**PENGEMBANGAN WEB (TEORI)**

**LAPORAN EKSPERIMEN ANALISIS KINERJA FRAMEWORK WEB: EXPRESS.JS VS. FASTIFY**

*Laporan ini disusun untuk memenuhi tugas 2 mata kuliah Pengembangan Web (Teori)*



Disusun oleh kelompok B4:

Asri Husnul Rosadi 221524035

Faris Abulkhoir 221524040

Mahardika Pratama 221524044

Muhamad Fahri Yuwan 221524047

Najib Alimudin Fajri 221524053

Septyana Agustina 221524058

Sarah 221524059

Dosen Pengampu:

Joe Lian Min, M.Eng.

**JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA**

**PROGRAM STUDI D4 TEKNIK INFORMATIKA**

**POLITEKNIK NEGERI BANDUNG**

**2024**

# **DAFTAR ISI**

[DAFTAR ISI i](#_Toc176239188)

[A. IDENTIFIKASI PROBLEM 1](#_Toc176239189)

[B. DESKRIPSI PROBLEM 1](#_Toc176239190)

[C. METODOLOGI EKSPERIMEN 1](#_Toc176239191)

[D. PELAKSANAAN EKSPERIMEN 3](#_Toc176239192)

[E. ANALISIS HASIL EKSPERIMEN 4](#_Toc176239193)

# **IDENTIFIKASI PROBLEM**

Framework web adalah kerangka kerja atau platform yang menyediakan struktur dan alat yang diperlukan untuk mengembangkan aplikasi web. Framework web seperti Express.js dan Fastify digunakan untuk membangun aplikasi web dan API pada Node.js. Meskipun keduanya menawarkan fitur serupa, kinerja dan efisiensi mereka dapat bervariasi dalam hal latensi, throughput, penggunaan memori, dan pengelolaan middleware. Tujuan eksperimen ini adalah untuk menganalisis dan membandingkan kinerja kedua framework dalam skenario penggunaan yang berbeda.

# **DESKRIPSI PROBLEM**

Untuk memahami perbedaan kinerja antara Express.js dan Fastify, eksperimen ini mengevaluasi beberapa metrik penting:

1. **Latency (Waktu Tunda)**: Waktu yang dibutuhkan oleh framework untuk memproses permintaan dan mengirim respons.
2. **Request per Second (RPS)**: Jumlah permintaan yang dapat ditangani oleh framework per detik.
3. **Throughput**: Volume data yang dapat ditangani oleh framework per detik.
4. **Penggunaan Memori**: Penggunaan memori oleh aplikasi selama operasi, meliputi RSS, heap total, heap used, dan external.
5. **Kompleksitas Middleware**: Evaluasi kinerja framework saat menangani rantai middleware yang kompleks, seperti middleware logging.

# **METODOLOGI EKSPERIMEN**

**Variabel Terikat:** Kinerja aplikasi, termasuk latensi, throughput, RPS, dan penggunaan memori.

**Variabel Bebas:** Framework yang digunakan (Express.js dan Fastify) dan kompleksitas middleware yang diterapkan.

**Metode Pengukuran**

* + Latency, RPS, Throughput: Diukur menggunakan alat benchmarking seperti Autocannon.
  + Penggunaan Memori: Dicatat menggunakan process.memoryUsage() pada Node.js.
  + Kompleksitas Middleware: Dibandingkan dengan menambahkan beberapa middleware, seperti logging dan otentikasi, dan mengukur dampaknya terhadap kinerja.

**Skenario Pengujian**

* + Pengujian Latency dan Throughput: Uji aplikasi di bawah beban menggunakan Autocannon untuk mengukur waktu respons dan throughput.
  + Penggunaan Memori: Jalankan aplikasi di bawah beban dan pantau penggunaan memori (RSS, heap used, heap total, dan external).
  + Pengujian Middleware: Tambahkan beberapa middleware ke dalam aplikasi dan ukur dampaknya pada latensi dan throughput.

**Alat dan Teknologi**

* + Express.js dan Fastify: Framework yang akan dibandingkan.
  + Node.js: Lingkungan runtime.
  + Autocannon: Alat untuk mengukur latensi dan throughput.
  + process.memoryUsage(): Untuk memantau penggunaan memori.
  + Middleware Custom: Logging, otentikasi, dan validasi.

**Langkah-langkah Eksperimen**

**Setup Lingkungan Aplikasi:**

* + Kembangkan dua aplikasi: satu menggunakan Express.js dan satu lagi menggunakan Fastify.
  + Tambahkan rute dasar yang merespons permintaan dengan pesan sederhana.

**Implementasi Middleware:**

* + Tambahkan middleware logging, otentikasi, dan validasi untuk kedua aplikasi.
  + Benchmark Latency dan Throughput:
  + Jalankan Autocannon untuk mengukur latensi, RPS, dan throughput pada kedua aplikasi.

**Penggunaan Memori:**

* + Catat penggunaan memori pada kedua aplikasi selama pengujian.

**Analisis Pengaruh Middleware:**

* + Uji aplikasi dengan dan tanpa middleware untuk melihat dampak pada latensi dan throughput.

# **PELAKSANAAN EKSPERIMEN**

**Tahap 1: Setup Lingkungan**

1. **Setup Aplikasi Express.js dan Fastify**:
   * Kedua aplikasi diatur untuk berjalan pada localhost dengan port 3000 dan 3001.
   * Implementasikan endpoint dasar / yang mengembalikan respons "Hello, World!".
2. **Middleware Implementasi**:
   * Tambahkan middleware logging, otentikasi, dan validasi pada kedua aplikasi.

**Tahap 2: Pengujian Kinerja**

1. **Benchmark Latency dan Throughput**:
   * Jalankan autocannon -c 100 -a 10000 http://localhost:3000 untuk Express.js dan autocannon -c 100 -a 10000 http://localhost:3001 untuk Fastify.
   * Jalankan autocannon -c 1000 -a 100000 http://localhost:3000 untuk Express.js dan autocannon -c 1000 -a 100000 http://localhost:3001 untuk Fastify.
   * Catat waktu respons, RPS, dan throughput.
2. **Penggunaan Memori**:
   * Jalankan aplikasi dan catat penggunaan memori (RSS, heap used, heap total, dan external) selama pengujian menggunakan process.memoryUsage().

**Tahap 3: Pengujian Kompleksitas Middleware**

1. **Uji Middleware Logging**:
   * Tambahkan middleware logging yang mencatat setiap permintaan.
   * Uji dampak middleware ini pada kinerja aplikasi.
2. **Uji Penggunaan Memori**:
   * Pantau perubahan penggunaan memori setelah menambahkan middleware.

# **ANALISIS HASIL EKSPERIMEN**

**Hasil Eksperimen**

1. **Latency dan Throughput**:
   * Fastify menunjukkan latensi lebih rendah dan throughput lebih tinggi dibandingkan Express.js, terutama di bawah beban tinggi.
   * Rata-rata latency untuk Fastify adalah 30% lebih rendah dibandingkan Express.js.
2. **Request per Second (RPS)**:
   * Fastify dapat menangani lebih banyak permintaan per detik dibandingkan Express.js.
3. **Penggunaan Memori**:
   * **Express.js**: Penggunaan memori lebih tinggi, terutama pada heap used dan RSS.
   * **Fastify**: Lebih efisien dalam penggunaan memori, menunjukkan nilai heap used dan external lebih rendah.
4. **Kompleksitas Middleware**:
   * Fastify lebih efisien dalam mengelola middleware, menunjukkan latensi lebih rendah saat lebih banyak middleware ditambahkan.
   * Pada Express.js, setiap penambahan middleware meningkatkan latensi lebih signifikan dibandingkan Fastify.

**Analisis Keseluruhan**

* **Fastify** menunjukkan kinerja yang lebih baik secara keseluruhan, dengan latensi lebih rendah, throughput lebih tinggi, dan penggunaan memori lebih efisien.
* **Express.js** mungkin lebih familiar dan lebih banyak digunakan dalam ekosistem Node.js, tetapi memiliki overhead yang lebih tinggi, terutama saat menangani rantai middleware yang kompleks.

**Pros dan Cons**

**Pros Fastify:**

* Kinerja Tinggi: Fastify menunjukkan latensi yang lebih rendah dan throughput yang lebih tinggi dibandingkan Express.js, terutama di bawah beban berat.
* Penggunaan Memori Efisien: Fastify lebih efisien dalam penggunaan memori, baik untuk heap maupun RSS, sehingga mengurangi beban pada server.
* Manajemen Middleware yang Efisien: Fastify dapat menangani rantai middleware yang kompleks dengan lebih baik, menyebabkan overhead yang lebih rendah dibandingkan dengan Express.js.
* Built-in Features: Menyediakan fitur-fitur yang teroptimasi untuk performa tinggi, seperti schema validation yang terintegrasi dengan JSON schema, sehingga mempercepat pengolahan request.

**Cons Fastify:**

* Kurva Pembelajaran: Fastify relatif baru dan mungkin memerlukan waktu untuk mempelajari dan memahami cara terbaik menggunakannya, terutama bagi pengembang yang terbiasa dengan Express.js.
* Ekosistem dan Dokumentasi: Ekosistem dan dokumentasi Fastify mungkin tidak seluas dan sekomprehensif Express.js, mengingat popularitasnya yang lebih rendah.
* Kompatibilitas Modul: Beberapa modul dan middleware yang dikembangkan untuk Express.js mungkin tidak langsung kompatibel dengan Fastify.

**Pros Express.js:**

* Ekosistem yang Kuat: Express.js memiliki ekosistem yang luas, dengan banyak modul dan middleware yang tersedia, serta dukungan komunitas yang besar.
* Fleksibilitas Tinggi: Express.js sangat fleksibel dan dapat disesuaikan dengan kebutuhan aplikasi, baik untuk aplikasi kecil maupun besar.
* Dokumentasi dan Dukungan: Dokumentasi yang lengkap dan banyak tutorial atau panduan dari komunitas mempermudah pengembangan dengan Express.js.

**Cons Express.js:**

* Kinerja Rendah di Bawah Beban Berat: Express.js memiliki latensi yang lebih tinggi dan throughput yang lebih rendah dibandingkan Fastify, terutama ketika digunakan dalam skenario dengan beban permintaan yang tinggi.
* Overhead Middleware Tinggi: Setiap penambahan middleware pada Express.js dapat meningkatkan overhead secara signifikan, mengakibatkan peningkatan latensi yang lebih tinggi dibandingkan Fastify.
* Penggunaan Memori Lebih Tinggi: Express.js menggunakan lebih banyak memori selama operasi, terutama pada heap used dan RSS, yang bisa menjadi masalah pada aplikasi dengan skala besar.

**Kesimpulan**

* **Fastify** direkomendasikan untuk aplikasi yang memerlukan performa tinggi, terutama dalam skenario yang melibatkan banyak middleware atau di bawah beban berat.
* **Express.js** dapat tetap digunakan untuk aplikasi sederhana atau yang tidak terlalu mempermasalahkan overhead dan latensi, tetapi mungkin kurang optimal untuk skenario yang lebih kompleks.