BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam era digital saat ini, kita tidak bisa lepas dari aplikasi web. Hampir setiap aktivitas kita sehari-hari, baik itu bekerja, belanja, hiburan, atau berkomunikasi dilakukan melalui aplikasi web. Seiring dengan perkembangan teknologi, aplikasi web pun semakin berkembang pesat. Mulai dari aplikasi sederhana seperti media sosial, hingga aplikasi kompleks seperti e-commerce, semuanya menggunakan berbagai jenis arsitektur perangkat lunak yang berbeda-beda.

Salah satu arsitektur perangkat lunak yang sedang populer saat ini adalah arsitektur microservices. Microservice itu sendiri adalah sebuah proses kohesif dan independen yang saling berinteraksi melalui pesan (Dragoni et al., 2017). Dengan kata lain Microservice adalah pendekatan dalam pengembangan perangkat lunak yang memecah sebuah aplikasi menjadi bagian-bagian yang lebih kecil dan mandiri. Setiap bagian tersebut memiliki tugas dan fungsi yang spesifik, dan dapat berkomunikasi dengan bagian lainnya melalui antarmuka yang terdefinisi dengan jelas (Lewis & Fowler, 2014). Sedangkan arsitektur microservice adalah aplikasi terdistribusi di mana semua modulnya adalah microservice (Dragoni et al., 2017).

Arsitektur microservices memiliki beberapa kelebihan dibandingkan dengan arsitektur monolitik, di mana seluruh aplikasi dibangun dalam satu kesatuan. Pertama, arsitektur microservices memungkinkan pengembang untuk bekerja secara mandiri pada bagian-bagian tertentu dari aplikasi tanpa harus mempengaruhi bagian lainnya (Richards, 2022). Hal ini memungkinkan pengembangan dan perbaikan aplikasi menjadi lebih cepat dan efisien.

Selain itu, arsitektur microservices juga memungkinkan skalabilitas yang lebih baik. Dalam arsitektur monolitik, jika terdapat peningkatan jumlah pengguna atau permintaan, maka seluruh aplikasi harus ditingkatkan, bahkan jika hanya bagian tertentu yang membutuhkan peningkatan. Dalam arsitektur microservices,

bagian-bagian yang membutuhkan peningkatan dapat ditingkatkan secara mandiri, sehingga aplikasi dapat lebih efisien dan responsif.

Salah satu kekurangan dari arsitektur microservices yang perlu dicermati adalah kompleksitas dalam manajemen infrastruktur dan koordinasi antar bagian aplikasi(Richards, 2022). Sebagai contoh, ketika sebuah aplikasi web menggunakan arsitektur microservices, maka setiap service harus berkomunikasi dengan service lainnya untuk memastikan semua bagian aplikasi dapat berjalan dengan lancar dan saling terintegrasi dengan baik. Ketika data yang ditransfer antar service semakin besar, maka bisa terjadi *bottleneck* dalam komunikasi antar service tersebut. Hal ini dapat memperlambat respons waktu aplikasi, bahkan dapat menyebabkan kegagalan dalam operasional aplikasi.

Namun, kompleksitas dalam manajemen infrastruktur dan koordinasi antar bagian aplikasi dalam arsitektur microservices tidak selalu menghasilkan performa yang optimal (Richardson, 2018). Seperti yang telah dijelaskan sebelumnya, ketika data yang ditransfer antar service semakin besar, maka bisa terjadi bottleneck dalam komunikasi antar service tersebut. Salah satu solusi untuk mengatasi masalah ini adalah dengan mengoptimalkan proses komunikasi antar service dalam arsitektur microservices. Salah satu cara yang dapat dilakukan adalah dengan mengurangi ukuran data yang ditransfer antar service dengan mengkompresinya. Pada umumnya format data yang sering digunakan dalam komunikasi antar service dalam arsitektur microservices adalah JSON (Salah et al., 2017). JSON merupakan format data yang ringan, mudah dibaca dan dipahami oleh manusia, dan juga mudah diinterpretasikan oleh mesin (Bray, 2017). Namun, meskipun JSON tergolong ringan, ukuran data yang dikirimkan antar service dapat tetap besar ketika jumlah data yang ditransfer semakin banyak. Contoh nya pada json array dimana sebuah array memiliki beberapa JSON object yang semua nya memiliki key yang sama.

Metode kompresi yang dapat mengkopresi json object yang pertama adalah menggunakan metode HPack. HPack adalah protokol kompresi header HTTP yang dirancang khusus untuk digunakan dalam HTTP/2. HPack adalah sebuah motode kompresi yang menghilangkan *header redundant*, membatasi kerentanan terhadap

serangan keamanan, dan memiliki persyaratan memori terbatas untuk digunakan dalam lingkungan yang terbatas (Peon & Ruellan, 2015).

HPack dapat di gunakan untuk mengatasi masalah komunikasi antar service, di mana jumlah request yang dilakukan oleh browser dalam satu waktu bisa sangat besar. Dalam situasi seperti ini, jika response dikirimkan dalam format yang tidak terkompresi, maka ukuran data yang dikirimkan dapat menjadi sangat besar. HPack pada penelitian ini bertujuan untuk mengurangi ukuran data response dengan cara mengeliminasi redundansi dalam JSON Object dan mengurangi ukuran index yang digunakan untuk mereferensikan kembali response yang sama.

Salah satu metode kompresi data yang populer adalah Gzip. Gzip merupakan metode kompresi data yang mengurangi ukuran data dengan cara menghilangkan redundansi dalam data. Dengan menggunakan Gzip, ukuran data yang dikirimkan antar service dapat dikurangi hingga ~95% (OBJELEAN, 2011). Hal ini dapat mengurangi beban komunikasi antar service dalam arsitektur microservices, sehingga performa aplikasi dapat meningkat.

Dalam konteks implementasi HPack dan Gzip pada optimasi aplikasi dengan arsitektur microservices, penggunaan HPack dan Gzip dapat membantu mengoptimalkan proses komunikasi antar service dengan mengurangi ukuran data response yang dikirimkan, sehingga dapat mengurangi beban komunikasi antar service dalam arsitektur microservices dan meningkatkan performa aplikasi.

Dalam penelitian sebelumnya disimpulkan bahwa HPack yang saat di kombinasikan dengan Gzip untuk mengkopresi JSON Object dapat mengurangi ukuran JSON sebanyak ~97% (OBJELEAN, 2011).

Topik ini sangat menarik untuk diteliti karena masih sedikit peneliti yang membahas masalah ini. Selain itu, HPack juga belum banyak digunakan dan belum banyak dibahas tentang metode ini. Selain itu, belum ada peneliti yang mengkombinasikan arsitektur microservice dan metode kompresi untuk tujuan optimasi.

Dari penjelasan di atas, penulis akan membahas topik "Penerapan HPack dan Gzip dalam Optimalisasi Aplikasi dengan Arsitektur Microservices" dalam penelitian ini.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka yang menjadi rumusan masalah dalam penelitian ini adalah :

1. Bagaimana implementasi dari Hpack dan Gzip dalam mengkompresi ukuran response pada arsitektur microservices?
2. Apakah kompresi response dari microservice dapat mengoptimasi aplikasi berbasis arsitektur microservices?

1.3 Tujuan Penelitian

Berdasarkan latar belakang dan rumusan masalah yang telah dijelaskan sebelumnya, tujuan dari penelitian ini adalah untuk mengoptimalkan proses komunikasi antar *service* dalam arsitektur microservices dengan menggunakan metode kompresi data. Penelitian ini akan memfokuskan pada implementasi dan evaluasi performa metode kompresi data Gzip dan HPack pada komunikasi antar *service* dalam aplikasi web berbasis arsitektur microservices yang menggunakan format data JSON. Tujuan dari penelitian ini adalah untuk memperoleh hasil evaluasi kinerja dari metode kompresi data Gzip dan HPack dalam mengoptimalkan proses komunikasi antar *service* dalam arsitektur microservices, sehingga dapat memberikan rekomendasi terbaik mengenai metode kompresi data yang paling sesuai digunakan dalam lingkungan arsitektur microservices dengan format data JSON.

1.4 Manfaat Penelitian

Beberapa keuntungan yang dapat diperoleh dari penelitian ini adalah:

1. Memberikan pemahaman lebih mendalam tentang penggunaan arsitektur microservices dalam pengembangan aplikasi web, khususnya dalam hal manajemen infrastruktur dan koordinasi antar bagian aplikasi.
2. Memberikan solusi dalam mengoptimalkan proses komunikasi antar service dalam arsitektur microservices dengan mengurangi ukuran data yang ditransfer antar *service* dengan menggunakan metode kompresi data, seperti Gzip dan HPack.

3. Meningkatkan performa aplikasi web yang menggunakan arsitektur microservices dengan mengurangi beban komunikasi antar *service* melalui penggunaan metode kompresi data.
4. Memberikan rekomendasi kepada pengembang aplikasi web dalam pemilihan metode kompresi data yang tepat sesuai dengan kebutuhan dan karakteristik aplikasi yang dikembangkan.
5. Menjadi referensi dan acuan bagi penelitian selanjutnya yang terkait dengan penggunaan arsitektur microservices dan metode kompresi data dalam pengembangan aplikasi web.

1.5 Ruang Lingkup Dan Batasan Penelitian

Penelitian ini akan fokus pada penerapan HPack dan Gzip dalam optimalisasi aplikasi dengan arsitektur microservices. Ruang lingkup penelitian meliputi:

1. Analisis terhadap teknik kompresi data JSON, khususnya HPack dan Gzip.
2. Evaluasi efektivitas HPack dan Gzip dalam mengurangi ukuran data yang ditransfer antar *service* pada arsitektur microservices.
3. Studi kasus pada aplikasi yang menggunakan arsitektur microservices untuk menunjukkan penerapan dan dampak penggunaan HPack dan Gzip.

Sebagai bagian dari ruang lingkup penelitian, penelitian ini akan memfokuskan pada analisis dan evaluasi terhadap teknik kompresi data JSON, khususnya HPack dan Gzip, dalam rangka mengoptimalkan kinerja aplikasi dengan arsitektur microservices. Namun, ada beberapa batasan dalam penelitian ini yang perlu diperhatikan, yaitu:

1. Penelitian terbatas pada metode kompresi HPack dan Gzip; metode kompresi lainnya tidak akan dibahas secara detail.
2. Penelitian ini tidak akan membahas secara mendalam mengenai aspek keamanan yang terkait dengan penggunaan HPack dan Gzip.
3. Penelitian ini hanya membahas optimasi melalui kompresi JSON *response* atau *request*.

4. Penelitian hanya akan mencakup aplikasi yang menggunakan arsitektur microservices; aplikasi dengan arsitektur lain, seperti monolitik, tidak akan dibahas di dalam penelitian ini.

BAB II

TINJAUAN KEPUSTAKAAN

2.1 Arsitektur Microservices

2.1.1 Definisi dan Karakteristik dari Arsitektur Microservices

Arsitektur Microservices adalah pendekatan yang digunakan dalam pengembangan aplikasi, di mana aplikasi dibagi menjadi beberapa komponen yang lebih kecil dan independen yang disebut "microservices" (Newman, 2021). Setiap microservice bertanggung jawab untuk menjalankan fungsionalitas spesifik dan berkomunikasi dengan microservice lain melalui API yang terdefinisi dengan baik (Lewis & Fowler, 2014). Karakteristik utama dari arsitektur microservices meliputi:

Modularitas: Microservices memungkinkan pemisahan fungsionalitas menjadi modul-modul kecil yang independen dan mudah dikelola (Newman, 2015).

Skalabilitas: Setiap microservice dapat diskalakan secara individual sesuai dengan kebutuhan, memungkinkan aplikasi untuk mengakomodasi beban yang beragam (Lewis & Fowler, 2014).

Keberagaman teknologi: Microservices memungkinkan penggunaan teknologi yang berbeda dalam satu aplikasi, karena setiap microservice dapat dikembangkan dengan bahasa pemrograman, kerangka kerja, dan basis data yang berbeda (Newman, 2021).

2.1.2 Kelebihan dan Kekurangan Arsitektur Microservices

Kelebihan arsitektur Microservices meliputi:

Kemudahan pengembangan dan pemeliharaan: Microservices memudahkan pengembangan dan pemeliharaan aplikasi dengan memisahkan fungsionalitas menjadi modul yang lebih kecil dan independen (Newman, 2021).

Fleksibilitas dalam pemilihan teknologi: Microservices memungkinkan tim untuk menggunakan teknologi yang paling sesuai untuk kebutuhan spesifik mereka (Lewis & Fowler, 2014).

Resiliensi: Kegagalan dalam satu microservice umumnya tidak akan mengakibatkan kegagalan seluruh aplikasi, sehingga meningkatkan keandalan (Newman, 2021).

Kekurangan arsitektur Microservices meliputi:

Kompleksitas: Mengelola dan mengkoordinasikan banyak microservices dapat meningkatkan kompleksitas sistem secara keseluruhan (Newman, 2021).

Tantangan komunikasi: Komunikasi antara microservices dapat menyebabkan overhead tambahan dan mempengaruhi kinerja aplikasi (Lewis & Fowler, 2014).

3. Kesulitan dalam pengujian: Pengujian aplikasi yang dibangun dengan arsitektur microservices mungkin lebih sulit karena ketergantungan antar layanan dan kebutuhan untuk menguji komunikasi antara layanan (Newman, 2021).

2.1.3 Implementasi Arsitektur Microservices pada Aplikasi

Untuk mengimplementasikan arsitektur microservices pada aplikasi, langkah-langkah berikut dapat diikuti:

Identifikasi komponen: Tentukan fungsionalitas utama aplikasi dan pecah menjadi komponen independen yang dapat dikembangkan sebagai microservices (Newman, 2021).

Desain API: Buat API yang terdefinisi dengan baik untuk komunikasi antara microservices (Lewis & Fowler, 2014).

Pilih teknologi: Tentukan teknologi yang paling sesuai untuk setiap microservice, seperti bahasa pemrograman, kerangka kerja, dan basis data (Newman, 2015).

Pengembangan dan integrasi: Kembangkan setiap microservice secara individual dan integrasikan dengan aplikasi secara keseluruhan melalui API yang telah ditentukan (Lewis & Fowler, 2014).

Monitor dan kelola: Gunakan alat pemantauan dan manajemen yang sesuai untuk mengawasi kinerja, keandalan, dan skalabilitas microservices dalam aplikasi (Newman, 2015).

Dengan mengimplementasikan arsitektur microservices, aplikasi akan menjadi lebih modular, fleksibel, dan andal. Namun, penting untuk

mempertimbangkan kompleksitas tambahan dan tantangan yang mungkin dihadapi selama pengembangan dan pengujian. Dalam banyak kasus, keuntungan dari arsitektur microservices akan melebihi kekurangannya, terutama dalam aplikasi skala besar yang memerlukan tingkat skalabilitas dan keandalan yang tinggi.

2.2 Optimasi Aplikasi

2.2.1 Definisi Optimasi Aplikasi

Optimasi aplikasi adalah proses meningkatkan kinerja, efisiensi, dan keandalan sebuah aplikasi melalui berbagai teknik dan metode (Fowler, 2019). Tujuan utama dari optimasi aplikasi adalah untuk memperbaiki pengalaman pengguna dan mencapai tujuan bisnis dengan memanfaatkan sumber daya yang ada secara efisien (Souders, 2007).

2.2.2 Jenis-jenis Optimasi Aplikasi

Optimasi aplikasi dapat dibagi menjadi beberapa jenis, di antaranya:

Optimasi kode: Melibatkan penulisan kode yang lebih efisien dan ringkas, meminimalkan kompleksitas, dan mengurangi redundansi (McConnell, 2004).

Optimasi basis data: Meningkatkan kinerja akses dan manipulasi data dengan mengoptimalkan query dan struktur basis data (Bai, 2011).

Optimasi desain UI/UX: Meningkatkan keterlibatan pengguna dan kepuasan dengan memperbaiki desain antarmuka dan alur aplikasi (Krug, 2013).

Optimasi infrastruktur: Menyesuaikan kapasitas dan konfigurasi sistem untuk memperbaiki kinerja dan skalabilitas aplikasi (Allspaw & Robbins, 2010).

2.2.3 Manfaat Optimasi Aplikasi

Beberapa manfaat dari optimasi aplikasi meliputi:

Peningkatan kinerja: Aplikasi yang dioptimalkan menawarkan waktu respons yang lebih cepat dan memproses data dengan lebih efisien (Souders, 2007).

Penggunaan sumber daya yang efisien: Optimasi mengurangi konsumsi sumber daya seperti memori, CPU, dan bandwidth, sehingga mengurangi biaya operasional (Allspaw & Robbins, 2010).

Keandalan dan stabilitas: Aplikasi yang dioptimalkan cenderung memiliki risiko kegagalan yang lebih rendah dan lebih mudah dikelola (Fowler, 2019).

Pengalaman pengguna yang lebih baik: Optimasi aplikasi menghasilkan antarmuka yang lebih responsif, intuitif, dan mudah digunakan, sehingga meningkatkan kepuasan pengguna (Krug, 2013).

2.2.4 Strategi Optimasi Aplikasi

Untuk mengoptimalkan aplikasi, beberapa strategi yang dapat diimplementasikan meliputi:

Profiling dan benchmarking: Mengukur kinerja aplikasi untuk mengidentifikasi area yang memerlukan perbaikan (Myers et al., 2011).

Analisis dan perbaikan kode: Mengkaji kode sumber dan mengimplementasikan perubahan yang diperlukan untuk meningkatkan kinerja dan efisiensi (McConnell, 2004).

Penyesuaian basis data: Mengoptimalkan query dan struktur basis data untuk mempercepat akses dan manipulasi data (Bai, 2011).

Optimasi desain dan alur aplikasi: Memperbaiki desain antarmuka dan alur aplikasi untuk meningkatkan keterlibatan dan kepuasan pengguna (Krug, 2013).

Monitoring dan analisis data: Mengumpulkan dan menganalisis data pengguna untuk mengidentifikasi area yang memerlukan perbaikan dan menyesuaikan strategi optimasi (Allspaw & Robbins, 2010).

Kompresi pengiriman data: Mengurangi besar ukuran data dengan cara mengompresi sebelum mengirim data.

Dengan menerapkan strategi optimasi yang sesuai, aplikasi akan menjadi lebih efisien, responsif, dan ramah pengguna, sehingga menciptakan pengalaman yang lebih baik bagi pengguna dan membantu mencapai tujuan bisnis.

2.3 HPack

2.3.1 Definisi HPack

HPack adalah algoritma kompresi yang digunakan untuk mengurangi ukuran data dalam komunikasi HTTP/2 menggunakan huffman encoding. Algoritma ini memanfaatkan struktur tabel dinamis yang memungkinkan pengkodean dan dekoding Header yang lebih efisien, mengurangi overhead, dan meningkatkan kinerja (Peon & Ruellan, 2015). Tujuan utama dari HPack adalah

untuk mengurangi latensi dan meningkatkan kecepatan transfer data dalam protokol HTTP/2.

2.3.2 Fungsi dan Kelebihan HPack

Fungsi utama HPack adalah untuk mengompresi Header pada protokol HTTP/2, sehingga mengurangi ukuran data yang perlu ditransmisikan. kelebihan HPack meliputi Efisiensi kompresi, HPack menggunakan struktur tabel dinamis dan stais yang mengurangi redundansi dalam data, sehingga menghasilkan tingkat kompresi yang lebih baik (Peon & Ruellan, 2015).

2.3.3 Cara Kerja HPack

HPack bekerja dengan menggunakan dua tabel, yaitu tabel statis dan tabel dinamis, untuk mengasosiasikan field header ke indeks. Tabel statis adalah tabel yang telah ditentukan sebelumnya dan berisi field header yang umum (sebagian besar dengan nilai kosong). Sedangkan tabel dinamis bersifat dinamis dan dapat digunakan oleh encoder untuk mengindeks field header yang diulangi dalam daftar header yang dikodekan (Peon & Ruellan, 2015).

Proses kompresi di HPack tidak secara spesifik menggambarkan algoritma untuk encoder. Sebaliknya, HPack mendefinisikan dengan tepat bagaimana decoder diharapkan beroperasi, memungkinkan encoder menghasilkan encoding apa pun yang diperbolehkan oleh definisi tersebut.

Untuk mendekompresi blok header, decoder hanya perlu mempertahankan tabel dinamis sebagai konteks decoding. Tidak ada status dinamis lain yang diperlukan. Saat digunakan untuk komunikasi dua arah, seperti dalam HTTP, tabel dinamis encoding dan decoding yang dipertahankan oleh titik akhir sepenuhnya independen, yaitu, tabel dinamis permintaan dan respons adalah terpisah (Peon & Ruellan, 2015).

Decoder memproses blok header secara berurutan untuk merekonstruksi daftar header asli. Blok header adalah penggabungan representasi field header. Setelah field header didekode dan ditambahkan ke daftar header yang direkonstruksi, field header tidak dapat dihapus. Field header yang ditambahkan ke daftar header dapat dengan aman diteruskan ke aplikasi. Dengan meneruskan field

header yang dihasilkan ke aplikasi, decoder dapat diimplementasikan dengan komitmen memori transitori minimal selain memori yang diperlukan untuk tabel dinamis (Peon & Ruellan, 2015).

Semua representasi field header yang terkandung dalam blok header diproses dalam urutan munculnya. Representasi yang diindeks melibatkan penambahan field header yang sesuai dengan entri yang dirujuk dalam tabel statis atau dinamis ke daftar header yang didekodekan. Representasi literal yang tidak ditambahkan ke tabel dinamis melibatkan penambahan field header ke daftar header yang didekodekan. Representasi literal yang ditambahkan ke tabel dinamis melibatkan penambahan field header ke daftar header yang didekodekan dan penambahan field header ke awal tabel dinamis. Penambahan ini bisa mengakibatkan pengusiran entri sebelumnya di tabel dinamis (Peon & Ruellan, 2015).

Untuk membatasi kebutuhan memori pada sisi decoder, ukuran tabel dinamis dibatasi. Ukuran tabel dinamis adalah jumlah ukuran entri-entrinya. Ukuran entri adalah jumlah panjang namanya dalam oktet, panjang nilainya dalam oktet, dan 32. Ukuran entri dihitung menggunakan panjang namanya dan nilai tanpa kode Huffman yang diterapkan.

2.4 Gzip

Gzip adalah program kompresi data yang dikembangkan oleh Jean-loup Gailly. Program ini menggunakan algoritma Lempel-Ziv (LZ77) untuk mengurangi ukuran file. Gzip menggantikan file asli dengan file yang memiliki ekstensi '.gz', sambil mempertahankan kepemilikan, mode, dan waktu akses dan modifikasi yang sama. Gzip hanya akan mencoba mengkompresi file reguler dan akan mengabaikan symbolic links .

2.4.1 Cara Kerja Gzip

Gzip mengkompresi file dengan menggantikan file asli dengan file yang memiliki ekstensi '.gz'. Jika nama file baru terlalu panjang untuk sistem file-nya, gzip akan memotongnya. Gzip mencoba memotong hanya bagian dari nama file

yang lebih panjang dari 3 karakter. Jika nama terdiri dari bagian-bagian kecil saja, bagian terpanjang akan dipotong.

2.4.2 Keuntungan Menggunakan Gzip

Gzip biasanya dapat mengurangi ukuran teks seperti kode sumber atau bahasa Inggris sebesar 60-70%. Kompresi umumnya jauh lebih baik daripada yang dicapai oleh LZW (seperti yang digunakan dalam kompres), Huffman coding (seperti yang digunakan dalam pack), atau adaptive Huffman coding (compact) (Gailly, 2022).

2.5 JSON(Javascript Object Notation)

JSON (JavaScript Object Notation) adalah format pertukaran data yang ringan dan mudah dibaca oleh manusia serta mudah diinterpretasikan oleh komputer. Dalam konteks pengembangan web dan aplikasi, JSON digunakan secara luas untuk mentransmisikan dan menyimpan data struktural.

2.5.1 Definisi dan Struktur JSON

JSON merupakan representasi struktural data yang terdiri dari pasangan "kunci-nilai" (key-value pairs). Data dalam format JSON disusun dalam objek (object), larik (array), dan tipe data primitif seperti string, angka, boolean, dan nilai null. Format JSON sering kali mengadopsi sintaks mirip dengan objek dan larik dalam JavaScript, sehingga mudah dipahami oleh para pengembang (Tiwary et al., 2021).

2.5.2 Keunggulan JSON

Salah satu keunggulan utama JSON adalah keterbacaannya (Tiwary et al., 2021). Dengan menggunakan sintaks yang intuitif dan mudah dipahami, JSON memungkinkan pengembang dan pengguna manusia untuk membaca dan mengerti data dengan mudah. Selain itu, JSON juga mendukung interoperabilitas, yang berarti format ini dapat digunakan secara luas dalam berbagai bahasa pemrograman dan dapat diimplementasikan di berbagai platform. JSON juga sering digunakan dalam API (Application Programming Interface) untuk mentransmisikan data antara server dan klien. Format JSON juga memungkinkan pengolahan data struktural dengan mudah, termasuk parsing, manipulasi, dan serialisasi.

2.5.3 Implementasi JSON

Pemrosesan JSON dapat dilakukan baik pada sisi server maupun pada sisi klien. Pada sisi server, banyak bahasa pemrograman menyediakan pustaka atau modul untuk membaca dan menulis data JSON. Sebagai contoh, PHP menyediakan fungsi json_encode() dan json_decode(), sedangkan Python menyediakan modul json. Pada sisi klien, JavaScript memiliki metode built-in untuk parsing dan mengonversi data JSON menjadi objek yang dapat diakses. Dengan menggunakan metode JSON.parse(), data JSON dapat diubah menjadi objek JavaScript yang dapat dimanipulasi dan digunakan dalam pengembangan aplikasi web (Tiwary et al., 2021).

JSON digunakan dalam berbagai kasus penggunaan di dalam pengembangan aplikasi dan pertukaran data. Salah satu contoh penggunaan umum adalah dalam komunikasi data antara server dan klien melalui API REST menggunakan format JSON. Dalam arsitektur ini, server akan menyediakan endpoint API yang mengirimkan data dalam format JSON kepada klien, yang kemudian dapat mengonsumsi data tersebut. Selain itu, JSON juga sering digunakan untuk menyimpan konfigurasi dan pengaturan aplikasi dalam file JSON yang dapat dengan mudah diakses dan dikelola. Selain itu, JSON juga memungkinkan pertukaran data antara aplikasi yang berbeda dengan menggunakan format yang seragam dan mudah dipahami.