[Diterjemahkan dari bahasa Inggris ke bahasa Indonesia - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/id/?utm_source=onlinedoctranslator&utm_medium=pdf&utm_campaign=attribution)





Perbandingan kinerja

Antara ASP.NET Core dan Express.js untuk membuat Web API

BIDANG UTAMA:Ilmu Komputer

PENGARANG:Oliver Karlsson

PENGAWAS:Peter Larsson-Hijau

JÖNKÖPING:Juli 2021

Tesis akhir ini telah dilaksanakan di Fakultas Teknik Universitas Jönköping dalam bidang ilmu komputer. Penulis bertanggung jawab atas pendapat, kesimpulan, dan hasil yang disajikan.

Pemeriksa: Anders Adlemo

Handledare: Peter Larsson-Green

Lingkup: 15 hp (gelar sarjana) Tanggal: 25-07-2021

Alamat email:

Alamat Kunjungan: Telepon:

Kotak 1026

Gjuterigatan 5 036-10 10 00 (vx)

551 11 Jönköping

# Abstrak

Aplikasi web modern semakin kompleks dan semakin banyak digunakan. Menggunakan kerangka kerja untuk membangun API adalah cara yang populer bagi pengembang hobi dan bisnis untuk mempercepat waktu pengembangan dan menghemat biaya. Dengan ketergantungan pada kerangka kerja yang menjadi landasan bagi aplikasi yang berpotensi besar, timbul kebutuhan untuk memahami kualitas kinerjanya dan bidang mana yang paling cocok untuk kerangka kerja tersebut.

Studi ini membandingkan kinerja dua kerangka kerja populer serupa ASP.NET Core dan Express.js, ketika digunakan bersama dengan Database MySQL untuk membangun API Web. Hal ini dilakukan dengan membangun dua implementasi API yang berbeda di setiap kerangka kerja, satu menggunakan pendekatan RESTful dan yang lainnya menggunakan bahasa kueri baru GraphQL. Eksperimen dijalankan di mana penggunaan puncak CPU, penggunaan memori puncak, dan waktu respons diukur.

Hasil eksperimennya adalah dalam RESTful API, ASP.NET Core lebih cepat dalam melayani permintaan selama beban lebih rendah sedangkan Express.js mengungguli ASP.NET Core ketika dihadapkan dengan jumlah permintaan bersamaan yang lebih tinggi yang mengambil banyak data. Dalam API GraphQL, Express.js mampu memiliki kinerja yang sama atau lebih baik dalam semua kasus dalam hal waktu respons dan penggunaan sumber daya dibandingkan dengan ASP.NET Core.

# Kata kunci

API, Framework, REST, GraphQL, HTTP, SQL, ASP.NET Core, Express.js, Sistem Operasi, CPU, Memori, Waktu respons.

Isi

1. Perkenalan 5
   1. Latar Belakang . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   2. Deskripsi masalah . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
   3. Keterbatasan. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
   4. Maksud & Tujuan Penelitian . . . . . . . . . . . . . . . . . . . . . . . 7
2. Kerangka Teoritis 7
   1. Kaitan antara pertanyaan penelitian dan teori . . . . . . . . . . . . . . 7
   2. Transfer Perwakilan Negara. . . . . . . . . . . . . . . . . . . . . 7
      1. Enam Prinsip REST. . . . . . . . . . . . . . . . . . . 8
      2. Permintaan & Tanggapan . . . . . . . . . . . . . . . . . . . . . . . 8
   3. GrafikQL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
      1. Skema . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
      2. Kueri . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
      3. Alur eksekusi . . . . . . . . . . . . . . . . . . . . . . . . . . 10
   4. Inti ASP.NET . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
      1. Utas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
      2. Tugas & Async/Menunggu . . . . . . . . . . . . . . . . . . . . . . 11
      3. Kompiler C# . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
      4. Manajemen Memori. . . . . . . . . . . . . . . . . . . . . . . 12
   5. Ekspres.js. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
      1. Perulangan Peristiwa. . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
      2. Mesin JavaScript V8. . . . . . . . . . . . . . . . . . . . . . . 15
      3. Manajemen Memori. . . . . . . . . . . . . . . . . . . . . . . 15
3. Metode dan Implementasi 16
   1. Proses Kerja. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
   2. Pengumpulan Data. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
   3. Desain Tes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
   4. Basis Data. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
      1. Skema . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
      2. Data. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   5. Implementasi. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
      1. Spesifikasi API. . . . . . . . . . . . . . . . . . . . . . . . . 20
   6. Inti ASP.NET . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
      1. GrafikQL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
   7. Ekspres.js. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
      1. GrafikQL. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
   8. Lingkungan. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
      1. Klien . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
      2. Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
   9. Validitas & Reliabilitas. . . . . . . . . . . . . . . . . . . . . . . . . . . 23
4. Data Empiris 24
   1. Istirahat. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
   2. GrafikQL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
5. Analisis 30
   1. Pertanyaan Penelitian 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
      1. Penggunaan CPU . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
      2. Penggunaan Memori . . . . . . . . . . . . . . . . . . . . . . . . . . 31
      3. Waktu respons . . . . . . . . . . . . . . . . . . . . . . . . . . 32
   2. Pertanyaan Penelitian 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
      1. Penggunaan CPU . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
      2. Penggunaan Memori . . . . . . . . . . . . . . . . . . . . . . . . . . 34
      3. Waktu respons . . . . . . . . . . . . . . . . . . . . . . . . . . 35
6. Kesimpulan & Pembahasan 36
   1. Temuan. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
   2. Implikasi. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
   3. Keterbatasan. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
   4. Kesimpulan . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
   5. Pekerjaan di Masa Depan. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
7. Referensi 39

A Lampiran 42

1. SQL. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Perintah Atas & 42
2. Grep. . . . . . . . . . . . . . . . . . . . . . . . . Aplikasi 43
3. Faker.js. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Repositori 43
4. Aplikasi. . . . . . . . . . . . . . . . . . . . . . . . . 44

# 1. Perkenalan

1.1 Latar Belakang

Penggunaan Antarmuka Pemrograman Aplikasi (API) saat membuat aplikasi seluler dan web merupakan cara yang efektif bagi suatu aplikasi untuk memiliki akses jarak jauh ke data atau perangkat lain, namun karena itu diperlukan kinerja yang lebih besar dibandingkan sebelumnya. API umumnya ditemukan di perangkat IoT (Internet Of Things) di mana bendabenda seperti barang-barang rumah tangga atau kendaraan dapat bertukar informasi atau dikendalikan oleh pengguna melalui internet. Mereka juga digunakan untuk membuat Aplikasi Halaman Tunggal yang merupakan aplikasi yang memuat satu dokumen web dengan HTML, CSS, dan JavaScript yang diperlukan. Halaman statis ini kemudian mengambil konten yang harus ditampilkan melalui Web API.

Kerangka kerja dapat dipandang sebagai aplikasi “semi-lengkap” yang menyediakan serangkaian struktur dan fungsi spesifik[1] yang dapat dikembangkan dan digunakan oleh pengembang untuk membangun perangkat lunak yang lengkap. Kerangka kerja mendorong penggunaan kembali kode, memungkinkan implementasi fungsionalitas dengan cepat sehingga memangkas waktu dan biaya pengembangan[2]. Hal ini menjadikannya solusi yang sangat menarik bagi pengembang dan perusahaan yang ingin memulai dengan cepat.

Memilih kerangka kerja untuk proyek tertentu sering kali berarti harus mempertimbangkan beberapa faktor: platform mana yang dapat didukung dan perangkat apa yang akan digunakan oleh basis pengguna untuk mengaksesnya. Berapa ukuran dan kompleksitas proyek yang diharapkan, apakah proyek tersebut harus mudah diskalakan untuk mengakomodasi fluktuasi lalu lintas pengguna? Kita juga harus mempertimbangkan pengetahuan yang diperlukan untuk menggunakan kerangka kerja untuk tujuan yang dimaksudkan. Apakah tim pengembang sudah paham dengan bahasa dan gaya yang digunakan, apakah mereka memerlukan waktu untuk mempelajarinya atau haruskah seseorang yang memiliki pengetahuan tersebut dipekerjakan?

ASP.NET Core dan Express.js adalah dua kerangka Web sumber terbuka yang dapat digunakan untuk mengembangkan API. Express.js dipilih untuk penelitian ini karena popularitasnya yang tinggi[3], kesederhanaannya dan karena menggunakan JavaScript untuk mengembangkan perangkat lunak yang merupakan bahasa populer untuk pengembangan front-end. ASP.NET Core berada di bawah Express.js dalam popularitas[3] dan menurut survei pengembang Stack Overflow yang dilakukan pada bulan Februari 2020, ini adalah kerangka web yang paling disukai oleh pengembang profesional[4].

## 1.2 Deskripsi masalah

Jumlah aplikasi web yang mengandalkan API untuk data semakin meningkat. ProgrammableWeb adalah sumber informasi dan berita mengenai API, mereka juga menyediakan direktori API yang digunakan untuk membuat katalog dan mendokumentasikan penggunaan API. Pada tahun 2019 mereka melaporkan bahwa mereka menambahkan 168 API ke direktori mereka setiap bulan sehingga menghasilkan rata-rata lebih dari 2000 API baru setiap tahunnya[5]. Dengan semakin kompleksnya aplikasi dan jumlah pengguna internet yang terus meningkat, hal ini dapat mempersulit pengembang untuk memutuskan teknologi mana yang tepat digunakan untuk hasil proyek yang diinginkan.

Waktu respons dan waktu pemuatan di situs web dan aplikasi sangat memengaruhi pengalaman pengguna dan sebagian besar pengaruhnya adalah pada kinerja server. Pada tahun 2018, BBC melaporkan bahwa untuk setiap detik tambahan yang diperlukan untuk memuat sebuah halaman, sekitar 10% pengguna meninggalkannya[6]. COOK adalah perusahaan yang menjual makanan beku melalui websitenya. Mereka mengira waktu pemuatan yang lambat berdampak negatif pada bisnis mereka. Sebuah studi kasus telah dilakukan dan menunjukkan bahwa mengurangi waktu muat rata-rata sebesar 850 milidetik akan meningkatkan jumlah pengguna yang melakukan pembelian sebesar 7%, dan meningkatkan keterlibatan pelanggan sebesar 10%[7].

Erlandsson & Remes[8] menguji dan membandingkan kinerja REST, SOAP dan GraphQL pada API yang dibangun di .NET Core. Mereka menemukan bahwa dalam sebagian besar kasus pengujian, GraphQL berkinerja lebih buruk daripada REST dan SOAP terlepas dari jenis database yang digunakan, sementara REST memiliki kinerja terbaik secara keseluruhan. Kasus di mana GraphQL mengungguli REST dan SOAP adalah ketika 100 baris diminta dalam database MySQL, dan ketika hanya satu baris yang diminta dalam database MongoDB. Hal ini menunjukkan bahwa ada kemungkinan bahwa jika kerangka kerja tertentu memiliki kinerja lebih baik dalam kondisi tertentu, maka akan lebih efektif jika menggunakan arsitektur GraphQL.

Dengan meningkatnya aplikasi yang bergantung pada API dan semakin banyaknya orang yang menggunakan internet. Pentingnya memilih teknologi yang mampu memberikan kinerja yang diinginkan pengguna menjadi lebih besar. Studi ini akan fokus pada perbandingan kinerja ASP.NET Core dan Express.js menggunakan arsitektur REST (REpresentational State Transfer), dan bahasa kueri baru GraphQL. Alasan pemilihan arsitektur ini adalah karena sebagian besar Web API menggunakan REST atau arsitektur yang mendekatinya, sedangkan GraphQL masih sangat baru dan belum banyak digunakan[9]. GraphQL telah terbukti membutuhkan lebih sedikit usaha dibandingkan dengan REST ketika mengimplementasikan fungsi yang sama [10] dan mengetahui kinerjanya dalam kerangka kerja yang berbeda mungkin menjadi hal yang menarik bagi pengembang baru dan berpengalaman.

## 1.3 Keterbatasan

Studi ini hanya akan berfokus pada kinerja masing-masing framework dan perbandingannya satu sama lain. Untuk memastikan bahwa hasilnya tidak dipengaruhi oleh variabel yang terus berubah seperti kecepatan internet, konsistensi, dan keandalan, kedua API akan dihosting secara offline dan diakses melalui jaringan area lokal yang terhubung melalui ethernet dengan latensi kurang dari 1 ms. Basis datanya juga akan seperti itu

terletak di mesin yang sama yang menghosting API untuk menghilangkan latensi apa pun yang akan terjadi jika dihosting di mesin terpisah.

Kedua API tersebut hanya akan diimplementasikan bersama dengan database MySQL. MySQL secara konsisten menjadi database yang populer di kalangan pengembang[11-14] karena kemudahan penggunaannya, kinerja tinggi, dan skalabilitasnya[15]. Untuk mengurangi ruang lingkup penelitian, database hanya akan digunakan untuk mengambil data dan oleh karena itu kinerja saat menulis ke database tidak akan dievaluasi dalam penelitian ini.

Karena kinerjanya yang tinggi dan kemampuan penyesuaiannya, API akan dihosting di mesin Linux. Klien yang mengirimkan permintaan akan dijalankan pada mesin Windows, hal ini karena Windows adalah sistem operasi yang paling banyak digunakan untuk desktop dan laptop[16].

## 1.4 Maksud & Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk mengukur, mengevaluasi dan membandingkan kinerja dua kerangka aplikasi web ASP.NET Core dan Express.js ketika digunakan untuk mengembangkan API Web, untuk menentukan seberapa cepat mereka mampu melayani permintaan, dan seberapa besar kinerja komputer. sumber daya yang mereka perlukan. Hasil dari penelitian ini akan menarik bagi setiap pengembang yang ingin mengembangkan API di salah satu kerangka kerja ini bersama dengan desain arsitektur RESTful atau GraphQL dan database relasional.

Penelitian ini akan memberikan jawaban atas pertanyaan-pertanyaan berikut:

* Apa perbedaan antara ASP.NET Core Web API dan Express.js Web API dalam penggunaan CPU, penggunaan memori, dan waktu respons ketika diimplementasikan dengan arsitektur RESTful?
* Apa perbedaan antara ASP.NET Core Web API dan Express.js Web API dalam penggunaan CPU, penggunaan memori, dan waktu respons saat diimplementasikan dengan GraphQL?

# 2 Kerangka Teoritis

## 2.1 Kaitan antara pertanyaan penelitian dan teori

ASP.NET Core dan Express.js adalah dua kerangka kerja dengan popularitas serupa yang dapat digunakan untuk tujuan yang sama. Setiap kerangka kerja memiliki arsitekturnya sendiri dan cara melakukan hal-hal yang memengaruhi penggunaan CPU, penggunaan memori, dan seberapa cepat kerangka tersebut dapat memproses permintaan. Menjadikannya cocok untuk studi banding.

## 2.2 Transfer Perwakilan Negara

Desain arsitektur RESTful adalah gaya khusus untuk mengimplementasikan API. Diperkenalkan pada tahun 2000 oleh Roy Fielding, ini mendefinisikan serangkaian batasan khusus tentang bagaimana API harus berfungsi dan berperilaku. Batasan ini dimaksudkan untuk membantu meningkatkan keandalan, skalabilitas, dan kinerja aplikasi [27]. Karena popularitasnya dalam aplikasi berbasis HTTP, hal ini sering disalahartikan dengan HTTP, namun secara teori sebuah aplikasi bisa menjadi RESTful asalkan mengikuti 6 prinsip REST.

2.2.1 Enam Prinsip REST

Server klien:Arsitektur client-server berarti pemisahan perhatian antara dua bagian. Setiap sistem bertanggung jawab atas tugas yang berbeda, misalnya server bertanggung jawab atas manipulasi data sedangkan klien mengambil informasi dari server dan menampilkannya kepada pengguna.

Tanpa kewarganegaraan:Setiap permintaan yang dikirim ke server harus berisi semua informasi yang diperlukan untuk menyelesaikan permintaan tersebut. Semua permintaan yang dibuat ke server tidak bergantung satu sama lain dan server tidak boleh menyimpan informasi apa pun tentang sesi klien.

Dapat di-cache:Respons terhadap suatu permintaan harus menyatakan apakah permintaan tersebut boleh disimpan dalam cache atau tidak. Jika respons dapat di-cache maka klien dapat menyimpan respons untuk digunakan nanti.

Antarmuka Seragam:Antarmuka yang seragam antara semua komponen membuat arsitektur sistem secara keseluruhan jauh lebih sederhana dan mudah ditangani jika semua bagian mengikuti aturan komunikasi yang sama antara satu sama lain. REST memiliki empat batasan berbeda untuk mencapai hal ini. Yang pertama adalah setiap sumber daya harus memiliki pengidentifikasi unik. Kedua, sumber daya harus dimanipulasi melalui representasi. Prinsip ketiga menyatakan bahwa semua pesan harus bersifat deskriptif. Prinsip keempat adalah hypermedia harus menjadi mesin penerapan negara.

Sistem Berlapis:Membagi sistem menjadi beberapa lapisan dimana setiap lapisan tidak dapat melihat melampaui lapisan yang berinteraksi dengannya. Hal ini memungkinkan setiap bagian dari sistem dengan mudah diganti atau diperluas tanpa mempengaruhi keseluruhan sistem.

Kode sesuai permintaan:Prinsip keenam dan terakhir adalah opsional. Prinsip ini memungkinkan klien mengunduh fungsionalitas dalam bentuk skrip. Hal ini memungkinkan klien untuk memperluas fungsionalitas mereka sendiri tanpa perlu mengimplementasikannya sendiri. Namun ini opsional karena klien mungkin tidak dapat mengunduh atau menjalankan kode ini sehingga REST tidak dapat mengandalkannya.

2.2.2 Permintaan & Tanggapan

Seperti yang dinyatakan sebelumnya, salah satu kendala untuk antarmuka yang seragam adalah setiap sumber daya harus memiliki pengenal unik. Ini hadir dalam bentuk Uniform Resource Identifiers (URI), contoh yang terlihat pada Gambar 1 adalah Uniform Resource Locator (URL) yang merupakan salah satu jenis URI. itu berisi protokol yang digunakan untuk berkomunikasi dan jalur ke sumber daya tertentu, dalam hal ini adalah pengguna dengan ID 1. Respons terhadap permintaan HTTP GET ke URL ini dalam hal ini akan dalam format JSON, dan terlihat seperti contoh terlihat pada 2.

https://localhost/pengguna/1

Gambar 1: Contoh URL

|  |
| --- |
| {  "pengguna": {  "pengenal":1,  "nama":"John Doe",  "usia":30,  "alamat":"Contoh"  }  } |

Gambar 2: Respons HTTP GET yang Tenang

## 2.3 GrafikQL

Dikembangkan oleh Facebook dan dirilis sebagai sumber terbuka pada tahun 2015, GraphQL adalah bahasa kueri yang dirancang untuk API. Berbeda dengan API tradisional yang terdiri dari beberapa titik akhir untuk membantu memisahkan permintaan dan sumber daya, API yang dibangun dengan GraphQL hanya terdiri dari satu titik akhir yang memvalidasi dan mengeksekusi kueri masuk untuk mengambil sumber daya apa pun[17].

2.3.1 Skema

Komponen GraphQL yang paling umum dan penting adalah objek skema. GraphQL menggunakan sintaksnya sendiri untuk mendefinisikan skema ini dan disebut sebagai SDL (Schema Definition Language) atau IDL (Interface Definition Language). Definisi skema berisi dua bagian utama, tipe dan semua bidang yang terkait dengannya. Ada lima jenis skalar berbeda yang secara asli didukung oleh GraphQL yaitu Int, Float, String, Boolean, dan ID, dengan bidang non-nullable dilambangkan dengan ! mengikuti definisi skalar.

|  |
| --- |
| {  jenisPengguna{  pengenal:PENGENAL! nama:Rangkaian! usia:Int  alamat:Rangkaian  }  jenisPertanyaan{  pengguna(pengenal:PENGENAL!):Pengguna  }  } |

Gambar 3: Definisi skema tipe GraphQL

Gambar 3 di atas menunjukkan definisi dua tipe berbeda, tipe objek dan tipe kueri. Jenis objekPenggunadibuat untuk mewakili suatu objek yang ada di

database dan harus mendefinisikan semua bidang yang terkait dengan objek tersebut yang dapat dikembalikan. ItuPertanyaantype adalah apa yang mendefinisikan titik masuk yang valid ke server. Ini mendefinisikan nama kueri, semua argumen yang diperlukan untuk menemukan operasi dan jenis objek yang akan diresponnya. Kueri ini kemudian dicocokkan dengan fungsi penyelesai yang bertanggung jawab untuk melaksanakan tugas yang terkait dengan kuerinya masing-masing.

2.3.2 Kueri

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | {  } | pengguna(pengenal:"1"){ nama  usia  } | | |  |  | | --- | --- | | {  } | "pengguna": {  "nama":"John Doe",  "usia":30  } | |

Gambar 4: Permintaan & Respons Kueri GraphQL

Gambar 4 di atas menunjukkan contoh dasar permintaan dan respons kueri GraphQL. Kueri ini akan mengambilnamaDanusiadari pengguna dengan id 1. Perhatikan bahwa meskipun skema di atas pada Gambar 3 didefinisikan juga menyertakan bidang untuk alamat dan ID, skema tersebut tidak disertakan dalam kueri. Beginilah cara GraphQL menangani pengambilan yang berlebihan, alih-alih server mengembalikan semua data yang terkait dengan objek tersebut, hal ini memungkinkan klien untuk menentukan dalam kueri itu sendiri bidang mana yang diperlukan. Meskipun metode ini menghasilkan lebih banyak overhead untuk permintaan HTTP itu sendiri, metode ini juga terbukti sangat mengurangi ukuran respons jika REST harus mencapai beberapa titik akhir untuk mengambil jumlah data yang sama[18].

2.3.3 Alur eksekusi

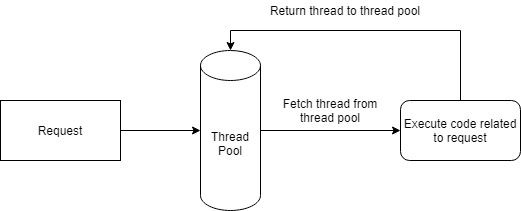
Kueri GraphQL melewati tiga fase berbeda setelah mencapai server. Pada fase pertama query diurai menjadi pohon sintaksis abstrak (AST). Fase kedua memvalidasi AST terhadap skema yang dibuat di mana ia memeriksa sintaksis yang benar dan apakah bidang yang disediakan telah ditentukan. Fase ketiga adalah eksekusi. Waktu proses akan melewati AST yang akan memanggil penyelesai yang benar, mengumpulkan semua hasil, dan mengembalikan JSON[19].

## 2.4 Inti ASP.NET

ASP.NET Core(Active Server Pages .NET Core) adalah kerangka web sumber terbuka untuk membuat aplikasi web dalam C# yang dirilis pada tahun 2016. Ini adalah desain ulang ASP.NET dan dikembangkan oleh Microsoft dan .NET Foundation. Aplikasi ASP.NET Core berjalan di Windows, Linux, macOS dan Docker, berbeda dengan ASP.NET yang hanya mendukung Windows[20]. Ini bisa sinkron dan asinkron dan menggunakan pendekatan multithread untuk menangani banyak permintaan secara bersamaan. Ini berarti bahwa setiap koneksi ke aplikasi oleh klien dialokasikan threadnya sendiri dengan jumlah RAM khusus.

2.4.1 Utas

Thread adalah konteks eksekusi suatu proses. Ketika proses aplikasi dimulai, Common Language Runtime (CLR) akan membuat satu thread latar depan untuk mengeksekusi kode aplikasi. Sebuah proses tunggal mungkin memiliki beberapa thread yang aktif di latar belakang untuk mengeksekusi kode yang terkait dengan proses tersebut[21]. Gambar 5 di bawah menunjukkan alur permintaan melalui aplikasi ASP.NET Core. Permintaan akan diberikan thread yang tersedia di kumpulan thread. Kode yang terkait dengan permintaan tersebut kemudian akan dieksekusi hingga menemukan operasi yang perlu ditunggu, di mana kelanjutan akan dijadwalkan dan thread akan dikembalikan ke kumpulan thread. Jika kumpulan thread tidak memiliki thread yang tersedia untuk menangani permintaan baru, thread baru dapat dibuat. Namun memunculkan thread baru tidak selalu merupakan solusi, sebuah thread memerlukan akses reguler ke sumber daya agar bisa maju. Situasi ketika thread tidak dapat memiliki akses rutin ke sumber daya bersama dan waktu CPU disebut sebagai kelaparan thread.



Gambar 5: Aliran kumpulan thread

2.4.2 Tugas & Async/Menunggu

Tugas adalah komponen kunci untuk mencapai pola asinkron berbasis tugas di . BERSIH[22]. Ini mewakili operasi yang tidak mengembalikan nilai, melainkan mengembalikan janji hasil di masa depan. Tugas dijadwalkan di kumpulan thread, tanpa kontrol atas thread mana yang harus dijalankan secara spesifik.

Panggilan I/O asinkron diimplementasikan melalui pola async/menunggu. Fungsi didefinisikan denganasinkronkata kunci yang memungkinkan penggunaan kata kunci tersebut menunggudi dalam fungsi. Gambar 6 di bawah menunjukkan contoh kode apamenunggu kata kunci tidak[23]. Pertama-tama ia memeriksa apakah hal yang ditunggunya sudah selesai. Jika dilakukan maka hanya mendapatkan hasil dari tindakan tersebut, jika tidak dilakukan maka akan dibuat lanjutan dari tindakan tersebut lalu segera kembali sehingga thread dapat dikembalikan ke thread pool dan melakukan pekerjaan lainnya. Setelah bagian yang ditunggu selesai, kelanjutannya dapat dijadwalkan di thread mana pun yang tersedia.

|  |
| --- |
| var$pelayan=sesuatu.GetAwaiter(); jika(! $waiter.IsCompleted) {  \_negara= 42;  AwaitOnCompleted(ref $awaiter); kembali; }  Label42:  $waiter.GetResult(); |

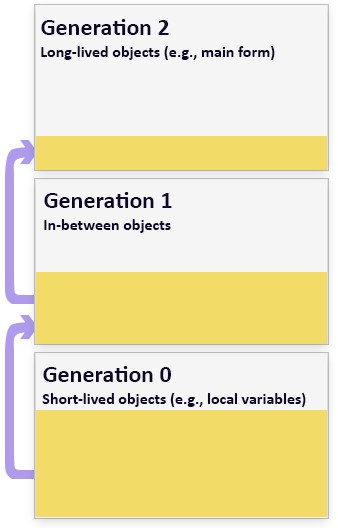
Gambar 6: menunggu prosedur

2.4.3 Kompiler C#

C# Menggunakan kompiler sumber terbukanya sendiri bernama Roslyn[24]. Ia bekerja dengan mengubah semua kode C# menjadi bahasa perantara (IL) yang kemudian disimpan di dalam rakitan (baik DLL atau EXE). IL kemudian dieksekusi oleh mesin virtual .NET Core CoreCLR. Hal ini dilakukan dengan memuat IL selama runtime yang kemudian dikompilasi menjadi instruksi CPU asli oleh kompiler just-in-time (JIT) [25]. Pada .NET Core 3.0, kompiler JIT menampilkan kompilasi berjenjang [23]. Ini adalah kompromi untuk mencapai waktu start-up yang lebih cepat daripada menghabiskan terlalu banyak waktu untuk mengoptimalkan kompilasi kode. Metode pertama-tama dikompilasi dengan fast-pass yang menerapkan sedikit optimasi pada kode. Namun seiring dengan dijalankannya metode ini berulang kali, semakin banyak optimasi yang dilakukan terhadap kualitas kode.

2.4.4 Manajemen Memori

Manajemen memori di ASP.NET Core bekerja dengan menempatkan objek ke dalam salah satu dari 3 generasi berbeda: 0, 1 atau 2 (lihat Gambar 7). Benda-benda yang ditempatkan pada generasi yang lebih rendah akan lebih sering diusahakan untuk dibebaskan dibandingkan dengan benda-benda yang ditempatkan pada generasi yang lebih tinggi. Masa hidup objek akan menentukan di generasi mana mereka ditempatkan. Objek yang dibuat selama permintaan web berumur pendek dan sering kali tetap berada di generasi yang lebih rendah, sementara objek seperti singleton tingkat aplikasi hidup selama durasi aplikasi sehingga pada akhirnya akan berakhir. pada generasi 2[26].



Gambar 7: Generasi Memori .NET

Saat objek dibuat, objek tersebut dialokasikan ke generasi 0. Ketika generasi 0 penuh, Pengumpul Sampah akan berjalan dan membuang semua objek yang tidak dapat dijangkau dari tumpukan. Semua objek yang dapat dijangkau kemudian dipromosikan ke generasi 1. Proses ini diulangi untuk setiap generasi ketika sudah penuh. Ketika proses pengumpulan sampah terjadi, proses ini akan dimulai pada generasi penuh terbaru dan kembali ke generasi 0[27].

## 2.5 Ekspres.js

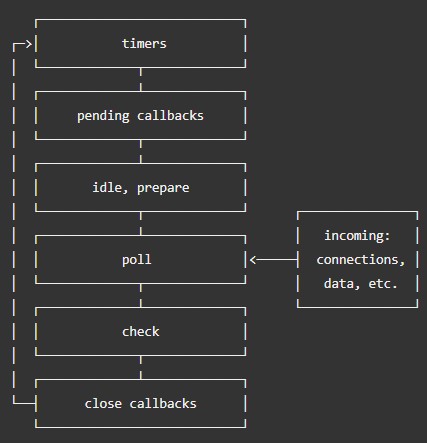
Express.js adalah kerangka aplikasi web sumber terbuka untuk Node.js. Ini dirilis pada tahun 2010 sebagai cara untuk mempermudah dan mempercepat pengembangan aplikasi web dan API di Node.js[28]. Express.js berjalan pada mesin JavaScript V8 Google dan mendeskripsikan dirinya sebagai "kerangka web yang cepat, tidak beropini, dan minimalis". Node.js adalah lingkungan eksekusi berulir tunggal yang digerakkan oleh peristiwa untuk JavaScript. Berbagai penelitian yang mengevaluasi kegunaan Express.js dan Node.js telah dilakukan. Baik Greiff & Johansson[29] dan Lund[30] menyimpulkan bahwa alasan utama popularitasnya adalah karena memungkinkan penggunaan JavaScript untuk pekerjaan front-end dan back-end.

2.5.1 Perulangan Peristiwa

Cara Node.js mengimplementasikan konkurensi adalah melalui event loop yang disediakan oleh perpustakaan libuv[31] yang ditulis dalam C. libuv juga menyediakan dukungan untuk I/ O asinkron dengan memanfaatkan kernel Sistem Operasi, libuv mengelola kumpulan threadnya sendiri yang digunakan untuk operasi yang terlalu berat untuk dijalankan di thread loop peristiwa seperti operasi sistem file atau fungsi DNS[32].

Perulangan peristiwa adalah bagian berulir tunggal dari Node.js dan merupakan utas tempat semua kode tertulis pengguna dieksekusi. Artinya setiap permintaan yang masuk ke aplikasi akan diproses satu per satu. Ketika menemukan operasi yang berpotensi memblokir, ia akan memindahkannya ke kernel sistem jika memungkinkan. Ketika kernel telah selesai memproses, ia akan memberi tahu aplikasi Node.js yang kemudian akan mengantri fungsi panggilan balik (fungsi yang diteruskan ke fungsi lain sebagai parameter) ke dalam antrean untuk dieksekusi.

Perulangan peristiwa memiliki enam tahapan berbeda yang harus dilewati sebelum dapat memulai iterasi baru atau “centang” [33]. Gambar 8 di bawah menunjukkan enam tahapan perulangan yang berbeda, perhatikan bahwamenganggur, bersiaplahfase ini hanya digunakan secara internal dan tidak mungkin mempunyai pengaruh langsung terhadap durasi atau pelaksanaan fase ini.



Gambar 8: Perulangan Peristiwa Node.js

Pengatur waktu:Apa pun yang dijadwalkan olehsetWaktu habis()DansetInterval()diatur untuk dieksekusi di sini. Acara yang dijadwalkan akan dipanggil sedini mungkin namun dapat tertunda karena acara tertentu seperti penjadwalan OS atau callback lainnya.

Panggilan balik yang tertunda:Fase ini bertanggung jawab untuk mengeksekusi callback yang terdaftar dengan operasi sistem seperti mengakses file, atau operasi jaringan.

Pemilihan:Ini adalah fase di mana permintaan diterima dan semua JavaScript yang ditulis pengguna akan dieksekusi, hingga menemukan sesuatu yang dapat didelegasikan

ke kernel sistem atau kumpulan thread libuv. Fase jajak pendapat bertanggung jawab atas dua hal. Pertama, ia menghitung berapa lama ia harus memblokir dan melakukan polling untuk operasi I/O, lalu mengeksekusi callback apa pun yang ada dalam antrean polling. Callback akan terus dijalankan hingga antrean jajak pendapat habis atau batas keras yang bergantung pada sistem tercapai.

Memeriksa:Jika ada callback yang dijadwalkan dengan setImmediate(), callback tersebut akan dieksekusi di sini. Timer ini digunakan ketika pengguna ingin menjalankan panggilan balik segera setelah fase polling berakhir. Jika fase polling menjadi tidak aktif dan skrip telah dimasukkan ke dalam antrean dengan setImmediate(), loop peristiwa akan melanjutkan ke fase pemeriksaan alihalih menunggu.

Tutup panggilan balik:Di sinilah semua panggilan balik untuk kejadian dekat akan dieksekusi, seperti penutupan soket web. Di sinilah event loop dibersihkan dan bersiap untuk tick berikutnya.

2.5.2 mesin JavaScript V8

Mesin JavaScript V8 ditulis dalam C++ dan awalnya dibuat untuk browser web Chrome, tetapi juga digunakan oleh Node.js untuk menjalankan JavaScript sisi server. V8 akan mengurai kode menjadi pohon sintaksis abstrak (AST). AST kemudian akan dikompilasi menjadi bytecode oleh penerjemahnya Ignition[35] dan kemudian dieksekusi satu kali. Eksekusi dan optimasi kode lebih lanjut dilakukan oleh kompiler just-in-time TurboFan[36]

2.5.3 Manajemen Memori

V8 juga bertanggung jawab untuk mengelola memori di Node.js. Hal ini dilakukan dengan membagi tumpukannya menjadi dua generasi berbeda: ruang baru dan ruang lama. Ruang baru berisi objek-objek yang baru saja dialokasikan, sedangkan ruang lama berisi objek-objek yang bertahan dalam beberapa siklus pengumpulan sampah[37].

V8 menggunakan metode optimasi yang disebutBerpura-pura.Objek yang telah dimiliki sebelumnya berarti objek tersebut dialokasikan langsung di ruang lama. Melalui tag khusus yang disebutkenang-kenanganyang dialokasikan bersama dengan objek di heap, mereka mengkodekan representasi tekstual dari kode yang membuat objek dan mencatat informasi ke dalam peta khusus. Data yang dikodekan kemudian digunakan untuk optimasi kompiler di masa depan dan memprediksi masa hidup objek [37].

Panggilan Balik Asinkron

Callback dalam JavaScript adalah fungsi yang diteruskan ke fungsi lain sebagai argumen[38] dan biasanya digunakan untuk mencapai asinkroni. Gambar 9 di bawah menunjukkan contoh bagaimana callback dilewatkan, dan kemudian dipanggil untuk mengembalikan suatu hasil.foo()akan memanggil fungsi

batang()dan meneruskan fungsi yang mengambil parameter lain yang disebut hasil. Sekalibatang()telah

menyelesaikan pekerjaan yang diperlukan untuk menghasilkan suatu hasil, ia hanya akan memanggil panggilan balik seperti fungsi normal lainnya, meneruskan hasilnya kembali kefoo()yang akan memproses hasil yang dikembalikan.

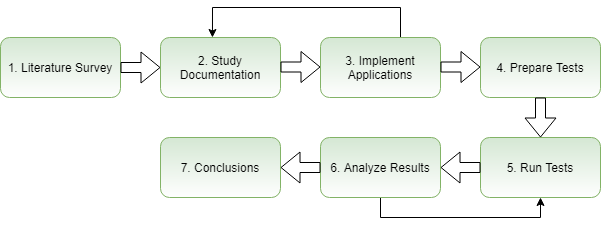
|  |
| --- |
| fungsifoo(){ batang(fungsi(hasil) {  peringatan (hasil)  }) }  fungsibar(panggilan balik){ varhasil= 1  panggilan balik (hasil) } |

Gambar 9: Contoh penggunaan panggilan balik

# 3 Metode dan Implementasi

Penelitian ini berfokus pada perbandingan kuantitatif antara ASP.NET Core dan Express.js. Pengumpulan data dilakukan dengan menjalankan eksperimen yang dirancang untuk mengambil data, sehingga penelitian ini tidak akan mengevaluasi kinerja saat menulis data ke database. Alasan memilih eksperimen sebagai metode untuk mengevaluasi teknologi ini adalah karena penelitian ini berfokus pada pengukuran kinerja, dan eksperimen dapat memverifikasi apakah suatu sistem dapat menyelesaikan tugas tertentu dengan penggunaan sumber daya yang wajar [39]. Untuk klien, metrik yang relevan adalah waktu respons individual untuk setiap permintaan, dan untuk server, metrik tersebut adalah penggunaan memori dan CPU aplikasi saat memproses permintaan. Setiap kasus uji akan bervariasi dalam berapa banyak permintaan paralel yang dikirim, berapa banyak data yang diminta serta kompleksitas query ke database.

## 3.1 Proses Kerja



Gambar 10: Proses Kerja

Gambar 10 di atas menunjukkan alur kerja selama penelitian. Survei literatur pertama kali dilakukan untuk mendapatkan informasi tentang penelitian sebelumnya yang dilakukan pada topik tersebut dan untuk mendapatkan pengetahuan lebih dalam tentang berbagai teknologi yang terlibat dalam penelitian ini. Seperti yang diharapkan, studi yang dilakukan pada kerangka kerja berbeda menggunakan arsitektur RESTful jauh lebih umum dibandingkan perbandingan dengan metode pengambilan data deklaratif seperti GraphQL.

Implementasi aplikasi dilakukan secara iteratif bersamaan dengan mempelajari dokumentasi resmi kedua framework tersebut untuk mempelajari bagaimana keduanya harus digunakan. Setelah tes dan data tes disiapkan maka dilakukan eksperimen sambil mengumpulkan datadata yang diperlukan untuk menjawab pertanyaan penelitian. Pengujian dijalankan masingmasing sebanyak 10 iterasi dan aplikasi dimulai ulang di antara setiap kasus pengujian. Keputusan untuk melakukan 10 iterasi karena penelitian serupa [29] dilakukan dengan hanya 5 iterasi dan menemukan bahwa fluktuasi metrik antar iterasi sangat kecil dan cukup untuk memberikan data yang akurat. Kasus uji untuk REST diselesaikan dan dianalisis terlebih dahulu sebelum melanjutkan ke GraphQL.

3.2 Pengumpulan Data

pelayan

Karena kedua aplikasi berjalan dalam satu proses, keduanya dapat dengan mudah dipantau oleh perintah teratas asli Linux yang menampilkan informasi tentang proses aktif. Menggabungkan top dengan perintah grep membuatnya top akan berjalan dalam interval dengan grep merekam output dari top ke file .txt yang diinginkan. Menyediakan grep dengan nama proses akan membuat hanya informasi dari proses tertentu yang disimpan. Tentu saja perintah top hanya akan dijalankan pada interval 3 detik, tetapi menjalankannya dengan flag -d memungkinkan interval diatur secara manual. Untuk pengujian ini intervalnya diatur ke 0,1 detik, menghasilkan 10 pembaruan per detik. Kedua perintah yang diperlukan dapat ditemukan di Lampiran B.

Klien

Apache JMeter v.5.4.1[40] digunakan untuk mengirim permintaan paralel dan mencatat waktu respons pada klien. Setiap pengujian diimplementasikan sebagai rencana pengujian terpisah yang menentukan titik akhir, metode http, dan jumlah permintaan paralel. Hasil dari setiap tes akan disimpan dalam format CSV untuk memudahkan penguraian. Tes dijalankan dengan perintahjmeter -n -t <nama\_tes>.jmx -l <file\_output>.jtl. -n flag menetapkan bahwa pengujian harus dijalankan dalam mode CLI (antarmuka baris perintah) untuk kinerja optimal dan inisialisasi thread yang tepat. Menjalankan pengujian dalam mode CLI disarankan karena jika tidak, JMeter mungkin tidak menginisialisasi semua thread dengan cukup cepat dan oleh karena itu beban yang diinginkan tidak akan tercapai.

## 3.3 Desain Tes

Kasus uji untuk REST ditentukan oleh tiga variabel: jumlah permintaan paralel, jumlah baris yang diminta dari database, dan jumlah gabungan antar tabel yang dilakukan. Permintaan paralel akan dibatasi hingga 100, 250, dan 500. Baris yang diminta adalah 1, 50, dan 100 sedangkan gabungan tabel adalah 0 dan 1. Artinya REST dan GraphQL masing-masing akan memiliki 18 kasus uji, dengan total 36 kasus uji. Tabel lengkap seluruh kombinasi dapat dilihat pada tabel di bawah ini.

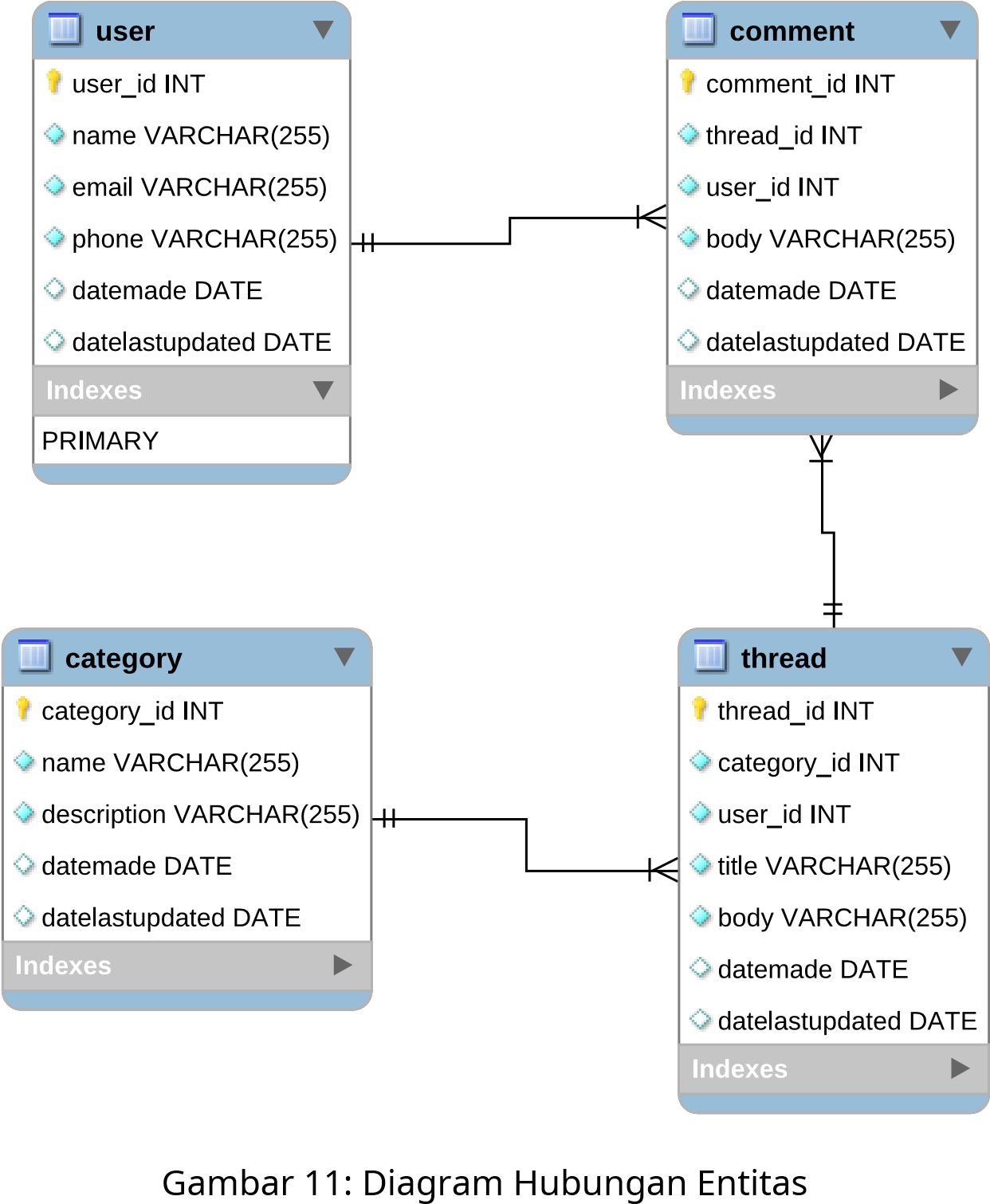
|  |  |  |  |
| --- | --- | --- | --- |
| Kasus cobaan | Permintaan | Baris | Bergabung |
| 1 | 100 | 1 | 0 |
| 2 | 100 | 50 | 0 |
| 3 | 100 | 100 | 0 |
| 4 | 100 | 1 | 1 |
| 5 | 100 | 50 | 1 |
| 6 | 100 | 100 | 1 |
| 7 | 250 | 1 | 0 |
| 8 | 250 | 50 | 0 |
| 9 | 250 | 100 | 0 |
| 10 | 250 | 1 | 1 |
| 11 | 250 | 50 | 1 |
| 12 | 250 | 100 | 1 |
| 13 | 500 | 1 | 0 |
| 14 | 500 | 50 | 0 |
| 15 | 500 | 100 | 0 |
| 16 | 500 | 1 | 1 |
| 17 | 500 | 50 | 1 |
| 18 | 500 | 100 | 1 |

Ketiga variabel ini menjadikan kasus pengujian adil bagi kedua kerangka kerja karena keduanya menangkap hasil penskalaan dengan menambahkan lebih banyak permintaan dan dengan mengubah jumlah data yang diambil oleh setiap permintaan. Domain yang ditetapkan untuk variabel juga mencerminkan beban yang lebih rendah, sedang, dan sangat tinggi. 500 permintaan per detik adalah sekitar 43 juta permintaan per hari. Dibandingkan dengan Wikipedia bahasa Inggris, 29 juta perangkat unik yang mengakses situs web setiap hari pada bulan Januari 2021[41] berarti server web yang menopang lalu lintas dalam jumlah besar.

## 3.4 Basis Data

3.4.1 Skema

Skema database menyerupai forum sederhana yang berisi empat tabel berbeda. Tabel ini menyimpan informasi tentang pengguna, thread, komentar, dan kategori thread. Skema ini memberikan hubungan realistis antara tabel dan memungkinkan penggunaan kueri sederhana dan lanjutan saat mengambil data. Pengaruh desain skema terhadap kinerja tidak akan relevan untuk penelitian ini. Hal ini karena kedua aplikasi menggunakan skema yang sama dan data yang sama. Kueri lengkap untuk menghasilkan tabel dapat ditemukan di Lampiran A dan diagram skema ER dapat dilihat di bawah pada Gambar 11.



3.4.2 Data

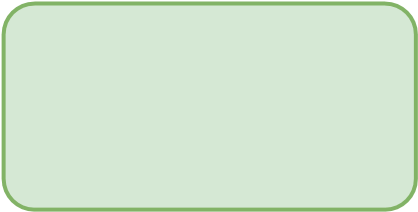
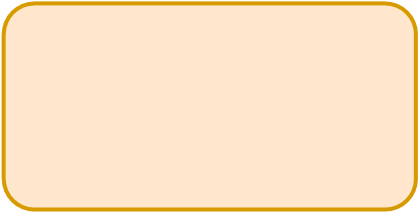
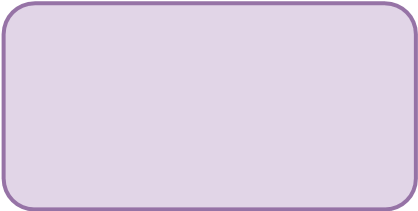
Aplikasi Node.js kecil dikembangkan untuk mengisi database dengan data dalam jumlah besar. Ia menggunakan paket NPM faker[42] untuk menghasilkan data tiruan yang kemudian dimasukkan ke dalam database. Faker memungkinkan beberapa jenis data dihasilkan dari benih tertentu sehingga sangat mudah untuk selalu menghasilkan data yang sama. Aplikasi ini menghasilkan 1000 pengguna dan kategori serta utas. Tabel yang digunakan untuk percobaan adalah tabel komentar, yang berisi 200 komentar untuk setiap pengguna sehingga menghasilkan total 200.000 baris untuk meniru database yang cukup besar. Semua kode yang relevan untuk aplikasi dapat dilihat pada Lampiran C.

## 3.5 Implementasi

Bagian ini akan membahas secara singkat rincian implementasi setiap kerangka kerja. Kedua kerangka kerja tersebut diimplementasikan dengan jumlah dependensi minimum yang diperlukan dan akan dibangun dengan versi Dukungan Jangka Panjang (LTS) yang stabil. Alasannya adalah untuk mengurangi jumlah variabel yang dapat memengaruhi hasil seperti pustaka pihak ketiga atau perubahan eksperimental pada versi yang lebih baru. Satu-satunya optimasi yang dilakukan pada setiap kerangka kerja adalah keduanya diimplementasikan dengan panggilan I/O asinkron dan menggunakan pengumpulan koneksi MySQL. Hal ini untuk mencegah kemungkinan database kehabisan koneksi yang tersedia dan mempengaruhi hasil percobaan. Kumpulan koneksi menggunakan jumlah default

kolam yang jumlahnya 100.

REST dan GraphQL akan diimplementasikan dalam aplikasi yang sama melalui titik akhir yang berbeda untuk kedua kerangka kerja. Hal ini kecil kemungkinannya akan berdampak pada kinerja karena aplikasi itu sendiri mengekspos sejumlah kecil endpoint sedangkan contoh di dunia nyata mengekspos jumlah endpoint yang jauh lebih besar. Titik akhir RESTful dilakukan sepenuhnya dengan cara asli dan tidak akan dibahas di bagian ini. Kode sumber aplikasi dapat dilihat pada repositorinya yang terdapat pada Lampiran D. Gambar 12 di bawah menunjukkan diagram aliran data aplikasi yang disederhanakan.



Basis

data

Pengendali

Basis

Data

Pengontrol/

Router

Klien

Gambar

12:

Aliran

Data

Aplikasi

3.5.1 Spesifikasi API

Tabel utama yang akan ditanyakan untuk semua kasus uji adalahkomentarmeja. Setiap RESTful API akan mengekspos empat titik akhir berikut:

* /komentar/:id
* /komentar/:batas
* /pengguna/:id/komentar
* /pengguna/:id/komentar/:batas

Kueri yang perlu ditentukan untuk GraphQL API adalah sebagai berikut:

* commentById(id: ID!): komentar
* commentsByLimit(batas: Int!): [komentar]
* commentByUserId(id: ID!): komentar
* commentsByUserId(id: ID!, batas: Int!): [komentar]

Pengambilan satu baris dari tabel akan dilakukan dengan menentukan ID untuk komentar. Pengambilan beberapa baris akan dilakukan dengan memberikan batasan berapa banyak komentar yang harus disertakan dalam kumpulan hasil kueri. Melakukan kueri di mana gabungan akan digunakan, thekomentar tabel akan digabungkan denganpenggunatabel pada ID pengguna tertentu. Pengambilan satu baris di sini kemudian dilakukan dengan meneruskan ID pengguna tertentu sebagai argumen, yang kemudian akan mengambil komentar pertama yang dibuat oleh pengguna tersebut. Beberapa baris diambil melalui ID pengguna dan batas yang kemudian akan mengambil daftar komentar yang dibuat oleh pengguna.

## 3.6 Inti ASP.NET

Aplikasi berjalan pada versi 3.1.17 LTS dan dibuat di .NET Core CLI dengan menjalankandotnet webapi barumemerintah. Perintah ini akan membuat semua file yang diperlukan. MySQL Connector/NET dikembangkan oleh MySQL sendiri dan digunakan untuk memungkinkan komunikasi dengan database. Itu diterapkan dalam produksi dengan menjalankan perintahdotnet terbitkan -konfigurasi Rilis dan kemudian memulai aplikasi melaluidotnet <nama proyek>.dll.

Untuk mencapai aliran asinkron penuh dalam aplikasi, fungsi akan ditentukan dengan asinkronkata kunci dan tipe pengembalian ditentukan sebagaiTugas<IActionResult>.

3.6.1 GrafikQL

Karena GraphQL belum asli ASP.NET Core, paket NuGet GraphQL[43] diinstal untuk penelitian ini. Aplikasi memerlukan tiga hal selain pengontrol HTTP agar dapat memproses permintaan GraphQL sepenuhnya. Hal pertama adalah aGrafikQLQuery objek yang merupakan model dasar untuk semua kueri yang diterima, berisi nama operasi, nama kueri, dan semua variabel. Semua objek kelas yang akan ditangani dalam titik akhir GraphQL perlu didefinisikan sebagai Tipe. Hal ini dilakukan dengan membuat kelas yang mewarisi dariTipe Grafik Objekkelas. Kelas ini akan memetakan semua bidang kelas dasar ke sebuahTipe Grafik Idagar dapat diurai. Hal terakhir yang diperlukan adalah skema kueri. Ini mendefinisikan semua kueri yang tersedia, tipe pengembaliannya, dan apa yang harus dilakukan setelah permintaan spesifik diterima.

## 3.7 Ekspres.js

Berjalan pada Express.js versi 4.17.1 dan Node.js versi 14.17.0. Aplikasi ini dibuat dengan cara dijalankannpm initdi baris perintah dan hanya akan membuat file package.json yang diperlukan untuk proyek tersebut. Karena Express.js tidak memiliki cara yang pasti dalam melakukan sesuatu, maka Express.js diterapkan dengan sangat minimal untuk menjadikannya perbandingan yang adil. Aplikasi ini terutama terdiri dari tiga hal, pengendali database yang menangani semua kueri SQL, router untuk titik akhir REST, dan router untuk titik akhir GraphQL. Express.js juga perlu diberitahukan secara eksplisit untuk dijalankan di Produksi yang dilakukan melalui variabel lingkungan. Jumlah maksimum memori yang diperbolehkan juga diatur agar aplikasi dapat menggunakan sebanyak mungkin memori yang tersedia yang dibutuhkannya.

* NODE ENV=produksi
* OPSI NODE=–ukuran ruang lama maksimal=8192

Panggilan I/O asinkron di Express.js dimungkinkan dengan menggunakan fungsi panggilan balik yang dipanggil setelah database selesai mengambil data.

3.7.1 GrafikQL

Agar GraphQL dapat diimplementasikan dalam aplikasi Express.js diperlukan dua paket npm. Paketgrafikql[44] memungkinkan pembuatan skema melalui objek buildSchema.Skema ini mendefinisikan semua tipe kueri yang akan diterima sebagai valid, serta tipe objek dan data yang akan berisi setiap kueri dan apa yang akan dikembalikan.

Paket express-graphql[45] menyediakangraphqlHTTPobjek, objek ini adalah pengendali rute dan akan menerima permintaan HTTP masuk apa pun yang dibuat ke titik akhir GraphQL yang ditentukan. Setelah permintaan diterima, permintaan tersebut diteruskan ke penyelesai yang sesuai. Setiap penyelesai didefinisikan di dalam objek akar yang akan mencocokkan kueri dengan penyelesai yang benar. Penyelesai ini kemudian mem-parsing data yang diperlukan seperti argumen apa pun yang disertakan dan meneruskan permintaan.

GraphQL di Node.js diperlukan untuk mengembalikan nilai langsung atau janji yang di masa depan akan diselesaikan menjadi suatu nilai. Oleh karena itu, semua penyelesai GraphQL ditentukan untuk mengembalikan aJanjiobyek. Gambar 13 menunjukkan pemecah masalah GraphQL yang kemudian mengembalikan janji bahwa hasil database pada akhirnya akan ditemukan.

|  |
| --- |
| komentar:({batas}) => { kembalikan Janji baru((putuskan, tolak) => { db.getComments(batas,fungsi(hasil){  menyelesaikan (hasil)  })  }); } |

Gambar 13: Janji JavaScript

## 3.8 Lingkungan

3.8.1 Klien

* OS: Windows 10 Rumah 64Bit
* Prosesor: CPU Intel(R) Core(TM) i7-8750 @ 2,2GHz
* RAM: 16GB

3.8.2 Server

Distribusi Linux yang dipilih untuk menghosting API adalah Ubuntu 20.04 Desktop. Hal ini karena popularitasnya di web hosting, dukungannya yang lama dan karena mendukung semua versi .NET Core dan Node.js. Server adalah instalasi Ubuntu yang benar-benar baru di mana semua data dan sistem operasi sebelumnya berada

DIHAPUS. Satu-satunya perangkat lunak yang diinstal adalah MySQL untuk meng-host database dan Visual Studio Code bersama dengan SDK dan runtime yang diperlukan untuk mengembangkan dan menjalankan aplikasi.

* OS: Desktop Ubuntu LTS 20.04
* CPU: CPU Intel(R) Core(TM) i5-4670K @ 3,4GHz
* RAM: 8GB 1333MHz
* Penyimpanan: SSD 120GB

## 3.9 Validitas & Reliabilitas

Kedua API akan menggunakan database relasional untuk menyimpan data, menggunakan data asli yang sama dan logika yang sama untuk mengakses data. Artinya, hasil yang diberikan pengujian hanya akan dipengaruhi oleh kinerja masing-masing framework.

Setiap pengujian akan dilakukan sebanyak 10 iterasi dan hasilnya kemudian digunakan untuk menghitung nilai rata-rata setiap kasus uji. Hal ini dilakukan untuk menyaring anomali apa pun yang mungkin terjadi seperti proses latar belakang yang menggunakan sumber daya dan untuk mengurangi signifikansi fluktuasi apa pun dalam nilai terukur.

# 4 Data Empiris

Bab ini akan menyajikan tabel yang berisi data mentah yang dikumpulkan dari menjalankan semua kasus uji. Waktu respons diukur dalam milidetik, sedangkan memori dan CPU diukur dalam persentase.

4.1 Istirahat

|  |  |  |
| --- | --- | --- |
| REST - Penggunaan CPU (%) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 111,28 | 66,35 |
| 2 | 265,81 | 90,54 |
| 3 | 310,18 | 101,36 |
| 4 | 96,59 | 61,89 |
| 5 | 266,53 | 107,00 |
| 6 | 327,55 | 108,91 |
| 7 | 145,64 | 109,01 |
| 8 | 314,72 | 125,64 |
| 9 | 346,00 | 137,00 |
| 10 | 133,29 | 108,46 |
| 11 | 313,00 | 126,92 |
| 12 | 337,90 | 137,37 |
| 13 | 205,47 | 150,54 |
| 14 | 333,90 | 147,48 |
| 15 | 348,80 | 147,00 |
| 16 | 194,99 | 140,65 |
| 17 | 327,00 | 144,55 |
| 18 | 350,89 | 142,55 |

Tabel 1: Penggunaan CPU REST

|  |  |  |
| --- | --- | --- |
| REST - Penggunaan Memori (%) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 1,38 | 0,84 |
| 2 | 1,53 | 0,88 |
| 3 | 1,56 | 0,97 |
| 4 | 1,46 | 0,78 |
| 5 | 1,54 | 0,87 |
| 6 | 1,54 | 0,88 |
| 7 | 2,00 | 0,89 |
| 8 | 1,63 | 1,04 |
| 9 | 1,73 | 1,16 |
| 10 | 2,09 | 0,87 |
| 11 | 1,64 | 1,12 |
| 12 | 1,73 | 1,16 |
| 13 | 2,82 | 1,04 |
| 14 | 1,52 | 1,19 |
| 15 | 1,75 | 1,32 |
| 16 | 2,69 | 1,17 |
| 17 | 1,66 | 1,18 |
| 18 | 1,77 | 1,34 |

Tabel 2: Penggunaan Memori REST

|  |  |  |
| --- | --- | --- |
| REST - Waktu Respons Rata-rata (ms) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 78,4 | 104,0 |
| 2 | 137,0 | 122,0 |
| 3 | 207,4 | 137,2 |
| 4 | 74,6 | 99,0 |
| 5 | 134,9 | 121,1 |
| 6 | 197,4 | 135,1 |
| 7 | 83,8 | 132,0 |
| 8 | 255,9 | 184,2 |
| 9 | 400,2 | 246,4 |
| 10 | 82,1 | 149,5 |
| 11 | 257,0 | 195,1 |
| 12 | 418,5 | 247,9 |
| 13 | 120,0 | 204,8 |
| 14 | 474,9 | 319,6 |
| 15 | 681,7 | 385,9 |
| 16 | 126,3 | 192,6 |
| 17 | 486,3 | 313,7 |
| 18 | 709,9 | 369,7 |

Tabel 3: Rata-rata Waktu Respon REST

|  |  |  |
| --- | --- | --- |
| REST - Waktu Respons Minimum (md) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 64,7 | 93,0 |
| 2 | 107,5 | 99,3 |
| 3 | 126,4 | 89,9 |
| 4 | 67,3 | 89,4 |
| 5 | 88,5 | 98,0 |
| 6 | 115,4 | 90,8 |
| 7 | 57,1 | 87,8 |
| 8 | 74,5 | 92,1 |
| 9 | 103,2 | 108,7 |
| 10 | 50,8 | 97,3 |
| 11 | 107,8 | 97,9 |
| 12 | 168,7 | 95,2 |
| 13 | 54,9 | 107,5 |
| 14 | 135,9 | 122,7 |
| 15 | 145,9 | 132,7 |
| 16 | 70,5 | 102,7 |
| 17 | 126,2 | 124,3 |
| 18 | 226,1 | 119,6 |

Tabel 4: Waktu Respons Minimum REST

|  |  |  |
| --- | --- | --- |
| REST - Waktu Respons Maksimum (ms) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 100,3 | 120,2 |
| 2 | 162,8 | 152,2 |
| 3 | 277,5 | 188,0 |
| 4 | 83,3 | 111,5 |
| 5 | 170,4 | 153,4 |
| 6 | 236,9 | 183,9 |
| 7 | 108,7 | 173,2 |
| 8 | 389,1 | 271,4 |
| 9 | 497,1 | 381,2 |
| 10 | 114,2 | 194,9 |
| 11 | 390,5 | 287,1 |
| 12 | 592,3 | 388,6 |
| 13 | 182,3 | 301,6 |
| 14 | 812,4 | 502,6 |
| 15 | 966,8 | 637,5 |
| 16 | 183,9 | 275,8 |
| 17 | 826,6 | 499,6 |
| 18 | 989,7 | 675,8 |

Tabel 5: Waktu Respons Maksimum REST

## 4.2 GrafikQL

|  |  |  |
| --- | --- | --- |
| GraphQL - Penggunaan CPU (%) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 263,85 | 99,04 |
| 2 | 345,28 | 124,74 |
| 3 | 368,91 | 152,45 |
| 4 | 229,73 | 105,00 |
| 5 | 337,10 | 132,64 |
| 6 | 355,73 | 144,64 |
| 7 | 323,01 | 142,01 |
| 8 | 370,00 | 163,36 |
| 9 | 389,00 | 163,00 |
| 10 | 349,91 | 147,45 |
| 11 | 365,00 | 161,53 |
| 12 | 379,00 | 161,45 |
| 13 | 350,55 | 168,45 |
| 14 | 383,00 | 155,55 |
| 15 | 382,00 | 163,90 |
| 16 | 354,00 | 164,55 |
| 17 | 375,00 | 163,90 |
| 18 | 382,00 | 166,36 |

Tabel 6: Penggunaan CPU GraphQL

|  |  |  |
| --- | --- | --- |
| GraphQL - Penggunaan Memori (%) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 2,50 | 0,88 |
| 2 | 2,97 | 0,97 |
| 3 | 3,17 | 1,14 |
| 4 | 2,53 | 0,87 |
| 5 | 2,94 | 0,97 |
| 6 | 3,11 | 1,16 |
| 7 | 3,56 | 0,95 |
| 8 | 3,86 | 1,31 |
| 9 | 4,51 | 1,39 |
| 10 | 3,08 | 0,98 |
| 11 | 3,93 | 1,29 |
| 12 | 4,36 | 1,41 |
| 13 | 4,34 | 1,23 |
| 14 | 4,85 | 1,53 |
| 15 | 5,36 | 1,62 |
| 16 | 4,64 | 1,29 |
| 17 | 4,53 | 1,55 |
| 18 | 5,23 | 1,60 |

Tabel 7: Penggunaan Memori GraphQL

|  |  |  |
| --- | --- | --- |
| GraphQL - Waktu Respons Rata-rata (ms) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 144,2 | 152,4 |
| 2 | 197,5 | 173,9 |
| 3 | 244,2 | 226,6 |
| 4 | 125,9 | 152,3 |
| 5 | 195,1 | 175,3 |
| 6 | 258,4 | 224,4 |
| 7 | 220,1 | 223,3 |
| 8 | 382,8 | 304,4 |
| 9 | 571,3 | 373,6 |
| 10 | 234,5 | 237,5 |
| 11 | 378,5 | 328,6 |
| 12 | 533,9 | 382,4 |
| 13 | 367,5 | 371,3 |
| 14 | 588,8 | 464,0 |
| 15 | 984,9 | 594,7 |
| 16 | 401,9 | 362,0 |
| 17 | 705,7 | 483,5 |
| 18 | 1003,5 | 584,6 |

Tabel 8: Rata-rata Waktu Respons GraphQL

|  |  |  |
| --- | --- | --- |
| GraphQL - Waktu Respons Minimum (md) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 110,2 | 145,9 |
| 2 | 129,6 | 137,6 |
| 3 | 168,1 | 162,2 |
| 4 | 87,5 | 143,7 |
| 5 | 115,7 | 137,9 |
| 6 | 135,0 | 159,6 |
| 7 | 97,8 | 170,3 |
| 8 | 130,9 | 165,9 |
| 9 | 153,2 | 174,6 |
| 10 | 109,0 | 183,8 |
| 11 | 130,7 | 188,1 |
| 12 | 191,8 | 177,0 |
| 13 | 89,3 | 210,4 |
| 14 | 165,8 | 188,9 |
| 15 | 177,3 | 161,5 |
| 16 | 144,6 | 193,6 |
| 17 | 202,1 | 185,9 |
| 18 | 253,6 | 173,4 |

Tabel 9: Waktu Respons GraphQL Minimum

|  |  |  |
| --- | --- | --- |
| GraphQL - Waktu Respons Maksimum (ms) | | |
| Kasus | Inti ASP.NET | Ekspres.js |
| 1 | 168,4 | 164,4 |
| 2 | 228,7 | 214,9 |
| 3 | 301,8 | 300,5 |
| 4 | 144,9 | 165,8 |
| 5 | 222,1 | 217,9 |
| 6 | 294,1 | 299,0 |
| 7 | 274,5 | 271,8 |
| 8 | 516,7 | 421,4 |
| 9 | 689,3 | 545,6 |
| 10 | 298,5 | 288,6 |
| 11 | 533,5 | 447,0 |
| 12 | 679,0 | 558,5 |
| 13 | 523,2 | 480,7 |
| 14 | 1002,5 | 696,8 |
| 15 | 1289,5 | 938,4 |
| 16 | 542,0 | 462,4 |
| 17 | 1019,4 | 715,9 |
| 18 | 1273,2 | 921,7 |

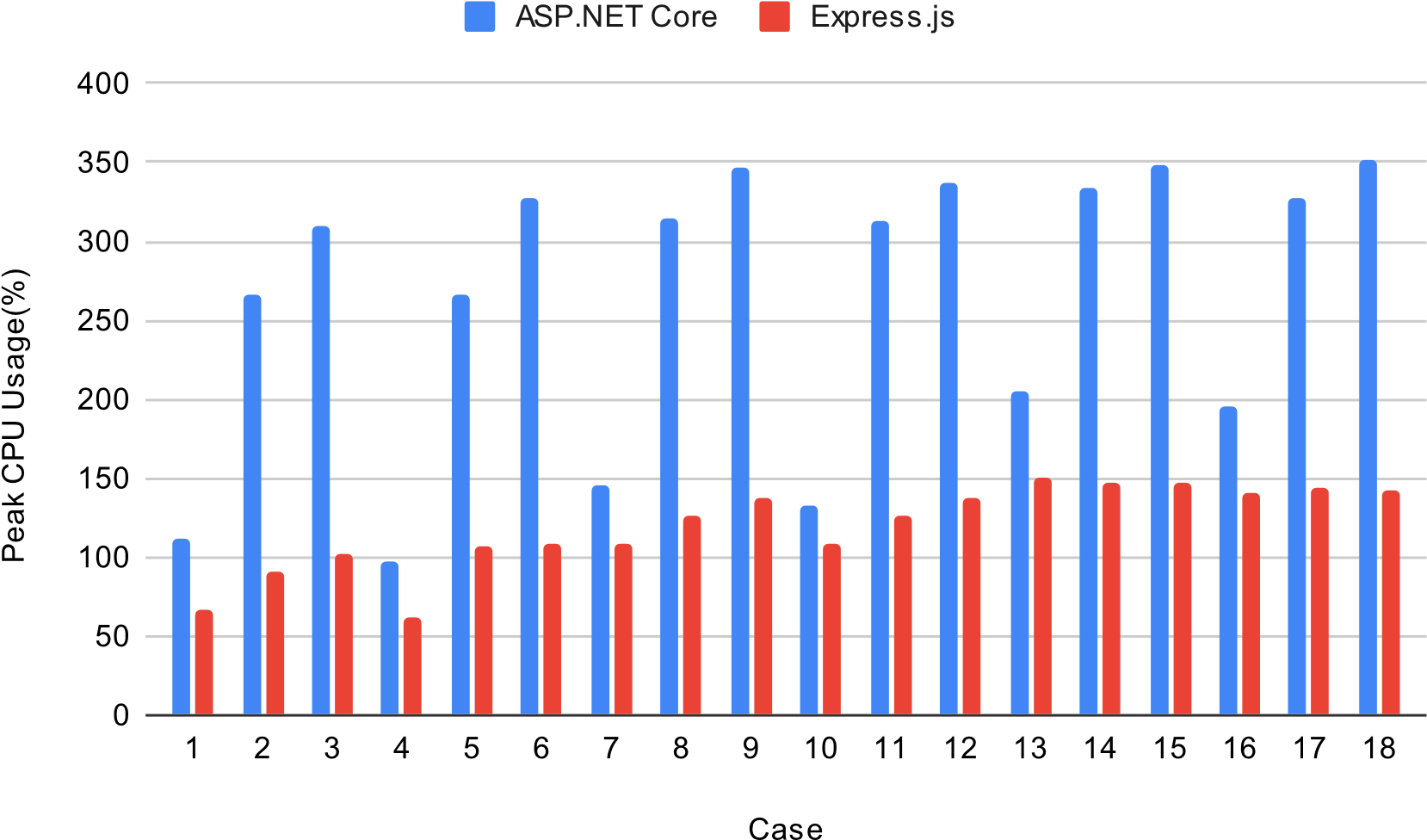
Tabel 10: Waktu Respons GraphQL Maksimum

# 5 Analisis

Bab ini akan membandingkan nilai rata-rata yang dihitung dari iterasi pengujian dan membandingkannya satu sama lain. Setiap pertanyaan penelitian dibagi menjadi tiga bagian, penggunaan CPU, penggunaan memori, dan terakhir waktu respons. Bagian CPU dan memori akan membandingkan nilai puncak rata-rata sedangkan waktu respons akan melihat waktu rata-rata keseluruhan dan waktu minimum & maksimum rata-rata. Karena CPU memiliki 4 core, maka nilai penggunaan CPU akan berkisar antara 0 hingga 400.

## 5.1 Pertanyaan Penelitian 1

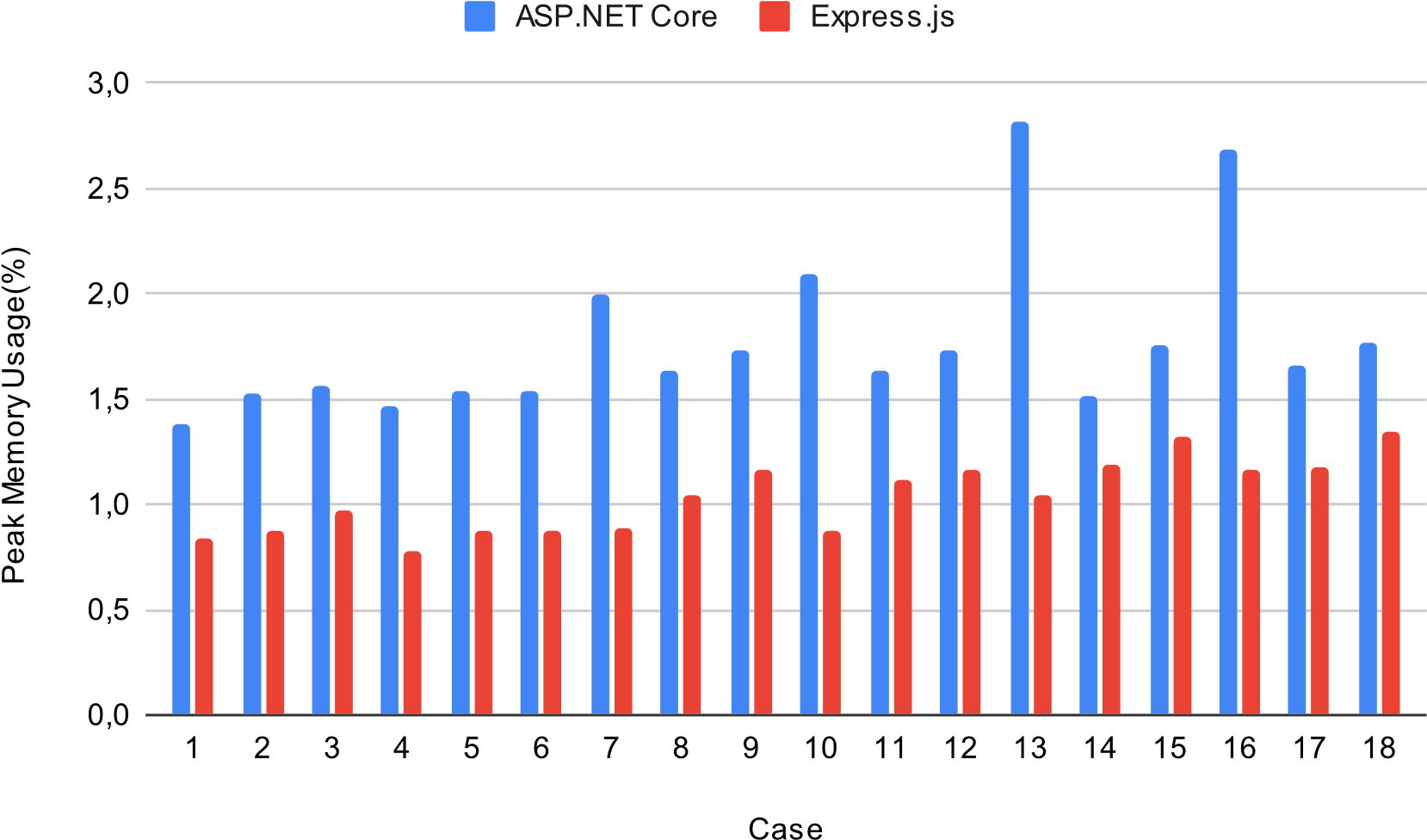
5.1.1 Penggunaan CPU



Gambar 14: REST Penggunaan CPU

Seperti terlihat pada gambar 14 di atas, penggunaan CPU sangat mirip untuk kedua kerangka kerja dalam kasus 1, 4, 10, 13, 16. Ini semua adalah kasus di mana hanya satu baris yang diminta dari database. Namun ASP.NET Core mengalami peningkatan penggunaan CPU yang jauh lebih besar karena jumlah baris yang diminta meningkat dibandingkan Express.js.

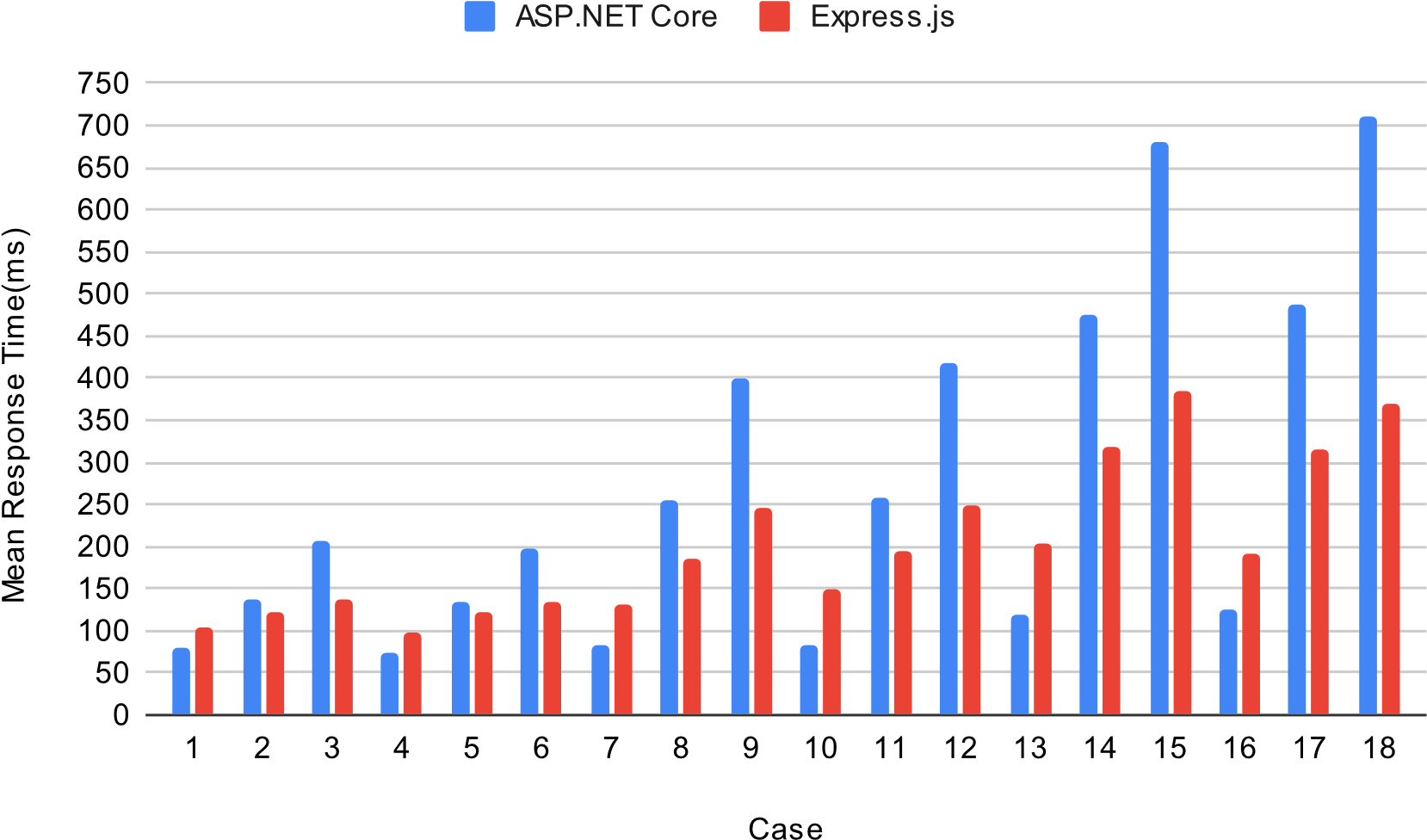
5.1.2 Penggunaan Memori



Gambar 15: Penggunaan Memori REST

Gambar 15 menunjukkan bahwa Express.js berperilaku serupa dalam kasus 1-6 dan menunjukkan peningkatan yang lebih besar untuk kasus pengujian lainnya. Ketika jumlah pengguna 250 dan 500, ASP.NET Core melihat pola yang menarik dalam penggunaan memorinya. Semua kasus di mana hanya satu baris yang diminta akan melihat % memori yang digunakan jauh lebih tinggi dibandingkan kasus berikut untuk 50 dan 100 baris dengan jumlah pengguna yang sama. Namun untuk kasus 1-6, ASP.NET Core melihat konsumsi memori yang sangat stabil.

5.1.3 Waktu respons



Gambar 16: Waktu Respons RESTful dalam milidetik

Express.js menunjukkan peningkatan linier sebagai respons terhadap peningkatan jumlah pengguna dan jumlah baris yang diminta dari database.

ASP.NET Core meningkat secara linier sebagian besar karena jumlah baris yang diambil. Jumlah ini meningkat jauh lebih besar dengan peningkatan pengguna ketika jumlah baris yang diambil adalah 50 dan 100. Kasus (1,4) & (7, 10) dengan masing-masing 100 dan 250 pengguna, meminta satu baris menunjukkan sedikit atau tidak ada perubahan dalam waktu respons sama sekali. Namun hal ini menunjukkan peningkatan yang lebih kecil ketika mencapai 500 pengguna yang meminta satu baris. ASP.NET Core juga lebih terpengaruh oleh peningkatan jumlah baris yang diminta dibandingkan Express.js.

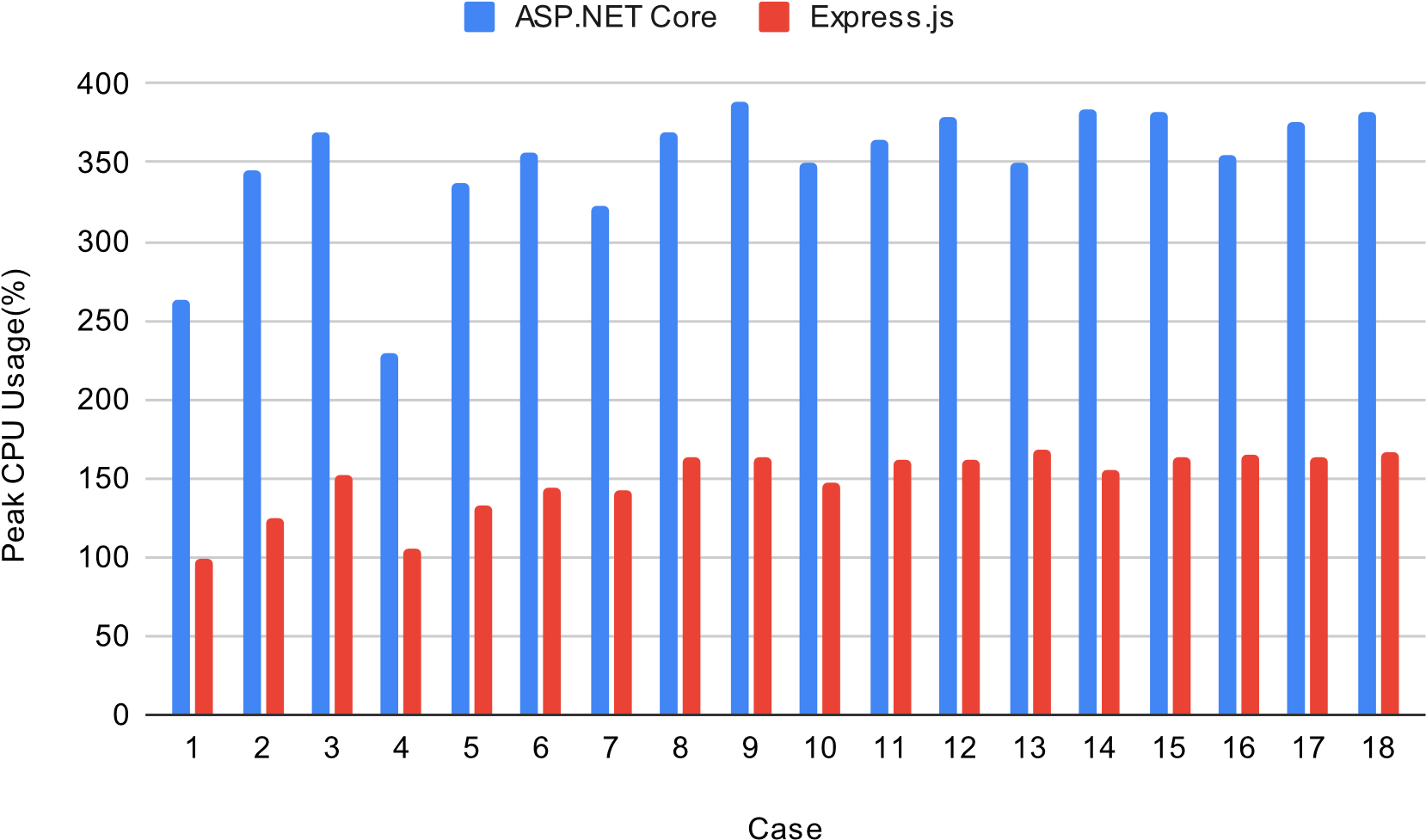
Semua kasus di mana ASP.NET Core memiliki waktu respons rata-rata yang lebih tinggi, ia juga memiliki waktu respons maksimum yang jauh lebih tinggi, yaitu respons paling lambat terhadap suatu permintaan.

Dalam semua kasus pengujian yang meminta satu baris, ASP.NET Core terbukti lebih cepat secara konsisten dibandingkan Express.js (lihat Gambar 16). Hal ini juga berlaku untuk waktu respons minimum dan maksimumnya, ASP.NET Core mampu melayani semua permintaan lebih cepat daripada Express.js.

Hal sebaliknya dapat dilihat pada kasus di mana 50 & 100 baris diminta. Di sini ASP.NET Core menampilkan minimum yang lebih tinggi dan maksimum yang lebih tinggi, hingga hampir 2x selama kasus 14, 15, 17 dan 18.

## 5.2 Pertanyaan Penelitian 2

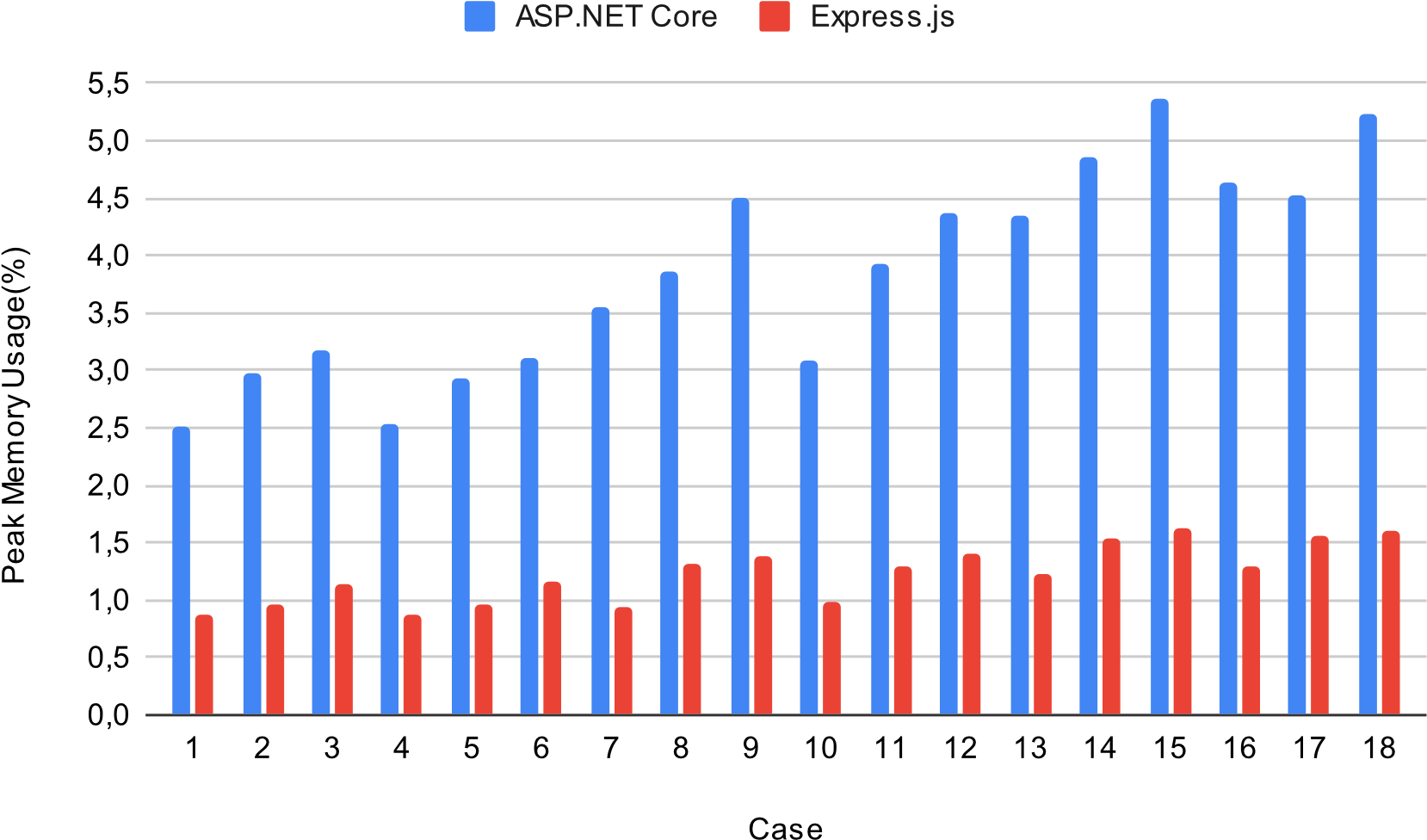
5.2.1 Penggunaan CPU



Gambar 17: Penggunaan CPU GraphQL

Seperti yang ditunjukkan pada Gambar 17 ASP.NET Core menggunakan jumlah CPU yang sangat tinggi bahkan untuk beban terendah. Dengan semua kecuali dua kasus uji yang menggunakan keempat inti. Untuk sebagian besar kasus pengujian, ASP.NET Core menggunakan lebih dari dua kali lipat jumlah yang digunakan Express.js.

5.2.2 Penggunaan Memori



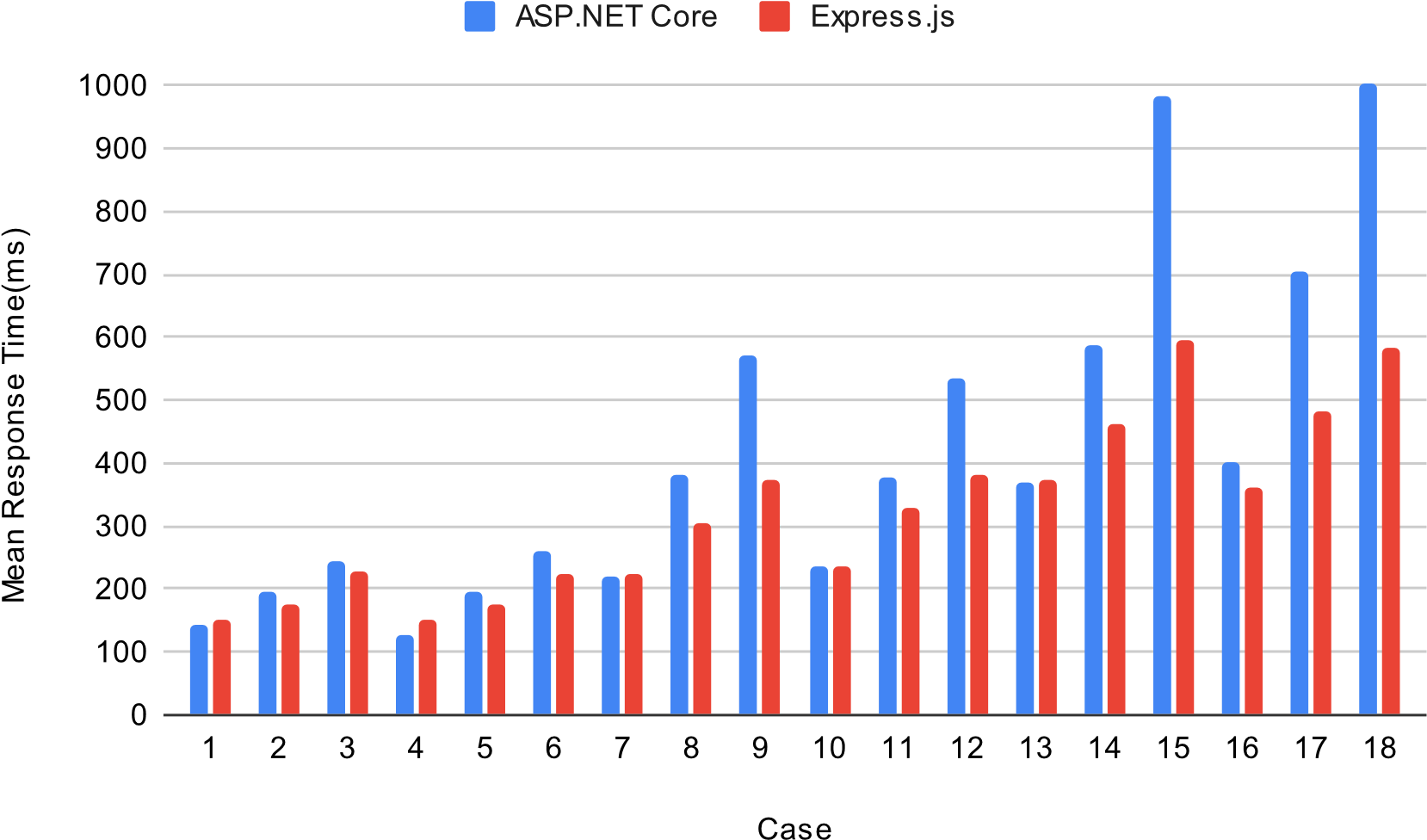
Gambar 18: Penggunaan Memori GraphQL

Untuk semua kasus, ASP.NET Core menggunakan lebih banyak memori secara signifikan dibandingkan

Express.js (lihat Gambar 18). Express.js mencapai puncaknya sedikit di atas 1,5% sedangkan ASP.NET

Core menggunakan hampir 5,2% selama kasus pengujian yang sama. Selama beban terendah, ASP.NET Core menggunakan sekitar 2,8x jumlah Express.js, dan hingga 3,6x lebih banyak pada beban tinggi.

5.2.3 Waktu respons



Gambar 19: Waktu Respons Rata-rata GraphQL

Kedua kerangka kerja menunjukkan peningkatan linier dalam waktu respons sebagai respons terhadap peningkatan jumlah data yang diambil dari database dan jumlah pengguna yang membuat permintaan. Express.js mengalami peningkatan yang jauh lebih kecil dan stabil dibandingkan dengan ASP.NET Core yang mengalami peningkatan lebih besar seiring bertambahnya beban. Peningkatan terbesar dapat dilihat pada Gambar 19 selama kasus 15 dan 18 untuk ASP.NET Core di mana 500 pengguna meminta 100 baris.

# 6 Kesimpulan & Pembahasan

## 6.1 Temuan

Pertanyaan Penelitian 1 - Apa perbedaan antara ASP.NET Core Web API dan Express.js Web API dalam penggunaan CPU, penggunaan memori, dan waktu respons ketika diimplementasikan dengan arsitektur RESTful?

ASP.NET Core melihat penggunaan CPU yang jauh lebih tinggi dibandingkan dengan Express.js. Alasannya adalah karena kesederhanaan event loop Node.js. Karena semua kode dieksekusi di thread utama, artinya ia hanya mengeksekusi kode untuk satu permintaan dalam satu waktu. Sedangkan ASP.NET Core menggunakan multithreading dan mengeksekusi kode untuk beberapa permintaan secara bersamaan, sehingga menghasilkan pemanfaatan CPU yang jauh lebih tinggi.

ASP.NET Core terbukti mampu melayani permintaan lebih cepat daripada Express.js seperti yang terlihat pada kasus 1, 4, 7, 10, 13, dan 16. Ini adalah kasus pengujian di mana hanya satu baris yang diminta. Namun karena penggunaan CPU-nya yang tinggi, hal ini juga berarti bahwa ketika permintaan mulai mengambil dan memproses lebih banyak data, permintaan tersebut dengan cepat mencapai penggunaan penuh CPU dan menjadi terhambat dalam jumlah pekerjaan yang dapat dilakukan secara bersamaan, sedangkan Express.js tidak terlalu terpengaruh. karena penundaan bawaan yang berasal dari loop acara. ASP.NET Core juga melihat penggunaan memori yang lebih tinggi di seluruh kasus pengujian, seperti penggunaan CPU. Hal ini disebabkan oleh multithreading, ia memproses banyak permintaan pada saat yang sama, sehingga mengalokasikan lebih banyak memori untuk objek dan thread.

Penggunaan memori ASP.NET Cores selama kasus 7, 10, 13 dan 16 terasa lebih tinggi meskipun jumlah data yang diambil lebih rendah dibandingkan kasus berikut. Hal ini disebabkan setiap permintaan hanya mengambil satu baris, yang berarti lebih sedikit objek yang dialokasikan dibandingkan jika lebih banyak baris yang diambil. Dan karena lebih sedikit objek yang dialokasikan, pengumpul sampah tidak berjalan sesering mungkin karena heap tidak terlalu sering penuh, sehingga menyebabkan lebih banyak memori yang digunakan.

Pertanyaan Penelitian 2 - Apa perbedaan antara ASP.NET Core Web API dan Express.js Web API dalam penggunaan CPU, penggunaan memori, dan waktu respons ketika diimplementasikan dengan GraphQL?

Sebagian besar perilaku yang dicatat dalam RESTful API juga dapat diamati di sini, kecuali diperkuat karena kerja ekstra yang dilakukan saat memproses permintaan GraphQL. Kedua kerangka kerja menunjukkan kecepatan waktu respons yang serupa selama beban lebih rendah dan berbeda dengan REST, tidak ada kasus di mana ASP.NET Core mampu melayani permintaan jauh lebih cepat daripada Express.js. ASP.NET Core melihat peningkatan besar dalam waktu respons saat mencapai beban yang lebih tinggi seperti kasus 14, 15, 17 dan 18. ASP.NET Core melihat penggunaan CPU yang jauh lebih tinggi bahkan untuk beban terendah dan dengan sangat cepat penggunaannya menjadi hampir sama.

maksimal, menyebabkan kelaparan benang yang lebih besar. Express.js melihat penggunaan memori yang jauh lebih rendah, memuncak pada 1,6% dibandingkan dengan puncak ASP.NET Cores sebesar 5,3%.

Hal menarik yang terjadi dalam penggunaan memori ASP.NET Core di sini adalah penggunaan memori mulai rendah dan bertambah secara bertahap akan meningkat selama beberapa tahun. iterasi hingga mencapai keadaan stabil. Alasannya adalah semakin banyak alokasi yang dilakukan, frekuensi upaya pengumpul sampah untuk mengosongkan memori juga meningkat. Gabungkan hal ini dengan kekurangan thread dan beberapa permintaan yang membutuhkan waktu sangat lama untuk dipenuhi, hal ini menyebabkan banyak objek masih direferensikan saat pengumpul sampah berjalan. Artinya, beberapa objek akan dipromosikan ke generasi lebih tinggi yang lebih jarang dibersihkan, sehingga menyebabkan peningkatan penggunaan memori.

## 6.2 Implikasi

Hasil dari studi eksperimental ini adalah perbandingan yang tidak memihak antara dua kerangka kerja berbeda yang digunakan untuk pengembangan Web API. Tergantung pada kasus penggunaan aplikasi, beban yang diharapkan, dan kinerja yang diinginkan, kerangka pilihan bervariasi. ASP.NET Core akan bekerja lebih baik untuk aplikasi yang diharapkan melakukan pekerjaan intensif CPU, dan untuk aplikasi sederhana yang mengambil lebih sedikit data dari database. Namun Express.js sangat cocok untuk aplikasi intensif I/O yang tidak akan melakukan banyak perhitungan. Perlu diingat bahwa ASP.NET Core sangat bergantung pada CPU yang kuat sehingga tidak mengalami kekurangan thread dan jika ketersediaan CPU terbatas, Express.js adalah pilihan yang valid.

## 6.3 Keterbatasan

Meskipun melakukan pengujian ini pada jaringan lokal membuat hasilnya lebih terfokus pada kinerja kerangka kerja yang sebenarnya, namun juga tidak memiliki hasil yang mencerminkan skenario dunia nyata di mana semua permintaan dikirim melalui internet. Demikian pula permintaan ini hanya dibuat untuk membaca data dari database yang berarti tidak mengukur kinerja saat data sedang ditulis.

Karena GraphQL bukan asli dari kedua kerangka kerja tersebut, ini berarti solusi pihak ketiga harus digunakan sebagai gantinya. Artinya, sebagian perbedaan performa antara kedua implementasi ini mungkin disebabkan oleh implementasi itu sendiri dan oleh karena itu tidak dapat diterapkan secara langsung ke semua versi GraphQL untuk kerangka kerja ini.

## 6.4 Kesimpulan

Seperti yang dibicarakan dalam pendahuluan, ada lebih banyak faktor selain kinerja yang menentukan kerangka kerja mana yang paling cocok untuk proyek tertentu. Namun penelitian dan studi kasus menunjukkan bahwa kinerja dapat berdampak signifikan terhadap keberhasilan aplikasi web modern.

Tujuan utama dari penelitian ini adalah untuk mempelajari dan membandingkan kinerja antara ASP.NET Core dan Express.js. Hal ini dilakukan dengan membangun dua aplikasi yang menggunakan arsitektur RESTful dan bahasa kueri GraphQL, lalu mengukur performanya selama berbagai jenis pemuatan. Dua set percobaan dilakukan. Set pertama mengukur kinerja aplikasi saat menggunakan arsitektur RESTful. Di sini diamati bahwa ASP.NET Core mampu menghasilkan respons yang jauh lebih cepat pada beban yang lebih rendah, namun seiring dengan meningkatnya beban, Express.js mampu mencapai waktu respons yang lebih cepat dengan penggunaan yang lebih sedikit.

sumber daya. Selama set kedua aplikasi menggunakan bahasa kueri GraphQL. Di sini, kerja ekstra yang diperlukan per permintaan di GraphQL membuat kinerja ASP.NET Core menjadi lebih buruk selama beban yang lebih tinggi karena kekurangan thread, sedangkan Express.js bekerja jauh lebih efisien selama semua kasus pengujian.

Hasil penelitian ini harus dilihat dengan mempertimbangkan keterbatasan. Meskipun hasilnya menunjukkan bahwa Express.js akan berperforma lebih baik dalam sebagian besar kasus, namun belum tentu hal tersebut akan selalu terjadi. Melakukan lebih banyak perhitungan per permintaan dapat mengubah hasil yang mendukung ASP.NET Core.

6.5 Pekerjaan di Masa Depan

Kedua framework ini hanyalah dua dari sekian banyak framework yang dapat digunakan untuk membuat Web API. Tes yang sama dapat dilakukan pada kerangka kerja yang berbeda untuk mendapatkan gambaran yang lebih baik mengenai kekuatan dan kelemahannya. Hal yang sama berlaku untuk lingkungan hosting dan database. Penelitian ini hanya mengevaluasi framework yang berjalan di Ubuntu dan menggunakan database MySQL. Mungkin satu kerangka kerja bekerja lebih baik di lingkungan lain seperti Windows atau Docker, atau saat menggunakan database nonrelasional. Karena penelitian ini juga hanya berfokus pada membaca dari database, maka dapat diperluas dengan mengukur kinerja saat menulis ke database atau mengelola berbagai jenis data.

Telah diamati bahwa menjalankan Apache JMeter pada mode non-CLI akan memberikan hasil yang sangat berbeda karena klien tidak menginisialisasi semua permintaan secepat yang diharapkan. Artinya, perbedaan dalam kecepatan internet dapat berdampak besar dalam skenario dunia nyata dan melakukannya pada jenis jaringan yang berbeda jika diperlukan.

# 7 Referensi

1. Douglas Schmidt dan F. Buschmann. “Pola, kerangka kerja, dan middleware: Hubungan sinergisnya”. Dalam: Juni 2003, hlm. 694–704.isbn:0-7695-1877-X. doi:10.[1109/ICSE.2003.1201256.](https://doi.org/10.1109/ICSE.2003.1201256)
2. Stefan Haefliger, Georg von Krogh, dan Sebastian Spaeth. “Penggunaan Kembali Kode dalam Perangkat Lunak Sumber Terbuka”. Di dalam:Ilmu Manajemen54.1 (2008), hlm.180–193. tidak:00251909, 15265[501.url:http://www.jstor.org/stable/20122369.](http://www.jstor.org/stable/20122369)

[3]Survei Pengembang Stack Overflow - Kerangka Web. Februari 2020.url:[https: // insights.stackoverflow.com/survey/2020#technology-web-frameworksallrespondents2.](https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-all-respondents2)

[4]Survei Pengembang Stack Overflow - Kerangka Web yang Paling Disukai, Ditakuti, dan Dicari. Februari 2020.url:htt[ps : // wawasan . aliran tumpukan. com / survey / 2020 # teknologi - kerangka web yang paling - disukai - ditakuti - dan - diinginkan -.](https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-web-frameworks)

1. Wendell Santos.API menunjukkan Tingkat Pertumbuhan Lebih Cepat pada tahun 2019 dibandingkan Tahu[n-Tahun Sebelumnya. Juli 2019.url:https://www.programmableweb.com/ news/apis-showfaster-growth-rate-2019-previous-years/research/2019/07/17.](https://www.programmableweb.com/news/apis-show-faster-growth-rate-2019-previous-years/research/2019/07/17)
2. Matius Clark.Bagaimana BBC membangun situs web berskala besar. 17 Januari 2018.

[url: https : / / www . blok kreatif . com / feature / how - the - bbc - builds - situs web berskala itu (dikunjungi p](https://www.creativebloq.com/features/how-the-bbc-builds-websites-that-scale)ada 19/09/2020).

[7]Studi Kasus Masak. 2017.url:h[ttps://blog.eggplantsoftware.com/ casestudies/cook-increases-conversions-by-seven-percent-thankstofaster-load-time](https://blog.eggplantsoftware.com/case-studies/cook-increases-conversions-by-seven-percent-thanks-to-faster-load-time) (dikunjungi pada 19/09/2020).

[8] Pontus Erlandsson dan Joakim Remes. “PERBANDINGAN KINERJA Antara GraphQL, REST & SOAP”. 2020.

[9]Wendell Santos.Jenis API dan Gaya Arsitektur Mana yang Paling Banyak Digunakan? Nov[ember 2017.url:https://www.programmableweb.com/news/ which-apitypes-and-architectural-styles-are-most-used/research/2017/11/ 26.](https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26)

[10] G. Brito dan MT Valente. “REST vs GraphQL: Eksperimen Terkendali”. Di dalam:

Konferensi Internasional IEEE tentang Arsitektur Perangkat Lunak (ICSA) 2020. 2020, hlm.81–91.

[11]Survei Pengembang Stack Overflow 2017.url:ht[tps://insights.stackoverflow. com/ survey/2017#technology-\_-databases (dikunjung](https://insights.stackoverflow.com/survey/2017#technology-_-databases)i pada 19/09/2020).

[12]Survei Pengembang Stack Overflow 2019.url:ht[tps://insights.stackoverflow. com/ survey/2018#technology-\_-databases (dikunjung](https://insights.stackoverflow.com/survey/2018#technology-_-databases)i pada 19/09/2020).

[13]Survei Pengembang Stack Overflow 2019.url:ht[tps://insights.stackoverflow. com/ survey/2019#technology-\_-databases (dikunjung](https://insights.stackoverflow.com/survey/2019#technology-_-databases)i pada 19/09/2020).

[14]Survei Pengembang Stack Overflow 2020.url:ht[tps://insights.stackoverflow. com/ survey/2020#technology-databases (dikunjun](https://insights.stackoverflow.com/survey/2020#technology-databases)gi pada 19/09/2020).

[15]Apa itu MySQL?url:https:[//dev.mysql.com/doc/refman/8.0/en/whatismysql.html (diku](https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html)njungi pada 19/09/2020).

[16]Pangsa Pasar Sistem Operasi.url:https://[bit.ly/3i0W3r3 (dikunjungi p](https://bit.ly/3i0W3r3)ada 18/09/2020).

[17]GraphQL— Bahasa kueri untuk API Anda.url:https://g[raphql.org/ (dikunjungi](https://graphql.org/) pada 07/02/2020).

1. Arnar Freyr Helgason. “Analisis kinerja Layanan Web: Perbandingan antara layanan web RESTful & GraphQL”. 2017.
2. Jonas Hefer.GraphQL menjelaskan. 23 Mei 2016.ur[l:https://www.apollograp](https://www.apollographql.com/blog/graphql/basics/graphql-explained/)hql. [com/blog/graphql/basics/graphql-explained/.](https://www.apollographql.com/blog/graphql/basics/graphql-explained/)

[20]Apa itu Inti ASP.NET?url:https://[dotnet.microsoft.com/learn/aspnet/what-isaspnet-core (dikunjungi pada 2](https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core)8/07/2020).

[21]Kelas Utas C#.url:https://[docs.microsoft.com/en-us/dotnet/api/ system.threading.thread?view=net-5.0.](https://docs.microsoft.com/en-us/dotnet/api/system.threading.thread?view=net-5.0)

[22]Kelas Tugas C#.url:htt[ps://docs.microsoft.com/en-us/dotnet/api/ system.threading.tasks.task?view=net-5.0.](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=net-5.0)

[23] Stephen Toub.Peningkatan Kinerja di .NET Core 3.0. 15 Mei 2019. url:https:// dev[blogs.microsoft.com/dotnet/kinerja-improvementsin-net-core-3-0/.](https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-core-3-0/)

[24]roslyn. 2021.url:htt[ps://github.com/dotnet/roslyn.](https://github.com/dotnet/roslyn)

[25] Mark J. Harga.C# 8.0 dan .NET Core 3.0 - Pengembangan Lintas Platform Modern. Edisi keempat. Paket, 2019.isbn:9781788478120.

[26]Manajemen memori dan pengumpulan sampah (GC) di ASP.NET Core. 2021. [url:https://docs.microsoft.com/en-us/aspnet/core/ Performance/ memory?view=aspnetcore-5.0](https://docs.microsoft.com/en-us/aspnet/core/performance/memory?view=aspnetcore-5.0).

[27].Konsep Manajemen Memori NET. 2021.url:https:[//www.jetbrains. com/ help/dotmemory/](https://www.jetbrains.com/help/dotmemory/NET_Memory_Management_Concepts.html#large-object-heap)

[NET\_Memory](https://www.jetbrains.com/help/dotmemory/NET_Memory_Management_Concepts.html#large-object-heap)\_Management\_Concepts.html#largeobject-heap.

[28]Cepat.url:https:[//expressjs.com/ (dikunjung](https://expressjs.com/)i pada 27/07/2020).

1. Magnus Greiff dan André Johansson. “Symfony vs Express: Perbandingan Kerangka Sisi Server”. 2019.
2. Niklas Lund. “Node.js vs PHP Perbandingan teknologi sisi server”. 2017.

[31]libuv. 2021.url:htt[p://docs.libuv.org/en/v1.x/.](http://docs.libuv.org/en/v1.x/)

[32]Penjadwalan kerja kumpulan thread - dokumentasi libuv.url:h[ttp://docs.](http://docs.libuv.org/en/latest/threadpool.html)

[libuv.org/en/latest/threadpool.html (dikun](http://docs.libuv.org/en/latest/threadpool.html)jungi pada 21/05/2021).

[33]Loop Peristiwa Node.js, Timer, dan process.nextTick().url:https : [/ / nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/.](https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/)

[34]Apa itu v8.url:http[s://v8.dev/docs.](https://v8.dev/docs)

[35]Mengaktifkan penerjemah Ignition.url:h[ttps://v8.dev/blog/](https://v8.dev/blog/ignition-interpreter) igniti[oninterpreter](https://v8.dev/blog/ignition-interpreter).

[36]Menggali TurboFan JIT.url:https://v8.[dev/blog/turbofan-jit.](https://v8.dev/blog/turbofan-jit)

[37] Georgiy Krylov dkk. “Evolusi Pengumpulan Sampah di V8: Mesin JavaScript

Google”. Di dalam:Konferensi Mediterania ke-9 tentang Komputasi Tertanam (MECO) tahun 2020. 2020, hlm.1–6.doi:[10.1109/MECO49872.2020.9134326.](https://doi.org/10.1109/MECO49872.2020.9134326)

[38]Fungsi panggilan balik.u[rl:https://developer.mozilla.org/en-US/docs/ Glosarium/Callback\_function.](https://developer.mozilla.org/en-US/docs/Glossary/Callback_function)

[39] Justin Zobel.Menulis untuk Ilmu Komputer. ke-3. Perusahaan Penerbitan Springer, Incorporated, 2015.isbn:1447166388.

[40]Apache JMeter. 2021.url:htt[ps://jmeter.apache.org/.](https://jmeter.apache.org/)

[41]Statistik Wikimedia. 2021.url:http[s://stats.wikimedia.org/#/en.](https://stats.wikimedia.org/#/en.wikipedia.org/reading/unique-devices/normal%7Cline%7C2-year%7C(access-site)~mobile-site*desktop-site%7Cmonthly)

[wikipedia.org/reading/unique- devices/normal%7Cline%7C2- tahun%](https://stats.wikimedia.org/#/en.wikipedia.org/reading/unique-devices/normal%7Cline%7C2-year%7C(access-site)~mobile-site*desktop-site%7Cmonthly)

[7C(akses - situs ) ~seluler - situs \* desktop - situs % 7Cbulanan](https://stats.wikimedia.org/#/en.wikipedia.org/reading/unique-devices/normal%7Cline%7C2-year%7C(access-site)~mobile-site*desktop-site%7Cmonthly) (dikunjungi pada 02/03/2020).

[42]pemalsu - npm. 2021.url:[https://www.npmjs.com/package/faker.](https://www.npmjs.com/package/faker)

[43]Galeri NuGet — GraphQL 3.3.2. 2021.url:https://[www.nuget.org/packages/](https://www.nuget.org/packages/GraphQL/) Grap[hQL/.](https://www.nuget.org/packages/GraphQL/)

[44]grafikql - npm. 2021.url:ht[tps://www.npmjs.com/package/graphql.](https://www.npmjs.com/package/graphql)

[45]ekspres-graphql - npm. 2021.url:htt[ps : / / www . npmjs . com / paket / express-graphql.](https://www.npmjs.com/package/express-graphql)

# Lampiran A Lampiran

Sebuah SQL

## BUAT TABEL`kategori`(

`kategori\_id`int BUKAN NULLAUTO\_INCREMENT, ` nama`varchar(255)BUKAN BATAL, `keterangan`varchar(255)BUKAN BATAL, ` tanggal dibuat`tanggal DEFAULT NULL, `tanggal terakhir diperbarui`tanggal DEFAULT

NULL, KUNCI UTAMA(`kategori\_id`)

) MESIN=InnoDB AUTO\_INCREMENT=1BAWAANSET KARAKTER=utf8mb4;

## BUAT TABEL`komentar`(

`komentar\_id`int BUKAN NULLAUTO\_INCREMENT, `

thread\_id`int BUKAN NULL, `identitas pengguna`int

BUKAN NULL,

`tubuh`varchar(255)BUKAN BATAL, ` tanggal dibuat`tanggal DEFAULT NULL, `tanggal terakhir diperbarui`tanggal DEFAULT NULL, KUNCI UTAMA(`komentar\_id`), KUNCI` thread\_id\_fk`(`thread\_id`), KUNCI`pengguna\_id\_fk`

(`identitas pengguna`)

) MESIN=InnoDB AUTO\_INCREMENT=1BAWAANSET KARAKTER=utf8mb4;

## BUAT TABEL`benang`(

`thread\_id`int BUKAN NULLAUTO\_INCREMENT, `

kategori\_id`int BUKAN NULL, `identitas pengguna`int

BUKAN NULL,

`judul`varchar(255)BUKAN BATAL, ` tubuh`varchar(255)BUKAN BATAL, ` tanggal dibuat`tanggal DEFAULT NULL, `tanggal terakhir diperbarui`tanggal DEFAULT

NULL, KUNCI UTAMA(`thread\_id`), KUNCI`kategori\_id\_fk`(`kategori\_id`), KUNCI`

pengguna\_id\_fk`(`identitas pengguna`)

) MESIN=InnoDB AUTO\_INCREMENT=1BAWAANSET KARAKTER=utf8mb4;

BUAT TABEL`pengguna`(

`identitas pengguna`int BUKAN NULL

AUTO\_INCREMENT, `nama`varchar(255)BUKAN BATAL, `surel`varchar(255)BUKAN BATAL, `telepon` varchar(255)BUKAN BATAL, `tanggal dibuat`tanggal DEFAULT NULL,

`tanggal terakhir diperbarui`tanggal DEFAULT NULL, KUNCI

UTAMA(`identitas pengguna`)

) MESIN=InnoDB AUTO\_INCREMENT=1BAWAANSET KARAKTER=utf8mb4i;

## B Perintah Atas & Grep

atas -d 0,1 | grep --line-buffered node >> node.txt top -d 0,1 | grep --line-buffered dotnet >> dotnet.txt Aplikasi C Faker.js

konstantapemalsu=memerlukan('pemalsu') konstanta

DbHandler=memerlukan('./dbhandler')

vardb=baruDbHandler() pemalsu.seed(1) konstantaiterasi= 1001;

//buat 1000 pengguna & kategori untuk(Saya= 1; Saya<iterasi; Saya++){ konsol.log("Lingkaran "+Saya)

varnama=pemalsu.internet.namapengguna() var surel=pemalsu.internet.email() vartelepon= pemalsu.telepon.nomorponsel() vartanggal dibuat =pemalsu.tanggal.masa lalu() varterakhir diperbarui=pemalsu.tanggal.masa depan() db.insertUser(nama, email, telepon, tanggal pembuatan, pembaruan terakhir, ↪→ fungsi(hasil){

})

varNama Kategori=pemalsu.lorem.kata() var keterangan=pemalsu.lorem.kata() vartanggal dibuat=pemalsu.tanggal.masa lalu() varterakhir diperbarui=pemalsu.tanggal.masa depan()

db.insertCategory(nama kategori, deskripsi, tanggal pembuatan, ↪→ terakhir diperbarui,fungsi(hasil){

})

}

//untuk setiap kategori, masukkan 1 thread untuk(kategoriId= 1; kategoriId<iterasi; kategoriId++){

varidentitas pengguna= 1000

varjudul=pemalsu.lorem.kata() vartubuh= pemalsu.lorem.kata() vartanggal dibuat= pemalsu.tanggal.masa lalu() varterakhir diperbarui

=pemalsu.tanggal.masa depan()

db.insertThread(categoryId, userId, judul, isi, tanggal pembuatan, ↪→ terakhir diperbarui,fungsi(hasil){

})

identitas pengguna-=1

}

//Untuk setiap pengguna, masukkan 200 komentar di thread 1-200 untuk(identitas pengguna= 1; identitas pengguna<iterasi; identitas pengguna++){ untuk(threadId= 1; threadId< 201; threadId++){ vartubuh= pemalsu.lorem.kata() vartanggal dibuat= pemalsu.tanggal.masa lalu() varterakhir diperbarui= pemalsu.tanggal.masa depan() db.insertComment(threadId, userId, body, datemade, ↪→ terakhir diperbarui,fungsi(hasil){

})

}

}

## D Repositori Aplikasi

<https://github.com/oliverkarlsson59/bachelor-aspnet><https://github.com/oliverkarlsson59/bachelor-express>