ImageTransformer

Нужно написать сервис, решающий такую задачу. По HTTP приходит запрос, содержащий картинку в формате png, координаты прямоугольной области и название фильтра. Надо применить к картинке фильтр, вырезать фрагмент, соответствующий заданной области, и отправить в ответ.

Вопросы по заданию можно задавать здесь:
https://t.me/joinchat/A3Kxg0zr49r5YbnfHpIT0A

Требования к решению

Ожидается, что ваш сервис будет подвергаться большой нагрузке. Поэтому он, конечно же, должен работать максимально быстро. Чем быстрее обрабатывается каждый запрос — тем лучше. Чем больше запросов в секунду обрабатывается — тем лучше.

Но у любой производительности есть предел, и ваш сервис должен быть готов столкнуться с нагрузкой, превышающей его возможности.

Требования к стабильности под экстремальной нагрузкой:

1. Если нагрузка превышает возможности сервиса — производительность не должна деградировать. То есть, если сервис способен обработать максимум X rps, при нагрузке X * 2 rps сервис должен всё ещё обрабатывать не меньше X rps. Запросы, до которых очередь дойдет не скоро, можно сразу отклонять с ошибкой.
2. Производительность сервиса не должна деградировать со временем. Если сервис круглосуточно работает под нагрузкой без перезапусков, никакие ресурсы не должны закончиться и пропускная способность не должна уменьшиться.
3. Клиенты не должны замечать кратковременные задержки в работе сервиса. Если случилась пауза из-за GC, или ещё по какой-то причине сервис короткое время не обрабатывал запросы, клиенты не должны получить ошибок, если пауза действительно была небольшая.

Известно, что сервис будет запущен на виртуальной машине с 8 GB оперативной памяти и 8 ядрами процессора. Операционная система — 64-битная Windows Server 2016.

Реализация

Сервис должен представлять собой self-hosted веб-сервер, написанный на C# под .NET 4.6.2.
Можно использовать вот этот шаблон проекта:
https://github.com/DQKrait/Kontur.ImageTransformer

Оформление решения

Пожалуйста, прочитайте этот раздел внимательно!
Проверка решений автоматизирована, и если в оформлении есть ошибки (например, исполняемый файл называется по-другому, или архив имеет другую структуру папок) — решение не будет проверено.

Решение должно представлять собой solution для Visual Studio, упакованный в zip-архив следующим образом:
`/Kontur.ImageTransformer.sln`
`/Kontur.ImageTransformer/Kontur.ImageTransformer.csproj`
…

Папки bin, obj и packages включать в архив не надо.

- Решение не должно требовать никакого предустановленного ПО, кроме .NET Framework 4.6.2.
- Решение может содержать nuget-зависимости.
- Решение может зависеть и от сторонних библиотек не из nuget, в таком случае они должны быть включены в решение (проверьте правильность путей в references).
- Решение может содержать любое количество проектов, но запущен будет исполняемый файл проекта `Kontur.ImageTransformer`.

После сборки командой `nuget restore && msbuild /p:Configuration=Release` должен появиться файл `/Kontur.ImageTransformer/bin/Release/Kontur.ImageTransformer.exe`, который будет использован для запуска приложения.

Сервис должен принимать запросы на префиксе `http://+:8080/`, например `http://localhost:8080/process/grayscale/0,0,20,20`

Пример правильно оформленного архива

Проверка решения

Проверка состоит из двух этапов: полностью автоматическое нагрузочное тестирование и code review.

На первом этапе сервис получит оценку производительности со штрафами за нарушение стабильности. Если на этом этапе решение покажет себя совсем плохо, второго этапа не будет.

На втором этапе будут оценены чистота кода и навыки проектирования архитектуры.

Авторов решений, получивших хорошие оценки на обоих этапах, мы позовем на собеседование.

Вероятные сценарии нагрузочного тестирования

1. Линейный рост нагрузки от 1 rps до 10000 rps. Здесь определяется x — максимальное количество запросов в секунду, которое способен обработать сервис.

2. Постоянная нагрузка x - 10% rps. Измеряется latency (время выполнения) запросов.

3. Постоянная нагрузка x * 2 rps. Измеряется количество успешно обработанных запросов (должно быть ~= x) и latency (должно быть ~= latency при x - 10% rps).

4. Постоянная нагрузка x - 10% rps со всплесками в x * 10 одновременных запросов. Измеряется rps и latency после всплесков (должны вернуться к нормальным показателям для x - 10% rps).

5. Постоянная нагрузка x * 2 rps в течение часа. Измеряется стабильность rps и latency.

Оценка

Пара наблюдений:

1. Все запросы равнозначны.

2. Latency — это важно. Пользователи не любят ждать. Если запрос обрабатывается дольше секунды, уже не важно, закончится он успехом или ошибкой.
При этом лучше, если часть пользователей не получит ответ вообще, но остальные получат ответ быстро, чем если все пользователи получат ответ одинаково медленно.
Для оценки latency в тестах можно использовать 95-й перцентиль.

Примерный вклад различных факторов в итоговую оценку производительности:

- Max RPS — 10%
- Latency — 50%
- Стабильность — 40%

(это распределение может измениться)

API

Сервис должен поддерживать единственный запрос:

`POST /process/<filter>/<coords>`

Body: данные изображения в формате png.
Размер изображения не должен превышать 100KB.
Если в запросе изображение большего размера — нужно отправить пустой ответ с кодом 400 Bad Request.

`<filter>` — строка `grayscale`, `sepia` или `threshold(x)`, где x — целое число от 0 до 100 включительно.
Пример корректной строки `<filter>`: `threshold(30)`
Если `<filter>` не соответствует этому формату — нужно отправить пустой ответ с кодом 400 Bad Request.

`<coords>` — строка вида `x,y,w,h`, где x и y — целые числа, задающие координаты левого верхнего угла прямоугольника, w и h — целые числа, задающие ширину и высоту прямоугольника.

Координаты отсчитываются от левого верхнего угла изображения (левый верхний пиксель имеет координаты 0, 0). Все 4 числа могут быть отрицательными, при этом каждое из них по модулю не превышает 2^{31}. Если прямоугольник выходит за границу изображения, он должен быть обрезан по границе изображения.

Пример корректной строки `<coords>` для изображения размером 100x100:
`-5,-30,200,50`
В этом случае в ответ должна попасть область изображения 0,0,100,20.

Если пересечение прямоугольника с изображением пустое — нужно вернуть пустой ответ с кодом 204 No Content.
Если `<coords>` имеет неправильный формат — нужно отправить пустой ответ с кодом 400 Bad Request.

Если сервис получает запрос другого формата — нужно отправить пустой ответ с кодом 400 Bad Request.

Пример взаимодействия с сервисом

Фильтры

Фильтры преобразуют каждый пиксель изображения по описанным правилам. Если в изображении есть альфа-канал, он должен остаться без изменений.

Все операции производятся в целых числах, если не сказано иное.

Не забывайте, что не все способы работы с изображениями одинаково производительны.

grayscale

```
intensity = (oldR + oldG + oldB) / 3

R = intensity
G = intensity
B = intensity
```

threshold(x)

```
intensity = (oldR + oldG + oldB) / 3

if intensity >= 255 * x / 100
      R = 255
      G = 255
      B = 255
else
      R = 0
      G = 0
      B = 0
```

sepia

Здесь используется single-precision арифметика (float в C#). При присвоении новых значений компонентам цвета дробная часть отбрасывается, если число получилось больше 255 — оно заменяется на 255.

```
R = (oldR * .393) + (oldG * .769) + (oldB * .189)
G = (oldR * .349) + (oldG * .686) + (oldB * .168)
B = (oldR * .272) + (oldG * .534) + (oldB * .131)
```