**БрГТУ**

**Руководство По Сборке и Конфигурации**

Дата: 7ое Января 2023

Автор: Иван Иваненко

Версия: 0.1

# Контроль Документа

**Последний раз сохранен:** 27/07/2022 18:33

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Author | Date |
| 0.1 | Создана первая версия документа | Иван Иваненко | Январь, 7, 2023 |

Содержание

[Контроль Документа 2](#_Toc109746424)

[Введение 4](#_Toc109746425)

[Назначение Приложения 5](#_Toc109746426)

[Основные Задачи Сервиса 5](#_Toc109746427)

[Конечные Пользователи Сервиса 5](#_Toc109746428)

[Функциональные Требования 6](#_Toc109746429)

[Системные Требования 7](#_Toc109746430)

# Требования

* ОС – Windows 10/11, любые UNIX системы, поддерживающий .NET 6 SDK.
* .NET 6 SDK. Установить можно здесь: <https://dotnet.microsoft.com/en-us/download/dotnet/6.0>.
* Node.js (только если требуется запуск веб клиента). Установить можно здесь: <https://nodejs.org/en/download/>
* PostgreSQL. Установить можно здесь: https://www.postgresql.org/download/
* RabbitMQ. Установить можно здесь: https://www.rabbitmq.com/download.html

# Конфигурация

Система состоит из нескольких приложений:

* RequestsServer – Web API системы, предоставляющее несколько конечных точек для работы с системой.
* RequestsProcessor – сервис, который мониторит очередь и обрабатывает входящие заявки.
* RequestsScheduler – сервис, который генерирует заявки и отправляет их на RequestsServer для дальнейшей обработки.

Разберем подробнее конфигурацию каждого из этих приложений.

## Конфигурация RequestsSсheduler

Приложение RequestsScheduler содержит следующие файлы конфигурации: *ReasonsForTravel.json, Schedule.json, appsettings.json.* Рассмотрим подробно каждый их этих файлов.

### ReasonsForTravel.json

Файл хранит json массив строковых значений, которые представляют собой возможные значения, которые может принимать поле ReasonForTravel заявки.

Пример:

[

"Job",

"School",

"Medicine",

"Shopping",

"Other"

]

### Schedule.json

Файл хранит json массив объектов, каждый из которых представляет собой названия остановок, а также конфигурацию, которая используется приложением для генерации заявок.

Пример такого объекта из файла Schedule.json:

{

"Name": "Stop41",

"DailyRequestsCount": 1500,

"TimePeriods": [

{

"From": "00:00:00",

"To": "05:59:59",

"RequestsCountCoefficient": 0.1

},

{

"From": "06:00:00",

"To": "21:00:00",

"RequestsCountCoefficient": 0.8

},

{

"From": "21:00:01",

"To": "23:59:59",

"RequestsCountCoefficient": 0.1

}

]

}

Разберем данный объект более подробно:

Name – название остановки. Каждый объект массива должен обладать уникальным именем, иначе произойдет ошибка валидации файла.

DailyRequestsCount – количество заявок на посадку с данной остановки.

TimePeriods – Массив временных периодов. Предназначается для того, чтобы конфигурировать частоту заявок в различные периоды суток. Обязательно должны покрывать 24-часовой интервал, иначе произойдет ошибка валидации файла.

From – начало интервала

To – конец интервала

RequestsCountCoefficient – коэффициент частоты генерации заявок. Сумма коэффициентов во всех объектах массива должна равняться одному, иначе произойдет ошибка валидации файла.

При соблюдении вышеперечисленных правил, каждый объект массива может обладать уникальным количеством заявок за сутки, а также уникальным набором временных интервалов с различными коэффициентами частоты генерации заявок, что делает настройку генератора заявок очень гибкой.

Пример файла Schedule.json с несколькими остановками:

[

{

"Name": "Stop1",

"DailyRequestsCount": 1000,

"TimePeriods": [

{

"From": "00:00:00",

"To": "06:59:59",

"RequestsCountCoefficient": 0.3

},

{

"From": "07:00:00",

"To": "18:59:59",

"RequestsCountCoefficient": 0.5

},

{

"From": "19:00:00",

"To": "23:59:59",

"RequestsCountCoefficient": 0.2

}

]

},

{

"Name": "Stop2",

"DailyRequestsCount": 1200,

"TimePeriods": [

{

"From": "00:00:00",

"To": "11:59:59",

"RequestsCountCoefficient": 0.5

},

{

"From": "12:00:00",

"To": "23:59:59",

"RequestsCountCoefficient": 0.5

}

]

},

{

"Name": "Stop3",

"DailyRequestsCount": 1500,

"TimePeriods": [

{

"From": "00:00:00",

"To": "05:59:59",

"RequestsCountCoefficient": 0.1

},

{

"From": "06:00:00",

"To": "10:30:00",

"RequestsCountCoefficient": 0.3

},

{

"From": "10:30:01",

"To": "15:59:59",

"RequestsCountCoefficient": 0.2

},

{

"From": "16:00:00",

"To": "21:00:00",

"RequestsCountCoefficient": 0.3

},

{

"From": "21:00:01",

"To": "23:59:59",

"RequestsCountCoefficient": 0.1

}

]

}

]

### appsettings.json

Файл представляет собой json с общими настройками приложения. Основные настройки:

* **ConfigurationFilePath** – относительный или абсолютный путь к файлу с конфигурацией остановок (Schedule.json)
* **ReasonsForTravelConfigPath** – относительный или абсолютный путь к файлу с конфигурацией причин поездок (ReasonsForTravel.json)
* **RequestsServerConfiguration** – конфигурация адреса RequestsServer, на который будут отправляться сгенерированные заявки для дальнейшей обработки.
  + **BaseUrl**  - URL RequestsServer API
  + **RequestsEndpoint** – адрес конечной точки метода, принимающего заявки посредством POST HTTP метода относительно базового URL API Requests Server (BaseUrl).
* **Serilog**  - настройки логгера. Подробнее можно прочесть тут: <https://github.com/serilog/serilog-settings-configuration>
  + **WriteTo** – секция, на которую стоит обратить внимание. Имеет 2 объекта. Каждый содержит конфигурацию логов. Первый пишет логи в консоль, второй в файл. Имеет атрибут “path”, отвечающий за путь к файлу с логами.

## Конфигурация RequestsServer

Приложение RequestsServer содержит следующие файлы конфиуграции: *appsettings.json.*

Рассмотрим подробнее каждый из этих файлов.

### appsettings.json

Файл представляет собой json с общими настройками приложения. Основные настройки:

* **Serilog**  - настройки логгера. Подробнее можно прочесть тут: <https://github.com/serilog/serilog-settings-configuration>
  + **WriteTo** – секция, на которую стоит обратить внимание. Имеет 2 объекта. Каждый содержит конфигурацию логов. Первый пишет логи в консоль, второй в файл. Имеет атрибут “path”, отвечающий за путь к файлу с логами.
* **RabbitMQConfiguration** – поскольку RequestsServer отправляет полученные заявки брокеру сообщений RabbitMQ, данная конфигурационная секция необходима
  + **QueueName** – название очереди
  + **HostName** – хост, на котором развернут RabbitMQ
  + **VirtualHost** – виртуальный хост, в котором расположена очередь
  + **UserName** – имя пользователя для аутентификации и авторизации
  + **Password** - пароль пользователя для аутентификации и авторизации
  + **Port** – порт RabbitMQ
* **ReasonsForTravelConfigPath –** относительный или абсолютный путь к файлу конфигурации причин поездки. Этот файл мы разбирали при рассмотрении конфигурационных файлов приложения RequestsScheduler.
* **RequestsConfigPath –** относительный или абсолютный путь к файлу конфигурации остановок. Этот файл мы разбирали при рассмотрении конфигурационных файлов приложения RequestsScheduler.

## Конфигурация RequestsProcessor

Приложение RequestsProcessor содержит следующие файлы конфиуграции: *appsettings.json.*

Рассмотрим подробнее каждый из этих файлов.

### appsettings.json

Файл представляет собой json с общими настройками приложения. Основные настройки:

* FaultedRequestsFilePath – относительный или абсолютный путь к файлу, в который записывается число неудачно обработанных заявок.
* **Serilog**  - настройки логгера. Подробнее можно прочесть тут: <https://github.com/serilog/serilog-settings-configuration>
  + **WriteTo** – секция, на которую стоит обратить внимание. Имеет 2 объекта. Каждый содержит конфигурацию логов. Первый пишет логи в консоль, второй в файл. Имеет атрибут “path”, отвечающий за путь к файлу с логами.
* **RabbitMQConfiguration** – поскольку RequestsProcessor получает заявки из брокера сообщений RabbitMQ, данная конфигурационная секция необходима
  + **QueueName** – название очереди
  + **HostName** – хост, на котором развернут RabbitMQ
  + **VirtualHost** – виртуальный хост, в котором расположена очередь
  + **UserName** – имя пользователя для аутентификации и авторизации
  + **Password** - пароль пользователя для аутентификации и авторизации
  + **Port** – порт RabbitMQ
* **ConnectionString** – содержит строку подключения к базе данных PostgreSQL для сохранения заявок. Подробнее о строках подключения можно узнать здесь: <https://www.connectionstrings.com/postgresql/>

# Сборка и запуск приложений

Для сборки и запуска приложений будет использоваться среда разработки Microsoft Visual Studio 2022 Community Edition. Установить ее можно тут: https://visualstudio.microsoft.com/ru/vs/community/

## Сборка и запуск RequestsServer

Открываем файл решения (RequestsServer.sln) в среде VisualStudio:

A screenshot of a computer

Description automatically generated with medium confidence

Сразу же открываем файл appsettings.json и задаем актуальные для вашего окружения настройки. Описание конфиуграционных файлов находится в главе Конфиуграция этого же документа. После этого нажимаем на кнопку запуска (зеленый треугольник), либо Ctrl+F5, это запустит наше приложение. После успешного запуска откроется окно запуска, на котором мы увидим SwaggerUI, это явяется описанием и документацией нашего API (описание конечных точек, запросов и ответов):

Graphical user interface, text, application, email

Description automatically generated

Проведем проверку корректности работы конечных точек нашего API с помощью SwaggerUI. Отправим POST запрос, содержащий заявку:

Graphical user interface, application

Description automatically generated

Background pattern

Description automatically generated

В ответе мы получили созданную заявку и статус 200, следовательно запрос обработан успешно. Для большей уверенность зайдем в RabbitMQ management console (если у вас не установлено данное расширение для RabbitMQ, то гайд по установке можно найти тут: <https://www.rabbitmq.com/management.html>), выберем нашу очередь и достанем оттуда 1 сообщение (которое API Должен был получить и отправить в эту самую очередь):

Graphical user interface, text, application, email

Description automatically generated

Как мы видим, сообщение было корректно доставлено в очередь.

Проверим оставшиеся 2 конечные точки (получение остановок и причин поездок). Перейдем в SwaggerUI и протестируем.

Получение причин поездок:

Graphical user interface, application

Description automatically generated

Получение названий остановок:

Graphical user interface, text, application

Description automatically generated

Все отработало корректно.

### Сборка и запуск RequestsScheduler

Открываем файл решения (RequestsScheduler.sln) в среде VisualStudio:

A screenshot of a computer

Description automatically generated with medium confidence

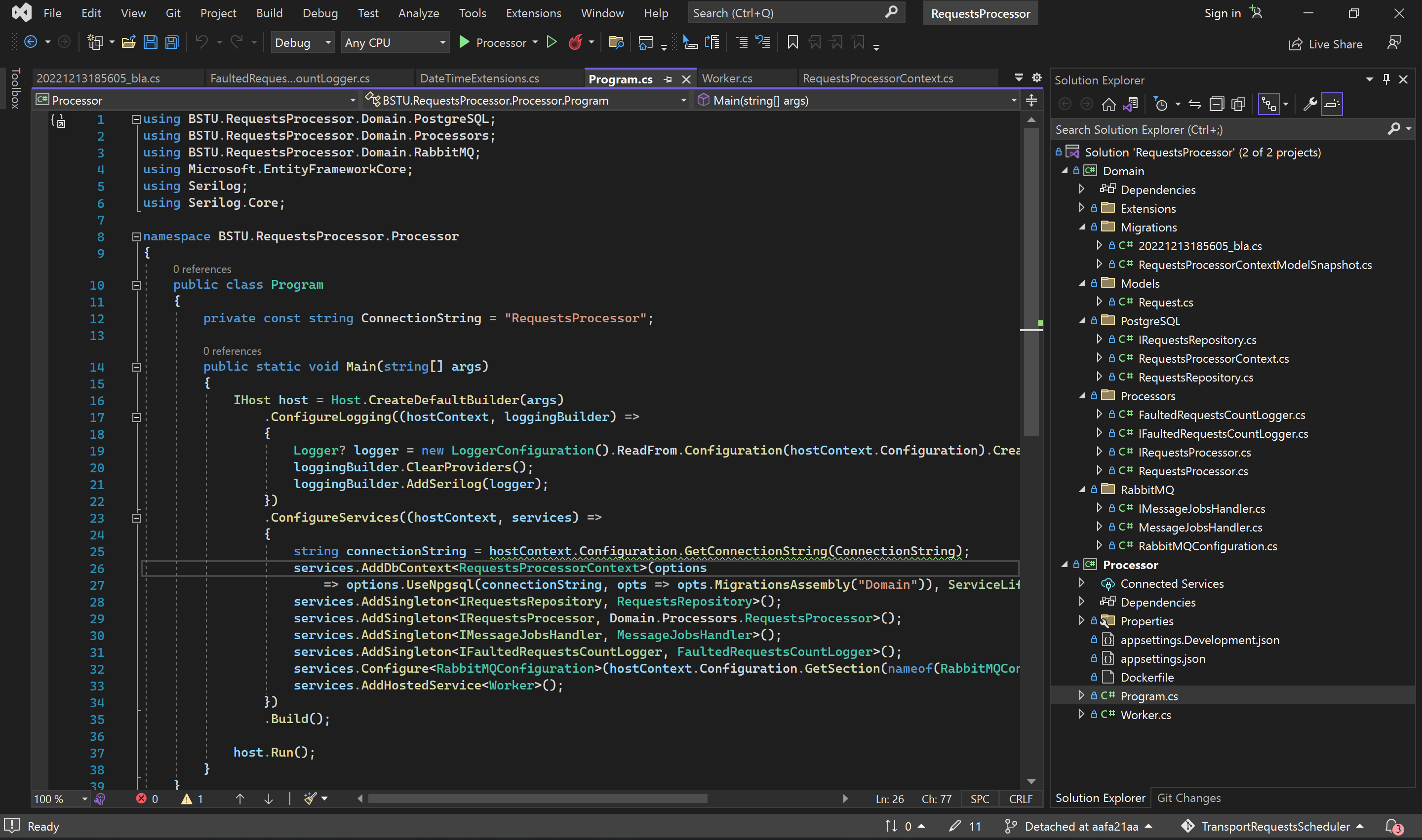
Сразу же открываем файл appsettings.json и задаем актуальные для вашего окружения настройки. Описание конфиуграционных файлов находится в главе Конфиуграция этого же документа. После этого нажимаем на кнопку запуска (зеленый треугольник), либо Ctrl+F5, это запустит наше приложение.

Text

Description automatically generated

Вот так выглядят логи приложения RequestsScheduler. До отправки первой заявки на API может пройти десяток секунд, так как происходит первоначальная генерация заявок(время генерации зависит от количества сконфигурированных остановок, времени суток и количества сконфигурированных заявок). В логах мы можем увидеть статус HTTP запроса и HTTP Ответы.

## Сборка и запуск RequestsProcessor

Открываем файл решения (RequestsProcessor.sln) в среде VisualStudio:  


Сразу же открываем файл appsettings.json и задаем актуальные для вашего окружения настройки. Описание конфиуграционных файлов находится в главе Конфиуграция этого же документа. Базу данных создавать не нужно. При запуске приложения, оно убедится, что база данных уже существует. Если она отсутствует, то будет создана автоматчиески со всеми нужными таблицами. После настройки нажимаем на кнопку запуска (зеленый треугольник), либо Ctrl+F5, это запустит наше приложение:

Text

Description automatically generated

Данный сервис мониторит очередь, и после получения заявок, добавляет их в базу данных. В логах мы можем увидеть полученные заявки и SQL запрос, вставляющий полученные данные в базу данных.