Rakamin Academy

AN NISA DIRA

Project-Based Internship:

Software Quality Assurance

Evermos x Rakamin Academy

Table of Contents

CONTENTS

INTRODUCTION

This Program Project-Based Internship will provide me with virtual internship experience as a Quality Assurance at Evermos. I will be positioned as a Quality Assurance Intern who will be exposed to issues, case studies, and projects that are part of Evermos' daily operations.

Quality Assurance is a series of systematic processes to determine whether a product or service meets specified requirements. QA determines and sets requirements for creating or developing specific products to ensure they have good quality.

QA can be integrated into all stages of development and used even after the post-release stage. QA specialists create and implement various strategies for software quality improvement using SDLC methods. They implement the necessary types of testing to ensure that the software will function correctly.

INTEGRATION TESTING

Create test scenarios for integration tests from these 2 APIs and implement them in K6 and add assertions from each test carried out in 1 test file.

- API Create
  - Base url : https://reqres.in
  - Path url : /api/users
  - Method : POST
  - Header : application/json
  - Request body :

```
{
  "name": "morpheus",
  "job": "leader"
}
```

- API Update
  - Base url : https://reqres.in
  - Path url : /api/users/2
  - Method : PUT
  - Header : application/json
  - Request body :

```
{
  "name": "morpheus",
  "job": "zion resident"
}
```

```
const BASE_URL = 'https://reqres.in';
const USERS_ENDPOINT = '/api/users';
```

Variabel dan Konstanta

- **BASE_URL**: Merupakan URL dasar dari API yang akan diuji.
- **USERS_ENDPOINT**: Merupakan endpoint untuk operasi CRUD pengguna.

```
export const options = {
  stages: [
    { duration: '10s', target: 5 }, // Ramp-up phase
    { duration: '20s', target: 5 }, // Steady state
    { duration: '10s', target: 0 }, // Ramp-down phase
  ],
};
```

Opsi Penguji

options: Objek ini mendefinisikan konfigurasi pengujian. Ini termasuk tahapan pengguna yang berbeda selama pengujian, yang terdiri dari fase peningkatan pengguna, fase stabil, dan fase penurunan pengguna.

Skenario 1: Create User

Bagian ini adalah implementasi dari pengiriman permintaan HTTP POST untuk membuat pengguna baru (create user) menggunakan k6.

```
const createUserPayload = {
  name: 'morpheus',
  job: 'leader',
};

const createUserRes = http.post(
  `${BASE_URL}${USERS_ENDPOINT}`,
  JSON.stringify(createUserPayload),
  {
    headers: { 'Content-Type': 'application/json' },
  }
);
```

createUserPayload: Variabel ini berisi objek yang mendefinisikan data yang akan dikirimkan dalam permintaan untuk membuat pengguna baru.

Permintaan ini dikirim ke URL lengkap yang disusun dari **BASE_URL** dan **USERS_ENDPOINT**, dengan menambahkan **BASE_URL** sebagai awalan dan **USERS_ENDPOINT** sebagai bagian alamat yang spesifik untuk operasi pembuatan pengguna.

Data **createUserPayload** yang telah dibuat sebelumnya diubah menjadi string JSON sebelum dikirimkan dalam permintaan **POST**.

Assertion untuk pengujian API Create

```
check(createUserRes, {
  'Create User Status is 201': (res) => res.status === 201,
  'Create User Response includes ID': (res) =>
    JSON.parse(res.body).hasOwnProperty('id'),
});
```

- **'Create User Status is 201'**: Memeriksa apakah status respons adalah 201 Created, yang menunjukkan bahwa permintaan berhasil.
- **'Create User Response includes ID'**: Memeriksa apakah respons memiliki properti 'id', yang menunjukkan bahwa pengguna telah berhasil dibuat dan respons mengandung informasi tentang ID pengguna yang baru saja dibuat.

Skenario 2: Update User

Bagian ini adalah implementasi dari pengiriman permintaan HTTP PUT untuk memperbarui pengguna yang sudah ada (update user) menggunakan k6.

```
const updateUserPayload = {
  name: 'morpheus',
  job: 'zion resident',
};

const updateUserRes = http.put(
  `${BASE_URL}${USERS_ENDPOINT}/2`,
  JSON.stringify(updateUserPayload),
  {
    headers: { 'Content-Type': 'application/json' },
  }
);
```

updateUserPayload: Variabel ini berisi objek yang mendefinisikan data yang akan dikirimkan dalam permintaan untuk memperbarui pengguna. .

Permintaan ini dikirim ke URL lengkap yang disusun dari **BASE_URL**, **USERS_ENDPOINT**, dan **2** yang menunjukkan ID pengguna yang akan diperbarui.

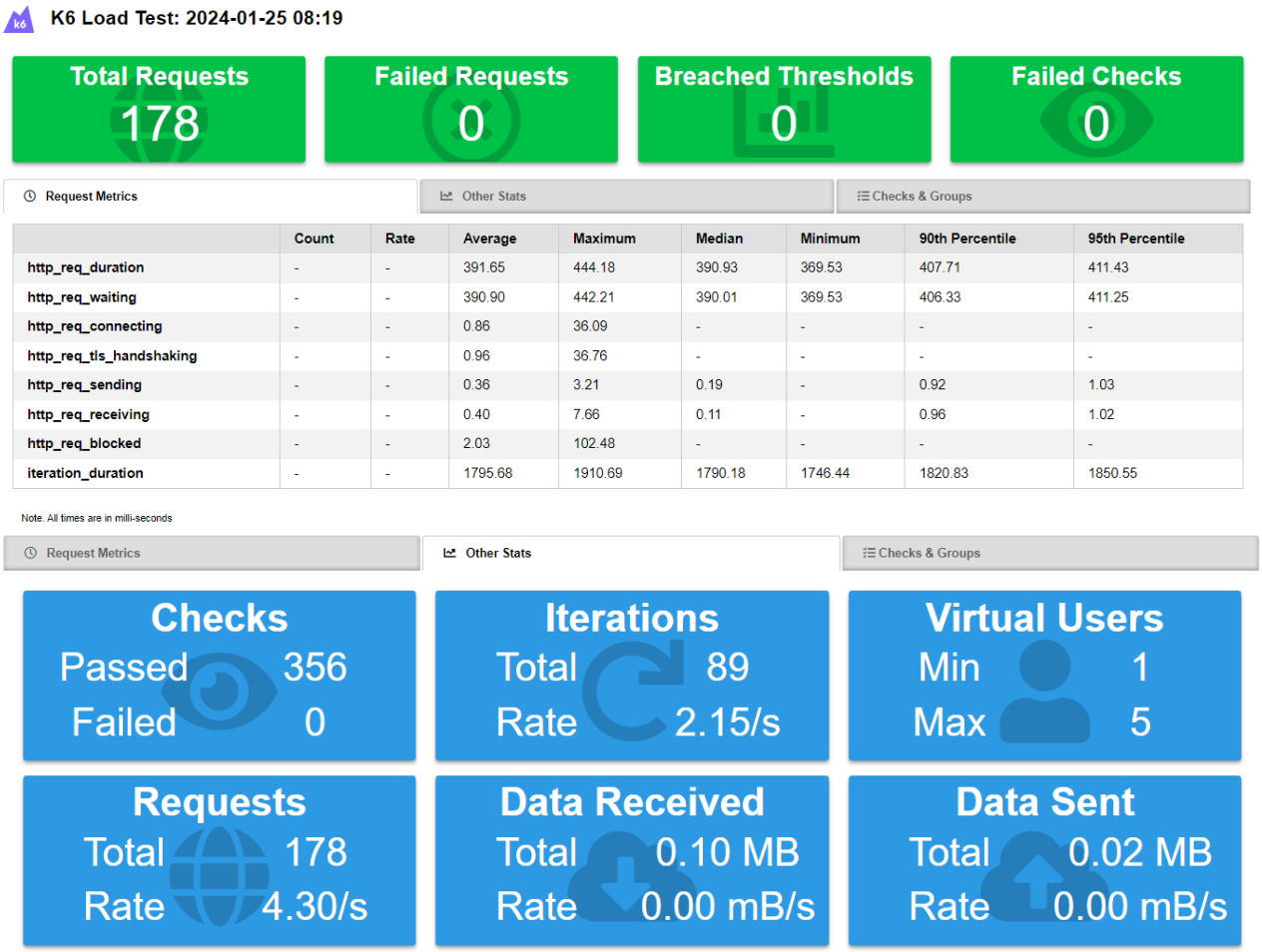
Data **updateUserPayload** yang telah dibuat sebelumnya diubah menjadi string JSON sebelum dikirimkan dalam permintaan PUT.

Assertion untuk pengujian API Update

```
check(updateUserRes, {
  'Update User Status is 200': (res) => res.status === 200,
  'Update User Response includes Updated At': (res) =>
    JSON.parse(res.body).hasOwnProperty('updatedAt'),
});
```

- **'Update User Status is 200'**: Memeriksa apakah status respons adalah 200 OK, yang menunjukkan bahwa permintaan untuk memperbarui pengguna berhasil.
- **'Update User Response includes Updated At**': Memeriksa apakah respons memiliki properti 'updatedAt', yang menunjukkan bahwa data pengguna telah diperbarui dan respons mengandung informasi tentang waktu terakhir pengguna diperbarui.

Result Integration Testing



K6 Load Test: 2024-01-25 08:19

Total Requests	Failed Requests	Breached Thresholds	Failed Checks
178	0	0	0

Request Metrics | Other Stats | Checks & Groups

	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
http_req_duration	-	-	391.65	444.18	390.93	369.53	407.71	411.43
http_req_waiting	-	-	390.90	442.21	390.01	369.53	406.33	411.25
http_req_connecting	-	-	0.86	36.09	-	-	-	-
http_req_tls_handshaking	-	-	0.96	36.76	-	-	-	-
http_req_sending	-	-	0.36	3.21	0.19	-	0.92	1.03
http_req_receiving	-	-	0.40	7.66	0.11	-	0.96	1.02
http_req_blocked	-	-	2.03	102.48	-	-	-	-
iteration_duration	-	-	1795.68	1910.69	1790.18	1746.44	1820.83	1850.55

Note. All times are in milli-seconds

Request Metrics | Other Stats | Checks & Groups

Checks	Iterations	Virtual Users
Passed 356	Total 89	Min 1
Failed 0	Rate 2.15/s	Max 5

Requests	Data Received	Data Sent
Total 178	Total 0.10 MB	Total 0.02 MB
Rate 4.30/s	Rate 0.00 mB/s	Rate 0.00 mB/s

- Tidak ada permintaan yang gagal selama pengujian. Semua 178 permintaan berhasil dilakukan tanpa kegagalan.
- Tidak ada metrik yang melebihi ambang batas yang ditetapkan selama pengujian. Semua metrik berada dalam batas yang diharapkan.
- Tidak ada pemeriksaan (checks) yang gagal selama pengujian. Semua pemeriksaan berhasil dilakukan tanpa masalah.

Detail Iterasi:

- Total Permintaan: 178 dengan kecepatan 4.295529/s.
- Virtual Users: 89 dalam tes.
- Durasi Iterasi: Rata-rata 1.795s, maksimum 1.91s. Persentil ke-90 dan ke-95: 1.825s dan 1.85s.

Dengan demikian, hasil pengujian menunjukkan bahwa sistem yang diuji berkinerja baik dan memenuhi semua kriteria yang diharapkan selama pengujian. Semua permintaan berhasil, tidak ada metrik yang melebihi ambang batas, dan semua pemeriksaan dilakukan dengan sukses.

PERFORMANCE TESTING

Create a test scenario to test the performance of these 2 APIs with a total of 1000 virtual users, 3500 iterations, and a maximum API response tolerance limit of 2 seconds and add assertions from each test carried out in 1 test file.

- API Create
  - Base url : https://reqres.in
  - Path url : /api/users
  - Method : POST
  - Header : application/json
  - Request body :

```
{
    "name": "morpheus",
    "job": "leader"
}
```

- API Update
  - Base url : https://reqres.in
  - Path url : /api/users/2
  - Method : PUT
  - Header : application/json
  - Request body :

```
{
    "name": "morpheus",
    "job": "zion resident"
}
```

```
export const options = {
    vus: 1000,
    iterations: 3500,
    thresholds: {
        http_req_duration: ['max<=2000'],
    },
};
```

- **vus: 1000**: Ini menetapkan jumlah pengguna virtual (Virtual Users - VUs) yang akan digunakan selama pengujian. Dalam hal ini, jumlahnya adalah 1000.
- **iterations: 3500**: Ini menetapkan jumlah iterasi atau siklus yang akan dieksekusi selama pengujian. Dalam hal ini, ada 3500 iterasi.
- **thresholds**: Ini adalah kriteria atau ambang batas yang digunakan untuk mengevaluasi kinerja aplikasi selama pengujian. Dalam hal ini, kita memiliki satu ambang batas yang dinamai **http_req_duration**. Ambang batas ini menetapkan bahwa durasi maksimum dari permintaan HTTP (http_req_duration) tidak boleh melebihi **2000** milidetik (2 detik).

Skenario 1: Performance Test for API Create

Bagian ini adalah bagian dari sebuah loop yang melakukan percobaan untuk mengirim permintaan API Create.

```
for (let i = 0; i < 2; i++) {
    // Performance Test Scenario for API Create
    const createPayload = {
        name: 'morpheus',
        job: 'leader',
    };

    const createRes = http.post('https://reqres.in/api/users',
    JSON.stringify(createPayload),
    {
        headers: { 'Content-Type': 'application/json' },
    });

    if (createRes.status === 201 && createRes.json().hasOwnProperty('id'))
    {
        check(createRes, {
            'Create User Status is 201': () => true,
            'Create User Response includes ID': () => true,
        });
        break; // Exit the loop if the request is successful
    } else {
        console.warn(`Attempt ${i + 1} failed. Retrying...`);
    }
}
```

Loop ini akan dieksekusi dua kali karena batasan iterasi adalah < 2. Ini berarti akan ada dua percobaan untuk mengirim permintaan API Create.

Pengecekan Respons:

- Setelah permintaan dikirim, kondisi if digunakan untuk memeriksa apakah status respons adalah 201 (Created) dan apakah respons JSON mencakup properti 'id'.
- Jika kondisi tersebut terpenuhi, **asertasi** (assertion) akan dilakukan menggunakan fungsi check(). Dua kondisi yang diverifikasi adalah apakah status respons adalah 201 dan apakah respons mencakup properti 'id'.
- Jika asertasi berhasil, loop akan dihentikan dengan menggunakan break. Ini berarti bahwa permintaan berhasil dan tidak perlu dilakukan percobaan lagi.
- Jika kondisi asertasi tidak terpenuhi, pesan log akan dicetak yang menunjukkan bahwa percobaan gagal dan program akan melanjutkan ke iterasi berikutnya.

Skenario 2: Performance Test for API Update

Bagian ini adalah bagian dari sebuah loop yang melakukan percobaan untuk mengirim permintaan API Update.

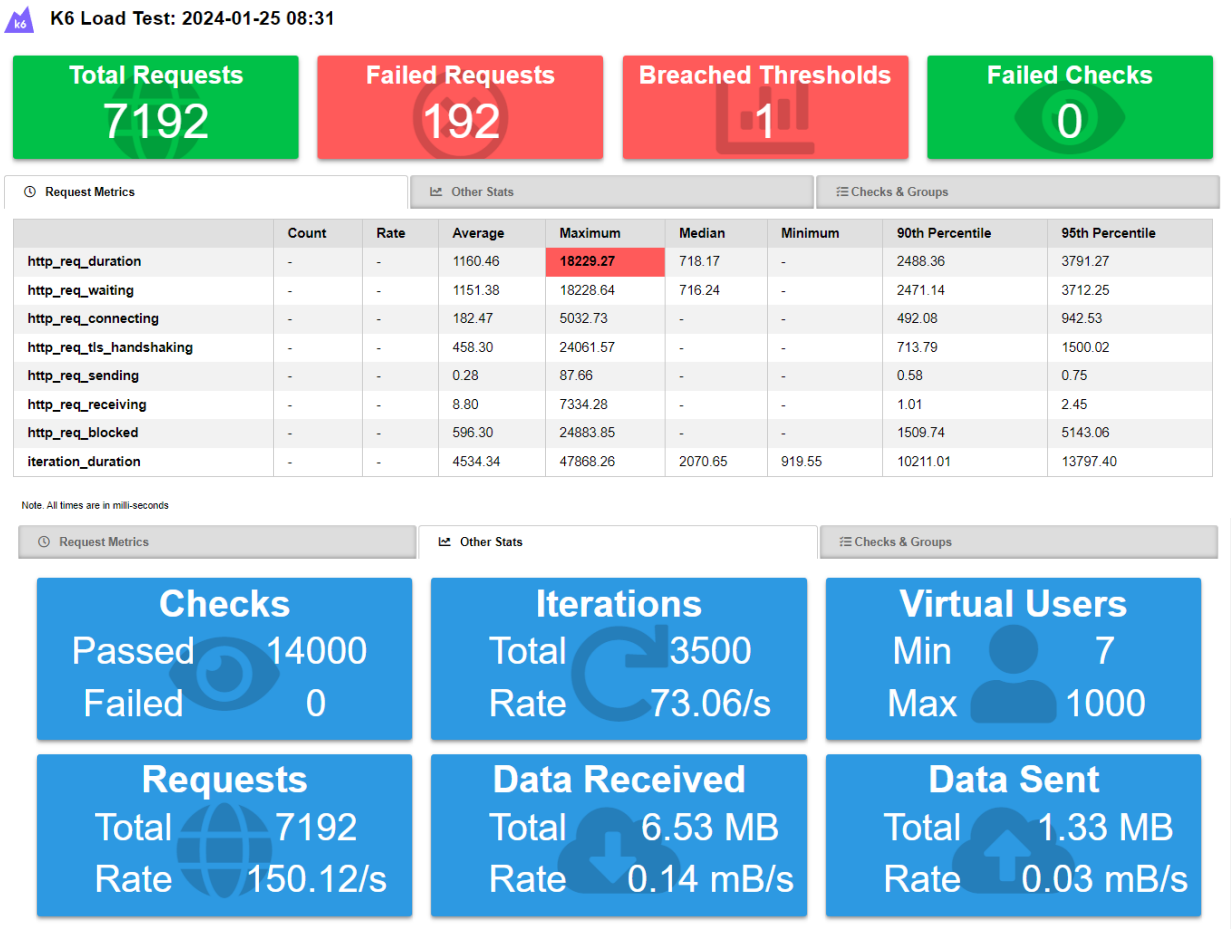
```
for (let i = 0; i < 2; i++) {
    // Performance Test Scenario for API Update
    const updatePayload = {
        name: 'morpheus',
        job: 'zion resident',
    };

    const updateRes = http.put('https://reqres.in/api/users/2',
    JSON.stringify(updatePayload), {
        headers: { 'Content-Type': 'application/json' },
    });

    if (updateRes.status === 200 && updateRes.json().hasOwnProperty('updatedAt'))
    {
        check(updateRes, {
            'Update User Status is 200': () => true,
            'Update User Response includes Updated At': () => true,
        });
        break; // Exit the loop if the request is successful
    } else {
        console.warn(`Attempt ${i + 1} failed. Retrying...`);
    }
}
```

Loop ini akan dieksekusi dua kali karena batasan iterasi adalah < 2. Ini berarti akan ada dua percobaan untuk mengirim permintaan API Update.

Pengecekan Respons:

- Setelah permintaan dikirim, dilakukan pengecekan menggunakan kondisi if untuk memastikan bahwa status respons adalah 200 (OK) dan bahwa respons JSON mencakup properti 'updatedAt'.
- Jika kedua kondisi tersebut terpenuhi, dilakukan **asertasi** (assertion) menggunakan fungsi check(). Dua kondisi yang diverifikasi adalah apakah status respons adalah 200 dan apakah respons mencakup properti 'updatedAt'.
- Jika asertasi berhasil, loop akan dihentikan dengan menggunakan break. Ini menunjukkan bahwa permintaan berhasil dan tidak perlu dilakukan percobaan lagi.
- Jika kondisi asertasi tidak terpenuhi, pesan log akan dicetak yang menunjukkan bahwa percobaan gagal, dan program akan melanjutkan ke iterasi berikutnya.

Result Performance Testing



K6 Load Test: 2024-01-25 08:31

Total Requests	Failed Requests	Breached Thresholds	Failed Checks
7192	192	1	0

Request Metrics | Other Stats | Checks & Groups

	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
http_req_duration	-	-	1160.46	18229.27	718.17	-	2488.36	3791.27
http_req_waiting	-	-	1151.38	18228.64	716.24	-	2471.14	3712.25
http_req_connecting	-	-	182.47	5032.73	-	-	492.08	942.53
http_req_tls_handshaking	-	-	458.30	24061.57	-	-	713.79	1500.02
http_req_sending	-	-	0.28	87.66	-	-	0.58	0.75
http_req_receiving	-	-	8.80	7334.28	-	-	1.01	2.45
http_req_blocked	-	-	596.30	24883.85	-	-	1509.74	5143.06
iteration_duration	-	-	4534.34	47868.26	2070.65	919.55	10211.01	13797.40

Note. All times are in milli-seconds

Request Metrics | Other Stats | Checks & Groups

Checks		Iterations		Virtual Users	
Passed	14000	Total	3500	Min	7
Failed	0	Rate	73.06/s	Max	1000

Requests		Data Received		Data Sent	
Total	7192	Total	6.53 MB	Total	1.33 MB
Rate	150.12/s	Rate	0.14 mB/s	Rate	0.03 mB/s

- Terdapat 192 permintaan yang gagal selama pengujian.
- Terdapat 1 permintaan yang melampaui batas threshold yang ditetapkan.
- Tidak ada permintaan yang gagal dalam pemeriksaan.

Detail Iterasi:

- Total Permintaan: 7192 dengan kecepatan 150.121432/s.
- Total Permintaan Gagal: 192 dari 7000 permintaan, dengan tingkat kegagalan sebesar 2.66%.
- Total Virtual Users: 7, dengan rentang dari 7 hingga 1000.
- Batasan metrik '**http_req_duration**' terlampaui. Rata-rata durasi permintaan adalah 1160.46 milidetik, dengan durasi maksimum sebesar 18229.27 milidetik.

Dari hasil ini, terlihat bahwa meskipun sebagian besar operasi berhasil, ada kegagalan serta beberapa metrik kinerja yang perlu dipertimbangkan untuk perbaikan. Evaluasi lanjutan diperlukan untuk mengidentifikasi penyebab masalah dan meningkatkan kinerja sistem secara keseluruhan.

CONCLUSIONS

- Kualitas Operasi: Pada integration testing, operasi pembuatan dan pembaruan pengguna berjalan lancar tanpa kegagalan, sementara pada performance test, terdapat kegagalan dalam jumlah permintaan.

- Kinerja Aplikasi: Meskipun kedua tes menunjukkan kinerja yang responsif, performance test menunjukkan variasi yang lebih besar dalam metrik kinerja dan terdapat batasan yang terlampaui.

- Keseragaman Kinerja: Integration test menunjukkan kinerja yang lebih konsisten dan stabil dibandingkan dengan performance test.

- Peringatan dan Perbaikan: Performance test memberikan peringatan terkait kegagalan dan batasan yang terlampaui, yang mengindikasikan adanya masalah yang perlu diperbaiki atau dioptimalkan.

Dalam keseluruhan, integration test menunjukkan kinerja yang lebih baik dan stabil dibandingkan dengan performance test. Hal ini menekankan pentingnya analisis kinerja secara menyeluruh dan identifikasi masalah untuk meningkatkan performa aplikasi secara keseluruhan.