ADMI Jurnal Ilmiah Teknik
JUIT Vol 1 No. 1 Januari 2022 | P-ISSN: 2828-6936 E-ISSN: 2828-6901, Page 09-19

IMPLEMENTASI ARSITEKTUR PEMROGRAMAN MICROSERVICES UNTUK MIGRASI SISTEM INFORMASI PT. DWIDAYA WORLD WIDE (DWIDAYA TOUR AND TRAVEL) DENGAN METODOLOGI SDLC RAPID APPLICATION DEVELOPMENT

Wisnu Sukma Maulana
Fakultas Ilmu Komputer dan Teknologi Informasi , wisnu_maulana@staff.gunadarma.ac.id, Universitas Gunadarma

ABSTRACT

Dwidaya Tour has a fairly rapid business development both branches in almost all of Indonesia and the services they offer to the public ranging from train tickets, planes, Umrah, and all services in the field of travel. Problems arise when all services and branches have their respective systems and are not connected and centralized. Thus, Dwidaya tour wants a rapid migration into one centralized system without stopping all of the systems that are running. The purpose of this study is the result of implementing the solution used to overcome these problems. The use of the microservices programming architecture for the migration process. Where microservices architecture allows each stand-alone system to be a service at the back-end level and connected to a centralized system at the front-end level. Because the migration process needed must be fast, the SDLC (System Development Life Cycle) approach based on the RAD (Rapid Application Development) methodology was used. This RAD methodology has a stage of user evaluation involvement as a key means for the development and deployment process to be more precise and short in duration for its development.

Keywords : architecture, microservices, system, sdlc, rad

ABSTRAK

Dwidaya Tour memiliki perkembangan bisnis yang cukup pesat baik secara cabang hampir di seluruh Indonesia maupun jasa yang mereka tawarkan ke masyarakat mulai dari tiket kereta api, pesawat, umroh, dan seluruh jasa di bidang travel. Permasalahan timbul ketika kesemua jasa dan cabang tersebut memiliki sistem masing-masing dan tidak terkoneksi dan terpusat. Sehingga, Dwidaya tour menginginkan suatu migrasi yang cepat menjadi satu sistem terpusat tanpa menghentikan ke semua sistem tersebut yang sedang berjalan. Tujuan penelitian ini adalah hasil implementasi solusi yang digunakan untuk mengatasi masalah tersebut. Penggunaan arsitektur pemrograman microservices untuk proses migrasinya. Dimana arsitektur microservices memungkinkan setiap sistem yang berdiri sendiri menjadi services di level back-end dan dikoneksikan ke sistem terpusat di level front-end. Karena proses migrasi yang dibutuhkan harus cepat, maka pendekatan SDLC (System Development Life Cycle) berbasiskan metodologi RAD (Rapid Application Development)digunakan. Metodologi RAD ini memiliki tahapan keterlibatan user evaluation sebagai sarana pengunci untuk proses development dan deployment lebih presisi dan singkat secara durasi pengembangannya

Kata Kunci : arsitektur, microservices, sistem, sdlc, rad

1. PENDAHULUAN

PT Dwidaya World Wide, atau yang dikenal dengan merek Dwidaya Tour & Travel atau disingkat menjadi Dwidayatour, merupakan salah satu agen travel terbesar di Indonesia yang telah berdiri sejak 19 Juli 1967. Hingga tahun 2020, Dwidayatour telah memiliki lebih dari 90 cabang yang tersebar di berbagai kota besar di seluruh Indonesia. Dwidaya juga telah menjadi partner utama khususnya di bidang tiket pesawat maskapai ternama seperti Singapore Airlines, Cathay Pacific, Qatar Airways, KLM Royal Dutch, Eva Air, Lufthansa German Airlines, Garuda Indonesia, dan masih banyak lainnya. Selain jasa tiket pesawat terbang, Dwidayatour juga memiliki jasa di pemesanan hotel, tiket kereta api, umroh, paket wisata baik individu maupun perusahaan, dan juga pengajuan visa ke luar negeri [3].

Kesemua lini usaha tersebut memiliki sistem informasi berupa aplikasi yang berbeda-beda dan berdiri sendiri tanpa adanya koneksi data terpusat. Hal ini mengakibatkan pelaporan data yang tidak bisa real-time dan akurat karena kantor pusat harus menarik data per masing-masing aplikasi tersebut di akhir bulan. Ditambah, pihak pusat harus membentuk tim tersendiri untuk rekap data dari seluruh cabang dan memakan waktu yang cukup lama. Jika ada data terbaru setelah dilakukan proses rekap data, maka data tersebut akan masuk di bulan berikutnya. Oleh karena itu, Dwidayatour melakukan SOP khusus dengan adanya tanggal stok-opname. Tanggal dimana seluruh transaksi harus dihentikan sementera. Jadi transaksi yang terjadi antara tanggal 25 hingga akhir bulan, dicatatkan ke transaksi bulan berikutnya.

Sebagai perusahaan yang sudah memiliki mitra baik di Indonesia maupun luar negeri, pelaporan data sebagai dasar kepercayaan dan transparansi maupun pelaporan keuangan untuk kepentingan pajak, hal ini tentunya tidak menjadi permasalahan serius dan sudah terjadi bertahun-tahun di Dwidayatour. Ketidaksinergian data antara sistem di seluruh lini usaha Dwidayatour dikarenakan pengembangan sistem yang masih mengadopsi arsitektur monolitik. Hal ini wajar terjadi, karena Dwidayatour memberikan kuasa penuh bagi setiap lini usaha dan cabangnya untuk mengembangkan sistem masing-masing. Sehingga, setiap lini usahanya (divisi) serta para cabang mengembangkan sistem secara mandiri dengan vendor konsultan IT masing-masing dengan implementasi arsitektur monolitik.

Penggunaan arsitektur monolitik secara umum lebih mudah diimplementasikan dan berfungsi dengan baik hanya untuk satu sistem terdedikasi tersendiri. Dimana secara jangka panjang, arsitektur monolitik itu sendiri tidak dapat mengakomodir pertukaran data terutama antara level front-end dan back-end yang memiliki engine, teknologi, maupun framework yang berbeda-beda [8].

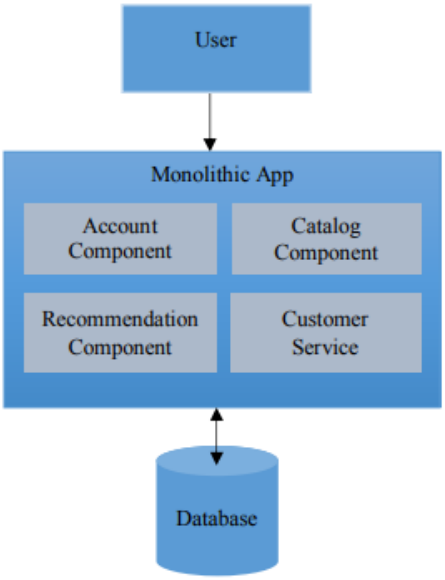
Disamping itu, Dwidayatour menginginkan adanya migrasi data dari seluruh sistem yang ada menjadi satu sistem terpusat tanpa adanya penghentian sistem yang sedang berjalan tersebut dan dalam waktu cepat. Jika dihentikan satu hari, maka dapat mengakibatkan adanya kerugian pemasukan bagi bisnis Dwidayatour itu sendiri. Seluruh lini usaha (divisi) serta cabang se-Indonesia akan merasa kesulitan mencapai target bisnis mereka masing-masing jika dihentikan beberapa waktu maupun hanya satu hari saja. Bisnis travel memiliki keunikan sendiri, karena transaksi yang dilakukan tidak mengenal tanggal kerja maupun tanggal merah. Bahkan hari libur atau tanggal merah merupakan tanggal dimana terjadinya transaksi besar. Adapun tujuan dari penelitian ini adalah untuk mengimplementasikan seluruh ruang lingkup penelitian menjadi solusi bagi permasalahan yang serupa dengan studi kasus di Dwidayatour. Ruang lingkup yang dimaksud adalah implementasi dari arsitektur pemrograman microservices untuk proses migrasi data dari banyak sistem yang berbeda-beda baik level front-end maupun back-endnya dengan durasi yang cukup singkat tanpa adanya penghentian sistem yang sedang berjalan atau disebut juga dengan proses paralel. Durasi singkat dan proses paralel dapat diatasi dengan pendekatan model SDLC khususnya metodologi RAD.

2. TINJAUAN PUSTAKA

Microservices

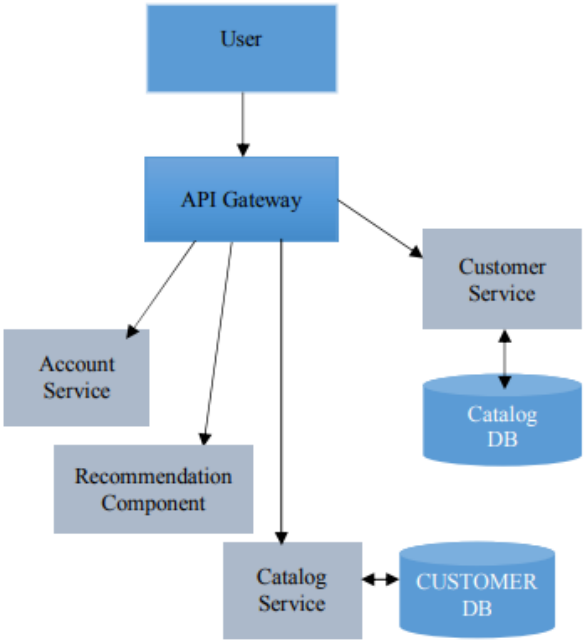
Arsitektur microservice memecah kompleks aplikasi ke beberapa layanan otonom proses kecil, independen yang berkomunikasi dengan masing-masing API lain yang menggunakan bahasa independen. Layanan ini sangat dipisahkan dan fokus pada melakukan tugas kecil, yang memungkinkan pendekatan modular untuk pembangunan sistem. Layanan seperti ini mudah diganti, dirancang untuk memenuhi spesifik kebutuhan bisnis, dan dapat diimplementasikan dengan menggunakan yang berbeda teknologi. Selain itu, karena layanan mikro otonom, mereka dibangun dan digunakan secara mandiri metodologi pengembangan perangkat lunak pengiriman berkelanjutan [5].

Dibandingkan dengan arsitektur microservice, arsitektur monolitik memiliki basis data tunggal untuk berinteraksi. Debugging itu sulit karena kita tidak tahu di mana sebenarnya kesalahan terletak. Itu dibangun menggunakan bahasa pemrograman yang sama dan jika tidak fleksibel dibandingkan dengan layanan arsitektur microservice [6].



Gambar 1. Arsitektur monolitik dengan database tunggal.
Sumber: Priyadarshini dan Shilpa (2017)

Sedangkan gaya arsitektur layanan microservice dalam pengembangan platform tunggal memungkinkan tim manajemen layanan untuk mendukung lebih banyak tim produk dan layanan dengan lebih mudah. Ketika masalah terjadi dalam produksi, lebih mudah untuk mengidentifikasi dan mengisolasi masalah operasi juga dapat mengidentifikasi siapa dari tim pengembangan harus dibawa untuk memecahkan masalah-masalah.

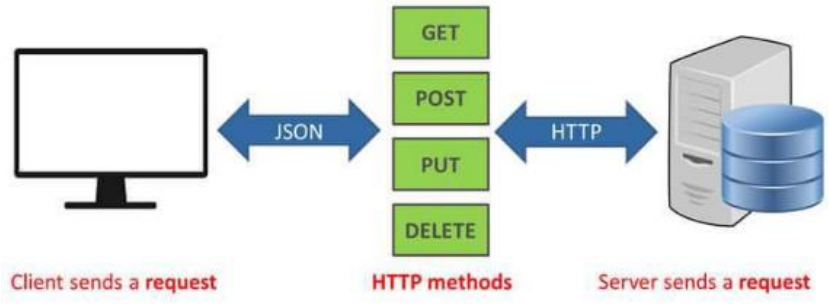


Gambar 2. Arsitektur microservice dengan multi database terpisah
Sumber: Priyadarshini dan Shilpa (2017)

API

Pada arsitektur microservice terdapat platform API sebagai perantara pertukaran data dan layanan transaksinya. API sendiri merupakan singkatan dari Antarmuka pemrograman aplikasi (API) yang berfungsi sebagai antarmuka komputasi ke komponen perangkat lunak atau sistem, yang menentukan bagaimana komponen atau sistem lain dapat menggunakannya [4].

Di dunia saat ini, API biasanya dikembangkan menggunakan gaya RESTful. API ini akan memiliki serangkaian kata kerja yang terkait dengan tindakan HTTP, seperti berikut ini:



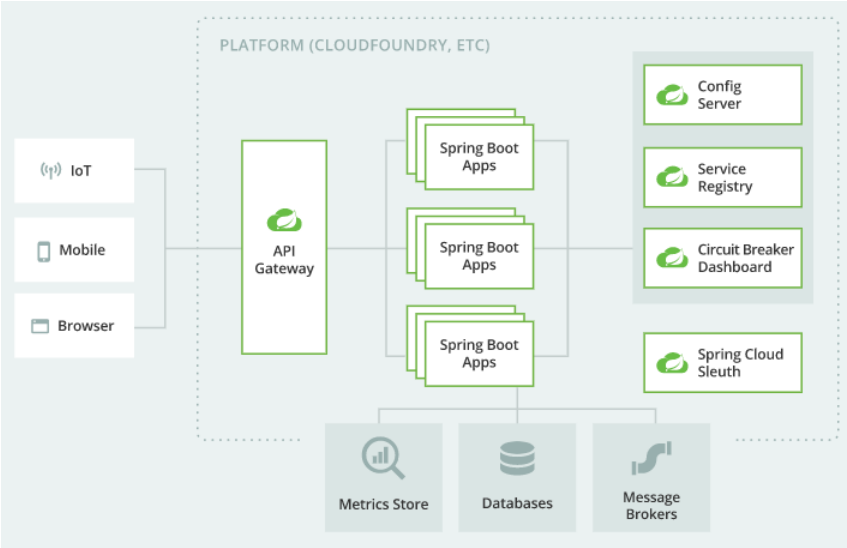
Gambar 3. Alur Data Restful API

1. Get (dapatkan satu item atau koleksi)
2. Post (tambahkan item ke koleksi)
3. Put (edit item yang sudah ada dalam koleksi)
4. Delete (hapus item dalam koleksi)

Keuntungan dari konsistensi ini melalui aplikasi yang berbeda adalah memiliki standar ketika melakukan berbagai tindakan. Keempat kata kerja HTTP berbeda di atas berkorelasi dengan kemampuan CRUD umum yang banyak digunakan aplikasi saat ini. Saat bekerja dengan API yang berbeda dalam satu aplikasi, ini membuat cara yang dapat dikenali untuk memahami implikasi dari tindakan yang diambil di berbagai antarmuka.

SpringBoot Restful Webservices

Salah satu teknologi yang dapat mengembangkan sistem berbasis arsitektur microservice adalah Java SpringBoot. Di dalam SpringBoot itu sendiri memiliki fitur webservices khususnya Restful API [9].



Gambar 4. SpringBoot untuk Microservices
Sumber: Spring (2020)

Dalam implementasinya untuk membangun sistem berbasiskan arsitektur microservice dengan framework SpringBoot, diperlukan anotasi-anotasi yang mengelola HTTP request dan response sebagai proses transaksi layanannya [1]. Anotasi tersebut adalah sebagai berikut:

- @RequestMapping anotasi memetakan permintaan HTTP ke metode pengendali pengendali Spring MVC dan REST, menangani semua jenis permintaan.

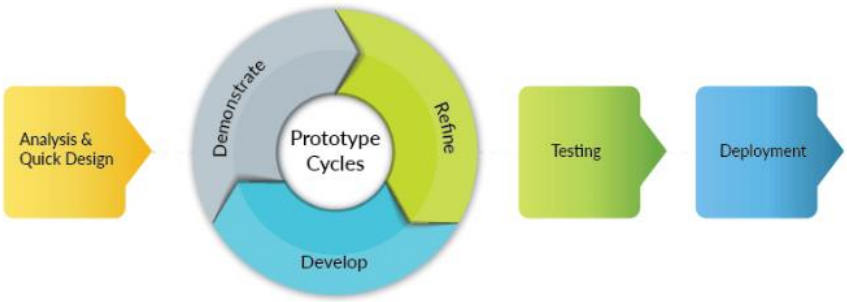
- @Controller anotasi mewakili Class dengan titik akhir.
- @ResponseStatus menunjukkan nilai kembalian dari *request* yang dikirimkan dari *client* ke *server*.
- @RestController anotasi adalah penjelasan kenyamanan yang dengan sendirinya dijelaskan dengan @Controller dan @ResponseBody.
- @GetMapping anotasi berfungsi untuk permintaan GET.
- @PostMapping anotasi berfungsi untuk permintaan POST.
- @PutMapping anotasi berfungsi untuk permintaan PUT.
- @DeleteMapping anotasi berfungsi untuk permintaan DELETE.

3. METODOLOGI PENELITIAN

Pengembangan proyek penelitian untuk migrasi sistem ke arsitektur microservice menggunakan pendekatan model SDLC. SDLC sendiri merupakan proses yang diikuti untuk proyek perangkat lunak, dalam organisasi perangkat lunak. Ini terdiri dari rencana terperinci yang menjelaskan bagaimana mengembangkan, memelihara, mengganti dan mengubah atau meningkatkan perangkat lunak tertentu. Siklus hidup mendefinisikan metodologi untuk meningkatkan kualitas perangkat lunak dan proses pengembangan secara keseluruhan [2].

Salah satu pendekatan dari model SDLC yang mendukung untuk pengembangan sistem dengan durasi singkat adalah metodologi RAD. RAD Model menggunakan pendekatan orientasi komponen terhadap pengembangan perangkat lunak. Model ini bertujuan mempersingkat waktu yang biasanya diperlukan dalam siklus hidup pengembangan konvensional [7].

Mengacu ke tahapan-tahapan yang menjadi standar dari RAD, maka penelitian untuk migrasi data dari beberapa sistem yang berjalan menjadi sistem terpusat dilakukan sebagai berikut:



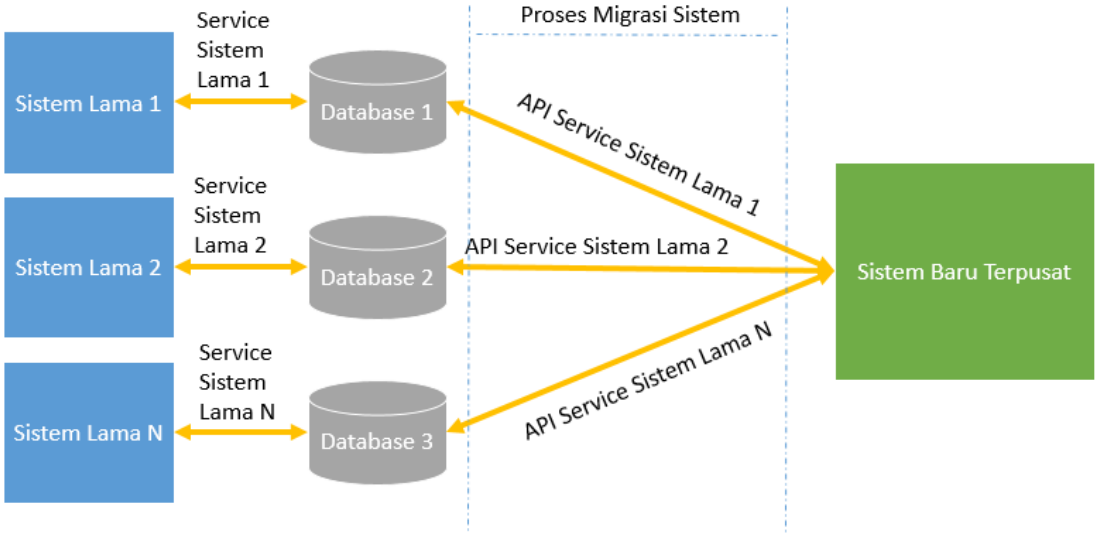
Gambar 5. Tahapan-tahapan dalam metodologi RAD
Sumber: Purwanto (2017)

1. Tahap *Analysis & Quick Design*
   - Pada tahapan ini tim menganalisa parameter-parameter serta layanan transaksi yang telah dimiliki oleh sistem lama untuk dapat dimigrasi ke sistem baru terpusat. Tentunya kesemua rancangan tersebut akan berbasiskan arsitektur microservices.

2. Tahap *Prototype Cycles*
   - Tahapan ini terdiri dari 3 sub-tahapan, yaitu:
     i. *Develop* artinya rancangan hasil dari analisa awal akan diprogram baik di level backend maupun frontend, dan dicocokan dengan sistem lama.
     ii. *Demonstrate* artinya user melihat hasil migrasi data dari sistem lama dengan sistem baru.
     iii. *Refine* artinya program direvisi sesuai hasil feedback dari user pada sub-tahapan *demonstrate* bila memang data yang dimigrasi tidak sepenuhnya berhasil berjalan. Dan akan kembali ke tahap awal yaitu *develop* dan seterusnya hingga program dari modul tersebut sudah sesuai oleh keinginan user

3. Tahap *Testing*
   - Pada tahapan ini pengujian yang dilakukan menggunakan *quality automation tools*. QA tools mempercepat testing dengan script yang ditanam di pemrograman sintaks modulnya.

4. Tahap *Deployment*
    - Setelah program lulus uji dan sesuai dengan arsitektur *microservices* yang telah ditentukan, maka tiap modul langsung diimplementasikan ke server. Dimana proses migrasi akan dapat langsung digunakan secara penuh paralel dengan sistem lama dan dapat dimonitoring pula pertukaran datanya.



Gambar 6. Proses Migrasi Sistem dengan Microservices

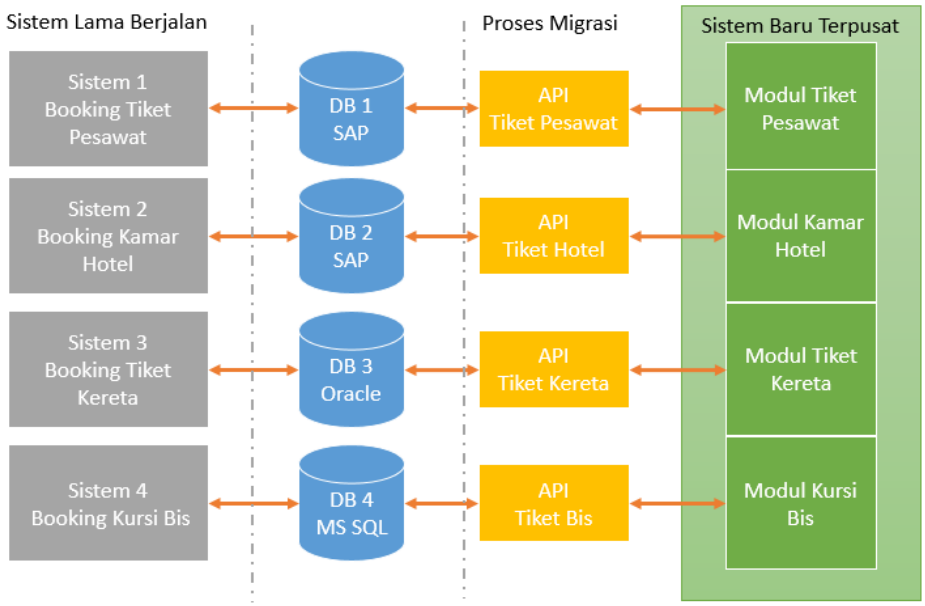
Secara sederhana proses migrasi yang dirancang dengan metodologi RAD akan dapat digunakan secara paralel dengan sistem lama. Hal ini dapat kita lihat di gambar 6, dimana tertera berapa banyak sistem lama yang ada dengan database masing-masing akan dapat berjalan beriringan dengan sistem baru yang dikembangkan. Hal ini dapat dilakukan karena sistem baru di level backend hanya mengelola API service masing-masing sistem lama dan ditampilkan hasil migrasinya di level frontend sistem baru yang terpusat.

4. HASIL DAN PEMBAHASAN

Hasil dari penelitian dengan metodologi RAD dalam migrasi sistem terpusat didokumentasi ke dalam beberapa tahapan sesuai dengan tahapan-tahapan yang menjadi standar dari metodologi RAD itu sendiri.

Tahap *Analysis & Quick Design*

Pada tahapan *analysis & quick design* ini, analisa kebutuhan dirampungkan menjadi 4 layanan API services yang dibuat secara singkat sebagaimana terlihat pada gambar 7. Dimana sistem lama berjalan adalah beberapa sistem yang masih aktif dalam proses migrasi dengan masing-maisng database engine yang berbeda-beda. Proses migrasi sendiri dirancang dengan menggunakan layanan webservices API yang dapat digunakan secara presisi menjadi modul-modul di sistem baru terpusat.



Gambar 7. Rancangan Singkat Alur Migrasi Data dari Layanan API Sistem Baru Terpusat

Tabel 1. API Tiket Pesawat

Modul	SubModul	@RestController	@RequestMapping	method
Tiket Pesawat	Master Maskapai	mstMaskapaiController()	createKota()	Post
			updateKota()	Put
			DeleteKota()	Delete
			listKota()	Get
	Master Bandara	mstBandaraController()	createBandara()	Post
			updateBandara()	Put
			DeleteBandara()	Delete
			listBandara()	Get
	Transaksi Tiket Pesawat	trxTiketPesawatController()	createTiketPesawat()	Post
			updateTiketPesawat()	Put
			DeleteTiketPesawat()	Delete
			listTiketPesawat()	Get

Tabel 2. API Tiket Kereta

Modul	SubModul	@RestController	@RequestMapping	method
Tiket Kereta	Master Stasiun	mstStasiunController()	createStasiun()	Post
			updateStasiun()	Put
			DeleteStasiun()	Delete
			listStasiun()	Get
	Master Kereta	mstKeretaController()	createKereta()	Post
			updateKereta()	Put
			DeleteKereta()	Delete
			listKereta()	Get
	Transaksi	trxTiketKeretaController()	createTiketKereta()	Post

	Tiket Kereta		updateTiketKereta()	Put
			DeleteTiketKereta()	Delete
			listTiketKereta()	Get

Tabel 3. API Kamar Hotel

Modul	SubModul	@RestController	@RequestMapping	method
Kamar Hotel	Master Kota	mstKotaController()	createKota()	Post
			updateKota()	Put
			DeleteKota()	Delete
			listKota()	Get
	Master Hotel	mstHotelController()	createHotel()	Post
			updateHotel()	Put
			DeleteHotel()	Delete
			listHotel()	Get
	Transaksi Kamar Hotel	trxKamarHotelController()	createKamarHotel()	Post
			updateKamarHotel()	Put
			DeleteKamarHotel()	Delete
			listKamarHotel()	Get

Tabel 4. API Tiket Bis

Modul	SubModul	@RestController	@RequestMapping	method
Tiket Bis	Master Bis	mstBisController()	createBis()	Post
			updateBis()	Put
			DeleteBis()	Delete
			listBis()	Get
	Master Terminal	mstTerminalController()	createTerminal()	Post
			updateTerminal()	Put
			DeleteTerminal()	Delete
			listTerminal()	Get
	Transaksi Tiket Bis	trxTiketBisController()	createTiketBis()	Post
			updateTiketBis()	Put
			DeleteTiketBis()	Delete
			listTiketBis()	Get

Tahap *Prototype Cycles*

Pada tahap prototype cycles detail dari daur hidup hasil dari feedback migrasi data oleh user sebagai berikut:

Tabel 5. Daur Hidup Revisi Prototipe API Sistem Terpusat

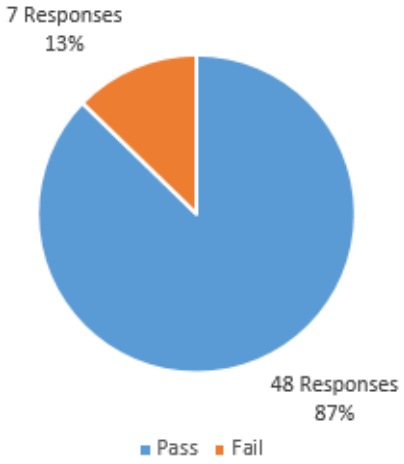
Modul	SubModul	@RestController	Cycles
Tiket Pesawat	Master Maskapai	masterMaskapaiController()	4
	Master Bandara	masterBandaraController()	
	Transaksi Tiket Pesawat	transTiketPesawatController()	
Tiket Kereta	Master Stasiun	mstStasiunController()	3

	Master Kereta	mstKeretaController()	
	Transaksi Tiket Kereta	trxTiketKeretaController()	
Kamar Hotel	Master Kota	mstKotaController()	3
	Master Hotel	mstHotelController()	
	Transaksi Kamar Hotel	trxKamarHotelController()	
Tiket Bis	Master Bis	mstBisController()	2
	Master Terminal	mstTerminalController()	
	Transaksi Tiket Bis	trxTiketBisController()	

Keterangan : Cycles menunjukkan indikator dari jumlah siklus perbaikan dari keseluruhan 3 sub-tahapan pengembangan yaitu, *develop*, *demonstrate*, dan *refine*.

Tahap *Testing*

Testing dilakukan dengan menggunakan alat bantu QA yaitu Postman. Postman sendiri mengecek url dari API services dari masing-masing modul. Hasil testing tersebut didapatkan bahwa rasio pass atau keberhasilan dari skenario pengujian lebih besar dibandingkan dengan fail atau kegagalannya. Dimana rasio perbandingannya terlihat pada gambar 8.



Gambar 8. Kurva Rasio Perbandingan Hasil Pengujian API Services

Dari rasio tersebut, rincian dari hasil *test* tersebut tertera di tabel 6.

Tabel 6. Rincian Hasil Pengujian API Services

Hasil Test	@ResponseStatus	Jumlah Respon	Jumlah per Hasil Test
Pass	200 Ok	36	48
	201 Created	12	
Fail	400 Bad Request	1	7
	403 Forbidden	2	
	405 Method Not Allowed	2	
	500 Internal Server Error	2	

Tahap *Deployment*

Hasil dari deployment dari metodologi RAD untuk migrasi sistem-sistem lama dan masih berjalan dalam implementasi ke sistem baru terpusat dengan menggunakan arsitektur microservices tercatat hanya dalam kurun waktu 3 bulan, dimana rinciannya tertera di tabel 7.

Tabel 7. Kurun Waktu *Deployment* Sistem Baru Terpusat

Modul	Agustus 2017	September 2017	Oktober 2017
Tiket Pesawat			
Tiket Kereta			
Kamar Hotel			
Tiket Bis			

Keterangan:

- Modul Tiket Pesawat di-deploy dalam kurun waktu 2 bulan yaitu Agustus hingga September
- Modul Tiket Kereta di-deploy dalam kurun waktu 1 bulan yaitu Agustus
- Modul Kamar Hotel di-deploy dalam kurun waktu 2 bulan yaitu September hingga Oktober
- Modul Tiket Bis di-deploy dalam kurun waktu 1 bulan yaitu Oktober

Sebagai gambaran bahwasanya, estimasi awal proyek migrasi ini diperhitungkan selama 5 bulan pengerjaan dengan rincian perbandingan yang tertera di tabel 8.

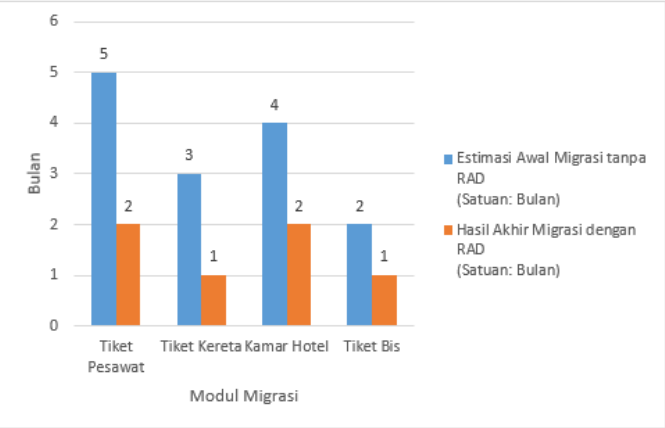
Tabel 8. Perbandingan Waktu Migrasi antara tanpa dan dengan RAD

Modul	Estimasi Awal Migrasi tanpa RAD (Satuan: Bulan)	Hasil Akhir Migrasi dengan RAD (Satuan: Bulan)
Tiket Pesawat	5	2
Tiket Kereta	3	1
Kamar Hotel	4	2
Tiket Bis	2	1
*Batas Lama Migrasi	5	3

Keterangan:

- Batas lama migrasi diambil dari total bulan yang dibutuhkan untuk migrasi sistem.
- Batas lama migrasi untuk estimasi awal migrasi tanpa RAD adalah 5 bulan. 5 bulan diambil dari total bulan maksimal diantara seluruh pengerjaan modul migrasi.
- Batas lama migrasi untuk hasil akhir migrasi dengan RAD adalah 3 bulan. 3 bulan dapat dilihat dari data di tabel 7.

Sebagai bahan penjelasan tentang perbandingan lama migrasi sistem terpusat dengan arsitektur *microservices* antara tanpa dan dengan RAD dapat terlihat di gambar 9.



Gambar 9. Grafik Perbandingan Lama Migrasi Antara Tanpa dan Dengan RAD

5. KESIMPULAN DAN SARAN

Dari keseluruhan penelitian tersebut didapatkan suatu kesimpulan yang cukup berarti. Kesimpulan tersebut adalah bahwa penggunaan arsitektur microservices di dalam pengembangan suatu proyek migrasi sistem dapat menjawab permasalahan yang ada. Permasalahan mengenai kesulitannya menyatukan berbagai macam sistem yang memiliki berbeda teknologi baik dari sisi level front-end maupun back-end ke dalam suatu sistem baru secara terpusat dapat dilakukan dengan baik. Selain itu, kesulitan migrasi dari sistem-sistem yang masih tetap berjalan secara paralel dengan sistem baru terpusat dengan arsitektur microservices ini dapat diatasi juga dengan baik. Kesulitan terakhir mengenai durasi migrasi yang diinginkan dengan lebih cepat dapat diatasi dengan penggunaan model SDLC khususnya metodologi RAD. Terlihat bahwa, metodologi RAD yang diimplementasikan di migrasi ini dapat menghemat waktu pengerjaan lebih banyak. Sehingga, secara keseluruhan durasi dari migrasi sistem terjadi lebih cepat sesuai dengan harapan yang diinginkan.

Sebagai saran dari penelitian ini, peneliti berharap bahwa dapat melakukan implementasi arsitektur microservices dengan metodologi RAD untuk sebuah pengembangan sistem informasi dari awal atau baru. Harapannya bahwa peneliti dapat melihat dan melakukan analisa tentang kegunaan arsitektur microservices dan metodologi RAD untuk pengembangan sistem informasi dari awal atau baru. Kiranya, penelitian dapat bermanfaat sebagai pondasi atau literatur bagi para peneliti yang akan melakukan penelitian terhadap studi kasus yang sama atau dapat dimodifikasi.

DAFTAR PUSTAKA

[1]. Dariawan, Desson. 19 Oktober 2019. Spring Boot RESTful Web Services CRUD Example. https://www.dariawan.com/tutorials/spring/spring-boot-restful-web services-crud-example.

[2]. Dora , Sujit Kumar, and Dubey, Pushkar. 2013. "Software Development Life Cycle (Sdlc) Analytical Comparison And Survey On Traditional And Agile Methodology". National Monthly Refereed Journal Of Research In Science & Technology. Vol No.2, Issue No.8 ISSN 2277-1174. Abhinav Journal Publisher. Odisha.

[3]. Dwidayatour. Tanggal akses 20 April 2020. Tentang Dwidayatour. https://www.dwidayatour.co.id/tentang-kami.

[4]. Fisher, Sharon. 1989. OS/2 EE to Get 3270 Interface Early. Google Books. California.

[5]. Katuwal, Keshab. 2016. "Microservices: A Flexible Architecture for the Digital Age Version 1.0". American Journal of Computer Science and Engineering. Volume 3(3): 20-24. Open Science Publisher. New York.

[6]. Priyadarshini and G.V, Shilpa. 2017. "Microservices: A Flexible Architecture for the Digital Age Version 1.0". International Research Journal of Computer Science (IRJCS). Issue 05, Volume 4 ISSN: 2393-9842. IRJCS Publisher. New Delhi.

[7]. Purwanto. 7 Februari 2017. Metodologi System Development Life Cycle (SDLC). https://medium.com/@purwanto.dev/metodologi-system-development-life-cycle-sdlc-2f0349df1364.

[8]. Richardson, Chris. Tanggal akses 20 April 2020. Pattern: Monolithic Architecture. https://microservices.io/patterns/monolithic.html.

[9]. Spring. Tanggal akses 20 April 2020. What Spring Can Do. https://spring.io/microservices.