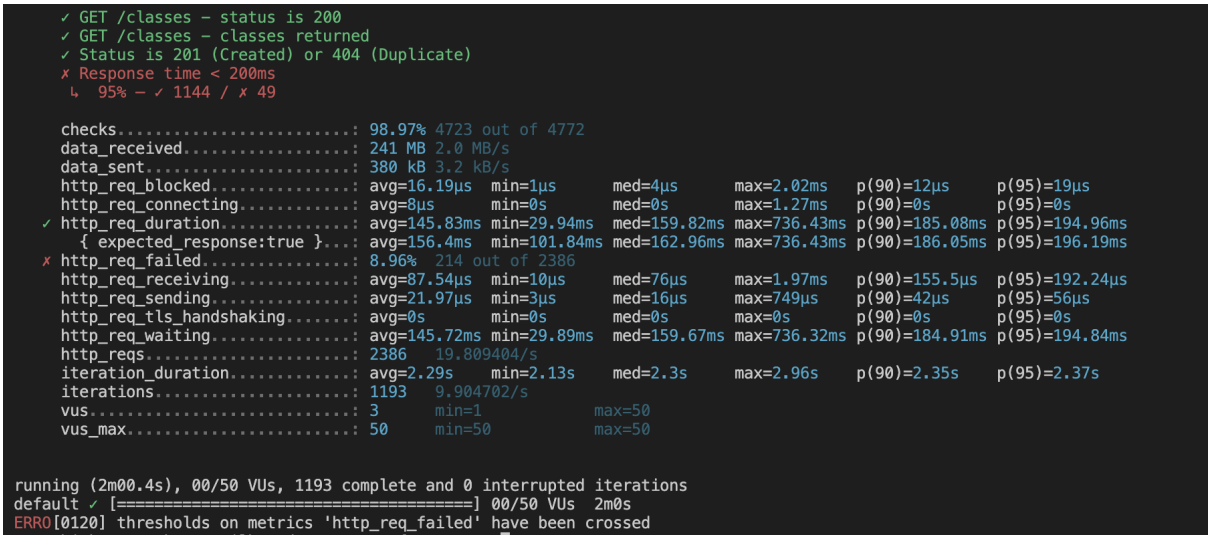
Stress performance testing

Load Test for get Classes and book a class

```
export const options = {
  stages: [
    { duration: "30s", target: 10 }, // Ramp up to 10 users
    { duration: "1m", target: 50 }, // Sustain 50 users
    { duration: "30s", target: 0 }, // Ramp down
  ],
  thresholds: {
    http_req_duration: ["p(95)<200"], // 95% of requests should complete within 200ms
    http_req_failed: ["rate<0.01"], // Less than 1% of requests should fail
  },
};

function testGetClasses() {
  const res = http.get("http://localhost:8080/classes");

  check(res, {
    "GET /classes - status is 200": (r) => r.status === 200,
    "GET /classes - classes returned": (r) => {
      const body = r.json();
      return Array.isArray(body) && body.length > 0;
    },
  });

  sleep(1); // Simulate user delay
}
```

```
function testPostBookings() {
  const payload = JSON.stringify({
    ClassID: Math.ceil(Math.random() * 10),
    BookingDate: new Date().toISOString(),
    Status: "CONFIRMED",
    MemberID: Math.floor(Math.random() * (1027 - 52 + 1)),
  });

  const res = http.post("http://localhost:8080/bookings", payload, {
    headers: { "Content-Type": "application/json" },
  });

  if (res.status === 201) {
    console.log(`Booking created successfully: ${JSON.stringify(res.json())}`);
  } else if (res.status === 404) {
    console.warn(
      `Duplicate booking detected: MemberID ${memberID}, ClassID ${classID}`
    );
  } else {
    console.error(`Unexpected error: ${res.status} - ${res.body}`);
  }

  sleep(1); // Simulate user delay
}
```



In total, 2,386 requests were made during the 2-minute load test

Response time < 200ms:
Average response time 145.83ms, which is a solid performance under typical load conditions.
95% of requests were completed in under 194.96ms, However, 49 requests (5%) exceeded the target of 200ms, indicating a slight bottleneck.

http_req_failed:
214 out of 2386 requests (8.96%) were marked as failed. These failures were primarily due to expected 404 Duplicate responses from the POST /bookings endpoint. We should not throw a 404 on a duplicate and therefore we should change the logic.

Average duration:
145.83ms.
95% of requests completed in under 194.96ms.

Stress testing for get Classes

```
export const options = {
  stages: [
    { duration: "1m", target: 100 },
    { duration: "2m", target: 100 },

    { duration: "1m", target: 200 },
    { duration: "2m", target: 200 },

    { duration: "1m", target: 500 },
    { duration: "2m", target: 500 },

    { duration: "30s", target: 0 },
  ],
  thresholds: {
    http_req_duration: ["p(95)<200"], // 95% of requests should complete within 200ms
    http_req_failed: ["rate<0.01"], // Less than 1% of requests should fail
  },
};

export default function () {
  const res = http.get("http://localhost:8080/classes");

  check(res, {
    "GET /classes - status is 200": (r) => r.status === 200,
    "GET /classes - classes returned": (r) => {
      const body = r.json();
      return Array.isArray(body) && body.length > 0;
    },
  });

  sleep(1); // Simulate user delay
}
```

```
    scenarios: (100.00%) 1 scenario, 500 max VUs, 10m0s max duration (incl. graceful stop):
             * default: Up to 500 looping VUs for 9m30s over 7 stages (gracefulRampDown: 30s, gracefulStop: 30s)


    ✓ GET /classes - status is 200
    ✓ GET /classes - classes returned

    checks.........................: 100.00% 88042 out of 88042
    data_received..................: 11 GB   20 MB/s
    data_sent......................: 3.8 MB  6.7 kB/s
    http_req_blocked...............: avg=15.54µs min=1µs      med=3µs    max=40.36ms  p(90)=8µs    p(95)=11µs
    http_req_connecting............: avg=4.05µs  min=0s       med=0s     max=13.27ms  p(90)=0s     p(95)=0s
  ✗ http_req_duration..............: avg=2.11s   min=107.1ms  med=1.25s max=10.09s   p(90)=4.44s p(95)=5.49s
      { expected_response:true }...: avg=2.11s   min=107.1ms  med=1.25s max=10.09s   p(90)=4.44s p(95)=5.49s
  ✓ http_req_failed................: 0.00%   0 out of 44021
    http_req_receiving.............: avg=141.3µs min=38µs     med=91µs   max=122.37ms p(90)=219µs p(95)=266µs
    http_req_sending...............: avg=22.75µs min=2µs      med=11µs   max=51.95ms  p(90)=24µs  p(95)=30µs
    http_req_tls_handshaking.......: avg=0s      min=0s       med=0s     max=0s       p(90)=0s     p(95)=0s
    http_req_waiting...............: avg=2.11s   min=106.99ms med=1.25s max=10.09s   p(90)=4.44s p(95)=5.49s
    http_reqs......................: 44021   77.059463/s
    iteration_duration.............: avg=3.11s   min=1.11s    med=2.26s max=11.09s   p(90)=5.44s p(95)=6.49s
    iterations.....................: 44021   77.059463/s
    vus............................: 1       min=1              max=500
    vus_max........................: 500     min=500            max=500


running (09m31.3s), 000/500 VUs, 44021 complete and 0 interrupted iterations
default ✓ [======================================] 000/500 VUs  9m30s
ERRO[0571] thresholds on metrics 'http_req_duration' have been crossed
```

Maximum of 500 Virtual Users (VUs) running over a total duration of 9 minutes and 30
Total 88,042 requests sent during the test.

Average duration: 2.11 seconds.

Max duration: 10.09 seconds.

95% of the requests completed in 5.49 seconds or less, while the remaining 5% of requests took longer than 5.49 seconds to complete.

The database is hosted on Azure using the cheapest tier option, which could have limited the performance during this stress test.

Spike testing for get Classes

```
export const options = {
  stages: [
    //warm up
    { duration: "30s", target: 100 },

    //spike
    { duration: "1m", target: 1_500 },
    { duration: "10s", target: 1_500 },
    { duration: "1m", target: 100 },

    //cool down
    { duration: "30s", target: 0 },
  ],
  thresholds: {
    http_req_duration: ["p(95)<200"], // 95% of requests should complete within 200ms
    http_req_failed: ["rate<0.01"], // Less than 1% of requests should fail
  },
};

export default function () {
  const res = http.get("http://localhost:8080/classes");

  check(res, {
    "GET /classes - status is 200": (r) => r.status === 200,
    "GET /classes - classes returned": (r) => {
      const body = r.json();
      return Array.isArray(body) && body.length > 0;
    },
  });

  sleep(1); // Simulate user delay
}
```

```
    ✓ GET /classes - status is 200
    ✗ GET /classes - classes returned
      ↳  85% — ✓ 13726 / ✗ 2360

    checks.........................: 92.66% 29812 out of 32172
    data_received..................: 3.5 GB 19 MB/s
    data_sent......................: 1.4 MB 7.3 kB/s
    http_req_blocked...............: avg=38.45µs min=1µs      med=4µs    max=7.26ms  p(90)=31µs    p(95)=259µs
    http_req_connecting............: avg=28.46µs min=0s       med=0s     max=7.2ms   p(90)=0s      p(95)=221µs
  ✗ http_req_duration..............: avg=6.44s   min=107.1ms  med=8.21s  max=10.82s  p(90)=10.15s  p(95)=10.21s
      { expected_response:true }...: avg=6.44s   min=107.1ms  med=8.21s  max=10.82s  p(90)=10.15s  p(95)=10.21s
  ✓ http_req_failed................: 0.00%  0 out of 16086
    http_req_receiving.............: avg=131.9µs min=10µs     med=94µs   max=19.81ms p(90)=226µs   p(95)=285µs
    http_req_sending...............: avg=21.11µs min=2µs      med=12µs   max=16.47ms p(90)=28µs    p(95)=39µs
    http_req_tls_handshaking.......: avg=0s      min=0s       med=0s     max=0s      p(90)=0s      p(95)=0s
    http_req_waiting...............: avg=6.44s   min=106.93ms med=8.21s  max=10.82s  p(90)=10.15s  p(95)=10.21s
    http_reqs......................: 16086  84.124134/s
    iteration_duration.............: avg=7.45s   min=1.11s    med=9.22s  max=11.83s  p(90)=11.15s  p(95)=11.21s
    iterations.....................: 16086  84.124134/s
    vus............................: 1      min=1              max=1500
    vus_max........................: 1500   min=1500           max=1500


running (3m11.2s), 0000/1500 VUs, 16086 complete and 0 interrupted iterations
default ✓ [======================================] 0000/1500 VUs  3m10s
ERRO[0194] thresholds on metrics 'http_req_duration' have been crossed
```

Total Requests: 16,086 requests were sent during the test.

85% success rate, with 13,726 successful responses and 2,360 failures.

Average Response Time: 6.44 seconds.

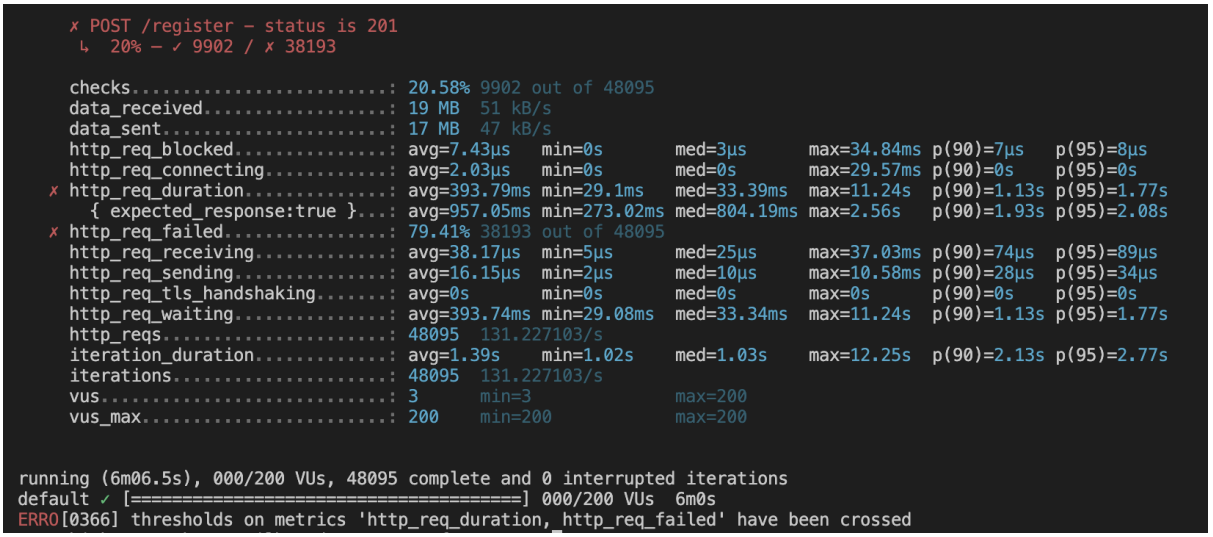
95th Percentile - 5% of requests took longer than 10.21 seconds.

Maximum Response Time: 10.82 seconds.

This test shows that the system struggled to handle the spike in requests efficiently, especially with high response times and failing checks.

Load - test for Register

```
export const options = {
  stages: [
    { duration: "30s", target: 200 }, // Ramp up to 200 users
    { duration: "5m", target: 200 }, // Sustain 200 users
    { duration: "30s", target: 0 }, // Ramp down
  ],
  thresholds: {
    http_req_duration: ["p(95)<200"], // 95% of requests should complete within 200ms
    http_req_failed: ["rate<0.01"], // Less than 1% of requests should fail
  },
};

export default function () {
  const payload = JSON.stringify({
    email: `user${Math.floor(Math.random() * 10000)}@example.com`,
    password: "password123",
    firstName: "Test",
    lastName: "User",
    phone: "123456789",
    address: "123 Test Street",
    dateOfBirth: "1990-01-01",
    membershipId: Math.ceil(Math.random() * 5), // Example MembershipID
    emergencyContact: "987654321",
  });

  const res = http.post("http://localhost:8080/register", payload, {
    headers: { "Content-Type": "application/json" },
  });

  check(res, {
    "POST /register - status is 201": (r) => r.status === 201,
  });

  sleep(1); // Simulate user delay
}
```



79.41% of requests failed (38193 out of 48095).

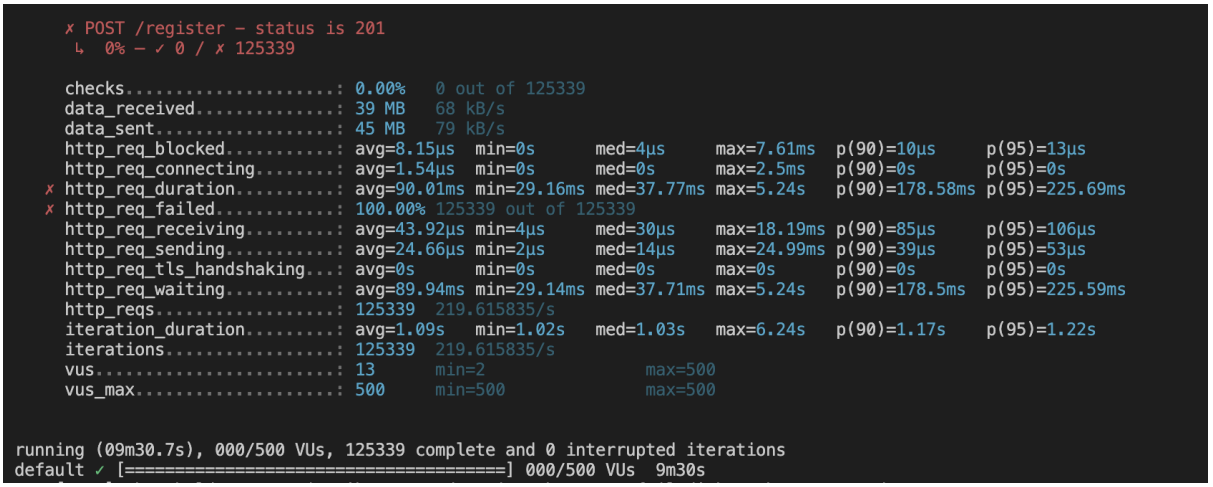
The average request duration is 393.79ms, with some requests taking as long as 11.24s.

High load on the database: The register operation involves creating both person and member records, which may strain your database.

In conclusion, the test results demonstrate that our application is currently unable to handle 500 users making simultaneous registration requests. The high failure rate (79.41%) and significant response time variability, with some requests taking over 11 seconds.

Stress - test for register

```
export const options = {
  stages: [
    { duration: "1m", target: 100 },
    { duration: "2m", target: 100 },

    { duration: "1m", target: 200 },
    { duration: "2m", target: 200 },

    { duration: "1m", target: 500 },
    { duration: "2m", target: 500 },

    { duration: "30s", target: 0 },
  ],
```

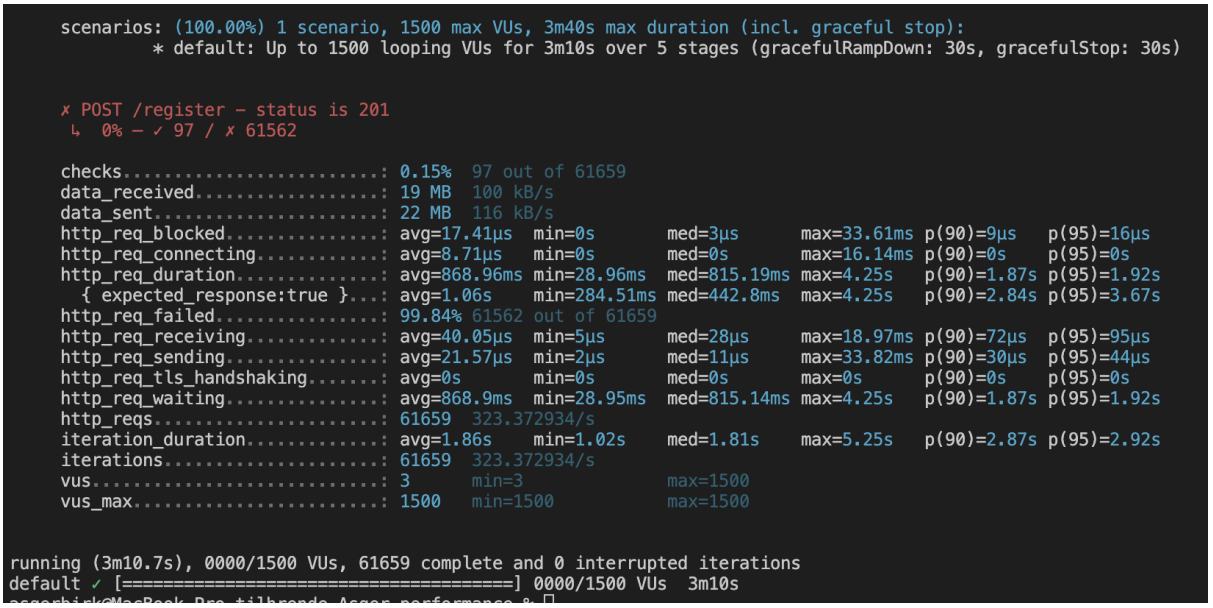


100% of requests failed: All 125,339 requests returned a failure

The database may be experiencing a bottleneck because of the volume of existing users.
Every new registration requires database operations such as:
Checking for unique constraints (e.g., email).
Writing to multiple tables (e.g., person and member).

Spike - test for register:

```
export const options = {
  stages: [
    //warm up
    { duration: "30s", target: 100 },

    //spike
    { duration: "1m", target: 1_500 },
    { duration: "10s", target: 1_500 },
    { duration: "1m", target: 100 },

    //cool down
    { duration: "30s", target: 0 },
  ],
```



99.84% of requests failed: Out of 61,659 total requests, only 97 succeeded, which indicates significant performance issues.

The average request duration is 868.96ms, but some requests took up to 4.25s.

The test clearly demonstrates that the application is unable to handle 1,500 virtual users (VUs) making simultaneous registration requests.