MODEL MIGRASI MICROSERVICES SPRING BOOT DENGAN SHARED DATABASE PADA SISTEM KEUANGAN E-GOVERNMENT

Roberto[1], Abdul Hamid Arribathi[2*]
[1,2] Magister Teknik Informatika, Universitas Raharja
roberto@raharja.info[1], abdulhamid@raharja.info[2]

ABSTRACT *- e-Government applications in many agencies still rely on monolithic web architectures which are quite difficult in the process of maintaining systems and quickly adapting to new needs. This study proposes a migration model on the Glue Framework-based e-government financial system (Struts, iBATIS, and JSP) towards a Spring Boot-based microservices architecture. In contrast to the general microservices migration model approach that places a distributed database on each service, the financial system migration model maintains the existing database schema to reduce the risk of migration and allow for co-existence phases, i.e. legacy applications and migrated applications to run simultaneously using the same data source during the system migration phase. This research uses the Design Science Research (DSR) approach with a case study on the e-Government financial system in BP Batam. As for the procedure, the migration model is formulated into six stages: legacy system assessment, service boundary determination, microservice extraction, shared database access governance, interface upgrade, and integration and validation. The model is validated through a case study of the financial system on the e-Government system, by documenting artifacts, decision points, and mitigation strategies against legacy constraints. The contribution of this research is in the form of a migration model that can be replicated to help modernize other e-Government systems that require high data integrity and service sustainability.*

Keywords: e-Government; legacy system migration; microservices; Great Spring; The Financial System.

Abstrak - e-Government di banyak instansi masih bergantung pada arsitektur web monolitik yang cukup menyulitkan dalam proses pemeliharaan sistem dan proses penyesuaian cepat dengan kebutuhan baru. Penelitian ini mengusulkan model migrasi pada sistem keuangan e-government berbasis Glue Framework (Struts, iBATIS, dan JSP) menuju arsitektur microservices berbasis Spring Boot. Dimana sedikit berbeda dengan pendekatan model migrasi microservices secara umum yang menempatkan database secara terdistribusi pada setiap service, maka pada model migrasi sistem keuangan ini mempertahankan skema basis data eksisting untuk menurunkan risiko migrasi serta memungkinkan fase ko-eksistensi, yaitu aplikasi lama dan aplikasi hasil migrasi berjalan bersamaan menggunakan sumber data yang sama selama fase migrasi sistem. Penelitian ini menggunakan pendekatan Design Science Research (DSR) dengan studi kasus pada sistem keuangan e-Government di lingkungan BP Batam. Adapun prosedurnya, model migrasi diformulasikan ke dalam enam tahap: asesmen sistem legacy, penentuan batas layanan, ekstraksi microservice, tata kelola akses basis data bersama, pemutakhiran antarmuka, serta integrasi dan validasi. Model divalidasi melalui studi kasus sistem keuangan pada sistem e-Government, dengan mendokumentasikan artefak, titik keputusan, dan strategi mitigasi terhadap kendala legacy. Kontribusi penelitian ini berupa model migrasi yang dapat direplikasi untuk membantu modernisasi sistem e-Government lain yang memerlukan integritas data tinggi dan keberlanjutan layanan.

Kata Kunci: *e-Government; migrasi sistem legacy; microservices; Spring Boot; sistem keuangan.*

PENDAHULUAN

Sistem e-Government pada sektor publik umumnya berkembang secara bertahap dalam jangka waktu yang panjang. Kondisi tersebut menyebabkan banyak sistem utama masih menggunakan teknologi legacy yang telah stabil dalam penggunaannya, namun demikian dikarenakan teknologi yang cukup lama atau usang maka semakin sulit dipelihara ketika ada kebutuhan untuk integrasi dengan sistem lain, isu keamanan, serta frekuensi perubahan kebijakan yang meningkat dan butuh penyesuaian secara cepat. Pada konteks penelitian ini, sistem keuangan (finance system) merupakan layanan kritikal yang menangani proses-proses seperti pencatatan pendapatan, pencatatan transaksi penerimaan, pembayaran, dan pelaporan. Kompleksitas proses bisnis serta sensitivitas data menuntut strategi modernisasi yang minim risiko dan tidak mengganggu layanan, sebagaimana ditekankan oleh Auer dkk. (2021) bahwa keputusan migrasi ke microservices harus berbasis

metrik objektif untuk menghindari kegagalan layanan.

Sistem keuangan legacy yang menjadi fokus penelitian dibangun menggunakan Glue Framework versi 3, ini merupakan sebuah produk buatan perusahaan POSCO ICT atau sekarang dikenal POSCO DX yang berasal dari Korea Selatan. Adapun versi 3.x Glue Framework ini telah rilis antara tahun 2008 hingga tahun 2012 . Teknologi legacy ini lazim memadukan Struts untuk lapisan presentasi-kontrol, iBATIS sebagai pemetaan akses data berbasis XML, serta JSP untuk antarmuka. Walaupun sistem tersebut dapat dikategorikan sebagai monolit semi-modular karena pemisahan modul logis telah diterapkan pada struktur kode, seluruh aplikasi masih dideploy sebagai satu kesatuan. Konsekuensinya, perubahan pada satu bagian dapat berdampak pada bagian lain, proses rilis menjadi lambat, dan isolasi kegagalan sulit dilakukan.

Arsitektur microservices menawarkan pendekatan pemisahan kapabilitas bisnis menjadi layanan-layanan kecil yang dapat dikembangkan dan dideploy secara mandiri. Namun, adopsi microservices pada organisasi publik kerap terhambat oleh kompleksitas migrasi, terutama pada pengelolaan data dan kebutuhan menjaga kontinuitas layanan. Tantangan data management pada microservices telah banyak dibahas dalam literatur (Laigner dkk., 2021, Söylemez & Tekinerdogan, 2022). Oleh sebab itu, penelitian ini memfokuskan migrasi pada lapisan aplikasi dan mempertahankan basis data eksisting sebagai strategi transisi (shared database) untuk meminimalkan risiko (Bozan dkk., 2021, Martínez Saucedo dkk., 2025).

Tujuan penelitian ini adalah: (1) merumuskan model migrasi modul keuangan dari monolit semi-modular berbasis Glue Framework ke microservices berbasis Spring Boot; (2) mendokumentasikan artefak dan titik keputusan pada setiap tahap migrasi; dan (3) memvalidasi penerapan model melalui studi kasus implementasi di lingkungan e-Government. Kontribusi utama penelitian adalah model migrasi yang bersifat praktis namun tetap sistematis sehingga dapat direplikasi pada modernisasi sistem serupa.

TINJAUAN PUSTAKA

Sistem Legacy Berbasis Glue Framework

Glue Framework dalam konteks penelitian ini merujuk pada arsitektur *Enterprise Java* (J2EE) tradisional yang mengintegrasikan tiga teknologi utama, yaitu Struts sebagai pengatur alur logika (*controller*), iBATIS sebagai lapisan akses data, dan *JavaServer Pages* (JSP) untuk antarmuka pengguna. Struts mengandalkan pola *Model-View-Controller* (MVC) dengan konfigurasi yang berpusat pada file XML, sementara iBATIS memfasilitasi pemetaan *query* SQL secara manual yang juga dikelola melalui XML. Meskipun kombinasi ini memberikan kendali penuh terhadap performa database di masa lalu, arsitektur monolitik ini menciptakan ketergantungan yang sangat kuat antara lapisan logika bisnis dan skema basis data. Kondisi tersebut sering kali memicu kompleksitas konfigurasi yang tinggi, di mana perubahan kecil pada satu bagian sistem memerlukan penyesuaian menyeluruh, sehingga menghambat fleksibilitas pengembangan.

Keterbatasan pada Glue Framework versi 3.x ini, terutama dalam hal *tight coupling* dan sulitnya isolasi modul, menjadi hambatan utama dalam aspek pemeliharaan (*maintainability*) dan evolusi sistem di era modern. Hal ini sejalan dengan kerangka penilaian yang dikemukakan oleh Auer dkk. (2021), yang menyatakan bahwa sistem monolitik dengan tingkat kompleksitas tinggi memerlukan evaluasi mendalam berbasis metrik nyata sebelum diputuskan untuk dilakukan re-arsitektur. Selain itu, kesulitan dalam memisahkan logika yang tertanam dalam file konfigurasi XML dan *query* SQL manual pada iBATIS menjadi tantangan teknis dalam proses dekomposisi layanan. Sebagaimana dijelaskan dalam tinjauan literatur oleh Hassan dkk. (2024), hambatan metodologis dalam memecah sistem monolitik yang saling terkait erat (*intertwined*) memerlukan strategi transisi yang hati-hati untuk memastikan integritas data tetap terjaga selama proses migrasi menuju *microservices.*

Arsitektur Microservices Berbasis Spring Boot

Spring Boot merupakan kerangka kerja Java modern yang mengusung prinsip *convention-over-configuration*, yang secara signifikan mempermudah pengembangan layanan mandiri (*self-contained services*) dengan kebutuhan konfigurasi minimal. Dalam arsitektur *microservices,* Spring Boot memungkinkan setiap layanan untuk memodelkan satu kapabilitas bisnis spesifik yang dapat dideploy secara independen dan berkomunikasi melalui antarmuka API, seperti RESTful web services. Karakteristik otonomi ini, menurut Newman (2021), merupakan kunci utama dalam meningkatkan ketahanan sistem dan kecepatan rilis fitur, karena kegagalan pada satu layanan tidak secara langsung meruntuhkan seluruh ekosistem aplikasi.

Meskipun arsitektur microservices idealnya mengarah pada desentralisasi data, penelitian ini menerapkan pola basis data bersama (*shared database pattern*) sebagai strategi transisi dari sistem *legacy*. Menurut Richardson (2018), pola ini dapat diterima dalam fase migrasi untuk menjaga integritas data dan meminimalkan kompleksitas manajemen transaksi terdistribusi yang sering kali muncul pada sistem monolitik berskala besar. Penggunaan Spring Boot dalam skema ini tetap memberikan keuntungan pada aspek observabilitas dan kemudahan pengelolaan layanan mandiri sebagaimana ditekankan oleh Larsson (2023). Dengan mempertahankan basis data Oracle eksisting, setiap layanan Spring Boot difokuskan pada pemisahan logika bisnis di lapisan aplikasi, sehingga risiko kerusakan data dapat ditekan selama proses modernisasi dari Glue Framework berlangsung.

Migrasi Monolit ke Microservices

Migrasi dari arsitektur monolitik ke microservices umumnya dilakukan secara bertahap (incremental migration) untuk meminimalkan gangguan layanan, khususnya pada sistem kritikal. Strategi yang sering digunakan adalah dekomposisi domain dan pendekatan Strangler/inkremental, yaitu menambahkan layanan baru sambil menjalankan fungsi legacy secara berdampingan hingga dapat dipensiunkan (Bozan dkk., 2021, Martínez Saucedo dkk., 2025).

Setelah domain teridentifikasi, Strangler pattern digunakan untuk mengekstraksi fungsionalitas secara bertahap menjadi

microservices. Tahap identifikasi batas layanan (service boundaries) merupakan aspek krusial dan sering menjadi hambatan karena berpengaruh langsung terhadap kualitas arsitektur hasil migrasi (Oumoussa & Saidi, 2025). Menurut Riyanto dkk. (2023), *Decomposition pattern* berperan dalam membagi aplikasi monolitik ke dalam beberapa domain bisnis yang didasarkan pada kategori layanan utamanya. Setelah domain tersebut teridentifikasi, *Strangler pattern* digunakan untuk memecah domain bisnis tersebut menjadi unit-unit *microservices* yang lebih kecil. Proses ini dilakukan melalui tiga tahapan sistematis, yaitu:

1) Transform: Membangun atau membuat fungsionalitas baru ke dalam layanan *microservices*.
2) Co-exist: Menjalankan layanan baru secara berdampingan dengan fungsi pada sistem monolitik yang masih ada.
3) Eliminate: Menonaktifkan komponen lama setelah layanan baru dipastikan siap dan beroperasi secara optimal.

Penerapan tahapan ini memungkinkan proses migrasi berjalan dengan risiko yang lebih rendah karena layanan lama tetap tersedia selama masa transisi. Selain itu, berdasarkan analisis performa, penggunaan pola dekomposisi dan *strangler* terbukti mampu meningkatkan efisiensi sistem dalam menangani permintaan pengguna dibandingkan saat masih menggunakan arsitektur monolitik yang kaku. Pendekatan ini sangat relevan untuk sistem lingkungan e-Government yang memerlukan keberlanjutan operasional tinggi selama proses transformasi arsitektur berlangsung.

METODE PENELITIAN

Desain Penelitian

Penelitian ini menggunakan pendekatan Design Science Research (DSR) dengan studi kasus pada sistem keuangan e-Government. Sesuai dengan panduan Hevner dkk. (2004), artefak utama yang dihasilkan berupa model migrasi yang dirancang untuk memberikan solusi inovatif terhadap permasalahan organisasi. Proses penelitian mengikuti tahapan Design Science Research Process sebagaimana dijelaskan oleh Peffers dkk. (2007), dengan evaluasi melalui penerapan model pada studi kasus, berfokus pada kelengkapan tahapan, konsistensi artefak, serta keberhasilan validasi fungsional.

Artefak dirumuskan dalam bentuk tahapan migrasi (T1–T6) beserta keluaran/artefak pada setiap tahap, yang diturunkan dari analisis sistem eksisting dan kebutuhan modernisasi. Instrumen evaluasi mencakup checklist kelengkapan artefak per tahap, skenario pengujian fungsional layanan hasil migrasi, serta verifikasi konsistensi akses data pada skema basis data bersama (*shared database*).

Objek Penelitian dan Batasan

Strategi migrasi dari arsitektur monolitik ke *microservices* merupakan proses transformasi yang memerlukan pendekatan terukur guna memitigasi risiko kegagalan sistem, terutama pada sistem keuangan yang sangat kritikal. Menurut Volynsky dkk. (2022), dekomposisi aplikasi monolitik menjadi entitas terdistribusi dapat dilakukan dengan tetap mempertahankan penggunaan basis data bersama (*shared database*). Pendekatan ini bertujuan untuk

mengidentifikasi batasan layanan (*service boundaries*) yang jelas dan melakukan orkestrasi layanan di lingkungan yang siap awan (*cloud-ready*) tanpa harus segera memecah struktur penyimpanan data.

Penerapan strategi ini sangat relevan dengan batasan penelitian ini, di mana fokus utama adalah modernisasi lapisan aplikasi dari *Glue Framework* ke *Spring Boot* dengan tetap mempertahankan skema basis data Oracle eksisting. Volynsky dkk. (2022) menegaskan bahwa model arsitektur dengan basis data bersama ini merupakan langkah esensial untuk menjaga konsistensi data dan menghindari kompleksitas manajemen transaksi terdistribusi yang sangat tinggi. Dengan demikian, layanan *microservices* hasil migrasi dapat beroperasi secara berdampingan (*co-exist*) dengan aplikasi *legacy* pada sumber data yang sama, memastikan integritas data keuangan tetap terjaga sementara fleksibilitas dan skalabilitas pada level aplikasi dapat tercapai.

Pengumpulan Data dan Prosedur

Selanjutnya data dikumpulkan melalui:

- Analisis dokumen dan artefak sistem (struktur modul, konfigurasi, dependensi).
- Observasi proses migrasi dan implementasi layanan Spring Boot.
- Uji fungsional (black-box) berdasarkan skenario bisnis modul keuangan.
- Catatan kendala (constraints) dan keputusan teknis selama migrasi.

Sedangkan prosedur ataupun tahapannya adalah model migrasi yang diusulkan terdiri dari enam tahap utama. Setiap tahap menghasilkan artefak yang dapat digunakan sebagai masukan bagi tahap berikutnya. Rincian tahap dan artefak ditunjukkan pada Tabel 1.

Tahap	Aktivitas Kunci	Artefak	Keluaran
T1. Asesmen sistem legacy	Inventarisasi modul & dependensi; identifikasi titik integrasi; analisis risiko	Daftar modul; peta dependensi; daftar kendala	Kandidat domain/kapabilitas untuk diekstraksi
T2. Definisi batas layanan	Penentuan bounded context; pemetaan use case ke layanan; definisi kontrak API awal	Dokumen service boundary; daftar endpoint awal	Rancangan layanan (service blueprint)
T3. Ekstraksi microservice	Implementasi layanan Spring Boot; pemisahan logika bisnis; pembuatan REST API	Kode layanan; spesifikasi API (OpenAPI opsional)	Layanan inti siap diuji
T4. Tata kelola shared database	Aturan akses tabel; kontrol perubahan skema; mitigasi konflik write; audit	Pedoman akses data; matriks tabel-per-layanan	Ko-eksistensi data yang terkontrol

T5. Pemutakhiran antarmuka	Decoupling UI; JSP digantikan UI berbasis Bootstrap; konsumsi API	Komponen UI; mapping halaman ke endpoint	Front-end modern siap integrasi
T6. Integrasi & validasi	Integrasi dengan sistem legacy; uji fungsional & regresi; rencana cut-over bertahap	Skenario uji; hasil uji; catatan isu	Layanan siap operasional dan terdokumentasi

Tabel 1. Tahapan model migrasi dan artefak yang dihasilkan.

Sedangkan untuk alur tahapan migrasi microservices berbasis pendekatan inkremental (Strangler) seperti terlihat pada Gambar 1.

Alur migrasi bertahap (incremental) dengan fase transisi shared database

Teknik Evaluasi dan Kriteria Keberhasilan

Evaluasi dilakukan untuk memastikan bahwa model migrasi dapat diterapkan pada sistem keuangan e-Government tanpa mengorbankan kontinuitas layanan dan integritas data. Evaluasi difokuskan pada validasi fungsional dan evaluasi struktural arsitektur, dengan memperhatikan tantangan microservices yang sering muncul pada praktik (Zhou dkk., 2023) serta isu data management (Laigner dkk., 2021). Teknik evaluasi yang digunakan meliputi:

- Validasi fungsional (black-box) berbasis skenario proses bisnis, mencakup transaksi, pembayaran, dan pelaporan.
- Pemeriksaan integritas data pada fase ko-eksistensi melalui aturan data ownership (single-writer) dan audit log.
- Evaluasi struktural microservices: otonomi layanan, keterlacakan kontrak API, serta kemampuan deployment independen.
- Telaah kelengkapan artefak model migrasi dan keterlacakan (traceability) dari tahap identifikasi hingga validasi.

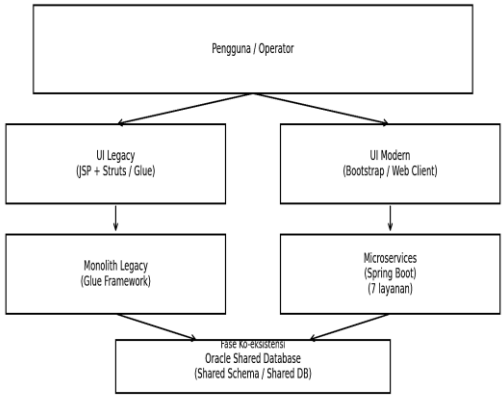
Kriteria keberhasilan minimal meliputi: (1) fungsi utama berjalan sesuai kebutuhan; (2) ko-eksistensi sistem lama dan baru tidak menimbulkan inkonsistensi data pada skenario uji; (3) layanan dapat dideploy independen; dan (4) artefak model migrasi lengkap dan dapat ditelusuri antar tahap.

HASIL DAN PEMBAHASAN

Bagian ini menyajikan hasil penerapan model migrasi pada studi kasus modul keuangan. Penyajian hasil disusun sesuai fokus evaluasi: strategi shared database pada fase ko-eksistensi, pemetaan layanan hasil migrasi, serta evaluasi konseptual melalui validasi fungsional, keterlacakan artefak, tata kelola akses data, dan tinjauan struktural arsitektur.

Strategi Share Database dan Ko-eksistensi

Untuk menurunkan risiko dan menjaga kontinuitas layanan, migrasi dilakukan tanpa memigrasikan skema Oracle eksisting. Pada fase awal, aplikasi legacy dan layanan microservices berjalan berdampingan dan mengakses database yang sama (shared database). Agar pendekatan shared database tetap terkendali, diterapkan tata kelola akses data (data access governance) dan prinsip data ownership (single-writer) untuk menekan konflik operasi tulis, sejalan dengan temuan bahwa pengelolaan data merupakan tantangan utama pada arsitektur microservices (Laigner dkk., 2021; Söylemez & Tekinerdogan, 2022).



Gambar 2. Arsitektur ko-eksistensi sistem legacy dan microservices dengan shared database.

Fase ko-eksistensi digunakan sebagai mekanisme transisi untuk memindahkan fungsi secara bertahap. Sebagian proses keuangan dialihkan ke layanan baru, sementara proses lain masih ditangani sistem legacy hingga layanan microservices stabil. Pendekatan inkremental ini mengurangi risiko cut-over dan memungkinkan validasi bertahap tanpa menghentikan layanan utama (Bozan dkk., 2021).

Implementasi Studi Kasus Sistem Keuangan e-Government

Layanan Hasil Migrasi

Pada studi kasus ini, modul keuangan dipetakan ke sejumlah kapabilitas bisnis, meliputi (1) Otentikasi (authentication), (2) Keuangan (finance), (3) Akuntansi (accounting), (4) Utama (master), (5) Menu Manager, serta (6) Virtual Account (VA). Kapabilitas tersebut kemudian didekomposisi menjadi layanan-layanan microservices yang berjalan pada implementasi, dengan total tujuh layanan sesuai kebutuhan organisasi. Rincian layanan dan tanggung jawab utamanya disajikan pada Tabel 3.

Kandidat Layanan	Tanggung Jawab Utama	Contoh Entitas/Tabel yang Diakses	Catatan
Authentication Service	Untuk menangani proses login, session, logout	Redis	Integrasi ke sistem SSO
Finance Service	Untuk pencatatan transaksi Bukti Kas/ Bank	TBL_BUKTIKAS TBL_BANK	
Invoice Service	Untuk pencatatan Transaksi Invoice/ Tagihan	TBL_INVOICE TBL_PAYMENT TBL_PIUTANG TBL_SURATTAGIHAN	Integrasi ke sistem eksternal (opsional)
Accounting Service	Untuk pencatat	TBL_JV	

	an transaksi akuntansi Jurnal Voucher	TBL_SALDOAWAL	
Master Service	Untuk pencatatan Data-data entitas seperti Pelanggan, Kurs, Daftar Bank, Chart of Account, Data Unit Kerja	TBL_CUSTOMER TBL_BANK TBL_KURS TBL_COA TBL_DEPARTEMEN	
Menu Manager Service	Untuk pengelolaan menu aplikasi dan otorisasi menu	TBL_MENU TBL_AUTH TBL_USER	Integrasi dengan sistem Otorisasi Terpusat
VA Service	Untuk pengelolaan pembayaran VA / API dengan Bank	TBL_VA TBL_PAYMENT	

Tabel 3. Daftar microservice hasil migrasi pada modul keuangan

Implementasi Backend dengan Spring Boot

Setiap microservice diimplementasikan sebagai aplikasi Spring Boot yang menyediakan endpoint REST. Implementasi menerapkan pemisahan lapisan controller-service-repository, validasi input, serta penanganan error yang seragam agar perilaku layanan konsisten. Dokumentasi kontrak API disediakan menggunakan OpenAPI/Swagger untuk mendukung keterlacakan endpoint dan mempercepat integrasi antar layanan.

2.1.1. Integrasi dengan Sistem Legacy dan Pelaksanaan Cut-Over

Integrasi dilakukan dengan mempertimbangkan ko-eksistensi sistem legacy dan microservices pada fase transisi. Cut-over dilaksanakan secara bertahap, dimulai dari fungsi read-only (pelaporan) sebelum fungsi write yang kritikal (transaksi/pembayaran). Mekanisme rollback operasional disiapkan untuk mengurangi risiko apabila ditemukan isu pada layanan baru pada fase awal penerapan.

Evaluasi Penerapan Model Migrasi

Evaluasi dilakukan untuk memastikan layanan hasil migrasi memenuhi kebutuhan fungsional dan dapat dioperasikan secara stabil pada fase ko-eksistensi. Evaluasi mencakup: (1) validasi fungsional melalui skenario uji proses bisnis, (2) validasi keterlacakan artefak model migrasi antar tahap, (3) evaluasi tata kelola shared database untuk mengendalikan akses data, serta (4) evaluasi struktural terkait maintainability dan kemampuan deployment independen.

Validasi Fungsional

Validasi fungsional dilakukan menggunakan pengujian black-box berbasis skenario proses bisnis modul keuangan. Skenario difokuskan pada alur kritikal (pencatatan transaksi, pembayaran, dan pelaporan) untuk memastikan layanan hasil

migrasi konsisten dengan kebutuhan. Ringkasan skenario uji disajikan pada Tabel 4.

Tabel 4. Ringkasan skenario uji fungsional pada modul keuangan

Catatan: status pada tabel merupakan ringkasan hasil uji pada lingkungan uji/staging.

No	Skenario Uji	Layanan Terkait	Kriteria Keberhasilan	Status
1	Otentikasi dan otorisasi pengguna	Auth	Hak akses sesuai peran	Lulus
2	Pencatatan transaksi penerimaan	Finance	Transaksi tersimpan dan tervalidasi	Lulus
3	Pencatatan transaksi pengeluaran	Finance	Transaksi tersimpan dan tervalidasi	Lulus
4	Pembuatan invoice/tagihan	AP	Invoice terbentuk sesuai data transaksi	Lulus
5	Proses pembayaran (VA/transfer)	Payment	Status pembayaran terbarui dan tercatat	Lulus
6	Pembentukan jurnal akuntansi	Accounting	Jurnal terbentuk dan seimbang	Lulus
7	Pengajuan permintaan pengadaan	Procurement	Permintaan tercatat dan dapat ditelusuri	Lulus
8	Validasi anggaran sebelum realisasi	Budgeting	Transaksi ditolak jika melampaui anggaran	Lulus
9	Pembuatan laporan periodik	Reporting	Laporan konsisten dengan data transaksi	Lulus
10	Rollback/kompensasi saat gagal proses	Finance/Payment	Tidak ada data yatim/duplikasi	Lulus

Validasi Artefak Model Migrasi

Validasi artefak dilakukan dengan menelusuri keterhubungan (traceability) antara tahapan model migrasi dan artefak yang dihasilkan pada studi kasus. Validasi ini memastikan bahwa setiap tahap menghasilkan keluaran yang dapat digunakan pada tahap berikutnya, sebagaimana disarankan pada kerangka penilaian migrasi (Auer dkk., 2021) dan studi pemetaan sistematis migrasi monolit ke microservices (Martínez Saucedo dkk., 2025).

Tabel 5. Keterlacakan tahapan migrasi dan artefak pada studi kasus

Tahap	Artefak Utama	Output pada Studi Kasus	Catatan
1. Identifikasi domain	Daftar domain & bounded context	Pemetaan kapabilitas + kandidat layanan	Mengacu pada modul dan proses bisnis
2. Kandidat microservice	Daftar layanan kandidat	7 layanan (Tabel 3)	Batas layanan dievaluasi ulang saat implementasi
3. Ekstraksi &	Skeleton layanan & API	Layanan Spring Boot +	Kontrak API didokumentasikan

implementasi		endpoint REST	
4. Integrasi ko-eksistensi	Rencana integrasi & routing	Ko-eksistensi legacy + microservices	Inkremental sesuai cut-over
5. Tata kelola data	Matriks data ownership	Matriks akses data (Tabel 6)	Single-writer per entitas
6. Cut-over & eliminasi	Rencana eliminasi modul	Migrasi bertahap per fitur	Eliminasi setelah stabil

Evaluasi Tata Kelola Shared Database pada Fase Ko-eksistensi

Strategi shared database digunakan untuk menjaga kontinuitas layanan dan meminimalkan risiko migrasi skema. Namun, shared database dapat meningkatkan coupling jika tidak dikendalikan. Oleh karena itu, penelitian ini menerapkan prinsip data ownership (single-writer) untuk setiap entitas inti. Satu microservice menjadi pemilik operasi tulis, sementara layanan lain mengakses melalui operasi baca (read) atau melalui API pemilik. Praktik ini sejalan dengan temuan bahwa data management adalah salah satu tantangan dominan pada microservices (Laigner dkk., 2021).

Tabel 6. Contoh matriks data ownership pada shared database

Entitas/Tabel	Pemilik (Write)	Konsumen (Read)	Kontrol
USER, ROLE	Auth	Semua layanan	Role-based access + audit
TRANSACTION_HEADER/DETAIL	Finance	Accounting, Reporting	Single-writer + validasi transaksi
INVOICE	AP	Payment, Reporting	Single-writer + status workflow
PAYMENT	Payment	Finance, Accounting, Reporting	Single-writer + idempotency key
JOURNAL	Accounting	Reporting	Single-writer + balancing check
BUDGET_ALLOCATION	Budgeting	Finance, Procurement	Single-writer + constraint anggaran
PROCUREMENT_REQUEST	Procurement	Budgeting, Reporting	Single-writer + audit trail

Evaluasi Struktural dan Dampak terhadap Maintainability

Evaluasi struktural meninjau perubahan pada unit deployment, pemisahan tanggung jawab, serta kemudahan evolusi. Hasil implementasi menunjukkan bahwa pemisahan layanan memungkinkan deployment independen dan mengurangi dampak perubahan lintas modul. Namun, shared database masih menyisakan coupling pada lapisan data sehingga tata kelola akses dan perubahan skema perlu dijalankan secara disiplin. Isu-isu ini juga dilaporkan secara luas pada studi industri mengenai praktik dan pain points microservices (Zhou dkk., 2023).

Tabel 7. Ringkasan indikator struktural sebelum dan sesudah migrasi

Aspek	Sebelum (Monolit Glue)	Sesudah (Microservices Spring Boot)	Implikasi
Unit deployment	1 paket aplikasi	7 layanan mandiri	Rilis fitur lebih terisolasi
Perubahan kode	Sering berdampak lintas modul	Terfokus pada layanan terkait	Perubahan lebih terisolasi
Skalabilitas	Skala aplikasi keseluruhan	Skala per layanan (selective scaling)	Efisiensi resource
Data	Terpusat dan terikat modul	Shared database dengan data ownership	Perlu governance cegah coupling
Observabilitas	Log tersebar	Dapat distandarkan per layanan	Mendukung troubleshooting

Diskusi Komparatif dan Tantangan Implementasi

Dibandingkan strategi big-bang atau full rewrite, model migrasi bertahap lebih sesuai untuk e-Government yang menuntut kontinuitas layanan dan minim risiko. Strategi inkremental memungkinkan penambahan layanan baru secara bertahap sambil mempertahankan operasi legacy (Bozan dkk., 2021). Studi pemetaan sistematis juga menunjukkan bahwa pendekatan bertahap menjadi pilihan dominan pada migrasi monolit ke microservices karena lebih adaptif terhadap kendala organisasi dan teknis (Martínez Saucedo dkk., 2025).

Tantangan utama pada model ini berada pada pengelolaan shared database: walau mempercepat adopsi awal, shared database dapat menciptakan ketergantungan implisit antar layanan jika aturan ownership dilanggar. Oleh karena itu, penerapan mekanisme kontrol seperti audit log, pembatasan hak tulis, dan praktik idempotensi penting untuk mengurangi risiko anomali data. Selain itu, aspek keamanan layanan (mis. autentikasi/otorisasi dan hardening API) perlu menjadi perhatian khusus pada sistem sektor publik (Berardi dkk., 2022, Ponce dkk., 2023).

KESIMPULAN

Penelitian ini mengusulkan model migrasi modul keuangan e-Government dari monolit semi-modular berbasis Glue Framework ke arsitektur microservices berbasis Spring Boot dengan mempertahankan basis data Oracle eksisting. Model terdiri dari enam tahap yang menghasilkan artefak terstruktur untuk membantu

proses migrasi bertahap dan memungkinkan fase ko-eksistensi sistem lama dan baru. Studi kasus menunjukkan bahwa pendekatan migrasi pada lapisan aplikasi dapat menjadi strategi realistis untuk modernisasi e-Government pada kondisi keterbatasan data dan kebutuhan kontinuitas layanan.

Saran

Pekerjaan mendatang mencakup: perumusan strategi transisi menuju database-per-service secara bertahap, penambahan observabilitas (logging terpusat, distributed tracing) dan standar kontrak API, pengujian performa yang lebih sistematis menggunakan beban kerja representatif serta replikasi dan evaluasi model pada domain lain di lingkungan e-Government.

DAFTAR PUSTAKA

Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From monolithic systems to Microservices: An assessment framework. *Information and Software Technology*, *137*(January 2020), 106600. https://doi.org/10.1016/j.infsof.2021.106600

Berardi, D., Giallorenzo, S., Melis, A., Prandini, M., Mauro, J., & Montesi, F. (2022). Microservice security: a systematic literature review. *PeerJ Computer Science*, *7*, 1–66. https://doi.org/10.7717/PEERJ-CS.779

Bozan, K., Lyytinen, K., & Rose, G. M. (2021). How to transition incrementally to microservice architecture. *Communications of the ACM*, *64*(1), 79–85. https://doi.org/10.1145/3378064

Hassan, H., Abdel-Fattah, M. A., & Mohamed, W. (2024). Migrating from Monolithic to Microservice Architectures: A Systematic Literature Review. *International Journal of Advanced Computer Science and Applications*, *15*(10), 104–116. https://doi.org/10.14569/IJACSA.2024.0151013

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Sceince in Information Systems. *MIS Quarterly*, *28*(1), 75–105.

Laigner, R., Zhou, Y., Salles, M. A. V., Liu, Y., & Kalinowski, M. (2021). Data management in microservices: State of the practice, challenges, and research directions. *Proceedings of the VLDB Endowment*, *14*(13), 3348. https://doi.org/10.14778/3484224.3484232

Larsson, M. (2023). *Microservices with Spring Boot 3 and Spring Cloud: Build Resilient and Scalable Microservices Using Spring Cloud, Istio, and Kubernetes*. Packt Publishing Ltd.

Martínez Saucedo, A., Rodríguez, G., Gomes Rocha, F., & Santos, R. P. dos. (2025). Migration of monolithic systems to microservices: A systematic mapping study. *Information and Software Technology*, *177*, 107590. https://doi.org/10.1016/J.INFSOF.2024.107590

Newman, S. (2021). *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc."

Oumoussa, I., & Saidi, R. (2025). The Ontology-Based Mapping of Microservice Identification Approaches: A Systematic Study of Migration Strategies from Monolithic to Microservice Architectures. *Computers*, *14*(4). https://doi.org/10.3390/computers14040133

Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2007). D Esign S Cience in I Nformation. *MIS Quarterly*, *24*(1), 45–77.

Ponce, F., Soldani, J., Astudillo, H., & Brogi, A. (2023). *Microservices Security: Bad vs. Good Practices BT - Software Architecture. ECSA 2022 Tracks and Workshops* (T. Batista, T. Bureš, C. Raibulet, & H. Muccini (eds.); pp. 337–352). Springer International Publishing.

Richardson, C. (2018). *Microservices patterns: with examples in Java*. Simon and Schuster.

Riyanto, Irman Hermadi, & Yani Nurhadryani.

(2023). Analisis Uji Performa Aplikasi Dari Hasil Implementasi Refactoring Arsitektur Monolitik Ke Mikroservis dengan Decomposition dan Strangler Pattern. *Jurnal Sistem Cerdas*, *6*(3), 189–203. https://doi.org/10.37396/jsc.v6i3.352

Söylemez, M., & Tekinerdogan, B. (2022). *applied sciences Challenges and Solution Directions of Microservice Architectures : A Systematic Literature Review*.

Volynsky, E., Mehmed, M., & Krusche, S. (2022). Architect: A Framework for the Migration to Microservices. *Proceedings - 2022 International Conference on Computing, Electronics and Communications Engineering. ICCECE 2022*, 71–76. https://doi.org/10.1109/iCCECE55162.2022.9875096

Zhou, X., Li, S., Cao, L., Zhang, H., Jia, Z., & Zhong, C. (2023). *The Journal of Systems & Software Revisiting the practices and pains of microservice architecture in reality : An industrial inquiry ☆. 195*, 111521.