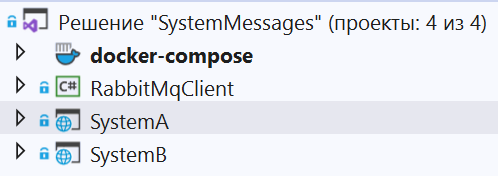
**Программная логика (системы A и B).**

Решение состоит из трех проектов:

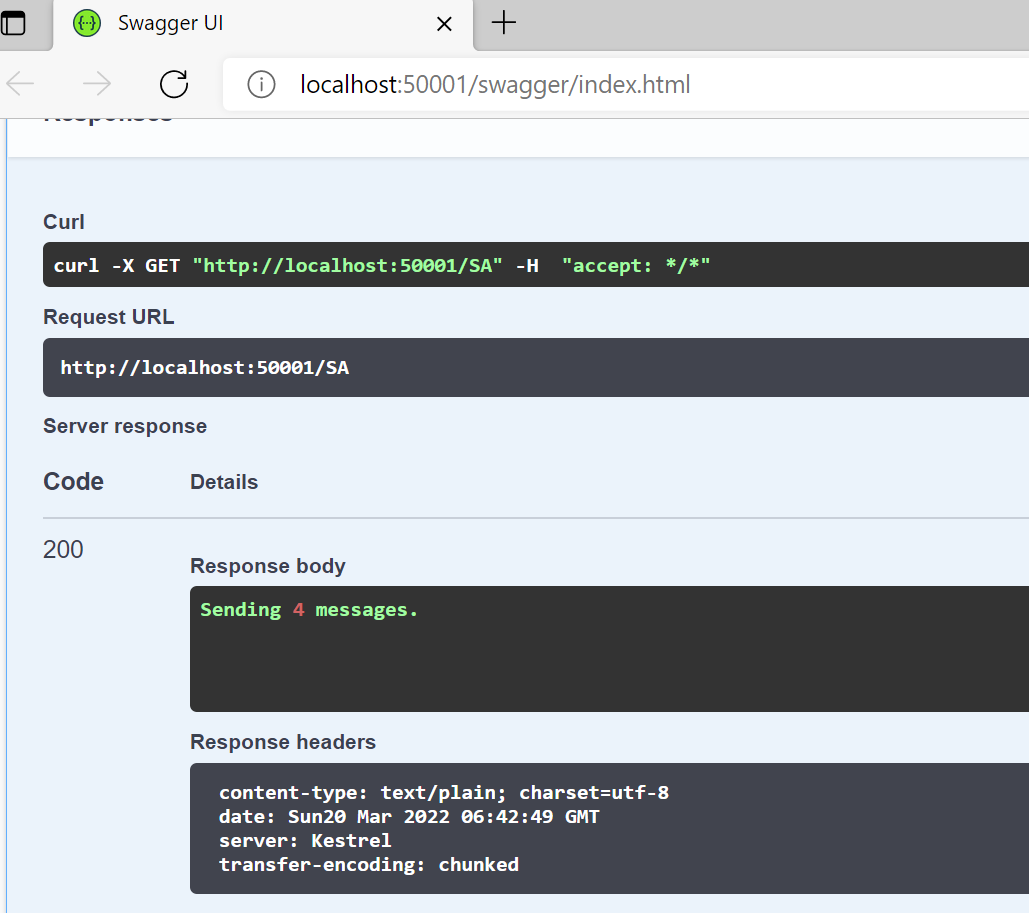


-простая библиотеки для работы с RabbitMQ (упрощенная версия-нет обработки исключений, реализаций повторного подключения);

-SystemA-Asp Net Core приложение. Генерирует случайное количество сообщений, если пользователь выполняет Get.

-SystemB-Asp Net Core приложение.

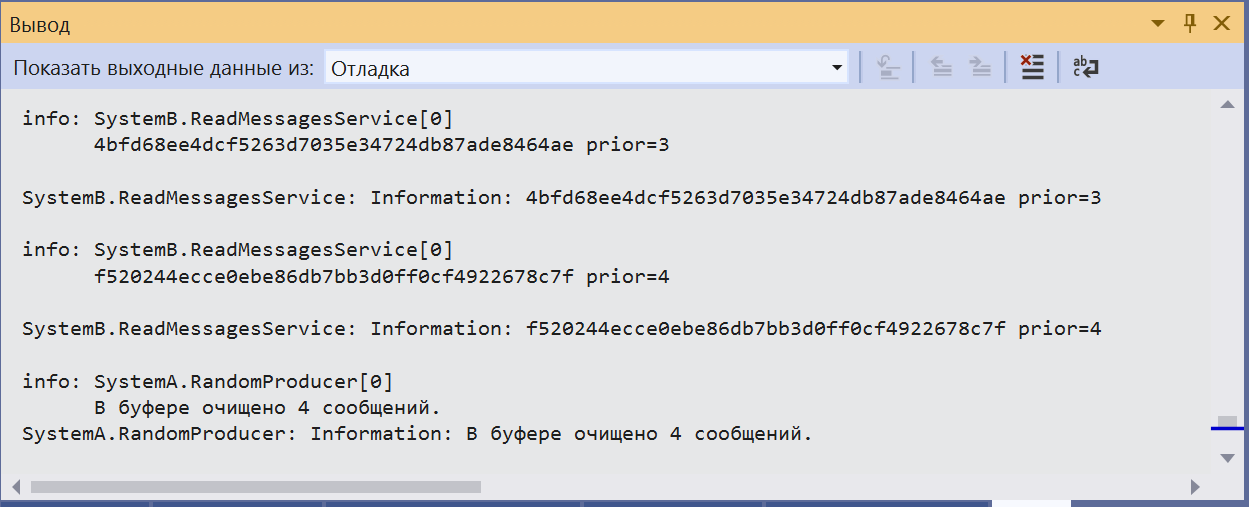
Для генерации сообщений пользователь должен выполнить Get(**http://localhost:50001/SA**) запрос к системе А:



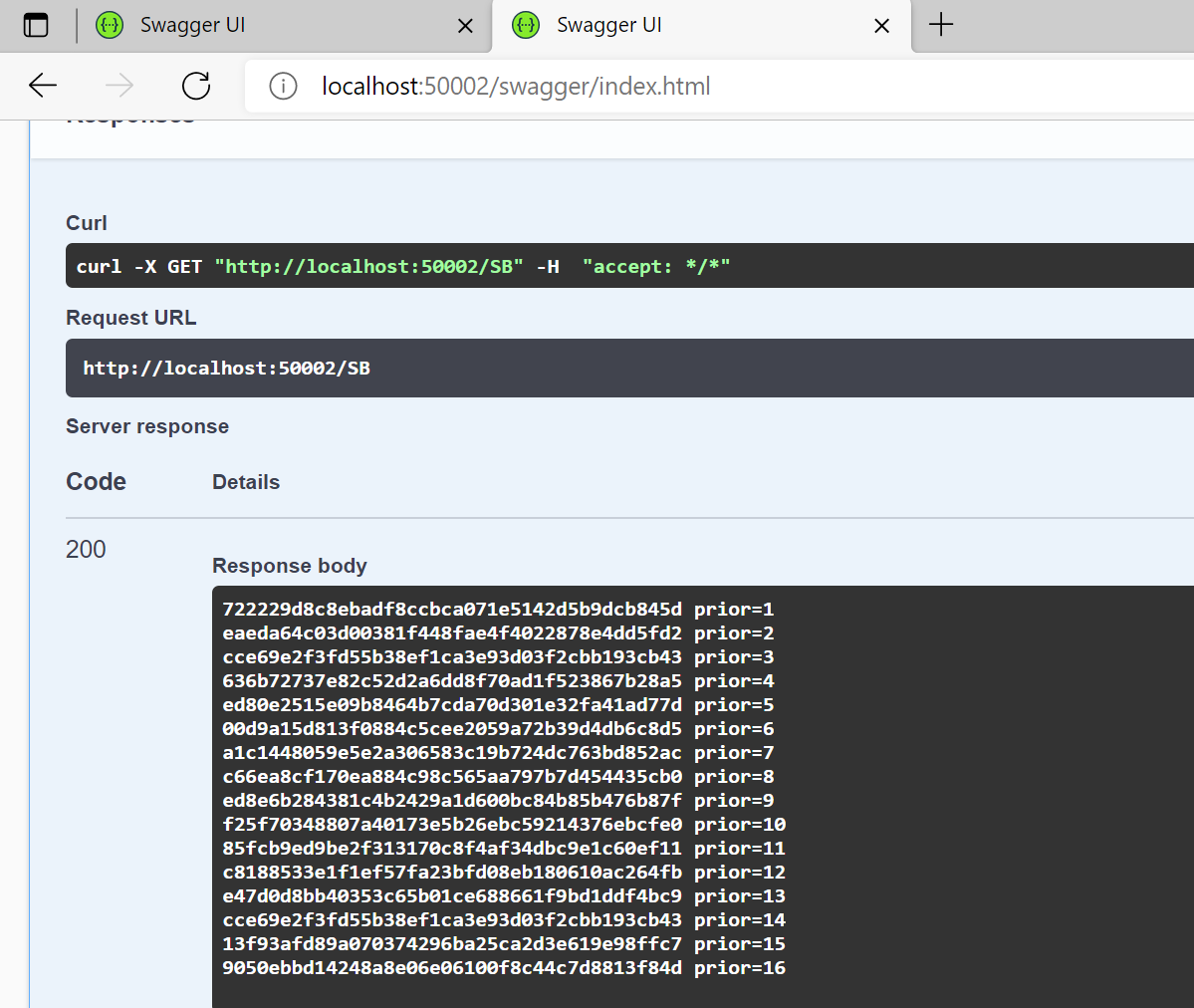
Будет сгенерировано псевдо случайное число(в диапазоне 1-10) сообщений.

Данное число является количеством сообщений, которые будут отправлены системе B. Далее средствами IRandomProducer будут сгенерированы случайные строки и отправлены системе B.

Система B раз в секунду будет читать сообщения и выводить их в консоль:



Так же можно получить все сообщения из Api сервиса B:



На рисунке сообщения при следующем запуске решения.

Настройки очередей описаны в appsettings.json обоих проектов.

**Описание классов.**

**SystemA.**

/// <summary>

/// Фоновая задача для создание очереди в RabbitMq.

/// Что бы не приходилось ждать Get запроса от пользователя,

/// в результате чего Подписчик не сможет работать.

/// </summary>

public class InitQueueService : IHostedService

/// <summary>

/// Интерфейс генерации и отправки случайных сообщений.

/// </summary>

public interface IRandomProducer

/// <summary>

/// Реализация IRandomProducer.

/// Класс для генерации случайных сообщений и отправке системе B.

/// </summary>

class RandomProducerImpl : IRandomProducer,IDisposable

/// <summary>

/// Главный контролер API системы A.

/// </summary>

[Route("[controller]")]

[ApiController]

public class SAController : ControllerBase

{

// GET: SAController>

[HttpGet]

public async Task<ActionResult> Get()

{

//Генерирую число отправляемых сообщений.

Random rnd = new Random();

int msgCount = rnd.Next(1, 10);

//Генерирую сообщения.

await producer.GenerateAsync(msgCount);

return Ok(String.Format("Sending {0} messages.",msgCount));

}

**SystemB.**

/// <summary>

/// Фоновая задача, читающая сообщения из брокера сообщений.

/// И симулирующая длительный процесс.

/// </summary>

public class ReadMessagesService : IHostedService

/// <summary>

/// Просто контейнер для возвращения результатов.

/// </summary> public interface ISystemResults

{

string Result { get; set; }

/// <summary>

/// Ичистка строки, если строка длиной более cnt символов.

/// </summary>

void Clear(int cnt);

[Route("[controller]")]

[ApiController]

public class SBController : ControllerBase

{

/// <summary>

/// Посмотреть результат работы медленной системы.

/// </summary>

/// <returns></returns>

[HttpGet]

public ActionResult Get()

{

return Ok(results.Result);

}