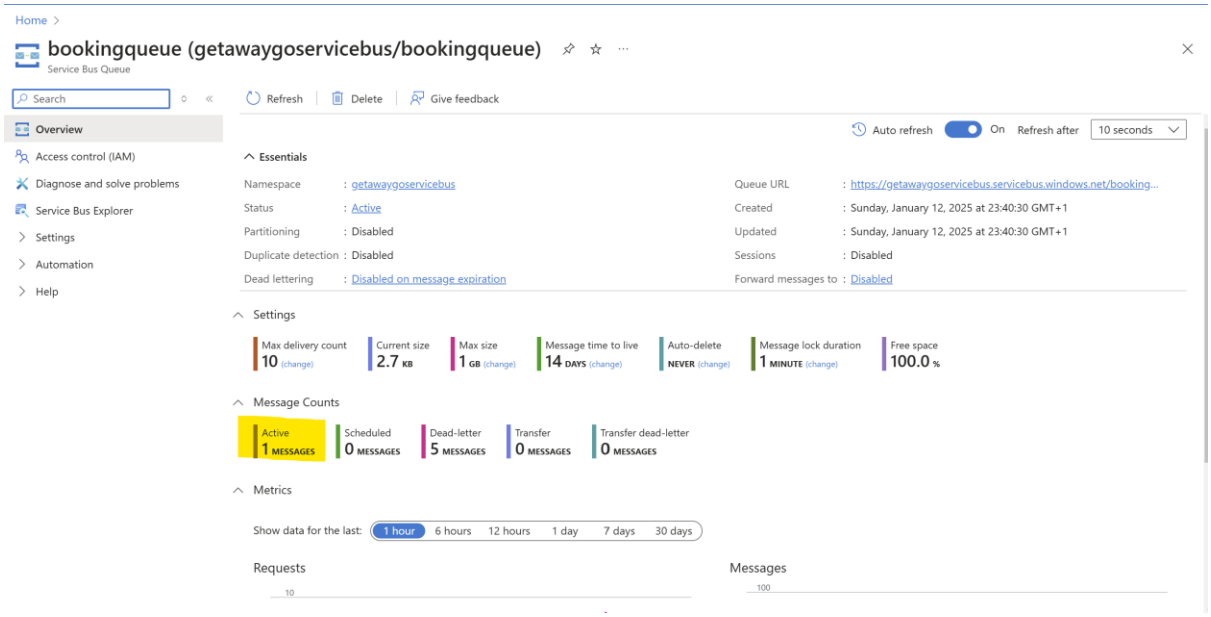
Implementation examples

Creation of a booking

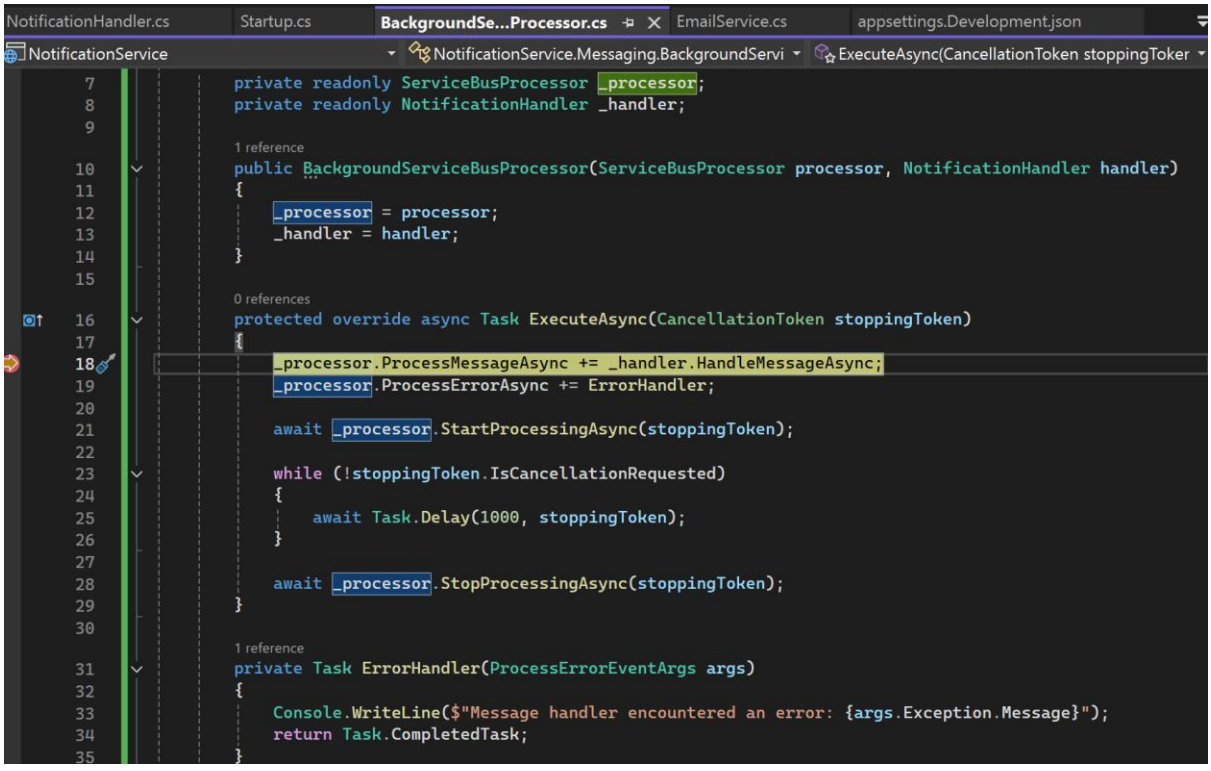
Firstly, the booking is being created with all the needed details including the creation of a payment intent using Stripe. When the booking is created, a message is sent to the booking queue on the service bus.

```
public async Task<BaseResponse<CreateBookingResponse>> ExecuteAsync(CreateBookingRequest request)
{
    try
    {
        var paymentIntent = await _stripePaymentService.CreatePaymentIntentAsync(request.TotalPrice, "eur");
        if (paymentIntent == null)
        {
            return new BaseResponse<CreateBookingResponse>("Payment failed", 400);
        }

        var bookingEntity = new BookingEntity
        {
            UserId = request.UserId,
            PropertyId = request.PropertyId,
            CheckInDate = request.CheckInDate,
            CheckOutDate = request.CheckOutDate,
            TotalPrice = request.TotalPrice,
            StatusId = 2
        };

        var savedBooking = await _bookingRepository.AddBookingAsync(bookingEntity);

        if (savedBooking == null)
        {
            return new BaseResponse<CreateBookingResponse>("Failed to save booking", 500);
        }

        var bookingResponse = new CreateBookingResponse
        {
            BookingId = savedBooking.BookingId,
            PaymentIntentId = paymentIntent.Id,
            ClientSecret = paymentIntent.ClientSecret
        };

        await _serviceBus.SendMessageAsync(savedBooking.BookingId.ToString());

        return new BaseResponse<CreateBookingResponse>(bookingResponse); ≤ 813ms elapsed
    }
    catch (Exception ex)
    {
        return new BaseResponse<CreateBookingResponse>(ex.Message, 500);
```

When the code for sending a message is executed, we can see there is 1 active message. The queues are configured by default to keep the messages alive for 14 days. If the recipient crashes or is unavailable at the moment, the message can be picked up later when the service is healthy again.



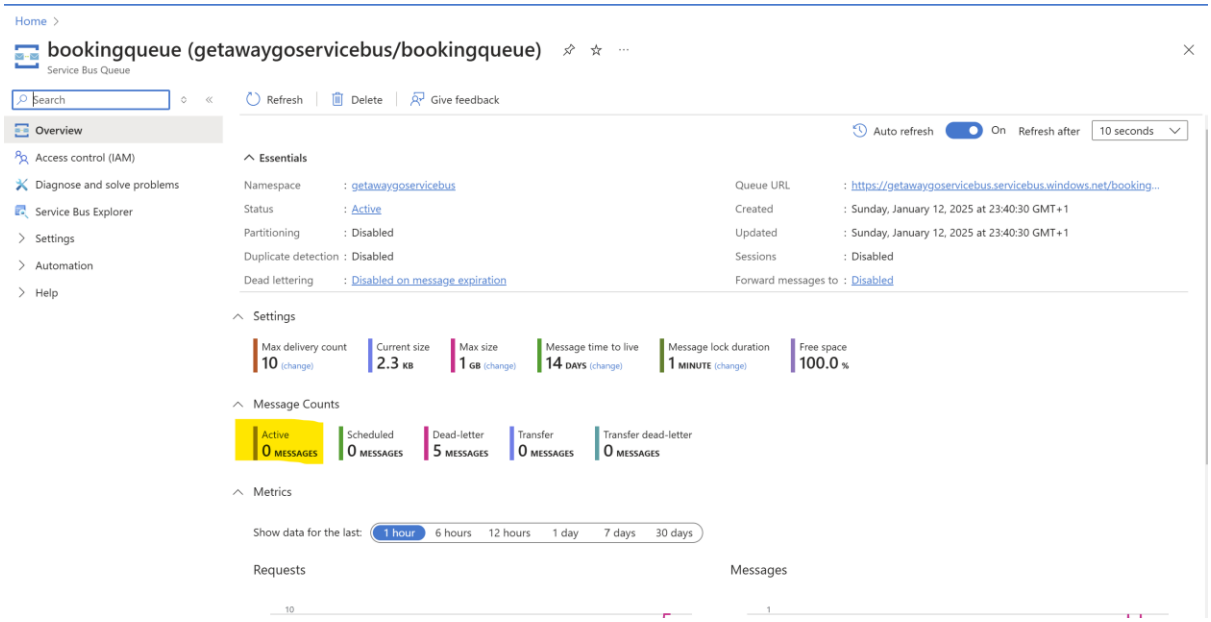
Then the Notification Service picks up the message in order to be able to send a confirmation email to the person that made the booking. The service is checking for new messages on the service bus constantly.



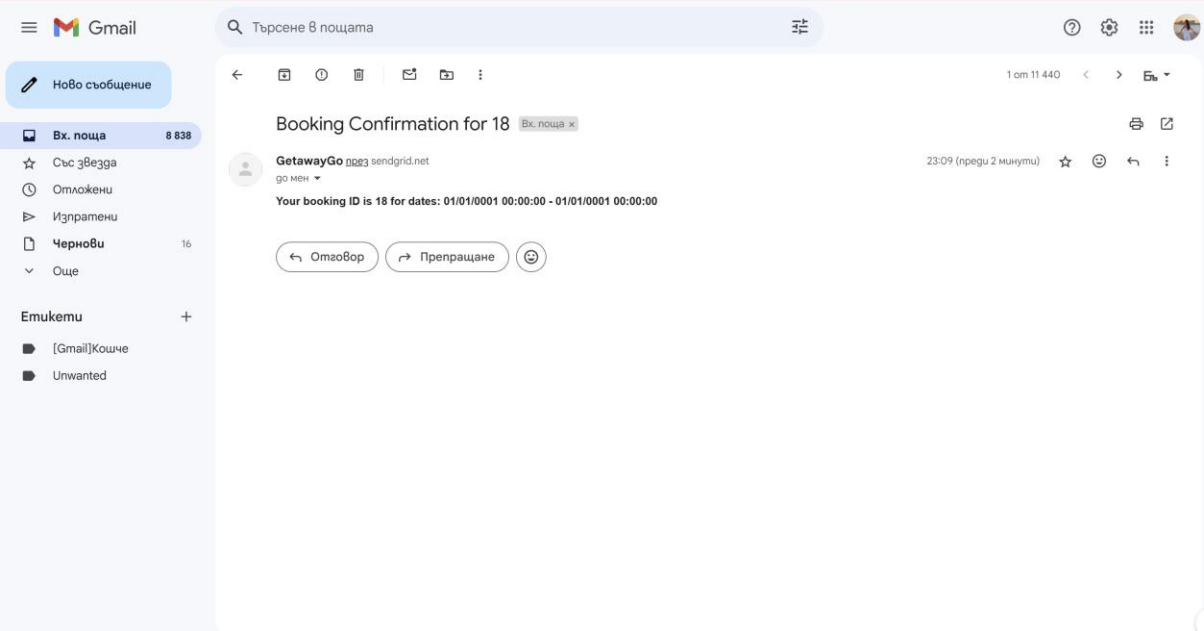
If there is a message, the NotificationHandler extracts the booking ID from the message and gets all booking details from the BookingService.

```
1 reference
public async Task HandleMessageAsync(ProcessMessageEventArgs args)
{
    var bookingId = args.Message.Body.ToString();

    try
    {
        var bookingDetails = await GetBookingDetailsAsync(bookingId);

        if (bookingDetails != null)
        {
            var emailSubject = $"Booking Confirmation for {bookingId}";
            var emailBody = $"Your booking ID is {bookingId} for dates: {bookingDetails.CheckInDate} - {bookingDetails

            await _emailService.SendEmailAsync(emailSubject, emailBody);
        }

        await args.CompleteMessageAsync(args.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error handling message: {ex.Message}");
        await args.AbandonMessageAsync(args.Message);
    }
}
```

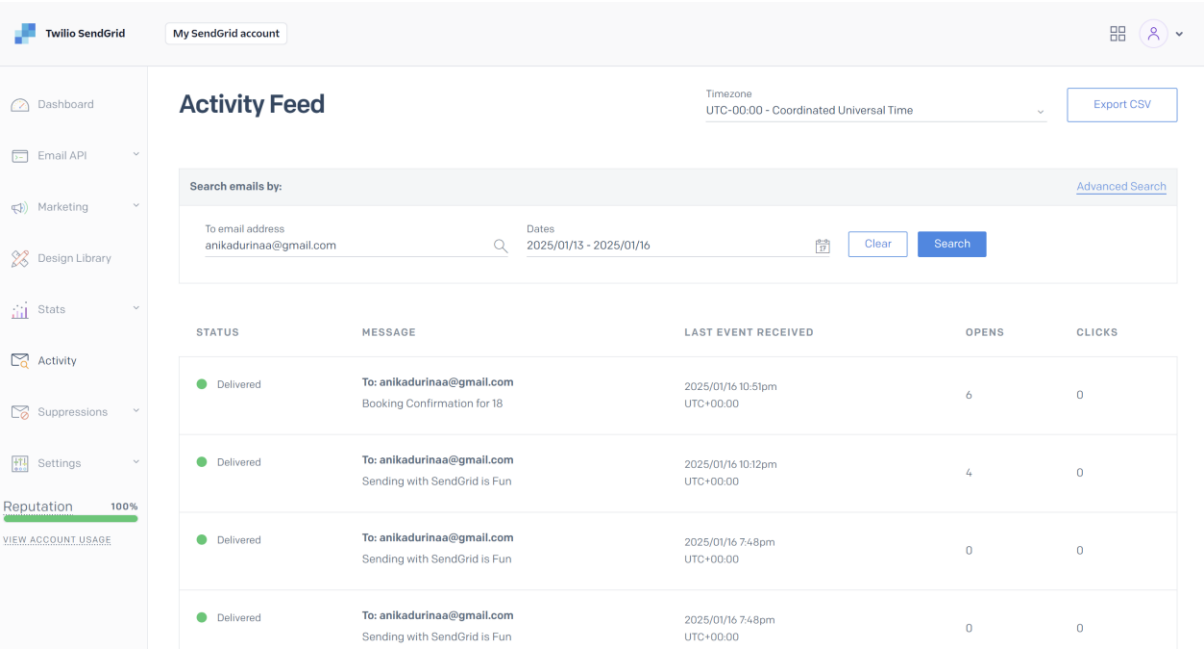
After the message is processed, it is removed from the “Active” section.



The end result of this process is the confirmation email of the booking.

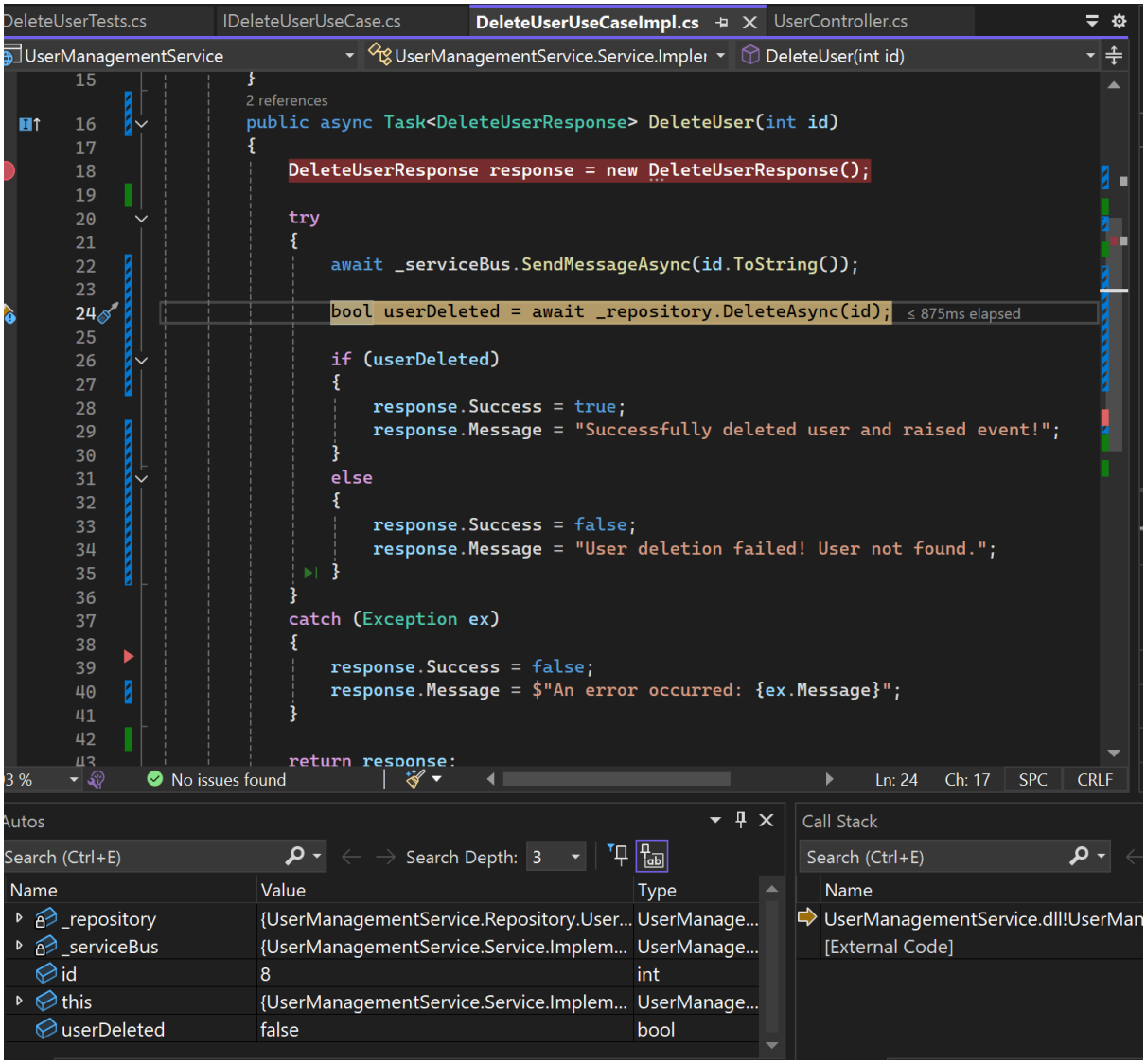


For the emails, I use SendGrid because it makes sending emails easy and reliable. It also offers features like email tracking, which helps me monitor how the emails perform.

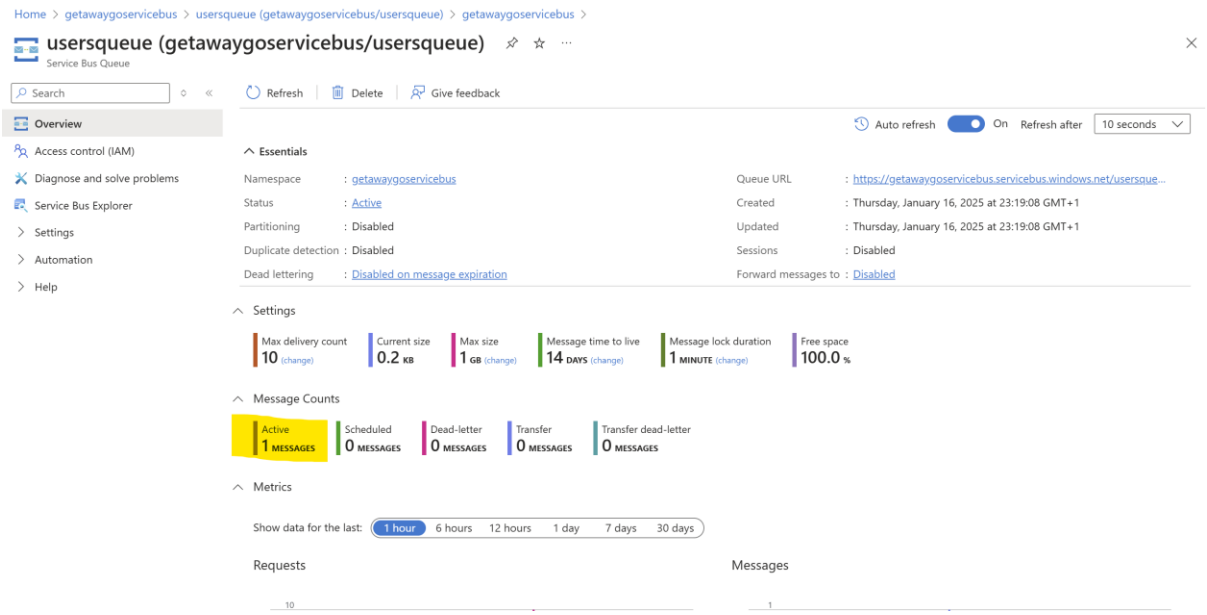


Deletion of user

For the deletion of a user, the same logic is utilized using the service bus but with the users queue. An event is raised and then asynchronously the user is also removed from the user repository itself.



When the message is send to the service bus, we can see it again in the “Active” section. Then the message is picked up by the BookingService, PropertyService, ReviewService, as shown for the creation of a booking process.



Conclusion

The implementation of Azure Service Bus in the GetawayGo system has significantly enhanced the communication and interaction between microservices. By decoupling services and enabling asynchronous messaging, the system achieves greater reliability, scalability, and fault tolerance. Overall, Azure Service Bus has proven to be a vital component of the GetawayGo architecture in various processes, providing a robust foundation for handling inter-service communication in a scalable and maintainable way.