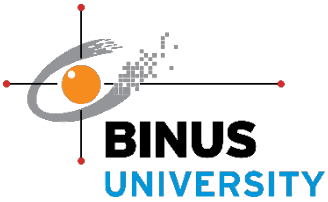
OUTLINE SKRIPSI NON KELAS/ARTIKEL ILMIAH

OTOMATISASI ARSITEKTUR MICROSERVICES: PEMANFAATAN GENERATOR UNTUK PENGEMBANGAN SUPER APP KEUANGAN YANG SKALABEL DAN MAINTAINABLE

MICROSERVICES ARCHITECTURE AUTOMATION: UTILIZING GENERATORS FOR DEVELOPING SCALABLE AND MAINTAINABLE FINANCIAL

Topik: Aplication Development

2602182556/Haikal Hamizan Hazmi/Computer Science/081275167471
2602208486/Defara Putra Nurimaba/Computer Science/085183066469
2602150182/Felisha Levana/Computer Science/087748874494



Computer Science
Computer Science Study Program
School of Computer Science
Universitas Bina Nusantara
Jakarta
2025

DAFTAR ISI

1. LATAR BELAKANG

Perkembangan teknologi digital saat ini telah mendorong perubahaan yang sangat signifikan dalam berbagai sektor, khususnya bidang keuangan digital. Menurut artikel (Liao et al., 2023), Pengembangan keuangan digital dapat secara efektif mendorong pertumbuhan ekonomi dan meningkatkan produktivitas faktor total daerah sekitarnya. Tantangan utama terletak pada bagaimana mengimplementasikan dan mengkomunikasikan inovasi tersebut secara efektif. Saat ini, keuangan digital tidak lagi sekadar tentang kemudahan bertransaksi, melainkan juga tentang menciptakan ekosistem finansial yang lebih efisien dan inklusif.

Perkembangan teknologi digital memicu gelombang transformasi di berbagai sektor, dengan bidang keuangan yang menjadi salah satu paling dinamis. Inovasi ini didorong teknologi seperti Artificial Intelligence (AI) dan Internet of Things (IoT) tidak sekedar mengoptimalkan operasional, tetapi juga secara fundamental mengubah cara konsumen berinteraksi dengan layanan keuangan.

Kehadiran teknologi digital finansial memberikan manfaat atau dampak positif bagi perusahaan yang penyedia jasa keuangan dan konsumen. Dengan menerapkan teknologi, perusahaan dapat menyediakan layanan keuangan untuk memudahkan dan memanjakan konsumen.

Meskipun platform low-code/no-code seperti Mendix menawarkan kemudahan dalam membangun aplikasi secara cepat dengan sedikit code, tetapi ada beberapa keterbatasan yang bisa membuatnya kurang cocok untuk pengembangan aplikasi kompleks seperti contoh yang kami bahas yaitu super app keuangan, antara lain:

1. **Keterbatasan** **Skalabilitas**
   Platform low-code pada umumnya tidak dirancang untuk menangani sistem dengan skala yang besar seperti super app dan integrasi multi-service (microservice). Kondisi ini menjadi kendala signifikan bagi aplikasi yang membutuhkan fleksibilitas tinggi dalam menangani ribuan hingga jutaan user serta mengelola volume master data yang besar

2. **Kurangnya Fleksibilitas dan Kustomisasi**

   Aplikasi berbasis low-code sering sangat bergantung pada fitur-fitur bawaan dari platform terkait. Ketika dibutuhkan fitur khusus seperti integrasi dengan payment gateway lokal, algoritma analisis keuangan, atau layanan microservices tertentu, pengembang kerap menghadapi keterbatasan signifikan pada platform low-code/no-code.

3. **Vendor Lock-in**

   Penggunaan platform Mendix umumnya menyebabkan aplikasi menjadi sangat bergantung pada ekosistem platform tersebut. Ketika suatu saat migrasi ke arsitektur lain seaperti menjadi microservice atau ke teknologi open-source bisa menjadi sulit, membutuhkan waktu yang signifikan dan biaya yang tinggi.

4. **Kurangnya Kontrol Teknis**

   Dalam konteks aplikasi keuangan yang kami saat ini bahas, keamanan dan keandalan adalah prioritas utama. Platform low-code membatasi akses developer terhadap detail-detail teknis (misalnya enkripsi, audit log, atau optimisasi performa), yang dapat menjadi sangat risiko dalam sistem keuangan berskala besar.

Sejumlah penelitian menegaskan bahwa low-code/no-code lebih sesuai untuk prototyping dan aplikasi berskala yang kecil, tetapi menghadapi keterbatasan yang signifikan saat diterapkan pada sistem enterprise dengan kebutuhan integrasi kompleks. Selain itu, lebih dari 40% diskusi developer di forum daring terkait low-code berfokus pada kesulitan menyesuaikan fitur bawaan dari platform terkait (Al Alamin et al., 2021; DOI: 10.48550/arXiv.2103.11429).

Dengan demikian, walaupun low-code/no-code mampu mempercepat pengembangan aplikasi, dan pendekatan ini tentunya belum sepenuhnya mampu menjawab kebutuhan pengembangan aplikasi keuangan berskala besar dan kompleks. Dan tentu hal ini membuka ruang bagi pendekatan lain, seperti penggunaan **generator berbasis arsitektur microservices**, yang menawarkan keseimbangan antara kecepatan pengembangan, fleksibilitas teknis, dan skalabilitas.

Software architecture adalah rancangan tingkat tinggi suatu *software* yang mendefinisikan komponen, interaksi, dan prinsip-prinsipnya. *Monolithic architecture* merupakan pendekatan tradisional di mana seluruh sistem dibangun sebagai satu

kesatuan yang menggunakan sumber daya dan ketergantungan yang sama. Aplikasi *microservice* merupakan pendekatan dalam pengembangan sebuah aplikasi tunggal sebagai kumpulan layanan kecil, masing-masing berjalan dalam prosesnya sendiri dan berkomunikasi melalui mekanisme ringan, sering kali menggunakan API (Application Programming Interface) yaitu antarmuka yang memungkinkan interaksi antar perangkat lunak, dengan protokol HTTPS (HyperText Transfer Protocol Secure) yaitu protokol komunikasi yang mengenkripsi data antara klien dan server melalui internet. Ketika arsitektur monolitik berkembang terlalu besar, mungkin sudah saatnya untuk beralih ke arsitektur microservice. (Hassan, 2024).

Beberapa tahun terakhir dalam bidang rekayasa perangkat lunak telah terjadi perubahan paradigma dari aplikasi *monolithic* menuju *microservice*, arsitektur di mana aplikasi dibagi menjadi entitas-entitas lebih kecil. Seiring bertambahnya jumlah layanan, kompleksitas pun meningkat. Peran generator menjadi penting. Untuk mendukung pendekatan ini, hadir *generator boilerplate* alat yang secara otomatis menghasilkan kode awal dan struktur dasar aplikasi. Otomatisasi ini memberikan permulaan efisien bagi pengembang dalam bekerja pada logika aplikasi (Aslam, 2022).

Meskipun platform low-code/no-code mampu mempercepat pembuatan aplikasi, pendekatan ini belum mampu menjawab kebutuhan aplikasi yang berskala besar yang menuntut skalibilitas, fleksibilitas, dan keamanan tinggi. Artsitektur microservice menjadi solusi yang lebih sesuai untuk kebutuhan ini memungkinkan pemisahana service yang mudah dikelola dan dikembangan. Namun, kompleksitas yang muncul ketika mulai banyaknya service juga memerlukan otomatisasi. Oleh karena itu, penelitian ini berfokus pada pemanfaatan boilerplate generator menggunakan tubro gen mendukung pengembangan super app keuangan yang lebih scalable, maintainable, dan efisien.

2. RUMUSAN MASALAH

Kompleksitas meningkat

Pengembangan aplikasi monolith sering menghadapi masalah kompleksitas kode yang tinggi, keterbatasan dalam pengelolaan modul, dan kesulitan dalam integrasi fitur baru secara cepat. Dalam konteks aplikasi keuangan yang menuntut fleksibilitas dan skalabilitas yang tinggi, tantangan ini menajdi lebih kritis.

Vertical scaling dependency

Arsitektur monolith cenderung sulit untuk diskalakan secara horizontal karena seluruh aplikasi dijalankan sebagai satu kesatuan. Hal ini menyebabkan *bottleneck* ketika beban transaksi meningkat, terutama pada aplikasi keuangan yang membutuhkan *throughput* tinggi dan *latency* yang rendah.

Maintenance monolith

Pemeliharaan aplikasi monolith sering kali rumit karena perubahan kecil yang berdampak luas pada sistem, sulit untuk melakukan deployment parsial, dan memerlukan waktu *downtime* lebih lama. Hal ini beresiko mengganggu kontinuitas layanan keuangan yang sensitif terhadap waktu

Implementasi Microservice Generator

Permasalahan utama yang sering dihadapi dalam pengembangan aplikasi adalah peralihan arsitektur perangkat lunak akibat berkembanganya suatu bisnis atau perusahaan yang tidak lagi memungkinkan penggunaan arsitektur monolitik. Model tersebut memiliki keterbatasan yang membuat proses *maintenance* semakin kompleks ketika aplikasi berkembang hingga mencapai skala besar. Alternatif seperti *low-code* atau *no-code* menawarkan kecepatan dalam pembuatan aplikasi. Namun, terbukti terbatas ketika harus menangani kebutuhan sistem berskala besar dan kompleks. *Microservices* sebagai solusi memberikan optimisasi dengan pengadaan *boilerplate* yang sesuai dengan kebutuhan akan fleksibilitas, skalabilitas, dan kemudahan pemeliharaan. Hasil akhir implementasi disajikan dalam bentuk peragaan *proof of concept* untuk aplikasi keuangan.

Keterbatasan ketika menggunakan platform low-code/no-code itu menjadi sangat penting ketika mau mengembangkan aplikasi skala besar. Walaupun platform ini mampu untuk mempercepat proses pembuatan aplikasi, pada praktiknya terdapat hambatan seperti keterbatasan skalabilitas dalam menangani jumlah user dan transaksi yang terus bertambah, kesulitan kustomisasi ketika adanya integrasi dengan service eksternal seperti payment gateway, dan isu vendor lock-in yang menyulitkan migrasi teknologi kedepannya. Kondisi ini membuat platform low-code/no-code kurang ideal digunakan untuk aplikasi yang menuntut keamanan tinggi, fleksibilitas, dan performa yang optimal.

Sebagai alternatif, arsitektur microservices dipandang sebagai solusi yang mampu mengatasi permasalahan tersebut. Dengan memecah aplikasi menjadi services independent, microservices memungkinkan setiap modul, seperti transaksi, laporan, notifikasi, maupun autentikasi, untuk dikembangkan, dipelihara, dan meningkatkan skalabilitas dan maintainabilitas, tetapi juga mendukung efisiensi pengembangan melalui pemanfaatan otomatisasi boilerplate generator menggunakan turbo gen yang dapat menghasilkan kode awal secara konsisten dan siap digunakan. Dengan adanya solusi ini berfokus pada bagaimana penerapan arsitektur microservices yang didukung oleh generator dapat menjadi solusi bagi developer yang ingin membuat super app atau aplikasi yang lebih scalable, maintainable, dan efisien.

3. RUANG LINGKUP

Dalam thesis ini, kita akan mengembangkan sebuah generator super app keuangan yang menggunakan arsitektur microservice, namun hanya sebatas pembuatan demo dan tidak akan mengimplementasikan super app ini secara penuh sampai deployment. Super app merupakan aplikasi yang mengintegrasikan berbagai layanan keuangan seperti pembayaran, peminjaman dalam satu platform terpadu, sehingga memudahkan pengguna untuk mengakses berbagai layanan tanpa perlu berpindah aplikasi. Dengan menggunakan arsitektur microservice, aplikasi ini dibangun sebagai kumpulan service kecil yang berdiri sendiri dan saling berkomunikasi, yang memungkinkan fleksibilitas, skalabilitas, dan kemudahan *maintenance*. Namun, karena ini sampai tahap demo, maka fokus pengembangan hanya fokus terbatas pada *microservices template generator* dan service penting saja, tanpa mengakomodasi service yang biasanya terdapat pada super app keuangan secara komprehensif. Pendekatan ini diambil agar proyek dapat diselesaikan dalam waktu terbatas dan memberikan gambaran konsep yang jelas mengenai penerapan arsitektur microservice dalam pengembangan superapp keuangan.

Pengembangan *microservices template generator* ditujukan bagi tim IT atau *developer* sebagai pelaksana utama dari pembangunan sistem perangkat lunak. Dengan adanya *template generator*, *developer* dapat mengadakan kode dasar yang konsisten di awal pengembangan. Keuangan menjadi fokus utama sebab setiap entitas bisnis maupun perusahaan pada dasarnya tidak dapat terlepas dari aspek keuangan sebagai komponen fundamental dalam operasional dan pengembangannya. Regulasi bisnis dan perusahaan di

berbagai sektor terus berkembang setiap tahunnya yang memerlukan pemeliharaan berkelanjutan. Perkembangan kebutuhan perangkat lunak kerap menuntut adanya pergeseran arsitektur sebagai solusi adaptif. Namun, melakukan pembangunan ulang aplikasi secara menyeluruh bukanlah pendekatan yang efisien, terutama apabila sistem masih berbasis arsitektur monolitik. Melalui penerapan *template generator* pada arsitektur *microservices*, proses pengembangan dapat berlangsung lebih terstruktur, konsisten, dan efisien dalam menghadapi dinamika perubahan.

Dalam pengembangan demo super app keuangan ini, cakupan implemtasi kami hanya membuat empat service/modul utama, yaitu:

1. Core

   Modul ini berfungsi sebagai pusat data utama yang menyimpan informasi dasar yang digunakan oleh modul lain. Master data dapat mencakup:

   - Produk/Item: dipakai oleh modul Inventory & Purchase.
   - Vendor/Supplier: dipakai oleh modul Purchase.
   - Kategori/Unit Satuan: mendukung pencatatan data barang.

   Tujuannya adalah memastikan setiap modul memiliki referensi data yang konsisten dan mengurangi duplikasi.

2. Inventory

   Modul ini berfungsi untuk mengelola persediaan barang, mencatat stok masuk dan keluar, serta menyediakan data yang diperlkan untuk membuat laporan ketersediaan produk. Dengan adanya modul ini, sistem dapat mendukung kontrol barang secara lebih akurat dan transparan.

3. Purchase

   Modul ini berfokus pada pengelolaan proses pembelian, mulai dari pencatatan pesanan, pengelolaan pemasok, hingga pencatatan transaksi pembelian. Tujuannya adalah memastikan keterhubungan antara kebutuhan stok dengan proses pengadaan barang.

4. Finance

   Modul ini hanya mencakup pencatatan transaksi keuangan sederhana berupa arus kas masuk (Cash In) dan arus kas keluar (Cash out).

- Cash In: berasal dari modul purchase yaitu transaksi pembayaran masuk dari transaksi pemesanan
- Cash Out: berasal dari modul purchase yaitu transaksi pembayaran ke pemasok

Tujuannya adalah mendemontrasikan bagaimana arus keuangan dasar dapat diintegrasikan secara modular dengan proses bisnis lain dalam atsitektur microservices. Modul ini masih terbatas pada pencatatan sederhana, belum mencakup fitur kompleks.

Limitasi ada pada lingkup *proof of concept*, yaitu pembuatan *prototype template generator*. Implementasi penuh tidak duperuntukkan di perusahaan nyata, melainkan pengujian terbatas dalam lingkungan simulasi. Aspek yang akan dibahas meliputi *design template* sebagai kerangka dasar yang dapat digunakan ulang (*reusable scaffold*) dan alur otomatisasi pembuatan service baru, bukan deployment skala penuh atau pemeliharaan pasca produksi.

4. TUJUAN DAN MANFAAT

Tujuan

Implementasi arsitektur microservice untuk super app keuangan melalui demo generator. Implementasi ini memperlihatkan bahwa arsitektur microservice merupakan pilihan yang tepat dalam pengembangan super app keuangan. Setiap layanan dibangun sebagai modul terpisah yang saling terintegrasi, sehingga mendukung skalabilitas, fleksibilitas, serta kemudahan pengelolaan berbagai fitur dan kebutuhan di bidang keuangan.

Arsitektur *microservice* menawarkan keunggulan signifikan dalam **skalabilitas, maintainabilitas,** dan **modularitas** dibandingkan dengan sistem *monolith* atau *low-code*. Secara fundamental, **skalabilitas** *microservices* ditingkatkan melalui kemampuan *horizontal scaling* independen, dimana setiap layanan di-skala sesuai kebutuhan spesifiknya tanpa mempengaruhi kesuluruhan aplikasi, yang menghasilkan pemanfaatan sumer daya lebih efisien dan isolasi kegagalan yang lebih baik [Nygard, 2007]. Hal ini berbeda dengan *monolith* yang sering memerlukan *scalling* secara menyeluruh, atau *low-code* yang mungkin memiliki batasan scalling dari platformnya. Dari sisi **maintainabilitas**, *microservices* memungkinkan pmbagian layanan menjadi unit-unit kecil

dengan basis kode yang fokus dan *responsibility* tunggal, sehingga lebih mudah dipahami, diuji dan diperbaiki oleh tim *development*

Fleksibilitas tercermin dalam kemampuan menggunakan teknologi untuk setiap layanan, serta pembaruan dan *deployment* yang lebih cepat dan independen. Sementara monolith sering bergulat dengan kode yang kompleks dan besar, serta *low-code* bergantung pada *maintainability* platform, *microservices,* dengan pembagian *responsibility* yang jelas antar layanan memungkinkan pengembangan paralel oleh tim development dalam reorganisasi atau penggantian modul tanpa mengganggu sistem lainnya.

Setiap *microservice* dapat dibangun dan di-deploy secara independen, memberikan otonomi dan kecepatan yang tidak ditemukan dalam pendekatan *monolith* tradisional atau keterbatasan yang mungkin ada pada solusi *low-code.*

Dengan modul yang akan kami implementasikan seperti Core (master data), Inventory, Purhcase, dan Finance akan menunjukkan bahwa sistem berbasis microservices tetap bisa berjalan secara terintegrasi sesama modul lainnya. Masing-masing service memiliki peran yang jelas dan berdiri secara indpendent, tetapi masih bisa untuk saling berkomunikasi antar service. Pendekatan ini memberikan gambaran bahwa meskipun hanya proof of concept(PoC), arsitektur microservices bisa membuat menjadi fleksibilitas, serta kemudahan maintainance yang lebih baik dibandingkan monolith.

Selain itu, penggunaan generator dalam pengembangan dengan arsitektur microservices berperan penting untuk mempercepat proses pembuatan suatu service baru yang akan di kembangkan. Dengan adanya generator, code dasar dan struktur awal sebuah aplikasi dapat dibuat secara otomatis dan konsisten di setiap modul, jadi developer tidak lagi mengulang pekerjaan yang sama dari awal. Pendekatan ini tidak hanya menghemat waktu, tetapi bisa mengurangi adanya potensi kesalahan konfigurasi, lalu juga membantu untuk menjaga kualitas dan efisiensi dalam pengembangan aplikasi skala besar berbasis microservices.

4.1.Manfaat

Manfaat utama menyediakan referensi teknis bagi developer atau tim IT dalam pengembangan aplikasi berskala terletak pada pemberian layaknya “peta jalan” yang konsisten untuk seluruh tim. Dokumentasi arsitektur, pola desain, serta standar

konfigurasi memastikan setiap layanan *mikroservice* seperti modul Code, Inventory, Purchase, dan Finance dapat berinteraksi tanpa konflik.

Ada juga penilitian ini bisa bermanfaat untuk membantu mengurangi risiko vendor lock-in yaitu keterbatasan dari platform low-code/no-code ini sering terjadi jika menggunakan platform tersebut. Dengan pendekatan berbasis generator dan arsitektur microservices, dan jika menggunakan pendekatan ini akan dapat dikembangkan dengan teknologi open-source lebih fleksibel dan independen. Hal ini memungkinkan developer memiliki kontorl atas arsitektur, konfigurasi, dan integrasi services, tanpa harus bergantung pada fitur atau ekosistem dari platform tersebut.

Penerapan demo ini berfungsi sebagai proof of concept yang dapat dijadikan dasar sebelum melakukan implementasi penuh pada lingkungan perusahaan. Tahap konsep diperlukan agar setiap keputusan teknis dan bisnis dapat divalidasi terlebih dahulu, sehingga risiko kegagalan dapat diminimalisasi, biaya implementasi lebih terkendali, serta memastikan bahwa arsitektur yang dipilih sesuai dengan kebutuhan nyata.

Demo memberikan gambaran praktis mengenai pengelolaan operasi keuangan secara modular. Sistem dirancang dalam empat modul utama yang merepresentasikan aspek penting dalam proses bisnis, diantaranya *core, inventory, purchase,* dan *finance*. Pemilihan keempat modul ini didasarkan pada perannya yang mewakili siklus utama dalam pengelolaan keuangan perusahaan. Pendekatan modular dipilih karena mampu memecah sistem yang kompleks menjadi bagian-bagian yang lebih kecil dan mudah dikelola. Setiap modul dapat dikembangkan, diuji, serta dipelihara secara terpisah tanpa harus memengaruhi keseluruhan sistem, membuat modular siap diimplementasikan dalam skala yang lebih besar.

Penggunaan *code generator* secara signifikan mempercepat proses pengembangan awal microservice dengan mengotomatiskan pembuatan koded dasar atau *boilerplate.* Dalam konteks arsitektur microservice, setiap layanan memiliki komponen struktural yang hampir identik seperti: model data (entity), repository, controller, DTO, validator, frontend, backend, config, dan test skeleton. Dengan *code generator,* developer cukup menentukan nama service dan fitur yang diinginkan, lalu seluruh struktur dan kode awal

akan dibuat secara otomatis. Proses ini mengurangi waktu yang dibutuhkan untuk inisiasi sebuah service dari beberapa jam menjadi hanya 15-30 menit lebih. Tidak hanya cepat, *dependency* ini menjamin konsistensi struktur kode secara menyeluruh, setiap service yang dihasilkan menggunakan pola yang sama, sehingga memudahkan pemahaman, debugging, dan perawatan sistem. Selain itu, generator kode mengurangi risiko human error seperti typo, struktur folder yang salah, atau pengabaian *component* penting.

5. METODOLOGI PENELITIAN

5.1 Pengumpulan Data

Studi Literature

Pengumpulan data dimulai dengan studi literatur komprehensif mengenai arsitektur perangkat lunak, khususnya perbandingan antara arsitektur monolith dan microservices. Sumber literatur yang digunakan meliputi jurnal ilmiah, conference proceedings, technical reports, dan dokumentasi resmi dari platform teknologi yang relevan. Fokus kajian literatur mencakup evolusi arsitektur software dalam aplikasi berskala besar, keunggulan dan keterbatasan masing-masing pendekatan arsitektur, serta best practices dalam implementasi microservices untuk aplikasi berskala enterprise.

5.2 Metode Pengembangan

Perancangan Arsitketur: Microservices merupakan sekumpulan services yang dibangun berdasarkan batasan tanggung jawab tertentu, di mana setiap layanan memiliki fungsi spesifik dan dapat beroperasi secara independen. Untuk mendukung interaksi antar layanan, NATS diimplementasikan sebagai perantara komunikasi antar layanan.

Pembuatan Generator:

Pada tahap pengembangan sistem, digunakan **Turbo Gen** sebagai generator berbasis boilerplate untuk mengotomatiskan proses pembuatan struktur dasar microservices. Generator ini dirancang untuk mengatasi tantangan konfigurasi manual yang membutuhkan waktu signifikan dan rentan terhadap ketidakkonsistenan.

Fitur Utama Generator:

- Pembuatan struktur direktori yang konsisten untuk setiap service
- Template kode dasar (boilerplate) yang siap pakai
- Konfigurasi otomatis untuk file package.json, TypeScript, ESLint, dan Prettier
- Skrip CLI untuk membuat service baru secara otomatis
- Standarisasi struktur project yang memudahkan maintenance

Implementasi Modul: Arsitektur dikembangkan ke dalam empat modul utama yang merefleksikan komponen kunci dalam aktivitas bisnis. Core berperan sebagai master data yang menyediakan informasi dasar bagi modul lain. Inventory bertugas mengelola persediaan barang sebagai bagian dari pencatatan aset. Purchase fokus pada pencatatan proses pembelian secara terstruktur, sedangkan Finance mengatur pencatatan arus kas masuk dan keluar sebagai inti dari aktivitas keuangan.

Integrasi & Testing: tahap ini dilakukan dengan cara menyambungkan semua modul yang sudah dibuat sebelumnya, yaitu Core, Inventory, Purhchase, dan Finance. Pengujuian dilakukan di localhost untuk memastikan tiap service bisa berjalan, saling berkomunikasi antar modul melalui NATS server, serta dapat diakses datanya sesuai kebutuhan modul. Kita fokus pada pengecekan fungsi dasar, misal apakah ada bug/erorr, apakah ada data dari suatu modul bisa dipkai di modul lain, dan memastikan semua berjalan secara bersamaan.

Analisis: tahap ini adalah tahap akhir dari metode yaitu setelah pengujian dengan cara evaluasi hasil implementasi yang sudah berjalan. Dari apa yang sudah dibuat disini kita bisa dilihat apakah arsitektur microservices apakah sesuai dengan tujuan awal, apakah komukisai antar modul stabil, serta apakah generator dan modul yang dibuat sudah cukup membantu memerpcet pengembangan. Hasil ini juga memberi gambara pada kelebihan, kekurangan, dan potensi pengembangan lebih lanjur dari sistem yang udah dibangun.

DAFTAR PUSTAKA

Hassan, M. (2024, February 25). *Software Architecture Between Monolithic and Microservices Approach.* Retrieved from papers.ssn.com: http://dx.doi.org/10.2139/ssrn.4753649

Aslam, M. Z. (2022, December 12). *CREATING A MICROSERVICE GENERATOR FOR GO-BASED MICROSERVICES.* Retrieved from Trepo: https://trepo.tuni.fi/bitstream/handle/10024/144420/AslamMuhammadZohaib.pdf?sequence=2

Dharika, K. (202, June 30). *Microservices Architecture in Financial Systems: Explain the role and benefits of microservices in fintech applications Creators* . Retrieved from Zenodo: https://zenodo.org/records/10889750

Liao, H., Wei, Y., Wang, Y., Lin, Y., & Tan, R. (2023, April 20). *The impact of digital finance on commercial banks risk-taking: Empirical analysis based on 176 Chinese commercial banks.* Retrieved from ScienceDirect: https://www.sciencedirect.com/science/article/pii/S154461232300301X?via%3Dihub