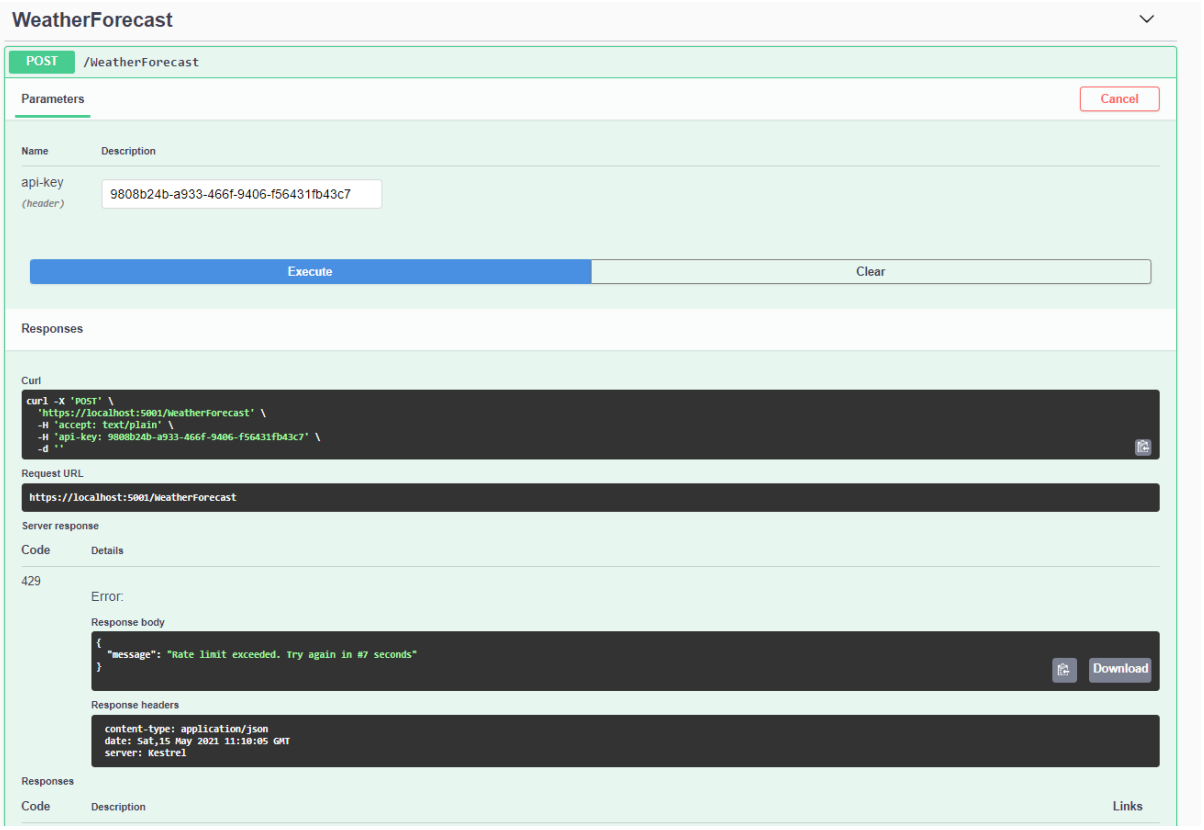
AirTasker RateLimiter is a .NET CORE 3.1 Web App

How to run

1. Open project in VisualStudio and run via F5
2. Navigate to /



3. Execute request POST /WeatherForecast with api-key: **9808b24b-a933-466f-9406-f56431fb43c7**

Default setting for user with api-key: **9808b24b-a933-466f-9406-f56431fb43c7** for method POST /WeatherForecast is 2 attempts is 10 seconds

```
{
  "Name": "9808b24b-a933-466f-9406-f56431fb43c7", /* Specific rules for specific client*/
  "Attempts": 3,
  "Seconds": 20,
  "Apis": [
    {
      "Name": "/weatherforecast", /* Specific rules for path*/
      "Attempts": 2,
      "Seconds": 10
    },
    {
      "Name": "/weatherforecastfortomorow", /* Specific rules for path*/
      "Attempts": 1,
      "Seconds": 20
    }
  ]
}
```

Configuration

```
"RateLimiter": { /* Rate limiter settings */
  "General": { /* General settings that should be checked before going to specific client (executed by api-key)*/
    "Attempts": 30,
    "Seconds": 100
  },
  "Clients": [ /* List of customized setting per clients (api-key) */
    {
      "Name": "UnknownApiKey", /* If there is no specific settings - use this one */
      "Attempts": 5, /* If there is no /path settings - using this one*/
      "Seconds": 30,
      "Apis": [
        {
          "Name": "/weatherforecast", /* Specific rules for path*/
          "Attempts": 2,
          "Seconds": 20
        },
        {
          "Name": "/weatherforecastfortomorow", /* Specific rules for path*/
          "Attempts": 2,
          "Seconds": 30
        }
      ]
    },
    {
      "Name": "9808b24b-a933-466f-9406-f56431fb43c7", /* Specific rules for specific client*/
      "Attempts": 3,
      "Seconds": 20,
      "Apis": [
        {
          "Name": "/weatherforecast", /* Specific rules for path*/
          "Attempts": 2,
          "Seconds": 10
        },
        {
          "Name": "/weatherforecastfortomorow", /* Specific rules for path*/
          "Attempts": 1,
          "Seconds": 20
        }
      ]
    }
  ]
}
```

There are 2 types of configuration:

1. General - it's Attempts/Seconds allowed for a client disregard on specific client's rules

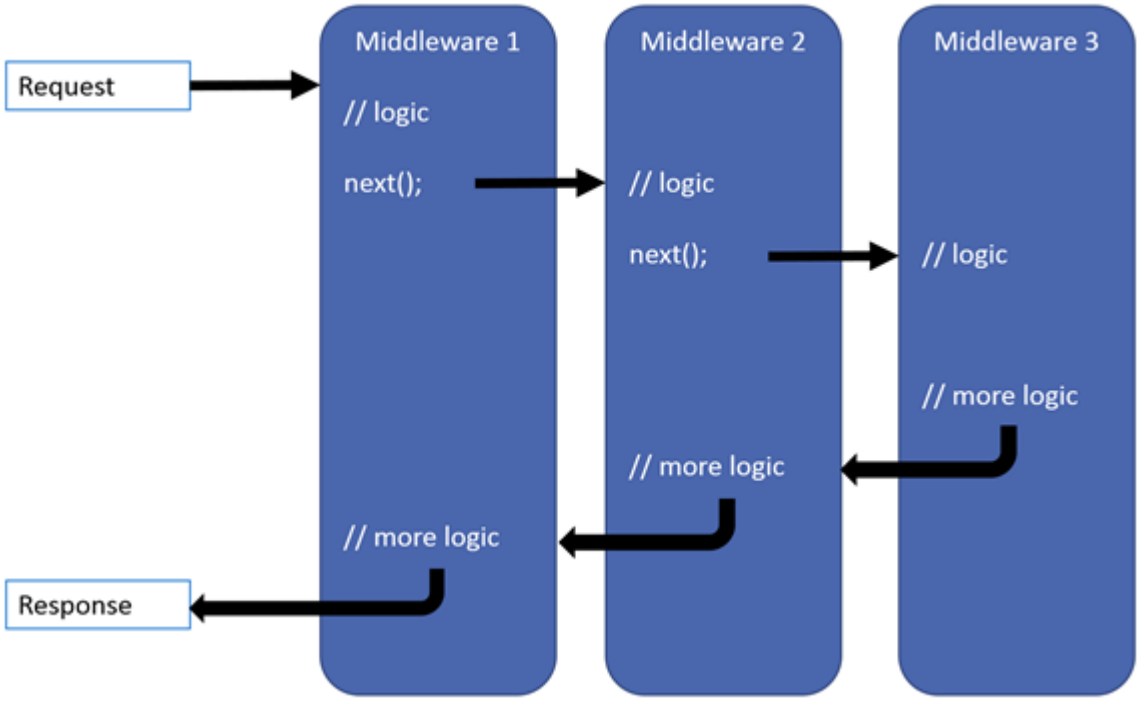
2. Client-specific rules
   2.a For any client you can specify rules that will be applicable only for this client
   2. b If there is no specific client rules - “UnknownApiKey” rules takes as default

```
{
  "Name": "9808b24b-a933-466f-9406-f56431fb43c7", /* Specific rules for specific client*/
  "Attempts": 3,
  "Seconds": 20,
  "Apis": [
    {
      "Name": "/weatherforecast", /* Specific rules for path*/
      "Attempts": 2,
      "Seconds": 10
    },
    {
      "Name": "/weatherforecastfortomorow", /* Specific rules for path*/
      "Attempts": 1,
      "Seconds": 20
    }
  ]
}
```

Each client has a method specific rules that regulate attempts/seconds per the path call. In case there is no such path - default client rules are taken.

Architecture decisions

1. Module will be implemented as a custom middleware



2. Configuration is customizable per each client per each path, there is no UI, configuration is stored in appsettings.json

3. RateLimiter don’t utilize distributed cache (ex: Redis) as we would have only one active instance of application
4. Module will be implemented as a standalone library that can be used for any ASP.NET Core WebApp
5. Adding new rate limits will be done easily by adding 1 class and registering that in the middleware

Unit testing

1. There will be 1 unit test covering algorithm of the application to prove ability to write unit tests
2. No integration tests will be written as a part of this code challenge

Additional features

To make it more enterprise ready I would add:

1. Integration tests - from unittests run httpClient and test all the custom middleware for behaviour (Check Response codes and messages)
2. Move storage to Redis to make it distributed between instances or to configure load balancer to send session of one client to one instance of application
3. Move configuration from appsettings.json to Redis and create a small UI for operation team
4. Deploy into environment, enable logging, monitoring, alerting
5. Create a CI\CD for one-click deployment
6. Create a user guid how to inject module inside any custom web api