Задание

Напишите сервер статистики для многопользовательской игры-шутера. Матчи этой игры проходят на разных серверах, и задача сервера статистики — строить общую картину по результатам матчей со всех серверов.
Сервер должен представлять собой standalone-приложение, реализующее описанный ниже RESTful API.

Общая схема работы такая: игровые сервера анонсируют себя advertise-запросами, затем присылают результаты каждого завершенного матча. Сервер статистики аккумулирует разную статистику по результатам матчей и отдает её по запросам (статистика по серверу, статистика по игроку, топ игроков и т.д.).

Нельзя использовать .NET Core.

API

API сервера статистики состоит из следующих методов:

```
/servers/<endpoint>/info PUT, GET
/servers/<endpoint>/matches/<timestamp> PUT, GET

/servers/info GET

/servers/<endpoint>/stats GET
/players/<name>/stats GET

/reports/recent-matches[/<count>] GET
/reports/best-players[/<count>] GET
/reports/popular-servers[/<count>] GET
```

Все данные передаются в формате JSON. Каждый метод проиллюстрирован примером корректного запроса и ответа.

Структура JSON-а будет ясна из примеров: можно считать, что все указанные в примере поля должны содержаться в корректном запросе и других полей там не будет. Типы полей также однозначно определяются по примерам. Если в примере значением поля является целое число, значит оно может быть только целым. Все строковые значения могут состоять из произвольных unicode-символов, если не сказано иное.

Если в описании метода API не указано тело запроса, значит оно должно быть пустым. Если не указано тело ответа, предполагается пустой ответ с кодом 200 OK.

Прием данных от игровых серверов

`PUT /servers/<endpoint>/info (advertise-запрос)`

Запрос:

```
{
      "name": "] My P3rfect Server [",
      "gameModes": [ "DM", "TDM" ]
}
```

Здесь `endpoint` является уникальным идентификатором сервера: при получении нового advertise-запроса с тем же endpoint-ом информация перезаписывается. `endpoint` имеет вид `<ipv4-address>-<port>` или `<hostname>-<port>`.

`PUT /servers/<endpoint>/matches/<timestamp>`

Здесь `timestamp` — строка вида `2017-01-22T15:17:00Z`, задающая момент окончания матча в UTC.

Сервер не должен как-либо привязываться к времени на хосте. Результаты матчей с timestamp-ом из прошлого или из будущего должны корректно сохраняться.
Запросы с одного сервера всегда приходят в порядке возрастания timestamp-ов, но, из-за возможного расхождения времени между серверами, глобальный порядок не гарантируется.
Решение может корректно обрабатывать запросы с меньшим timestamp-ом, чем самый большой на данный момент, но это не является обязательным требованием.

Запрос:

```
{
      "map": "DM-HelloWorld",
      "gameMode": "DM",
      "fragLimit": 20,
      "timeLimit": 20,
      "timeElapsed": 12.345678,
      "scoreboard": [
            {
                  "name": "Player1",
```

```
                "frags": 20,
                "kills": 21,
                "deaths": 3
        },
        {
                "name": "Player2",
                "frags": 2,
                "kills": 2,
                "deaths": 21
        }
    ]
}
```

Здесь `scoreboard` содержит отсортированную по игровым очкам таблицу результатов матча. Победителем матча всегда является первый игрок в `scoreboard`.

Результаты матчей от серверов, не приславших advertise-запрос, не должны сохраняться. Таким серверам нужно отвечать пустым ответом с кодом 400 Bad Request.

Получение текущей информации об игровых серверах

```
GET /servers/<endpoint>/info
```

Ответ:

Этот метод должен вернуть последнюю версию информации, полученную PUT-запросом по этому адресу в том же формате.

Если сервер с таким `endpoint` никогда не присылал advertise-запрос, нужно вернуть пустой ответ с кодом 404 Not Found.

```
GET /servers/info
```

Ответ:

```
[
    {
        "endpoint": "167.42.23.32-1337",
        "info": {
            "name": "] My P3rfect Server [",
            "gameModes": [ "DM", "TDM" ]
        }
```

```
	},
	{
		"endpoint": "62.210.26.88-1337",
		"info": {
			"name": ">> Sniper Heaven <<",
			"gameModes": [ "DM" ]
		}
	}
]
```

Ответ должен содержать последнюю версию информации о всех серверах, когда-либо присылавших advertise-запрос.

`GET /servers/<endpoint>/matches/<timestamp>`

Ответ:

Этот метод должен вернуть информацию о матче, полученную PUT-запросом по этому адресу в том же формате.

Если PUT-запроса по этому адресу не было, нужно вернуть пустой ответ с кодом 404 Not Found.

Получение статистики

Для методов из этой категории (имеются в виду методы `*/stats` и `reports/*`) скорость ответа важнее его актуальности. **Считается допустимым, если в статистике не будут учтены результаты матчей, присланные за последнюю минуту**.

Код подсчета статистики должен быть написан с заделом на возможное расширение: возможно добавление новых полей в методах `*/stats` и новых методов в категории `reports/*`.

`GET /servers/<endpoint>/stats`

Ответ:

```
{
	"totalMatchesPlayed": 100500,
	"maximumMatchesPerDay": 33,
	"averageMatchesPerDay": 24.456240,
	"maximumPopulation": 32,
```

```
      "averagePopulation": 20.450000,
      "top5GameModes": [ "DM", "TDM" ],
      "top5Maps": [
            "DM-HelloWorld",
            "DM-1on1-Rose",
            "DM-Kitchen",
            "DM-Camper Paradise",
            "DM-Appalachian Wonderland",
      ]
}
```

Списки `top5GameModes` и `top5Maps` должны быть упорядочены по убыванию популярности (чем чаще встречается режим игры или карта среди всех матчей, тем он/она популярнее).

`maximumMatchesPerDay, averageMatchesPerDay` — максимальное и среднее количества сыгранных матчей на сервере за один календарный день по UTC (с 00:00Z до 00:00Z следующего дня). **Матч, сыгранный на границе дней, относится к тому дню, где он закончился.**
Для расчета среднего используется количество дней от первого матча на этом сервере до последнего матча среди всех серверов.

`maximumPopulation, averagePopulation` — максимальное и среднее количества игроков, принявших участие в одном матче.

```
GET /players/<name>/stats
```

Здесь `name` — urlencoded имя игрока.

Ответ:

```
{
      "totalMatchesPlayed": 100500,
      "totalMatchesWon": 1000,
      "favoriteServer": "62.210.26.88-1337",
      "uniqueServers": 2,
      "favoriteGameMode": "DM",
      "averageScoreboardPercent": 76.145693,
      "maximumMatchesPerDay": 33,
      "averageMatchesPerDay": 24.456240,
      "lastMatchPlayed": "2017-01-22T15:11:12Z",
      "killToDeathRatio": 3.124333
}
```

`averageMatchesPerDay`: для расчета среднего используется количество дней от первого матча этого игрока до последнего матча среди всех матчей.

`averageScoreboardPercent` считается так:

Для конкретного матча `scoreboardPercent = playersBelowCurrent / (totalPlayers - 1) * 100%`.

Пример 1, в таблице 4 игрока:

Player1 — 100%
Player2 — 66.666667%
Player3 — 33.333333%
Player4 — 0%

Пример 2, в таблице 3 игрока:

Player1 — 100%
Player2 — 50%
Player3 — 0%

Если в матче один игрок, scoreboardPercent = 100%.

`averageScoreboardPercent` — это средний `scoreboardPercent` данного игрока по всем сыгранным матчам.

`favoriteServer` — сервер, на котором игрок появлялся чаще всего.

`uniqueServers` — количество уникальных серверов, на которых появлялся игрок.

`favoriteGameMode` — режим игры, в матчах с которым чаще всего участвовал игрок.

Имена игроков должны сравниваться без учета регистра.

`GET /reports/recent-matches[/<count>]`

Параметр `count` в этом и следующих методах задает число записей, которые нужно включить в отчет. Если записей меньше, нужно включить в отчет все. **Параметр необязательный и по умолчанию считается равным 5. Также count не может превосходить 50. Если в запросе он больше 50, нужно считать его равным 50, а если равен 0 или меньше 0 — вернуть пустой массив** `[]`**.**

Ответ:

```
[
	{
		"server": "62.210.26.88-1337",
		"timestamp": "2017-01-22T15:11:12Z",
		"results": {
			"map": "DM-HelloWorld",
			"gameMode": "DM",
			"fragLimit": 20,
			"timeLimit": 20,
			"timeElapsed": 12.345678,
			"scoreboard": [
				{
					"name": "Player1",
					"frags": 20,
					"kills": 21,
					"deaths": 3
				},
				{
					"name": "Player2",
					"frags": 2,
					"kills": 2,
					"deaths": 21
				}
			]
		}
	},
	...
]
```

Здесь `timestamp` — время окончания матча в UTC, указанное в URL при загрузке его результатов.

Последние матчи должны отсчитываться от матча с самым большим timestamp-ом, а не от текущего времени на хосте.

```
GET /reports/best-players[/<count>]
```

Ответ:

```
[
	{
```

```
            "name": "Player1",
            "killToDeathRatio": 3.124333
      },
      ...
]
```

Здесь игроки должны быть отсортированы по убыванию `killToDeathRatio`. **При этом нужно игнорировать игроков, сыгравших менее 10 матчей, а также игроков, которые ни разу не умирали.**

`killToDeathRatio = totalKills / totalDeaths`, где `totalKills` — сумма `kills` игрока по всем сыгранным матчам, `totalDeaths` — сумма `deaths` игрока по всем сыгранным матчам.

`GET /reports/popular-servers[/<count>]`

Ответ:

```
[
      {
            "endpoint": "62.210.26.88-1337",
            "name": ">> Sniper Heaven <<",
            "averageMatchesPerDay": 24.456240
      },
      ...
]
```

Здесь сервера должны быть отсортированы по убыванию `averageMatchesPerDay`.

Требования к решению

- Сервер должен реализовывать описанное выше API.
- Сервер должен поддерживать работу на произвольном HTTP-префиксе (задается при запуске).
- Сервер не должен терять данные при перезапуске. Если на PUT-запрос сервер вернул 200 OK, данные не должны быть потеряны даже в случае аварийного завершения приложения. Допустима потеря данных при внезапном выключении хоста.
  Для хранения данных можно использовать как собственное решение, так и готовую embedded-базу данных. В качестве БД не рекомендуется использовать MSSQL LocalDB.

- Сервер не должен падать при получении некорректных запросов. На такие запросы нужно отправлять ответ с кодом, отличным от 200 OK. Допустимы любые коды вида 4xx и 5xx.
- Сервер должен быстро обслуживать все виды запросов.
- Сервер должен уметь логировать ошибки.
- Код должен быть покрыт тестами.
- Код должен быть написан аккуратно и с любовью к деталям.

Оформление решения

- Все внешние модули, от которых зависит решение, должны содержаться в папке с решением. Также допускаются зависимости от nuget-модулей.
- Решение должно успешно собираться с помощью msbuild: `nuget restore && msbuild /p:Configuration=Release`. В результате сборки папка `Kontur.GameStats.Server/bin/Release` внутри папки с решением должна содержать всё необходимое для работы приложения.
- Сервер должен запускаться из папки `bin/Release` следующим образом: `Kontur.GameStats.Server.exe --prefix http://+:8080/`, где вместо `http://+:8080/` может быть любой UrlPrefix.
- Можно использовать в качестве заготовки проект из репозитория на гитхабе: https://github.com/DQKrait/Kontur.GameStats

Отправка решения

- Решение можно отправить на проверку через форму подачи заявки на стажировку.
- До дедлайна можно присылать новые версии решения, оценена будет последняя.

Оценка решений

Порядок проверки решения примерно такой, но может незначительно поменяться на практике:

1. Проверка требований к оформлению решения. При несоответствии дальнейшие этапы проверки не проводятся.
2. Оценка полноты реализации API.
3. Оценка полноты соответствия требованиям к решению.
4. Нагрузочное тестирование. Примерный объем нагрузки описан ниже.
5. Оценка качества кода (для решений, достойно прошедших оценку по критериям 1-4).

Нагрузочное тестирование

В распоряжении вашего сервера будет:

- Xeon E5-2650 @ 2.30 GHz
- 8 GB памяти
- 80 GB диска

Примерные ориентиры по нагрузке в худшем случае:

- Среднее число игроков в одном матче: ~50
- Максимальное число игроков в одном матче: ~100
- Общее количество серверов: ~10 000
- Всего уникальных игроков: ~1 000 000
- Всего дней в истории: ~14
- Среднее число матчей в день на одном сервере: ~100
- Количество уникальных режимов игры вряд ли превысит 10.
- Длина названия режима игры почти всегда не превосходит 3-х символов.
- Длина имени сервера, игрока или карты почти всегда не превосходит 50 символов.