**Всего дано в API 3 группы endpointов (информация из swagger)**

**Группа эндпоинтов Cities (таблица с городами)**

*Варианты запросов:*

1. GET /Cities/{id} - получение города по id - самый частый запрос при проверке прогноза Weather Forecast (получаем имя города по id). Используется часто. Не тяжелый

http://91.185.85.213/Cities/4

{

"id": 4,

"name": "Саратов"

}

2. GET /Cities - получение списка городов - если пользователь хочет посмотреть погоду в определенном городе. Используется реже. Тяжелый

http://91.185.85.213/Cities

[

{

"id": 1,

"name": "Москва"

},

{

"id": 2,

"name": "Санкт Петербург"

},...

3. PUT /Cities/{id} - очень редкий, при изменении названия или id города (выполняется только администратором). Разовый. Не тяжелый

http://91.185.85.213/Cities/169

{

"id": 169,

"name": "Самара"

}

4. POST /Cities - не работает (not allow)

**Группа эндпоинтов Forecast**

*Варианты запросов:*

1. GET /Forecast/{id} - получение прогноза по id - самый частый запрос при проверке прогноза Weather Forecast (получаем имя города по id). Используется часто. Не тяжелый

http://91.185.85.213/Forecast/4

{

"id": 4,

"cityId": 42,

"dateTime": 202311281130,

"temperature": 5,

"summary": ok

}

2. PUT /Forecast/{id} - записываем данные в таблицу с соответствующим id (cityId,dateTime, temp, summary). Например каждый день обновляется погода. Используется не часто. (Тяжелый зависит от частоты обновления погоды).

http://91.185.85.213/Forecast/4

..

3. POST /Forecast/{cityId} - получаем id прогноза по cityId. Часто, не тяжелый

http://91.185.85.213/Forecast/4

{

"id": 4

}

4. GET /Forecast - список всех записей о прогнозах во всех городах. Используется редко. Тяжелый

[

{

"id": 1,

"cityId": 777,

"dateTime": 202311301130,

"temperature": 99,

"summary": "okey"

},

......

**Группа эндпоинтов WeatherForecast**

*Запросы:*

1. GET /WeatherForecast - возвращает всех городов + всех прогнозов из обоих таблиц. Самый тяжелый запрос, который возвращает все данные из всех таблиц. Возможно программистам стоит его оптимизировать

{

"id": 4,

"name": "Самара",

"forecast": [

{

"id": 5,

"cityId": 4,

"dateTime": 202311031500,

"temperature": -3,

"summary": cold

},

{

"id": 641,

"cityId": 4,

"dateTime": 202310281130,

"temperature": 12,

"summary": warm

}

]

},

...

**План тестов:**

По группе Cities тестируем:

GET /Cities

GET /Cities/{id}

PUT /Cities/{id}

По сценарию:

20% - GET возьмем тяжелый запрос, запрос списка всех городов

75% - GET {id} легкий запрос название города по Id

5% - PUT {id} изменение в базе City

.randomSwitch(

20.0 -> exec(Cities.getCities),

75.0 -> exec(Cities.getCitiesId),

5.0 -> exec(Cities.updateCityById)

)

По группе Forecast тестируем:

GET /Forecast

GET /Forecast/{id}

PUT /Forecast/{id}

POST /Forecast/{cityId}

По сценарию:

10% - GET возьмем тяжелый запрос, запрос всех записей о прогнозах во всех городах

40% - GET {id} получение прогноза по id

40% - POST {id} получаем id прогноза по cityId

10% - PUT {id} записываем данные прогноза погоды

  .randomSwitch(

  10.0 -> exec(Forecast.getForecast),

  40.0 -> exec(Forecast.getForecastById),

  40.0 -> exec(Forecast.postForecastById),

  10.0 -> exec(Forecast.updateForecastById)

)

По группе WeatherForecast тестируем:

GET /WeatherForecast

По сценарию:

100% получаем данные по запросу GET

.exec(WeatherForecast.WeatherFc)

Напишем первый сценарий:

CitiesScenario.scn.inject(

      rampUsersPerSec(0) to 5 during(1 minute),

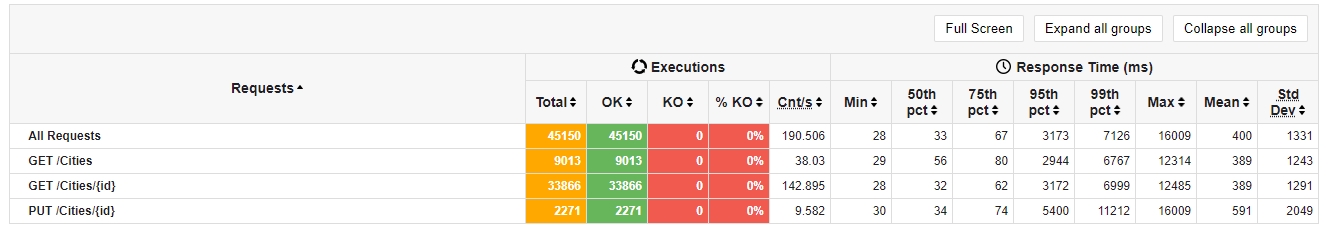
      // полка

      constantUsersPerSec(250) during(3 minutes)

Запустим Stability для Cities:

Результат:





Запустим Stability для Forecast:

  .randomSwitch(

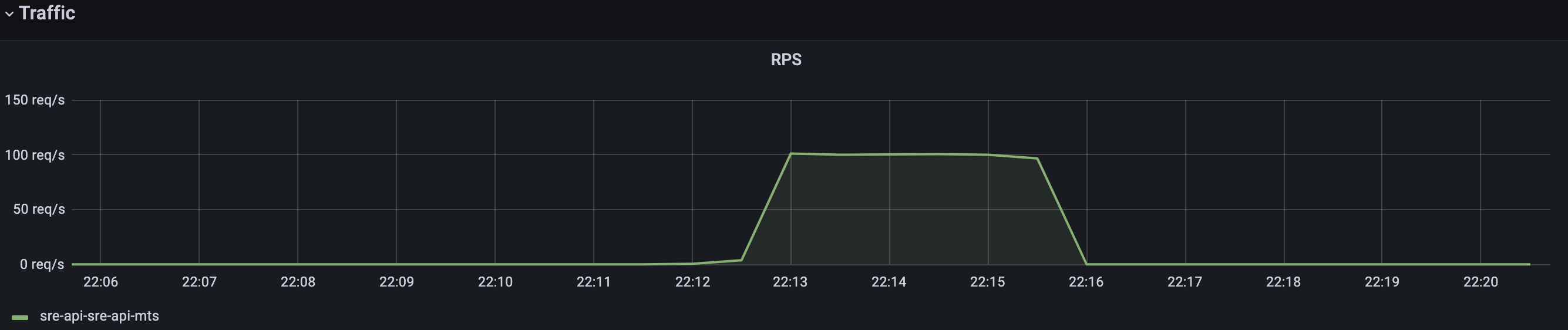
  10.0 -> exec(Forecast.getForecast),

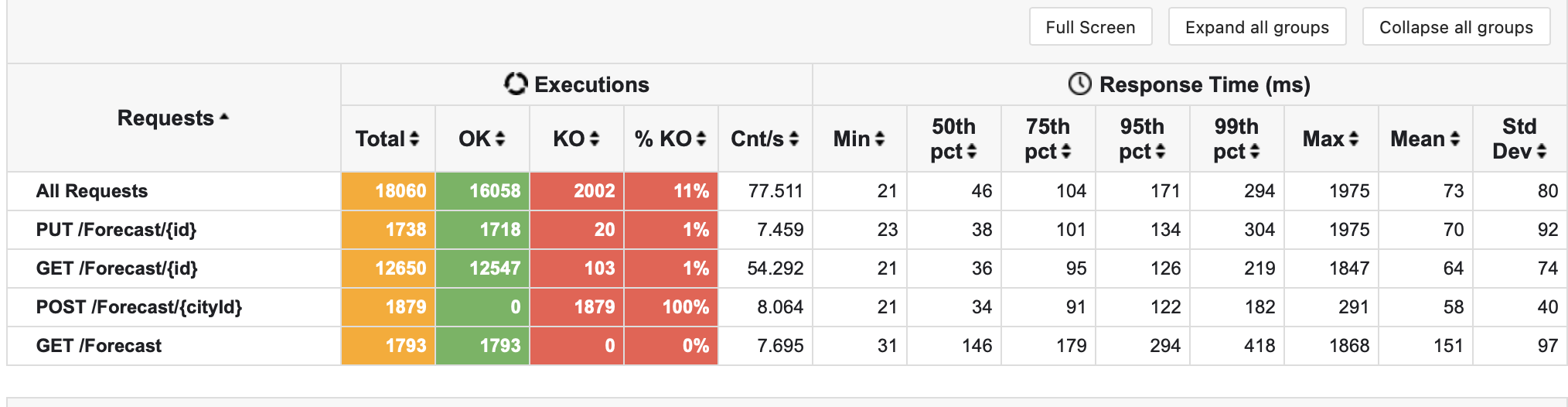
  70.0 -> exec(Forecast.getForecastById),

  10.0 -> exec(Forecast.postForecastById),

  10.0 -> exec(Forecast.updateForecastById)

Результат:





Запустим Stability для Forecast:

  WeatherForecastScenario.scn.inject(

// разгон

rampUsersPerSec(0) to 2 during(10 seconds),

// полка

constantUsersPerSec(10) during(3 minutes)

Результат на грани. Чуть больше запросов, и начинаются ошибки

