PENERAPAN DAN PERBANDINGAN ARSITEKTUR MICROSERVICE TERHADAP ARSITEKTUR MONOLITIK UNTUK KASUS RAWAT JALAN PADA PERANGKAT LUNAK SISTEM INFORMASI MANAJEMEN RUMAH SAKIT

TUGAS AKHIR

Diajukan sebagai syarat untuk menyelesaikan
Program Studi Strata-1 Departemen Informatika

Disusun Oleh:
EDRIC LAKSA PUTRA
1114065

INSTITUT TEKNOLOGI HARAPAN BANGSA

Veritas vos liberabit

DEPARTEMEN INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2018

INSTITUT TEKNOLOGI HARAPAN BANGSA
Veritas vos liberabit

DEPARTEMEN INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA

LEMBAR PENGESAHAN

PENERAPAN DAN PERBANDINGAN ARSITEKTUR MICROSERVICE TERHADAP ARSITEKTUR MONOLITIK UNTUK KASUS RAWAT JALAN PADA PERANGKAT LUNAK SISTEM INFORMASI MANAJEMEN RUMAH SAKIT

Disusun oleh:
Nama: Edric Laksa Putra
NIM : 1114065

Telah Disetujui dan Disahkan

Sebagai laporan Tugas Akhir Departemen Informatika

Institut Teknologi Harapan Bangsa

Bandung, Desember 2017

Disetujui,

Pembimbing

Irfin Afifudin, S.T., M.T.

NIK.

PERNYATAAN HASIL KARYA PRIBADI

Saya yang bertanda tangan di bawah ini:

Nama : Edric Laksa Putra
NIM : 1114065

Dengan ini menyatakan bahwa laporan Tugas Akhir dengan Judul : ” **PENERAPAN DAN PERBANDINGAN ARSITEKTUR MICROSERVICE TERHADAP ARSITEKTUR MONOLITIK UNTUK KASUS RAWAT JALAN PADA PERANGKAT LUNAK SISTEM INFORMASI MANAJEMEN RUMAH SAKIT**” adalah hasil pekerjaan saya dan seluruh ide, pendapat atau materi dari sumber lain telah dikutip dengan cara penulisan referensi yang sesuai.

Pernyataan ini saya buat dengan sebenar-benarnya dan jika pernyataan ini tidak sesuai dengan kenyataan maka saya bersedia menanggung sanksi yang akan dikenakan pada saya.

Bandung, Desember 2017

Yang membuat pernyataan,

Edric Laksa Putra

ABSTRAK

Analisis sentimen pada media sosial telah menjadi salah satu topik penelitian yang paling ditargetkan pada *Natural Languange Processing* (NLP) [2]. Analisis sentimen ini bertujuan untuk menentukan nilai polaritas dari sebuah dokumen secara otomatis. Salah satu tantangan pada analisis sentimen adalah melakukan klasifikasi terhadap teks sarkasme [3]. Dalam penelitian ini, dikembangkan sistem analisis sentimen yang dapat melakukan klasifikasi teks positif, teks negatif, teks netral, dan teks sarkasme. Metode klasifikasi yang digunakan adalah *Support Vector Machine* (SVM). Beberapa fitur yang digunakan untuk memberikan informasi dari dokumen adalah *number of interjection word*, *question word* [5], *unigram, sentiment score, capitalization*, *topic* [4], *part of speech* dan *punctuation based* [3]. Pengujian dilakukan dengan 2 teknik klasifikasi, yaitu *levelled method* dan *direct method* [5]. Berdasarkan pengujian yang dilakukan, hasil akurasi mencapai 72% yang diperoleh menggunakan metode SVM dengan teknik klasifikasi *direct method*.

Kata Kunci: *Natural Language Processing*, *Classification*, *Support Vector Machine*, Analisis Sentimen

ABSTRACT

Sentiment analysis on social media has become one of the most targeted research topics in Natural Languange Processing (NLP) [2]. This sentiment analysis aims to determine the polarity value of a document automatically. One of the challenges in the sentiment analysis is to classify sarcasm text [3]. In this study, developed a system of sentiment analysis that can classify positive text, negative text, neutral text, and sarcasm text. The classification method used is Support Vector Machine (SVM). Some of the features used to provide information from documents are number of interjection word, question word [5], unigram, sentiment score, capitalization, topic [4], part of speech and punctuation based [3]. Testing is done by 2 classification techniques, namely levelled method and direct method [5]. Based on the tests performed, the accuracy result was 72% obtained using the SVM method with the classification technique direct method.

Keyword: Natural Language Processing, Classification, Support Vector Machine, Sentiment Analysis

PEDOMAN PENGGUNAAN TUGAS AKHIR

Laporan tugas akhir yang tidak dipublikasikan terdaftar dan tersedia di Perpustakaan Institut Teknologi Harapan Bangsa, dan terbuka untuk umum dengan ketentuan bahwa hak cipta ada pada pengarang dan pembimbing Tugas Akhir. Referensi kepustakaan diperkenankan dicatat, tetapi pengutipan atau peringkasan hanya dapat dilakukan dengan seizin pengarang atau pembimbing Tugas Akhir dan harus disertai dengan ketentuan penulisan ilmiah untuk menyebutkan sumbernya.

Tidak diperkenankan untuk memperbanyak atau menerbitkan sebagian atau seluruh laporan tugas akhir tanpa seizin dari pengarang atau pembimbing Tugas Akhir yang bersangkutan.

KATA PENGANTAR

Terima kasih kepada Tuhan yang Maha Esa karena dengan bimbingan-Nya dan karunia-Nya penulis dapat melaksanakan Tugas Akhir yang berjudul ”PENERAPAN SUPPORT VECTOR MACHINE UNTUK DETEKSI SARKASME PADA ANALISIS SENTIMEN MEDIA SOSIAL INDONESIA TENTANG PENELITIAN TUGAS AKHIR DEPARTEMEN INFORMATIKA“. Laporan ini disusun sebagai salah satu syarat kelulusan di Institut Teknologi Harapan Bangsa. Pada kesempatan ini penulis menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Tuhan Yang Maha Esa, karena oleh bimbingan-Nya penulis selalu mendapat pengharapan untuk menyelesaikan tugas akhir ini.
2. Ibu Ria Chaniago, S.T., M.T., selaku pembimbing I Tugas Akhir yang senantiasa memberi dukungan, semangat, ilmu-ilmu, saran dan dukungan kepada penulis selama tugas akhir berlangsung dan selama pembuatan laporan tugas akhir ini.
3. Bapak Firhat Hidayat, S.T., M.T., selaku penguji I Tugas Akhir. Terima kasih atas dukungan, semangat, ilmu-ilmu, dan masukan yang telah diberikan kepada penulis dalam menyelesaikan Laporan Tugas Akhir ini
4. Ibu Ir. Inge Martina, M.T., selaku penguji II dalam Tugas Akhir Terima kasih atas dukungan, semangat, ilmu-ilmu, dan masukan yang telah diberikan kepada penulis dalam menyelesaikan Laporan Tugas Akhir ini.
5. Seluruh dosen dan staff Departemen Teknik Informatika ITHB yang telah membantu dalam menyelesaikan Laporan Tugas Akhir ini.

6. Segenap jajaran staf dan karyawan ITHB yang turut membantu kelancaran dalam menyelesaikan Laporan Tugas Akhir ini.
7. Kedua orang tua tercinta yang selalu menyediakan waktu untuk memberikan doa, semangat dan dukungan yang tak habis-habisnya kepada penulis untuk menyelesaikan Laporan Tugas Akhir ini. Terima kasih untuk nasihat, masukan, perhatian, teguran dan kasih sayang yang diberikan hingga saat ini.

Penulis menyadari bahwa laporan ini masih jauh dari sempurna karena keterbatasan waktu dan pengetahuan yang dimiliki oleh penulis. Oleh karena itu, kritik dan saran untuk membangun kesempurnaan tugas akhir ini sangat diharapkan. Semoga tugas akhir ini dapat membantu pihak-pihak yang membutuhkannya.

Bandung, Agustus 2016

Hormat Saya,

Candra Ricky Susanto.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Sistem komputer adalah interaksi dari perangkat lunak dan perangkat keras yang membentuk sebuah jaringan elektronik. Tugas dari sebuah sistem adalah menerima input, memproses data input, menyimpan data olahan, dan menampilkan output sebagai bentuk informasi. Dalam penerapannya, kita menyebut sistem aplikasi sebagai program komputer yang bertugas untuk menyelesaikan kebutuhan khusus. Terdapat beberapa tahapan umum dalam mengembangkan sistem aplikasi yaitu perencanaan, analisa, desain, pengembangan, testing, implementasi, dan pemeliharaan [1]. Tahap yang cukup penting dan akan menjadi fokus diskusi adalah desain dan pengembangan, yang dimana peran arsitektur perangkat lunak sangat berperan penting untuk menetapkan landasan dasar pengembangan aplikasi dari awal sampai selesai. Hasil dari arsitektur perangkat lunak merupakan struktur yang melandasi keberadaan komponen-komponen perangkat lunak, cara komponen untuk saling berinteraksi dan organisasi komponen dalam membentuk perangkat lunak [2]. Arsitektur yang paling sering digunakan saat ini adalah model monolitik. Arsitektur monolitik merupakan arsitektur yang mudah dimengerti dan dimodifikasi karena lebih sederhana implementasinya. Arsitektur ini menggunakan kode sumber dan teknologi yang serupa untuk menjalankan semua tugas-tugasnya. Secara garis besar keunggulan dari arsitektur monolitik dapat dirasakan apabila aplikasi ingin mudah untuk dikembangkan, mudah untuk di deploy, dan dapat selalu dipantau pertumbuhan perfomanya [5]. Namun apabila aplikasi semakin besar dan anggota tim semakin banyak, arsitektur monolitik akan menghadapi kekurangan yang semakin lama akan semakin signifikan. Kelemahan dari arsitektur monolitik dapat dirasakan dari sisi *deployment*, tingkat keamanan, tingkat ketersediaan, dan manajemen aplikasi yang semakin sulit dan kualitasnya menurun.

Pada tahun 2015 sebuah arsitektur microservice merupakan arsitektur alternatif yang dapat mengatasi kelemahan dari monolitik. Arsitektur microservice menjadikan sebuah aplikasi lebih mandiri dan efisien. Namun kendala yang sering menjadi hambatan adalah setiap kasus memiliki masalah dan desain yang berbeda satu dengan yang lain, sehingga dibutuhkan desain yang tepat untuk setiap kasus

yang diangkat. Kendala lain adalah sulitnya implementasi arsitektur baru, dibutuhkan panduan yang menjelaskan tahap migrasi dari arsitektur monolitik hingga menjadi microservice. Pada tahap terakhir, testing dibutuhkan untuk menjadi parameter keberhasilan arsitektur yang baru.
Penelitian ini akan mengangkat contoh kasus penerapan arsitektur microservice pada sistem informasi manajemen rumah sakit Apertura. Penelitian ini bertujuan untuk membuktikan bahwa arsitektur microservice dapat memberikan performa yang lebih baik untuk aplikasi Apertura di kemudian hari.

1.2 Rumusan Masalah

Berikut ini adalah rumusan masalah yang dibuat berdasarkan latar belakang di atas:

1. Bagaimana melakukan migrasi dari arsitektur monolitik ke model arsitektur microservice?
2. Bagaimana melakukan perbandingan performa yang dihasilkan dari arsitektur microservice yang baru terhadap arsitektur monolitik?

1.3 Batasan Masalah

Batasan masalah yang terdapat pada penelitian ini adalah penelitian hanya berfokus pada proses rawat jalan dan tidak melibatkan rawat inap.

1.4 Tujuan Penelitian

Berdasarkan batasan masalah di atas, berikut ini adalah tujuan penelitian dari tugas akhir ini:

1. Menentukan bagaimana tahap yang benar untuk melakukan migrasi dari arsitektur monolitik ke arsitektur microservice.
2. Membandingkan apa saja kelebihan dan kekurangan dari arsitektur microservice dibandingkan dengan arsitektur monolitik.

1.5 Kontribusi Penelitian

Berikut ini adalah kontribusi penelitian yang diberikan pada pengembangan sistem analisis sentimen ini:

1. Memberikan contoh panduan untuk melakukan migrasi dari arsitektur konvensional ke arsitektur microservice pada sistem informasi manajemen rumah sakit.
2. Memberikan hasil analisa perbandingan performa dari kedua jenis arsitektur.

1.6 Metode Penelitian

Tahapan-tahapan yang akan dilakukan dalam pelaksanaan penelitian ini adalah sebagai berikut:

1. Analisis
   Melakukan studi literatur dan analisa melalui wawancara dengan pihak perancang aplikasi monolitik sebelumnya. Data juga dikumpulkan dari jurnal-jurnal, karya ilmiah, dan situs yang memberikan informasi yang menunjang mengenai konsep arsitektur microservice dan tahap-tahap implementasi pada aplikasi monolitik.
2. Identifikasi
   Melakukan identifikasi mengenai target-target yang ingin dicapai setelah menerapkan arsitektur baru. Identifikasi ini berguna untuk menentukan perbandingan apa saja yang akan diperhatikan antara arsitektur microservice dan arsitektur monolitik ketika pengujian dilakukan.
3. Perancangan
   Perancangan arsitektur microservice meliputi perancangan model dasar (proses bisnis), perancangan sistem basis data, perancangan kelas service yang baru dengan konsep microservice, juga perancangan jalur komunikasi dan pertukaran data antara kelas-kelas service yang akan dibuat.
4. Implementasi
   Melakukan implementasi hasil perancangan dalam bentuk web service application (server side). Selanjutnya dibuat aplikasi client sederhana untuk input data dan pengujian performa dari arsitektur server yang telah dibuat.
5. Pengujian
   Melakukan pengujian terhadap rancangan aplikasi microservice yang baru dan aplikasi monolitik dengan menggunakan data rumah sakit untuk mengetahui hasil kinerja dan perbandingan performa antara kedua jenis arsitektur.

1.7 Sistematika Penulisan

Pada penelitian ini peneliti menyusun berdasarkan sistematika penulisan sebagai berikut:

BAB I PENDAHULUAN

Berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan laporan penelitian.

BAB II LANDASAN TEORI

Membahas tentang definisi dari arsitektur microservice, teori-teori pendukung, dan metode penerapan arsitektur baru yang akan dijadikan landasan dan dipelajari serta dirangkum dari berbagai sumber, seperti buku, karya tulis, jurnal, artikel dari situs ilmiah. Juga pembahasan sekilas mengenai arsitektur konvensional yang digunakan sebelumnya.

BAB III ANALISIS DAN PERANCANGAN

Berisi analisa mengenai bagaimana konsep penerapan arsitektur microservice pada aplikasi rumah sakit dan modul-modul apa saja yang akan di migrasi menjadi modul baru untuk diimplementasikan pada tahap selanjutnya. Selanjutnya membahas perancangan aplikasi dari hasil analisa dengan menggunakan arsitektur microservice yang baru.

BAB IV IMPLEMENTASI DAN PENGUJIAN

Berisi implementasi dari hasil perancangan aplikasi dalam bentuk perangkat lunak menggunakan teknologi *web service*. Selanjutnya perangkat lunak diuji fungsi dan performanya dari sisi *client* (pengguna).

BAB V KESIMPULAN DAN SARAN

Memberikan kesimpulan berdasarkan hasil analisis, perancangan, implementasi dan pengujian, serta evaluasi yang dilakukan, juga saran-saran yang dibutuhkan untuk pengembangan lebih lanjut.

BAB II

LANDASAN TEORI

Pada bab ini akan dijelaskan teori pendukung serta metode yang digunakan untuk mengekstraksi modul-modul yang terdapat pada aplikasi monolitik. Penjelasan teori dimulai dengan pengertian dari arsitektur Microservice, prinsip dan pemodela microservice, membangun arsitktur microservice dari arsitektur monolitik, metode pengembangan aplikasi berbasis Microservice, serta aplikasi rumah sakit Apertura sendiri.

2.1 Arsitektur Microservice

Arsitektur aplikasi yang memiliki struktur hubungan service yang renggang namun kolaboratif. Tiap service memiliki fungsi yang lebih sempit dan saling berhubungan. Tiap service ini saling berkomunikasi menggunakan web service dan dapat dikembangkan dan di *deploy* secara mandiri. Tiap service memiliki database masing-masing yang saling memisahkan data.

2.1.1 Definisi Arsitektur Microservice

Aritektur microservice pertama kali muncul untuk memenuhi kebutuhan dan menunjukan bagaimana sebuah aplikasi dapat lebih efektif dalam tahap *production*, juga menunjukan bagaimana cara *development* yang lebih baik dengan memberikan kemampuan kepada mesin untuk saling berkomunikasi. Microservice juga termasuk ke dalam perancangan insfrastrktur mesin sampai skala yang dibutuhkan. Banyak organisasi telah membuktikan dengan berpindah ke arsitektur microservice, aplikasi mereka menjadi lebih cepat dan berani untuk menggunakan teknologi yang baru. Microservice memberikan *developer* kebebasan untuk bereaksi dan mengambil keputusan yang berbeda, memberikan respon yang lebih cepat atas segala kebutuhan dari pengguna aplikasi [9].

2.1.2 Prinsip Pendekatan Arsitektur Microservice

Terdapat beberapa tahapan dalam pendekatan dalam arsitektur mikroservis yang menjadikan desain system yang baik, pendekatan ini berguna untuk mendefinisikan prinsip dan petunjuk yang bergantung pada gol yang kita tuju, tahapan pendekatan tersebut yaitu :

1. *Strategic Goals.* Strategic goals harus memberikan arahan kemana perusahaan

ingin beranjak dan bagaimana memenuhi kebutuhan konsumen. Bahasan ini harus berisi tujuan tertinggi dan tidak membahas teknologi sama sekali. Goals ini bisa dibahas di level perusahaan atau juga di level divisi. Kuncinya adalah untuk membuat kemana arah organisasi akan bergerak [9].

2. *Principles.* Principles adalah aturan yang harus dibuat agar dapat memenuhi goals, prinsip ini kadang berubah sesuai dengan kondisi. Misalnya apabila strategic goals perusahaan adalah untuk mengurangi waktu pengiriman barang-barang baru, maka organisasi terebut akan mendefinisikan prinsip yang mengatakan bahwa tim pengiriman mempunyai kontrol penuh terhadap *lifecycle* produk mereka untuk dikirimkan kapanpun produk siap. Namun apabila goals adalah untuk mengembangkan pertumbuhan produk dengan cepat di sebuah negara, maka organisasi akan memutuskan untuk mengimplementasi prinsip bahwa semua system harus bisa bekerja secara portable agar dapat di *deploy* secara local dan memastikan bahwa data akurat. Prinsip ini juga jangan terlalu banyak, kurang dari 10 adalah angka yang baik, karena semakin banyak prinsip akan beresiko menjadikan aturan-aturan tersebut saling bentrok satu sama lain [9].

3. *Practices.* Tahap ke tiga adalah untuk memastikan semua prinsip telah dilakukan. *Practices* adalah sebuah detail set, bagaimana untuk melakukan task-task agar goals dapat dicapai sesuai dengan aturan yang ada. Tahap ini termasuk dengan spesifikasi teknologi, dan harus cukup sedetail mungkin agar semua *developer* dapat paham. *Practices* dapat termasuk petunjuk bagaimana *lifecycle*coding dilakukan. Sesuai dengan sifat naturalnya, *practices* akan lebih sering berubah dibandingkan dengan principal di tahap ke 2 [9].

4. *Combining Principles and Practices.* Ide dari point terakhir ini adalah ketika system berevolusi dengan ide baru, organisasi tetap siap dengan segala detail yang dibutuhkan agar semua orang tahu bagaimana mengimplementasi ide baru tersebut. Terdengar mudah untuk dilakukan di lingkup yang kecil, namun untuk lingkup besar, bisa terdapat perbedaan antara teknologi dengan praktek yang dilakukan. Misalnya tim .NET akan mempunyai set *practices* yang berbeda dengan tim Java [9].

2.1.3 Konsep Microservice

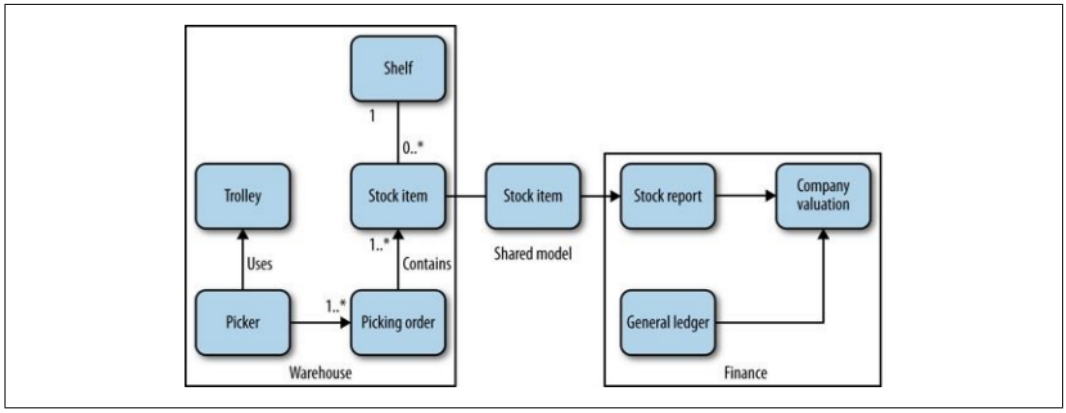
Setelah pengertian umum mengenai arsitektur microservice, pada bagian ini akan dijelaskan bagaimana cara berfikir dengan batasan-batasan microservice yang akan memaksimalkan semua potensinya. Dalam point ini peneliti

menginginkan pembaca fokus terhadap dua konsep kunci microservice, yaitu *loose coupling* dan *high cohesion*.

Loose Coupling. Ketika service telah *loosely coupled*, perubahan yang dilakukan terhadap satu service tidak akan mengakibatkan perubahan pada service yang lain. Prinsip ini menekankan bagaimana microservice dapat melakukan perubahan pada satu service dan melakukan *deploy* tanpa harus melakukan perubahan apapun pada sistem. Namun sebuah sistem dapat memiliki kebutuhan berkomunikasi antar service, hal ini mengakibatkan arsitek harus membatasi limit panggilan dari satu service terhadap service yang lain, karena selain dapat menyebabkan masalah performa, hal ini pula dapat mengakibatkan terjadinya *tight coupling* [9].

High Cohesion. Model microservice menginginkan sifat-sifat yang berkaitan untuk berada di satu wadah, dan yang tidak berkaitan ditempatkan di wadah yang lain, karena apabila ada perubahan yang terjadi, hanya satu wadah tersebut yang akan berubah dan perubahan dapat langsung di implementasikan dengan cepat. Apabila service dibuat terlalu tercecer, maka akan menyebabkan perubahan di banyak tempat dan akan membuang banyak waktu. Point yang diinginkan adalah menempatkan service dengan sifat yang mirip di satu wadah, namun tetap berkomunikasi dengan wadah lain selonggar mungkin [9].

Dalam buku yang dijelaskan Sam Newman, penulis mengambil contoh sebuah departemen keuangan dan departemen *warehouse* di sebuah organisasi untuk menjelaskan tentang shared dan hidden model. Kedua departemen mempunyai *interface* yang berbeda ketika ditampilkan. Departemen keuangan tidak perlu tahu segala detail di *warehouse*. Namun walau begitu tetap ada data yang dibutuhkan seperti misalnya stok barang agar mendapatkan perhitungan terbaru. Pada model microservice maka ke dua modul ini akan dibuat terpisah. Berikut penggambarannya :



Gambar 2.1 Model pembagian dari departemen keuangan dan warehouse.

Untuk dapat menjalankan alur informasi, pegawai keuangan membutuhkan data stok. *Stock item* menjadi *shared model* antara dua departemen. Perlu diingat bahwa tidak semua data *warehouse* harus diperlihatkan di keuangan, jadi terdapat representasi internal dan representasi external yang diperlihatkan. Desain diatas memperlihatkan konsep *loose coupling* dan *high cohesion* yang digambarkan menjadi sebuah modul. Desain seperti ini sangat mempermudah proses perpindahan dari monolitik dan menyakinkan bahwa desain microservice telah *loosely coupled* dan *stongly cohesive* [9].

2.2 Integrasi Teknologi

Mengintegrasikan dengan benar merupakan tahap yang paling penting, memungkinkan perubahan yang signifikan dengan tingkat kemandirian aplikasi yang tinggi. Dalam tahap integrasi ini ada beberapa point penting yang harus dianalisis sebelum memilih teknologi yang digunakan dan mengimplementasikannya.

1. ***Menjaga teknologi API agar tetap agnostik.*** IT industri adalah berubah dengan sangat cepat, *tools* baru, *framework* dan bahasa baru, serta ide-ide implementasi yang selalu berkembang. Hal inilah yang menjadi pertimbangan agar memastikan bahwa API inisial harus dapat digunakan terus menerus ketika mengimplementasikan microservice.

2. ***Hindari perubahan major pada aplikasi.*** Perubahan arsitektur dapat mengakibatkan perubahan pada bagian-bagian aplikasi yang lainnya, pemilihan teknologi yang tepat bertujuan agar perubahan ini terjadi sekecil mungkin.

3. ***Buat service sederhana untuk dipakai.*** Arsitektur microservice yang baru harus cepat beradaptasi dengan penggunanya, maka dari itu modul service harus bersifat *user-friendly*.

Seperti yang dikutip dari website Chris Richardson, microservice merupakan sekumpulan teknologi yang saling bekerjasama. Teknologi tersebut tidak dibatasi oleh sebuah wadah tertentu, namun saling terpisah yang menyebabkan luasnya pemilihan teknologi yang akan digunakan. Point berikutnya akan menjelaskan beberapa pilihan teknologi yang baik yang dapat diimplementasikan. Mulai dari lapisan paling dalam, yaitu managemen data, pembagian service, metode berkomunikasi antar service, sampai dengan API gateway yang akan digunakan oleh clien [9].

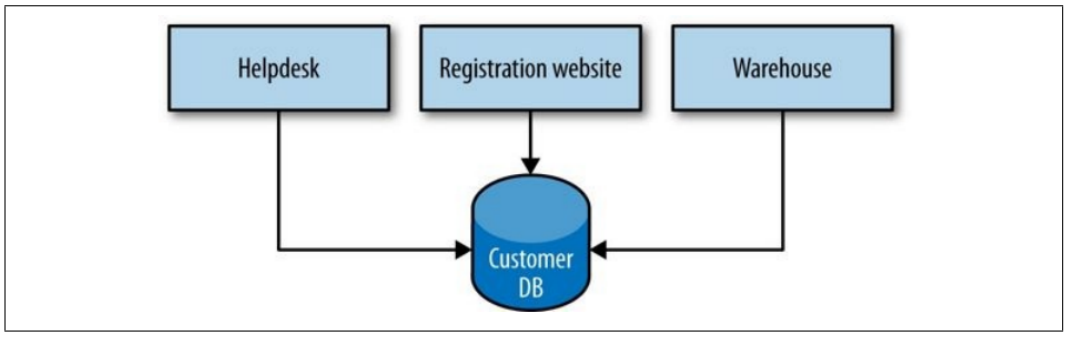
2.2.1 Manajemen Data

Bab ini akan membahas bagaimana penyimpanan data pada arsitektur monolitik dan apa kelemahannya. Kemudian dilanjutkan dengan penjelasan dan pembahasan metode penyimpanan data yang baik dan sesuai dengan arsitektur microservice.

2.2.1.1 Model Penyimpanan Data Arsitektur Monolitik

Pada arsitektur biasa, umumnya database disimpan dalam 1 tempat dan terdiri dari beberapa table. Ketika terjadi permintaan untuk membaca data, maka sistem akan mengambil data tersebut dari database, sama halnya apabila data diubah, maka sistem akan langsung mengubah database. *Life cycle* seperti ini sangat simpel dan sangat cepat sehingga sampai saat ini dipakai oleh banyak sistem.

Contoh dalam gambar 2-1, *customer* yang akan melakukan registrasi akan melakukan *querry* ke database, juga aplikasi *call center* yang menampilkan dan mengubah data akan langsung melakukan *querry* ke database, begitu pula dengan informasi *update warehouse* mengenai pesanan konsumen, akan melakukan *query* pada database. Ini adalah contoh pattern yang sangat umum, namun banyak kelemahan dari pattern database ini [9]



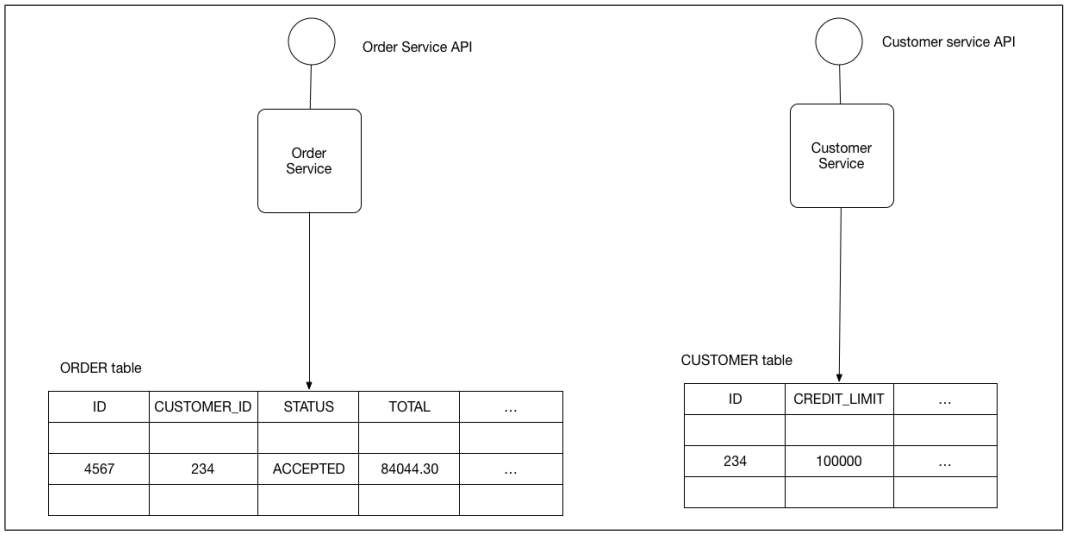
Gambar 2.2 Pemodelan cara akses database yang umum.

Pertama, model ini mengizinkan langsung pihak luar untuk mengubah data internal. Struktur data yang di simpan di DB dipakai oleh semua *user*, apabila terjadi perubahan pada DB, maka semua *user* akan terkena dampaknya. Database menjadi sangat besar, dan *shared* API menjadi rapuh. Apabila akan terjadi perubahan, misalnya perubahan table customer di database, maka harus sangat berhati-hati agar *schema* yang dipakai oleh service lain tidak rusak. Hal ini membutuhkan usaha testing regresi yang besar. Hal ini melanggar konsep *loose coupling* [9]. Kedua, semua *client* menjadi terikat dengan sebuah teknologi spesifik. Mungkin saat ini database berjalan dengan baik dengan menggunakan

relational database, namun bagaimana bila seiring berjalannya waktu, performa untuk menyimpan data lebih baik menggunakan *nonrelational* database? *Client* menjadi terikat dengan model implementasi. Hal ini melanggar konsep *cohesion*.

2.2.1.2 Model Penyimpanan Data Arsitektur Micoservice

Dalam perancangan arsitektur microservice, terdapat 2 pattern database ditinjau dari pembagian database tersebut. Pattern pertama adalah *shared database* dan yang kedua adalah *database per service*. Untuk contoh kedua pattern, penulis Chris Richardson memberikan contoh dengan menggunakan modul *customer* dan modul *order*. Hubungan kedua modul tersebut terjadi ketika ada transaksi baru, dimana modul *order* harus memastikan bahwa jumlah pesanan yang baru tidak melebihi limit kredit yang dimiliki *customer*. Relasi kedua modul itu dapat dilihat dari gambar dibawa [6]



Gambar 2.3 *Shared database.*

Shared database. Model yang pertama adalah database yang di *share* untuk diakses oleh beberapa service. Modul *order* bisa langsung mengakses table *customer* untuk mendapatkan limit kredit *customer* tersebut. Model ini dapat menjadi pilihan ketika *developer* ingin model yang familiar dan tegas untuk menjaga konsistensi data. Model *shared database* memiliki beberapa kelemahan, antara lain :

1. *Development time coupling.* Modul *order* harus tahu apabila terjadi perubahan pada *schema customer*, hal ini menjadikan kedua modul menjadi memiliki ketergantungan. Hal ini akan memperlambat proses *development*.

2. *Runtime coupling*. Karena beberapa service dapat mengakses database yang sama, terdapat potensi mengganggu proses yang lain, misalnya ketika modul *customer* sedang melakukan *update* terhadap *customer* dengan id 234 untuk merubah kredit limit, maka modul *order* harus menunggu sampai *update customer* selesai dilakukan karena *customer* dengan id 234 akan di *block* sementara waktu.

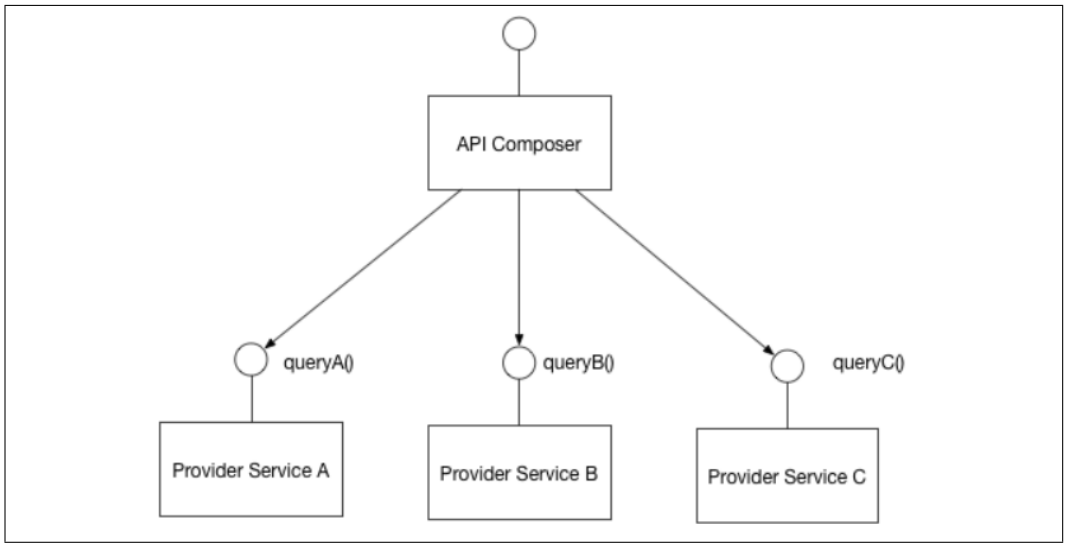
Database per service. Pattern database yang kedua menjadikan sebuah table menjadi *private* hanya untuk 1 buah service dan hanya dapat diakses via API, service lain tidak bisa mengakses database tersebut. Kelebihan dari model ini adalah menjadikan service lebih *loose coupled*, tingkat ketergantungan antar service rendah. Tiap service pun dapat memiliki database yang cocok untuk dirinya sendiri. Namun model database ini memiliki beberapa kesulitan, antara lain :

1. Mengimplementasikan proses bisnis yang melibatkan banyak service menjad lebih sulit, dan lebih baik dihindari karena akan menemukan kesulitan di integritas data, terutama database modern (NoSQL). Solusi terbaik adalah dengan menggunakan konsep SAGA pattern (dibahas pada point berikutnya), yang menjalankan *event* ketika ada perubahan data. Service lain yang men-*subscribe event* tersebut akan merespon dan melakukan *update* [6].

2. Mengimplementasi *query* yang menggabungkan data dari banyak database akan lebih sulit [6].

2.2.2 *API Composition*

Ketika mengaplikasikan arsitektur microservice menggunakan pattern *database per service*, praktis implementasi dari *straightforward query* yang menggabungkan data lebih dari satu tabel tidak dapat dilakukan lagi, karena tabel dalam database terpisah di berbagai lokasi service. Solusi untuk dapat mengatasi permasalahan ini adalah dengan menggunakan *API Composition*. Ide dari konsep ini adalah menggabungkan data hasil service dan melakukan *in-memory* join di level sistem. Namun kelemahan dari metode ini adalah menyebabkan data join yang besar dalam *memory* system. Untuk menangani masalah tersebut, dapat dilakukan dengan meningkatkan performa *hardware*, misal menambah *core* dari *processor* dan menambah *memory* sistem [6].



Gambar 2.4 Meminta data dari service yang dibutuhkan dan melakukan *in-memory join*

2.2.3 Dekomposisi Modul

Tujuan dari dekomposisi modul ini adalah menemukan service terkecil penyusun aplikasi. Dengan memisahkan modul menjadi sebuah service tunggal, menjadikan aplikasi lebih mandiri dan mudah untuk di koordinasikan dalam tim, yang dapat mempercepat proses *development*. Tujuan yang lebih besar lagi adalah menghindari perubahan besar pada aplikasi ketika ada proses bisnis yang berubah. Dengan mendekomposisi modul, akan membantu untuk memastikan hanya ada sebuah service saja yang akan terkena dampaknya.
Tantangan yang didapat ketika hendak melakukan dekomposisi modul adalah :

1. Rancangan arsitektur service yang baru harus stabil.
2. Sebuah service harus terdiri dari susunan *functions* yang erat fungsinya.
3. Service harus memastikan tidak melanggar *Common Closure Principle*, yaitu apabila terjadi perubahan tidak akan melibatkan *package* lain.
4. Setiap service sebagai API yang terenkapsulasi dari pengguna. Segala perubahan tidak boleh memberikan dampak secara langsung pada pengguna.
5. Sebuah service harus cukup kecil untuk ditangani oleh tim yang terdiri dari 6-10 orang.
6. Service harus bisa di tes, dan setiap tim harus dapat melakukan proses *develop* dan *deploy* tanpa banyak interaksi dengan tim lain.

Terdapat 2 metode dalam mendekomposisi modul aplikasi, yaitu berdasarkan

subdomain dan berdasarkan proses bisnis yang dilakukan [6].

Dekomposisi berdasarkan subdomain. Apabila aplikasi yang dibentuk terpisah berdasarkan *domain-driven design* (DDD) atau disebut juga subdomain, maka metode ini dapat lebih mudah digunakan. DDD memisahkan aplikasi berdasarkan kebutuhannya dan membentuk sebuah subdomain sendiri, setiap subdomain bertanggung jawab untuk menangani proses bisnis yang berbeda.Subdomain dapat diklasifikasikan sebagai berikut :

1. *Core* – kunci pembeda dari proses bisnis dan menjadi bagian yang sangat penting dari sebuah aplikasi.
2. *Supporting* – berhubungan dengan proses bisnis namun bukan menjadi pembeda.
3. *Generic* – tidak berhubungan dengan proses bisnis dan biasanya hanya menjadi proses pendukung saja.

Kelebihan dari metode ini antara lain:

1. Arsitektur service baru yang stabil, karena subdomain sendiri secara relatif sudah stabil.
2. Service yang dibentuk akan cenderung memenuhi *loosely coupled* dan *cohesive*.

Masalah yang dihadapi dari metode ini antara lain:

1. Sulit diterapkan untuk aplikasi yang tidak *domain-driven* atau aplikasi yang memiliki keterikatan modul yang kuat.

Dekomposisi berdasarkan proses bisnis. Pattern alternatif yang lain adalah mendefinisikan service berdasarkan kemampuan bisnis. Kemammpuan bisnis adalah proses yang dilakukan untuk menghasilkan sebuah nilai. Kemampuan bisnis biasanya berhubungan dengan objek bisnis. Misalnya *order management* bertanggung jawab untuk *orders*, *customer management* bertanggung jawab untuk *customer*. Kemampuan bisnis biasanya tersusun menjadi hirarki multi level, misalnya untuk aplikasi perusahaan memiliki *product/service development*, setelah itu ada *product/service delivery*, lalu diikuti dengan *aftersale service* [6] Kelebihan dari metode ini antara lain:

1. Arsitektur service baru yang stabil, karena kemampuan bisnis pun relatif sudah stabil.
2. Service yang dibentuk akan memenuhi konsep *loosely coupled* dan *cohesive*.

Masalah yang akan dihadapi dari metode ini antara lain:

1. Sulit untuk mengidenfitikasi kemampuan bisnis aplikasi. Identifikasi pembentukan service membutuhkan pemahaman dari proses bisnis, maka harus dilakukan analisa mendalam dari tujuan perusahaan, struktur, dan cakupan area. Analisa dapat dilakukan pertama-tama dari struktur organisasi, karena perbedaan grup dalam organisasi berkaitan dengan kemampuan bisnis dari grup tersebut.

2.2.4 Service Deployment

Setelah mengindentifikasi dan membentuk service, tahap selanjutnya yang akan dilakukan adalah melakukan *deployment* service untuk nantinya digunakan. Tiap service sendiri bisa ditulis menggunakan bahasa dan framework yang berbeda-beda, namun secara mandiri harus *deployable* dan *scalable*. Dalam beberapa kasus, akan terdapat set service yang harus terisolasi dari service lainnya. Kebutuhan lainnya adalah adanya keperluan untuk dapat memantau penggunaan *resources* (CPU dan memori) yang digunakan oleh service dan memantau dengan mudah sifat dan kegunaan dari service tersebut. Semua kebutuhan ini juga harus sebanding dengan biaya yang dikeluarkan.
Terdapat beberapa metode *deployment* yang dapat dilakukan, tiap metode cocok digunakan sesuai dengan kebutuhan dari *deployment* itu sendiri. Point berikutnya akan menjelaskan metode-metode *deployment* service.

1. **Multiple service instances per host.**
   Metode *deployment* ini cocok digunakan apabila service yang terbentuk tidak banyak dan adanya kebutuhan untuk menghemat penggunaan sumber daya. Set service yang terbentuk disimpan dalam sebuah host (fisik atau *virtual machine*). Terdapat 2 cara dalam melakukan *deploy* set service dalam sebuah host. Pertama, *deploy* set-set service tersebut sebagai JVM, misalnya dengan Tomcat atau Jetty untuk sebuah set service. Kedua adalah melakukan *deploy* semua set service dalam sebuah JVM yang sama, mislanya sebagai web aplikasi [6].
   Metode deployment ini juga terdapat beberapa kelemahan, antara lain:

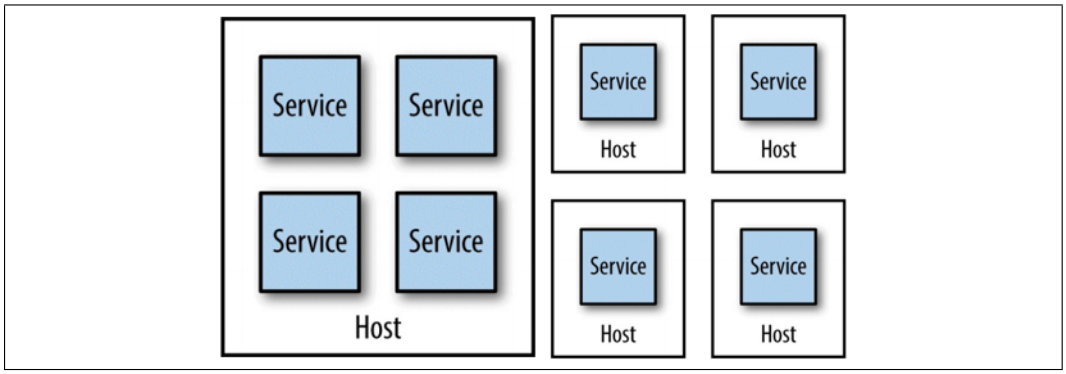
   (a) Beresiko terjadi konflik terhadap kebutuhan sumber daya (memori dan CPU).

   (b) Beresiko terjadi konflik versi dependency.

   (c) Sulit untuk membatasi konsumsi sumber daya yang digunakan sebuah service (berhubungan dengan point pertama).

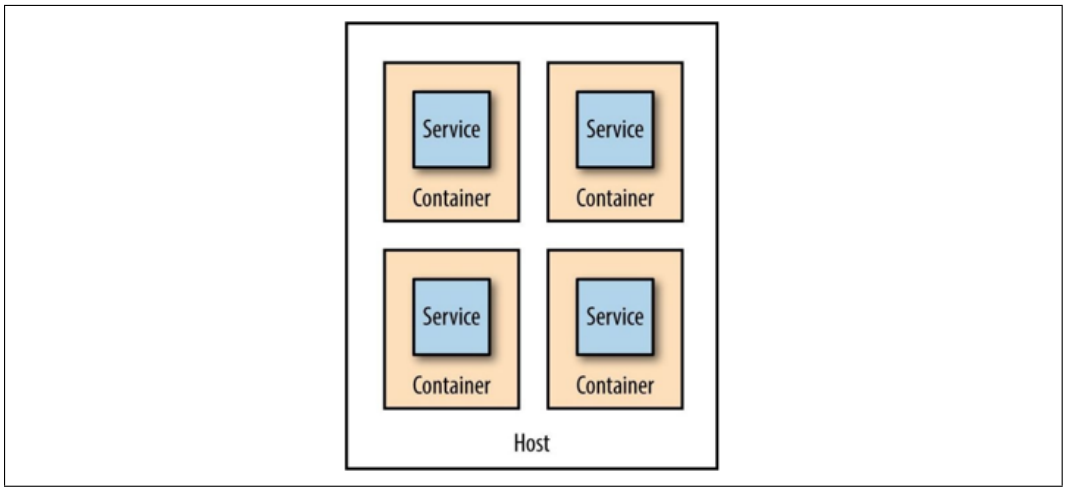
(d) Apabila banyak set service di deploy dalam sebuah mesin yang sama, maka akan sulit untuk memonitor konsumsi sumber daya dari tiap service, karena sulit untuk melakukan isolasi terhadap service.

2. **Single service instance per host.**
   Pilihan yang kedua adalah setiap service memiliki hostnya sendiri. Dengan menerapkan pattern ini, setiap service akan terisolasi dari yang lainnya, serta tidak akan ada resiko konflik akan sumber daya dan dependency. Setiap service lebih mudah untuk dimonitor dan dikelola. Namun kekurangan dari patern ini apabila dibandingkan dengan multiple service per host adalah kurangnya efisiensi penggunaan sumber daya, karena bisa jadi akan ada banyak host yang digunakan [9]



Gambar 2.5 *Banyak service per host dan satu service per host.*

3. **Service instance per container.**
   Apabila masing-masing service dibuat dengan bahasa atau framework yang berbeda, maka proses *deployment* dari setiap service akan berbeda-beda pula. Dengan menggunakan *container*, semua detail teknologi yang digunakan oleh setiap service akan dibuat terenkapsulasi dari service lain. *Container* juga menspesifikasikan dengan jelas bagaimana proses *deployment* yang harus dilakukan dalam sebuah mesin, sehingga ketika sebuah aplikasi hendak dijalankan dalam mesin yang berbeda, user tidak perlu tahu proses apa saja yang harus dilakukan [9]. Contoh dari *container* adalah Docker, namun Docker tidak bisa melakukan *deployment* dalam banyak mesin. Maka dari itu Google mengembangkan *tools* yang bernama Kubernetes, Kubernetes memungkinkan agar Docker bisa dalam satu saat bersamaan dijalankan dalam banyak mesin sekaligus [9]



Gambar 2.6 *Menggunakan container untuk deployment.*

4. **Serverless deployment.**
   Ide dari serverless deployment adalah mengurangi interaksi dari pengguna dengan server. Segala hal yang berhubungan dengan infrastruktur server disembunyikan dari pengguna. Pengguna hanya dikenakan biaya penyewaan server saja, namun tidak perlu lagi melakukan pengaturan apapun pada server. Untuk melakukan deployment, pengguna meembuat package dari kode (misalnya ZIP file), lalu melakukan upload kepada penyedia jasa server dan melakukan pengaturan performa. Ada beberapa penyedia jasa lingkungan serverless, misalnya AWS Lambda, Google Cloud Function, Microsoft Azure [6].
   Kelebihan dari penggunaan serverless ini antara lain:

   (a) Tidak perlu membuang-buang waktu untuk mengurus menejemen infrastruktur low-level. Pengguna bisa lebih fokus untuk mengembangkan aplikasinya saja.

   (b) Arsitektur dari serverless sangat elastis. Server secara otomatis menghitung beban dari service yang digunakan agar tidak ada resource yang terbuang.

   Kekurangan dari serverless antara lain:

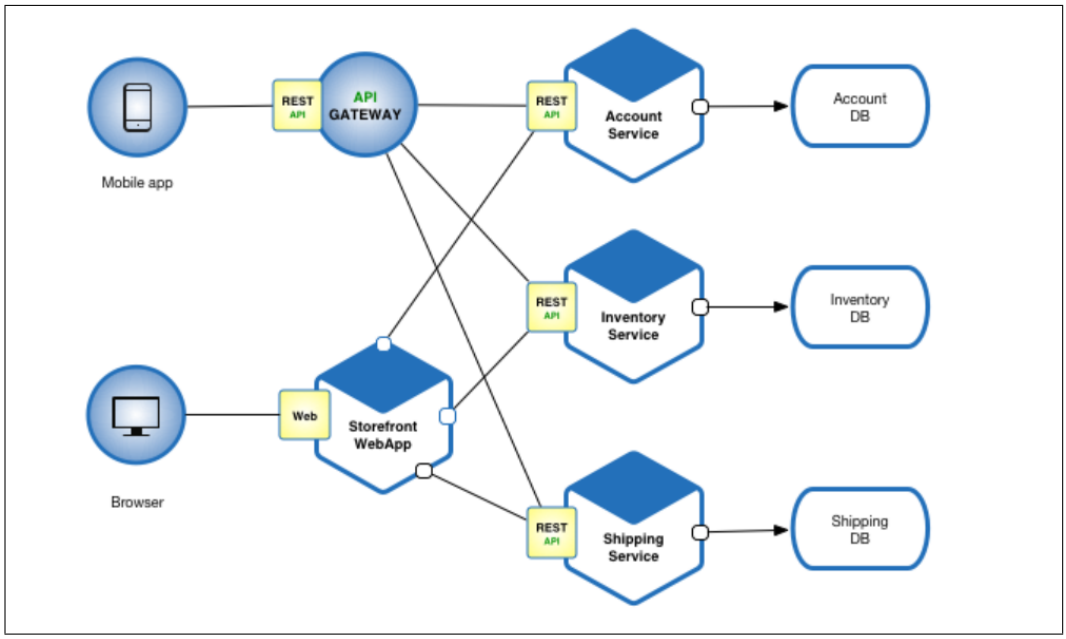
   (a) Adanya batasan lingkungan, misalnya server hanya bisa support untuk beberapa bahasa saja.

2.2.5 Metode Berkomunikasi Antar Service

Dengan memiliki modul service yang berbeda-beda, timbul sebuah masalah yang berkaitan dengan pertukaran informasi yang berasal dari banyak

service. *Remote Procedure Invocation* (RPI) adalah protocol yang menyediakan teknik komunikasi antara service yang berbeda lokasi. Teknologi RPI ini ada yang menggunakan kode biner sebagai format pertukaran data, ada pula yang menggunakan format pesan XML seperti SOAP. Implementasi RPI ini berguna untuk mendapatkan data dengan sangat cepat yang dikirimkan melalui jaringan, hal yang menjadi keuntungan utama dari RPI adalah kemudahan penggunaannya. Contoh RPI lain yang menjadi fokus disini adalah Representational State Transfer (REST), point berikutnya akan menjelaskan mengapa REST menjadi pilihan terbaik untuk menangani proses komunikasi di microservice [9].

Representational State Transfer (REST). REST adalah standar arsitektur web yang menggunakan protokol HTTP. HTTP sendiri mempunyai kemampuan yang sangat cocok untuk REST, salah satunya HTTP faham apa yang harus dilakukan apabila menerima perintah GET, POST, PUT dari REST. Kelebihan penting yang dimiliki REST adalah pengguna bisa menghindari kontak langsung dari pengguna dengan server secara langsung. Konsep ini kemudian disebut sebagai *hypermedia as the engine of application state* (HATEOAS). *Hypermedia* adalah konsep dimana sebuah konten mempunyai link yang berhubungan dengan konten lainnya yang bisa berupa berbagai format (text, gambar, suara) [9]. Ide dari HATEOAS adalah *client* berhubungan dengan server hanya dengan menggunakan *link* yang telah disediakan. Misalnya seperti gambar 2.6 dibawah. Ketika pengguna ingin mengubah data *account*, maka pengguna akan mengirimkan *request* pada *account service*, *account service* dengan logic yang dimilikinya akan menentukan apakah request tersebut dapat diterima. *Account service* disini menjaga semua interaksi yang berhubungan dengan data *account* itu sendiri. *Client* tidak perlu tahu dan tidak perlu beradaptasi apabila terjadi perubahan pada server. *Client* akan merasakan perubahan hanya apabila terjadi perubahan sifat atau ketika hilangnya kontrol yang merepresentasikan *account*. Format data yang dikirimkan REST di HTTP dapat beragam, namun yang paling populer adalah format JSON, karena JSON mudah dimengerti dan mudah dikonsumsi langsung. REST pada HTTP sangat baik untuk diimplementasikan pada interaksi *sevice-to-service* [9]



Gambar 2.7 *Contoh pemanfaatan REST sebagai media user-server*

2.3 Strategi Pengujian

Strategi pengujian berguna untuk memberikan gambaran dari test yang akan dilakukan terhadap *software*. Testing ini berguna untuk memberi tahu kepada proyek menejer, tester, dan tim pengembang apabila ditemukannya masalah dalam *software*. Strategi pengujian ini termasuk tujuan dari test, metode yang digunakan, sumber daya yang digunakan, juga lingkungan ketika menjalankan proyek. Test strategi mendeskripsikan seberapa tinggi resiko kesalahan (kegagalan) yang dapat terjadi [12].

2.3.1 Pengujian Arsitektur Microservice

Uji kelayakan arsitektur microservice menurut Martin Fowler adalah dengan melakukan 5 test yang mirip dengan software testing pada umumnya, test tersebut antara lain adalah:

1. *Unit Testing*. Bagian terkecil dalam software yang ditest untuk menentukan apakah unit tersebut bekerja sesuai harapan. Umumnya, unit yang dimaksud adalah kelas penyusun atau grup kecil yang menghubungkan kelas-kelas tersebut dan method yang terdapat dalam kelas.

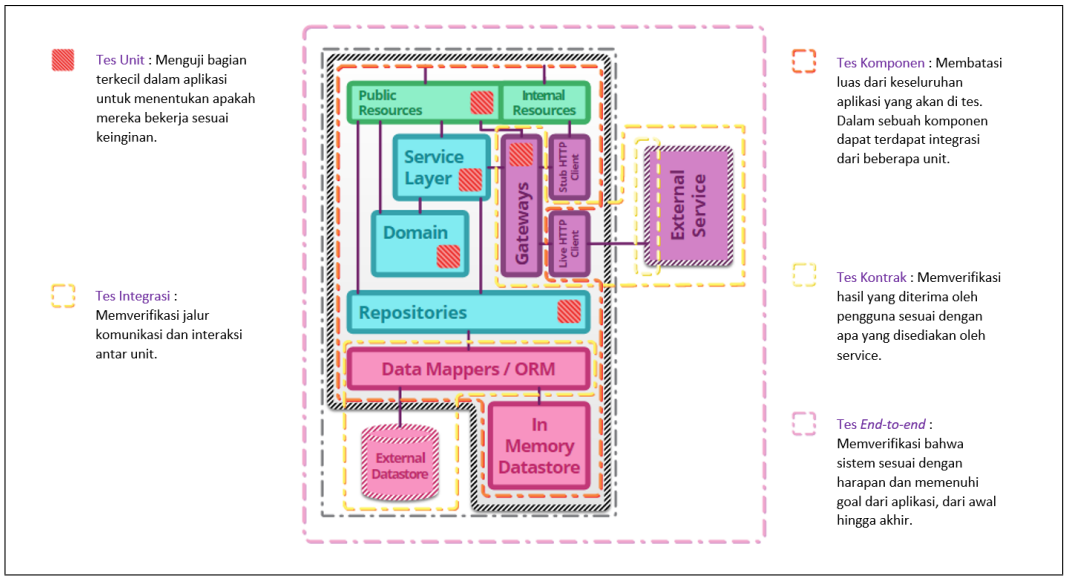
2. *Integration Testing*. Goal dari test ini adalah membuktikan bahwa tidak terdapat masalah dari komunikasi dan interaksi dari setiap unit.

3. *Component Testing.* Komponen membatasi lingkup sebuah software. Dalam 1

buah komponen, dapat terdiri dari beberapa integrasi yang terjadi antara unit kelas.

4. *Contract testing.* Test yang memverifikasi bahwa pihak luar yang mengakses sebuah service akan mendapatkan hasil yang sesuai dengan harapan.

5. *End-to-end Testing.* Memverifikasi bahwa sistem memenuhi telah berhasil memenuhi kebutuhan dan sesuai dengan rancangan goal. Test dilakukan dari awal sampai dengan output yang dihasilkan.



Gambar 2.8 *Testing pada software microservice*

Tujuan utama dari testing yang pertama adalah memastikan bahwa fungsi bisnis dari arsitektur yang baru sudah memenuhi *requirement* yang sama dengan arsitektur monolitik sebelumnya.

2.3.2 Uji Perbandingan Performa

Test kedua adalah menunjukan bahwa performa dari arsitektur microservice akan memberikan hasil yang lebih baik dari monolitk. Strategi pengujian yang akan dilakukan adalah dengan membuat point-point pembanding yang akan diuji, kemudian dari point tersebut akan dibuat *test-plan* masing-masing ang terdiri dari sekitar 1-3 buah *test-plan*. Hasil tes kemudian akan disimpan dalam dokumen berupa *matrix traceability*. Menurut Paulo Merson (Software Architecture di TCU; SOA/microservice trainer dan consultan), point pembanding tersebut adalah [12]:

1. *Deployability*. Lebih agile untuk meluncurkan versi terbaru karena siklus *build, test, build* yang lebih pendek. Tingkat fleksibilitas yang lebih tinggi untuk

menggunakan layanan keamanan, replikasi, persistensi, dan konfigurasi pemantauan.

2. *Reliability*. Kesalahan dalam microservice hanya akan berpengaruh pada microservice itu sendiri dan konsumennya, sedangkan dalam model monolitik kesalahan layanan dapat merusak seluruh sistem monolitik.

3. *Availability*. Waktu *downtime* yang dibutuhkan ketika ingin mengeluarkan versi terbaru dari microservice lebih sedikit dibandingkan monolitik.

4. *Scalability*. Setiap microservice dapat diskalakan secara terpisah menggunakan *pool, cluster, grid*. Karakter menyebar membuat microservice lebih fleksibel dibandingkan dengan monolitik.

5. *Modifiability and Management*. Sifat fleksibel untuk menggunakan *framework, libraries*, dan sumber data baru karena ditunjang sifat *loose-coupled* yang dimiliki microservice, modular komponen hanya dapat diakses oleh contracts (pihak yang berhak), dan sifat yang cenderung tidak mudah berubah menjadi software besar yang sulit ditangani. Upaya pengembangan aplikasi juga dapat dibagi dalam tim yang lebih kecil dan bekerja lebih mandiri.

2.4 Tinjauan Studi

Tabel 2.1 Tabel Tinjauan Studi

No	Peneliti	Judul	Tahun	Konsep
1	Sam Newman	*Building Microservice*	2015	Menjelaskan pendekatan yang paling tepat untuk membangun service yang saling berkolaboratif dan memberi panduan untuk mengintegrasikan teknologi agar menjadi padu dengan service yang dibangun.

Tabel 2.1 Tabel Tinjauan Studi

No	Peneliti	Judul	Tahun	Konsep
2	Susan J. Fowler	*Microservices in Production*	2017	Menjelaskan standar dan goal yang harus dicapai dari perancangan microservice. Standar yang harus dipenuhi tersebut terdiri dari performa, stabilitas, availabilitas, reliabilitas, dan skalabilitas.
3	K. Siva Prasad Reddy	*Beginning Spring Boot 2*	2017	Memberikan panduan untuk membangun *web service* menggunakan REST API dan bekerja dengan *NoSQL database* menggunakan framework java yang bernama Spring.
4	Chris Richardson	*Microservices Patterns*	2017	Memberikan panduan untuk melakukan migrasi sistem dari arsitektur monolitik menjadi arsitektur microservice. Menjelaskan teknologi yang dapat diintegrasikan dengan microservice dan cara implemetasinya.

Pada penelitian satu, Sam Newman memulai dengan memberikan pengenalan dari microservice lalu dilanjutkan dengan keuntungan dan kerugian dari arsitektur tersebut. Kemudian Sam Newman melanjutkan dengan memberikan

cara untuk memodelkan service menggunakan *domain-driven* desain untuk membantu analisa berpikir. Tahap selanjutnya menjelaskan integrasi teknologi yang dapat membantu perancangan desain. Pada bagian akhir Sam Newman memberikan contoh *splitting* monolitik menjadi microservice dan bagaimana *deployment* dilakukan [9].

Pada penelitian dua, Susan J. Fowler menjelaskan bahwa terdapat 5 standar yang menjadi parameter microservice dapat dinyatakan berhasil pada tahap *production*. Ke 5 standar tersebut adalah availabilitas yang dapat ditunjukan dengan seberapa lama waktu *downtime* yang terjadi pada sistem. Stabilitas yang ditunjukan dengan kestabilan dari sisi *development, deployment,* dan tahap pengenalan. Reliabilitas yang dapat ditunjukan dengan proses *deployment* yang terjamin, *planning* dan perlindungan ketika terjadi kegagalan pada sistem dan tingkat keamanan dari segi pertukaran data. Skalabilitas yang menunjukan bahwa pertumbuhan sistem dapat didefinisikan secara kuantitatif, identifikasi sumber masalah *bottlenecks*, dapat direncanakan kapasitas yang dibutuhkan sistem, dan dapat *data storage* yang dapat diprediksi. Performa yang dapat diukur dari kemampuan sistem menangani masalah yang diberikan dan seberapa efisien sistem menggunakan sumber daya yang tersedia [10].

Pada penelitian tiga, *Beginning Spring Boot 2* memberikan panduan dalam menggunakan salah satu *framework* populer dari java yaitu Spring *framework*. Pada awal panduan, penulis memberi tahu pembaca untuk menyiapkan *JDK 1.8*, *IDE Netbeans* atau *Eclipse*, *build tools* seperti *maven*, dan server basis data seperti *MySQL* atau *PostgreSQL*. K. Siva Prasad Reddy kemudian memulai penjelasan dari apa itu *Spring Boot* dan bagaimana *framework* ini dapat membantu produktivitas *developer*, kemudian bagaimana *auto configuration* dari Spring Boot bekerja dibelakang layar, dan bagaimana membuat *Spring Boot starters*. Kemudian K. Siva Prasad Reddy melanjutkan pembahasan sampai bagaimana membuat *Reactive Web Application* dan membuat REST API menggunakan *Spring Boot* [11].

Pada *chapter* pertama di penelitian empat, Chris Richardson mendeskripsikan arsitektur microservice serta kelebihan dan kekurangannya, serta bagaimana microservice dapat menyelesaikan masalah dari monolitik. *Chapter* kedua menjelaskan bagaimana untuk merubah *aplikasi* menjadi *set of services* dengan menggunakan 2 strategi dekomposisi. *Chapter* tiga menjelaskan REST API sebagai media komunikasi *web service* dan bagaimana REST API bekerja. Pada *chapter* terakhir Chris Richardson membahas cara *deployment* microservice

[13].

2.5 Objek Penelitian

Pada bagian ini akan dibahas objek-objek yang terkait dengan analisis penelitian.

2.5.1 Perangkat Lunak

Rekayasa perangkat lunak adalah sebuah profesi yang dilakukan oleh seorang perekayasa perangkat lunak yang berkaitan dengan pembuatan dan pemeliharaan aplikasi perangkat lunak dengan menerapkan teknologi dan praktik dari ilmu komputer, manajemen proyek, dan bidang-bidang lainnya. Perangkat lunak adalah instruksi langsung komputer untuk melakukan pekerjaan dan dapat ditemukan di setiap aspek kehidupan modern dari aplikasi yang kritis untuk hidup *(life-critical)*, seperti perangkat pemantauan medis dan pembangkit tenaga listrik sampai perangkat hiburan, seperti video game. Banyak produk perangkat lunak berisi jutaan baris kode yang diharapkan dapat melakukan pekerjaan dengan baik dalam menghadapi perubahan kondisi. Semua perangkat lunak juga membutuhkan keandalan yang tinggi dan harus dihasilkan secara ekonomis. Teknik rekayasa perangkat lunak akan meningkatkan fungsionalitas dan efisiensi aplikasi dan juga kemudahan dan efisiensi dari pengembang perangkat lunak. Pelopor rekayasa perangkat lunak adalah Barry Boehm, Fred Brooks, CAR Hoare, dan David Parnas. [14]

2.5.2 Microservice

Microservice adalah pendekatan untuk mendistribusikan sistem yang membantu mengoptimalkan penggunaan service dengan siklus hidup mereka sendiri namun secara bersamaan saling berkolaborasi. Microservice terutama dimodelkan pada domain bisnis untuk memecahkan masalah dari arsitektural tradisional. Microservice mengintegrasikan teknologi dan teknik yang muncul selama dekade terakhir yang diharapkan dapat membantu menghindari kebuntuan dari sebuah sistem [9].

BAB III

ANALISIS DAN PERANCANGAN SISTEM

Bab ini akan membahas tentang proses bisnis dan arsitektur dari rumah sakit Apertura. Lalu akan membahas analisa terkait proses bisnis dan arsitektur yang baru untuk diterapkan pada aplikasi, termasuk segala komponen penyusun sistem yang baru mulai dari database, *tools*, IDE (*Integrated Development Environment*), *framework* yang digunakan dalam membangun aplikasi. Selanjutnya akan dibahas mengenai perancangan *service* dengan menggunakan konsep REST dan pembuatan API dari *service* tersebut.

3.1 Arsitektur Monolitik pada Software Apertura

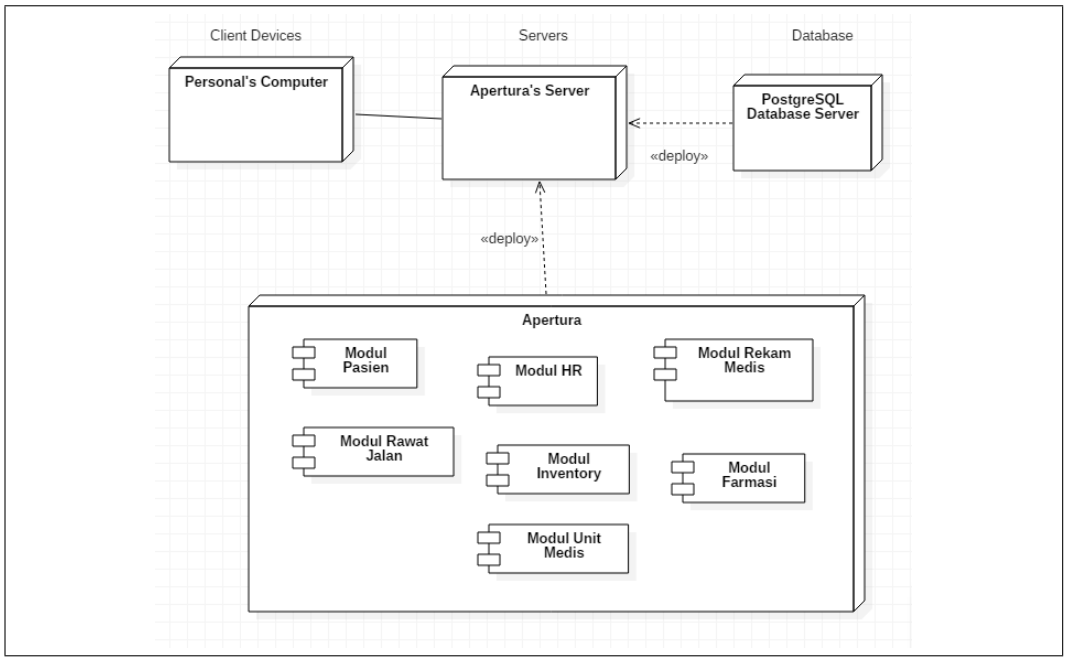
Menurut Sam Newman, terdapat 2 parameter yang membedakan arsitektur monolitik dengan microservice, pertama adalah tingkat *coupling* dan yang kedua adalah tingkat *cohesion* dari aplikasi tersebut. Pernyataan ini kemudian didukung oleh Chris Richardson yang menjelaskan bahwa kedua parameter ini dapat ditinjau dari berbagai sisi yang membentuk aplikasi tersebut, antara lain: data menejemen, cara berkomunikasi, dan metode *deployment* sebuah aplikasi.

1. **Data manajemen.** Aplikasi Apertura menerapkan model tersentralisasi yang sangat besar. Kurang lebih terdapat 120 tabel yang terdapat dalam 1 buah database tunggal. Namun semakin banyak model shared database yang digunakan, maka semakin tinggi pula derajat *coupling* aplikasi tersebut, dan tingginya derajat *coupling* merupakan salah satu ciri dari monolitik.

2. **Cara berkomunikasi.** Modul dalam database Apertura dapat mengakses langsung tabel milik modul lain dengan melakukan *querry* terhadap tabel tersebut. Cara berkomunikasi seperti ini seperti ini menjadi ciri dari arsitektur monolitik.

3. **Bentuk deployment.** Tabel-tabel dan database Apertura disimpan dalam 1 buah database server yang sama. Penempatan server terpusat ini dianggap kurang menunjang konsep *high availability* apabila terjadi masalah pada server. Aplikasi menjadi sangat tergantung dengan server tunggal tersebut dan menjadi ciri derajat *coupling* yang tinggi.

Berdasarkan analisis dari ketiga faktor diatas, maka dapat disimpulkan bahwa arsitektur dari aplikasi Apertura adalah monolitik.
Arsitektur monolitik Apertura dapat digambarkan dengan deployment diagram dibawah:



Gambar 3.1 Pemodelan Software Apertura dengan deployment diagram.

3.2 Tinjauan Umum Proses Bisnis Pelayanan Rawat Jalan

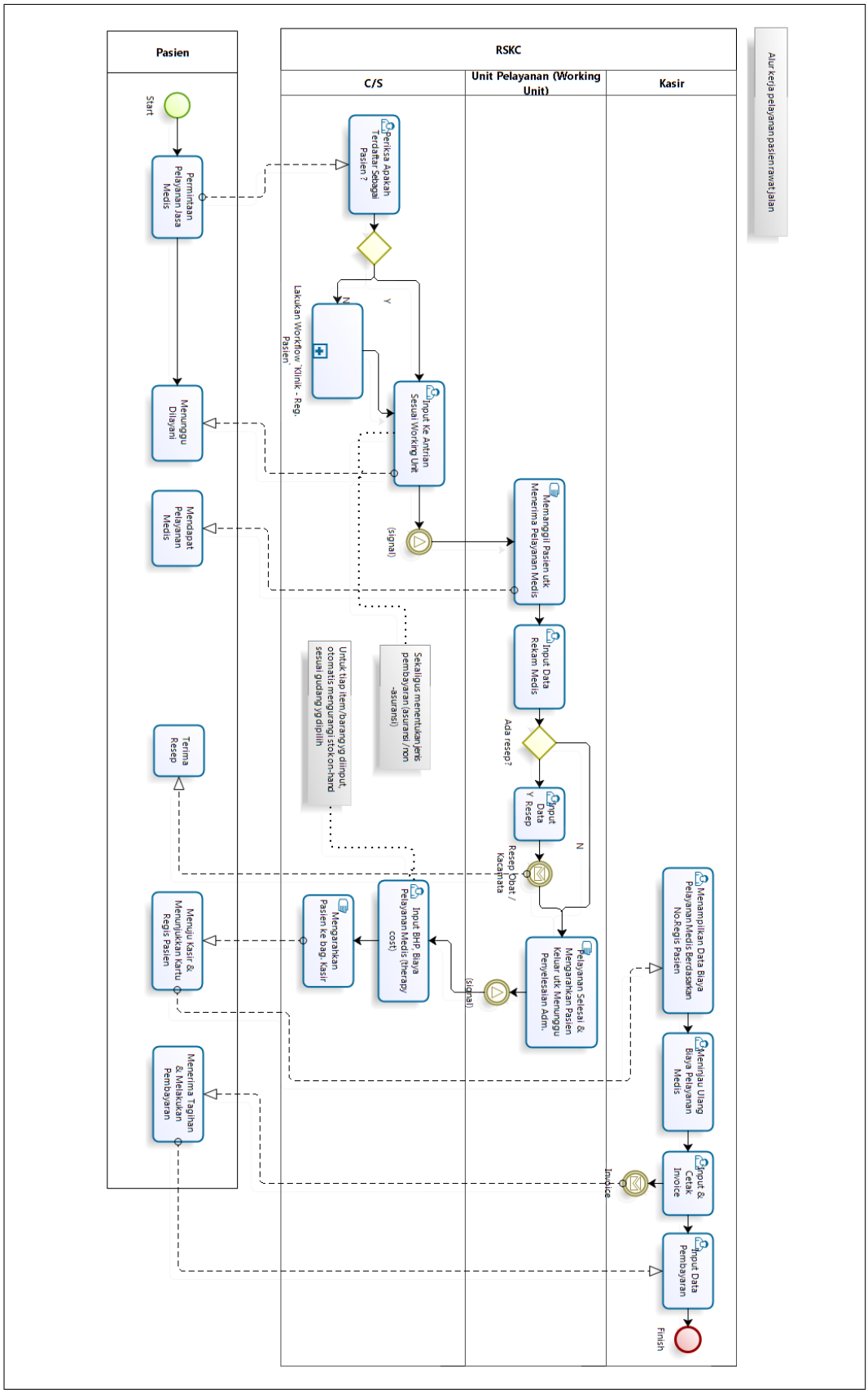
Proses bisnis dalam rumah sakit Apertura melibatkan beberapa modul yang saling berinteraksi, modul-modul tersebut terdiri dari bagian yang lebih kecil lagi. Analisis berdasarkan proses bisnis berdasarkan kegunaannya akan lebih mudah untuk menentukan service yang akan terbentuk nantinya.
Modul-modul dari aplikasi Apertura antara lain: modul pasien, modul HR (human resources) yang meliputi data dokter, bidan, dan perawat, modul rekam medis, modul farmasi, modul inventori, modul akunting dan finansial, modul invoicing dan pembayaran, modul rawat jalan, modul rawat inap, modul pemeriksaan penunjang, dan modul integrasi SEP (Surat Eligibilitas Pasien) BPJS.

Dari semua modul tersebut, penulis akan mengambil contoh kasus rawat jalan. Rawat jalan adalah tindakan perawatan pasien yang tidak menginap. Pasien yang datang akan mendaftar ke unit rawat jalan dan dicek apakah pasien tersebut terdaftar di BPJS, kemudian berdasarkan masalah pasien, pasien akan dirujuk ke unit medis yang ada di rumah sakit. Unit medis dari rumah sakit terdiri dari unit medis spesialis anak, spesialis jantung, dan spesialis penyakit dalam. Tiap unit medis memiliki satu atau lebih dokter spesialis dari bidang unit medis tersebut.

Pasien kemudian akan ditangani oleh dokter yang bertugas di unit medis tersebut. Proses penanganan pasien dimulai dari konsultasi keluhan, pemeriksaan penunjang, dan diagnosis penyakit.

Hasil dari pemeriksaan tersebut akan disimpan dalam rekam medis pasien di rumah sakit. Modul rawat jalan akan mengeluarkan resep obat yang dapat pasien ambil di farmasi. Modul rawat jalan juga akan mengeluarkan detail faktur yang dibutuhkan untuk proses pembayaran.

Maka dari itu, modul rawat jalan akan melibatkan modul pasien, HR, unit medis, farmasi, rekam medis, pemeriksa penunjang, pembayaran dan penagihan, integrasi BPJS, dan modul rawat jalan itu sendiri.
Berikut adalah penggambaran proses bisnis dari pelayanan rawat jalan:



Gambar 3.2 Proses bisnis rawat jalan.

3.3 Pembahasan Modul dan Kelas Penyusun

Pada bagian ini akan dijelaskan fungsi dan kelas-kelas dari setiap modul dalam kaitannya dengan proses bisnis di rumah sakit.

1. **Modul Pasien.** Modul pasien berfungsi untuk menyimpan, memperbaharui, pencarian, dan menampilkan data pasien. Data pasien juga meliputi detail lengkap dari data keluarga atau kerabat yang menjadi penanggung jawab pasien. Kelas dari modul pasien adalah pasien itu sendiri.

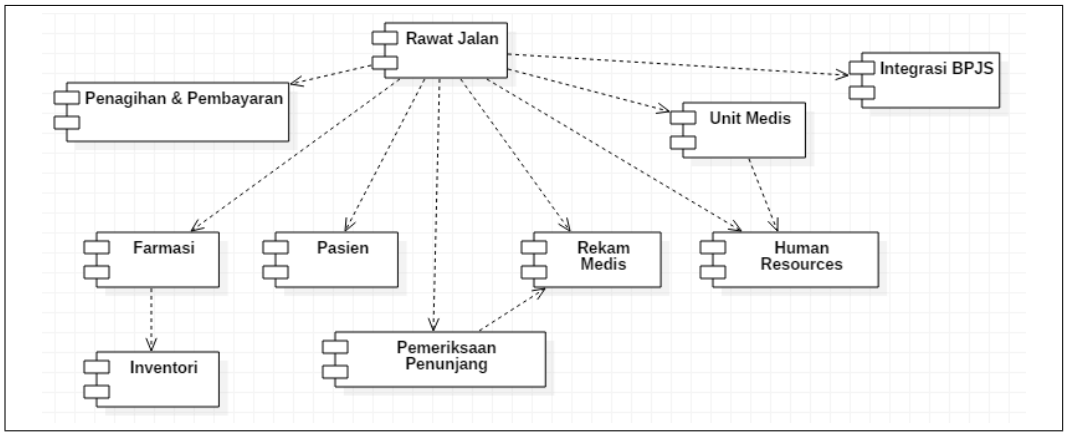
2. **Modul *Human Resource*.** *Human Resource* adalah modul yang mengelola semua pengguna dan pegawai dalam rumah sakit, modul ini dapat disebut juga sebagai modul karyawan. Beberapa contoh yang termasuk dalam modul ini adalah dokter, bidan, dan perawat. Modul ini menjadi modul dasar yang dibutuhkan modul lain, karena terkait pencatatan pengguna yang melakukan input data. Kelas-kelas dari modul ini meliputi dokter, bidan, dan perawat. Namun dalam contoh kasus rawat jalan yang akan diangkat, kelas yang diambil hanya dokter saja.

3. **Modul Unit Medis.** Unit medis dari rumah sakit terdiri dari unit medis spesialis anak, spesialis jantung, dan spesialis penyakit dalam. Tiap unit medis memiliki satu atau lebih dokter spesialis dari bidang unit medis tersebut. Pasien akan ditangani oleh dokter yang bertugas di unit medis tersebut.

4. **Modul Rekam Medis.** Rekam medis bertugas mencatat semua riwayat medis milik pasien yang meliputi hasil diagnosa, resep dan obat yang pernah diberikan, alergi, penyakit kronis, riwayat tindakan atau perawatan medis, dan hasil pemeriksaan penunjang. Pemeriksaan penunjang dapat berupa banyak aksi, tergantung apa yang dibutuhkan dokter. Pemeriksaan penunjang antara lain seperti radiologi, tes darah, tensi tekanan, dan lain lain. Fungsi dari pemeriksaan penunjang adalah menunjang ditegakannya diagnosis.

5. **Modul Rawat Jalan.** Rawat jalan adalah pelayanan medis kepada pasien untuk tujuan perawatan tanpa mengharuskan pasien tersebut untuk menginap. Rawat jalan menjadi perantara interaksi dari pasien dengan unit medis, rawat jalan menyimpan data perawatan yang kemudian akan disimpan dalam rekam medis. Hasil dari rawat jalan akan kemudian dibutuhkan oleh bagian pembayaran dan juga farmasi.

Berikut adalah penggambaran relasi dari modul rawat jalan menggunakan komponen diagram:



Gambar 3.3 Komponen diagram rawat jalan.

3.4 Identifikasi Teknologi yang Digunakan pada Aplikasi Apertura

Saat ini teknologi aplikasi yang digunakan Rumah Sakit Apertura menggunakan arsitektur monolitik dengan database yang tersentralisasi pada 1 buah server. Database Apertura sendiri menggunakan PostgreSQL 9. PostgreSQL merupakan salah satu object-relational database management system (ORDMS) yang tersedia secara *open source*. Database Apertura terdiri dari kurang lebih 120 tabel yang saling berelasi. Cara pertukaran data yang digunakan oleh Apertura saat ini adalah query *database* langsung dan database dibuat *shared* sehingga semua modul bisa mengakses *database* secara bebas. Aplikasi Apertura digunakan oleh beberapa kelompok pengguna, antara lain bagian kasir rumah sakit, registrasi rawat jalan, registrasi rawat inap, kasir apotik, kasir rawat jalan, dokter, staf administrasi umum, staf keuangan, staf rekam medis, staf administrasi BPJS. Dalam setiap harinya terjadi rata-rata 400 pendaftaran rawat jalan yang ditangani oleh kurang lebih 30 dokter yang bertugas. Platform yang diimplementasi oleh Apertura adalah bentuk aplikasi desktop. *User interface* yang ditampilkan dalam aplikasi tidak terpisah secara moduler, melainkan satu kesatuan aplikasi besar yang kemudian dipisahkan berdasarkan kebutuhan user yang menggunakan aplikasi.

3.4.1 Peluang dan Antisipasi Pengembangan Sistem

Berdasarkan analisis yang dilakukan, peluang pengembangan sistem baru yang diusulkan dalam penelitian kali ini melihat 3 faktor utama, yaitu faktor data, platform, dan kelincahan proses kerja. Adapun peluang pengembangan tersebut dijelaskan pada point di bawah :

1. Desain arsitektur yang baru dibuat lebih moduler dan terpisah, dengan menjadikan setiap layanan menjadi service mandiri dapat mengurangi tingkat *coupling* yang terlalu tinggi yang dapat menyebabkan kendala dalam proses

pengembangan sistem. Pada desain microservice, perubahan yang dilakukan dalam sebuah modul tidak perlu diketahui oleh modul lain, menjadikan proses *deployment* lebih lincah dan cepat dari sebelumnya karena tiap tim pengembang sistem dapat bekerja secara mandiri. Selain itu dengan tidak ada data yang disimpan terpusat pada sebuah server, desain microservice juga meningkatkan tingkat keamanan dan *availabilitas* dari keseluruhan sistem.

2. Desain yang baru disiapkan untuk kebutuhan *cross-platform*, menjadikan sistem lebih lincah dan dapat digunakan di lebih banyak perangkat seperti komputer pribadi dan juga telepon genggam. Dengan siapnya fitur *cross-platform* ini, dapat menjadi peluang agar interaksi dari *user* dapat ditingkatkan, misalnya untuk kebutuhan pendaftaran antrian rawat jalan menggunakan aplikasi di telepon genggam.

3. Sistem lebih kuat untuk menangani *multiple user* dikarenakan arsitektur microservice memiliki server masing-masing yang bekerja khusus untuk layanannya sendiri, tidak bercampur dengan modul lain.

3.5 Perancangan Arsitektur Microservice

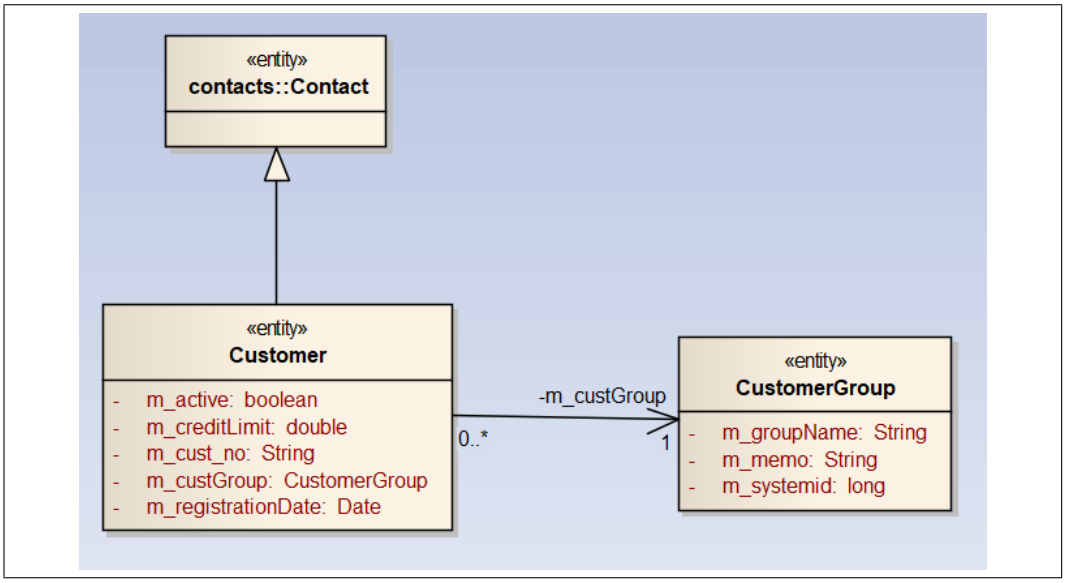
Dengan menggunakan pattern dekomposisi berdasarkan kemampuan bisnisnya, service yang akan terbentuk terdapat 5 buah, yaitu *service Customer*, *Human Resources*, *Medical Unit*, *Medical Records* dan *service Outpatient* (rawat jalan). Modul-modul *service* tersebut didapat dari hasil analisa proses bisnis dalam pelayanan rawat jalan (gambar 3.2). Apabila sebelumnya semua entitas berada dalam satu buah *server* yang sama, kali ini masing-masing service tersebut akan berdiri mandiri di *server* yang berbeda. Perancangan berikut akan menjelaskan migrasi ke arsitektur baru dan menjawab rumusan masalah yang pertama. Berikut adalah analisa perancangan service yang baru.

3.5.1 Perancangan Service *Customer*

Customer merupakan turunan dari *class* Contact yang berisi biodata pasien rumah sakit. Customer bisa dibedakan menjadi 2, yaitu *customer* yang memiliki asuransi dan yang tidak memiliki asuransi. Namun selain Customer, *class* Contact juga digunakan oleh modul *Human Resources* dikarenakan atribut *contact* dapat digunakan untuk *customer* maupun juga *employee*. Ketergantungan dari desain seperti ini dianggap terlalu tinggi dan menyebabkan data *customer* dan *employee* menjadi tercecer dikarenakan ditempatkan dalam satu tabel yang sama. Desain baru yang diimplementasikan untuk penelitian kali ini adalah memisahkan fungsi *contact* menjadi lebih spesial untuk *customer* dan juga *employee*. Kelebihan dari

desan ini adalah tingkat ketergantungan yang lebih rendah dan tingkat *availabilitas* yang lebih tinggi. Maka dalam modul *customer* akan berisi entitas *customer*, *contact*, dan *class* relasi dari kedua entitas tersebut.

Berikut adalah rancangan untuk *webservice customer* digambarkan menggunakan *class diagram*



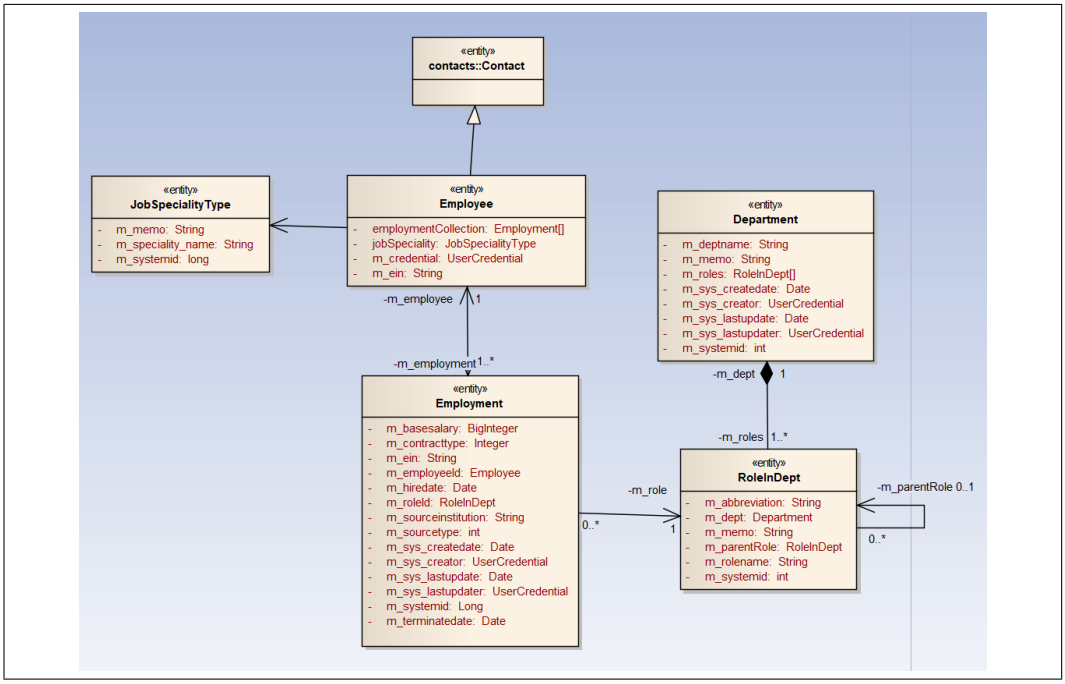
Gambar 3.4 Rancangan service *customer*

.

3.5.2 Perancangan Service *Human Resource*

Modul *Human Resource* berisi data *employee* yang bekerja di rumah sakit. Modul ini pula menjelaskan departemen, dan spesialitas pekerjaan *employee* tersebut di rumah sakit. Sama halnya dengan *customer*, *employee* juga merupakan turunan dari *class contact* yang berdiri mandiri dan terpisah fungsinya dengan yang digunakan untuk *customer*, kelebihan dari desan ini adalah tingkat ketergantungan yang lebih rendah dan tingkat *availabilitas* yang lebih tinggi.

Berikut adalah rancangan untuk *webservice human resource* digambarkan menggunakan *class diagram*



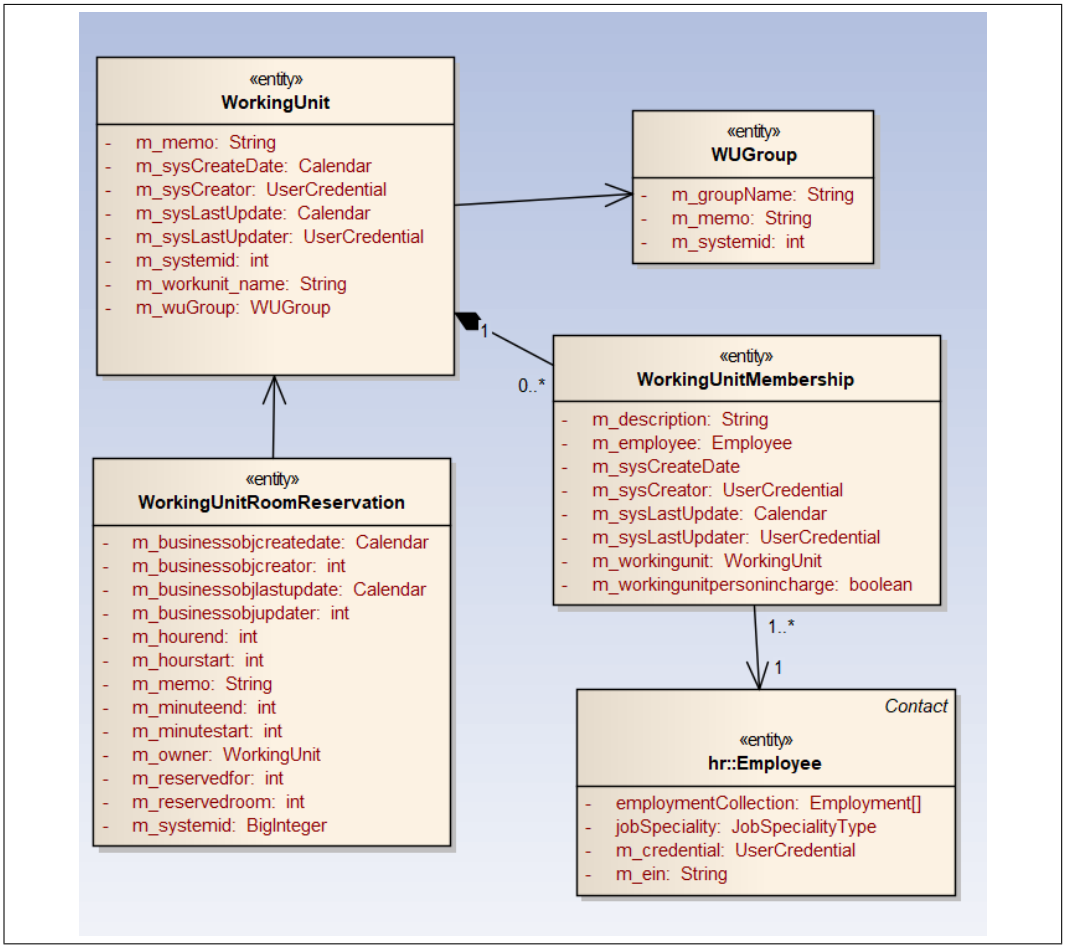
Gambar 3.5 Rancangan service *human resource*

.

3.5.3 Perancangan Service Unit Medis

Modul Unit Medis merupakan unit representasi tempat *employee* bekerja, unit medis sendiri adalah turunan dari *class working unit* dan memiliki lebih dari satu *working unit membership*, sedangkan *working unit membership* merupakan kumpulan dari *employee* yang bekerja pada *working unit tersebut*. Dalam pemodelan microservice yang diterapkan dalam penelitian ini, modul *human resource* tidak dijadikan satu dalam modul unit medis seperti yang diterapkan di model monolitik, maka pada pemodelan microserviec yang baru dibutuhkan objek representasi dari objek *employee* atau disebut juga sebagai *value object* (VO), kegunan dari VO ini adalah sebagai kerangka objek hasil pemanggilan dari *webservice* dan bukan sebagai objek penghubung dengan *database*.

Berikut adalah rancangan untuk *webservice* unit medis digambarkan menggunakan *class diagram*



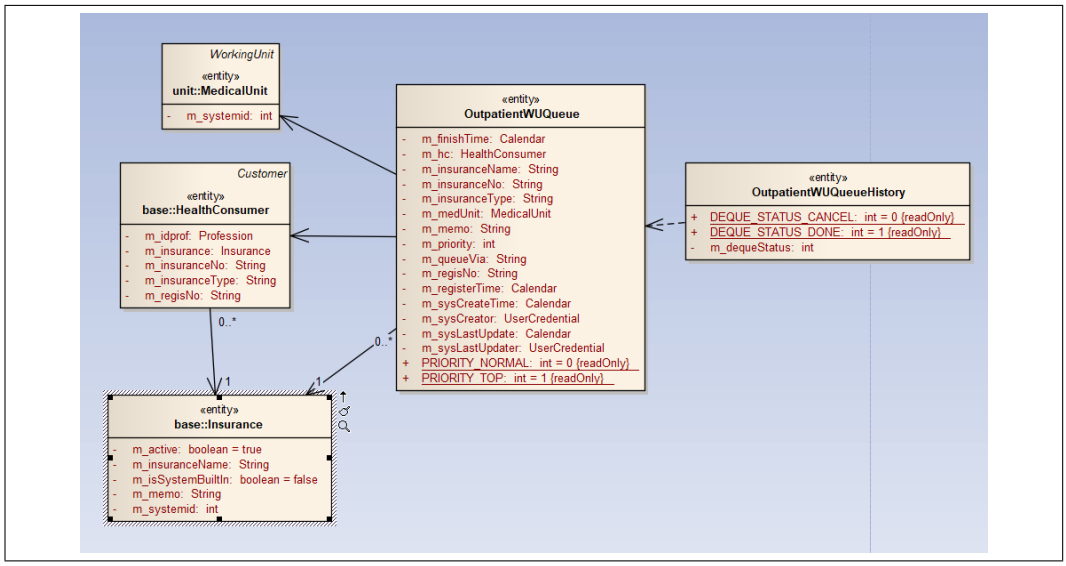
Gambar 3.6 Rancangan service *medical unit*

.

3.5.4 Perancangan Service Rawat Jalan

Pada modul rawat jalan dibutuhkan 2 buah *value object* yang membantu merepresentasikan objek *customer* dan *employee*. Hal ini diperlukan karena model microservice yang baru tidak menggabungkan kedua objek tersebut dalam modul rawat jalan secara langsung.

Berikut adalah rancangan untuk *webservice* rawat jalan digambarkan menggunakan *class diagram*



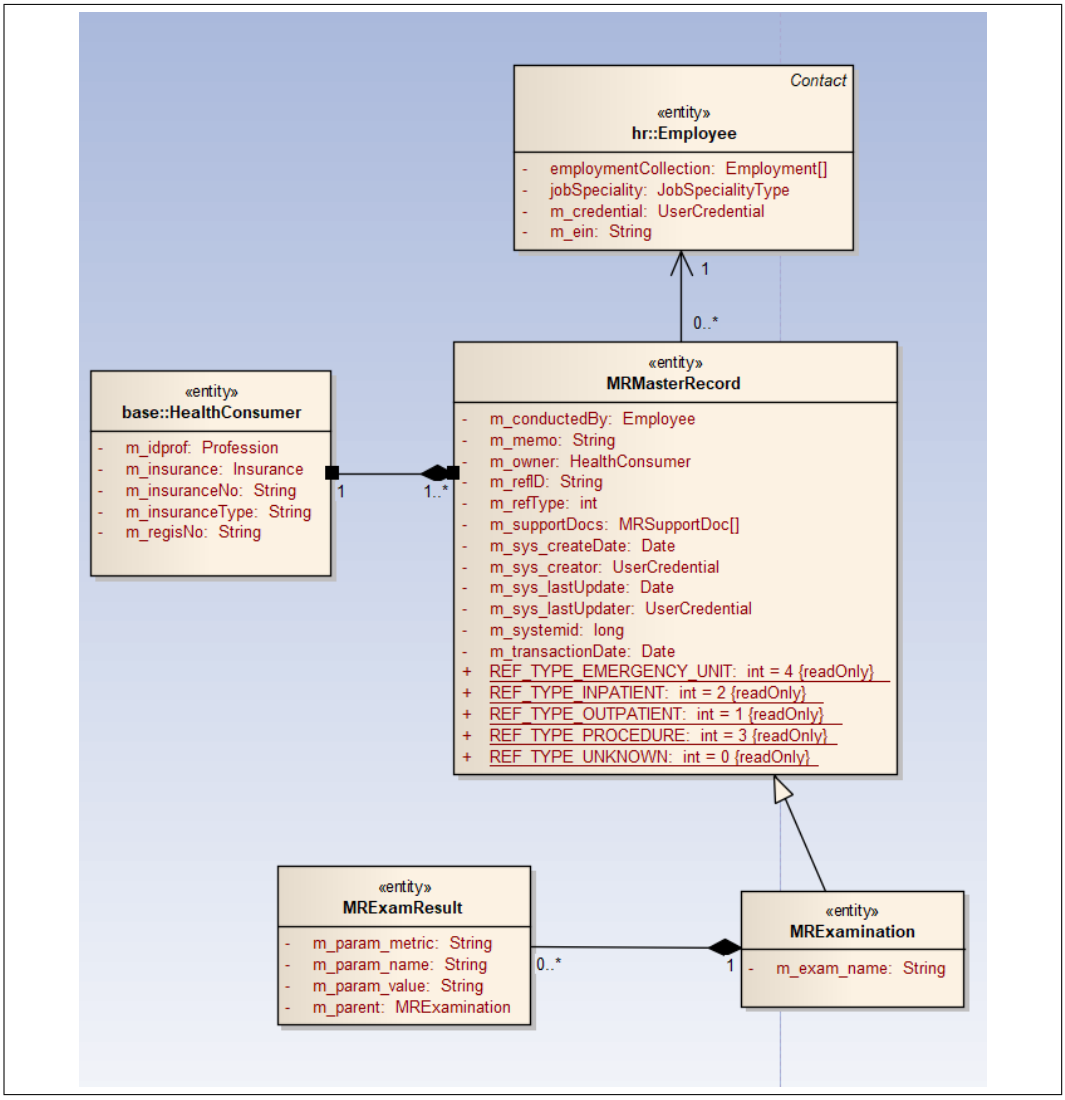
Gambar 3.7 Rancangan service rawat jalan

.

3.5.5 Perancangan Service *Medical Records*

Modul *medical records* menyimpan reman tindakan medis yang dilakukan untuk *customer* dan oleh siapa tindakan medis tersebut dilakukan. Maka dari itu modul rekam medis juga membutuhkan dua *value object* yang merepresentasikan objek *customer* dan *employee*.

Berikut adalah rancangan untuk *webservice medical records* digambarkan menggunakan *class diagram*



Gambar 3.8 Rancangan service unit medis

.

3.5.6 Pemilihan Model Penyimpanan Data

Setelah kelas-kelas service terbentuk, hal berikutnya yang harus diperhatikan adalah pemodelan manajemen data. Apabila ditinjau dari service yang dihasilkan, maka akan lebih baik menerapkan pattern database per *service*, dimana satu service memiliki sebuah set database yang hanya bisa diakses oleh service tersebut dan tidak bisa saling mengakses database service yang lain.

3.5.7 Pemilihan Komunikasi Antar Service

Setelah membuat service dan database yang digunakan, hal berikutnya adalah membuat API dari service tersebut agar dapat diakses menggunakan *web service*. Berdasarkan kebutuhan arsitektur microservice sendiri, model *web service* yang akan digunakan adalah RESTful (*Representational State Transfer*) dengan

format data yang dikirim berupa JSON (*JavaScript Object Notation*). Server sementara yang akan digunakan selama tahap perancangan adalah mesin pribadi milik peneliti. Setelah perancangan selesai dan siap, server baru akan dipindahkan ke server milik rumah sakit. Untuk tes pemanggilan API sementara dapat dilakukan dalam satu jaringan yang sama.

3.5.8 Rancangan Deployment

Model *deployment* yang akan digunakan adalah *multiple service per host*, dikarenakan lebih cocok dalam kasus penelitian yang hanya mengambill proses rawat jalan. Semua set service yang terbentuk akan di *upload* dalam beberapa server agar dapat membuktikan tercapainya *high availability*. Apertura sendiri memiliki 2 buah server aktif yang dapat digunakan ketika proses *deployment*, serta server lainnya dapat menggunakan mesin pribadi untuk melakukan pemanggilan service dari server lain.

3.6 Strategi Pengujian

Strategi pengujian yang dirancang untuk kasus penulis adalah dengan melakukan uji perbandingan performa dengan arsitektur monolitik yang digunakan pada aplikasi saat ini. Tujuan dari test performa ini adalah menunjukan bahwa arsitektur microservice yang baru akan lebih unggul dalam 5 point diatas apabila dibandingkan dengan monolitik.

3.6.1 Uji Perbandingan Performa

Strategi pengujian yang akan dilakukan adalah dengan membuat *test plan* yang mengacu pada masalah *deployability, reliability, availability, scalability,* dan *modifiability.*. *Test plan* akan dibuat sekitar 15-20 point agar kelebihan dan kekurangan dari tiap model arsitektur dapat terlihat. *Test result* kemudian akan disimpan dalam dokumen berupa *matrix traceability*. Tujuan dari test ini adalah membandingkan apakah model arsitektur microservice memberikan performa yang lebih baik apabila dibandingkan dengan arsitektur monolitik.

3.7 Konsumer Webservice

Konsumer adalah pihak yang melakukan pemanggilan terhadap webservice API. Konsumer disini dibedakan berdasarkan webservice yang dipanggil. Apabila ditinjau dari modul yang ada pada aplikasi Apertura, maka konsumer dapat didefinisikan menjadi :

1. Konsumer dari webservice customer. Konsumer dari webservice ini bila ditinjau dari proses bisnis di gambar 3.2 adalah staff pelayanan rawat jalan yang mendaftarkan pasien ke dalam modul rawat jalan dan staff unit medis yang akan melihat riwayat rekam medis pasien.

2. Konsumer dari webservice medical unit. Konsumer dari webservice ini adalah staff medical unit dan juga employee itu sendiri.

3. Konsumer dari webservice human resource. Konsumer dari webservice ini apabila ditinjau dari proses bisnsi di gambar 3.2 adalah staff rawat jalan, staff unit medis yang berguna untuk mendaftarkan employee pada medical unit, staff rekam medis untuk menyimpan data dokter di rekam medis pasien, dan juga employee sendiri

4. Konsumer dari webservice medical records. Konsumer dari webservice ini adalah staff unit medis yang dapat melihat rekam medis pasien dan juga staff rawat jalan.

5. Konsumer dari webservice rawat jalan yaitu adalah staff rawat jalan sendiri yang bertugas mulai dari pendaftaran sampai proses rawat jalan itu selesai.

BAB IV

IMPLEMENTASI DAN PERBANDINGAN

Pada bab ini akan menjelaskan tentang pengimplementasian dan pengujian terhadap analisis sentimen yang telah dibangun berdasarkan bab-bab sebelumnya.

4.1 Lingkungan Aplikasi

Dalam aplikasi terbagi menjadi dua bagian, yaitu lingkungan implementasi perangkat keras dan perangkat lunak. Di dalam penelitian ini, perangkat keras yang digunakan adalah:

1. Asus A45A
2. Processor Intel i3-2370M CPU 2.40GHz
3. RAM 6 GB.

Spesifikasi perangkat lunak yang digunakan untuk pengembangan sistem adalah:

1. Sistem Operasi : Windows 10 Enterprise 1709 64-bit.
2. Tool Pengembangan : Eclipse Java EE IDE for Web Developers.
3. Versi : Neon.3 Release (4.6.3).

4.2 Daftar *Project*, *Class* dan *Method*

Pada bagian ini akan dijelaskan mengenai *Project*, *class* dan *method* yang digunakan dalam pengembangan sistem analisis.

4.2.1 *Project* Customer

Project Customer berisi susunan *class* pembentuk *webservice customer*. Dalam project ini terdapat 4 buah *package* yang berfungsi untuk memisahkan *class* sesuai dengan fungsinya masing-masing. Ke empat *package* ini yaitu *package* controller yang berisi method-method yang dijalankan pada *web service*, *package* model yang berisi entitas dan tabel, *package* repository yang menghubungkan entitas di *package* model dengan database, dan *package* service yang menginisialisasi *webservice* yang akan terbentuk. Pada bagian ini akan dijelaskan

mengenai *class* dan *method* yang digunakan pada pengembangan *webservice customer*:

4.2.1.1 *Package* Controller

Package controller berisi method API yang dapat digunakan untuk berkomunikasi dengan *service* customer. *Class* yang terdapat dalam *package* ini adalah *class* CustomerController dan CustomergroupController yang berisi 5 buah fungsi API. Di bawah ini merupakan daftar *class* untuk *package* controller pada *webservice* customer.

Tabel 4.1 Daftar *Class* pada *Package* Controller

Package	**Class**	**Jenis Class**
Controller	CustomerController	*Class*
	CustomergroupController	*Class*

Tabel 4.2 Daftar *Method* pada *Class* CustomerController

No	***Method***	***Input***		***Output***	**Keterangan**
		Tipe	**Variabel**		
1	addCustomer	void	Customer	Customer, HttpStatus	*Method* ini digunakan untuk membuat *object* customer yang baru melalui *webservice*
2	updateCustomer	void	Customer	HttpStatus	*Method* ini digunakan untuk mengubah attribut dari *object* customer yang baru melalui *webservice*
3	getCustomer	void	id	Customer, HttpStatus	*Method* ini digunakan untuk mengambil satu *object* customer yang baru melalui *webservice*

4	getAllCustomer	void	-	<List<Customer>>, HttpStatus	*Method* ini digunakan untuk mengambil semua *list object* customer yang baru melalui *webservice*
5	deleteCustomer	void	id	HttpStatus	*Method* ini digunakan untuk menghapus satu *object* customer yang baru melalui *webservice*

Tabel 4.3 Daftar *Method* pada *Class* CustomergroupController

No	***Method***	***Input***		***Output***	**Keterangan**
		Tipe	**Variabel**		
1	addCustomerGroup	Customer	customer	customer, HttpStatus	*Method* ini digunakan untuk membuat *object* CustomerGroup yang baru melalui *webservice*
2	updateCustomerGroup	Customer	customer	HttpStatus	*Method* ini digunakan untuk mengubah attribut dari *object* CustomerGroup yang baru melalui *webservice*
3	getCustomerGroup	Customer	customer	customer, HttpStatus	*Method* ini digunakan untuk mengambil satu *object* CustomerGroup yang baru melalui *webservice*
4	getAllCustomerGroup	Customer	customer	customer, HttpStatus	*Method* ini digunakan untuk mengambil semua *list object* CustomerGroup yang baru melalui *webservice*

5	deleteCustomerGroup	Customer	customer	customer, HttpStatus	*Method* ini digunakan untuk menghapus satu *object* CustomerGroup yang baru melalui *webservice*

4.2.1.2 *Package* Model Customer

Package customer model berisi entitas-entitas penyusun dari *service* customer. Berikut ini merupakan daftar *class* untuk *package* model.

Tabel 4.4 Daftar *Class* pada *Package* model

Package	**Class**	**Jenis Class**
Model	Regency	*Class*
	AddressType	*Class*
	Contact	*Class*
	ContactAddress	*Class*
	ContactEducation	*Class*
	Country	*Class*
	Customer	*Class*
	Customergroup	*Class*
	HealthConsumer	*Class*
	Insurance	*Class*
	InsuranceBridgeConf	*Class*
	Profession	*Class*
	Province	*Class*

Tabel 4.5 Daftar attribut pada *Class* Customer

Variabel:		**Variabel:**	
boolean	cust_no	Date	registrationdate
boolean	active	int	idPriceLevel
double	creditlimit	double	disc
Customergroup	customergroup		

Tabel 4.6 Daftar attribut pada *Class* Customergroup

Variabel:	
long	systemid
String	groupname
String	memo

Tabel 4.7 Daftar attribut pada *Class* Contact

Variabel:		**Variabel:**	
long	systemid	String	initial
String	lastname	int	amountofchildren
int	maritalstatus	List<ContactAddress>	arrAddress
String	middlename	Collection<ContactEducation>	arrEdu
Date	birthday	String	notes
String	birthplace	String	officeext
String	bloodtype	String	passportid
double	bodyheight	byte[]	photo
double	bodyweight	String	prefixtitle
String	citizenid	String	sms
String	citizentype	String	suffixtitle
Country	citizenship	Date	sys_createdate
String	email	String	sys_creator
String	firstname	Date	sys_lastupdate
int	gender	String	sys_lastupdater
String	homephone	String	taxid
String	messaging	String	web

Tabel 4.8 Daftar attribut pada *Class* Contact Address

Variabel:		**Variabel:**	
Long	systemid	String	postcode
AddressType	addresstype	Regency	regency
String	street	boolean	asbillingaddress
String	fax	Date	sys_createdate
Contact	owner	String	sys_creator
String	phone	Date	sys_lastupdate
String	sys_lastupdater		

Tabel 4.9 Daftar attribut pada *Class* Address Type

Variabel:	
Integer	systemid
String	memo
String	typename

Tabel 4.10 Daftar attribut pada *Class* Contact Education

Variabel:		**Variabel:**	
Date	finishrolling	Contact	owner
String	gpa	String	sponsors
int	grade	Date	startrolling
String	institutionaladdress	Date	sys_createdate
String	institutionname	int	sys_creator
String	major	Date	sys_lastupdate
String	notes	int	sys_lastupdater

Tabel 4.11 Daftar attribut pada *Class* Country

Variabel:	
Long	systemid
String	name

Tabel 4.12 Daftar attribut pada *Class* HealthConsumer

Variabel:	
String	regis_no
Profession	id_prof
Insurance	insurance
String	insurance_type
String	insurance_no
String	insurance_prg

Tabel 4.13 Daftar attribut pada *Class* Insurance

Variabel:	
List<InsuranceBridgeConf>	insuranceBrigdeConfList
int	systemid
String	insurance
String	memo
boolean	active
boolean	sysbuiltin

Tabel 4.14 Daftar attribut pada *Class* InsuranceBridgeConf

Variabel:	
Long	systemid
String	field
String	def_val
String	memo

Tabel 4.15 Daftar attribut pada *Class* Profession

Variabel:	
Integer	systemid
String	profname
String	memo
Calendar	sys_createdate
Calendar	last_createdate

Tabel 4.16 Daftar attribut pada *Class* Province

Variabel:	
Long	systemid
Country	countryCode
String	name
Set<Regency>	regencySet

Tabel 4.17 Daftar attribut pada *Class* Regency

Variabel:	
Long	systemid
Province	provId
String	name

4.2.1.3 *Package* Repository

Package repository berisi *class-class* yang menjadi representasi model dari tabel-tabel yang berada pada basis data. Berikut ini merupakan *class-class* yang terdapat pada *package* repository.

Tabel 4.18 Daftar *Class* pada *Package* repository

Package	**Class**	**Jenis Class**
repository	CustomerRepository	*Interface*
	CustomergroupRepository	*Interface*

4.2.1.4 *Package* Service

Package service berisi *class-class* yang menginisialisasi *web service* customer yang akan terbentuk dan digunakan pada *package* controller. Berikut ini merupakan *class-class* yang terdapat pada *package* service.

Tabel 4.19 Daftar *Class* pada *Package* serviec

Package	Class	Jenis Class
service	CRUDService	*Interface*
	CustomergroupService	*Interface*
	CustomerService	*Interface*
	DefaultCustomergroupService	*Class*
	DefaultCustomerService	*Class*

4.2.2 *Project Human Resource*

Project Human Resource berisi susunan *class* pembentuk *webservice Human Resource*. Dalam project ini terdapat 4 buah *package* yang berfungsi untuk memisahkan *class* sesuai dengan fungsinya masing-masing. Ke empat *package* ini yaitu *package* controller yang berisi method-method yang dijalankan pada *web service*, *package* model yang berisi entitas dan tabel, *package* repository yang menghubungkan entitas di *package* model dengan database, dan *package* service yang menginisialisasi *webserviec* yang akan terbentuk. Pada bagian ini akan dijelaskan mengenai *class* dan *method* yang digunakan pada pengembangan *webservice customer*:

4.2.2.1 *Package* Controller

Package controller berisi method API yang dapat digunakan untuk berkomunikasi dengan *service* Human Resource. *Class* yang terdapat dalam *package* ini adalah *class* EmployeeController dan JobSpecialityController yang berisi 5 buah fungsi API. Di bawah ini merupakan daftar *class* untuk *package* controller pada *webservice Human Resource*.

Tabel 4.20 Daftar *Class* pada *Package* Controller

Package	Class	Jenis Class
Controller	EmployeeController	*Class*
	JobspecialityController	*Class*

Tabel 4.21 Daftar *Method* pada *Class* EmployeeController

No	*Method*	*Input*		*Output*	Keterangan
		Tipe	**Variabel**		
1	addEmployee	void	Employee	Employee, HttpStatus	*Method* ini digunakan untuk membuat *object* employee yang baru melalui *webservice*
2	updateEmployee	void	Employee	HttpStatus	*Method* ini digunakan untuk mengubah attribut dari *object* employee yang baru melalui *webservice*
3	getEmployee	void	id	Employee, HttpStatus	*Method* ini digunakan untuk mengambil satu *object* employee yang baru melalui *webservice*
4	getAllEmployees	void	-	<List<Employee>>, HttpStatus	*Method* ini digunakan untuk mengambil semua *list object* employee yang baru melalui *webservice*

5	deleteEmployee	void	id	HttpStatus	*Method* ini digunakan untuk menghapus satu *object* employee yang baru melalui *webservice*

Tabel 4.22 Daftar *Method* pada *Class* JobspecialityController

No	***Method***	***Input***		***Output***	**Keterangan**
		Tipe	**Variabel**		
1	addJobSpeciality	void	Jobspeciality	Jobspeciality, HttpStatus	*Method* ini digunakan untuk membuat *object* Jobspeciality yang baru melalui *webservice*
2	updateJobSpeciality	void	Jobspeciality	HttpStatus	*Method* ini digunakan untuk mengubah attribut dari *object* Jobspeciality yang baru melalui *webservice*
3	getJobSpeciality	void	id	Jobspeciality, HttpStatus	*Method* ini digunakan untuk mengambil satu *object* Jobspeciality yang baru melalui *webservice*

4	getAllJobsSpeciality	void	-	<List <Jobspeciality>>, HttpStatus	*Method* ini digunakan untuk mengambil semua *list object* Jobspeciality yang baru melalui *webservice*
5	deleteJobSpeciality	void	id	Jobspeciality, HttpStatus	*Method* ini digunakan untuk menghapus satu *object* CustomerGroup yang baru melalui *webservice*

4.2.2.2 *Package* Model Human Resource

Package Human Resource model berisi entitas-entitas penyusun dari *service* Human Resource. Berikut ini merupakan daftar *class* untuk *package* model.

Tabel 4.23 Daftar *Class* pada *Package* model

Package	**Class**	**Jenis Class**
Model	Regency	*Class*
	AddressType	*Class*
	Contact	*Class*
	ContactAddress	*Class*
	ContactEducation	*Class*
	Country	*Class*
	Department	*Class*
	Employee	*Class*
	Employment	*Class*
	Jobspeciality	*Class*
	Province	*Class*
	RoleInDept	*Class*

Tabel 4.24 Daftar attribut pada *Class* Employment

Variabel:		**Variabel:**	
Long	systemid	Employee	employee
String	ein	Date	hiredate
BigInteger	basesalary	RoleInDept	roleIndept
Integer	contracttype	String	sourceinstitution
int	sourcetype	Date	createdate
Date	lastupdate	Date	terminatedate

Tabel 4.25 Daftar attribut pada *Class* Employee

Variabel:	
String	m_ein
Jobspeciality	jobspeciality
Collection<Employment>	employmentCollection

Tabel 4.26 Daftar attribut pada *Class* Departement

Variabel:	
int	systemid
String	deptname
String	memo
Collection<RoleInDept>	roles
Date	createdate
Date	lastupdate

Tabel 4.27 Daftar attribut pada *Class* Contact

Variabel:		**Variabel:**	
long	systemid	String	initial
String	lastname	int	amountofchildren
int	maritalstatus	List<ContactAddress>	arrAddress
String	middlename	Collection<ContactEducation>	arrEdu
Date	birthday	String	notes
String	birthplace	String	officeext
String	bloodtype	String	passportid
double	bodyheight	byte[]	photo
double	bodyweight	String	prefixtitle
String	citizenid	String	sms
String	citizentype	String	suffixtitle
Country	citizenship	Date	sys_createdate
String	email	String	sys_creator
String	firstname	Date	sys_lastupdate
int	gender	String	sys_lastupdater
String	homephone	String	taxid
String	messaging	String	web

Tabel 4.28 Daftar attribut pada *Class* ContactAddress

Variabel:		**Variabel:**	
Long	systemid	String	postcode
AddressType	addresstype	Regency	regency
String	street	boolean	asbillingaddress
String	fax	Date	sys_createdate
Contact	owner	String	sys_creator
String	phone	Date	sys_lastupdate
String	sys_lastupdater		

Tabel 4.29 Daftar attribut pada *Class* AddressType

Variabel:	
Integer	systemid
String	memo
String	typename

Tabel 4.30 Daftar attribut pada *Class* ContactEducation

Variabel:		**Variabel:**	
Date	finishrolling	Contact	owner
String	gpa	String	sponsors
int	grade	Date	startrolling
String	institutionaladdress	Date	sys_createdate
String	institutionname	int	sys_creator
String	major	Date	sys_lastupdate
String	notes	int	sys_lastupdater

Tabel 4.31 Daftar attribut pada *Class* Country

Variabel:	
Long	systemid
String	name

Tabel 4.32 Daftar attribut pada *Class* Jobspeciality

Variabel:	
Long	systemid
String	specialityName
String	memo

Tabel 4.33 Daftar attribut pada *Class* RoleInDept

Variabel:	
int	systemid
Department	department
String	rolename
String	abbreviation
RoleInDept	parentRole
String	memo

Tabel 4.34 Daftar attribut pada *Class* Province

Variabel:	
Long	systemid
Country	countryCode
String	name
Set<Regency>	regencySet

Tabel 4.35 Daftar attribut pada *Class* Regency

Variabel:	
Long	systemid
Province	provId
String	name

4.2.2.3 *Package* Repository

Package repository berisi *class-class* yang menjadi representasi model dari tabel-tabel yang berada pada basis data. Berikut ini merupakan *class-class* yang terdapat pada *package* repository.

Tabel 4.36 Daftar *Class* pada *Package* repository

Package	**Class**	**Jenis Class**
repository	EmployeeRepository	*Interface*
	JobspecialityRepository	*Interface*

4.2.2.4 *Package* Service

Package service berisi *class-class* yang menginisialisasi *web service* human resource yang akan terbentuk dan digunakan pada *package* controller. Berikut ini merupakan *class-class* yang terdapat pada *package* service.

Tabel 4.37 Daftar *Class* pada *Package* service

Package	Class	Jenis Class
service	CRUDService	*Interface*
	EmployeeService	*Interface*
	JobspecialityService	*Interface*
	DefaultEmployeeService	*Class*
	DefaultJobspecialityService	*Class*

4.2.3 *Project Medical Unit*

Project Medical Unit berisi susunan *class* pembentuk *webservice Medical Unit*. Dalam project ini terdapat 4 buah *package* yang berfungsi untuk memisahkan *class* sesuai dengan fungsinya masing-masing. Ke empat *package* ini yaitu *package* controller yang berisi method-method yang dijalankan pada *web service*, *package* model yang berisi entitas dan tabel, *package* repository yang menghubungkan entitas di *package* model dengan database, dan *package* service yang menginisialisasi *webserviec* yang akan terbentuk. Pada bagian ini akan dijelaskan mengenai *class* dan *method* yang digunakan pada pengembangan *webservice Unit Medis*:

4.2.3.1 *Package* Controller

Package controller berisi method API yang dapat digunakan untuk berkomunikasi dengan *service* Human Resource. *Class* yang terdapat dalam *package* ini adalah *class* UnitmedisController yang berisi 5 buah fungsi API. Di bawah ini merupakan daftar *class* untuk *package* controller pada *webservice* Unit Medis.

Tabel 4.38 Daftar *Class* pada *Package* Controller

Package	Class	Jenis Class
Controller	UnitmedisController	*Class*

Tabel 4.39 Daftar *Method* pada *Class* UnitmedisController

No	*Method*	*Input*		*Output*	Keterangan
		Tipe	**Variabel**		
1	addMedicalUnit	void	MedicalUnit	MedicalUnit, HttpStatus	*Method* ini digunakan untuk membuat *object* unit medis yang baru melalui *webservice*
2	updateMedicalUnit	void	MedicalUnit	HttpStatus	*Method* ini digunakan untuk mengubah attribut dari *object* unit medis yang baru melalui *webservice*
3	getMedicalUnit	void	id	MedicalUnit, HttpStatus	*Method* ini digunakan untuk mengambil satu *object* unit medis yang baru melalui *webservice*
4	getAllMedicalUnit	void	-	<List <MedicalUnit>>, HttpStatus	*Method* ini digunakan untuk mengambil semua *list object* unit medis yang baru melalui *webservice*
5	deleteMedicalUnit	void	id	HttpStatus	*Method* ini digunakan untuk menghapus satu *object* unit medis yang baru melalui *webservice*

4.2.3.2 *Package* Model Medical Unit

Package Medical Unit model berisi entitas-entitas penyusun dari *service* Medical Unit. Berikut ini merupakan daftar *class* untuk *package* model.

Tabel 4.40 Daftar *Class* pada *Package* model

Package	**Class**	**Jenis Class**
Model	MedicalUnit	*Class*
	WorkingUnit	*Class*
	WorkingUnitMembership	*Class*
	WorkingUnitMembershipPK	*Class*
	WUGroup	*Class*
	AddressType	*Class*
	Contact	*Class*
	ContactAddress	*Class*
	ContactEducation	*Class*
	Country	*Class*
	Department	*Class*
	Employee	*Class*
	Employment	*Class*
	Jobspeciality	*Class*
	Province	*Class*
	RoleInDept	*Class*
	Regency	*Class*

Tabel 4.41 Daftar attribut pada *Class* WorkingUnit

Variabel:	**Variabel:**
Long	systemid
WUGroup	wugroup
String	workunit_name
String	memo
Calendar	createdate
Calendar	lastupdate

Tabel 4.42 Daftar attribut pada *Class* MedicalUnit

Variabel:	
Long	systemid

Tabel 4.43 Daftar attribut pada *Class* WorkingUnitMembership

Variabel:	
WorkingUnit	workingunit
Employee	employee
boolean	workingunitpersonincharge
String	description
Calendar	sys_createdate
Calendar	syssys_lastupdate

Tabel 4.44 Daftar attribut pada *Class* WUGroup

Variabel:	
Long	systemid
String	groupname
String	memo

Tabel 4.45 Daftar attribut pada *Class* WorkingUnitMembershipPK

Variabel:	
Long	workingunit
Long	employee

Tabel 4.46 Daftar attribut pada *Class* Employment

Variabel:		**Variabel:**	
Long	systemid	Employee	employee
String	ein	Date	hiredate
BigInteger	basesalary	RoleInDept	roleIndept
Integer	contracttype	String	sourceinstitution
int	sourcetype	Date	createdate
Date	lastupdate	Date	terminatedate

Tabel 4.47 Daftar attribut pada *Class* Employee

Variabel:	
String	m_ein
Jobspeciality	jobspeciality
Collection<Employment>	employmentCollection

Tabel 4.48 Daftar attribut pada *Class* Departement

Variabel:	
int	systemid
String	deptname
String	memo
Collection<RoleInDept>	roles
Date	createdate
Date	lastupdate

Tabel 4.49 Daftar attribut pada *Class* ContactAddress

Variabel:		**Variabel:**	
Long	systemid	String	postcode
AddressType	addresstype	Regency	regency
String	street	boolean	asbillingaddress
String	fax	Date	sys_createdate
Contact	owner	String	sys_creator
String	phone	Date	sys_lastupdate
String	sys_lastupdater		

Tabel 4.50 Daftar attribut pada *Class* Contact

Variabel:		**Variabel:**	
long	systemid	String	initial
String	lastname	int	amountofchildren
int	maritalstatus	List<ContactAddress>	arrAddress
String	middlename	Collection<ContactEducation>	arrEdu
Date	birthday	String	notes
String	birthplace	String	officeext
String	bloodtype	String	passportid
double	bodyheight	byte[]	photo
double	bodyweight	String	prefixtitle
String	citizenid	String	sms
String	citizentype	String	suffixtitle
Country	citizenship	Date	sys_createdate
String	email	String	sys_creator
String	firstname	Date	sys_lastupdate
int	gender	String	sys_lastupdater
String	homephone	String	taxid
String	messaging	String	web

Tabel 4.51 Daftar attribut pada *Class* AddressType

Variabel:	
Integer	systemid
String	memo
String	typename

Tabel 4.52 Daftar attribut pada *Class* ContactEducation

Variabel:		**Variabel:**	
Date	finishrolling	Contact	owner
String	gpa	String	sponsors
int	grade	Date	startrolling
String	institutionaladdress	Date	sys_createdate
String	institutionname	int	sys_creator
String	major	Date	sys_lastupdate
String	notes	int	sys_lastupdater

Tabel 4.53 Daftar attribut pada *Class* Country

Variabel:	
Long	systemid
String	name

Tabel 4.54 Daftar attribut pada *Class* Jobspeciality

Variabel:	
Long	systemid
String	specialityName
String	memo

Tabel 4.55 Daftar attribut pada *Class* RoleInDept

Variabel:	
int	systemid
Department	department
String	rolename
String	abbreviation
RoleInDept	parentRole
String	memo

Tabel 4.56 Daftar attribut pada *Class* Province

Variabel:	
Long	systemid
Country	countryCode
String	name
Set<Regency>	regencySet

Tabel 4.57 Daftar attribut pada *Class* Regency

Variabel:	
Long	systemid
Province	provId
String	name

4.2.3.3 *Package* Repository

Package repository berisi *class-class* yang menjadi representasi model dari tabel-tabel yang berada pada basis data. Berikut ini merupakan *class-class* yang terdapat pada *package* repository.

Tabel 4.58 Daftar *Class* pada *Package* repository

Package	**Class**	**Jenis Class**
repository	UnitmedisRepository	*Interface*

4.2.3.4 *Package* Service

Package service berisi *class-class* yang menginisialisasi *web service* human resource yang akan terbentuk dan digunakan pada *package* controller. Berikut ini merupakan *class-class* yang terdapat pada *package* service.

Tabel 4.59 Daftar *Class* pada *Package* service

Package	**Class**	**Jenis Class**
service	CRUDService	*Interface*
	UnitmedisService	*Interface*
	DefaultUnitmedisService	*Class*

4.2.4 *Project* Rawat Jalan

Project Rawat Jalan berisi susunan *class* pembentuk *webservice* Rawat Jalan. Dalam project ini terdapat 4 buah *package* yang berfungsi untuk memisahkan *class* sesuai dengan fungsinya masing-masing. Ke empat *package* ini yaitu *package* controller yang berisi method-method yang dijalankan pada *web service*, *package* model yang berisi entitas dan tabel, *package* repository yang menghubungkan entitas di *package* model dengan database, dan *package* service yang menginisialisasi *webserviec* yang akan terbentuk. Pada bagian ini akan dijelaskan mengenai *class* dan *method* yang digunakan pada pengembangan *webservice* Rawat Jalan:

4.2.4.1 *Package* Controller

Package controller berisi method API yang dapat digunakan untuk berkomunikasi dengan *service* Rawat Jalan. *Class* yang terdapat dalam *package* ini adalah *class* OutpatientWUQueueController yang berisi 5 buah fungsi API. Di bawah ini merupakan daftar *class* untuk *package* controller pada *webservice* Rawat Jalan.

Tabel 4.60 Daftar *Class* pada *Package* Controller

Package	**Class**	**Jenis Class**
Controller	OutpatientWUQueueController	*Class*

Tabel 4.61 Daftar *Method* pada *Class* OutpatientWUQueueController

No	*Method*	*Input*		*Output*	Keterangan
		Tipe	**Variabel**		
1	addOutpatient Queue	void	Outpatient WUQueue	Outpatient WUQueue, HttpStatus	*Method* ini digunakan untuk membuat *object* rawat jalan yang baru melalui *webservice*
2	updateOutpatient Queue	void	Outpatient WUQueue	HttpStatus	*Method* ini digunakan untuk mengubah attribut dari *object* rawat jalan yang baru melalui *webservice*
3	getOutpatient Queue	void	id	Outpatient WUQueue, HttpStatus	*Method* ini digunakan untuk mengambil satu *object* rawat jalan yang baru melalui *webservice*
4	getAllOut patientQueue	void	-	<List <OutpatientWU Queue>>, HttpStatus	*Method* ini digunakan untuk mengambil semua *list object* rawat jalan yang baru melalui *webservice*
5	deleteOutpatient Queue	void	id	HttpStatus	*Method* ini digunakan untuk menghapus satu *object* rawat jalan yang baru melalui *webservice*

4.2.4.2 *Package* Model Rawat Jalan

Package Rawat Jalan model berisi entitas-entitas penyusun dari *service* Rawat Jalan. Berikut ini merupakan daftar *class* untuk *package* model.

Tabel 4.62 Daftar *Class* pada *Package* model

Package	Class	Jenis Class
Model	OutpatientWUQueue	*Class*
	OutpatientWUQueueHistory	*Class*
	OutpatientWUQueuePK	*Class*
	WorkingUnit	*Class*
	WorkingUnitMembership	*Class*
	WorkingUnitMembershipPK	*Class*
	WUGroup	*Class*
	AddressType	*Class*
	Contact	*Class*
	ContactAddress	*Class*
	ContactEducation	*Class*
	Customer	*Class*
	Customergroup	*Class*
	HealthConsumer	*Class*
	Insurance	*Class*
	InsuranceBridgeConf	*Class*
	Profession	*Class*
	Country	*Class*
	Department	*Class*
	Employee	*Class*
	Employment	*Class*
	Jobspeciality	*Class*
	Province	*Class*
	RoleInDept	*Class*
	Regency	*Class*
	MedicalUnit	*Class*

Tabel 4.63 Daftar attribut pada *Class* OutpatientWUQueue

Variabel:		**Variabel:**	
OutpatientWUQueue	hc	Calendar	syscreatetime
MedicalUnit	medunit	Calendar	syslastupdate
String	regis_no	Insurance	insurance
Date	registertimee	String	insuranceno
Date	finishtime	String	insurancetype
int	priority	String	via
String	memo	String	rujukan_tipe
String	rujukan_doc_no	String	rujukan_nama_asal
OutpatientWUQueueHistory	history	String	insuranceprg

Tabel 4.64 Daftar attribut pada *Class* outpatienthistory

Variabel:		**Variabel:**	
OutpatientWUQueue	hc	Calendar	syscreatetime
MedicalUnit	medunit	Calendar	syslastupdate
String	regis_no	Insurance	insurance
Date	registertimee	String	insuranceno
Date	finishtime	String	insurancetype
int	priority	String	via
String	memo	String	rujukan_tipe
String	rujukan_nama_asal	String	rujukan_doc_no
String	insuranceprg		

Tabel 4.65 Daftar attribut pada *Class* OutpatientWUQueuePK

Variabel:	
long	hc
long	medunit

Tabel 4.66 Daftar attribut pada *Class* HealthConsumer

Variabel:	
String	regis_no
Profession	id_prof
Insurance	insurance
String	insurance_type
String	insurance_no
String	insurance_prg

Tabel 4.67 Daftar attribut pada *Class* Insurance

Variabel:	
List<InsuranceBridgeConf>	insuranceBrigdeConfList
int	systemid
String	insurance
String	memo
boolean	active
boolean	sysbuiltin

Tabel 4.68 Daftar attribut pada *Class* InsuranceBridgeConf

Variabel:	
Long	systemid
String	field
String	def_val
String	memo

Tabel 4.69 Daftar attribut pada *Class* Profession

Variabel:	
Integer	systemid
String	profname
String	memo
Calendar	sys_createdate
Calendar	last_createdate

Tabel 4.70 Daftar attribut pada *Class* Customer

Variabel:		**Variabel:**	
boolean	cust_no	Date	registrationdate
boolean	active	int	idPriceLevel
double	creditlimit	double	disc
Customergroup	customergroup		

Tabel 4.71 Daftar attribut pada *Class* Customergroup

Variabel:	
long	systemid
String	groupname
String	memo

Tabel 4.72 Daftar attribut pada *Class* WorkingUnit

Variabel:	
Long	systemid
WUGroup	wugroup
String	workunit_name
String	memo
Calendar	createdate
Calendar	lastupdate

Tabel 4.73 Daftar attribut pada *Class* MedicalUnit

Variabel:	
Long	systemid

Tabel 4.74 Daftar attribut pada *Class* WorkingUnitMembership

Variabel:	
WorkingUnit	workingunit
Employee	employee
boolean	workingunitpersonincharge
String	description
Calendar	sys_createdate
Calendar	syssys_lastupdate

Tabel 4.75 Daftar attribut pada *Class* WUGroup

Variabel:	
Long	systemid
String	groupname
String	memo

Tabel 4.76 Daftar attribut pada *Class* WorkingUnitMembershipPK

Variabel:	
Long	workingunit
Long	employee

Tabel 4.77 Daftar attribut pada *Class* Employment

Variabel:		**Variabel:**	
Long	systemid	Employee	employee
String	ein	Date	hiredate
BigInteger	basesalary	RoleInDept	roleIndept
Integer	contracttype	String	sourceinstitution
int	sourcetype	Date	createdate
Date	lastupdate	Date	terminatedate

Tabel 4.78 Daftar attribut pada *Class* Employee

Variabel:	
String	m_ein
Jobspeciality	jobspeciality
Collection<Employment>	employmentCollection

Tabel 4.79 Daftar attribut pada *Class* Departement

Variabel:	
int	systemid
String	deptname
String	memo
Collection<RoleInDept>	roles
Date	createdate
Date	lastupdate

Tabel 4.80 Daftar attribut pada *Class* ContactAddress

Variabel:		**Variabel:**	
Long	systemid	String	postcode
AddressType	addresstype	Regency	regency
String	street	boolean	asbillingaddress
String	fax	Date	sys_createdate
Contact	owner	String	sys_creator
String	phone	Date	sys_lastupdate
String	sys_lastupdater		

Tabel 4.81 Daftar attribut pada *Class* Contact

Variabel:		**Variabel:**	
long	systemid	String	initial
String	lastname	int	amountofchildren
int	maritalstatus	List<ContactAddress>	arrAddress
String	middlename	Collection<ContactEducation>	arrEdu
Date	birthday	String	notes
String	birthplace	String	officeext
String	bloodtype	String	passportid
double	bodyheight	byte[]	photo
double	bodyweight	String	prefixtitle
String	citizenid	String	sms
String	citizentype	String	suffixtitle
Country	citizenship	Date	sys_createdate
String	email	String	sys_creator
String	firstname	Date	sys_lastupdate
int	gender	String	sys_lastupdater
String	homephone	String	taxid
String	messaging	String	web

Tabel 4.82 Daftar attribut pada *Class* AddressType

Variabel:	
Integer	systemid
String	memo
String	typename

Tabel 4.83 Daftar attribut pada *Class* ContactEducation

Variabel:		**Variabel:**	
Date	finishrolling	Contact	owner
String	gpa	String	sponsors
int	grade	Date	startrolling
String	institutionaladdress	Date	sys_createdate
String	institutionname	int	sys_creator
String	major	Date	sys_lastupdate
String	notes	int	sys_lastupdater

Tabel 4.84 Daftar attribut pada *Class* Country

Variabel:	
Long	systemid
String	name

Tabel 4.85 Daftar attribut pada *Class* Jobspeciality

Variabel:	
Long	systemid
String	specialityName
String	memo

Tabel 4.86 Daftar attribut pada *Class* RoleInDept

Variabel:	
int	systemid
Department	department
String	rolename
String	abbreviation
RoleInDept	parentRole
String	memo

Tabel 4.87 Daftar attribut pada *Class* Province

Variabel:	
Long	systemid
Country	countryCode
String	name
Set<Regency>	regencySet

Tabel 4.88 Daftar attribut pada *Class* Regency

Variabel:	
Long	systemid
Province	provId
String	name

4.2.4.3 *Package* Repository

Package repository berisi *class-class* yang menjadi representasi model dari tabel-tabel yang berada pada basis data. Berikut ini merupakan *class-class* yang terdapat pada *package* repository.

Tabel 4.89 Daftar *Class* pada *Package* repository

Package	**Class**	**Jenis Class**
repository	OutpatientWUQueueRepository	*Interface*

4.2.4.4 *Package* Service

Package service berisi *class-class* yang menginisialisasi *web service* human resource yang akan terbentuk dan digunakan pada *package* controller. Berikut ini merupakan *class-class* yang terdapat pada *package* service.

Tabel 4.90 Daftar *Class* pada *Package* service

Package	**Class**	**Jenis Class**
service	CRUDService	*Interface*
	OutpatientWUQueueService	*Interface*
	DefaultOutpatientWUQueueService	*Class*

4.3 Perbandingan Arsitektur

Pada bab ini, akan dijelaskan perbandingan dari arsitektur microservice dan monolitik. Pengujian akan dilakukan dengan membuat *test plan* yang kemudian akan didokumentasikan ke dalam*matrix traceability*. Dari hasil *matrix traceability* tersebut dapat ditentukan apakah desain microservice yang baru memberikan performa yang lebih baik dibandingkan dengan model monolitik yang lama.

4.3.1 Pengujian dan Perbandingan Model Arsitektur Microservice dan Monolitik

Pada bagian ini akan dilakukan pengujian terhadap kedua jenis model arsitektur yang didokumentasikan ke dalam *matrix of traceability*. Pengujian akan dilakukan dengan membandingkan hasil *test plan* yang telah dibuat. *Test plan* yang dibuat akan mengacu pada masalah *deployability, reliability, availability, scalability,* dan *modifiability.*

Tabel 4.91 Tabel *matrix of traceability* pengujian model arsitektur

No	Test Plan	Microservice		Monolitik		Harapan
		Ya	Tidak	Ya	Tidak	
1	Perubahan yang dilakukan pada class dalam sebuah modul harus diketahui oleh modul lain yang saling melakukan pertukaran data		✓	✓		**Tidak**
2	Apabila terjadi masalah terhadap database, maka masalah tersebut akan turut dirasakan oleh keseluruhan modul		✓	✓		**Tidak**
3	Tingkat sekuritas yang lebih baik	✓			✓	**Ya**
4	Tingkat ketergantungan yang besar apabila aplikasi dikerjakan oleh beberapa tim berbeda		✓	✓		**Tidak**
5	Cara komunikasi antar modul lebih mudah untuk diimplementasikan		✓	✓		**Ya**
6	Konsumen yang mengkakses *method* yang diubah harus mengetahui apa perubahan yang terjadi terhadap *method* tersebut		✓	✓		**Tidak**
7	Waktu siklus *build-test-build* yang relatif lebih cepat	✓			✓	**Ya**
8	Tidak saling berebut *resources* untuk setiap modulnya	✓			✓	**Ya**
9	Mudah untuk melakukan *querry join* dari setiap modul yang ada		✓	✓		**Ya**

10	Relatif dapat menangani permasalahan *big data* dan *multiple user*	✓			✓	**Ya**
11	Aplikasi menjadi lebih *long lasting* dan relatif tidak memiliki resiko perombakan major apabila terdapat perubahan pada aplikasi	✓			✓	**Ya**
12	Mudah untuk menambahkan modul baru	✓		✓		**Ya**
13	Lebih mudah dan cepat beradaptasi apabila aplikasi menjadi lebih besar	✓			✓	**Ya**
14	Aplikasi lebih mudah bila ingin dikembangkan menjadi lintas *platform*	✓			✓	**Ya**
15	Aplikasi lebih mudah untuk melakukan *deploy* ke *server*	✓			✓	**Ya**
16	Pengujian alpikasi mudah dan cepat	✓		✓		**Ya**
17	Lebih mudah untuk menerapkan teknologi baru	✓			✓	**Ya**
18	Konfigurasi aplikasi mudah		✓	✓		**Ya**
19	Perlu dilakukan banyak automasi agar aplikasi dapat memberikan performa terbaiknya		✓	✓		**Ya**
20	Cocok untuk sistem yang dinamis dan konstan berkembang	✓			✓	**Ya**
21	Jumlah waktu *downtime* yang dibutuhkan aplikasi apabila terjadi *maintenance* lebih sedikit	✓			✓	**Ya**

Berdasarkan hasil pengujian di atas, desain arsitektur microservice memenuhi 17 dari 21 kasus *test plan* yang dibuat, sedangkan desain arsitektur monolitik memenuhi 6 dari 21 kasus *test plan*.

BAB V

PENUTUP

Bab ini berisi kesimpulan dan saran dari implementasi dan perbandingan performa arsitektur microservice dengan monolitik.

5.1 Kesimpulan

Kesimpulan dari pembuatan sistem analisis dan pengujian-pengujian yang telah dilakukan adalah sebagai berikut:

1. Untuk dapat melakukan migrasi dari aplikasi monolitik menjadi microservice dibutuhkan perencanaan yang matang, dimulai dari analisis proses bisnis aplikasi, menentukan service yang akan terbentuk, memilih teknologi yang akan digunakan, memilih cara berkomunikasi, dan memilih metode deployment yang tepat.
2. Model arsitektur microservice memberikan hasil uji yang lebih baik dari segi *deployability, reliability, availability, scalibility*, dan *modifiability*. Namun memiliki tingkat kesulitan implementasi yang lebih tinggi dibandingkan dengan model arsitektur monolitik.
3. Arsitektur microservice lebih cocok untuk diimplementasi pada sistem yang membutuhkan landasan yang kuat dan siap untuk berkembang menjadi aplikasi yang besar.
4. Dibutuhkan tools automasi yang lebih banyak untuk diterapkan pada aplikasi berbasis microservice agar mencapai hasil performa yang maksimal.

5.2 Saran

Saran dari penulis untuk pengembangan yang dilakukan untuk arsitektur microservic adalah:

1. Lebih banyak eksplorasi dan menggunakan tools yang dapat menunjang kemudahan deployment dari arsitektur microservice.
2. Menggunakan *framework* untuk penunjang integritas data.

DAFTAR PUSTAKA

[1] Hartono, M. (7). Langkah mudah membangun sistem informasi ERP. Jakarta: PT. Elex Media Komputindo.

[2] Pangera, A. A., & Ariyus, D. ARSITEKTUR SISTEM OPERASI.

[3] Pradhananga, Y., & Rajarajeswari, P. (2017). Tiarrah Computing: The Next Generation of Computing. International Journal of Informatics and Communication Technology (IJ-ICT), 6(2), 129-138.

[4] Chris Richardson (2017). Pattern: Monolithic Architecture.

[5] Polychniatis, T., van der Rijnst, S., van Vliet, R., & Wirken, G. (2013). Software architecture WordPress. Utrecht, NE: Utrecht University. URL: http://www. cs. uu. nl/wiki/pub/Swa/CourseLiterature/arch-E. pdf [June 24, 2015].

[6] Chris Richardson (2017). Pattern: Microservice Architecture

[7] Mazzara, M., Mustafin, R., Safina, L., & Lanese, I. (2016). Towards microservices and beyond: An incoming paradigm shift in distributed computing. arXiv preprint arXiv:1610.01778.

[8] Thönes, J. (2015). Microservices. IEEE Software, 32(1), 116-116.

[9] Sam Newman (2015). Building Microservices. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

[10] Susan J. Fowler (2017). Microservices in Production. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

[11] Ry, K. Siva Prasad (2017). Beginning Spring Boot 2 Applications and Microservices with the Spring Framework. Springer Science and Business Media _ Apress

[12] Ammann, Paul and Offutt, Jeff (2008). Introduction to software testing. New York: Cambridge University Press.

[13] Chris Richardson (2017). MEAP Edition Manning Early Access Program Microservices Patterns With examples in Java Version 11. Manning Publications Co.

[14] Simarmata, J. (2010). Rekayasa Perangkat Lunak. Penerbit Andi.

[15] Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice architecture: aligning principles, practices, and culture. ” O’Reilly Media, Inc.”.

LAMPIRAN