Journal of Digital Business and Technology Innovation (DBESTI)

https://journal.nurulfikri.ac.id/index.php/DBESTI
P-ISSN: 3047-1028 E-ISSN: 3032-775X

PENERAPAN *MICRO FRONTEND* DENGAN NEXT.JS DAN *MODULE FEDERATION* PADA APLIKASI *CASH MANAGEMENT*

Muhammad Fikri[1], Ishom Muhammad Drehem[2], Ahmad Rio Adriansyah[3]

[1,2,3]Teknik Informatika, Sekolah Tinggi Teknologi Terpadu Nurul Fikri
Depok, Jawa Barat, Indonesia 16451
muha21036ti@student.nurulfikri.ac.id, ishom.drehem@nurulfikri.ac.id, arasy@nurulfikri.ac.id

Abstract

The digitalization of banking has made cash management applications essential for business efficiency; however, the monolithic architecture of the front-end in PT. Bank XYZ's cash management application presents significant challenges in terms of scalability and flexibility, particularly with the cash pooling feature. This study implements a Micro Frontend architecture using Next.js and Module Federation to enhance development and maintenance efficiency. The results show that build times have been reduced from 1 hour to 37 minutes, with an average build time of 12 minutes achieved through parallel processes. Deployments have also become faster, with only a few hours between releases. A survey of 11 respondents, including developers, quality assurance personnel, and other key stakeholders, recorded an average satisfaction score of 4.44 out of 5, indicating strong support. This implementation accelerates feedback loops, facilitates parallel development, and streamlines the feature release process, making it highly applicable to all features within the cash management application, thereby improving scalability and flexibility comprehensively.

Keywords: *Cash Pooling, Micro Frontend, Module Federation, Next.js, Scalability.*

Abstrak

Digitalisasi perbankan menjadikan *cash management application* vital untuk efisiensi bisnis, tetapi arsitektur monolitik pada *front-end* aplikasi PT. Bank XYZ menimbulkan tantangan dalam skalabilitas dan fleksibilitas, khususnya pada fitur *cash pooling*. Penelitian ini mengimplementasikan arsitektur *Micro Frontend* menggunakan Next.js dan *Module Federation* untuk meningkatkan efisiensi pengembangan dan pemeliharaan. Hasilnya, waktu *build* aplikasi berkurang dari 1 jam menjadi 37 menit dengan rata-rata waktu per *build* 12 menit melalui proses paralel, sementara *deployment* menjadi lebih cepat dengan jeda antar-*release* hanya beberapa jam. Kuesioner terhadap 11 responden, meliputi pengembang, QA, dan pemangku kepentingan lainnya, mencatat skor kepuasan rata-rata 4,44 dari 5, menunjukkan dukungan yang sangat positif. Implementasi ini mempercepat *feedback*, mendukung pengembangan paralel, dan mempermudah proses *release* fitur, menjadikannya layak diterapkan untuk seluruh fitur *cash management application* guna meningkatkan skalabilitas dan fleksibilitas.

Kata kunci: *Cash Pooling, Micro Frontend, Module Federation*, Next.js, Skalabilitas.

1. PENDAHULUAN

Transformasi digital telah menjadi kebutuhan utama bagi perusahaan di berbagai sektor, termasuk perbankan, untuk meningkatkan efisiensi dan pengalaman nasabah. Salah satu layanan penting bagi nasabah korporasi adalah *cash management*, yang membantu mengelola arus kas mereka secara efisien. PT. Bank XYZ, sebagai salah satu bank besar di Indonesia, menawarkan layanan *cash management application* dengan fitur unggulan seperti *cash pooling* untuk konsolidasi dana.

Namun, arsitektur *monolithic* pada *front-end cash management application* PT. Bank XYZ menghadirkan beberapa keterbatasan, terutama dalam hal efisiensi pengembangan dan skalabilitas. Berdasarkan data internal, waktu *build* rata-rata aplikasi *monolithic* ini mencapai satu jam. Selain itu, pengembangan fitur baru sering mengganggu fitur lain, menghambat proses pengembangan secara keseluruhan. Arsitektur *monolithic* dapat menjadi hambatan utama dalam pengembangan aplikasi perbankan skala besar, sementara transisi ke arsitektur *microservices* dapat meningkatkan fleksibilitas dan skalabilitas sistem [1].

Diterima 31 Januari 2025, Direvisi 30 Mei 2025, Diterima untuk publikasi 30 Mei 2025

Konsep *micro frontend*, memungkinkan pengembangan *front-end* yang lebih modular dan terdesentralisasi, di mana setiap tim dapat bertanggung jawab atas fitur tertentu [2]. Meski demikian, penerapan *micro frontend* dalam konteks aplikasi perbankan yang kompleks seperti *cash management*, khususnya fitur *cash pooling*, belum banyak dibahas dalam literatur.

Penelitian ini bertujuan untuk mengimplementasikan arsitektur *micro frontend* pada fitur *cash pooling* di *cash management application* PT. Bank XYZ. Pendekatan ini menggunakan *module federation*, sebuah teknik yang memungkinkan integrasi antar modul *front-end* secara dinamis. Dengan penelitian ini, diharapkan tercapai efisiensi yang lebih tinggi, skalabilitas yang lebih baik, dan risiko gangguan antar fitur yang lebih rendah. Selain itu, penelitian ini juga mengidentifikasi tantangan teknis dan operasional dalam migrasi dari arsitektur *monolithic* ke *micro frontend*, serta strategi mitigasi yang dapat diterapkan.

Untuk memfokuskan penelitian dan memastikan ketercapaian tujuan dalam waktu dan sumber daya yang tersedia, penelitian ini dibatasi pada implementasi arsitektur *micro frontend* khusus untuk fitur *cash pooling* dalam *cash management application* PT. Bank XYZ. Implementasi hanya akan mencakup pengembangan *front-end* menggunakan *framework* Next.js dan *module federation* dengan *webpack* 5 sebagai *bundler* utama, tanpa melakukan perubahan signifikan pada *back-end* sistem yang sudah ada. Evaluasi efisiensi pengembangan dilakukan berdasarkan waktu *build*, waktu *deployment*, serta kemudahan dalam melakukan perubahan pada fitur individual. Performa aplikasi akan dinilai dari waktu muat dan penggunaan sumber daya *browser*.

Pengujian dilakukan dalam lingkungan pengembangan PT. Bank XYZ, dengan uji coba terbatas yang melibatkan *subset* pengguna internal. Fokus penelitian adalah pada integrasi *micro frontend* dengan sistem *cash management* yang ada, tanpa mengubah sistem lain yang terintegrasi. Penelitian ini direncanakan selesai dalam waktu lima bulan sesuai jadwal MBKM PT. Bank XYZ, dengan harapan memberikan kontribusi signifikan terhadap pengembangan arsitektur aplikasi perbankan yang lebih *fleksibel* dan *scalable*, khususnya pada fitur *cash pooling* di sektor perbankan Indonesia.

2. METODE PENELITIAN

Penelitian ini menggunakan metode eksperimen, yang sering digunakan untuk menguji efektivitas atau kinerja suatu pendekatan tertentu. Menurut berbagai ahli, pendekatan eksperimen memiliki beragam definisi dan aplikasi yang dapat disesuaikan dengan konteks penelitian. Metode eksperimen dalam penelitian ini diterapkan untuk menguji implementasi arsitektur *micro frontend* pada *cash management application* yang sedang dikembangkan [3]. Eksperimen ini dirancang untuk mengevaluasi kinerja sistem dengan membandingkan kondisi sebelum dan setelah penerapan arsitektur *micro frontend*. Evaluasi meliputi berbagai aspek seperti performa sistem, fleksibilitas fitur, dan skalabilitas aplikasi. Dengan demikian, pendekatan ini bertujuan untuk memberikan gambaran yang jelas mengenai dampak implementasi terhadap peningkatan kualitas sistem secara keseluruhan.

Beberapa parameter yang akan diukur untuk membandingkan performa sebelum dan sesudah implementasi adalah sebagai berikut.

a. Skalabilitas

Diukur dengan melihat kemampuan aplikasi untuk menangani peningkatan penambahan fitur baru.

b. Fleksibilitas

Diukur berdasarkan kemudahan dalam memodifikasi atau menambahkan fitur baru tanpa mempengaruhi keseluruhan aplikasi.

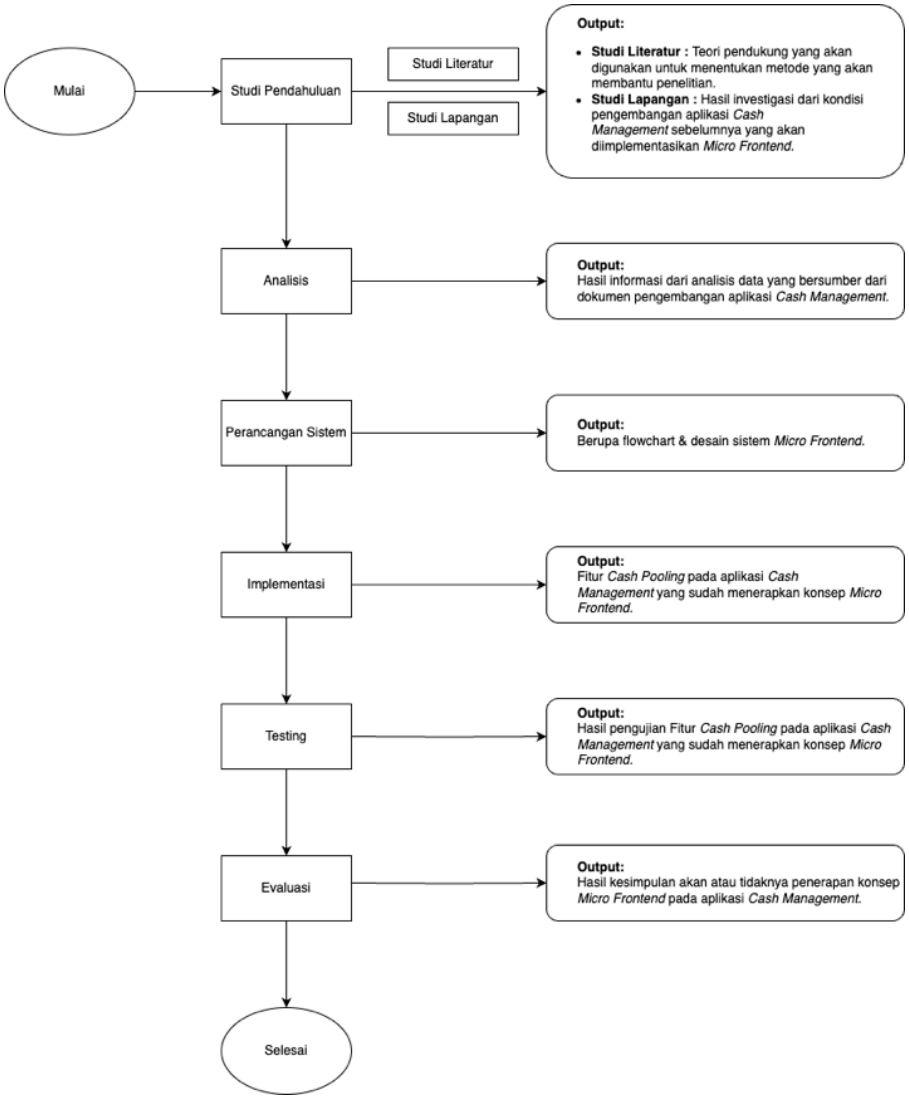
2.1 Metode pengumpulan data, instrumen penelitian, dan metode pengujian

Metode pengumpulan data untuk mendukung implementasi *micro frontend* pada aplikasi *cash management* PT. Bank XYZ dilakukan melalui kuesioner, observasi, analisis dokumen, dan data hasil uji. Kuesioner dirancang untuk diisi oleh pihak-pihak yang relevan, seperti *System Analysis and Design* (SAD), pengembang *front-end*, *squad leader*, dan *Quality Assurance* (QA). Tujuan kuesioner ini adalah untuk memahami sistem yang sedang berjalan serta mengidentifikasi potensi dampak yang dihasilkan setelah implementasi *micro frontend*. Melalui informasi ini, implementasi dapat lebih terarah sesuai kebutuhan organisasi. Observasi dilakukan oleh peneliti yang juga merupakan bagian dari tim pengembangan di PT. Bank XYZ. Peneliti memanfaatkan pengamatan langsung untuk memahami arsitektur, kerangka kerja (*framework*), serta kebiasaan tim dalam mengembangkan *cash management application*. Informasi dari observasi ini memberikan wawasan kontekstual tentang kondisi yang mendukung atau menjadi tantangan dalam penerapan *micro frontend*.

Selain itu, analisis dokumen dilakukan untuk menggali lebih dalam kebutuhan teknis dan bisnis yang mendasari pengembangan aplikasi. Dokumen seperti *Business Requirements Document (BRD)*, *Business Model Canvas (BMC)*, dan desain sistem menjadi referensi utama dalam proses ini. Analisis dokumen membantu menyesuaikan implementasi *micro frontend* dengan kebutuhan strategis perusahaan. Tidak kalah penting, data hasil uji yang diperoleh pada tahap implementasi digunakan untuk menganalisis kelayakan serta memvalidasi hipotesis penelitian. Data ini memberikan dasar yang kuat untuk mengevaluasi keberhasilan penerapan *micro frontend*, sehingga keputusan lebih terukur dapat diambil.

Pengujian dilakukan untuk memastikan keberhasilan dan kelayakan implementasi *micro frontend* dari berbagai aspek. Dalam *blackbox testing*, QA bertanggung jawab untuk menguji kesesuaian fungsionalitas, seperti tombol, formulir, dan *modal*, berdasarkan skenario pengujian yang telah dirancang. QA juga memastikan bahwa desain yang dimigrasikan dari arsitektur monolitik tetap konsisten tanpa mengalami perubahan atau cacat visual [4]. Selanjutnya, *User Acceptance Test (UAT)* dilakukan untuk memastikan bahwa implementasi sesuai dengan kebutuhan pengguna dan *system requirement*, sehingga aplikasi dapat diterima dengan baik oleh pengguna [5]. Sebagai langkah perlindungan tambahan, dilakukan *penetration testing* oleh *pentester* untuk mengidentifikasi kerentanan dalam sistem serta memastikan keamanan implementasi dari ancaman eksternal [6]. Dokumentasi hasil pengujian ini menjadi landasan penting untuk mengevaluasi kesuksesan akhir implementasi *micro frontend* secara menyeluruh.

2.2 Tahapan penelitian



Gambar 1. Tahapan Penelitian

Berikut ini adalah penjelasan tahapan penelitian yang ada pada Gambar 1.

a. Studi Pendahuluan

Studi pendahuluan mempunyai dua bagian utama Studi literatur dan Studi lapangan. Pada Studi Literatur peneliti mengumpulkan dan mempelajari berbagai sumber referensi terkait *micro frontend*, *module federation*, dan Next.js. Pada Studi Lapangan Pengamatan dan pengumpulan data langsung dari lapangan atau industri yang telah menerapkan konsep serupa.

b. Analisis

Berdasarkan hasil studi pendahuluan, peneliti melakukan analisis mendalam terhadap kebutuhan sistem, tantangan yang mungkin dihadapi, dan solusi potensial. Tahap ini menghasilkan penilaian kelayakan yang akan menjadi dasar untuk tahap perancangan sistem.

c. Perancangan Sistem

Pada tahap ini, peneliti merancang arsitektur *micro frontend* menggunakan *module federation* dan Next.js berdasarkan

hasil analisis. Perancangan mencakup struktur aplikasi, pembagian modul, dan integrasi antar komponen.

d. Implementasi

Berdasarkan rancangan yang telah dibuat, peneliti mulai mengimplementasikan sistem. Ini melibatkan pengodean, konfigurasi *module federation*, dan pengaturan Next.js untuk mendukung arsitektur *micro frontend*. Hasil dari tahap ini adalah *prototype* sistem yang siap untuk diuji.

e. Testing

Prototype yang telah diimplementasikan kemudian melalui serangkaian pengujian untuk memastikan fungsionalitas, performa, dan integrasi antar modul berjalan dengan baik. Hasil pengujian didokumentasikan dalam laporan hasil testing.

f. Evaluasi

Berdasarkan hasil testing, peneliti melakukan evaluasi menyeluruh terhadap sistem. Jika ditemukan kekurangan atau area yang perlu diperbaiki, proses akan kembali ke tahap implementasi untuk penyempurnaan. Tahap ini menghasilkan laporan evaluasi yang mencakup temuan dan rekomendasi.

g. Sosialisasi Pengembangan

Setelah sistem dievaluasi dan dianggap memenuhi kriteria, peneliti menyusun Dokumen Pengembangan yang komprehensif. Dokumen ini mencakup panduan implementasi, *best practices*, dan *lessons learned* selama proses penelitian. Tahap ini bertujuan untuk memudahkan adopsi dan pengembangan lebih lanjut dari hasil penelitian.

3. HASIL DAN PEMBAHASAN

3.1 Penilaian kelayakan

Penilaian kelayakan ini dibutuhkan untuk menguji apakah implementasi *micro frontend* ini seharusnya diterapkan atau tidak. Penilaian kelayakan ini dilakukan terhadap 2 hal yaitu kelayakan Teknik dan kelayakan operasional.

a) Kelayakan Teknik

Kelayakan Teknik di sini adalah kelayakan pada teknologi yang dipakai, apakah teknologi ini sudah *mature* untuk digunakan atau belum, dengan dilihat dari perusahaan atau *user* yang menggunakan teknologi ini untuk pengimplementasian *micro frontend*. Jika kita lihat pada situs resmi masing-masing teknologi yang dipakai untuk implementasi ini, seperti pada situs resmi *Next.js* [7] dan pada situs resmi *module federation* [8] banyak perusahaan besar yang menggunakan teknologi ini, jadi sudah bisa dipastikan bahwa teknologi yang dipakai sangat layak untuk membantu implementasi *micro frontend ini.*

b) Kelayakan Operasi

Kelayakan operasi ditinjau dari beberapa hal untuk memastikan sebelum atau setelah implementasi *micro frontend* dan tentunya proses *development* dan juga *existing* sistem yang berjalan tidak ada masalah, detail nya sebagai berikut.

c) Proses *Development*

Sesuai dengan tujuan dari implementasi *micro frontend* yaitu untuk menambah skalabilitas dan fleksibilitas, maka proses *development* harus dilihat apakah ada masalah atau tidak pada saat prosesnya berjalan. Harapannya dengan implementasi *micro frontend* ini proses *development* bisa lebih lancar sesuai dengan tujuan dari implementasi *micro frontend*.

d) *Existing* Sistem Sebelum atau Sesudah Implementasi

Dikarenakan pada saat implementasi ini *existing* sistem masih berjalan, harapannya implementasi *micro frontend* ini dilakukan secara parsial, jadi implementasinya tidak mengganggu *existing* sistem yang berjalan.

3.2 Spesifikasi kebutuhan

Sebelum implementasi dilakukan ada beberapa hal terlebih dahulu yang perlu diketahui, diantaranya adalah sebagai berikut.

a) Informasi mengenai *existing* sistem yang berjalan, dimulai dari teknologi yang dipakai dan infrastruktur yang dipakai.

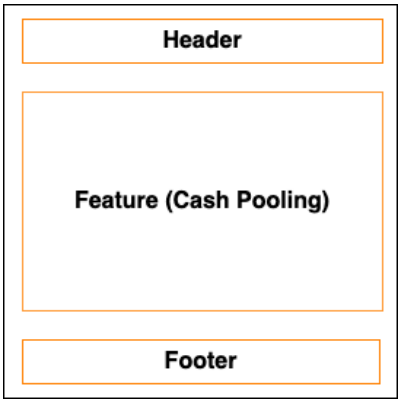
b) Kebiasaan pengembang *front-end* pada saat sebelum pengimplementasian *micro frontend* dimulai dari *git strategy*, penggunaan *component user interface*, dll.

c) *User* dari *cash management application* adalah korporasi besar dan para nasabah *wholesale.*

3.3 Perancangan sistem

Tahapan ini menjelaskan tentang desain sistem, kegiatan pada desain sistem ini menghasilkan *flowchart* fitur *cash pooling* dan desain sistem untuk *micro frontend.*

a. *Layout micro frontend*



Gambar 2. *Layout Micro Frontend*

Gambar 2 merupakan *layout micro frontend* pada *cash management application* fitur *cash pooling*. *Layout* di atas sudah menggambarkan implementasi *micro frontend* pada *cash management application* dibagi menjadi 3 komponen utama yaitu *Header, Feature, dan Footer* itu merupakan *layout* standart pada aplikasi berbasis web akan tetapi yang membedakan pada *micro frontend* adalah *independency* & integration dikarenakan setiap komponen pada pada *layout* itu sudah berbeda *source code, repository*, dan *purpose*.

Pada dasarnya *layout* di atas dibuat bukan khusus untuk fitur *cash pooling* karena tujuan utama penerapan *micro frontend* adalah memisah semua fitur menjadi bagian terkecil, independent, dan mempermudah proses *deployment* aplikasi. Namun dikarenakan pada penelitian ini mempunyai batasan masalah maka batasan tersebut hanya pada fitur *cash pooling* saja. Pembagian komponen berdasar *repository* dan *team* adalah sebagai berikut pada tabel 1.

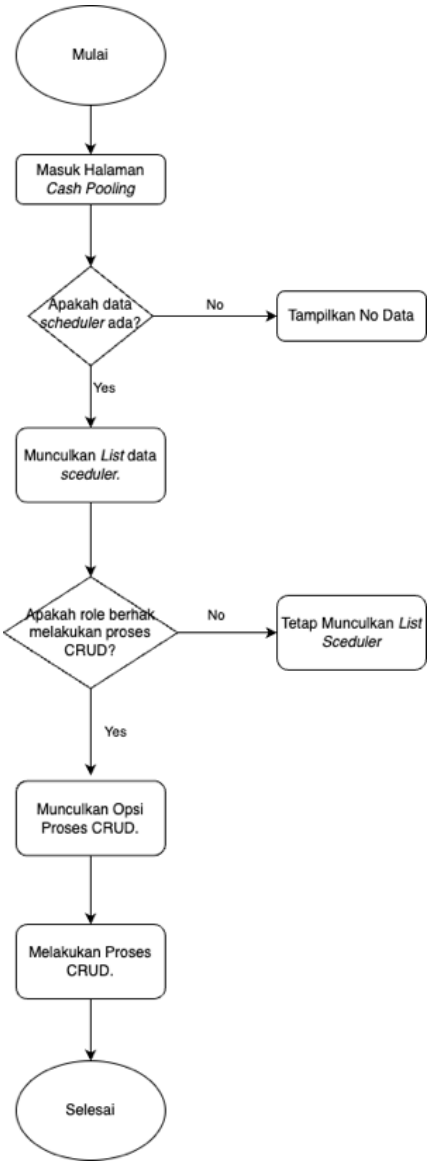
Tabel 1. *Repository & Component*

No	*Repository*	*Component*
1	*ui-container*	*All View Port*
2	*ui-header-footer*	*Header & Footer*
3	*ui-cash-pooling*	*Cash pooling*

Terdapat 3 *repository*/komponen yang terbuka ketika fitur *cash pooling* diakses oleh *browser*. Ketika komponen itu digabung menjadi satu oleh *framework* Next.js & *module federation* yang menghasilkan tampilan yang sesuai.

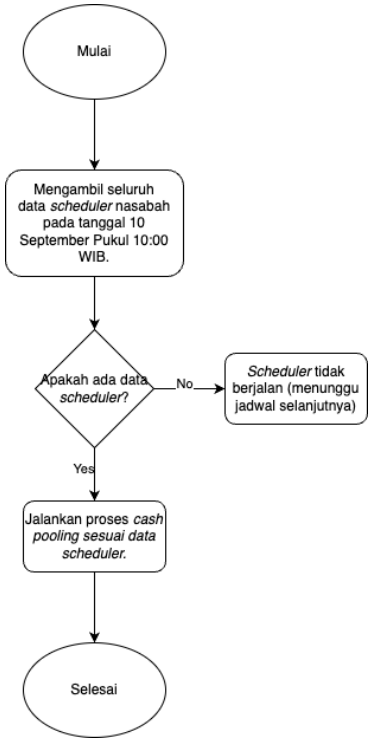
Ketika komponen itu sudah dipisah *team* mana saja yang akan mengembangkan komponen tersebut sudah tidak perlu risau lagi dengan adanya tabrakan antar *team*, dikarenakan sudah bisa dipastikan komponen itu hanya bisa diubah oleh *team* yang bersangkutan.

b. *Flowchart cash pooling*



Gambar 3. *Flowchart Cash Pooling*

Gambar 3 merupakan *flowchart* dari fitur *cash pooling* yang mana fitur *cash pooling* ini bisa diakses oleh semua nasabah pengguna *cash management application.*

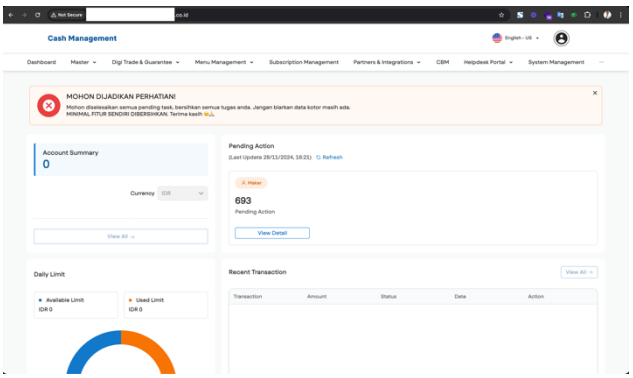


Gambar 4. *Flowchart Scheduler Cash Pooling*

Scheduler pada Gambar 4 merupakan kunci pada fitur *cash pooling* ini, karena sesuai dengan desain sistem *cash pooling* ini dana nasabah yang ada pada rekening anak atau rekening induk akan dialihkan sesuai tujuan sesuai dengan *scheduler* yang berjalan. Proses *scheduler* ini berjalan pada *backend* aplikasi jadi secara pengerjaan ini bukanlah termasuk *capability*, peneliti hanya memastikan inputan dari *form* yang sudah diimplementasikannya *micro frontend* sudah sesuai *request* nya kepada *backend* yang akan dijalankan oleh *scheduler*.

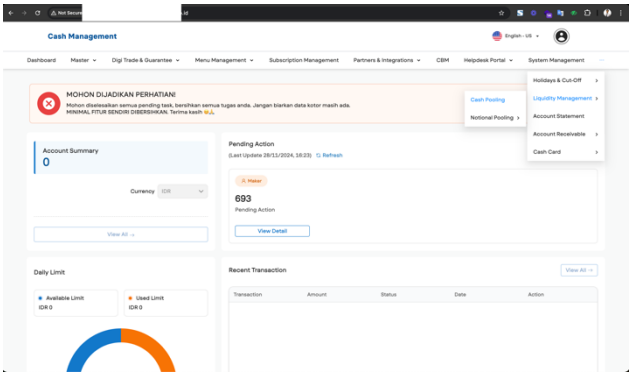
3.4 Implementasi

Tahapan ini hanya akan menampilkan antarmuka fitur *cash pooling* dari versi *monolithic* ke versi *micro frontend*, berikut adalah implementasinya.



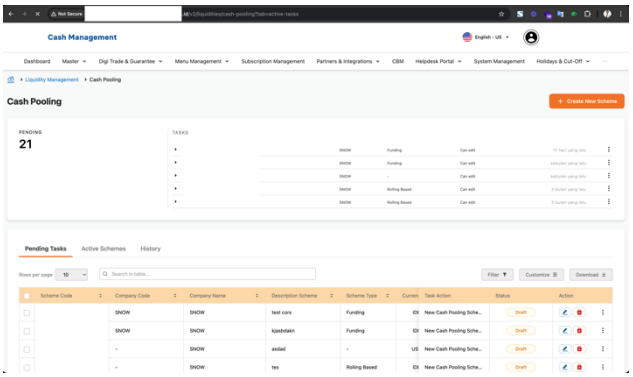
Gambar 5. *Dashboard Cash Management Application*

Pada Gambar 5 merupakan halaman *dashboard cash management application* jika kita lihat terdapat banyak menu yang bisa diakses sesuai *role* yang terdapat pada *user*.



Gambar 6. Menu *Cash Pooling*

Menu *cash pooling* pada gambar 6 di atas terdapat pada group *liquidity management* yang mana ketika *cursor hover* akan muncul *list menu cash pooling*.



Gambar 7. Halaman *Cash Pooling*

Setelah menu diklik maka *browser* akan mengarahkan ke halaman *cash pooling* pada gambar 7 di atas yang mana *cash pooling* sudah menggunakan versi *micro frontend*. *Identifier* versi *monolithic* atau *micro frontend* bisa dilihat pada *route* setelah *domain* utamanya dengan detail berikut.

- Versi *monolithic* setelah domain utama akan mengarah pada *route* sesuai nama fiturnya, semisal https://cash-management.co.id/cash-pooling.
- Versi *micro frontend* setelah *domain* utama mempunyai *identifier* v2 yang mana *indentifier* ini menjadi pembeda antara versi *monolithic* dengan versi *micro frontend*, contohnya https://cash-management.co.id/v2/cash-pooling.

Untuk mengatur segmentasi antara *monolithic* dan *micro frontend* itu sendiri peneliti menggunakan *reverse proxy* yang mana itu merupakan fitur dari *web server*.

Sesuai dengan layout *micro frontend* yang sudah ditentukan, bagian *header*, *footer*, dan *menu* pada versi *micro frontend* merupakan bagian Next.js dan *module federation* tersendiri dan di-*deploy* pada *web server* sendiri begitu juga dengan

bagian *content*/fitur *cash pooling*. Kedua Next.js dan *module federation* tadi disatukan ke dalam *container* yang membungkus *header, footer*, dan fitur *cash pooling* nya.

3.5 Evaluasi

Pada tahap evaluasi akan dilakukan pengujian menggunakan *blackbox testing, user acceptance testing*, dan kuesioner.

3.5.1 *Blackbox testing*

Tabel 2. Hasil *Blackbox Testing*

No	Pengujian	Ekspektasi	Hasil
1	Perpindahan dari domain *monolithic* kepada domain *micro frontend*, ataupun sebaliknya.	Ketika terjadi perpindahan dari *monolithic* ke *micro frontend* melalui *bypass* URL berjalan dengan baik.	Berhasil
2	Navigasi menu pada *header micro frontend*.	Navigasi menggunakan menu pada halaman *monolithic* berjalan dengan baik.	Berhasil
3	Navigasi menuju halaman utama *cash pooling.*	Navigasi menuju halaman utama *cash pooling* menggunakan menu pada halaman *monolithic* berjalan dengan baik.	Berhasil
4	Navigasi menuju halaman *create cash pooling.*	Navigasi menggunakan tombol *create* halaman *cash pooling* berjalan dengan baik.	Berhasil
5	Navigasi menuju halaman edit *cash pooling.*	Navigasi menggunakan tombol *action edit* pada tabel *cash pooling* berjalan dengan baik.	Berhasil
6	Navigasi menuju halaman view *cash pooling*.	Navigasi menggunakan *scheme code* pada tabel *cash pooling* berjalan dengan baik.	Berhasil
7	*Logout* ketika berada pada halaman *cash pooling.*	Melakukan proses *logout* dari halaman *cash pooling micro frontend* berjalan dengan baik.	Berhasil
8	*Bypass* menu melalui url atau kembali ke *cash pooling monolithic.*	*Bypass* URL menuju halaman *cash pooling monolithic* tidak diperbolehkan.	Gagal

Rumus untuk menghitung keberhasilan dan kegagalan *blackbox testing*.

Persentase = (Jumlah sukses atau gagal / Jumlah *scenario*) x 100

Data yang didapat dari tabel 2 *blackbox* testing sebagai berikut.

- Jumlah Skenario: 8
- Skenario Berhasil: 7
- Skenario Gagal: 1

Persentase keberhasilan = (7 / 8) x 100 = 87.55%

Persentase kegagalan = (1 / 8) x 100 = 12.5%

Dengan data yang didapat menandakan bahwa sistem sudah cukup stabil dengan 87.5% keberhasilan, dan 12.5%. Dengan *ratio* 7:1 maka dapat disimpulkan implementasi ini layak dilanjutkan

3.5.2 *User acceptance testing*

Tabel 3. Hasil *User Acceptance Testing*

No	Pengujian	Hasil
1	Perpindahan dari domain *monolithic* kepada domain *micro frontend*, ataupun sebaliknya.	Sesuai
2	Navigasi menu pada *header micro frontend*.	Sesuai
3	Navigasi menuju halaman utama *cash pooling.*	Sesuai
4	Navigasi menuju halaman *create cash pooling.*	Sesuai
5	Navigasi menuju halaman edit *cash pooling.*	Sesuai
6	Navigasi menuju halaman view *cash pooling.*	Sesuai
7	*Logout* ketika berada pada halaman *cash pooling.*	Sesuai
8	Tidak adanya dampak *negative* dari implementasi *micro frontend*.	Sesuai

Hasil dari *user acceptance testing* pada Tabel 3. Menghasilkan kesesuaian yang telah divalidasi oleh *user* secara langsung.

3.5.3 Kuesioner

Metode pengambilan data pada penelitian kali ini salah satunya menggunakan kuesioner, sudah dilakukan penyebaran kuesioner menggunakan *google form*, menggunakan skala likert untuk opsi pilihannya dikarenakan skala likert sangat cocok untuk menentukan kualitas kepuasan pengguna [9], dengan detail seperti berikut :

- Total Responden 11 dengan detail 3 SAD *(System Analysis and Design)*, 1 *Squad Leader*, 3 *Front-end*, dan 4 QA.
- Analisis data kuesioner ini dihitung dengan skala likert yang memiliki kriteria sebagai berikut.
  - 1 = Sangat Tidak Setuju (STS)
  - 2 = Tidak Setuju (TS)

- 3 = Netral (N)
- 4 = Setuju (S)
- 5 = Sangat Setuju (SS)

- Interpretasi data untuk menentukan rata-rata skor setiap penilaian, sebagai berikut.
  - 1.0–1.8 = Sangat Tidak Setuju
  - 1.9–2.6 = Tidak Setuju
  - 2.7–3.4 = Netral
  - 3.5–4.2 = Setuju
  - 4.3–5.0 = Sangat Setuju
- Rumus rata-rata menghitung skor sebagai berikut.

  Rata-rata skor = (Jumlah skor / Jumlah responden)
- Terdapat 15 Pertanyaan, dibagi menjadi 3 bagian, setiap bagian mempunyai 5 pertanyaan, dengan detail sebagai berikut pada tabel 4.

Tabel 4. Bagian 1

No	Pertanyaan	STS *1	TS *2	N *3	S *4	SS *5	Rata-rata
1	Implementasi arsitektur *micro frontend* mempermudah proses pengembangan fitur dibandingkan dengan arsitektur *monolithic*.			3		50	4.82
2	Arsitektur *micro frontend* mampu mengurangi beban kerja tim pengembang pada fitur *cash pooling*.			3	20	25	4.37
3	Proses integrasi antara modul pada arsitektur *micro frontend* lebih efisien dibandingkan arsitektur *monolithic*.				12	40	4.73
4	Implementasi *micro frontend* memberikan fleksibilitas lebih tinggi dalam pengembangan fitur baru.				4	50	4.91
5	Penerapan arsitektur *micro frontend* meningkatkan kecepatan *deployment* fitur dibandingkan arsitektur *monolithic*.				8	45	4.82

Rata-rata bagian 1 = (23.65 / 5) = 4.73. Jika diinterpretasikan pada acuan maka 4.73 adalah Sangat Setuju.

Tabel 5. Bagian 2

No	Pertanyaan	STS *1	TS *2	N *3	S *4	SS *5	Rata-rata
1	Implementasi arsitektur *micro frontend* meningkatkan efisiensi proses pengembangan secara keseluruhan.				20	30	4.55
2	Arsitektur ini memungkinkan fitur *cash pooling* untuk lebih mudah diskalakan sesuai kebutuhan bisnis.			6	12	30	4.36
3	Perubahan atau pembaruan pada fitur *cash pooling* menjadi lebih fleksibel tanpa memengaruhi fitur lain.				20	30	4.55
4	Penerapan arsitektur ini mengurangi waktu pemeliharaan aplikasi secara keseluruhan.			6	8	35	4.55
5	Kolaborasi antar tim pengembang lebih terorganisir setelah implementasi arsitektur *micro frontend*.			3	16	30	4.45

Hasil pada tabel 5 terdapat rata-rata bagian 2 = (22.45 / 5) = 4.49. Jika diinterpretasikan pada acuan maka 4.49 adalah Sangat Setuju.

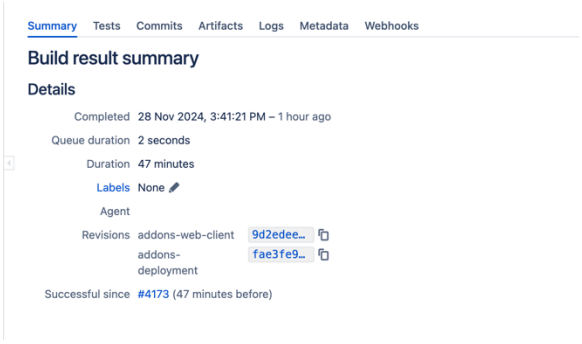
Tabel 6. Bagian 3

No	Pertanyaan	STS *1	TS *2	N *3	S *4	SS *5	Rata-rata
1	Proses migrasi dari arsitektur *monolithic* ke *micro frontend* berjalan sesuai dengan harapan.		2	3	12	30	4.27
2	Kendala teknis yang dihadapi selama migrasi dapat diselesaikan dengan strategi yang tepat.			3	12	35	4.55
3	Arsitektur *micro frontend* mempermudah pengujian dan deteksi kesalahan pada fitur *cash pooling*.			6	16	25	4.27
4	Pelatihan teknis untuk tim terkait arsitektur baru sudah memadai.	1		12	8	20	3.64
5	Dokumentasi implementasi arsitektur baru sudah cukup untuk mendukung pengembangan ke depan.		2	9	16	15	3.82

Hasil pada tabel 6 terdapat rata-rata bagian 3 = (20.55 / 5) = 4.11. Jika diinterpretasikan pada acuan maka 4.11 adalah Setuju.

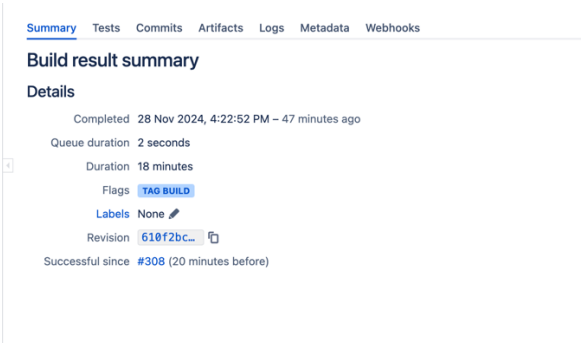
Rata-rata dari bagian 1-3 jika dihitung dengan rumus yang sama rata-rata semua bagian = (13.33 / 3) = 4.44 yang mana jika diinterpretasikan menjadi Sangat Setuju. Hasil dari kuesioner ini membuktikan bahwa implementasi *micro frontend* ini layak dilakukan dengan penyempurnaan di beberapa bagian, terutama bagian yang mempunyai skor rata-rata terkecil.

3.5.4 *Build time*



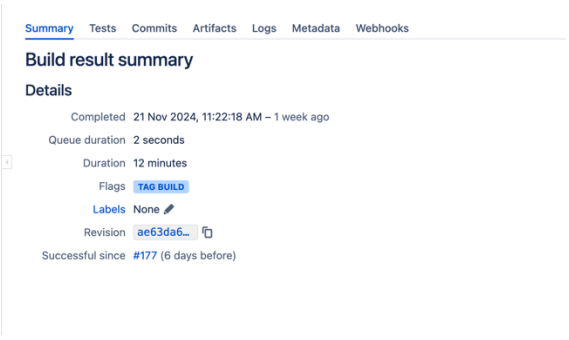
Gambar 8. *Build Time Monolithic*

47 menit adalah *build time* pada aplikasi *monolithic cash managemen application*, dan dapat dilihat pada gambar 8.



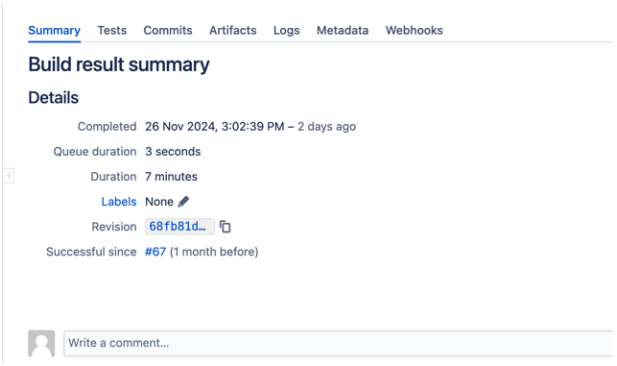
Gambar 9. *Build Time ui-container*

Pada gambar 9 ini merupakan *build time ui-container*, hanya 18 menit.



Gambar 10. *Build Time ui-header-footer*

Pada gambar 10 ini merupakan *build time ui-header-footer*, hanya 12 menit.

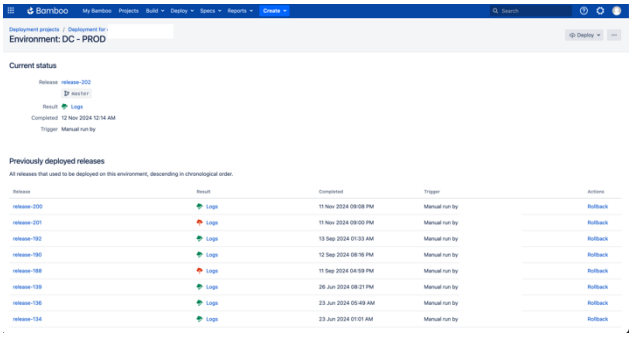


Gambar 11. *Build Time ui-cash-pooling*

Pada gambar 11 ini merupakan *build time ui-cash-pooling*, hanya 7 menit. Karena pada CI/CD terdapat beberapa *agent* untuk proses *build*, walaupun 3 *pipeline* itu ditotal menghasilkan waktu 37 menit, tetapi tetap lebih cepat proses *build micro frontend*. Pada kenyataannya proses *build micro frontend* itu dilakukan secara paralel oleh CI/CD itu sendiri [10], jadi waktu *build* untuk ke 3 *pipeline* itu menjadi lebih cepat rata-rata 12 menit per-*build*.

3.5.5 *Deployment* frekuensi

Untuk menilai efisiensi, skalabilitas, dan fleksibilitas maka peneliti melihat berdasar *history deployment* CI/CD, dengan melihat pada gambar 12 terdapat *history deployment* maka bisa diambil penilaian untuk mengukur 3 aspek yang menunjukkan keberhasilan implementasi ini.



Gambar 12. *Deployment* Frekuensi

a. Efisiensi

Deployment terakhir dilakukan pada 12 November 2024 pukul 12:14 AM, hanya beberapa jam setelah *deployment* sebelumnya pada 11 November 2024 pukul 09:08 PM, menunjukkan bahwa proses *release* fitur menjadi lebih cepat dibandingkan sebelumnya.

b. Fleksibilitas

Proses *enhancement* dilakukan dalam satu *repository* tanpa menyebabkan masalah pada fitur lainnya, yang mengindikasikan bahwa implementasi ini mendukung pengembangan yang lebih terintegrasi dan bebas konflik.

c. Skalabilitas

Riwayat *deployment* pada bulan-bulan sebelumnya menunjukkan selisih antara tanggal yang tidak terlalu jauh, menandakan bahwa implementasi *micro frontend* pada fitur *cash pooling* memungkinkan pengembangan yang lebih *scalable*.

4. KESIMPULAN

Berdasarkan hasil analisis, implementasi, dan pengujian pada fitur *cash pooling* dalam *cash management application* PT. Bank XYZ, penerapan arsitektur *micro frontend* terbukti berhasil mengatasi keterbatasan arsitektur *monolithic*. Waktu *build* yang sebelumnya sekitar 1 jam dapat dipersingkat menjadi rata-rata 37 menit melalui proses paralel dengan durasi 12 menit per *build*. Pengelolaan fitur dalam *repository* terpisah juga mengurangi konflik integrasi antar fitur, membuat pengembangan lebih independen. Implementasi ini memberikan dampak positif pada efisiensi, skalabilitas, dan fleksibilitas pengembangan. Proses *release* menjadi lebih cepat, pengembangan fitur baru dapat dilakukan tanpa mengganggu fitur lain, dan tim pengembang dapat fokus pada fitur masing-masing.

Tantangan teknis, seperti kesenjangan pengetahuan terkait *micro frontend*, dapat diatasi melalui sosialisasi dan dokumentasi yang lengkap. Sementara itu, tantangan operasional berupa potensi keluhan nasabah tidak ditemukan hingga implementasi selesai. Pengujian berlapis, mulai dari *Blackbox testing, User Acceptance Test, hingga Penetration Test*, menunjukkan bahwa arsitektur ini layak diterapkan. Dukungan data kuesioner dari 11 responden dengan skor rata-rata 4.44 (interpretasi sangat setuju) semakin memperkuat kesimpulan bahwa implementasi ini efektif dan dapat dilanjutkan.

Keberhasilan ini membuka peluang untuk menerapkan arsitektur *micro frontend* pada fitur lain, khususnya fitur yang kompleks atau sering diperbarui. Pemerataan pengetahuan di antara pengembang perlu ditingkatkan melalui pelatihan berkala dan pembentukan komunitas internal untuk berbagi pengalaman. Standar dan pedoman pengembangan *micro frontend* harus dikembangkan dan didukung dengan dokumentasi yang selalu diperbarui. Optimalisasi *pipeline* CI/CD juga perlu dilakukan, dengan fokus pada otomatisasi proses pengujian dan *deployment*. Selain itu, penerapan sistem *monitoring real-time* yang komprehensif akan memastikan performa aplikasi tetap optimal dan membantu mencegah masalah sebelum berdampak pada pengguna.

DAFTAR PUSTAKA

[1] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, “From monolithic to microservices: An experience report from the banking domain,” *IEEE Softw*, vol. 35, no. 3, pp. 50–55, 2018.

[2] Y. Prajwal, J. V. Parekh, and R. Shettar, “A brief review of micro-frontends,” *United International Journal for Research and Technology*, vol. 2, no. 8, 2021.

[3] S. University, “Penelitian Eksperimen, Tujuan, Karakteristik Hingga Macamnya.” Accessed: Oct. 08, 2024. [Online]. Available: https://www.sampoernauniversity.ac.id/id/penelitian-eksperimen/

[4] N. M. D. Febriyanti, A. A. K. O. Sudana, and I. N. Piarsa, “Implementasi Black Box Testing pada Sistem Informasi Manajemen Dosen,” *Jurnal Ilmiah Teknologi Dan Komputer*, vol. 2, no. 3, pp. 535–544, 2021.

[5] J. P. da Silva and S. Borges, “Live Acceptance Testing using Behavior Driven Development,” 2020.

[6] M. C. Ghanem and T. M. Chen, “Reinforcement learning for efficient network penetration testing,” *Information*, vol. 11, no. 1, p. 6, 2019.

[7] Vercel, “About React and Next.js.” Accessed: Oct. 05, 2024. [Online]. Available: https://nextjs.org/learn/react-foundations/what-is-react-and-nextjs

[8] Webpack, “Module Federation.” Accessed: Oct. 05, 2024. [Online]. Available: https://webpack.js.org/concepts/module-federation/

[9] M. Awaludin, H. Mantik, and F. Fadillah, “Penerapan metode servqual pada skala likert untuk mendapatkan kualitas pelayanan kepuasan pelanggan,” *JSI (Jurnal sistem Informasi) Universitas Suryadarma*, vol. 10, no. 1, pp. 89–106, 2023.

[10] O. Nikulina and K. Khatsko, “Method of converting the monolithic architecture of a Front-End application to microfrontends,” *Bulletin of National Technical University “KhPI”. Series: System Analysis, Control and Information Technologies*, vol. 2, no. 10, pp. 79–84, 2023.