SinkrOn
JURNAL & PENELITIAN TEKNIK INFORMATIKA
Publikasi Jurnal & Penelitian Teknik Informatika
Volume 3 Nomor 1, Oktober 2018
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Analisis Model Arsitektur *Microservice* Pada Sistem Informasi DPLK

Ghifari Munawar
Jurusan Teknik Komputer dan Informatika
Politeknik Negeri Bandung
Bandung
ghifari.munawar@polban.ac.id

Ade Hodijah
Jurusan Teknik Komputer dan Informatika
Politeknik Negeri Bandung
Bandung
adehodijah@jtk.polban.ac.id

***Abstract*—** Penelitian didasari oleh keberhasilan proses migrasi dari sistem monolitik ke dalam arsitektur *microservice* pada beberapa penelitian sebelumnya melalui metode identifikasi kandidat *microservice*. Pada penelitian ini, peneliti melakukan identifikasi *microservice* pada sistem informasi Dana Pensiun Lembaga Keuangan (DPLK) melalui 6 (enam) tahapan migrasi yang dimulai dari analisa setiap subsistem monolitik terhadap fungsi bisnis dan tabel *database*. Sistem DPLK dikembangkan dengan arsitektur monolitik dan memiliki beberapa modul sistem, diantaranya sistem pendaftaran nasabah, sistem pengelolaan investasi, sistem pengelolaan bisnis, sistem pelaporan, dlsb. Banyaknya tanggung jawab yang perlu dikelola dalam sistem DPLK menyebabkan sistem akan sulit untuk beradaptasi terhadap perubahan kebutuhan. Kompleksitas inilah yang akan dipecah dalam beberapa *service* melalui pendekatan *microservices*. Sehingga diharapkan dengan mengembangkan model arsitektur *microservices* ini, sistem informasi DPLK dapat lebih adaptif (*adaptability*) terhadap perubahan kebutuhan (*requirement changes*) sistem untuk kedepannya.

Keywords*—microservice; requirement changes; adaptability*

I. PENDAHULUAN

Sistem informasi *enterprise* pada umumnya dibangun dengan pendekatan monolitik, dimana aplikasi terbungkus dalam satu *package* besar, dan ketika terjadi perubahan (*requirement changes*) pada salah satu bagian kode program, akan berpengaruh besar terhadap kode program lainnya. Saat ini pendekatan monolitik telah bergeser menjadi pendekatan terdistribusi, dimana aplikasi dibagi menjadi bagian-bagian kecil yang berfungsi spesifik (*high cohesion*) dan tidak bergantung pada komponen program lainnya (*loose coupling*), dengan dikembangkannya *service* melalui antarmuka API (*application programming interface*) [1]. Salah satu tantangan yang muncul pada arsitektur monolitik adalah kemampuan adaptasi terhadap perubahan kebutuhan sistem (*requirement changes*) terutama dalam pengelolaan kompleksitas kode, dan *maintainability*-nya, dimana hal ini akan menyebabkan *bottleneck* pada proses pendistribusiannya karena tingkat dependensi yang tinggi (*tightly coupling*) [2]. Oleh karenanya perlu mempertimbangkan alternatif arsitektur lain agar pengelolaan sistem kedepannya menjadi lebih adaptif terhadap perubahan (*requirement changes*).

Arsitektur *microservice* merupakan alternatif arsitektur yang lebih terukur dan lebih fleksible. Pada arsitektur *microservice*, sistem informasi dirancang untuk terdistribusi dan menyediakan layanan secara lebih fokus dan spesifik. Permasalahan besar akan dipecah menjadi beberapa solusi kecil yang disusun dalam satu *service*, dimana setiap *service* memiliki tanggung jawabnya sendiri [3]. Dengan pendekatan ini, suatu sistem informasi akan terdiri dari beberapa *service* yang dapat dikelola dan didistribusikan secara *independent*, hal ini akan lebih memudahkan sistem untuk beradaptasi terhadap perubahan kebutuhan.

Berdasarkan beberapa penelitian sebelumnya, (1) *Microservice* sebagai sebuah arsitektur sistem yang terdistribusi telah menunjukkan peningkatan kualitas pada aspek resiliensi dengan studi kasus sistem manajemen asosiasi/keanggotaan, evaluasi model dilakukan melalui pembuatan *proof of concept* dari modifikasi arsitektur *software* yang terdistribusi berbasis *microservice* dan *Docker-container* [4]. Kualitas resiliensi ini dibuktikan ketika beberapa *node*

service mengalami gangguan, sistem dapat tetap berjalan sebagaimana mestinya. Proses migrasi dari sistem berbasis monolitik menjadi sistem berbasis *microservice* dilakukan melalui 15 langkah (*Microservices Migration Patterns*). (2) Penelitian [6] telah mengembangkan metode migrasi dari arsitektur monolitik ke arsitektur *microservice* melalui 6 (enam) tahapan, dimana studi kasusnya adalah sistem perbankan yang mengelola 3,5 juta data nasabah dan hampir 2 juta transaksi perharinya. Identifikasi *microservice* telah berhasil dilakukan dengan menganalisa keterhubungan pada setiap subsistem (*SS*), fungsi bisnis (F,B), dan tabel *database* (D).

Penelitian ini mencoba mengadopsi metode migrasi yang diusulkan [6] untuk menganalisa model arsitektur *microservice* pada sistem informasi dana pensiun lembaga keuangan (DPLK) dengan tujuan agar dapat meningkatkan kemampuan adaptasi sistem informasinya. Proses bisnis DPLK mengacu kepada aturan pemerintah, yakni PP Nomor 77 tahun 1992 tentang Dana Pensiun Lembaga Keuangan. Sistem informasi DPLK dikembangkan dengan arsitektur monolitik dan memiliki beberapa modul sistem, diantaranya sistem pendaftaran nasabah, sistem pengelolaan investasi, sistem pengelolaan bisnis dan sistem pelaporan. Sistem ini mengelola data sensitif terkait data keuangan nasabah, aturan pencairan dana, historis transaksi, dll. Banyaknya tanggung jawab yang perlu dikelola dalam sistem menyebabkan sistem sulit untuk beradaptasi terhadap penyesuaian kebutuhan. Kompleksitas inilah yang akan dipecah dalam beberapa *service* dengan pendekatan *microservices*. Sehingga diharapkan dengan model ini, sistem informasi DPLK dapat lebih mudah beradaptasi terhadap perubahan kebutuhan untuk kedepannya.

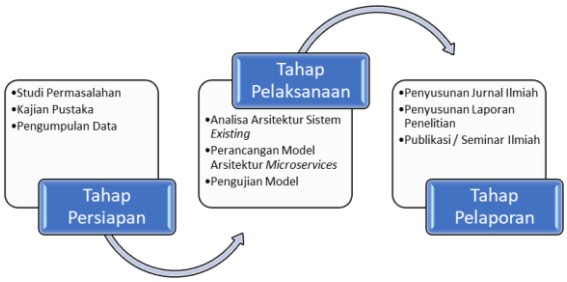
Alur proses pada tiap-tiap modul sistem digambarkan dengan model *flowchart*. Alur ini menggambarkan aliran proses yang terjadi pada suatu kegiatan. Secara umum modul pada sistem DPLK terbagi kedalam 5 (lima) subsistem, seperti terlihat pada Tabel I.

TABEL I. DAFTAR SUBSISTEM DAN MODUL SISTEM DPLK

Subsistem	Modul
Kepesertaan	Permohonan Pendaftaran Perusahaan
	Pendaftaran Peserta
	Pendaftaran Peserta Pindahan
	Upload Peserta Kolektif
	Penambahan Kode Peserta Baru
	Perubahan Data Peserta
Transaksi	Setor Tunai Iuran
	Upload Iuran
	Permohonan Pencairan
	Penarikan Iuran
	Pengalihan Ke DPLK Lain
	Cetak Buku Tabungan
	Cetak Rekening Koran
	Koreksi Iuran
Investasi	Penempatan Investasi
	Perpanjangan Penempatan Investasi
	Pencairan Investasi
	Open Business Date
	Close Business Date
	EOD
	EOM
	EOY
Pengelolaan	Pengelolaan Bisnis • Pengelolaan Produk • Pengelolaan Instrumen Keuangan • Pengelolaan Paket Investasi • Pengelolaan Kode Transaksi • Pengelolaan Biaya Administrasi • Pengelolaan Premi Asuransi • Pengelolaan Kalendar Hari Libur
	Pengelolaan Mitra • Pengelolaan Regulator • Pengelolaan Institusi Bank • Pengelolaan Institusi DPLK • Pengelolaan counterpart • Pengelolaan Institusi Agen Penjualan • Pengelolaan Cabang Agen Penjualan • Pengelolaan Agen Penjual • Pengelolaan Target Agen Penjual
	Pengelolaan User
	Pengelolaan API • Pengelolaan API • Pengelolaan Client • Pengelolaan Mapping API
	Pengelolaan Hak Akses • Pengelolaan Group Akses • Pengelolaan Group Menu • Pengelolaan User Akses
	Pengelolaan Framework • Pengelolaan Modul • Pengelolaan Menu • Pengelolaan Dependecies • Pengelolaan Jobs • Pengelolaan Codes • Pengelolaan System Options • Pengelolaan System Parameters
Laporan	Laporan Kepesertaan • Laporan Peserta DPLK • Laporan Kandidat Peserta Pensiun • Laporan Peserta Pensiun • Laporan Peserta Pindahan dari DPLK Lain
	Laporan Transaksi • Laporan Seluruh Transaksi • Laporan Iuran • Laporan Penarikan • Laporan Pencairan
	Laporan Dana Kelolaan • Laporan Penempatan • Laporan Jatuh Tempo • Laporan Pencairan

II. METODE PENELITIAN

Agar penelitian dapat dilaksanakan dengan baik, maka disusunlah tahapan-tahapan penelitian sebagai berikut :



GAMBAR I. METODOLOGI PENELITIAN

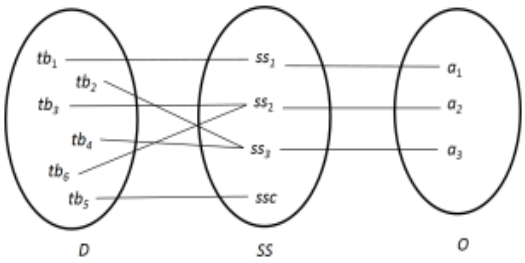
TABEL II. TAHAPAN PENELITIAN

Fase	Jenis Kegiatan	Keterangan	Target Luaran
Tahap Persiapan			
1	**Studi Permasalahan**	Studi permasalahan mengenai sistem informasi DPLK yang saat ini berjalan	Memahami kondisi permasalahan yang timbul dari sistem informasi DPLK yang saat ini berjalan.
2	**Kajian Pustaka**	Kajian pustaka dari berbagai sumber terkait proses analisis, perancangan arsitektur sistem, *microservice*, teknologi *cloud*, dll	Memahami lingkup teoritis dan praktis untuk solusi permasalahan dalam penelitian.
3	**Pengumpulan Data**	Melakukan pengumpulan data pada objek penelitian, berupa dokumentasi teknis, petunjuk penggunaan sistem, dll.	Dokumen Teknis, Petunjuk Penggunaan Sistem, *Source Code* Aplikasi *Existing*.
Tahap Pelaksanaan			
4	**Analisa Arsitektur Sistem *Existing***	Menganalisa model arsitektur monolitik yang sesuai dengan sistem informasi DPLK yang saat ini berjalan.	Arsitektur Monolitik pada Sistem *Existing*.
5	**Perancangan Model Arsitektur *Microservice***	Merancang arsitektur *microservice* berdasarkan hasil analisis	Model Arsitektur *Microservice* yang dikembangkan
6	**Pengujian Model**	Menguji model arsitektur yang dikembangkan	Hasil Pengujian Model
Tahap Pelaporan			
7	**Penyusunan Jurnal Ilmiah**	Menyusun jurnal ilmiah sesuai dengan hasil temuan pada penelitian.	Draft Jurnal Ilmiah
8	**Penyusunan Laporan Penelitian**	Membuat laporan penelitian, baik laporan progress, dan laporan akhir serta pertanggung jawaban dana.	Laporan Kemajuan Penelitian, Laporan Akhir Penelitian, dan Laporan Pertanggung Jawaban Dana Penelitian.
9	**Publikasi / Seminar Ilmiah**	Melakukan publikasi / seminar ilmiah	Jurnal Ilmiah yang sudah dipublikasikan.

III. TAHAPAN MIGRASI

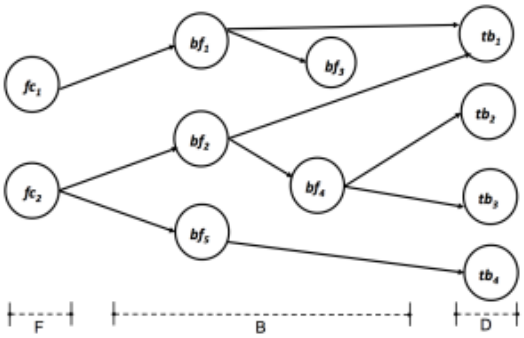
Terdapat 6 (enam) tahapan migrasi untuk melakukan identifikasi *microservice* dari sistem monolitik. Berikut tahapannya [6]:

a) Petakan tabel *database* $tb_1 \in D$ kedalam subsistem $ss_1 \in S$. Setiap subsistem mewakili sebuah area bisnis (a_i) dari organisasi O.



GAMBAR II. PEMETAAN TABEL DATABASE PADA SUBSISTEM [6]

b) Buat grafik ketergantungan (V, E) dimana *node* (*vertices*) menggambarkan *facades* ($fc_i \in F$), fungsi bisnis ($bf_i \in B$), atau tabel *database* ($tb_i \in D$), dan garis (*edges*) menggambarkan: (i) panggilan dari *facades* ke fungsi bisnis; (ii) panggilan diantara fungsi bisnis; dan (iii) akses dari fungsi bisnis ke tabel *database*. Satu subsistem bisa memiliki lebih dari satu *facades* dan satu *facades* bisa memanggil lebih dari satu fungsi bisnis dan satu fungsi bisnis bisa memanggil lebih dari satu fungsi bisnis lainnya atau memanggil lebih dari satu tabel *database*. Dimana *facades* merupakan *entry point* dari sistem yang memanggil fungsi bisnis dan fungsi bisnis merupakan *methods* yang mewakili implementasi sebuah *business rules* dan bergantung pada tabel *database*.



GAMBAR III. GRAFIK KETERGANTUNGAN ANTAR *VERTICES* (V) [6]

c) Identifikasi pasangan (fc_i, tb_j) dimana $fc_i \in F$ dan $tb_j \in D$, dan terdapat jalur (*path*) dari fc_i ke tb_j pada grafik ketergantungan.

d) Pilih pasangan (fc_i, tb_j) yang diidentifikasi pada langkah sebelumnya untuk setiap subsistem ss_i pada langkah ke-1, dimana $tb_j \in ss_i$.

e) Identifikasi kandidat yang akan diubah menjadi *microservice*. Untuk setiap pasangan (fc_i, tb_j), sebuah kandidat *microservice* (M) didefinisikan sebagai berikut:

- *Name*: nama *service* sesuai pola [subsystemname].[processname].
- *Purpose*: satu kalimat yang menggambarkan tujuan bisnis utama dari operasi, yang secara langsung terkait dengan domain entitas *database* yang diakses.
- *Input/Output*: operasi pada *microservice* memerlukan data sebagai *input* untuk menghasilkan *output*.
- *Features*: *business rules* untuk penggunaan *microservice*, yang digambarkan sebagai kata kerja (*action*), objek (terkait dengan tabel *database*), dan pelengkap.
- *Data*: tabel *database* yang digunakan *microservice*.

Identifikasi *microservice* ini diterapkan untuk hanya satu subsistem dari sistem monolitik yang menangani operasi pada setiap tabel *database*. Sedangkan subsistem dengan memiliki tabel yang muncul pada lebih dari satu daftar subsistem maka tidak akan dilakukan identifikasi *microservice* untuk subsistem tersebut. Selanjutnya, setelah setiap pasangan (fc_i, tb_j) teridentifikasi sebagai kandidat *microservice* kemudian buat database sendiri untuk tabel dari subsistem kandidat *microservice* dan hilangkan subsistem kandidat *microservice* dari sistem S (setelah *gateway* API dibuat pada langkah 6). Berikut kriteria subsistem non-kandidat *microservice*:

- Subsistem yang berbagi tabel *database* yang sama.
- *Microservice* yang mewakili operasi yang berada ditengah operasi lain.
- Operasi yang melibatkan lebih dari satu subsistem pada sebuah transkasi (misalnya, transaksi transfer uang dari subsistem rekening *check* ke subsistem rekening tabungan).

f) Buat *gateway* API untuk setiap *facades* sebagai migrasi *microservice* untuk *client*. *Gateway* API merupakan sebuah *intermediate layer* diantara *client* dan *server*, yakni sebuah komponen baru yang menangani panggilan (*request*) dari *client* dan mensikronisasi panggilan ke *microservice* M dan fc_1' (fc_1 versi baru tanpa *source code*, karena *source code* tersebut telah diekstrak dan diimplementasikan ke *microservice* M). Terdapat tiga kasus yang harus dipertimbangkan ketika sinkronisasi:

i) Ketika *input* dari fc_1' adalah *output* dari M atau *input* dari M adalah *output* dari fc_1'.

ii) Ketika *input* dari M dan fc_1' sama dengan *input* API *gateway* dan urutan intansiasi tidak diperhatikan.

iii) Ketika harus membagi fc_1' menjadi dua fungsi fc_1' dan fc_1'' dan *microservice* M harus dipanggil setelah fc_1' dan sebelum fc_1''.

Jika sinkronisasi panggilan dapat dilakukan seperti kasus (i) dan (ii), maka identifikasi *microservice* M sebagai "*strong candidate*", tetapi jika sinkronisasi panggilan hanya bisa seperti kasus (iii) maka identifikasi *microservice* M sebagai "*candidate with additional effort*". Selain itu, jika *business rule* yang diimplementasikan pada *microservice* perlu mempebaharui data di $tb_i \in ss_x$ dan tb_j bukan $\in ss_x$ dalam lingkup transaksi yang sama, maka identifikasi *microservice* M sebagai "*non candidate*".

IV. HASIL IDENTIFIKASI *MICROSERVICE*

Berikut tahapan identifikasi *microservice* dari sistem informasi DPLK:

A. Identifikasi Subsistem dan Tabel *Database*

Langkah #1, terdapat 5 subsistem yang didefinisikan, yakni Kepesertaan (KP), Transaksi (TR), Investasi (INV), Pengelolaan (PG), dan Pelaporan (RPT), serta 88 tabel *database*. Pada tahapan ini, tabel pada database dipetakan kepada masing-masing subsistemnya, dimana tabel tersebut digunakan oleh subsistem tersebut. Disamping itu, terdapat pula tabel data yang digunakan oleh lebih dari 1 (satu) subsistem. Hasil pemetaan tabel *database* pada subsitem secara menyeluruh dapat dilihat pada Tabel III.

TABEL III. PEMETAAN SUBSISTEM DAN TABEL DATABASE

KP	TR	INV	PG	RPT
approvals	transaction	investment	**products**	members
approval_history	transaction_codes	investment_extens	**instruments**	member_diversions
cifs	transaction_		**packet**	transaction_
			packet_details	

member_codes **corporates** contacts addresses additionals documents bank_accounts pics **rules** rule_formulas accounts option_values banks insurances members member_accounts member_diversions heirs	fees accounts member_codes approvals approval_history **upload** **upload_details** **equation_trx_histories**	ion escrows investment_accruals investment_histories investment_details **streams** transactions **operationals** **operational_details**	**counterpart_products** transaction_codes fees **holidays** **users** **sessions** **roles** **regulators** **dplks** banks insurances **bank_counterparts** investment_managers **agent_institutions** **agent_branch** **agents** **apis** **clients** **modules** **menus** contacts addresses bank_accounts documents pics heirs **goals** additionals **notifications** **approval_settings** **reports** **report_templates** **upload_settings** **jobs** **codes** rule_formulas **options** option_values **system_parameters** **translations** **dashboard_settings** **dashboard_roles**	codes transaction_fees fees investments investment_extensions investment_managers escrows

Subsistem yang memiliki tabel *database* dengan warna abu-abu tidak akan dilakukan analisis pada langkah selanjutnya. Hal ini dikarenakan tabel *database* tersebut muncul pada lebih dari satu daftar subsistem (tabel *database* digunakan oleh lebih dari satu subsistem). Dimana identifikasi *microservice* yang dilakukan oleh [6] memastikan bahwa hanya satu subsistem yang menangani operasi atau transaksi di setiap tabel *database*.

B. Identifikasi *Facades* dan Fungsi Bisnis

Langkah #2, dilakukan analisis ketergantungan fungsi bisnis (B) dan tabel *database* (D) berdasarkan *facades* (F) (*entry point* dari sistem) sesuai hasil analisis ketergantungan subsistem dan tabel *database* pada langkah #1 dengan mengabaikan tabel *database* yang digunakan oleh lebih dari satu subsistem dengan *v-vertices*; *e-edges*, seperti terlihat pada Tabel III.

TABEL IV. ANALISIS KETERGANTUNGAN VERTICES (V)

Subsistem	***Facades***	***Function* (v)**	***Function Calls* (e)**	***Data Base* (e)**
KP	CorporateController	22	14	8
	RuleController	20	12	9
TR	UploadController	12	10	5
INV	StreamController	13	8	4
PG	ProductController	14	9	4
	InstrumentController	16	11	5
	PacketController	15	9	3
	CountProdController	14	13	6
	HolidayController	19	7	3
	UserController	17	11	4
	SessionController	14	7	3
	RoleController	14	11	5
	RegulatorController	14	8	3
	DplkController	14	8	3
	BankCountController	13	9	4
	AgentInstController	12	7	3
	AgentBranchController	13	9	4
	AgentController	13	9	4
	ApiController	13	9	4
	ClientController	15	10	4
	ModuleController	14	9	4
	MenuController	16	11	5
	GoalController	12	8	4
	AppSettingController	12	8	4
	ReportController	20	10	4
	CodeController	11	8	4
	OptionController	13	10	5
	SysParamController	14	8	3
	TransController	13	7	3

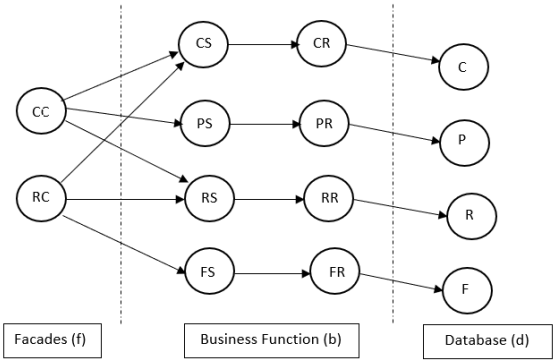
Setelah analisis ketergantungan didapatkan, kemudian dilakukan proses identifikasi kandidat *microservice*, seperti terlihat pada Tabel IV.

TABEL V. KANDIDAT MICROSERVICE PADA SUBSISTEM

	KP	**TR**	**INV**	**PG**
Tables (v)	3	3	3	33
Functions (v)	42	12	13	355
Function Calls (e)	26	10	8	226
Database Accesses (e)	17	5	4	98
Kandidat *Microservices*	**2**	**1**	**1**	**22**

C. Identifikasi Pasangan *Facades* dan Tabel *Database*

Langkah #3 dan #4, dari kandidat *microservice* yang telah terpilih pada langkah #2, selanjutnya dilakukan dekomposisi fungsi-fungsi bisnis sesuai operasi/transaksi pada setiap tabel *database*. Sebagai contoh terlihat pada Gambar IV adalah grafik ketergantungan pada subsistem Kepesertaan (KP) antara *façades* (*f*), fungsi bisnis (*b*), dan tabel *database* (*d*) :



GAMBAR IV. GRAFIK KETERGANTUNGAN PADA SUBSISTEM KEPESERTAAN (KP)

Keterangan :

CC	CorporateController	PR	PacketRepository
RC	RuleController	RR	RuleRepository
CS	CorporateService	FR	FormulaRepository
PS	PacketService	C	Corporates
RS	RuleService	P	Packets
FS	FormulaService	R	Rules
CR	CorporateRepository	F	Formulas

Grafik pada gambar IV menunjukan ketergantungan antara *facades*, fungsi bisnis, dan *database* pada subsistem kepesertaan (KP). CC memanggil fungsi bisnis yang terdapat pada CS, kemudian CS memanggil fungsi *database* yang tersimpan pada CR, dan CR mengakses *database* ke tabel C, begitu pula dengan *facades* yang lain, sehingga hubungan tersebut akan membentuk grafik ketergantungan. Proses ini dilakukan pula pada subsistem lainnya.

Hasil dari proses ini dapat dilihat secara menyeluruh pada Tabel VI.

TABEL VI. DEKOMPOSISI FUNGSI BISNIS PADA KANDIDAT MICROSERVICE

No	*Façade* (f)	*Business Function* (b)	Tabel (d)
Kepesertaan (KP)			
1	Corporate Controller	CorporateService	corporates
		MemberCodeService	member codes
		ApprovalService	approvals
		ProductService	products
		PacketService	packets
		InsuranceService	insurances
		BankService	banks
		AgentService	agents
2	Rule Controller	RuleService	rules
		CorporateService	corporates
		ApprovalService	approvals
		ProcessService	processes
		FormulaService	formulas
		SystemParameter Service	system parameters
Transaksi (TR)			
3	Upload Controller	UploadService	uploads
Investasi (INV)			
4	Stream Controller	StreamService	streams
Pengelolaan (PG)			
5	Product	ProductService	products
6	Instrument	InstrumentService	instruments
7	Packet Controller	PacketService	packets
		InstrumentService	instruments
8	Count Prod Controller	CounterpartProduct Service	counterpart products
		InstrumentService	instruments
9	Holiday Controller	HolidayService	holidays
10	User Controller	UserService	users
		RoleService	roles
11	Session Controller	SessionService	sessions
12	Role Controller	RoleService	roles
		MenuService	menus
13	Regulator Controller	RegulatorService	regulators
14	Dplk Controller	DplkService	dplks
15	Bank Count Controller	CounterpartService	counterparts
		BankService	banks
16	Agent Controller	AgentService	agents
		AgentBranchService	agent branches
		AgentInstitution Service	agent institutions
17	Api Controller	ApiService	apis
18	Client Controller	ClientService	clients
		ApiPermission Service	api permissions
19	Menu Controller	MenuService	menus
		ModuleService	modules
20	Goal Controller	GoalService	goals
21	App Setting Controller	ApprovalSetting Service	approval settings
22	Report Controller	ReportService	reports
		AgentService	agents
23	Code Controller	CodeService	codes

24	Option Controller	OptionService	options
25	SysParam Controller	SystemParameter Service	system parameters
26	Trans Controller	TranslationService	translations

Selanjutnya dilakukan pemilahan pada setiap subsistem yang telah didefinsikan, identifikasikan (f_1, t_1) *facades* terhadap tabel, dimana tabel tersebut merupakan anggota subsistem ($t_1 \epsilon ss_1$).

D. Identifikasi Kandidat *Microservice*

Langkah #5, terdapat 21 kandidat *microservice* dan berikut deskripsi kandidat *microservice* tersebut yang diperoleh dari hasil dekomposisi fungsi-fungsi bisnis pada setiap operasi tabel *database* di langkah #3 dan langkah #4, seperti terlihat pada Tabel VI.

TABEL VII. DESKRIPSI KANDIDAT MICROSERVICE

***Facades* (Fungsi Bisnis)**	**Deskripsi Kandidat Microservice**				
	Nama	**Tujuan**	**Input/ Output**	**Fitur**	**Data**
CorporateService	Kepesertaan.Corporate	Pengelolaan Service Corporate	CorporateForm (input), Corporate Object (output)	save(), select(), find(), delete()	corporates
RuleService, CorporateService	Kepesertaan.Rule	Pengelolaan Service Rule	RuleForm (input), Rule Object (output)	save(), select(), find(), delete()	rules, corporates
UploadService	Transaksi.Upload	Pengelolaan Service Upload	UploadForm (input), Upload Object (output)	save(), select(), find(), delete()	uploads
StreamService	Investasi.Stream	Pengelolaan Service Stream	StreamForm (input), Stream Object (output)	save(), select(), find(), delete()	streams
CounterpartProductService	Pengelolaan.CounterpartProduct	Pengelolaan Service Counterpart Product	CounterpartForm (input), Counterpart Object (output)	save(), select(), find(), delete()	counterpart_products
HolidayService	Pengelolaan.Holiday	Pengelolaan Service Holiday	HolidayForm (input), Holiday Object (output)	save(), select(), find(), delete()	holidays
UserService, RoleService	Pengelolaan.User	Pengelolaan Service User	UserForm (input), User Object (output)	save(), select(), find(), delete()	users, roles
SessionService	Pengelolaan.Session	Pengelolaan Service Session	SessionForm (input), Session Object (output)	save(), select(), find(), delete()	sessions
RoleService, MenuService	Pengelolaan.Role	Pengelolaan Service Role	RoleForm (input), RoleObject (output)	save(), select(), find(), delete()	roles, menus
RegulatorService	Pengelolaan.Regulator	Pengelolaan Service Regulator	RegulatorForm (input), Regulator Object (output)	save(), select(), find(), delete()	regulators
DplkService	Pengelolaan.Dplk	Pengelolaan Service Dplk	DplkForm (input), DplkObject (output)	save(), select(), find(), delete()	dplks
BankCounterpartService	Pengelolaan.BankCounterpart	Pengelolaan Service Counterpart	Counterpart Form (input), Counterpart Object (output)	save(), select(), find(), delete()	counterparts
ApiService	Pengelolaan.Api	Pengelolaan Service Api	ApiForm (input), ApiObject (output)	save(), select(), find(), delete()	apis
ClientService, ApiPermissionService	Pengelolaan.Client	Pengelolaan Service Client	ClientForm (input), ClientObject (output)	save(), select(), find(), delete()	clients, api_permissions
MenuService, ModuleService	Pengelolaan.Menu	Pengelolaan Service Menu	MenuForm (input), MenuObject (output)	save(), select(), find(), delete()	menus, modules
GoalService	Pengelolaan.Goal	Pengelolaan Service Goal	GoalForm (input), GoalObject (output)	save(), select(), find(), delete()	goals
ApprovalSettingService	Pengelolaan.ApprovalSetting	Pengelolaan Service Approval Setting	ApprovalSettingForm (input), ApprovalSetting Object (output)	save(), select(), find(), delete()	approval_settings
ReportService	Pengelolaan.Report	Pengelolaan Service Report	ReportForm (input), Report Object (output)	save(), select(), find(), delete()	reports
CodeService	Pengelolaan.Code	Pengelolaan Service Code	CodeForm (input), Code Object (output)	save(), select(), find(), delete()	codes
OptionService	Pengelolaan.Options	Pengelolaan Service Option	OptionForm (input), Option Object (output)	save(), select(), find(), delete()	options
TranslationService	Pengelolaan.Translation	Pengelolaan Service Translation	TranslationForm (input), Translation Object (output)	save(), select(), find(), delete()	translations

E. Identifikasi API *Gateway*

Langkah #6, berikut API *gateway* untuk setiap *facades* pada langkah #5 sebagai *microservice* yang dapat diakses oleh *client*, seperti terlihat pada Tabel VII.

TABEL VIII. API GATEWAY

No.	Daftar API	Operasi HTTP
1	api/corporates	GET, POST, PUT, DELETE
2	api/rules	GET, POST, PUT, DELETE
3	api/upload-transactions	GET, POST, PUT, DELETE
4	api/streams	GET, POST, PUT, DELETE
5	api/counterpart-products	GET, POST, PUT, DELETE
6	api/holidays	GET, POST, PUT, DELETE

7	api/users	GET, POST, PUT, DELETE
8	api/sessions	GET, POST, PUT, DELETE
9	api/roles	GET, POST, PUT, DELETE
10	api/regulators	GET, POST, PUT, DELETE
11	api/dplks	GET, POST, PUT, DELETE
12	api/bank-counterparts	GET, POST, PUT, DELETE
13	api/apis	GET, POST, PUT, DELETE
14	api/clients	GET, POST, PUT, DELETE
15	api/menus	GET, POST, PUT, DELETE
16	api/goals	GET, POST, PUT, DELETE
17	api/approval-settings	GET, POST, PUT, DELETE
18	api/reports	GET, POST, PUT, DELETE
19	api/codes	GET, POST, PUT, DELETE
20	api/options	GET, POST, PUT, DELETE
21	api/translations	GET, POST, PUT, DELETE

V. KESIMPULAN DAN SARAN

Sampai saat ini, penelitian berhasil melakukan identifikasi 21 *microservice* dari sistem monolitik sistem informasi DPLK menggunakan pendekatan 6 (enam) langkah migrasi. Langkah migrasi dilakukan melalui analisis keterhubungan subsistem monolitik terhadap *facades* (*f*) (*entry points* atau *user interface* untuk *input* data), tabel *database* (*d*) dan fungsi bisnis (*b*) atau *method* dari operasi tabel *database* tersebut untuk setiap transaksi bisnis. Dapat dikatan bahwa pendekatan ini cukup efektif sebagai tahap awal dari proses migrasi sistem monolitik ke *microservice*, karena *service* yang memiliki ketergantungan pada lebih dari 1 (satu) subsistem tidak akan menjadi kandidat *microservice*, hal ini dilakukan untuk meminimalisir resiko migrasi sistemnya. Tahapan berikutnya adalah menguji model arsitektur *microservice* yang telah dikembangkan.

Implementasi model arsitektur akan dikembangkan sesuai dengan teknologi yang digunakan pada sistem DPLK, yakni menggunakan DBMS PostgreSQL, framework *Springboot*, Bahasa Pemrograman Java, serta Apache Tomcat sebagai *WebServer*. Pola komunikasi diimplementasikan menggunakan REST *Service* dengan format JSON. Setelah model diimplementasikan, selanjutnya dilakukan pengujian *microservice* melalui 3 (tiga) model pengujian : (1) individual *microservice* (*unit test*), (2) *component test*, dan (3) *end-to-end test*. *Unit test* menguji potongan kecil dan terisolasi dari fungsi (*service*), sementara *component test* melakukan pengujian antarmuka eksternal dari layanan individual atau pada tingkat pengujian subsistem dari kelompok *service*, kemudian *end-to-end test* dilakukan untuk memastikan seluruh sistem telah bekerja, dimana pada tahap ini semua fungsionalitas bisnis yang dirancang diuji kinerjanya.

REFERENCES

[1] Newman, S. (2015). Building Microservices. O'Reilly Media, Inc.

[2] Priyadarshini, S. (2017). Microservices Architecture. International Research Journal of Computer Science (IRJCS), Issue 04, Volume 4.

[3] Messina, Antonio, dkk. (2016). A Simplified Database Pattern for the Microservice Architecture. The Eight International Conference on Advances in Databases, Knowledge, and Data Applications.

[4] Suryotrisongko, Hatma. (2017). Arsitektur Microservice untuk Resiliensi Sistem Informasi. Jurnal Sisfo Vol. 06 No. 02 hal. 235-250.

[5] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015). *Microservices Migration Patterns*

[6] Levcovitz, A., Terra, R., Valente, M.T., (2017). Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems.

[7] Introduction to Spring Framework. http://docs.spring.io/spring-framework/docs/3.0.x/reference/overview.html. Diakses pada : 10 Juni 2017.

[8] Fowler, M., Scott, K. (1999). UML Distilled Second Edition: A Brief Guide to the Standard Object Modeling Language, Addison Wesley Publisher.

[9] Long, Josh. (2013). Spring Framework Live Lessons. Addison-Wesley Professional.

[10] Messina, Antonio, et.al. (2016). A Simplified Database Pattern for the Microservice Architecture. The Eight International Conference on Advances in Databases, Knowledge, and Data Applications.