Terbit *online* pada laman : http://teknosi.fti.unand.ac.id/

Jurnal Nasional Teknologi dan Sistem Informasi

| ISSN (Print) 2460-3465 | ISSN (Online) 2476-8812 |

Artikel Penelitian

ABizLSM: Kerangka Migrasi Sistem Lawas Genting Arsitektur Monolitik ke *Microservices* untuk Perusahaan *Startup* dengan Dinamika Bisnis Tinggi

Michael Susanto [a,*], *Muhammad Hafizhuddin Hilman* [b]

[a,b] *Computer Systems Lab (CSL), Fakultas Ilmu Komputer, Universitas Indonesia, Kota Depok, Jawa Barat 16424, Indonesia*

INFORMASI ARTIKEL

Sejarah Artikel:
Diterima Redaksi: 05 November 2024
Revisi Akhir: 23 April 2025
Diterbitkan *Online*: 30 April 2025

KATA KUNCI

Migrasi Sistem,
Microservices,
Domain-Driven Design

KORESPONDENSI

E-mail: michael.susanto21@ui.ac.id*

A B S T R A C T

Proses rekayasa ulang migrasi sistem monolitik lama ke sistem berbasis *microservices* sering kali penting bagi kelangsungan bisnis. Pada ruang lingkup perusahaan *startup*, dinamika perubahan bisnis cenderung lebih gesit dan cepat, yang membuat sistem sering kali tidak terawat dan terdokumentasi dengan baik. Di sisi lain, pengembang ditantang untuk memperbarui sistem lama dengan perubahan bisnis perusahaan *startup* yang dinamis dari waktu ke waktu. Banyak studi telah mengusulkan proses migrasi sistem monolitik lama. Namun, penelitian umumnya terfokus pada perusahaan korporat, yang sering tidak mempertimbangkan proses migrasi dalam lingkungan bisnis perusahaan *startup* yang gesit. Dengan demikian, penelitian ini membahas masalah tersebut dan mengusulkan kerangka kerja *Agile Business Legacy System Migration* (ABizLSM) yang berasal dari prinsip-prinsip *Domain-Driven Design* (DDD) dengan fokus pada migrasi sistem monolitik lama di perusahaan *startup*. Penelitian ini menggunakan metode penelitian campuran dengan melakukan wawancara dan mengevaluasi sistem yang dimigrasikan dengan membandingkan latensi dengan sistem lama. Hasil penelitian menunjukkan bahwa kerangka kerja ABizLSM telah berhasil memigrasikan sistem monolitik lama dalam perusahaan *startup* yang gesit, sehingga memberikan kemudahan pemeliharaan dan skalabilitas yang lebih baik serta meningkatkan kualitas sistem secara keseluruhan. Selain itu, tidak ada penurunan kinerja pada sistem yang telah dimigrasikan, sehingga kerangka kerja ABizLSM berhasil memigrasikan sistem monolitik lawas genting.

1. PENDAHULUAN

Perkembangan teknologi memengaruhi implementasi dan arsitektur sistem baik dari segi perangkat keras maupun perangkat lunak. Perangkat keras dan perangkat lunak selalu mengalami perubahan dan menjadi semakin baik dari waktu ke waktu. Keberagaman pilihan implementasi sistem saat ini membuat suatu sistem pada perusahaan dapat dibangun dari komponen-komponen perangkat keras dan perangkat lunak yang dikelola sendiri, pihak ketiga, atau gabungan keduanya [1]. Hal ini memberikan tantangan baru dalam pengelolaan sistem di masa kini, yang dimulai dari pengembangan, konfigurasi, hingga *deployment* [2].

Setiap perusahaan pada umumnya memiliki suatu sistem atau aplikasi yang bersifat kritis bagi keberlangsungan bisnisnya. Sistem-sistem ini seringkali sudah menyajikan logika bisnis utama perusahaan dalam jangka waktu yang lama [3], tanpa adanya pengelolaan dan perancangan yang baik pada sisi arsitektur aplikasinya. Seiring berjalannya waktu, persaingan bisnis akan semakin ketat yang membuat perusahaan harus cepat dan tanggap dalam menghadapi permintaan atau permasalahan yang sedang terjadi pada konsumen atau pengguna aplikasinya [4]. Salah satu bentuk nyata dari perubahan ini adalah sistem perlu cepat dan tanggap terkait penambahan atau perbaikan fungsi atau fitur yang ada di dalam aplikasi perusahaan tersebut. Pada satu sisi, proses penyesuaian fungsi pada aplikasi perusahaan seringkali menyebabkan sistem perusahaan secara keseluruhan tidak dirancang atau didesain dengan baik arsitekturnya. Hal ini membuat sistem-sistem tersebut akan menjadi sistem lawas [5]. Di sisi lain, sistem milik perusahaan juga harus efektif, efisien, dan *scalable* secara biaya dan operasional [6].

https://doi.org/10.25077/TEKNOSI.v11i1.2025.27-36

Pengembang aplikasi dihadapi tantangan untuk mengembangkan dan memelihara sistem lawas genting agar dapat terus mengikuti kecepatan perkembangan bisnis dari perusahaan [7]. Sistem lawas ini pada umumnya memiliki arsitektur monolitik yang kurang baik serta menggunakan teknologi yang tertinggal, yang mengurangi kemampuan sistem lawas tersebut untuk mengimbangi kecepatan perubahan bisnis [8]. Selain itu, proses perpindahan pengetahuan yang kurang baik di perusahaan juga mempersulit pengelolaan sistem lawas ini di masa mendatang [9]. Untuk mengatasi permasalahan ini, banyak perusahaan yang melakukan pengkinian implementasi sistem, seperti melakukan migrasi dari arsitektur monolitik menuju ke arsitektur *microservices* untuk kemudahan pengelolaan dan *scalability* di masa mendatang.

Arsitektur *microservices* secara umum dipilih menjadi target arsitektur untuk proses modernisasi sistem yang sebelumnya memiliki arsitektur monolitik. Arsitektur ini diturunkan dari arsitektur berorientasi layanan, atau *Service-Oriented Architecture* (SOA), yang juga memiliki prinsip-prinsip yang serupa dengan SOA [10]. Layanan-layanan mikro di dalam arsitektur *microservices* memiliki fokusnya masing-masing dalam memenuhi logika bisnis perusahaan dan dapat bekerja bersama-sama untuk memenuhi logika bisnis perusahaan yang lebih besar [11].

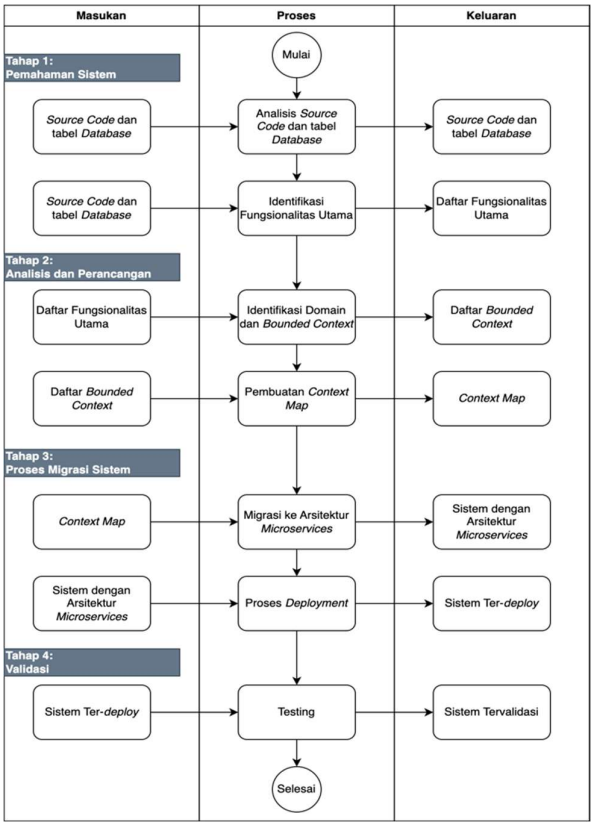
Pada lingkugan perusahaan *startup* yang memiliki perubahan kebutuhan bisnis yang relatif cepat, umumnya sistem dibangun dan dimulai dari arsitektur monolitik yang pada dasarnya mudah untuk dikelola dalam skala kecil. Seiring bertumbuhnya perusahaan *startup*, tanpa perancangan dan pengelolaan yang baik, sistem arsitektur monolitik tersebut akan semakin besar, kompleks, dan sulit dikelola di masa mendatang [12]. Sistem ini yang di kemudian hari menentukan apakah perusahaan dapat cepat tanggap terhadap kebutuhan konsumen nya atau tidak.

Melihat permasalahan ini, penelitian ini mengusulkan sebuah kerangka migrasi sistem lawas genting untuk perusahaan *Startup* yang memiliki dinamika bisnis tinggi, yang diberi nama ABizLSM (*Agile Business Legacy System Migration*). ABizLSM dirancang dengan pendekatan *Domain-Driven Design* (DDD) dengan beberapa penyesuaian sesuai sumber daya yang ada pada perusahaan *startup*, yang secara umum perusahaan *startup* memiliki sumber daya yang terbatas [13]. Pemilihan proses migrasi berdasarkan prinsip *Domain-Driven Design* (DDD) bertujuan agar setiap *microservice* yang dirancang nantinya dapat sesuai dengan ranah atau domain-domain bisnis di perusahaan [14]. Selain itu, perusahaan *startup* juga cenderung kurang memiliki sumber daya dan dokumentasi yang baik atas sistem yang ada, hal ini dikarenakan perusahaan *startup* memiliki fokus utama untuk memasarkan produk atau aplikasi terlebih dahulu [15]. Kerangka ABizLSM memastikan sistem hasil migrasi memiliki kualitas yang lebih baik dan pengelolaan yang lebih mudah di masa mendatang, dibandingkan dengan sebuah sistem monolitik besar.

Kerangka yang diusulkan dalam penelitian ini divalidasi dan diterapkan dengan pendekatan studi kasus pada salah satu *e-commerce startup* untuk kebutuhan furnitur dan desain interior, yang selanjutnya disebut sebagai PT XYZ dalam penelitian ini. PT XYZ memiliki sebuah sistem lawas genting yang telah berjalan selama 9 tahun dan melayani fungsi bisnis utamanya, terkait jual beli furnitur dan desain interior. Proses pengembangan sistem di PT XYZ memiliki fokus utama untuk menyediakan produk dan fitur setanggap mungkin terhadap perubahan lingkungan bisnis agar memiliki keuntungan kompetitif [16]. Seiring berjalannya waktu, sistem monolitik ini menunjukkan adanya tanda-tanda kesulitan dalam hal pengelolaan dan *scalability*.

2. METODE

Untuk menghasilkan sebuah kerangka migrasi sistem lawas genting menuju arsitektur *microservices*, penelitian ini menggunakan metode *exploratory research*. Penelitian diawali dengan metode kualitatif untuk memahami kriteria dan ciri-ciri sistem lawas genting yang ada di perusahaan *startup*. Hal ini dilakukan melalui pengumpulan data baik dari observasi sistem maupun wawancara. Metode kualitatif ini digunakan untuk memahami lebih lanjut sistem lawas genting yang ada, agar dapat menghasilkan rancangan arsitektur *microservices* sesuai dengan karakteristik sistem dan sumber daya yang ada. Setelah kriteria sistem dipahami, proses migrasi dilakukan dengan merancang arsitektur *microservices* berdasarkan prinsip *Domain-Driven Design* (DDD). Seluruh proses yang dilakukan secara kualitatif akan diusulkan sebagai kerangka proses migrasi ABizLSM yang kemudian divalidasi secara kuantitatif melalui validasi sistem dan penyebaran survei. Gambar 1 memberikan gambaran umum terkait proses migrasi sistem menggunakan kerangka ABizLSM.



Gambar 1. Proses Migrasi Sistem dengan Kerangka ABizLSM

https://doi.org/10.25077/TEKNOSI.v11i1.2025.27-36

2.1. Tahap Pemahaman Sistem

Semakin berkembangnya sistem monolitik di suatu perusahaan, sistem tersebut akan cenderung menjadi besar, kompleks, dan menjadi sulit dikelola [12]. Sebelum arsitektur *microservices* diusulkan, karakteristik dari sistem yang sudah ada saat ini perlu dipahami lebih lanjut sehingga dapat mengetahui apa yang dapat diperbaiki dari arsitektur sistem. Dalam perusahaan *startup*, dokumentasi sistem cenderung kurang dapat diandalkan karena secara umum perusahaan *startup* kurang memiliki dokumentasi yang lengkap [9]. Tahap pertama dimulai dengan analisis *source code*, *database*, dan mengidentifikasi fungsionalitas utama pada sistem lawas genting yang ada.

2.1.1. Analisis Source Code dan Database

Untuk memahami karakteristik dan arsitektur sistem lawas genting yang ada, perlu dilakukan penggalian terhadap sistem yang ada. Cara yang dapat dilakukan adalah melakukan analisis *source code* dan struktur *database* dari sistem monolitik [17]. Dengan memahami bagaimana penyusunan dan pengorganisasian kode pada sistem, hal ini dapat mempercepat proses analisis sistem. Pada tahap ini, bagaimana sistem terstruktur secara internal dan interaksi dengan sistem atau aplikasi pihak ketiga juga perlu diketahui sebagai pertimbangan dalam merancang arsitektur *microservices*. Selain itu, karakteristik dari pola dan jumlah *request* yang dilayani sistem untuk setiap fitur setiap harinya juga perlu diketahui agar pemecahan *microservice* juga dapat optimal [18].

2.1.2. Identifikasi Fungsionalitas Utama

Setelah memahami karakteristik sistem monolitik secara garis besar, tahap selanjutnya adalah melakukan identifikasi fungsionalitas utama pada sistem. Berbagai fungsionalitas dan fitur-fitur yang terdapat pada sistem penting diketahui agar memudahkan proses pengelompokan *microservices*. Proses ini perlu dilakukan secara inkremental dan iteratif [17]. Proses yang dilakukan secara berulang ini bertujuan agar semua fungsionalitas utama di sistem dapat teridentifikasi dan tidak ada yang terlewat.

Proses identifikasi fungsionalitas utama dapat dilakukan dengan cara melihat keterkaitan konteks bisnis pada setiap entitas pada *source code* atau tabel dalam *database*, secara *bottom-up*. Setelah mencatat tabel-tabel yang ada pada *database*, proses pengelompokan dilihat dari pemberian nama tabel, melihat bagaimana relasinya dengan tabel lain (dalam bentuk *foreign key*), dan logika bisnis yang ada pada *source code* [19]. Fungsionalitas ini yang pada menjadi landasan terbentuknya domain pada tahap berikutnya.

2.2. Tahap Analisis dan Perancangan

Fungsionalitas utama sistem monolitik yang telah teridentifikasi menjadi landasan untuk membangun domain-domain bisnis. Dalam tahap ini, *Domain-Driven Design* (DDD) digunakan sebagai pendekatan kolaborasi antara pihak yang mewakili sisi bisnis dan pihak yang mewakili sisi pengembang aplikasi. Oleh karena perusahaan *startup* cenderung memiliki tingkat *turn-over* yang tinggi [20], kolaborasi dapat dilakukan antara perwakilan dari sisi bisnis dan pengembang yang ada di perusahaan saat ini. Tahap kedua ini terdiri dari identifikasi domain, *bounded context*, dan pembuatan *context map* sebagai kerangka arsitektur *microservices*.

2.2.1. Identifikasi Domain dan Bounded Context

Model domain adalah artefak tingkat tinggi yang mengatur dan menjelaskan pengetahuan domain organisasi [17]. Sebuah domain merepresentasikan entitas objek di dunia nyata. Walaupun demikian, hal ini tidak berarti bahwa setiap *microservice* harus menerapkan model yang sama untuk satu entitas atau fungsionalitas terkait [21]. Oleh karena fungsionalitas utama yang didapatkan pada tahap sebelumnya sudah berupa entitas, maka entitas tersebut dapat menjadi landasan bagi representasi domain pada sistem lawas genting.

Setelah mengetahui domain pada sistem, berikutnya ditentukan relasinya satu sama lain untuk memeroleh pemahaman bagaimana interaksi antar-domain tersebut. Proses identifikasi relasi dan interaksi antar-domain dilakukan dengan analisis logika-logika bisnis yang terdapat pada *source code* sistem. Pengelompokan domain-domain dengan fungsionalitas yang serupa dan sudah disesuaikan dengan kebutuhan unit fungsional bisnis yang terdapat pada struktur organisasi membentuk *bounded context*. Setiap entitas dunia nyata dimungkinkan untuk memiliki representasi yang berbeda-beda di setiap *bounded context* sesuai dengan kebutuhan konteks [17].

2.2.2. Pembuatan Context Map

Semua *bounded context* yang telah dirancang akan dipetakan satu sama lain untuk mengetahui relasi nya, yang kemudian membentuk sebuah *context map*. Proses pembuatan *context map* ini bertujuan agar pihak perwakilan bisnis dan pengembang di perusahaan saling memahami terkait domain, konteks, dan relasi antar-konteks. Oleh karena pembuatan *context map* ini juga perlu memastikan bahwa setiap *bounded context* dapat memfasilitasi komunikasi yang baik antar-departemen atau antar-tim di perusahaan, maka proses ini perlu dilakukan secara inkremental dan iteratif pula [22]. *Context map* yang dibuat menjadi landasan atau rancangan arsitektur *microservices* yang akan diimplementasikan ke sistem.

2.3. Tahap Migrasi

Dalam tahap ini, *context map* yang sudah didefinisikan sebelumnya menjadi fondasi dalam proses dekomposisi sistem monolitik pada level implementasi. Sistem lawas genting akan dimigrasikan berdasarkan *context map* yang telah dibuat dan diuji coba terlebih dahulu pada *environment staging*. Proses *deployment* dapat dilakukan secara terisolasi di *cloud* maupun pada *data center* yang dimiliki oleh perusahaan.

2.3.1. Proses Migrasi ke Arsitektur Microservices

Pada sistem monolitik yang besar, proses identifikasi konteks dapat menjadi tantangan tersendiri. Memilih konteks dengan hubungan yang lebih sedikit dengan konteks lain dan membuat batasan di antara konteks tersebut dengan *bounded context* akan membantu dalam identifikasi *microservice*. Setiap *bounded context* berdiri sebagai sebuah *microservice*. *Microservice* yang dirancang harus bersifat independen untuk mengurangi komunikasi antar-tim yang tidak efektif dan efisien di dalam

perusahaan, serta kemudahan pengelolaan setiap *microservice* tanpa ketergantungan satu sama lain [17].

2.3.2. Proses Deployment

Setelah arsitektur *microservices* didefinisikan, sistem harus di-*deploy* untuk pengujian lebih lanjut. Oleh karena perusahaan *startup* cenderung memiliki sumber daya dan dokumentasi yang terbatas, sistem hasil migrasi perlu diuji terlebih dahulu dikarenakan pada umumnya perusahaan *startup* memiliki prioritas lebih tinggi untuk mengeluarkan fitur atau produk mereka daripada waktu untuk merancang dan mengelola sistem [15]. Untuk memastikan sistem yang dimigrasikan dapat diuji secara menyeluruh, sistem yang dimigrasikan harus di-*deploy* sementara ke *environment staging* untuk pengujian lebih lanjut. Perusahaan *startup* cenderung melakukan *deployment* sistem mereka pada layanan *cloud* [23], yang akan menjadi alasan utama mengapa ABizLSM juga mengadopsi layanan *cloud* untuk tujuan *deployment*.

2.4. Tahap Validasi

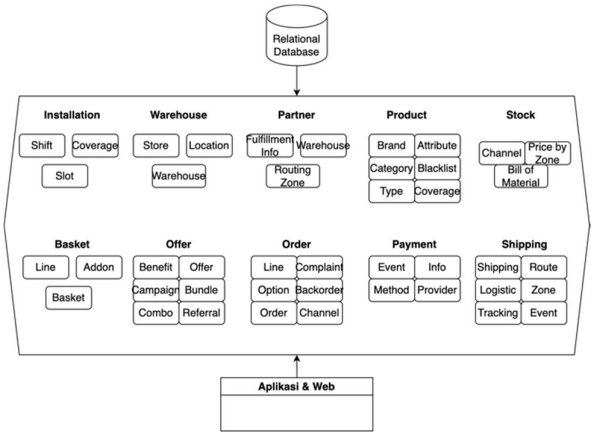
Setelah sistem dimigrasikan, maka tahap selanjutnya adalah melakukan validasi atas sistem dengan arsitektur *microservices* yang baru. Proses validasi berupa *testing* diperlukan untuk memastikan keseluruhan sistem hasil migrasi bekerja seperti sistem monolitik yang lama. Perusahaan *startup* cenderung kurang memiliki dokumentasi dan *testing* yang baik atas sistem nya karena berfokus pada pemasaran fitur atau produk yang cepat [15]. Melihat hal ini, kerangka ABizLSM juga mengusulkan proses *testing* keseluruhan sistem yang telah dimigrasikan, melalui pendefinisian ulang *unit test*, *integration test*, dan *end-to-end test*. Dengan adanya *testing* terhadap sistem hasil migrasi, hal ini memastikan sistem hasil migrasi tidak hanya memiliki arsitektur baru yang mudah dikelola, tetapi juga memiliki kualitas yang baik dalam mendukung proses bisnis perusahaan. Ketika sistem baru telah melakukan semua pengujian yang diperlukan, sistem baru tersebut dapat digunakan untuk dijalankan secara paralel dengan sistem lama selama beberapa waktu sebelum sistem baru tersebut menggantikan sistem lama sepenuhnya.

3. STUDI KASUS SISTEM LAWAS GENTING

Kerangka ABizLSM yang diusulkan dalam penelitian ini diterapkan dan divalidasi pada PT XYZ, salah satu perusahaan *startup* di Indonesia yang bergerak di bidang *home & living*. PT XYZ memiliki 3 kegiatan bisnis utama yang terdiri dari penyediaan furnitur, jasa desain interior, dan jual-beli properti hunian. Ketiga kegiatan bisnis tersebut saling terhubung satu sama lain dalam sistem milik PT XYZ, dimulai dari konsumen dapat mencari jual-beli rumah melalui sistem, melakukan dekorasi ulang interior rumah, yang produknya juga tersedia dan terintegrasi langsung pada *e-commerce* milik PT XYZ. Sejak bulan Oktober 2024, PT XYZ sudah memiliki sekitar 30 toko cabang yang tersebar di seluruh Indonesia. Transaksi baik dari *e-commerce* maupun langsung dari toko juga sudah terintegrasi satu sama lain, bersama dengan proses pemenuhan pesanan dan rantai suplai di belakang nya. Hal ini yang menyebabkan sistem *e-commerce* milik PT XYZ bersifat strategis bagi perusahaan.

Sistem lawas *e-commerce* milik PT XYZ disusun sebagai satu aplikasi web dengan *database* tunggal yang dibangun dengan *framework* Django di atas bahasa pemrograman Python. Di masa awal PT XYZ berdiri, sebuah *framework e-commerce* berbasis Django bernama Oscar digunakan oleh PT XYZ untuk mendukung kegiatan awal perusahaan saat itu. Django Oscar sudah memiliki fungsionalitas *e-commerce* tanpa perlu banyak modifikasi. Di sisi lain, para pengembang awal juga memiliki tingkat familiaritas yang tinggi dengan bahasa pemrograman Python. Sistem *e-commerce* PT XYZ secara dasar memiliki tiga komponen utama, yaitu (i) *View*, (ii) *Model* dan (iii) *Service*. Komponen *View* bertugas sebagai bagian atau antarmuka yang menyambungkan *user* dengan sistem PT XYZ. Logika bisnis pada sistem PT XYZ menjadi tanggung jawab komponen *Service* dengan komponen *Model* sebagai representasi entitas yang ada di sistem.

Untuk mengakomodasi pertumbuhan bisnis, PT XYZ terus melakukan kustomisasi dan perubahan pada sistem yang berbasis Django Oscar agar sesuai dengan kebutuhan bisnisnya. Seiring berjalannya tahun, sistem tersebut menjadi semakin besar dan mulai menunjukkan tanda-tanda permasalahan, seperti sulit dikelola dan banyak versi-versi dari dependensi yang digunakan sudah kedaluwarsa. Dengan adanya kebutuhan bisnis baru, terkadang diperlukan dependensi dengan versi yang lebih baru agar dapat diimplementasikan ke dalam sistem. Di sisi lain, sistem *e-commerce* ini tidak dapat langsung diperbaharui karena kode-kode kustomisasi yang sudah terlalu banyak dan memerlukan penyesuaian yang besar untuk diperbaharui yang dapat meningkatkan resiko menghambat operasional perusahaan. Permasalahan lain yang ditemui secara teknikal adalah penulisan kode yang kurang baik dan komponen yang terlalu bergantung satu sama lain yang menyebabkan sistem lawas *e-commerce* milik PT XYZ sulit dikelola dan diperbaharui. Komponen-komponen seperti *View*, *Model*, dan *Service* juga diberikan penamaan yang kurang baik dan ambigu. Permasalahan-permasalahan tersebut membuat sedikit perubahan pada sistem lawas PT XYZ dapat berdampak besar bagi kestabilan performa sistem. Secara umum, arsitektur monolitik dan fitur-fitur yang terdapat pada sistem lawas milik PT XYZ ditunjukkan pada Gambar 2.



Gambar 2. Arsitektur Monolitik Sistem Lawas PT XYZ

Permasalahan yang disebutkan sebelumnya membuat sistem lawas PT XYZ menjadi semakin kompleks seiring

 https://doi.org/10.25077/TEKNOSI.v11i1.2025.27-36

berkembangnya PT XYZ. PT XYZ berencana untuk melakukan *re-engineering* atas sistem lawas tersebut dengan menjalankan sistem hasil migrasi paralel dengan sistem yang lama untuk sementara waktu, sebelum sistem monolitik sepenuhnya diberhentikan. ABizLSM diusulkan atas dasar masalah demikian, yang juga terjadi pada kebanyakan perusahaan *startup* di dunia [24] [25].

4. HASIL

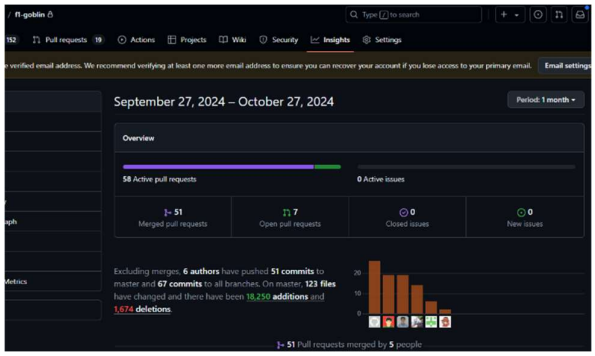
Bagian ini menjelaskan aplikasi dari tahapan ABizLSM pada sistem lawas genting milik PT XYZ dalam bentuk studi kasus. Langkah-langkah migrasi dijelaskan secara detail pada bagian ini, termasuk apa saja yang berubah pada setiap tahap kerangka ABizLSM. Sistem lawas milik PT XYZ dibangun di atas kerangka Django Oscar, sehingga replikasi minimal dari Django Oscar digunakan untuk keperluan demonstrasi dalam penelitian ini. Setiap tahapan migrasi selanjutnya akan divalidasi oleh perwakilan sisi domain bisnis dan pengembang di PT XYZ.

4.1. Tahap Pemahaman Sistem

Dalam tahap ini, sistem lawas genting milik PT XYZ perlu dipelajari lebih lanjut sebelum dirancang dalam bentuk arsitektur *microservices*. Hal ini dikarenakan PT XYZ tidak memiliki dokumentasi yang lengkap terkait sistem mereka. Analisis *source code* dan struktur *database* perlu dilakukan secara iteratif dan inkremental untuk memastikan semua detail fungsional teridentifikasi. Proses ini dapat terus diulang hingga dirasa sudah cukup atau tidak ada detail baru lagi yang diidentifikasi.

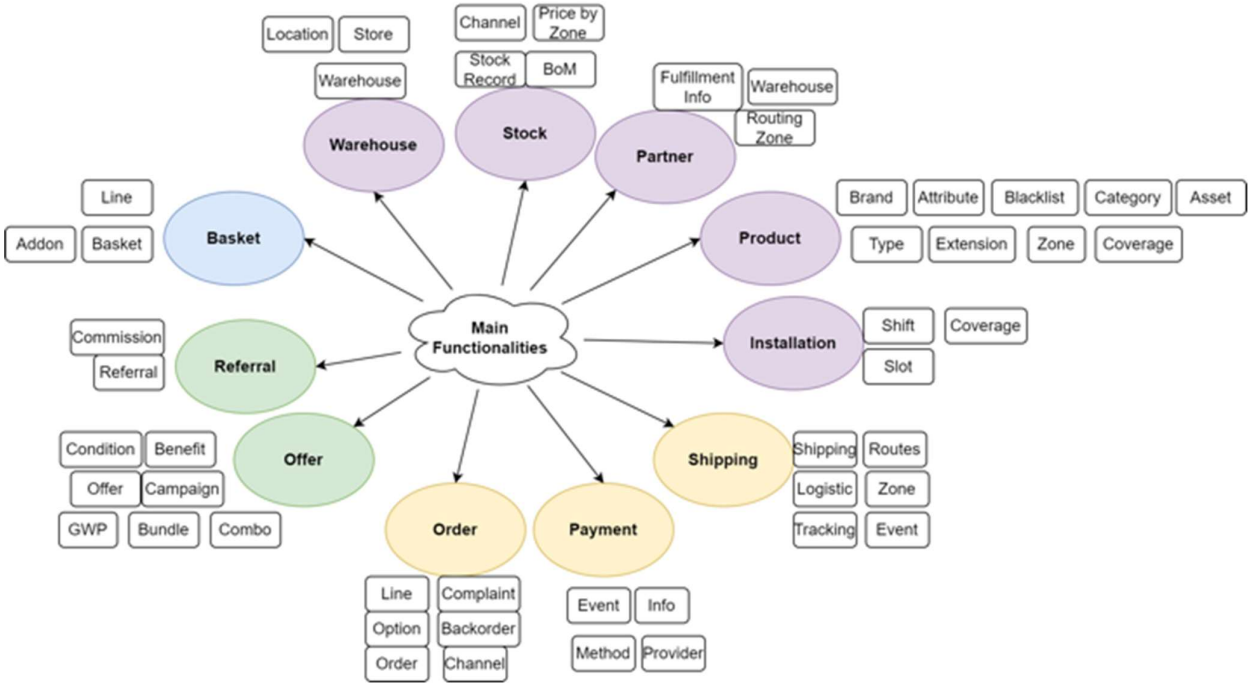
4.1.1. Analisis Source Code dan Database

Sistem lawas PT XYZ disimpan dalam repositori kode GitHub, dengan 741159 baris kode, 1600 *file*, dan 512 tabel *database*. Sistem menggunakan bahasa pemrograman Python 2 yang berjalan di atas *framework* Django Oscar, yang mengikuti *design pattern* Model-View-Template (MVT). Sistem ini terdiri dari tiga komponen utama, yaitu *Model*, *View*, dan *Service*. Komponen *View* berperan sebagai komponen *Template* pada MVT yang bertugas sebagai jembatan interaksi antara sistem-sistem (*Application Programming Interface* / API) atau sistem-manusia (*user interface* / UI), serta menyimpan abstraksi logika bisnis. Logika bisnis secara detail disimpan pada komponen *Service* dengan komponen *Model* sebagai representasi entitas di dalam sistem.



Gambar 3. Aktivitas Repositori GitHub Sistem Lawas PT XYZ

Selama periode 1 bulan (dari 27 September 2024 hingga 27 Oktober 2024), terpantau aktivitas penambahan kode sejumlah 18250 baris kode dan penghapusan sejumlah 1674 baris kode, seperti yang ditunjukkan pada Gambar 3. Dalam periode tersebut pula, terdapat sejumlah 51 *pull request* yang berhasil di-*merge* ke cabang *master* di repositori GitHub. Hal ini menunjukkan seberapa cepat sistem berubah untuk menyesuaikan kebutuhan bisnis PT XYZ.



Gambar 4. Fungsionalitas Utama Sistem Lawas PT XYZ

4.1.2. *Identifikasi Fungsionalitas Utama*

Setelah memahami *source code* dan tabel *database* sistem, tahap selanjutnya adalah identifikasi fungsionalitas utama. Proses identifikasi dilakukan dengan menelusuri semua entitas bisnis yang terdaftar di sistem [26]. Entitas ini dikelompokkan dan diabstraksikan menjadi entitas domain bisnis utama untuk membantu memahami fungsionalitas utama pada sistem. Gambar 4 menunjukkan 11 fungsionalitas utama pada sistem lawas milik PT XYZ yang dilakukan secara iteratif dan inkremental.

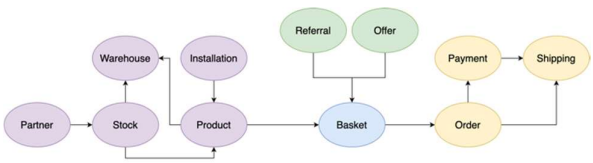
Proses identifikasi dilakukan dengan melihat relasi antar konteks bisnis dari setiap entitas yang ada di *source code* dan tabel *database* secara *bottom-up*. Setelah semua entitas diidentifikasi, pengelompokkan fungsionalitas utama dilihat dari bagaimana kemiripan penamaan entitas di sistem, bagaimana hubungan antar-entitas (dalam bentuk *foreign key*), dan logika bisnis pada *source code* [19]. Sebagai contoh, jika beberapa tabel *database* memiliki prefiks yang serupa, seperti *ShippingRoute* dan *ShippingArea*, maka kedua tabel tersebut dikelompokkan sebagai satu fungsionalitas utama, yaitu terkait pengiriman (*Shipping*). Selain itu, beberapa tabel seperti *Brand*, *Attribute*, *Category*, dan lainnya memiliki relasi pada satu tabel yang sama, yaitu terkait manajemen produk (*Product*). Fungsionalitas ini menjadi dasar dalam pembentukan domain di tahap berikutnya.

4.2. *Tahap Analisis dan Perancangan*

Proses identifikasi fungsionalitas utama hanya membantu memahami sistem secara garis besar. Pendekatan *Domain-Driven Design* (DDD) akan membantu untuk menggali lebih dalam terkait karakteristik sistem. Tahap ini juga perlu dilakukan secara iteratif dan inkremental yang melibatkan kolaborasi antara pengembang dan perwakilan ahli domain bisnis di perusahaan. Hal ini bertujuan untuk memastikan tidak ada informasi yang terlewat ketika arsitektur *microservices* dari sistem akan dirancang.

4.2.1. *Identifikasi Domain dan Bounded Context*

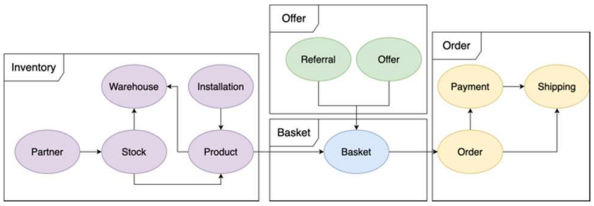
Fungsionalitas utama pada tahap sebelumnya diidentifikasi melalui entitas yang terdaftar pada sistem sehingga dapat merepresentasikan domain bisnis. Domain bisnis yang diidentifikasi sudah mencakup keseluruhan domain bisnis yang ada di sistem lawas milik PT XYZ. Seperti yang terlihat pada Gambar 4, terdapat 4 domain bisnis dengan 11 subdomain pada sistem lawas PT XYZ. Domain bisnis secara umum terdiri dari 4 domain utama: (i) Pengelolaan Produk (*Inventory*), (ii) Diskon dan Penawaran (*Offer*), (iii) Keranjang Belanja (*Basket*), dan (iv) Pemenuhan Pesanan (*Order Fulfillment*). Relasi antar-domain secara lebih lanjut ditunjukkan pada Gambar 5.



Gambar 5. Identifikasi Domain

Bounded context diidentifikasi berdasarkan domain bisnis utama yang dipetakan terhadap unit fungsional atau divisi yang ada di perusahaan PT XYZ, seperti yang terlihat pada Gambar 6. Terdapat tiga departemen utama yang berinteraksi dengan sistem lawas PT XYZ dalam menjalankan kegiatan operasionalnya, yaitu departemen *Production* (terkait produksi barang / *Inventory*), departemen *Marketing* (terkait penawaran, promosi, dan diskon / *Offer*) dan departemen *Supply Chain* (terkait pemenuhan pesanan dan penyediaan barang / *Order Fulfillment*). Selain itu, terdapat pula konteks keranjang belanja (*Basket*) yang digunakan oleh konsumen (untuk pembelanjaan secara daring) dan karyawan toko di lokasi (untuk pembelanjaan secara luring).

Pada konteks *Basket* dan *Order*, ada kebutuhan untuk memisahkan entitas *Product* dan *Offer* dalam konteks *Basket* dan *Order*. Keduanya merujuk pada entitas objek dunia nyata yang sama dengan representasi yang berbeda dalam sistem, hal ini yang menyebabkan *Product* dan *Offer* terdapat pula dalam konteks *Basket* dan *Order*. Departemen yang bekerja dalam konteks *Inventory* perlu menentukan spesifikasi *Product* yang terperinci, demikian pula untuk spesifikasi *Offer* dalam konteks *Offer*. Di sisi lain, *Basket* dan *Order* tidak memerlukan representasi *Product* dan *Offer* yang sama, karena keduanya hanya memerlukan informasi spesifik yang terkait dengan *Basket* dan *Order*. Misalnya, tim yang bekerja dalam konteks *Order* hanya ingin mengetahui nama produk dan jumlah produk yang dibeli.



Gambar 6. Identifikasi *Bounded Context* dan *Context Map*

4.2.2. *Pembuatan Context Map*

Setelah *bounded context* didefinisikan, tahap selanjutnya adalah menghubungkan relasi antar-*bounded context* untuk membentuk *context map*. Hal ini dilakukan untuk memahami dependensi antar-konteks lebih lanjut. Pada Gambar 6, konteks *Order* bergantung pada apakah ada pesanan yang dibuat dari keranjang belanja (*Basket*), sedangkan konteks *Basket* bergantung pada produk (*Product*) apa yang ditambahkan dan diskon (*Offer*) apa yang dipakai.

Ketika ahli domain bisnis dan pengembang menemukan suatu hal yang baru, maka langkah perlu diulangi kembali dari mendefinisikan domain, *bounded context*, hingga memperbaharui *context map*. Dengan demikian, hal ini memastikan kedua pihak memahami terkait domain bisnis dan relasinya. *Context map* yang sudah final menjadi fondasi dalam merancang arsitektur *microservices* di tahap berikutnya.

4.3. *Tahap Migrasi*

Dalam tahap ini, *context map* yang sudah didefinisikan sebelumnya menjadi fondasi dalam proses dekomposisi sistem monolitik pada level implementasi. Setiap *bounded context* berdiri sebagai sebuah *microservice* [17]. Setelah sistem baru

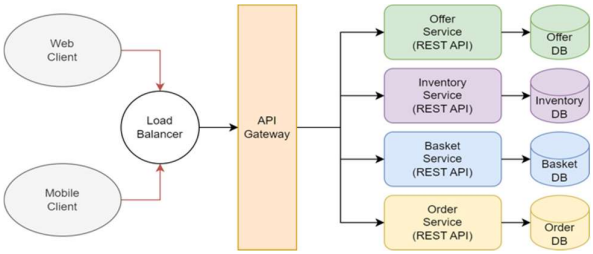
https://doi.org/10.25077/TEKNOSI.v11i1.2025.27-36

dengan arsitektur *microservices* sudah terbentuk, maka selanjutnya di-*deploy* pada *environment staging*. Setelah itu, sistem hasil migrasi diuji kesesuaiannya dengan menjalankan *unit testing*, *integration testing*, dan *end-to-end flow testing*.

Proses pembentukan *microservices* juga perlu memerhatikan independensi dalam pengelolaan dan kemudahan komunikasi departemen dalam organisasi. Kemudahan pengelolaan sistem juga menjadi salah satu harapan dari pemecahan sistem lawas monolitik ini. Di sisi lain, perancangan *context map* pada tahap sebelumnya juga sudah memerhatikan faktor komunikasi antar-departemen di PT XYZ. Dengan demikian, proses migrasi ke arsitektur *microservices* dapat dilakukan.

4.3.1. *Proses Migrasi ke Arsitektur Microservices*

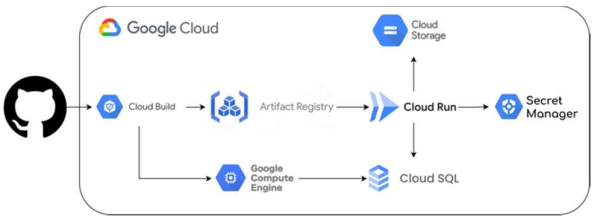
Proses migrasi dilakukan dengan membuat *microservice* untuk masing-masing *bounded context* yang telah didefinisikan. Berdasarkan hasil pada tahap sebelumnya, didapatkan 4 *bounded context* yang masing-masing menjadi sebuah *microservice* secara mandiri, yaitu *Inventory Service*, *Offer Service*, *Basket Service*, dan *Order Service*. Hasil rancangan arsitektur *microservices* yang siap diimplementasikan terlihat pada Gambar 7.



Gambar 7. Arsitektur Sistem Hasil Migrasi

4.3.2. *Proses Deployment*

Untuk kebutuhan *deployment* sistem hasil migrasi pada *environment staging*, penelitian ini menggunakan layanan *cloud* yang sama seperti yang digunakan di PT XYZ, yaitu Google Cloud Platform (GCP) menggunakan CI/CD. Secara umum, proses CI/CD terdiri dari 3 tahap utama, yaitu menjalankan *testing*, mengunggah *source code* ke Google Cloud Artifact Registry, dan melakukan perubahan pada tabel *database* (jika ada) pada Google Cloud SQL. Proses perubahan pada *database* dibantu melalui sebuah mesin virtual pada Google Compute Engine yang berjalan secara otomatis. Proses *deployment* secara umum terlihat pada Gambar 8.

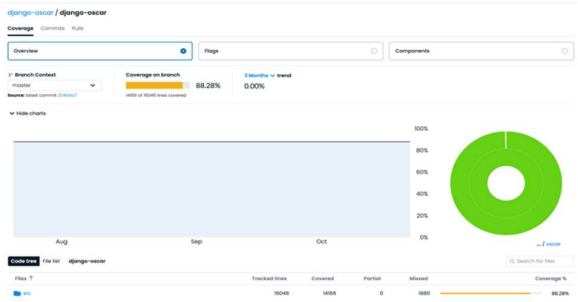


Gambar 8. Proses *Deployment*

Setiap *microservice* berjalan di atas sebuah *container* yang berjalan di Google Cloud Run yang membaca versi *image source code* dari Google Cloud Artifact Registry. Setiap kali ada perubahan versi *image*, maka Google Cloud Run akan melakukan pembaharuan pada *container*-nya. Oleh karena setiap *microservice* pada Google Cloud Run bersifat *stateless* (tidak ada data terkait aplikasi yang disimpan untuk periode tertentu), maka digunakan pula Google Cloud Storage (penyimpanan statis), Google Cloud Secret Manager (pengelolaan variabel sistem), dan Google Cloud SQL (sebagai *database*).

4.4. *Tahap Validasi*

Tahap terakhir adalah menjalankan *unit testing*, *integration testing*, dan *end-to-end flow testing*. Oleh karena dalam penelitian ini proses migrasi hanya terbatas untuk fitur-fitur utama pada Django Oscar yang juga dipakai oleh PT XYZ dalam sistem lawas milik PT XYZ (tanpa ada modifikasi logika bisnis secara khusus), maka sistem lawas dalam penelitian ini direplikasi dan disesuaikan (tidak menyimpan logika bisnis sendiri), serta hanya menggunakan logika bisnis dari Django Oscar. Terlihat pada tanggal 27 Oktober 2024, Django Oscar sendiri sudah lolos semua unit testing dan integration testing dengan code coverage 88,28%, seperti yang tampak pada Gambar 9.



Gambar 9. *Code Coverage* Django Oscar

Untuk menguji kesesuaian sistem hasil migrasi terhadap kebutuhan bisnis, dilakukan pula *end-to-end flow testing* untuk melihat apakah sistem yang sudah dimigrasikan tetap dapat berjalan sesuai dengan sistem lawas ketika sebelum dimigrasikan. Pengujian ini dilakukan dengan melakukan penambahan produk, mengubah barang pada keranjang belanja, penambahan kupon diskon, dan melakukan pembuatan pesanan (*checkout*). Pengujian *end-to-end flow testing* berjalan sesuai dengan ekspetasi, dengan hasil pengujian sistem dideskripsikan lebih lanjut pada Tabel 1.

5. PEMBAHASAN

Pada bagian ini dijelaskan terkait evaluasi sistem lawas yang telah dimigrasikan dengan kerangka ABizLSM beserta evaluasi proses migrasi dari kerangka ABizLSM itu sendiri. Evaluasi sistem dilakukan melalui uji perbandingan *latency* antara sistem lawas sebelum dan sesudah dimigrasikan. Selain itu, untuk mengevaluasi proses migrasi menggunakan kerangka ABizLSM dilakukan melalui wawancara dengan perwakilan sisi ahli bisnis dan sisi pengembang di PT XYZ.

5.1. *Evaluasi Hasil Testing*

Sistem lawas *e-commerce* milik PT XYZ bersifat *customer-centric*, yang berfokus pada kenyamanan pengguna. Oleh karena

sistem lawas dimigrasikan ke arsitektur *microservices*, latensi menjadi hal yang perlu diperhatikan dalam arsitektur *microservices* [27]. Dalam interaksi manusia dengan komputer atau *Human-Computer Interaction* (HCI), latensi atau *delay* ketika pengguna menggunakan sistem dapat memengaruhi kenyamanan atau pengalaman mereka selama menggunakan sistem karena latensi adalah faktor fundamental pada HCI [28]. Oleh karena itu, uji latensi dilakukan sebagai tolak ukur performa sistem hasil migrasi terhadap sistem sebelum dimigrasikan. Secara umum, sistem dengan arsitektur *microservices* akan memiliki latensi lebih tinggi dibandingkan sistem monolitik [29] [30]. Dengan demikian, latensi yang diharapkan adalah mendekati latensi sistem monolitik. Selain itu, *end-to-end flow testing* juga dilakukan bersamaan dengan uji latensi untuk memastikan semua fitur berjalan sesuai ekspetasi.

Tabel 1. Uji Latensi dan *End-to-End Flow Testing*

Skenario	**Latensi Monolitik (*ms*)**	**Latensi *Microservices* (*ms*)**
1. Melihat keranjang	314	691
2. Melihat produk	261	264
3. Menambahkan produk 1	423	574
4. Melihat detail keranjang	410	461
5. Menambahkan produk 2	440	572
6. Melihat detail keranjang	367	569
7. Menambahkan diskon	417	575
8. Melihat detail keranjang	387	715
9. Melakukan *checkout*	683	1136
10. Melihat pesanan	384	722

Sistem lawas PT XYZ melayani kegiatan jual-beli pada *e-commerce* milik PT XYZ, sehingga proses *testing* dilakukan pada alur utama dari sisi pengguna, seperti penambahan produk dan diskon pada keranjang belanja, mendapatkan informasi tentang keranjang belanja, melakukan *checkout*, dan melihat informasi tentang pesanan. Berdasarkan hasil pada Tabel 1, terlihat bahwa secara umum latensi dari sistem hasil migrasi lebih tinggi dibandingkan sistem lawas monolitik. Hal ini dikarenakan dalam arsitektur *microservices*, setiap layanan perlu berkomunikasi satu sama lain terlebih dahulu, yang mana tidak terjadi pada arsitektur monolitik karena dapat secara mandiri melaksanakan tugasnya. Hal ini selaras dengan hasil yang ditunjukkan pada Tabel 1, bahwa terdapat kenaikan latensi sekitar 1,01 hingga 2,20 kali pada arsitektur *microservices*. Jika dilihat lebih lanjut, kenaikan latensi tidak lebih dari 500 ms pada setiap skenario, yang mana masih sangat mendekati latensi dari arsitektur monolitik. Dengan demikian, terlihat bahwa sistem hasil migrasi dari kerangka ABizLSM dapat membantu proses migrasi sistem menuju ke arsitektur *microservices*.

5.2. Evaluasi Proses Migrasi

Tahapan dan proses migrasi dari sistem lawas milik PT XYZ menuju ke arsitektur *microservices* juga dievaluasi lebih lanjut melalui wawancara dengan perwakilan sisi ahli bisnis dan sisi pengembang di PT XYZ. Menggunakan prinsip *Domain-Driven Design* (DDD), perwakilan sisi domain ahli bisnis terdiri dari dua orang senior *Product Manager*, sedangkan perwakilan dari sisi pengembang terdiri dari seorang senior *Platform Software Developer*. Proses evaluasi berlangsung melalui wawancara secara kolaboratif dengan masing-masing pihak.

Dari perspektif narasumber kedua *Product Manager* dan *Software Engineer*, proses identifikasi fungsionalitas utama pada sistem lawas sudah mencakup semua kebutuhan bisnis dalam PT XYZ. Semua narasumber memberikan konfirmasi bahwa ada empat domain bisnis dan 11 subdomain bisnis dalam sistem lawas XYZ, seperti yang telah diidentifikasi sebelumnya. Meskipun semua narasumber setuju mengenai hubungan eksternal dari *bounded context*, narasumber *Product Manager* memberikan umpan balik berkaitan dengan hubungan internal di dalam konteks *Inventory* dan konteks *Order*. Narasumber *Product Manager* berpendapat bahwa *Stock* dan *Installation* (dalam konteks *Inventory*) seharusnya terhubung langsung ke konteks *Basket*, karena *Product* adalah abstraksi untuk keduanya. Selain itu, dalam konteks *Order*, narasumber *Product Manager* mengatakan bahwa *Payment* tidak diperlukan untuk *Shipping*, dan keduanya dapat beroperasi secara independen.

Di sisi lain, umpan balik dari narasumber *Platform Software Engineer* adalah terkait granularisasi dari *bounded context*. Narasumber mengatakan bahwa pada konteks *Inventory* dapat dibuat granular lebih lanjut dengan memisahkan *Product* dan *Stock*, serta *Order* dan *Shipping* di dalam konteks *Order*. Selain itu, narasumber juga memberikan tanggapan bahwa *Installation* seharusnya berada pada konteks *Order*. Walaupun demikian, *Installation* juga diperlukan untuk mengetahui apakah keranjang belanja (*Basket*) yang ada pada konsumen juga memesan jasa rakit atau instalasi pemasangan. Hal yang serupa terjadi pada *Offer*, yang mana menurut narasumber *Platform Software Engineer*, *Offer* seharusnya juga terhubung pada *Order* namun masih tetap diperlukan di *Basket* saat ini. Selain itu, *Warehouse* pada konteks *Inventory* juga hanya sebagai referensi. Semua hal ini menunjukkan bahwa dalam pembentukan *Bounded Context* untuk sistem lawas PT XYZ, terdapat satu entitas atau *domain* yang memiliki banyak representasi di berbagai konteks.

Untuk usulan desain arsitektur *microservices*, semua narasumber setuju bahwa tidak perlu adanya *API Gateway*. Menurut narasumber *Software Engineer*, *load balancer* yang semula diletakkan di depan *API Gateway* dapat diletakkan di masing-masing *microservices*. Selain itu, semua narasumber juga menyebutkan bahwa diperlukan pengembangan sistem *monitoring* dan *logging* yang tepat untuk memantau keseluruhan sistem ini nantinya. Dengan sistem *monitoring* dan *logging* yang terpusat, proses penelurusan masalah dapat menjadi lebih mudah dengan melihat urutan waktu kejadian (*timestamp*) di setiap *microservice*. Untuk melakukan migrasi sistem lawas PT XYZ saat ini menuju ke rancangan arsitektur yang diusulkan, semua narasumber setuju bahwa proses migrasi perlu dilakukan secara bertahap melalui satu *bounded context* terlebih dahulu. Proses migrasi ini juga memerlukan duplikasi data dan sistem yang berjalan paralel dengan sistem lawas monolitik yang lama untuk memastikan proses migrasi aman dan lancar.

6. KESIMPULAN

Penelitian ini mengusulkan sebuah kerangka migrasi sistem lawas genting dari arsitektur monolitik menuju arsitektur *microservices* pada lingkungan perusahaan *startup* dengan

https://doi.org/10.25077/TEKNOSI.v11i1.2025.27-36

dinamika bisnis yang tinggi, yaitu ABizLSM (*Agile Business Legacy System Migration*). Keranga proses migrasi ABizLSM dirancang berdasarkan prinsip *Domain-Driven Design* (DDD), yang berfokus pada kolaborasi dari pihak ahli domain bisnis dan pengembang. Selain itu, penelitian ini juga menyajikan hasil bagaimana ABizLSM diterapkan dan dievaluasi pada salah satu perusahaan *startup* di Indonesia yang bergerak di bidang *home & living*. Hasil penelitian menunjukkan bahwa kerangka ABizLSM berhasil memigrasikan sistem lawas di PT XYZ dengan *maintainability* dan *scalability* yang baik. Sistem hasil migrasi juga menunjukkan tidak ada tanda-tanda penurunan performa sistem yang dievaluasi melalui uji latensi dan *end-to-end flow testing*.

Saat ini, kerangka ABizLSM baru diterapkan pada PT XYZ. Proses migrasi pada kerangka ABizLSM diharapkan dapat membantu memberikan standar dan pemahaman melalui penerapannya di perusahaan-perusahaan *startup* dengan dinamika bisnis tinggi yang memiliki sistem lawas genting dengan arsitektur monolitik. Sebagai perkembangan untuk penelitian selanjutnya, kerangka ABizLSM akan dianalisis lebih lanjut pada studi kasus lain yang lebih luas untuk melihat proses migrasi apa yang dapat diperbaiki di masa mendatang. Penyebaran survei yang ditujukan bagi praktisi di perusahaan *startup* juga direncanakan untuk melihat kesesuaian proses migrasi ABizLSM di perusahaan *startup* lainnya untuk memperbaiki kerangka ABizLSM di masa depan.

DAFTAR PUSTAKA

[1] H. Sneed and C. Verhoef, "Re-implementing a legacy system," *Journal of Systems and Software*, vol. 155, pp. 162–184, 2019.

[2] J. Soldani, L. Luthmann, N. Gottwald, M. Lochau, and A. Brogi, "Compositional testing of management conformance for multi-component enterprise applications," *Service Oriented Computing and Applications*, vol. 16, no. 3, pp. 209–225, 2022.

[3] H. Bakar, R. Razali, and D. I. Jambari, "A guidance to legacy systems modernization," *Int J Adv Sci Eng Inf Technol*, vol. 10, no. 3, pp. 1042–1050, 2020.

[4] P. de Vrieze and L. Xu, "Resilience analysis of service-oriented collaboration process management systems," *Service oriented computing and applications*, vol. 12, no. 1, pp. 25–39, 2018.

[5] R. Cao and M. Iansiti, "Digital transformation, data architecture, and legacy systems," *Journal of Digital economy*, vol. 1, no. 1, pp. 1–19, 2022.

[6] P. Pääkkönen and D. Pakkala, "Mechanism and architecture for the migration of service implementation during traffic peaks," *Service Oriented Computing and Applications*, vol. 9, pp. 193–209, 2015.

[7] A. Immonen and D. Pakkala, "A survey of methods and approaches for reliable dynamic service compositions," *Service Oriented Computing and Applications*, vol. 8, pp. 129–158, 2014.

[8] M. Ali, S. Hussain, M. Ashraf, and M. K. Paracha, "Addressing software related issues on legacy systems–a review," *International journal of scientific & technology research*, vol. 9, no. 03, pp. 3738–3742, 2020.

[9] H. Abu Bakar, R. Razali, and D. I. Jambari, "Legacy systems modernisation for citizen-centric digital government: A conceptual model," *Sustainability*, vol. 13, no. 23, p. 13112, 2021.

[10] H. Calderón-Gómez *et al.*, "Evaluating service-oriented and microservice architecture patterns to deploy ehealth applications in cloud computing environment," *Applied Sciences*, vol. 11, no. 10, p. 4350, 2021.

[11] C. E. da Silva, Y. de L. Justino, and E. Adachi, "SPReaD: service-oriented process for reengineering and DevOps: Developing microservices for a Brazilian state department of taxation," *Service Oriented Computing and Applications*, vol. 16, no. 1, pp. 1–16, 2022.

[12] F. Tapia, M. Á. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From monolithic systems to microservices: A comparative study of performance," *Applied sciences*, vol. 10, no. 17, p. 5797, 2020.

[13] S. A. Zahra, "The resource-based view, resourcefulness, and resource management in startup firms: A proposed research agenda," *J Manage*, vol. 47, no. 7, pp. 1841–1860, 2021.

[14] M. F. Ramadhan and Z. Zukhri, "Pengembangan REST API Sistem UIIAdmisi dengan Menggunakan Pendekatan Domain Driven Design," *Jurnal Ilmiah Informatika*, vol. 11, no. 02, pp. 176–182, 2023.

[15] A. Nguyen-Duc, K.-K. Kemell, and P. Abrahamsson, "The entrepreneurial logic of startup software development: A study of 40 software startups," *Empir Softw Eng*, vol. 26, pp. 1–55, 2021.

[16] M. A. Arokodare and B. R. Falana, "Strategic agility and the global pandemic: The agile organizational structure, a theoretical review," *Information Management and Business Review*, vol. 13, no. 1 (I), pp. 16–27, 2021.

[17] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, "Does migrating a monolithic system to microservices decrease the technical debt?," *Journal of Systems and Software*, vol. 169, p. 110710, 2020.

[18] N.-B. Wang, W. Shen, C. Guo, and H.-P. Wan, "Moving load test-based rapid bridge capacity evaluation through actual influence line," *Eng Struct*, vol. 252, p. 113630, 2022.

[19] M. Abdellatif *et al.*, "A taxonomy of service identification approaches for legacy software systems modernization," *Journal of Systems and Software*, vol. 173, p. 110868, 2021.

[20] R. Ferdiana, S. Sulistyo, and others, "The role of information technology usage on startup financial management and taxation," *Procedia Comput Sci*, vol. 161, pp. 1308–1315, 2019.

[21] P. Johnson, "Is EventStorming effective in defining the bounded contexts used to break down monolithic software into microservices?," 2022.

[22] D.-H. Dang, D. M. Le, and V.-V. Le, "Agl: Incorporating behavioral aspects into domain-driven design," *Inf Softw Technol*, vol. 163, p. 107284, 2023.

[23] T. Abell, A. Husar, and L. May-Ann, "Cloud computing as a key enabler for tech start-ups across Asia and the Pacific," 2021.

[24] A. Behl, P. Dutta, S. Lessmann, Y. K. Dwivedi, and S. Kar, "A conceptual framework for the adoption of big

data analytics by e-commerce startups: a case-based approach," *Information systems and e-business management*, vol. 17, pp. 285–318, 2019.

[25] L. Huang, Y. Huang, R. Huang, G. Xie, and W. Cai, "Factors influencing returning migrants' entrepreneurship intentions for rural E-commerce: an empirical investigation in China," *Sustainability*, vol. 14, no. 6, p. 3682, 2022.

[26] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. El Fazziki, "A multi-model based microservices identification approach," *Journal of Systems Architecture*, vol. 118, p. 102200, 2021.

[27] Y. Gan and C. Delimitrou, "The architectural implications of cloud microservices," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 155–158, 2018.

[28] A. A. Laghari, X. Zhang, Z. A. Shaikh, A. Khan, V. V Estrela, and S. Izadi, "A review on quality of experience (QoE) in cloud computing," *J Reliab Intell Environ*, vol. 10, no. 2, pp. 107–121, 2024.

[29] N. Bjørndal *et al.*, "Migration from monolith to microservices: Benchmarking a case study," *Tech. Rep.*, 2020.

[30] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022.

BIODATA PENULIS

Michael Susanto
Michael Susanto adalah seorang mahasiswa magister teknologi informasi di Fakultas Ilmu Komputer, Universitas Indonesia. Ia memiliki fokus penelitian pada *resource management* dan komputasi di *cloud* dan sistem terdistribusi.

Muhammad Hafizhuddin Hilman
Muhammad Hafizhuddin Hilman adalah seorang dosen dan peneliti di Fakultas Ilmu Komputer, Universitas Indonesia. Ia memiliki bidang minat pada sistem komputer terkait *resource management* dan *scheduling* di *cloud* dan sistem terdistribusi. Secara khusus, Ia memiliki fokus penelitian pada sistem untuk *workflow management*, simulasi dan pemodelan *cloud*, *scientific computing*, dan *high-performance computing*.

 https://doi.org/10.25077/TEKNOSI.v11i1.2025.27-36