Vol. 4, No. 2, Agustus 2023, hal. 366-375

Jurnal Computer Science and Information Technology (CoSciTech)

p-ISSN: 2723-567X
e-ISSN: 2723-5661

http://ejurnal.umri.ac.id/index.php/coscitech/index

# Optimasi *micro frontend website* untuk meningkatkan *load times*: teknik, tantangan, dan *best practice*

**Aldy Akbarrizky[*1], Afdal Ramdan Daman Huri[2], Bambang Wisnuadhi[3], Lukmannul Hakim Firdaus[4]**
Email: [1]aldy.akbarrizky.tif419@polban.ac.id, [2]afdal.ramdan.tif419@polban.ac.id, [3]bwisnuadhi@polban.ac.id, [4]lukmannul.hakim@polban.ac.id

[1234]Teknik Informatika, Jurusan Teknik Komputer dan Informatika, Politeknik Negeri Bandung

Diterima: 15 Juli 2023 | Direvisi: - | Disetujui: 28 Agustus 2023


**Abstrak**

Dalam beberapa tahun terakhir, terjadi peningkatan popularitas dari arsitektur *micro frontend* dikarenakan mulai diimplementasikan oleh perusahaan besar seperti IKEA, Starbucks, dan Amazon. Karakteristiknya yang menyerupai *microservice* membuat arsitektur ini mulai banyak diterapkan untuk meningkatkan *developer experience*. Namun, arsitektur ini memiliki beberapa masalah, salah satunya adalah performa *page load time* yang rendah. Tujuan dari dilakukannya penelitian ini adalah untuk menentukan bagaimana *best practice* dalam mengoptimasi performa *page load time* dari aplikasi *micro frontend*. Penelitian dilakukan dengan mengimplementasikan teknik optimasi seperti *code splitting, lazy loading, tree shaking, minification*, dan *utility module* kepada setiap *micro frontend* yang dimiliki oleh suatu *website*, kemudian dilakukan pengujian sebanyak 200 kali yang didapatkan menggunakan formula Lemeshow di *local* dan *server environment* menggunakan *browser* Google Chrome dengan metrik *fully loaded*, yaitu ukuran waktu yang dibutuhkan suatu *website* untuk memuat seluruh resources yang digunakan oleh *website* tersebut. Penelitian dilakukan pada aplikasi *Enterprise Resources Planning* (ERP) yang terdiri dari lima *micro frontend* dengan *framework* React, Vue, dan Angular. Hasil eksperimen yang dilakukan menunjukkan bahwa mengimplementasikan setiap teknik optimasi pada seluruh *micro frontend* dapat meningkatkan performa *page load time* aplikasi sebesar 31,79% pada *local* dan 47,5% pada *server environment*.

**Kata kunci**: *micro frontends, page load time,* optimasi*, best practices*

## *Micro frontend website optimization to increase load times: techniques, challenges, and best practice*

***Abstract***

*In recent years, there has been an increasing popularity of the micro frontend architecture due to its implementation by large companies such as IKEA, Starbucks, and Amazon. Due to its characteristics that similar to microservices, this architecture started to be implemented by various companies to improve their developer experience. However, this architecture has some issues, one of which is the performance of page load time. The objective of this research is to find and determine the best practices for optimizing the page load time of micro frontend applications and to identify the challenge involved. The research is conducted by implementing optimization techniques such as code splitting, lazy loading, tree shaking, minification, and utility modules to micro frontend website. After that, the website is tested with a sample size of 200 which determined by using Lemeshow formula. The research is conducted in both local and server environments using the Google Chrome browser and used "fully loaded" metric. The research use a simple Enterprise Resource Planning (ERP) application consisting of five micro frontends built with React, Vue, and Angular frameworks. The experimental results show that implementing all of the optimization techniques on all micro frontends can improve the application's page load time performance by 31.79% in the local and 47.5% in the server environment.*

***Keywords***: *micro frontends, page load times, optimizing, best practices*

## 1. PENDAHULUAN

### 1.1. Latar Belakang

Seiring dengan berkembangnya zaman, aplikasi berbasis *website* pada sisi *frontend development* yang menggunakan arsitektur *monolithic* dalam pengembangannya mulai menimbulkan masalah dikarenakan *codebase* yang dikembangkan mulai memiliki ukuran yang besar, sehingga mulai sulit untuk dilakukan *maintenance* terhadap *codebase* tersebut (Thoughtworks, 2020). Untuk mengatasi masalah tersebut, diusulkan suatu arsitektur baru di mana *codebase* dari aplikasi berbasis *website* sisi *frontend* dipecah menjadi aplikasi-aplikasi kecil. Arsitektur ini dikenal dengan nama *micro frontend architecture* (MFA). Selama beberapa tahun terakhir sejak arsitektur ini pertama kali diperkenalkan tahun 2016, arsitektur *micro frontend* sudah mulai digunakan di perusahaan-perusahaan besar seperti IKEA, Starbucks, dan Amazon [1]. Akan tetapi, pengimplementasian aplikasi *website* pada sisi *frontend* menggunakan arsitektur ini menimbulkan masalah dalam beberapa hal, seperti *payload size* yang besar, *shared dependencies*, dan *code duplication* [1]. Selain itu, performa *page load time* (PLT) dari aplikasi yang menggunakan MFA memiliki nilai yang lebih buruk dibandingkan dengan aplikasi *website* yang menggunakan arsitektur *monolithic* [2].

Penurunan performa PLT yang dialami *website* dengan MFA dieksplorasi lebih lanjut oleh Kroiß [3]. Eksplorasi dilakukan dengan membandingkan performa PLT dari aplikasi *Enterprise Resource Planning* (ERP) sederhana yang dibangun dengan menggunakan arsitektur *monolithic* dan MFA. Dari hasil eksplorasi yang sudah dilakukan, diketahui bahwa aplikasi ERP yang dibangun menggunakan MFA memiliki performa PLT yang lebih buruk dibandingkan dengan aplikasi yang dibangun secara *monolithic*. Aplikasi ERP dengan MFA mendapatkan nilai performa PLT sebesar 2190 ms, sedangkan dengan arsitektur *monolithic*, performa PLT dari aplikasi tersebut hanyalah sebesar 1331 ms. Performa PLT yang tidak baik merupakan salah satu masalah yang krusial dalam web *development*. Dalam buku yang ditulis oleh Rappl [4] menyebutkan bahwa performa yang diberikan oleh suatu teknologi hampir selalu dianggap sebagai fitur dari teknologi tersebut. Meskipun *performance requirement* dari setiap *website* berbeda, pada dasarnya prinsip yang selalu harus dipegang adalah "*faster is better*". Lebih lanjut lagi Rappl juga memberikan contoh pada *website* Amazon, di mana apabila performa *website* mereka menurun sebesar 100 ms, maka akan menyebabkan penjualan mereka menurun sebesar 1%.

### 1.2. Rumusan Masalah

Berdasarkan latar belakang masalah yang disampaikan, dipandang perlu dilakukan penelitian eksperimental untuk mengetahui bagaimana *best practice* dalam melakukan optimasi terhadap aplikasi *website* dengan MFA dan apa saja tantangan yang dihadapi dalam melakukan optimasi tersebut. Hal ini ditujukan agar optimasi yang dilakukan dapat menjadi efisien sehingga performa PLT dari aplikasi *micro frontend* menjadi meningkat dan memiliki *user experience* serta *user engagement* yang lebih baik. Selain itu, peningkatan performa PLT dari MFA dapat membuat para *frontend developer* menjadi tertarik untuk menerapkan arsitektur tersebut, melihat bahwa PLT merupakan salah satu pertimbangan *developer* dalam mengimplementasikan suatu teknologi.

### 1.3. Ruang Lingkup dan Batasan Masalah

Ruang lingkup penelitian ini dimulai dari kajian literatur mengenai MFA dan PLT *optimization*, penelitian kemudian dilanjutkan dengan melakukan persiapan data, aplikasi, serta *environment* yang dibutuhkan untuk menunjang kebutuhan eksperimen yang akan dilakukan. Selanjutnya, akan dilakukan pengujian performa PLT terhadap aplikasi penunjang eksperimen. Hasil dari pengujian tersebut akan dijadikan sebagai nilai *baseline* yang selanjutnya akan digunakan sebagai pembanding dengan performa PLT dari aplikasi setelah optimasi dilakukan. Setelah itu, akan dilakukan eksperimen pada aplikasi penunjang dengan mengimplementasikan teknik-teknik optimasi yang diajukan. Terakhir, hasil dari eksperimen akan dianalisis lebih lanjut untuk ditarik kesimpulan penelitian serta disusun *best practice* dan tantangan dari proses optimasi. Agar penelitian yang dilakukan dapat terfokus, ditetapkan batasan-batasan penelitian yaitu sebagai berikut.

a. Aplikasi yang dilakukan optimasi adalah aplikasi *micro frontend* berbasis *website.*
b. Optimasi hanya dilakukan pada sisi *frontend* saja, tidak ada optimasi yang dilakukan pada infrastruktur dari aplikasi/
c. Performa *website* yang dijadikan sebagai parameter adalah nilai *fully loaded*, yaitu nilai yang merepresentasikan waktu yang dibutuhkan untuk menampilkan *web page* secara utuh sejak *webpage* tersebut di-*request* oleh *user* hingga *web page* dapat digunakan oleh *user*.
d. Pengujian performa hanya dilakukan pada *browser* Chrome.
e. *Framework* yang digunakan oleh aplikasi adalah React, Vue, dan Angular

### 1.4. *Research Question* dan Hipotesis

Berdasarkan konteks penelitian yang akan dilakukan, disusun *research question* (RQ) sebagai berikut.

RQ$_1$: Bagaimana pengaruh penerapan teknik optimasi *code splitting*, *lazy loading*, *tree shaking*, *minification*, dan *utility module* terhadap PLT dari *website* dengan MFA*?*

Selain itu disusun juga hipotesis penelitian sebagai berikut.

H$_0$: Penerapan teknik optimasi *code splitting*, *lazy loading*, *tree shaking, minification,* dan *utility module* dapat meningkatkan PLT pada aplikasi *website* dengan MFA minimal sebesar 30%.

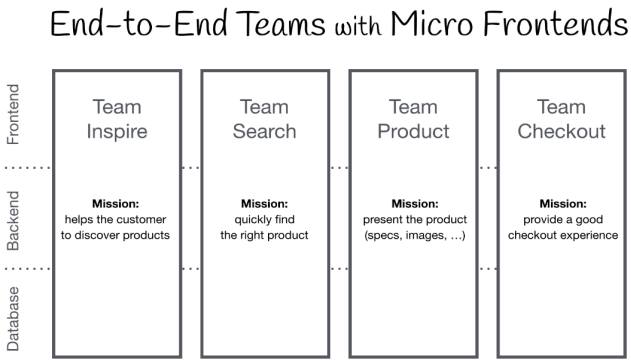
## 2. TINJAUAN LITERATUR

Untuk menunjang pelaksanaan penelitian yang dilakukan, dilakukan tinjauan terhadap beberapa literatur untuk memperdalam domain permsalahan yang akan diselesaikan. Berikut ini adalah tinjauan literatur yang dilakukan dalam penelitian.

### 2.1. *Micro frontend*

Dalam beberapa tahun terakhir *microservice* mendapatkan perhatian besar dalam bidang akademik, dalam bidang industri pun banyak perusahaan yang mengubah implementasi *back-end* aplikasi *monolitik* menjadi *microservice* [1]. *Micro frontend* adalah sebuah konsep yang terbilang cukup baru tetapi telah mendapatkan popularitas cukup signifikan di kalangan praktisi pada bidang ini. *Micro frontend* menerapkan gaya arsitektur tunggal dari *microservice*, oleh karena itu *micro frontend* menerapkan prinsip-prinsip dari *microservice* yang diterapkan oleh [5], yaitu *modeled around business domains, culture of automation, hide implementation details, decision decentralization, independent deployment*, serta *failure isolation*.

Menurut [1], seiring dengan berjalannya waktu, pengembangan aplikasi dari sisi *frontend* semakin meningkat kompleksitasnya, salah satu penyebabnya adalah dari sisi *client-side,* aplikasi terus berkembang sehingga pengembangannya menjadi lebih sulit untuk diukur terutama jika dilibatkan juga tim yang berbeda-beda dalam satu front-end yang sama secara bersamaan. Dengan adanya masalah tersebut, *micro frontend* hadir untuk memperluas konsep dari *micro service* ke dalam sisi *frontend* aplikasi. Ide dibalik konsep *micro frontend* adalah untuk berpikir bahwa suatu *website* merupakan gabungan dari beberapa fitur yang masing-masing fitur dimiliki oleh tim yang independen. Setiap tim memiliki tujuan bisnis yang harus dicapai masing-masing, dan melakukan *development* dengan konsep *end-to-end*, yaitu dari *database* hingga ke *user interface* (UI). Dapat dilihat pada Gambar 1 bagaimana pembagian tugas dalam konsep *micro frontend* bisa diilustrasikan.




Gambar 1 Ilustrasi pembagian tugas pada konsep *micro frontend*

### 2.2. JavaScript

JavaScript ditemukan pertama kali oleh Brendan Eich pada tahun 1995. Bahasa ini awalnya dikembangkan untuk Netscape 2 dan menjadi ECMA-262 pada tahun 1997. Bahasa ini telah menjadi salah satu bahasa pemrograman yang paling banyak digunakan di dunia. Bahasa ini dikenal sebagai bahasa pemrograman yang digunakan di browser web tetapi juga diadopsi secara luas untuk server dan aplikasi lainnya. Menurut [6], Javascript merupakan bahasa pemrograman *web* yang bersifat *client side programming language*. *Client side programming language* adalah tipe bahasa pemrograman yang pemrosesannya dilakukan oleh *client*. Bahasa pemrograman yang digunakan untuk mengimplementasikan *micro frontend* adalah JavaScript. JavaScript dipakai karena penggunaan *framework* pada research yang dilakukan yaitu menggunakan Single-spa. Single-spa salah satu *framework* pada JavaScript yang memudahkan penggunanya dalam melakukan implementasi *micro frontend*.

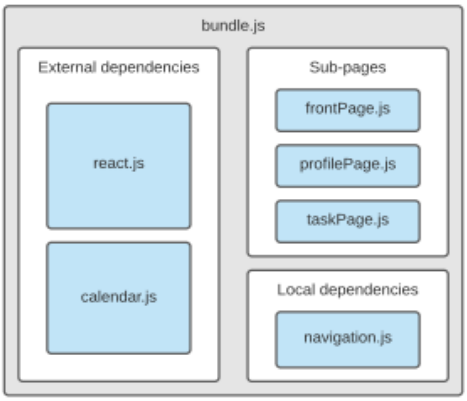
### 2.3. *Page load time*

Performa pada sebuah *website* memiliki peran yang sangat penting. Menurut [4], performa hampir selalu dianggap sebagai fitur, sebutan yang umumnya adalah "*faster is better*". Menurut Amazon yang dikutip oleh [4] juga menyatakan bahwa setiap performa web nya lambat sebesar 100 ms mereka bisa kehilangan sekitar 1% dalam penjualannya, hal tersebut merupakan hal yang tak dapat diremehkan. Hal ini menunjukkan bahwa suatu *website* harus memiliki performa yang baik. Salah satu metrik penting yang mengukur performa dan kinerja suatu *website* adalah metrik *Page Load Time* [7]. PLT adalah jumlah waktu yang dibutuhkan oleh *browser* untuk memuat seluruh *resources* yang dibutuhkan oleh suatu *web page*. Pengukuran PLT dimulai ketika *browser* mengirimkan *Hypertext Transfer Protocol* (HTTP) *request* terhadap suatu *web page* kepada *web server* hingga *resource* terakhir dari *web page* tersebut berhasil dimuat oleh *browser*. Secara umum, metrik PLT terbagi lagi menjadi dua, yaitu *load times* dan *fully loaded*. Kedua metrik tersebut mengukur hal yang sama, tetapi metrik *fully loaded* akan menunggu beberapa saat setelah *resource* terakhir dimuat untuk memastikan bahwa tidak ada lagi komunikasi antara *client* dengan *web server*. Pada penelitian ini, metrik yang digunakan adalah *fully laoded* dikarenakan metrik tersebut merupakan metrik standar untuk mengukur PLT yang sudah diterapkan oleh beberapa *tools* untuk menguji performa *website* seperti GTMetrix.

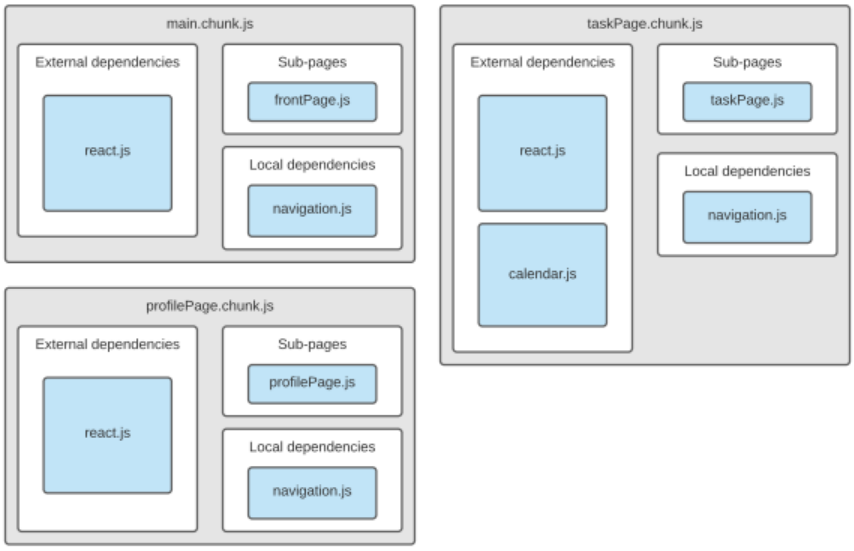
## 2.4. Teknik Optimasi PLT

### 2.4.1. *Code Splitting*

Pada saat proses memuat sebuah situs *website*, terkadang semua kode Javascript dimuat bersama dalam satu *file* besar, sehingga membutuhkan waktu lama untuk dapat dimuat. Oleh karena itu, diperlukan sebuah solusi untuk meminimalkan waktu muat yang tersebut, salah satunya adalah menggunakan *code splitting*. *Code splitting* merupakan salah satu cara yang sering digunakan untuk meningkatkan performa suatu *website*. Dengan menggunakan *code splitting*, kode yang besar akan dibagi menjadi beberapa *file* kecil dengan salah satu contohnya adalah pada Gambar 2 dan Gambar 3. Pada penelitian yang dilakukan oleh [8], penerapan *code splitting* tidak akan membuat jumlah kode pada *website* berkurang, tetapi akan mengurangi proses untuk memuat kode yang bahkan tidak dibutuhkan oleh pengguna, serta mengurangi jumlah kode yang dimuat pada proses awal.




Gambar 2 *Bundling* dari suatu *website* sebelum menerapkan *code splitting*




Gambar 3 *Bundling* dari suatu *website* setelah menerapkan *code splitting*

### 2.4.2. *Lazy Loading*

*Website* yang memiliki banyak aset yang dapat memiliki ukuran besar seperti gambar dapat menyebabkan mengakses *website* menjadi lambat. Oleh karena itu, dengan menggunakan teknik *lazy loading*, aset seperti gambar dan kode seperti JavaScript atau CSS hanya akan dimuat saat diperlukan. Hal tersebut dapat memperbaiki performa aplikasi dan mengurangi beban pada jaringan. Pada penelitian yang dilakukan oleh [9], waktu mengakses *website* dapat dikurangi secara signifikan, dari 25,27 detik menjadi 0.3 detik. Hal tersebut karena website memiliki banyak gambar, dan dengan menggunakan *lazy loading*, *website* menjadi lebih cepat karena ukuran gambar dikurangi sebesar 94,3%. Melanjutkan teknik *code splitting, lazy loading* juga menjadi sebuah kesatuan dalam teknik *website optimization*. Dengan *code splitting*, *lazy loading* akan lebih mudah memuat halaman hanya yang diperlukan.

### 2.4.3. *Tree Shaking*

Dengan menggunakan metode *tree shaking* dalam pengembangan, eksperimen yang dilakukan oleh [10]dapat mengurangi ukuran keseluruhan *file* JavaScript. *Tree shaking* adalah teknik optimasi JavaScript dengan membuang bagian-bagian kode atau modul yang tidak digunakan oleh suatu *website*. Tujuan dari dilakukannya teknik ini adalah untuk mengurangi ukuran *file* akhir sehingga mempercepat waktu muat aplikasi. Teknik ini diperkenalkan awal pada ECMAScript edisi ke-6 tahun 2015. Istilah *tree shaking* sendiri diambil dari mental model dan dependenciesnya yang menyerupai struktur pohon. Setiap *node* merepresentasikan dependensi yang menyediakan fungsionalitas berbeda. *Tree shaking* memanfaatkan fitur modul yang memungkinkan pengembang untuk menentukan bagian-bagian spesifik dari *library* atau kode sumber yang harus diambil untuk digunakan dalam aplikasi. Untuk mengimplementasikan teknik ini perlu ditulis dengan cara yang memungkinkan yaitu menggunakan *function*

*import* dan untuk mengaktifkannya harus menggunakan *method eksport.* Jika ingin menggunakan teknik ini pada *module bundler* Webpack, maka diperlukan teknik *import* dan *export* dengan standar ES6 *script*. Dengan menggunakan standar tersebut, Webpack dapat mengetahui modul mana saja yang di-*export* tanpa di-*import* oleh modul lain, modul tersebut yang akan dibuang di dalam *bundle* akhir.

### 2.4.4. *Minification*

Untuk mengurangi ukuran kode pada Javascript sehingga dapat mengurangi waktu muat yang diperlukan, kode JavaScript dapat disederhanakan tanpa menghilangkan fungsionalitas dari kode tersebut. Beberapa contoh penyederhanaan kode JavaScript adalah menjadikan nama suatu variabel menjadi lebih pendek, menghapus komentar dan *whitespace*, dan lain sebagainya. Proses penyederhanaan ini dikenal dengan istilah *minification* [11]. Kode JavaScript yang menerapkan konsep *minification* pada umumnya akan membuat kode tersebut sangat sulit untuk dibaca atau dipahami oleh manusia, tetapi masih dapat dipahami oleh komputer. Untuk lebih memahami teknik ini, Gambar 4 di bawah merupakan salah satu contoh penerapan teknik *minification*.

```
// Boolean optimization
!a || !b ? b : c
// transforms into
a && b ? c : b

// Constant statement evaluation
const a = 3, b = a ? 5 : 7;
// transforms into
const a = 3, b = 5;

// Variable name mangling
const exampleFunction = (value) => value + 1;
// transforms into
const o = o => o + 1;
```

Gambar 4 Penerapan teknik *minification* pada kode JavaScript

### 2.4.5. *Utility Module*

*Utility module* merupakan suatu teknik optimasi dengan membuat modul Javascript baru yang berperan untuk menampung *dependency, library*, atau sejenisnya yang bersifat redundan. Secara konsep, setiap aplikasi mikro pada *website* dengan MFA merupakan suatu aplikasi *website* utuh, yang berarti memiliki HTML, CSS, dan JavaScript secara independen. Hal ini menimbulkan masalah baru apabila terdapat dua atau lebih aplikasi mikro yang menggunakan *dependency* yang sama dikarenakan *dependency* tersebut akan dimuat oleh *browser* lebih dari satu kali. Untuk mengatasi masalah redudansi ini, perlu dibuat suatu modul khusus untuk menampung *dependency* tersebut, sehingga *browser* hanya perlu memuat sekali tetapi dapat digunakan oleh seluruh aplikasi mikro pada *website*.

## 2.5. Formula Lemeshow

Formula Lemeshow adalah sebuah formula yang digunakan untuk menentukan jumlah sampel yang akan digunakan pada suatu penelitian [12]. Formula ini pertama kali digagas pada tahun 1990 oleh Stanley Lemeshow, David W. Hosmer Jr, dan Janelle Klar dengan tujuan untuk menentukan jumlah suatu *sample* apabila populasinya tidak diketahui. Penentuan jumlah *sample* penting dilakukan dikarenakan penelitian yang dilakukan terhadap seluruh populasi akan memakan waktu dan *resource* yang terlalu besar. Berikut ini adalah formula Lemeshow yang dapat digunakan untuk menentukan jumlah *sample* apabila jumlah populasinya tidak diketahui.

$$n = \frac{z_{1-\alpha/2}^2 P(1-P)}{d^2}$$

Berikut ini adalah keterangan setiap notasi yang terdapat pada **Error! Reference source not found.** di atas.

$n$ = Jumlah sampel yang akan dicari
$z$ = angka standar *error* yang jauh dari rata-rata (dengan 1.645 apabila 90% keyakinan, 1.960 apabila 95% keyakinan, 2.576 apabila 99% keyakinan)
$d$ = presisi yang dapat dibuat sekecil yang diinginkan hanya dengan meningkatkan ukuran sample n (range dari 0.01 s.d. 0.25)
$P$ = proporsi populasi (dengan *range* 0.05 sampai 0.90 dengan *increment* 0.05)

## 3. METODE PENELITIAN

### 3.1. Variabel penelitian

Dalam penelitian ini, terdapat satu variabel bebas dan satu variabel terikat yang akan menjadi fokus dari penelitian. Berikut ini adalah penjelasan dari masing-masing variabel penelitian.

a. Variabel bebas

Terdapat satu variabel bebas pada penelitian ini, yaitu teknik optimasi yang akan diterapkan pada aplikasi penunjang eksperimen. Teknik optimasi yang digunakan merupakan teknik-teknik optimasi yang akan biasa diimplementasikan untuk melakukan optimasi pada *website* dengan arsitektur *monolithic.* Teknik-teknik ini akan digunakan untuk mengoptimasi aplikasi penunjang eksperimen, yaitu aplikasi *website* dengan MFA. Terdapat beberapa teknik optimasi yang akan digunakan, yaitu *code splitting*, *lazy loading, tree shaking, minification*, dan satu teknik optimasi khusus untuk *website* dengan MFA yaitu *utility module*. Teknik-teknik tersebut akan diimplementasikan baik secara tunggal maupun secara bersamaan untuk menemukan kombinasi teknik yang paling baik untuk mengoptimasi aplikasi dengan arsitektur *micro frontend*.

b. Variabel terikat

Variabel terikat pada penelitian ini adalah nilai performa PLT dari aplikasi *website* MFA. Menurut [3], nilai PLT dari suatu *website* terbagi lagi menjadi dua, yaitu nilai *initial load time* dan *repeat load time*. Pada penelitian ini, nilai *load time* yang akan digunakan sebagai variabel terikat adalah nilai *initial load time*. Alasan pemilihan *initial load time* sebagai variabel terikat adalah karena selain *page load time* merupakan parameter performa paling penting untuk suatu *website* [7], nilai *repeat load time* dari aplikasi *website* MFA tidak memiliki perbedaan terlalu signifikan dibandingkan dengan nilai *repeat load time* dari aplikasi *monolith* [3]. Hal ini dikarenakan pada *repeat load time* dari aplikasi *website* MFA, akan dilakukan optimasi otomatis seperti proses *caching*.

### 3.2. Objek penelitian

Objek penelitian yang akan digunakan dalam penelitian ini adalah aplikasi ERP sederhana yang sebelumnya digunakan oleh [3] dalam eksperimennya untuk membandingkan performa PLT dari aplikasi *website* dengan arsitektur *monolithic* dan arsitektur *micro frontend*. Aplikasi ini dapat diakses secara *open source* pada tautan Gitlab milik Kroiß[1]. Aplikasi ini terdiri dari lima *micro frontend*, yaitu *authentication, dashboard, navigation, sales,* dan *article*, serta memiliki satu *micro frontend* berjenis aplikasi *shell* yang bertugas sebagai pembungkus dari seluruh *micro frontend* tersebut.

### 3.3. Perangkat pendukung

Perangkat pendukung yang digunakan pada penelitian ini untuk menunjang pelaksanaan eksperimen diantaranya adalah :

a. Visual Studio Code, adalah suatu aplikasi *code editor* yang akan digunakan untuk membangun aplikasi penunjang eksperimen dan juga melakukan optimasi terhadap aplikasi tersebut. Aplikasi VSCode dipilih dikarenakan memiliki banyak *extension* yang dapat membantu proses pengembangan menjadi lebih cepat, seperti ES7+ yang dapat membantu pengembangan aplikasi dengan *framework* React menjadi lebih cepat.
b. Github, adalah suatu *tools* yang digunakan untuk melakukan *version control* terhadap *library* yang dikembangkan.
c. JavaScript, adalah bahasa pemrograman yang akan digunakan untuk membangun aplikasi penunjang eksperimen dan juga bahasa yang akan digunakan untuk mengoptimasi aplikasi tersebut.
d. Google Cloud Platform, adalah suatu layanan *cloud service* yang disediakan oleh Google. GCP menyediakan banyak sekali layanan seperti *cloud computing*, *cloud storage*, *kubernetes engine*, dan lain-lain. GCP dipilih dalam penelitian ini sebagai layanan *cloud service* dikarenakan menyediakan program *free credit* sebesar $300 kepada *user*.

### 3.4. Tahapan pelaksanaan penelitian

Untuk menyelesaikan masalah dan mencapai tujuan penelitian maka dirancang tahapan pelaksanaan penelitian. Tahapan dimulai dari identifikasi masalah pada arsitektur *micro frontend* sampai dengan mendapatkan informasi mengenai efektivitas metode optimasi yang diajukan dalam meningkatkan PLT dari aplikasi *website* dengan arsitektur *micro frontend*. Setelah mendapatkan perumusan masalah dan celah penelitian dari salah satu paper rujukan, dicari pemahaman-pemahaman teori untuk mendukung penelitian dengan melakukan studi literatur. Seperti yang telah dijelaskan pada tinjauan pustaka, bahwa teori-teori pendukung tersebut menyangkut arsitektur *micro frontend*, performa PLT, dan *website optimization*.

a. Studi literatur

Tahapan studi literatur dilakukan dengan melakukan pencarian kemudian mempelajari referensi yang ditemukan terkait segala hal yang berkaitan dengan penelitian yang akan dilakukan. Literatur yang dipelajari lebih dalam adalah literatur arsitektur *micro frontend* pada pengembangan aplikasi berbasis *website*, literatur terkait teknik-teknik optimasi pada JavaScript yang dapat meningkatkan performa dari suatu website, literatur terkait definisi PLT terkait suatu *website*, dan hal apa saja yang mempengaruhi performa tersebut, serta literatur terkait penentuan jumlah sampel apabila jumlah populasi tidak diketahui untuk menentukan jumlah berapa kali pengujian harus dilakukan.

b. Analisis problem domain

Pada tahapan ini, dilakukan analisis lebih dalam terkait referensi-referensi yang sudah didapatkan pada tahapan studi literatur. Tahapan ini dilakukan dengan tujuan untuk memperdalam pemahaman terkait setiap konsep yang dibutuhkan dalam penelitian

[1] Aplikasi ERP milik Kroiß dapat diakses pada https://gitlab.com/01526926

yang akan dilakukan. Analisis yang dilakukan adalah analisis terhadap data hasil pengujian performa dari aplikasi *website* dengan MFA dan analisis terhadap teknik-teknik optimasi pada *website* yang dapat digunakan untuk meningkatkan performa PLT.

c. Penyusunan skenario eksperimen

Tahapan selanjutnya dalam penelitian adalah melakukan penyusunan skenario dari eksperimen yang akan dilakukan. Tujuan dari dilakukannya tahapan ini adalah untuk memastikan setiap variabel bebas dan variabel terikat yang sudah dijelaskan sebelumnya diuji ketika tahapan eksperimen dilakukan.

d. Persiapan aplikasi eksperimen

Aplikasi yang akan digunakan sebagai penunjang kebutuhan eksperimen adalah aplikasi ERP sederhana yang digunakan oleh [3] dalam penelitiannya untuk mengetahui perbandingan waktu *load time* dari aplikasi dengan MFA dan aplikasi dengan arsitektur *monolithic*. Pada tahapan ini, persiapan aplikasi akan dilakukan untuk memastikan aplikasi milik Kroiß dapat dijalankan pada *environment* yang serupa dengan yang digunakan oleh Kroiß pada penelitiannya. Untuk mencapai tujuan tersebut, tahapan ini terbagi menjadi dua bagian utama, yaitu pengimplementasian bagian *backend* dari aplikasi dan persiapan *environment* untuk aplikasi.

Pengimplementasian bagian *backend* dari aplikasi perlu dilakukan untuk memopulasikan aplikasi. Sedangkan untuk persiapan *environment* dari aplikasi selanjutnya terbagi lagi menjadi dua bagian, yaitu persiapan *local environment* dan *server environment*. *Local environment* diperlukan untuk mengetahui dengan pasti nilai peningkatan performa PLT dari suatu teknik optimasi. Hal ini dikarenakan pada *server environment*, terdapat suatu variabel yang tidak dapat dikontrol atau dikendalikan, sehingga nilai performa PLT tidak akan selalu konsisten setiap kali proses pengujian dilakukan. Sedangkan *server environment* dibutuhkan dalam eksperimen untuk mengetahui bagaimana pengaruh suatu teknik optimasi pada kasus dunia nyata. Persiapan *server environment* dilakukan dengan memanfaatkan *free trial* dari Google Cloud Platform. Berikut ini adalah spesifikasi untuk *server* dan *local environment* dari penelitian yang dilakukan. Tabel 1 dan 2 akan menunjukkan spesifikasi yang diperlukan untuk melaksanakan eksperimen ini.

Tabel 1 Spesifikasi dari *server environment*

| Komponen | Rincian |
|---|---|
| *Machine type* | n1-standard-1 (1 vCPU) |
| *Memory* | 3,75 GB |
| *Number of nodes* | 1 |
| *Boot disk type* | *Standard Persistent Disk* |

Tabel 2 Spesifikasi dari *local environment*

| Komponen | Rincian |
|---|---|
| *Operating System* | Windows 11 |
| *Processor* | Intel® Core™ i7-11700 2.5 GHz (16 CPUs) |
| *Memory* | 16 GB |

e. Pelaksanaan eksperimen

Sebelum pelaksanaan eksperimen dilakukan, diperlukan beberapa hal yang perlu disiapkan sebelumnya, yaitu data pengujian awal yang akan menjadi data *baseline* dari eksperimen, *environment* tempat aplikasi diujikan, aplikasi dengan berbagai *micro frontend*, teknik-teknik optimasi yang akan digunakan, serta skenario eksperimen. Pelaksanaan eksperimen akan dilakukan dengan menggunakan *tools* pengujian aplikasi *website* WebPageTest. Dari skenario yang telah disusun, terlihat bahwa eksperimen terbagi menjadi dua kelompok pengujian, yaitu *server* dan *local*. Pada setiap kelompok pengujian terdapat 5 skenario yang merupakan kombinasi 5 teknik optimasi yang sudah ditentukan pada tinjauan literatur.

f. Analisis hasil eksperimen

Berdasarkan pelaksanaan eksperimen yang dilakukan pada tahapan sebelumnya, akan didapatkan suatu data yang meliputi segala informasi terkait pengujian yang sudah dilakukan. Sebelum dilakukan analisis, akan dilakukan *data cleaning* terlebih dahulu untuk mengekstrak informasi yang dibutuhkan, di mana dalam penelitian ini adalah informasi mengenai performa PLT aplikasi. Setelah informasi PLT didapatkan, akan dilakukan analisis mengenai informasi tersebut. Analisis dilakukan dengan cara memantau hasil dari eksperimen beserta *environment* dari pengujian untuk menemukan anomali-anomali yang terjadi saat pengujian. Kemudian, akan dilakukan perhitungan untuk mengetahui seberapa besar (dalam satuan %) peningkatan dari performa PLT pada aplikasi penunjang eksperimen setelah menerapkan teknik optimasi berdasarkan skenario yang telah dibuat. Data peningkatan tersebut selanjutnya akan dianalisis lebih lanjut dengan mengamati peningkatan performa PLT yang diberikan oleh setiap kombinasi teknik optimasi untuk menyusun *best practices* dalam mengoptimasi aplikasi berbasis *website* yang menggunakan MFA.

g. Penarikan kesimpulan

Tahapan terakhir yang akan dilakukan pada penelitian ini adalah penarikan kesimpulan berdasarkan hasil analisis pada tahapan sebelumnya. Dari hasil analisis tersebut, akan ditarik kesimpulan *best practice* dan tantangan dari proses optimasi aplikasi dengan MFA. *Best practice* akan menjelaskan bagaimana cara melakukan optimasi dengan benar, sehingga performa PLT menjadi lebih

baik. Dan tantangan akan menjelaskan apa saja yang dihadapi ketika melakukan optimasi. *Best practice* dan tantangan yang sudah disusun ini selanjutnya akan digunakan untuk menjawab RQ yang telah dirumuskan.

## 4. HASIL DAN PEMBAHASAN

Eksperimen dilakukan dengan mengimplementasikan variasi dari teknik-teknik optimasi yang didapatkan dengan melakukan studi literatur, yaitu *code splitting*, *tree shaking*, *minification*, dan *utility module*. Eksperimen juga dilakukan pada dua *environment* yang berbeda, yaitu *server environment* dan *local environment*. Eksperimen yang dilakukan pada *local environment* ditujukan untuk mengetahui pengaruh suatu teknik optimasi terhadap performa PLT *website* dengan pasti dikarenakan *local environment* merupakan *environment* yang lebih mudah untuk dikontrol. Sementara itu, eksperimen dilakukan pada *server environment* untuk mengetahui bagaimana pengaruh suatu teknik optimasi terhadap performa PLT *website* pada skenario dunia nyata, mengingat pada *server environment*, terdapat beberapa variabel yang tidak dapat dikendalikan seperti *network* dan *traffic route*. Untuk mendapatkan akurasi dan presisi yang baik, pengujian pada setiap skenario akan dilakukan secara berulang sebanyak 200 kali yang didapatkan dari hasil perhitungan formula Lemeshow.

Untuk mengetahui efektivitas teknik optimasi yang diimplementasikan, diperlukan *ground truth* sebagai data pembanding. Tabel 3 di bawah memperlihatkan *ground truth* untuk performa PLT dari *website* dengan MFA tanpa diterapkan teknik optimasi apapun. Nilai performa PLT dari *website* dengan MFA tanpa dilakukan optimasi pada *server environment* adalah 3954,05925925925 ms, sedangka pada *local environment* adalah 1751, 805 ms. Pada penelitian ini ditetapkan nilai performa PLT minimum untuk dikatakan baik adalah apabila PLT meningkat setidaknya sebesar 30% pada setiap *environment*.

Tabel 3 Nilai Ground Truth dari Eksperimen

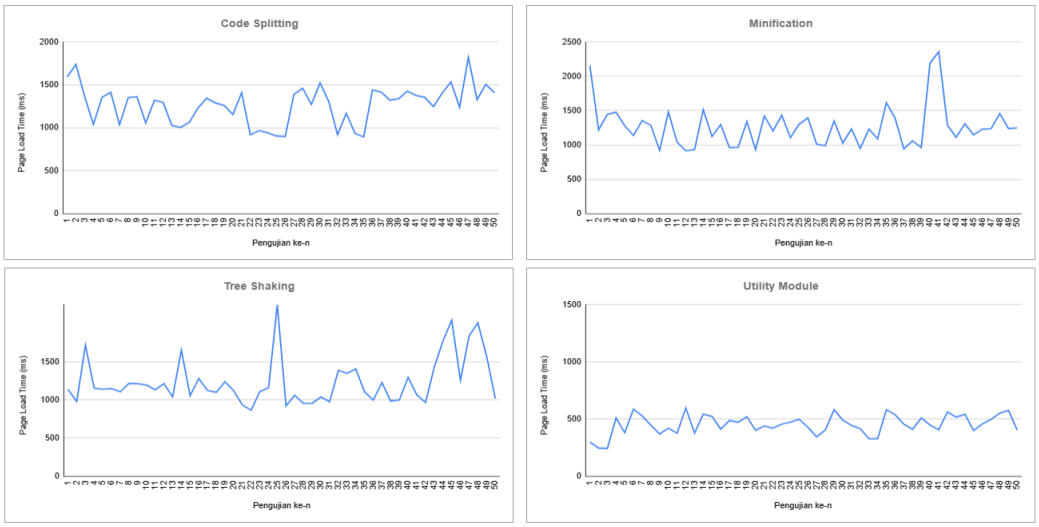
| Environment | Page Load Time (ms) |
|---|---|
| *Server* | 3954,05925925925 |
| *Local* | 1751,805 |

Analisis pengujian pada *local environment* akan dilakukan dengan menggunakan hasil eksperimen yang dilakukan pada *environment* lokal. Seperti yang sudah disebutkan sebelumnya, pengujian setiap skenario dilakukan secara berulang sebanyak 200 kali untuk mendapatkan akurasi dan presisi yang baik. Akan tetapi, dikarenakan keterbatasan *tools* yang digunakan, pengujian setiap skenario dilakukan sebanyak 4 tahap, di mana setiap tahapan akan dilakukan pengujian secara berulang sebanyak 50 kali. Selanjutnya, hasil dari keempat tahap tersebut akan dihitung nilai rata-ratanya untuk mendapatkan nilai performa PLT dari aplikasi. Tabel 4 di bawah akan menampilkan hasil eksperimen yang sudah dilakukan pada *local environment*.

Tabel 4 Hasil Pengujian pada *Local Environment*

| Baseline (ms) | Teknik Optimasi | Pengujian | | | | Rata-Rata Page Load Time (ms) | Peningkatan |
|---|---|---|---|---|---|---|---|
| | | Tahap 1 (ms) | Tahap 2 (ms) | Tahap 3 (ms) | Tahap 4 (ms) | | |
| 1751,805 | *Code Splitting* | 1267,44 | 1545.88 | 1300,54 | 1223,34 | 1334.3 | 23,83% |
| | *Tree Shaking* | 1107,23 | 1181,42 | 1393,65 | 1193,66 | 1218,99 | 30,42% |
| | *Minification* | 1247.22 | 1265.62 | 1217.14 | 1431.32 | 1290,32 | 26,34% |
| | *Utility Module* | 1322.08 | 1535.62 | 1296,02 | 1423,83 | 1394.38 | 20,40% |
| | *All Technique* | 1068,34 | 1280,44 | 1236,64 | 1193,87 | 1194,82 | 31,79% |

Pengujian pertama-tama dilakukan terhadap aplikasi penunjang yang dioptimasi hanya menggunakan teknik *code splitting* dan *lazy loading*. Setelah itu, dilakukan pengujian terhadap teknik lainnya secara bertahap menggunakan teknik *minification*, *tree shaking*, dan *utility module*. Pengujian dilakukan dengan cara dipecah menjadi 50 pengujian yang diulang 4 kali. Gambar 5 di bawah merupakan salah satu hasil pengujian dengan teknik optimasi *code splitting dengan lazy loading, tree shaking, minification,* dan *utility module*.



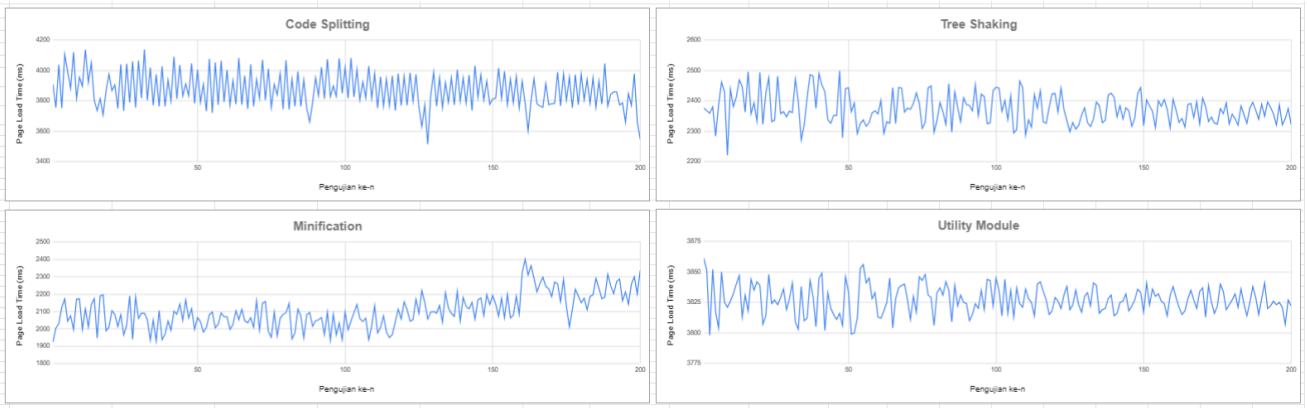

Gambar 5 Hasil eksperimen dengan menggunakan teknik *code splitting*, *tree shaking*, *minification*, dan *utility module* pada *local environment*

Analisis pengujian pada *server environment* akan dilakukan dengan menggunakan hasil eksperimen yang dilakukan pada Google Cloud Platform. Untuk pengujian pada *server environment* akan dilakukan seperti yang sudah direncanakan , yaitu sebanyak 200 kali pengulangan untuk setiap skenario. Sama seperti analisis pada *local environment*, pertama-tama akan dilihat bagaimana pengaruh masing-masing teknik optimasi dengan performa PLT. Selanjutnya, akan dilihat bagaimana pengaruh apabila seluruh teknik optimasi diterapkan secara bersamaan dengan performa PLT. Tabel 5 di bawah akan menampilkan hasil eksperimen yang sudah dilakukan pada *server environment*.

Tabel 5 Hasil eksperimen pada *server environment*

| **Teknik Optimasi** | **Rata-rata Page Load Time (ms)** | **Nilai Baseline (ms)** | **Peningkatan** |
|---|---|---|---|
| *Code Splitting* | 3872.5 | 3954.059259 | 2.06% |
| *Lazy Loading* | | | |
| *Tree Shaking* | 2373 | | 39.99% |
| *Minification* | 2102 | | 46.84% |
| *Utility Module* | 3827.133333 | | 3.21% |
| *All* | 2076 | | 47.50% |

Pengujian pertama-tama dilakukan terhadap APE yang dioptimasi hanya menggunakan teknik *code splitting* dan *lazy loading*. Akan tetapi, pengujian dilakukan secara sekaligus sebanyak 200 kali (tidak dipecah menjadi 50 pengujian yang diulang 4 kali). Setelah itu, pengujian akan dilakukan dengan mengimplementasikan *tree shaking*, *minification*, dan *utility module* secara bergantian. Gambar 6 di bawah memperlihatkan grafik dari hasil pengujian yang sudah dilakukan menggunakan teknik-teknik tersebut pada *server environment*.




Gambar 6 Hasil eksperimen dengan menggunakan teknik *code splitting*, *tree shaking*, *minification*, dan *utility module* pada *server environment*

Jika diperhatikan pada Gambar 6 dan Tabel, dapat terlihat bahwa nilai PLT dari aplikasi mengalami peningkatan yang signifikan ketika mengimplementasikan *tree shaking* dan *minification*, serta mengalami peningkatan yang tidak terlalu signifikan ketika mengimplementasikan teknik lainnya. Perbedaan peningkatan ini dikarenakan aplikasi yang digunakan bukanlah aplikasi berkala besar, sehingga teknik yang membagi modul dan kode seperti *code splitting* tidak akan terlalu efektif untuk diterapkan pada aplikasi dengan skala seperti ini.

## 5. KESIMPULAN

Optimasi terhadap *website* dengan MFA dilakukan dengan tujuan untuk meningkatkan performa PLT dari *website* tersebut. Peningkatan performa ini dilakukan untuk meningkatkan *user experience* dari *user* yang akan menggunakan *website*. Proses optimasi dilakukan dengan mengimplementasikan teknik optimasi *code splitting, lazy loading, tree shaking, minification,* dan *utility module* kepada setiap *micro frontend* yang dimiliki oleh aplikasi penunjang eksperimen, kemudian dilakukan pengujian performa PLT terhadap aplikasi tersebut. *Code splitting* merupakan teknik untuk membagi suatu modul menjadi modul-modul yang lebih kecil. Pengimplementasian *code splitting* ini biasanya dilakukan bersamaan dengan teknik *lazy loading*, yaitu teknik-teknik yang membuat modul-modul hasil *splitting* dimuat apabila dibutuhkan saja. Kemudian, *tree shaking* merupakan teknik untuk membuang modul atau bagian pada aplikasi yang tidak digunakan. Sedangkan *minification* merupakan teknik optimasi untuk membuang karakter-karakter yang tidak digunakan pada *source code* (seperti komentar dan *whitespace*), sehingga hasil *bundling* dari *source code* akan memiliki ukuran lebih kecil. Terakhir, *utility module* merupakan teknik untuk memisahkan *resources* serupa yang digunakan oleh lebih dari satu *micro frontend* menjadi satu modul tersendiri, sehingga *resources* tersebut tidak akan dimuat dua kali.

Dari hasil eksperimen yang sudah dilakukan, dapat diketahui bahwa untuk mencapai nilai performa PLT paling optimal, seluruh teknik optimasi harus diimplementasikan secara bersamaan kepada setiap *micro frontend* yang dapat dioptimasi oleh teknik tersebut, di mana pada penelitian ini, terjadi peningkatan performa PLT sebesar 47,5% pada *server environment*, dan peningkatan sebesar 31,79% pada *local environment*. Akan tetapi, sebelum teknik optimasi diimplementasikan, harus dilakukan analisis terlebih dahulu kepada setiap *micro frontend* untuk mengetahui teknik apa saja yang dapat diimplementasikan pada *micro frontend* tersebut. Selain itu, *effort* yang harus dikeluarkan untuk mengoptimasi aplikasi dengan MFA lebih besar dibandingkan dengan aplikasi *monolithic* dikarenakan optimasi harus dilakukan terhadap setiap *micro frontend*.

## Ucapan Terimakasih

Kami ingin mengucapkan terima kasih yang sebesar-besarnya kepada para dosen, Jurusan Teknik Komputer Informatika, Program Studi Teknik Informatika, dan Politeknik Negeri Bandung atas dukungan, pendanaan, dan fasilitas yang telah diberikan dalam penelitian kami.

## DAFTAR PUSTAKA

[1] S. Peltonen, L. Mezzalira, and D. Taibi, "Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review," *Inf Softw Technol*, vol. 136, Aug. 2021, doi: 10.1016/j.infsof.2021.106571.

[2] Y. Prajwal, J. Viren Parekh, and R. Shettar, "A Brief Review of Micro-frontends," *United International Journal for Research & Technology*, vol. 02, no. 08, pp. 123–126, 2021.

[3] M. Kroiß, "From Backend to Frontend: Case Study on Adopting Micro Frontends from a Single Page ERP Application Monolith," Vienna University of Technology, Vienna, 2021. [Online]. Available: www.tuwien.at

[4] F. Rappl and L. Schöttner, *The Art of Micro Frontends*, 1st ed., vol. 1. Birmingham: Packt, 2021.

[5] S. Newman, *Building microservices : designing fine-grained systems*, 1st ed., vol. 1. Sebastopol: O'Reilly Media, Inc., 2015.

[6] A. A. Setiawan, A. S. M. Lumenta, and S. R. U. A. Sompie, "Rancang Bangun Aplikasi UNSRAT E-catalog," *Jurnal Teknik Informatika*, vol. 14, no. 4, 2019.

[7] B. Pourghassemi, "What-If Analysis of Page Load Time in Web Browsers Using Causal Profiling," *Proc. ACM Meas. Anal. Comput. Syst*, vol. 3, p. 23, 2019, doi: 10.1145/3326142.

[8] D. Tanudjaja and R. Tanone, "Analisis Penerapan Code Splitting Library React pada Aplikasi Penjualan Mebel Berbasis Website," *Jurnal Teknik Informatika dan Sistem Informasi*, vol. 7, no. 2, Aug. 2021, doi: 10.28932/jutisi.v7i2.3493.

[9] F. Yan, Z. Xu, Y. Shan Zhong, S. HaiBei, and C. Yun Ge, "Research on Performance Optimization Scheme for Web Front-End and Its Practice," in *Lecture Notes in Electrical Engineering*, Springer Science and Business Media Deutschland GmbH, 2021, pp. 883–890. doi: 10.1007/978-981-15-8411-4_1.

[10] S. Huang, "Load time optimization of JavaScript web applications," Blekinge Institute of Technology, Karlskrona, 2019.

[11] J. Koivukoski, "Reducing the loading time of a single-page web application Title Reducing the loading time of a single-page web application," Thesis, Aalto University, Espoo, 2021. [Online]. Available: www.aalto.fi

[12] S. Lemeshow, D. W. Hosmer Jr, J. Klar, and S. K. Lwanga, "Statistical Methods for Sample Size Determination," in *Adequacy of Sample Size in Health Studies*, 1st ed.1990, pp. 1–44.