

**PENERAPAN DAN PERBANDINGAN ARSITEKTUR  
MICROSERVICE TERHADAP ARSITEKTUR MONOLITIK  
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 ARSITEKTUR MICROSERVICE UNTUK PERBANDINGAN PERFORMA PADA SOFTWARE 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 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 Language 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 Language 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

<b>LEMBAR PENGESAHAN</b>	<b>i</b>
<b>LEMBAR PERNYATAAN HASIL KARYA PRIBADI</b>	<b>ii</b>
<b>ABSTRAK</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>PEDOMAN PENGGUNAAN TUGAS AKHIR</b>	<b>v</b>
<b>KATA PENGANTAR</b>	<b>vi</b>
<b>DAFTAR ISI</b>	<b>x</b>
<b>DAFTAR TABEL</b>	<b>xii</b>
<b>DAFTAR GAMBAR</b>	<b>xiii</b>
<b>I PENDAHULUAN</b>	<b>1-1</b>
1.1 Latar Belakang Masalah . . . . .	1-1
1.2 Rumusan Masalah . . . . .	1-3
1.3 Batasan Masalah . . . . .	1-3
1.4 Tujuan Penelitian . . . . .	1-4
1.5 Kontribusi Penelitian . . . . .	1-4
1.6 Metode Penelitian . . . . .	1-4
1.7 Sistematika Penulisan . . . . .	1-5
<b>II LANDASAN TEORI</b>	<b>2-1</b>
2.1 Arsitektur Microservice . . . . .	2-1
2.1.1 Definisi Arsitektur Microservice . . . . .	2-1

2.1.2	Prinsip Pendekatan Arsitektur Microservice . . . . .	2-1
2.1.3	Konsep Microservice . . . . .	2-2
2.2	Integrasi Teknologi . . . . .	2-4
2.2.1	Manajemen Data . . . . .	2-5
2.2.2	Dekomposisi Modul . . . . .	2-7
2.2.3	Service Deployment . . . . .	2-9
2.2.4	Metode Berkomunikasi Antar Service . . . . .	2-12
2.3	Strategi Pengujian . . . . .	2-13
2.3.1	Pengujian Arsitektur Microservice . . . . .	2-13
2.3.2	Uji Perbandingan Performa . . . . .	2-14

<b>III ANALISIS DAN PERANCANGAN SISTEM</b>	<b>3-1</b>	
3.1	Arsitektur Monolitik pada Software Apertura . . . . .	3-1
3.2	Tinjauan Umum Proses Bisnis Pelayanan Rawat Jalan . . . . .	3-2
3.3	Pembahasan Modul dan Kelas Penyusun . . . . .	3-5
3.4	Identifikasi Teknologi yang Digunakan pada Aplikasi Apertura . . . . .	3-6
3.4.1	Kekurangan Arsitektur Monolitik . . . . .	3-7
3.5	Perancangan Arsitektur Microservice . . . . .	3-7
3.5.1	Pemilihan Model Penyimpanan Data . . . . .	3-8
3.5.2	Pemilihan Server Basisdata . . . . .	3-8
3.5.3	Pemilihan Komunikasi Antar Service . . . . .	3-9
3.5.4	Rancangan Deployment . . . . .	3-9
3.6	Strategi Pengujian . . . . .	3-9
3.6.1	Uji Perbandingan Performa . . . . .	3-9
3.7	Batasan Implementasi . . . . .	3-10
3.8	Lingkungan Implementasi . . . . .	3-11
<b>IV IMPLEMENTASI DAN PENGUJIAN</b>	<b>4-1</b>	
4.1	Lingkungan Aplikasi . . . . .	4-1

4.2	Daftar <i>Class</i> dan <i>Method</i>	4-1
4.2.1	<i>Class Loader</i>	4-1
4.2.2	<i>Class Preprocessing</i>	4-2
4.2.3	<i>Class Features</i>	4-5
4.2.4	<i>Class Feature Extraction</i>	4-7
4.2.5	<i>Class Sentiment Extraction</i>	4-11
4.2.6	<i>Class Negation</i>	4-12
4.2.7	<i>Class Topic</i>	4-12
4.2.8	<i>Class SVM</i>	4-13
4.2.9	<i>Class FMeasure</i>	4-16
4.2.10	<i>Class Learning</i>	4-17
4.2.11	<i>Class Main</i>	4-18
4.3	Implementasi Perangkat Lunak	4-19
4.3.1	Implementasi Pengambilan Data	4-19
4.3.2	Implementasi <i>Text Preprocessing</i>	4-20
4.3.3	Implementasi <i>Feature Extraction</i>	4-21
4.3.4	Implementasi SVM dengan SMO	4-23
4.4	Pengujian	4-24
4.4.1	Pengujian Kombinasi Fitur	4-24
4.4.2	Pengujian Parameter pada SMO	4-35
4.4.3	Pengujian Klasifikasi 4 kelas, 3 kelas dan 1 kelas	4-36

<b>V</b>	<b>PENUTUP</b>	<b>5-1</b>
5.1	Kesimpulan	5-1
5.2	Saran	5-2

<b>DAFTAR PUSTAKA</b>	<b>xiv</b>
-----------------------	------------

## DAFTAR TABEL

4.1 Daftar <i>Method</i> pada <i>Class Loader</i> . . . . .	4-2
4.2 Daftar <i>Method</i> pada <i>Class Preprocessing</i> . . . . .	4-2
4.3 Daftar <i>Method</i> pada <i>Class Features</i> . . . . .	4-5
4.4 Daftar <i>Method</i> pada <i>Class FeatureExtraction</i> . . . . .	4-8
4.5 Daftar <i>Method</i> pada <i>Class SentimentExtraction</i> . . . . .	4-11
4.6 Daftar <i>Method</i> pada <i>Class Negation</i> . . . . .	4-12
4.7 Daftar <i>Method</i> pada <i>Class Topic</i> . . . . .	4-12
4.8 Daftar <i>Method</i> pada <i>Class SVM</i> . . . . .	4-13
4.9 Daftar <i>Method</i> pada <i>Class FMeasure</i> . . . . .	4-16
4.10 Daftar <i>Method</i> pada <i>Class FMeasure</i> . . . . .	4-17
4.11 Daftar <i>Method</i> pada <i>Class FMeasure</i> . . . . .	4-18
4.12 Tabel Pengujian Kombinasi Fitur 1 . . . . .	4-25
4.12 Tabel Pengujian Kombinasi Fitur 1 . . . . .	4-26
4.13 Tabel Pengujian Kombinasi Fitur 2 . . . . .	4-27
4.14 Analisis <i>Error</i> Fitur <i>Unigram</i> pada Klasifikasi Non-Sarkasme . . . . .	4-28
4.15 Analisis <i>Error</i> Fitur <i>Sentiment Score</i> pada Klasifikasi Non-Sarkasme	4-29
4.16 Analisis <i>Error</i> Fitur <i>Punctuation Based</i> pada Klasifikasi Non-Sarkasme . . . . .	4-29
4.17 Analisis <i>Error</i> Fitur <i>Unigram</i> pada Klasifikasi Sarkasme . . . . .	4-30
4.18 Analisis Fitur <i>Topic</i> . . . . .	4-31
4.18 Analisis Fitur <i>Topic</i> . . . . .	4-32
4.18 Analisis Fitur <i>Topic</i> . . . . .	4-33
4.19 Analisis <i>Error</i> Fitur <i>Topic</i> pada Klasifikasi Sarkasme . . . . .	4-34
4.20 Analisis <i>Error</i> Fitur <i>Capitalization</i> pada Klasifikasi Sarkasme . . . . .	4-35
4.21 Pengujian Parameter SMO . . . . .	4-36

4.22 Pengujian Klasifikasi 4 Kelas (Positif, Negatif, Netral, Sarkasme) . .	4-37
4.23 Pengujian Klasifikasi 3 Kelas (Positif, Negatif, Netral) . . . . .	4-37
4.24 Pengujian Klasifikasi 1 Kelas (Sarkasme/Non-Sarkasme) . . . . .	4-37

## **DAFTAR GAMBAR**

2.1	Model pembagian dari departemen keuangan dan warehouse. . . . .	2-3
2.2	Pemodelan cara akses database yang umum. . . . . . . . . . . . . . .	2-5
2.3	<i>Shared database.</i> .	2-6
2.4	<i>Banyak service per host dan satu service per host.</i> . . . . . . . . .	2-10
2.5	<i>Menggunakan container untuk deployment.</i> . . . . . . . . . . . . . . .	2-11
2.6	<i>Contoh pemanfaatan REST sebagai media user-server</i> . . . . . . . . .	2-13
2.7	<i>Testing pada software microservice</i> . . . . . . . . . . . . . . . . . . .	2-14
3.1	Pemodelan Software Apertura dengan deployment diagram. . . . .	3-2
3.2	Proses bisnis rawat jalan. .	3-4
3.3	Komponen diagram rawat jalan. .	3-6
3.4	Pembentukan service berdasarkan proses bisnis rawat jalan. . . . .	3-8

# BAB I

## PENDAHULUAN

### 1.1 Latar Belakang Masalah

Sistem computer 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 disini adalah desain dan pengembangan, yang dimana peran arsitektur perangkat lunak sangat berperan penting untuk menetapkan landasan dasar pengembangan aplikasi dari awal sampai selesai. Hasi dari arsitektur perangkat lunak merupakan struktur-struktur yang menjadikan landasan untuk menentukan keberadaan komponen-komponen perangkat lunak, cara komponen-komponen untuk saling berinteraksi dan organisasi komponen-komponen dalam membentuk perangkat lunak [2]. Secara umum perangkat lunak bekerja untuk pengguna pada *desktop browser*, *mobile browser*, dan aplikasi *browser* lainnya. Aplikasi tersebut mungkin akan menggunakan API (*Application Programming Interface*) sebagai pihak ke 3. Aplikasi juga dapat saling berintegrasi dengan aplikasi lain dengan menggunakan *web service*. Aplikasi bekerja dengan menerima *request* (*HTTP request* dan pesan) dengan menjalankan logika perhitungan, mengakses database, bertukar pesan dengan sistem lain, dan mengembalikan HTML/JSON/XML sebagai respon balikan [4].

IEEE 803:1993 mengelompokkan kebutuhan non-fungsional ke dalam sejumlah kategori kualitas dari suatu perangkat lunak, yaitu: ketepatan (*correctness*), *robustness*, performa, ketersediaan dan kualitas antarmuka (*interface*), keandalan (*reability*), ketersediaan (*availability*) [8]. Ketika skala aplikasi masih kecil dan sedikit data yang digunakan, kebutuhan masih mudah untuk dipenuhi, namun ketika aplikasi semakin besar, akan terjadi masalah yang selain disebabkan oleh data yang banyak, namun juga oleh *load* komputasi yang besar yang berasal dari *multiple user* dari berbagai lokasi.

Model 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. Contoh yang dapat diambil adalah aplikasi Wordpress. Wordpress merupakan contoh yang mudah untuk menggambarkan sebuah aplikasi monolitik, dimana semua fitur seperti *security*, performa, manajemen konten, statistik dan semuanya dibangun dengan menggunakan PHP dan database MySQL dalam kode yang sama [5]. 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. Pertama, ketika aplikasi semakin besar, barisan code monolitik akan menyulitkan *developer* terutama yang baru bergabung bersama tim, aplikasi akan sulit dimengerti dan di modifikasi. Akibatnya pertumbuhan aplikasi akan melambat dan terlebih karena sulit dimengerti, kualitas kode akan semakin menurun. Kedua, semakin banyak code yang ditulis, maka akan semakin lambat IDE (*Integrated Development Environment*) yang digunakan, semakin tidak produktif pula proses development yang dilakukan. Hal ini juga berpengaruh pada waktu yang dibutuhkan untuk menjalankan aplikasi pertama kali, serta menjadi semakin sulit untuk memodifikasi aplikasi. Seperti untuk mengubah sebuah komponen, *developer* harus *redeploy* keseluruhan aplikasi. Ketiga, akan sulit untuk membagi team secara fungsionalitas, seperti misalnya membagi tim akunting dan tim inventori. Kedua tim tersebut tidak dapat secara mandiri bekerja sendiri, karena hanya ada 1 aplikasi besar yang mengakibatkan adanya saling ketergantungan [7].

Model arsitektur microservice adalah pattern alternatif yang dapat mengatasi keterbatasan dari arsitektur monolitik, model ini mulai muncul ke permukaan di tahun 2015 (Google trend). Menurut Thones, J. (2015) kebanyakan aplikasi mulai dari arsitektur monolitik, sampai hingga aplikasi itu sulit di kembangkan lagi, kemudian aplikasi dipecah menjadi model microservice, hal itu yang terjadi pada perusahaan besar seperti Netflix dan Amazon [8]. Secara garis besar arsitektur microservice mendefinisikan struktur service yang lebih sempit dengan area fungsi yang saling berkaitan. Tiap servis saling berkomunikasi menggunakan protokol seperti HTTP dan setiap servis bisa mempunyai databasenya sendiri masing-masing. Arsitektur microservice mengubah servis aplikasi menjadi modul

yang mandiri, kecil (dibandingkan monolitik), dan setiap servis berjalan sesuai dengan perannya masing-masing dan tidak saling ketergantungan [6]. Dalam artikel yang dijelaskan oleh Chris Richardson, ada banyak pattern model dari arsitektur microservice, tergantung dari aplikasi yang akan dimigrasi.

Dalam contoh kasus penelitian ini, peneliti akan menerapkan arsitektur microservice pada aplikasi monolitik rumah sakit Apertura. Aplikasi rumah sakit Apertura mengalami kendala dalam hal perawatan dan pengembangan lanjutan terhadap aplikasinya. Hal ini dapat dirasakan dengan banyaknya data yang semakin banyak dan sulit untuk dipelihara. Kasus lain apabila terjadi kegagalan hardware yang menyebabkan database tidak dapat diakses. Arsitektur aplikasi saat ini tidak dapat menunjang untuk mengatasi masalah tersebut, sehingga segala kegiatan yang berhubungan dengan database akan lumpuh total. Performa kecepatan yang menurun seiring dengan banyaknya *actor* yang mengakses aplikasi yang disebabkan selain karena aplikasi yang terus membesar, juga karena data input dan kebutuhan tiap *actor* yang semakin hari semakin banyak.

Tujuan dari penelitian ini adalah menerapkan arsitektur microservice yang paling cocok untuk kasus aplikasi rumah sakit Apertura, menjelaskan bagaimana tahapan migrasi ke arsitektur microservice, menjelaskan bagaimana analisis yang baik agar dapat menentukan desain arsitektur microservice yang benar, serta membandingkan performa antara arsitektur microservice dan arsitektur monolitik dari segi kecepatan, beban hardware yang dibutuhkan, kemudahan pengembangan, *high availability*, dan lain lain.

### 1.2 Rumusan Masalah

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

1. Bagaimana menentukan pattern microservice dan desain arsitektur yang paling tepat/sesuai untuk aplikasi rumah sakit Apertura?
2. Bagaimana melakukan migrasi dari arsitektur monolitik ke model arsitektur microservice?
3. 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 berfokus pada proses rawat jalan dan tidak melibatkan rawat inap.

1. Penelitian berfokus pada proses rawat jalan dan tidak melibatkan rawat inap.

## I. PENDAHULUAN

---

2. Sebagian data pasien dan HR yang digunakan dalam penelitian bersifat tertutup karena menyangkut hal privasi dari rumah sakit Apertura.
3. Fokus utama dari penelitian ini adalah berupa analisis perancangan arsitektur dan uji coba perbandingan performa yang dibuat dalam bentuk prototype.

### 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.
3. Mengetahui bagaimana analisa untuk membagi servis monolitik menjadi servis microservice yang lebih fokus dan padat.

### 1.5 Kontribusi Penelitian

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

1. Memberikan panduan untuk melakukan migrasi dari arsitektur konvensional ke arsitektur microservice.
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.

## **I. PENDAHULUAN**

---

### **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

## **I. PENDAHULUAN**

---

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 pemodelan microservice, membangun arsitektur 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

Arsitektur microservice pertama kali muncul untuk memenuhi kebutuhan dan menunjukkan bagaimana sebuah aplikasi dapat lebih efektif dalam tahap *production*, juga menunjukkan bagaimana cara *development* yang lebih baik dengan memberikan kemampuan kepada mesin untuk saling berkomunikasi. Microservice juga termasuk ke dalam perancangan infrastruktur 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 sistem 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

## II. LANDASAN TEORI

---

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 tersebut 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 *lifecyclecoding* 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 praktik 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

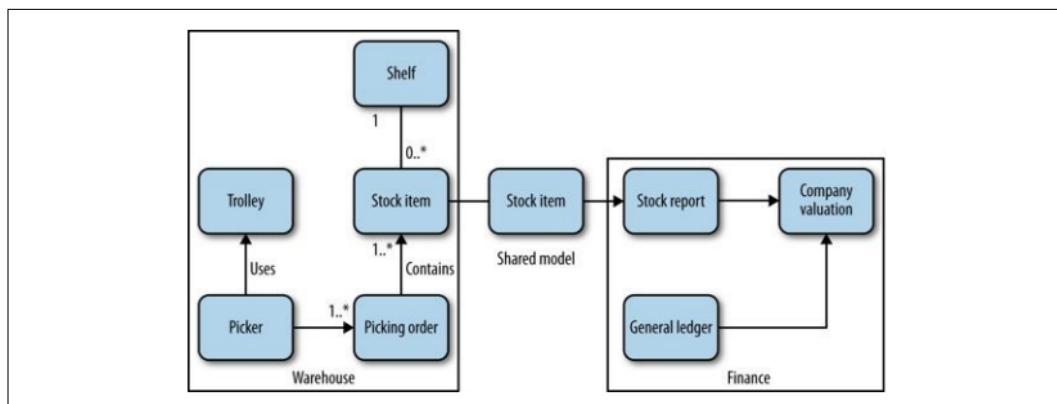
## II. LANDASAN TEORI

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 tercerer, 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 *strongly 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 client [9].

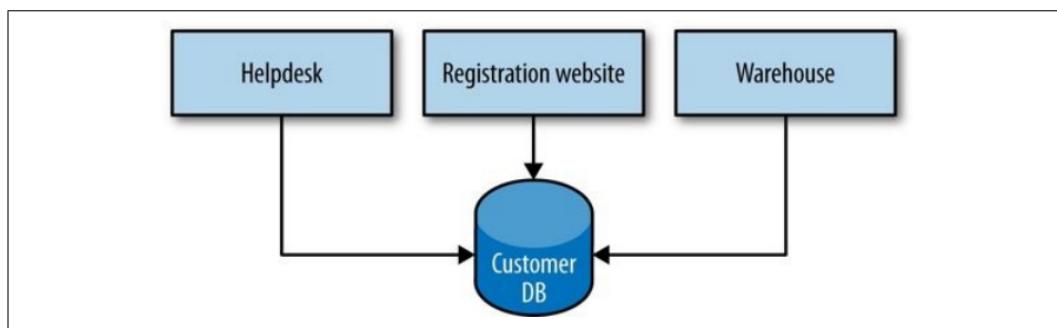
### 2.2.1 Manajemen Data

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.

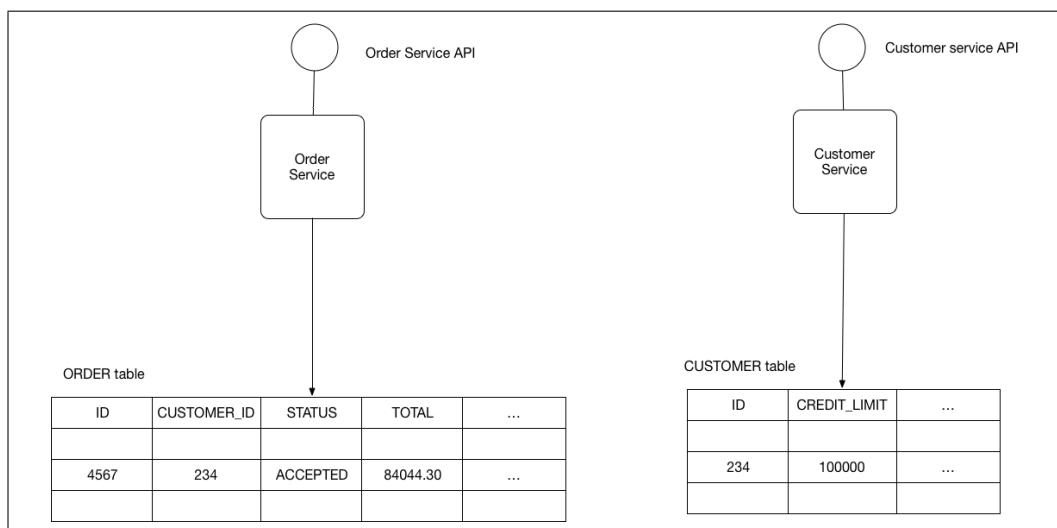
Pertama, model ini mengizinkan langsung pihak luar untuk mengubah data internal. Struktur data yang di simpan di DB dipakai oleh semua *user*, apabila terjadi perubahan pada DB, maka semua *user* akan terkena dampaknya. Database menjadi sangat besar, dan *shared API* menjadi rapuh. Apabila akan terjadi 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

## II. LANDASAN TEORI

*relational* database, namun bagaimana bila seiring berjalananya 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*.

## II. LANDASAN TEORI

---

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 menjadi 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 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

## II. LANDASAN TEORI

---

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. Kemampuan bisnis adalah proses yang dilakukan untuk menghasilkan sebuah nilai. Kemampuan bisnis biasanya berhubungan dengan objek bisnis. Misalnya *order management*

## II. LANDASAN TEORI

---

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 mengidentifikasi 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.

### 2.2.3 Service Deployment

Setelah mengidentifikasi 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

## II. LANDASAN TEORI

---

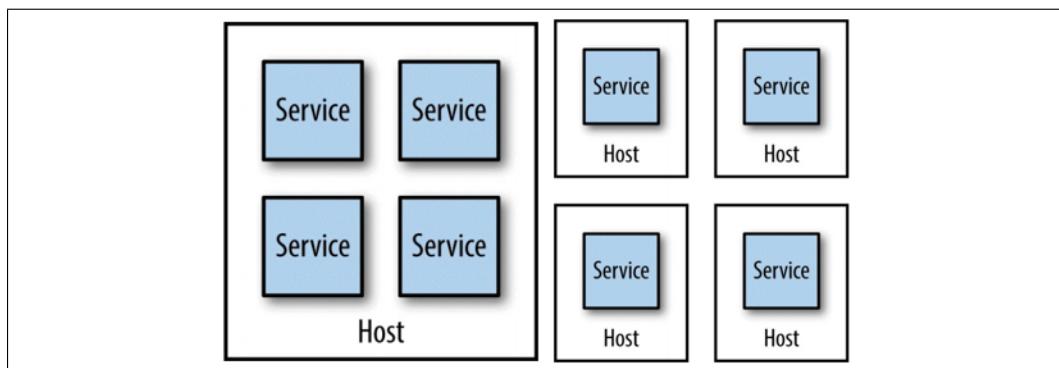
untuk sebuah set service. Kedua adalah melakukan *deploy* semua set service dalam sebuah JVM yang sama, mislanya sebagai web aplikasi [6].

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]

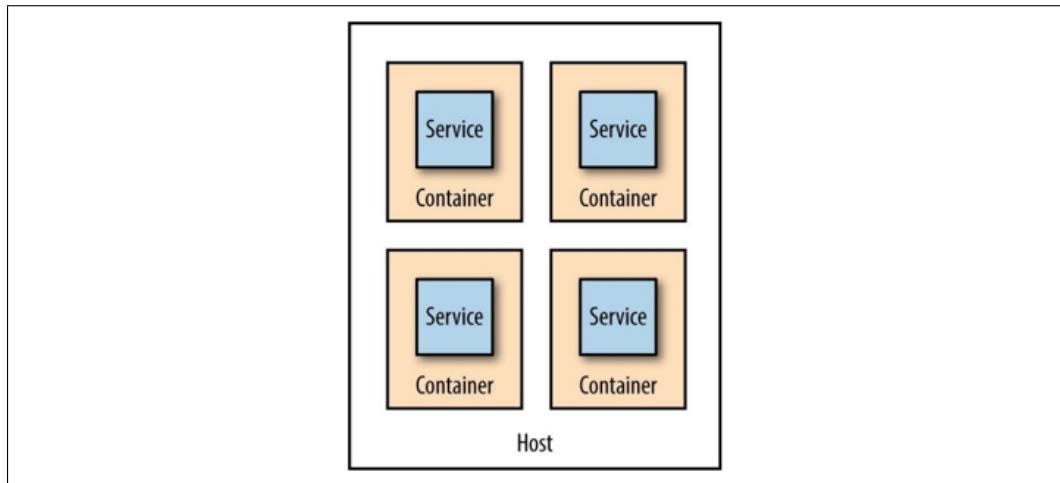


**Gambar 2.4** Banyak service per host dan satu service per host.

### 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 2.5** 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 membuat 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.

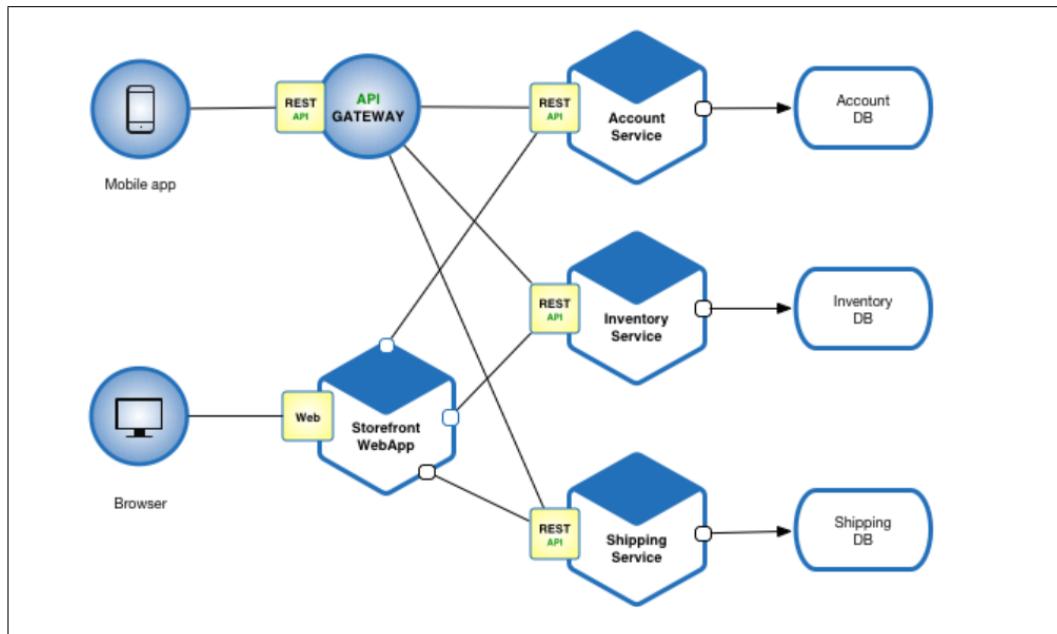
### 2.2.4 Metode Berkommunikasi Antar Service

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

## II. LANDASAN TEORI

HTTP sangat baik untuk diimplementasikan pada interaksi *service-to-service* [9]



Gambar 2.6 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].

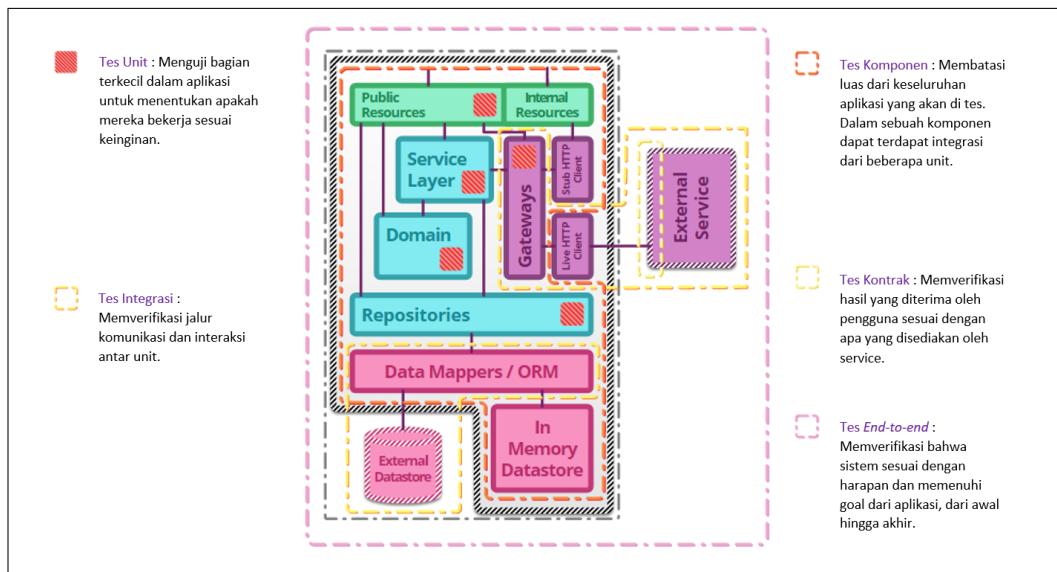
#### 2.3.1 Pengujian Arsitektur Microservice

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.

## II. LANDASAN TEORI

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 2.7 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 menunjukkan bahwa performa dari arsitektur microservice akan memberikan hasil yang lebih baik dari monolitik. Strategi pengujian yang akan dilakukan adalah dengan membuat point-point pembanding yang akan diuji, kemudian dari point tersebut akan dibuat *test-plan* masing-masing yang 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 consultant), point pembanding tersebut adalah [12]:

1. *Deployability*. Lebih agile untuk meluncurkan versi terbaru karena siklus

## II. LANDASAN TEORI

---

*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.

Tujuan dari test performa ini adalah menunjukan bahwa arsitektur microservice yang baru akan lebih unggul dalam 5 point diatas apabila dibandingkan dengan monolitik. Strategi berikutnya adalah membuat *test-plan* yang sesuai dengan ke-5 point tersebut, *test-plan* akan didokumentasikan dalam bentuk *matrix traceability* (Bab 4).

## 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

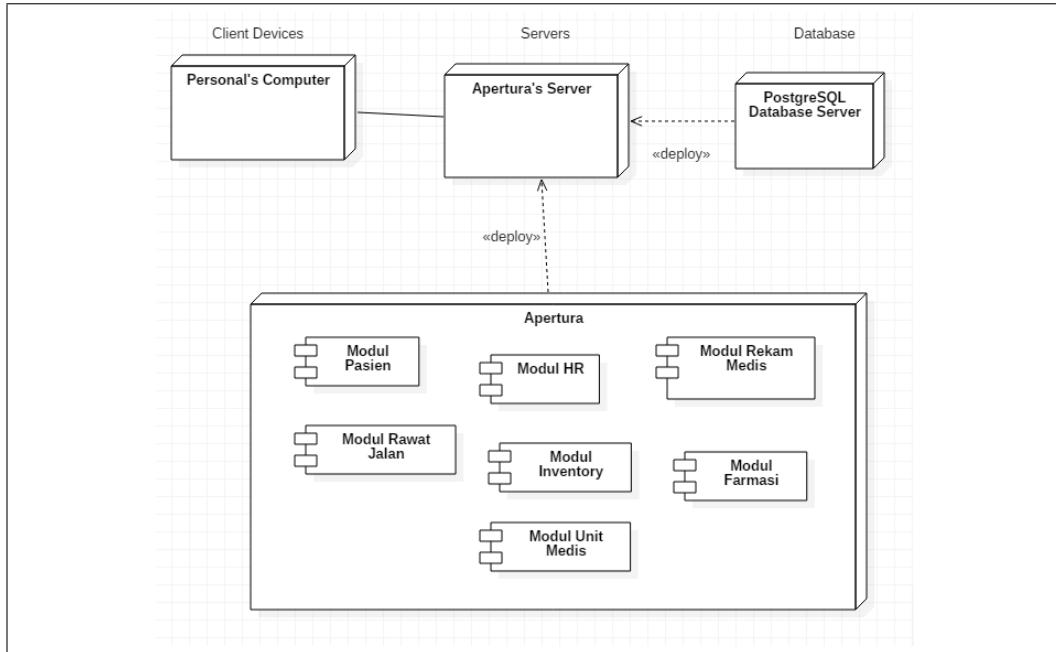
Menurut Sam Newman, terdapat 2 parameter yang membedakan arsitektur monolitik dengan microservice, pertama adalah tingkat *coupling* dan yang kedua adalah tingkat *cohesion* dari aplikasi tersebut. Pernyataan ini kemudian didukung oleh Chris Richardson yang menjelaskan bahwa kedua parameter ini dapat 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 *query* 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.

### III. ANALISIS DAN PERANCANGAN SISTEM

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.

### **III. ANALISIS DAN PERANCANGAN SISTEM**

---

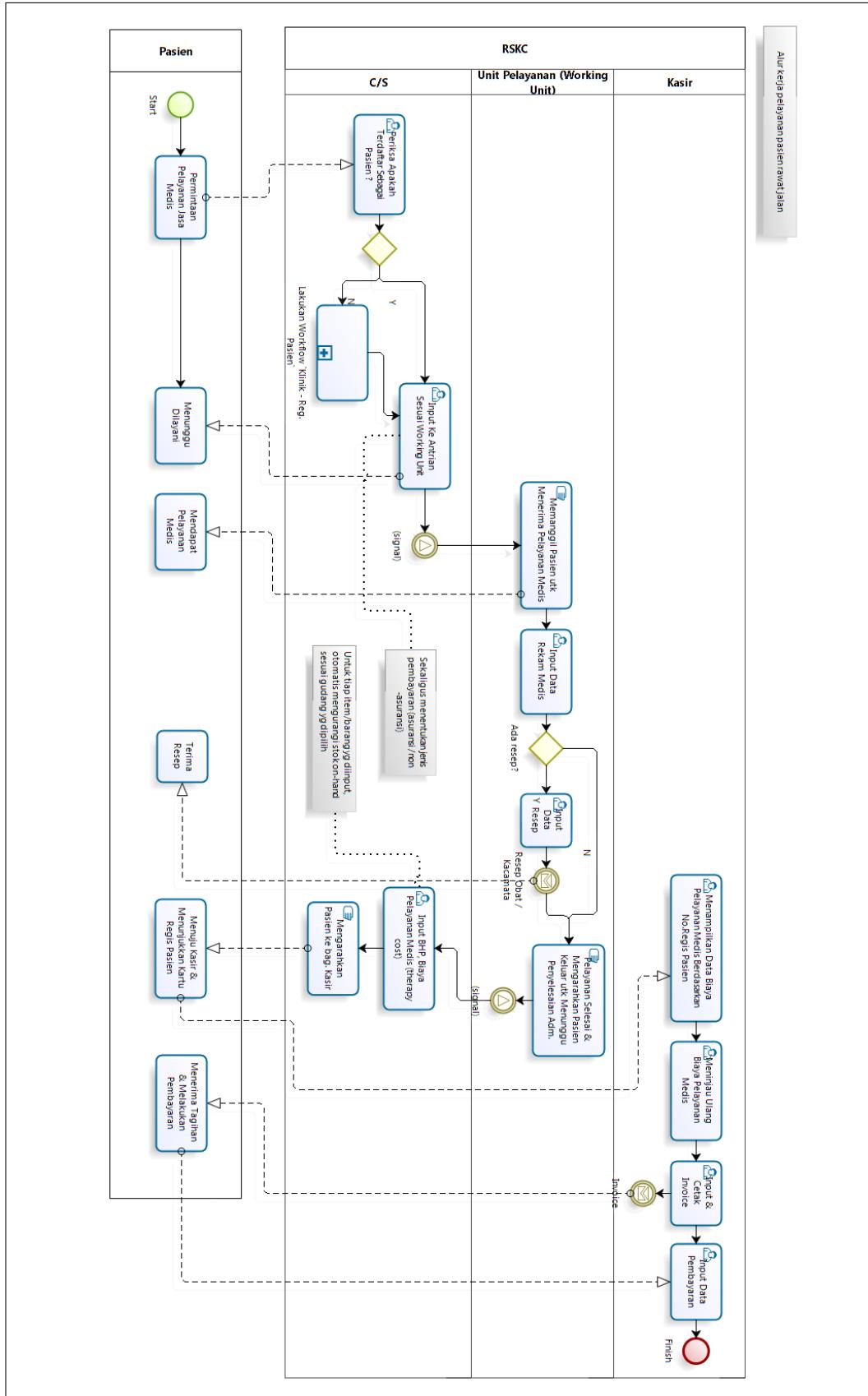
Pasien kemudian akan ditangani oleh dokter yang bertugas di unit medis tersebut. Proses penanganan pasien dimulai dari konsultasi keluhan, pemeriksaan penunjang, dan diagnosis penyakit.

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:

### III. ANALISIS DAN PERANCANGAN SISTEM



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.

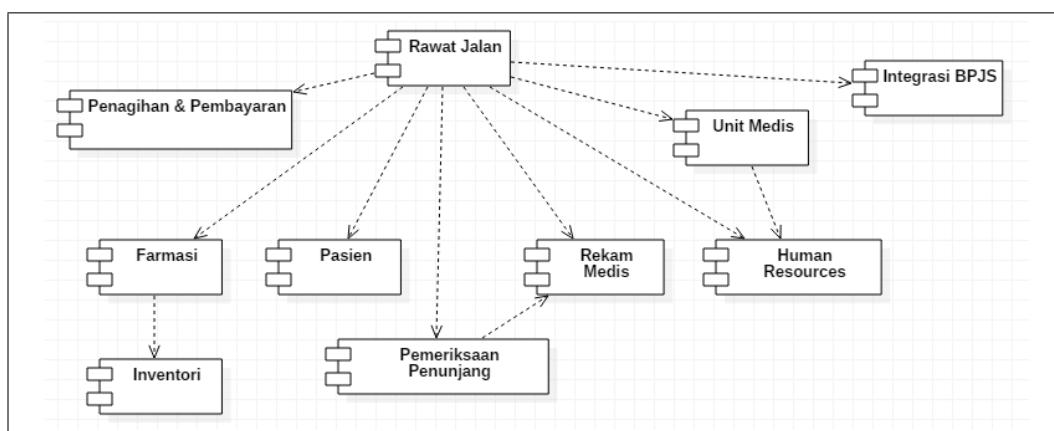
1. **Modul Pasien.** Modul pasien berfungsi untuk menyimpan, memperbarui, pencarian, dan menampilkan data pasien. Data pasien juga meliputi detail lengkap dari data keluarga atau kerabat yang menjadi penanggung jawab pasien. Kelas dari modul pasien adalah pasien itu sendiri.
2. **Modul Human Resource.** *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 Farmasi.** Modul farmasi bertugas untuk mengatur dan menyimpan obat-obatan. Modul farmasi menerima permintaan obat dari bagian IGD, rawat inap, rawat jalan, dan juga penjualan umum. Dalam aplikasi Apertura, modul ini juga meliputi fungsi dari apotek. Untuk IGD, rawat inap, dan rawat jalan pasti akan memiliki resep, namun untuk penjualan obat umum dapat menggunakan resep rujukan maupun tidak.
6. **Modul Inventori.** Inventori dibagi menjadi 2 tipe, yaitu barang dan jasa. Barang meliputi obat, alat kesehatan (alkes), dan barang selain obat dan alkes.

### III. ANALISIS DAN PERANCANGAN SISTEM

Adapun jasa yang dimaksud adalah jasa pelayanan kesehatan yang disediakan rumah sakit. Inventori berkaitan langsung dengan modul farmasi, karena modul farmasi membutuhkan data barang yang dijual.

7. **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. 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. Aplikasi Apertura diimplementasikan untuk diakses dalam 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 Kekurangan Arsitektur Monolitik

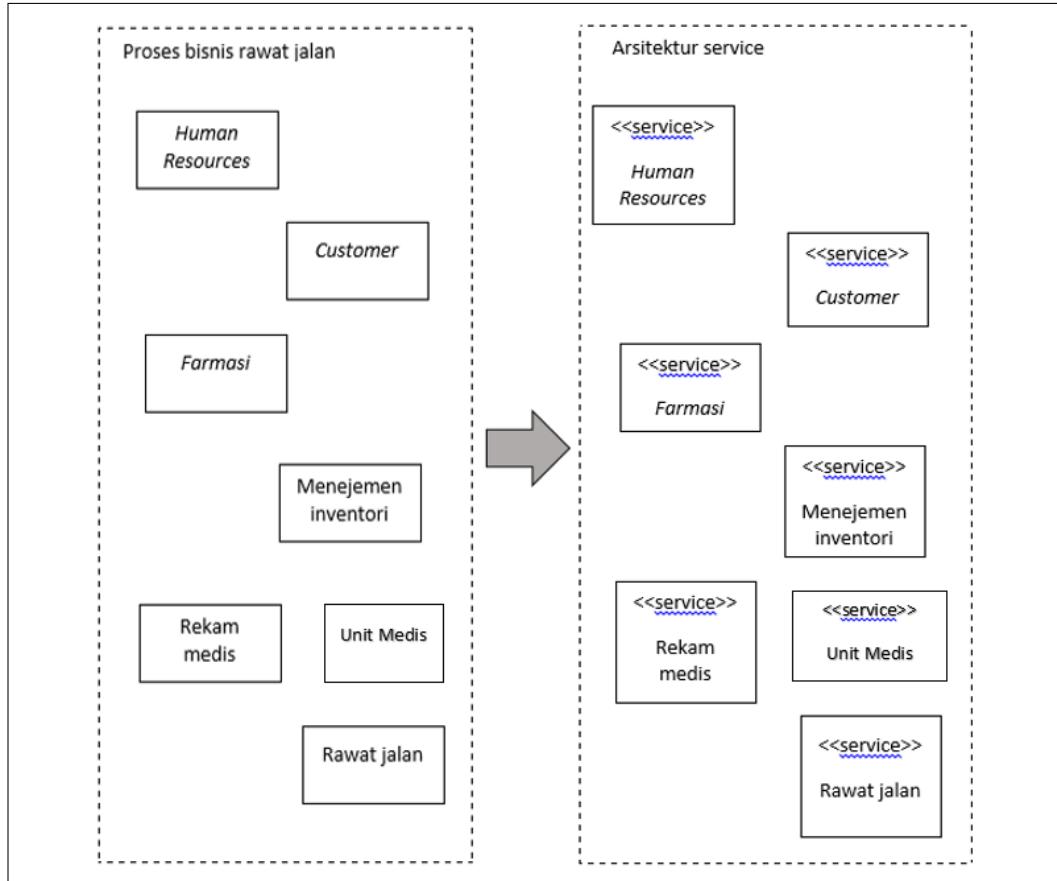
Kelemahan dari sistem Apertura sebenarnya berkaitan dengan kelemahan dari arsitektur monolitik itu sendiri. Berdasarkan analisis yang dilakukan, kelemahan dari arsitektur monolitik itu sendiri adalah:

1. Tingkat *coupling* yang terlalu tinggi. Semua modul dalam sistem saling terikat, mengakibatkan kendala dalam proses pengembangan sistem. Perubahan dalam 1 buah objek harus diketahui oleh objek lain. Proses *deployment* akan terhambat karena apabila hendak melakukan tes terhadap 1 buah modul, maka keseluruhan aplikasi harus di jalankan, mengakibatkan beban hardware lebih besar dan memakan waktu lebih banyak.
2. Penyimpanan data yang terpusat pada 1 buah tempat (database) kurang baik. Apabila database mengalami *down*, maka keseluruhan modul dari aplikasi tidak bisa melakukan pekerjaannya.
3. Isu keamanan. Penyimpanan data dalam 1 buah database tidak cukup aman. Misalnya ada pihak yang tidak berwenang berhasil mengakses database, maka seluruh data aplikasi akan bocor.
4. Sistem tidak cukup baik ketika menangani banyak user. Sistem akan kewalahan apabila banyak pengguna yang melakukan request dalam satu waktu, isu yang ditimbulkan adalah performa.

#### 3.5 Perancangan Arsitektur Microservice

Dengan menggunakan pattern dekomposisi berdasarkan kemampuan bisnisnya, maka service yang akan terbentuk dapat dibayangkan berdasarkan modul bisnis itu sendiri. Dalam kasus rawat jalan yang diangkat, maka modul yang saling berelasi adalah modul *customer*, modul HR, modul inventori, modul farmasi, dan modul rawat jalan. Ke 5 modul ini akan dibuat menjadi service yang mandiri.

### III. ANALISIS DAN PERANCANGAN SISTEM



Gambar 3.4 Pembentukan service berdasarkan proses bisnis rawat jalan.

#### 3.5.1 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.

Pemodelan database juga bergantung dengan bagaimana model database itu sendiri. Apabila model database yang digunakan nanti adalah *relational database*, maka *service* tersebut akan memiliki sebuah set database yang terdiri dari beberapa schema berhubungan yang hanya bisa diakses oleh *service* tersebut. Apabila database yang digunakan adalah *non-relational database*, maka cukup sebuah *schema* tabel untuk 1 *service*.

#### 3.5.2 Pemilihan Server Basisdata

Aplikasi akan menggunakan 2 buah database untuk kegunaan yang berbeda. Database server yang pertama akan tetap sama seperti yang saat ini sistem gunakan yaitu PostgreSQL versi 9.4. Namun untuk kasus seperti rekam

medis yang berbentuk multimedia (video, audio, foto) peneliti akan menggunakan MongoDB yang dinilai mudah, cepat, dan *open source*. Walaupun database yang digunakan sama dengan yang digunakan oleh sistem sebelumnya, namun perancangan dalam database sangat berbeda terlebih setelah memilih model penyimpanan data.

#### 3.5.3 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.

#### 3.5.4 Rancangan Deployment

Model *deployment* yang akan digunakan adalah *multiple service per host*, dikarenakan lebih cocok dalam kasus penelitian yang hanya mengambil 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 menunjukkan bahwa arsitektur microservice yang baru akan lebih unggul dalam 5 point diatas apabila dibandingkan dengan monolitik.

#### 3.6.1 Uji Perbandingan Performa

Uji yang kedua adalah menunjukkan bahwa performa dari arsitektur microservice akan memberikan hasil yang lebih baik dari monolitik. Strategi pengujian yang akan dilakukan adalah dengan membuat point-point pembanding yang akan diuji, kemudian dari point tersebut akan dibuat *test-plan* masing-masing

### **III. ANALISIS DAN PERANCANGAN SISTEM**

---

yang terdiri dari sekitar 1-3 buah *test-plan*. *Test result* kemudian akan disimpan dalam dokumen berupa *matrix traceability*.

Menurut Paulo Merson (*Software Architecture* di TCU; *SOA/microservice trainer dan consultant*), 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.

Tujuan dari test performa ini adalah menunjukkan bahwa arsitektur microservice yang baru akan lebih unggul dalam 5 point diatas apabila dibandingkan dengan monolitik.

#### **3.7 Batasan Implementasi**

Adapun batasan implementasi dalam pengembangan aplikasi yaitu:

1. Server cloud yang dapat digunakan ada sebanyak 2 buah milik Apertura. Keterbatasan sumber daya menjadi hambatan dalam menambah server baru.
2. Dikarenakan *deployment* service ada yang dilakukan di server lokal (mesin pribadi), maka untuk kasus tersebut pemanggilan API dibatasi dalam 1 jaringan yang sama.

#### **3.8 Lingkungan Implementasi**

Adapun spesifikasi mesin pribadi ketika melakukan proses implementasi adalah sebagai berikut:

1. Perangkat keras: 1 buah komputer sebagai server, 1 buah komputer sebagai client.
2. Spesifikasi komputer: 64-bit prosesor dan sistem operasi.
3. OS: Windows 10 Pro.
4. Processor: Intel(R) Core i3-4150 CPU, 3.50GHz
5. Memory: 8 GB RAM.
6. Storage: 50 MB.
7. Integrated Development Environment (IDE): Netbeans atau Eclipse neon 3.
8. Java version: 1.8.0
9. DB : PostgreSQL 9.6

## **BAB IV**

### **IMPLEMENTASI DAN PENGUJIAN**

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. Lenovo Ideapad 310
2. Processor Intel R Core T i5-6200U CPU 2.30GHz-2.40GHz
3. RAM 8 GB.

Spesifikasi perangkat lunak yang digunakan untuk pengembangan sistem adalah:

1. Sistem Operasi : Windows 10 Home Single Language 64-bit.
2. Tool Pengembangan : Python 2.7.14 64-bit, Pycharm (Python IDE) 2017.2.4, Bottle 0.12.13.
3. *Library* : Scikit-learn, Gensim, IPosTagger, Sastrawi, NLTK, NumPy.

#### **4.2 Daftar *Class* dan *Method***

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

##### **4.2.1 *Class Loader***

*Class loader* merupakan *class* yang digunakan untuk mengambil data dan label dari sebuah *file* dengan *extension* .txt. Berikut ini adalah daftar *method* pada *class loader*:

## IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.1** Daftar *Method* pada *Class Loader*

<b>Variabel:</b>	
String	Filename

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	Load	-	-	Array, Array	<i>Method</i> ini digunakan untuk mengambil data teks, beserta label pada file .txt.

### 4.2.2 *Class Preprocessing*

*Class preprocessing* merupakan *class* yang digunakan untuk meminimalisir kata, mengurangi bahasa yang non-formal, dan menghapus huruf ataupun tanda baca yang tidak digunakan. Berikut ini adalah daftar *method* pada *class preprocessing*:

**Tabel 4.2** Daftar *Method* pada *Class Preprocessing*

<b>Variabel:</b>		<b>Variabel:</b>	
Stemmer (Class Lib)	stemmer	Array	data
Array (String)	question_dic	Array	label
Array (String)	negasi	Int	size
Array (String)	stopword	Array(String)	abbreviation_dic

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	case_folding	- String	- tweet	String	<i>Method</i> ini digunakan untuk mengecilkan setiap huruf pada tweet.

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	<b>Keterangan</b>
		Tipe	Variabel		
2	remove_hashtag-mention_url	- String	- tweet	String	<i>Method</i> ini digunakan untuk menghapus semua textithashtag, <i>mention</i> , URL pada <i>tweet</i> .
3	remove_punctuation	- String	- tweet	String	<i>Method</i> ini digunakan untuk menghapus semua tanda baca yang ada pada <i>tweet</i> kecuali tanda seru (!), tanda tanya (?), tanda petik (”), tanda petik tunggal (‘), dan tanda pemisah (-).
4	tokenize	- String	- tweet	Array	<i>Method</i> ini digunakan untuk memisahkan teks menjadi token-token yang terdiri dari satu kata.
5	misuse_of_word	- String	- tweet	String	<i>Method</i> ini digunakan untuk menghilangkan huruf sama yang saling bersebelahan.
6	abbreviation_word	- String	- tweet	String	<i>Method</i> ini berguna untuk mengubah kata-kata singkatan menjadi kata persamaanya.
7	stopword_removal	- String	- tweet	String	<i>Method</i> ini digunakan untuk menghilangkan kata-kata yang dianggap tidak memiliki makna.

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	<b>Keterangan</b>
		Tipe	Variabel		
8	stemming	- String	- tweet	String	<i>Method</i> ini digunakan untuk mengubah kata menjadi kata dasar.
9	get_count_max	- String	- tweet	Int, Int, Int, Int	<i>Method</i> ini digunakan untuk mendapatkan kemunculan terbanyak tanda baca tanya (?), tanda seru (!), tanda quotation (", '), dan kata kapital.
10	write_file	- Array - Array	- data - label	Array, Array	<i>Method</i> ini digunakan untuk melakukan random terhadap data dan label, kemudian menyimpan data dan label kedalam file .txt.
11	get_size	- Array	- label	Int	<i>Method</i> ini digunakan untuk menghitung jumlah data yang akan digunakan sebagai data <i>training</i> , dengan mengalikan jumlah keseluruhan data dengan 75%.
12	Fit	- Array - Array	- data - label	Int, Int, Int, Int, Array	Fitur ini digunakan untuk melakukan <i>preprocessing</i> secara keseluruhan terhadap data <i>tweet</i> , serta mengembalikan hasil kemunculan maksimal tanda baca, dan kata kapital.

## IV. IMPLEMENTASI DAN PENGUJIAN

---

### 4.2.3 Class Features

*Class features* merupakan *class* yang digunakan untuk menangani hal terkait *feature*, dimulai dari penambahan *feature set*, penyimpanan nilai fitur ke dalam *file*, pengambilan *feature* berdasarkan jenis klasifikasi. Jenis klasifikasi yang ada pada sistem ini adalah klasifikasi 4 kelas (positif, negatif, netral, sarkasme), klasifikasi 3 kelas (positif, negatif, netral), 1 kelas (sarkasme/non-sarkasme). Berikut ini adalah *method* pada *class features*:

**Tabel 4.3** Daftar *Method* pada *Class Features*

<b>Variabel:</b>			<b>Variabel:</b>
FeatureExtraction (Class)	feature_extraction	Array	IDF
Array	feature_training	Array	label_training
Topic (Class)	topic_mod		

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	get_sar_feature	<ul style="list-style-type: none"> <li>- Array</li> <li>- Array</li> </ul>	<ul style="list-style-type: none"> <li>- data</li> <li>- label</li> </ul>	Array, Array	<i>Method</i> ini digunakan untuk menghapus semua data kecuali data positif dan sarkasme.
2	get_net_feature	<ul style="list-style-type: none"> <li>- Array</li> <li>- Array</li> </ul>	<ul style="list-style-type: none"> <li>- data</li> <li>- label</li> </ul>	Array, Array	<i>Method</i> ini digunakan untuk menghapus semua data sarkasme.
3	add_non_sar_feature	<ul style="list-style-type: none"> <li>- String</li> </ul>	<ul style="list-style-type: none"> <li>- tweet</li> </ul>	Array	<i>Method</i> ini digunakan untuk mendapatkan semua nilai fitur untuk kelas positif, negatif dan netral.
4	add_sar_feature	<ul style="list-style-type: none"> <li>- String</li> </ul>	<ul style="list-style-type: none"> <li>- tweet</li> </ul>	Array	<i>Method</i> ini digunakan untuk mendapatkan semua nilai fitur untuk kelas sarkasme.

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
5	Fit	<ul style="list-style-type: none"> <li>- Array</li> <li>- String</li> </ul>	<ul style="list-style-type: none"> <li>- data</li> <li>- type</li> </ul>	Array	<p><i>Method</i> ini digunakan untuk memanggil <i>method</i> add_sar_feature dan add_non_sar_feature untuk menambahkan fitur dari data <i>training</i> berdasarkan <i>type</i>, jika <i>type</i> sama dengan sarkasme, maka akan menambahkan fitur sarkasme, dan sebaliknya.</p>
6	save_feature	<ul style="list-style-type: none"> <li>- Int</li> <li>- Array</li> <li>- Array</li> </ul>	<ul style="list-style-type: none"> <li>- n\_classify</li> <li>- temp\_train</li> <li>- temp\_test</li> </ul>	-	<p><i>Method</i> ini digunakan untuk menyimpan hasil fitur ekstraksi data <i>training</i> dan labelnya ke dalam file .txt.</p>

## IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	<b>Keterangan</b>
		Tipe	Variabel		
7	get_train_feature	<ul style="list-style-type: none"> <li>- Int</li> <li>- Array</li> <li>- Array</li> <li>- Array</li> <li>- Array</li> </ul>	<ul style="list-style-type: none"> <li>- n_classify</li> <li>- data_train</li> <li>- label_train</li> <li>- data_test</li> <li>- label_test</li> </ul>	<ul style="list-style-type: none"> <li>Array,</li> <li>Array,</li> <li>Array,</li> <li>Array</li> </ul>	<i>Method</i> ini digunakan untuk memisahkan dan mengekstraksi fitur dari data yang akan digunakan menjadi dua, yaitu data untuk fitur ekstraksi non-sarkasme, dan data untuk fitur ekstraksi sarkasme.
8	get_test_feature	<ul style="list-style-type: none"> <li>- Int</li> <li>- Array</li> <li>- Array</li> </ul>	<ul style="list-style-type: none"> <li>- n_classify</li> <li>- data_test</li> <li>- label_test</li> </ul>	<ul style="list-style-type: none"> <li>Array,</li> <li>Array</li> </ul>	<i>Method</i> ini digunakan untuk menghapus data sarkasme pada klasifikasi tiga kelas, dan menghapus data negatif dan netral pada klasifikasi 1 kelas.

### 4.2.4 *Class FeatureExtraction*

*Class FeatureExtraction* merupakan *class* yang digunakan untuk mengekstraksi atau mengambil nilai dari sebuah *tweet* yang akan digunakan sebagai fitur. Berikut ini adalah *method* pada *class FeatureExtraction*:

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.4** Daftar *Method* pada *Class FeatureExtraction*

<b>Variabel:</b>		<b>Variabel:</b>	
Preprocessing (Class Lib)	preprocess	Int	max_qout
SentimentExtraction (Class)	sentiments	Int	max_cap
Tagger (Class Lib)	tagger	Int	max_excl
Array (String)	interjection_dic	Int	max_quest

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	unigram	- Array - String - Array	- features - tweet - IDF	Array	<i>Method</i> ini digunakan untuk menghitung kemunculan kata pada teks, dan mengembalikan nilai fitur kata yang sudah dihitung menggunakan TF-IDF
2	tagging	- String	- tweet	String	<i>Method</i> ini digunakan untuk melakukan <i>tagging</i> terhadap setiap kata, sebagai contoh "mementingkan/VBT"

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
3	part_of_speech	- Array - String	- features - tweet	Array	Method ini digunakan untuk menerima <i>tweet</i> yang sudah diberi <i>tag</i> , dan akan menghitung kemunculan <i>tag</i> seperti kemunculan kata benda, kata sifat, kata kerja, kata keterangan dan kata negasi.
4	sentiment_score	- Array - String	- features - tweet	Array	<i>Method</i> ini digunakan untuk menerima parameter <i>tweet</i> yang sudah diberi <i>tag</i> , kemudian melakukan <i>scoring</i> sentimen terhadap <i>tweet</i> .
5	punctuation_based	- Array - String - String	- features - tweet - type	Array	<i>Method</i> ini digunakan untuk menghitung kemunculan tanda tanya (?), tanda seru (!), tanda petik (") dan tanda petik tunggal (''). Setiap kemunculan akan dibagi jumlah kemunculan terbanyak pada data <i>training</i> .

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
6	capitalization	- Array - String	- features - tweet	Array	<i>Method</i> ini digunakan untuk menghitung kemunculan kata kapital kemudian jumlah kemunculan dibagi jumlah kemunculan terbanyak pada data <i>training</i> .
7	stopword_removal	- String	- tweet	String	<i>Method</i> ini digunakan untuk menghilangkan kata-kata yang dianggap tidak memiliki makna.
8	Topic	- Array - String - Class	- features - tweet - topic_mod	Array	<i>Method</i> ini digunakan untuk menerima <i>class topic</i> yang sudah dilatih pada data <i>training</i> , dan digunakan untuk mendapatkan topik dari <i>tweet</i> .
9	interjection_word	- Array - String	- features - tweet	Array	<i>Method</i> ini digunakan untuk menghitung jumlah kemunculan kata <i>interjection</i> seperti, "wow", "wahh".
10	question_word	- Array - String	- features - tweet	Array	<i>Method</i> ini digunakan untuk memberikan nilai fitur kata tanya sebagai 1 ( <i>true</i> ) atau 0 ( <i>false</i> ), jika terdapat kata tanya pada <i>tweet</i> .

## IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
11	Idf	- Array - Array	- data - label	Array	<i>Method</i> ini digunakan untuk menghitung nilai IDF dari masukan data <i>training</i> .

### 4.2.5 Class SentimentExtraction

*Class* SentimentExtraction merupakan *class* yang digunakan untuk mendapatkan nilai sentimen positif dan negatif dari sebuah *tweet*. Berikut ini adalah *method* pada *class* SentimentExtraction:

**Tabel 4.5** Daftar *Method* pada *Class* SentimentExtraction

Variabel:	Variabel:		
Defaultdict (Collection)	senti_score	Negation (class)	negasi
Preprocessing (Class)	preprocess		

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	score_sentence	- Array	- tweet	Array [][]	<i>Method</i> ini akan menerima masukan array <i>tweet</i> yang sudah diberi <i>tag</i> , kemudian melakukan iterasi setiap kata, dan memanggil <i>method</i> score_word untuk mendapatkan nilai sentimen positif dan negatif.
2	score_word	- String - String	- word - tag	Array [][]	<i>Method</i> ini akan memberi nilai sentimen positif dan negatif berdasarkan kata dan <i>tag</i> dari kata.

## IV. IMPLEMENTASI DAN PENGUJIAN

### 4.2.6 Class Negation

*Class Negation* merupakan *class* yang digunakan untuk mengatasi kata negasi yang terdapat pada *tweet*. Berikut ini adalah *method* pada *class Negation*:

**Tabel 4.6** Daftar *Method* pada *Class Negation*

<b>Variabel:</b>	
Preprocessing (Class)	preprocess

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	<b>Keterangan</b>
		Tipe	Variabel		
1	negation_handling	- String - String - Array [][], - Bool	- prev - pprev - score - change	Array [], Bool	<i>Method</i> ini akan digunakan pada <i>sentiment score</i> , untuk melakukan pengecekan kata negasi, jika terdapat kata negasi maka nilai dari kata setelah kata negasi akan ditambah dan dikali dua.

### 4.2.7 Class Topic

*Class Topic* merupakan *class* yang digunakan untuk melakukan *topic modelling* yang akan dilatih dari data *training*. Berikut ini adalah *method* pada *class Topic*:

**Tabel 4.7** Daftar *Method* pada *Class Topic*

<b>Variabel:</b>		<b>Variabel:</b>	
Int	nbtopic	String	alpha
Preprocessing (Class)	preprocess	LdaModel (Class Lib)	lda
Corpora (Class Lib)	dictionary		

## IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	Fit	- Array	- tweet	Array	<i>Method</i> ini akan digunakan untuk melakukan <i>topic modelling</i> dengan <i>library</i> LDA dari gensim.
2	transform	- String	- tweet	Array	<i>Method</i> ini akan digunakan untuk menadapatkan topik dari sebuah <i>tweet</i> .

### 4.2.8 Class SVM

*Class SVM* merupakan *class* yang digunakan untuk melakukan *training* pada data *training* dan mendapatkan hasil dari klasifikasi pada data *testing*. Berikut ini adalah *method* pada *class SVM*:

**Tabel 4.8** Daftar *Method* pada *Class SVM*

Variabel:		Variabel:	
Float	c	Array	alphas
Float	tol	Array	bias
Int	max_passes	FMeasure (class)	f_measure

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	smo	- Array - Array	- features - label	Array, Float	<i>Method</i> ini akan digunakan untuk mendapatkan nilai alpha dan bias, dari data <i>training</i> .

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

No	Method	Input		Output	Keterangan
		Tipe	Variabel		
2	error	- Array - Array - Array - Float - Int	- feature - label - alpha - bias - i	Float	<i>Method</i> ini akan digunakan untuk mendapatkan nilai <i>error</i> yang akan digunakan pada method smo.
3	classification	- Array - Float - Array - Array - Array - Array - Array	- alpha - b - feature_train - feature_test - label_train - label_test - model_type	Array, Array, Array, Array	<i>Method</i> ini untuk menghitung hasil prediksi menggunakan persamaan SVM, mengembalikan nilai <i>f-measure</i> , akurasi dari setiap metode klasifikasi yang digunakan, dan hasil prediksi.
4	f_measure_sarkasme	- Array - Array	- value_prediction - label_actual	Float	<i>Method</i> ini digunakan untuk mengubah nilai value_prediction menjadi nilai 1 atau -1, khusus klasifikasi 1 kelas.

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	<b>Keterangan</b>
		Tipe	Variabel		
5	direct_method	- Array - Array	- value_prediction - label_actual	Array, Array, Array	<i>Method</i> ini digunakan untuk melakukan klasifikasi dengan <i>direct method</i> . Keluaran dari <i>method</i> ini adalah <i>f-measure</i> dan prediksi.
6	levelled_method	- Array - Array	- value_prediction - label_actual	Array, Array, Array	<i>Method</i> ini digunakan untuk melakukan klasifikasi dengan <i>levelled method</i> . Keluaran dari <i>method</i> ini adalah <i>f-measure</i> dan prediksi.
7	convert_label	- Array - String	- label - type	Array	<i>Method</i> ini digunakan untuk mengubah label dari data sesuai dengan kelas yang akan melalui proses <i>training</i> .
8	get_data_label	- String	- prediksi	String	<i>Method</i> ini digunakan untuk mendapatkan nilai label seperti semula. Sebagai contoh, jika prediksi "sarkasme" maka akan diubah kembali menjadi 2, untuk melalui proses perhitungan akurasi.

## IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
9	save_alpha_bias	- Int - Array - Float	- n_classify - alpha - bias	-	<i>Method</i> ini untuk menyimpan nilai alpha dan bias ke dalam file.

### 4.2.9 Class FMeasure

*Class* FMeasure merupakan *class* yang digunakan untuk mendapatkan akurasi dari klasifikasi data *testing*. Berikut ini adalah *method* pada *class* FMeasure:

**Tabel 4.9** Daftar *Method* pada *Class* FMeasure

Variabel:		Variabel:	
Float	c	Array	alphas
Float	tol	Array	bias
Int	max_passes	FMeasure (class)	f_measure

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	confusion_matrix	- Array - Array	- label_actual - label_prediction	Array, Array, Array	<i>Method</i> ini digunakan untuk mendapatkan nilai <i>True Positive</i> , <i>False Positive</i> dan <i>False Negative</i> dari label prediksi dan label sebenarnya.

## IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
2	f_measure_all			Array	<i>Method</i> ini digunakan untuk menghitung f_measure dari setiap kelas yang ada.
3	precision_recall			Float	<i>Method</i> ini untuk mendapatkan nilai precision dan recall berdasarkan true positive, false positive dan false negative yang sudah didapatkan.
4	accuracy			Float	<i>Method</i> ini digunakan untuk mendapatkan f-measure berdasarkan nilai precision dan recall.

### 4.2.10 Class Learning

*Class* Learning merupakan *class* yang digunakan untuk melakukan pemodelan pada data *training* dan klasifikasi pada data *testing* dimulai dari pengambil data menggunakan *class* Loader, melakukan *preprocessing* menggunakan *class* Preprocessing, mendapatkan *feature set* menggunakan *class* Feature hingga melakukan klasifikasi menggunakan *class* SVM. Berikut ini adalah *method* pada *class* Learning:

**Tabel 4.10** Daftar *Method* pada *Class FMeasure*

<b>Variabel:</b>	<b>Variabel:</b>		
Loader (Class)	load	Preprocessing (Class)	preprocess
Features (Class)	fitur	SVM (Class)	svm_classifier

## IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	get_model_type	- Int	- n_classify	Array	<i>Method</i> ini akan digunakan untuk mengembalikan jenis klasifikasi yang akan dilakukan, jika n_classify=0, maka akan dilakukan klasifikasi 4 kelas yaitu positif, negatif, netral dan sarkasme. Jika n_classify=1, akan melakukan klasifikasi tanpa sarkasme. Jika n_classify=2, akan melakukan klasifikasi sarkasme atau non-sarkasme.
2	model	- String	- file name	-	<i>Method</i> ini berguna untuk melakukan <i>learning</i> terhadap data <i>training</i> dan melakukan klasifikasi terhadap data <i>testing</i> .

### 4.2.11 Class Main

*Class Main* merupakan *class* yang digunakan untuk menjalankan sistem berupa *website* pada *localhost*. Berikut ini adalah *method* pada *class Main*:

**Tabel 4.11** Daftar *Method* pada *Class FMeasure*

<b>Variabel:</b>	
Learning (Class)	learn

## IV. IMPLEMENTASI DAN PENGUJIAN

---

No	<i>Method</i>	<i>Input</i>		<i>Output</i>	Keterangan
		Tipe	Variabel		
1	classify	- String	- teks	String	<i>Method</i> ini untuk mengembalikan hasil klasifikasi dari teks masukan kepada UI.
2	upload	-	-	Array, Array, Array, Array, Array, Array, Array	<i>Method</i> ini digunakan untuk menerima <i>file</i> yang diisi pada <i>form</i> masukan <i>file</i> pada website, kemudian melakukan <i>training</i> pada <i>file</i> tersebut. Hasil akurasi dan data yang digunakan untuk <i>training</i> dan <i>testing</i> akan ditampilkan pada tampilan.
3	home	-	-	-	<i>Method</i> ini untuk melakukan <i>redirect</i> ke halaman awal website.

### 4.3 Implementasi Perangkat Lunak

Pada bagian ini akan dijelaskan mengenai implementasi aplikasi analisis sentimen dimulai dari pengambilan data, *text preprocessing*, *feature extraction*, hingga klasifikasi SVM dengan SMO.

#### 4.3.1 Implementasi Pengambilan Data

Pada bagian ini akan dilakukan proses pengambilan data dari *file* ekstensi .txt. Berikut ini adalah proses yang akan dilakukan dalam pengambilan data:

1. Memilih data *file* dengan ekstensi .txt yang berisi data dan labelnya yang dipisahkan dengan *tab* (\t)
2. Membuka *file* dengan *method* pada Python, yaitu `open(filename)`, kemudian memisahkan data dan label berdasarkan *tab* (\t) dengan *method* pada Python `reader(file, delimiter = "\t")`.
3. Melakukan *looping* atau pengulangan pada data yang sudah dipisahkan berdasarkan *tab*, dan menyimpan data beserta labelnya ke dalam variabel dengan tipe array.

#### 4.3.2 Implementasi *Text Preprocessing*

Sebelum melakukan pengambilan data beserta labelnya, setiap data akan melalui proses *text preprocessing* dengan melakukan iterasi atau pengulangan pada data yang sudah disimpan pada array. *Text preprocessing* yang dilakukan secara umum adalah *remove hashtag*, *URL*, *mention*, *remove punctuation*, *tokenize*, *stopword removal*, yang lainnya akan dijalankan saat penambahan fitur. Sebagai contoh fitur *part of speech* hanya akan melakukan *case folding*. Berikut ini merupakan proses yang akan dilakukan dalam *text preprocessing*:

##### 1. *Case Folding*

Mengubah setiap huruf pada *tweet* menjadi huruf kecil dengan *method* variabel.lower().

##### 2. *Remove Hashtag, URL, Mention*

Menghapus semua *hashtag*, *URL*, *mention* pada *tweet* dengan *method* pada *library* Tweet-preprocessor, yaitu clean(*tweet*).

##### 3. *Remove Punctuation*

- 1 Menginisialisasikan tanda baca yang ada ke sebuah variabel dengan tipe array dengan *method* pada Python, yaitu set(string.punctuation)
- 2 Menghapus semua tanda baca kecuali tanda baca tanya (?), tanda seru (!), tanda petik(“, ’) dan tanda pemisah (-) dengan melakukan iterasi setiap *character* pada *tweet*, dan menghapus kemunculan tanda baca yang terdapat pada variabel tanda baca pada tahap 1.

##### 4. *Tokenization*

Melakukan *tokenize* pada *tweet* dengan *method* pada *library* NLTK, yaitu word\_tokenize(*tweet*).

##### 5. *Misuse of Word*

Menggabungkan setiap huruf yang sama dan bersebelahan, dengan melakukan iterasi setiap token yang dihasilkan pada tahap *tokenization*, dengan *method* itertools.groupby(token).

##### 6. *Abbreviation Word*

- 1 Menginisialisasi kamus kata singkatan yang dibuat secara manual pada *file* ekstensi .txt ke dalam variabel *abbreviation.dic* tipe array.
- 2 Melakukan iterasi setiap token dan menggantinya dengan persamaannya jika terdapat pada variabel kamus kata singkatan.

## IV. IMPLEMENTASI DAN PENGUJIAN

---

### 7. Stopword Removal

- 1 Menginisialisasi kamus kata *stopword* pada *file* ekstensi .txt ke dalam variabel *stopword* tipe array.
- 2 Melakukan iterasi setiap token dan menghapus token tersebut jika terdapat pada variabel *stopword*.

### 8. Stemming

Mengubah kata menjadi bentuk kata dasarnya dengan *method* pada *library* sastrawi, yaitu *stem(tweet)*.

#### 4.3.3 Implementasi *Feature Extraction*

Setelah *text preprocessing*, akan dilakukan *feature extraction* untuk mendapatkan nilai fitur dari setiap teks. Berikut ini merupakan proses yang akan dilakukan dalam *feature extraction*:

##### 1. Unigram

- 1 Melakukan *case folding* terhadap *tweet*, *stemming*, *misuse of word*, *tokenize*.
- 2 Menghitung jumlah kemunculan kata pada *tweet* dan disimpan ke dalam variabel tipe object ({}).
- 3 Menghitung nilai TF setiap kata dan dikalikan dengan IDF katanya.
- 4 Hasil fitur akan disimpan dalam variabel *features* tipe object ({}).

##### 2. Part of Speech

- 1 Melakukan *case folding* terhadap *tweet*.
- 2 Melakukan *tagging* terhadap *tweet* dengan *method* pada *library* IPosTagger, yaitu *taggingStr(tweet)*
- 3 Hasil *tweet* yang sudah diberi *tag* akan dilakukan perhitungan kemunculan setiap *tag*.
- 4 Hasil fitur akan disimpan dalam variabel *features* tipe object ({}).

##### 3. Sentiment Score

- 1 Menginisialisasi variabel *senti\_score* dengan tipe *defaultdict* untuk menyimpan kata, *tag* beserta nilai sentimen positif dan negatifnya.
- 2 Melakukan *case folding* terhadap *tweet*.
- 3 Melakukan *tagging* terhadap *tweet* dengan *method* pada *library* IPosTagger, yaitu *taggingStr(tweet)*
- 4 Hasil *tweet* yang sudah diberi *tag* akan dicek ke dalam variabel *senti\_score* untuk mendapatkan nilai sentimennya.
- 5 Hasil fitur akan disimpan dalam variabel *features* tipe object ({}).

## IV. IMPLEMENTASI DAN PENGUJIAN

---

### 4. *Punctuation Based*

- 1 Menghitung kemunculan tanda baca tanya (?), tanda seru (!) dan tanda petik (‘, ’) dengan *method* pada Python, yaitu `count(tanda_baca)`.
- 2 Hasil fitur akan disimpan dalam variabel features tipe object ({}).

### 5. *Capitalization*

- 1 Menghitung kemunculan kata kapital, dan membagi jumlah kemunculan kapital dengan kemunculan maksimal kata kapital pada data *training*.
- 2 Hasil fitur akan disimpan dalam variabel features tipe object ({}).

### 6. *Topic*

- 1 Melakukan *training topic modelling* LDA dengan *method* pada *library gensim*, yaitu `LdaModel(corpus, dictionary, jumlah_topik, alpha)`. Menyimpan model dan *dictionary* kedalam *file pickle* dengan *method save(filename)*. Model dan *dictionary* disimpan ke dalam variabel `lda` dan `dictionary`.
- 2 Setelah mendapatkan model *topic*, lakukan *case\_folding, stemming, misuse of word, tokenize* pada *tweet*.
- 3 Melakukan perhitungan kemunculan kata dengan *method* pada *gensim*, yaitu `doc2bow(token)` dan menyimpan hasilnya ke dalam variabel `corpus_sentence`.
- 4 Mendapatkan probabilitas *tweet* terhadap topik yang ada dengan memanggil model LDA, yaitu `lda[corpus_sentence]`.
- 5 Hasil fitur akan disimpan dalam variabel features tipe object ({}).

### 7. *Interjection*

- 1 Melakukan *case folding, misuse of word, tokenize*.
- 2 Melakukan iterasi dan menghitung kemunculan kata interjeksi
- 3 Hasil fitur akan disimpan dalam variabel features tipe object ({}).

### 8. *Question Word*

- 1 Melakukan *case folding, misuse of word, tokenize*.
- 2 Melakukan iterasi dan menghitung kemunculan kata tanya
- 3 Hasil fitur akan disimpan dalam variabel features tipe object ({}).

### 9. *TF-IDF*

- 1 Melakukan *case\_folding*, menghapus semua kata negasi pada variabel array, dan kata tanya pada variabel array. Setelah itu *stemming, misuse of word, tokenize*.
- 2 Menghitung kemunculan kata pada sejumlah data *training*.
- 3 Menghitung nilai IDF yang disimpan pada variabel tipe `defaultdict`.

Semua hasil fitur ekstraksi pada data *training* akan disimpan ke dalam file *fitur\_training.pickle* dengan *method* pada *library numpy*, yaitu *numpy.save(filename)*.

### 4.3.4 Implementasi SVM dengan SMO

Setelah melalui proses *feature extraction*, selanjutnya melakukan proses *learning* dengan *Simplified Sequential Minimal Optimization* (SMO) dan klasifikasi menggunakan SVM linear. Berikut ini merupakan proses yang dilakukan dalam *learning* dan klasifikasi:

1. Menggunakan hasil fitur ekstraksi data *training* untuk mencari nilai alpha dan bias.
2. Mengubah setiap label menjadi +1 atau -1 sesuai dengan model yang ingin dibuat, yaitu *label\_train['all']* untuk label model positif, negatif, netral, dan *label\_train['sar']* untuk label model sarkasme.
3. Menghitung nilai alpha dan bias untuk setiap model menggunakan SMO. Hasil dari perhitungan alpha dan bias akan disimpan kedalam *file pickle* dengan metode *numpy.save(filename)*.
4. Pada tahap testing akan melakukan tahap 1 sampai dengan 2, kemudian menghitung nilai  $f(x)$  menggunakan alpha dan bias pada tahap 3 berdasarkan masing-masing model.
5. Pengklasifikasian pada data *testing* akan dilakukan dengan 2 macam cara klasifikasi, yaitu *direct method* dan *levelled method*.
6. Pengecekan pada *direct method* akan melakukan perhitungan  $f(x)$  pada model positif, negatif, netral dan sarkasme. Kemudian hasil dari model positif, negatif, netral akan dicari nilai tertingginya. Jika nilai yang didapatkan adalah model positif, maka akan melakukan pengecekan model sarkasme. Jika nilai  $\text{sign}(f(x)) \geq 1$  maka *tweet* akan diklasifikasikan sebagai sarkasme, jika bukan maka *tweet* akan diklasifikasikan sebagai positif.
7. Pengecekan pada *levelled method* akan melakukan perhitungan  $f(x)$  pada model netral. Jika  $\text{sign}(f(x)) \geq 1$  maka akan diklasifikasikan *tweet* sebagai netral, sebaliknya termasuk kelas opini. Jika termasuk kelas opini, maka akan melakukan pengecekan pada model positif dan negatif. Perhitungan nilai  $\text{sign}(f(x))$  pada model positif dan negatif akan diambil nilai tertinggi. Jika nilai tertinggi adalah pada model positif, maka akan dilakukan pengecekan pada

model sarkasme. Jika  $\text{sign}(f(x)) \geq 1$  maka akan termasuk kelas sarkasme, sebaliknya kelas positif.

### 4.4 Pengujian

Pada bab ini, akan dilakukan berbagai pengujian untuk menentukan kombinasi fitur terbaik, perbandingan tahap *preprocessing*, menentukan parameter regularisasi SMO terbaik, dan menentukan metode klasifikasi terbaik untuk analisis sentimen.

#### 4.4.1 Pengujian Kombinasi Fitur

Pada bagian ini akan dilakukan pengujian kombinasi fitur untuk mendapatkan kombinasi fitur terbaik. Kombinasi fitur yang akan dilakukan pada pengujian ini ada dua, yaitu KF1 dan KF 2. KF1 adalah kombinasi fitur 1 pada kelas positif, negatif dan netral yang terdiri dari fitur *unigram*, *sentiment score*, *question word*, dan *punctuation based*. KF 2 adalah kombinasi fitur 2 pada kelas sarkasme yang terdiri dari fitur *unigram*, *sentiment score*, *topic*, *part of speech*, *punctuation based*, *interjection word* dan *capitalization*.

Berikut ini adalah singkatan fitur yang akan digunakan pada tabel pengujian:

1. U adalah *Unigram*
2. SS adalah *Sentiment Score*
3. QW adalah *Question Word*
4. PB adalah *Punctuation Based*
5. T adalah *Topic*
6. POS adalah *Part of Speech*
7. IW adalah *Interjection Word*
8. C adalah *Capitalization*.

Berikut ini adalah singkatan metode klasifikasi pada tabel pengujian:

1. DM adalah *Direct Method*
2. LM adalah *Levelled Method*

Berikut ini adalah singkatan dari setiap *f-measure* pada tabel pengujian:

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

1. FP\_L/FP\_D adalah *f-measure* positif *Levelled Method* dan *f-measure* positif *Direct Method*.
2. FG\_L/FG\_D adalah *f-measure* negatif *Levelled Method* dan *f-measure* negatif *Direct Method*.
3. FT\_L/FT\_D adalah *f-measure* netral *Levelled Method* dan *f-measure* netral *Direct Method*.
4. FS\_L/FS\_D adalah *f-measure* sarkasme *Levelled Method* dan *f-measure* sarkasme *Direct Method*.

Pengujian dilakukan dengan data *training* 177, data *testing* 59, dengan parameter C=5, tol=0.001, dan max\_passes=5. Pengujian ini akan mencari kombinasi fitur terbaik untuk klasifikasi 4 kelas (positif, negatif, netral, sarkasme) dan 3 kelas (positif, negatif, netral). Berikut ini adalah tabel pengujian pada kombinasi fitur 1 pada klasifikasi 4 kelas dan 3 kelas:

**Tabel 4.12** Tabel Pengujian Kombinasi Fitur 1

No / Kelas	Kombinasi fitur non-sarkasme	Kombinasi fitur sarkasme	Akurasi <i>F-Measure</i>				
			FP_L/ FP_D	FN_L/ FN_D	FT_L/ FT_D	FS_L/ FS_D	LM/ DM
1/4	KF1	KF2	0.72 / 0.71	0.71 / 0.69	0.82 / 0.83	0.24 / 0.25	0.62 / 0.62
1/3	KF1	-	0.88 / 0.81	0.75 / 0.75	0.88 / 0.83	-	0.84 / 0.80
2/4	U + PB + QW	KF2	0.61 / 0.63	0.55 / 0.53	0.73 / 0.79	0.13 / 0.13	0.51 / 0.52
2/3	U + PB + QW	-	0.63 / 0.62	0.63 / 0.59	0.91 / 0.81	-	0.72 / 0.67
3/4	U + SS + QW	KF2	0.71 / 0.71	0.65 / 0.65	0.74 / 0.74	0.27 / 0.27	0.59 / 0.59
3/3	U + SS + QW	-	0.79 / 0.71	0.71 / 0.71	0.78 / 0.74	-	0.76 / 0.72
4/4	U + SS + PB	KF2	0.70 / 0.74	0.73 / 0.72	0.75 / 0.83	0.25 / 0.29	0.61 / 0.65

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.12** Tabel Pengujian Kombinasi Fitur 1

No / Kelas	Kombinasi fitur non-sarkasme	Kombinasi fitur sarkasme	Akurasi F-Measure				
			FP_L/ FP_D	FN_L/ FN_D	FT_L/ FT_D	FS_L/ FS_D	LM/ DM
4/3	U + SS + PB	-	0.82 / 0.79	0.81 / 0.77	0.88 / 0.83	-	0.84 / 0.80
5/4	U + SS	KF2	0.63 / 0.63	0.74 / 0.79	0.65 / 0.72	0.29 / 0.29	0.58 / 0.61
5/3	U + SS	-	0.67 / 0.67	0.74 / 0.76	0.69 / 0.72	-	0.70 / 0.72
6/4	U + PB	KF2	0.69 / 0.69	0.63 / 0.65	0.76 / 0.78	0.24 / 0.13	0.58 / 0.56
6/3	U + PB	-	0.58 / 0.6	0.67 / 0.67	0.85 / 0.88	-	0.70 / 0.72
7/4	U + QW	KF2	0.72 / 0.72	0.53 / 0.57	0.74 / 0.70	0.14 / 0.14	0.53 / 0.53
7/3	U + QW	-	0.65 / 0.60	0.59 / 0.61	0.80 / 0.76	-	0.68 / 0.66
8/4	SS + PB + QW	KF2	0.63 / 0.63	0.44 / 0.44	0.81 / 0.81	0.19 / 0.19	0.52 / 0.52
8/3	SS + PB + QW	-	0.63 / 0.63	0.63 / 0.59	0.87 / 0.84	-	0.71 / 0.69

Kolom "No/Kelas" pada pengujian tabel di atas, menunjukkan nomor urut pengujian dan klasifikasi kelasnya. Sebagai contoh, 1/4 adalah nomor urut pengujian ke-1 dengan klasifikasi 4 kelas (positif, negatif, netral, sarkasme), dan 1/3 adalah nomor urut pengujian ke-1 dengan klasifikasi 3 kelas (positif, negatif, netral). Berdasarkan pengujian kombinasi fitur di atas, kombinasi fitur terbaik yang dihasilkan pada fitur non-sarkasme klasifikasi 4 kelas dan 3 kelas adalah *unigram*, *punctuation based* dan *sentiment score*. Pengujian kombinasi fitur 1 untuk klasifikasi 3 kelas tidak akan dilakukan lagi, karena kombinasi fitur 2 tidak akan digunakan untuk klasifikasi 3 kelas. Setelah mendapatkan kombinasi fitur

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

terbaik pada pengujian tabel 4.11, maka selanjutnya mencari kombinasi fitur terbaik pada kombinasi fitur 2 dengan percobaan menggunakan hasil terbaik dari kombinasi fitur 1. Berikut ini adalah tabel pengujian pada kombinasi fitur 2 pada klasifikasi 4 kelas:

**Tabel 4.13** Tabel Pengujian Kombinasi Fitur 2

No	Kombinasi fitur non-sarkasme	Kombinasi fitur sarkasme	Akurasi F-Measure				
			FP_L/ FP_D	FN_L/ FN_D	FT_L/ FT_D	FS_L/ FS_D	LM/ DM
1	U + SS + PB	KF2	0.70 / 0.74	0.73 / 0.72	0.75 / 0.83	0.25 / 0.29	0.61 / 0.65
2	U + SS + PB	T + IW + C + POS + SS + PB	0.69 / 0.71	0.73 / 0.75	0.75 / 0.83	0.22 / 0.25	0.60 / 0.64
3	U + SS + PB	U + IW + C + POS + SS + PB	0.76 / 0.78	0.73 / 0.75	0.75 / 0.83	0.25 / 0.29	0.62 / 0.66
4	U + SS + PB	U + T + IW + C + SS + PB	0.70 / 0.74	0.73 / 0.75	0.75 / 0.83	0.38 / 0.40	0.64 / 0.68
5	U + SS + PB	U + T + IW + C + PB	0.72 / 0.73	0.73 / 0.75	0.75 / 0.83	0.47 / 0.47	0.67 / 70
6	U + SS + PB	U + T + IW + C	0.75 / 0.76	0.71 / 0.71	0.79 / 0.83	0.57 / 0.57	0.70 / 0.72
7	U + SS + PB	U + T + C	0.75 / 0.76	0.71 / 0.71	0.79 / 0.83	0.57 / 0.57	0.70 / 0.72
8	U + SS + PB	U + T	0.78 / 0.77	0.71 / 0.71	0.79 / 0.83	0.46 / 0.5	0.69 / 0.70
9	U + SS + PB	U + C	0.75 / 0.76	0.71 / 0.71	0.79 / 0.83	0.57 / 0.57	0.70 / 0.72
10	U + SS + PB	U	0.74 / 0.75	0.71 / 0.71	0.79 / 0.83	0.36 / 0.36	0.65 / 0.66

Berdasarkan pengujian di atas, kombinasi fitur 2 yang terbaik adalah *unigram, topic, dan capitalization*, dengan metode klasifikasi terbaik adalah *direct method*.

#### 4.4.1.1 Analisis *Error* Fitur Non-Sarkasme

Pada bagian ini akan dilakukan analisis *error* terhadap setiap fitur yang ada pada fitur non-sarkasme, yaitu *unigram*, *punctuation based* dan *sentiment score*.

##### 1. Fitur *Unigram*

Pada bagian ini akan dilakukan analisis *error* ketika hanya menggunakan fitur *sentiment score* dan *punctuation based*, tanpa fitur *unigram*. Berikut ini adalah analisis *error* pada fitur *unigram*:

**Tabel 4.14** Analisis *Error* Fitur *Unigram* pada Klasifikasi Non-Sarkasme

KF	Teks	Fitur	Label	Prediksi
U + PB + SS	sarankan tema mading sekolah	positive_sentiment: 0.031	Netral	Netral
		negative_sentiment: 0.0		
		questionM: 0.0		
		sekolah: 0.13		
		<b>saran: 0.48</b>		
		<b>tema: 0.48</b>		
SS + PB	sarankan tema mading sekolah	<b>positive_sentiment: 0.031</b>	Netral	Positif
		negative_sentiment: 0.0		
		questionM: 0.0		

Berdasarkan hasil percobaan klasifikasi di atas, tanpa fitur *unigram*, teks akan selalu diklasifikasikan berdasarkan nilai sentimen. Hal ini terjadi karena tidak ada fitur kata seperti "saran" dan "tema" yang termasuk kata netral pada data *training*, sehingga klasifikasi tanpa *unigram* bergantung dengan nilai sentimen dari sebuah kalimat. Sebagai contoh teks netral pada data *training* yang menggunakan kata "saran": "aku minta saran dong enaknya jualan apa ya modalnya gak terlalu banyak bisa online+tawarin temen sekolah. makasih sarannya<3".

##### 2. Fitur *Sentiment Score*

Pada bagian ini akan dilakukan analisis *error* ketika hanya menggunakan fitur *unigram* dan *punctuation based*, tanpa fitur *sentiment score*. Berikut ini adalah analisis *error* pada fitur *sentiment score*:

## IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.15** Analisis Error Fitur *Sentiment Score* pada Klasifikasi Non-Sarkasme

KF	Teks	Fitur	Label	Prediksi
U + PB + SS	ruang sekolah	questionM: 0.0	Netral	Netral
		sekolah: 0.27		
		<b>positive_sentiment: 0.034</b>		
		<b>negative_sentiment: 0.017</b>		
U + PB	ruang sekolah	questionM: 0.0	Netral	Positif
		<b>sekolah: 0.27</b>		

Berdasarkan hasil percobaan klasifikasi di atas, tanpa fitur *sentiment score* teks netral yang pendek dan tidak memiliki tanda baca tanya (?) tidak akan bisa diklasifikasikan ke kelas yang seharusnya, yaitu netral. Hal ini disebabkan oleh klasifikasi yang menjadi bergantung terhadap fitur kemunculan kata atau *unigram*. Pada percobaan di atas, teks netral diprediksi sebagai teks positif, karena cukup banyak kemunculan kata "sekolah" pada data *training* yang termasuk pada teks positif.

### 3. Fitur *Punctuation Based*

Pada bagian ini akan dilakukan analisis *error* ketika hanya menggunakan fitur *unigram* dan *sentiment score*, tanpa fitur *punctuation based*. Berikut ini adalah analisis *error* pada fitur *punctuation based*:

**Tabel 4.16** Analisis Error Fitur *Punctuation Based* pada Klasifikasi Non-Sarkasme

KF	Teks	Fitur	Label	Prediksi
U + PB + SS	Sekolah mu berani terima tantangan ? keberanian mention ig	positive_sentiment: 0.69	Netral	Netral
		negative_sentiment: 0.49		
		<b>questionM: 0.2</b>		
		sekolah: 0.069		
		terima: 0.16		
U + SS	sekolah mu berani terima tantangan ? keberanian mention ig	<b>positive_sentiment: 0.69</b>	Netral	Positif
		negative_sentiment: 0.49		
		sekolah: 0.069		
		terima: 0.16		

Berdasarkan hasil percobaan klasifikasi di atas, tanpa fitur *punctuation based*, teks netral yang memiliki nilai sentimen tidak akan bisa diklasifikasikan ke

## IV. IMPLEMENTASI DAN PENGUJIAN

---

kelas yang seharusnya, yaitu netral. Hal ini terjadi karena klasifikasi menjadi sangat bergantung terhadap fitur *sentiment score*. Pada percobaan di atas, nilai sentimen positif lebih tinggi dibanding negatif, oleh karena itu teks diklasifikasikan sebagai teks positif.

### 4.4.1.2 Analisis Error Fitur Sarkasme

Pada bagian ini akan dilakukan analisis *error* terhadap setiap fitur yang ada pada fitur sarkasme, yaitu *unigram*, *topic*, dan *capitalization*.

#### 1. Fitur *Unigram*

Pada bagian ini akan dilakukan analisis *error* ketika hanya menggunakan fitur *topic* dan *capitalization*, tanpa fitur *unigram*. Berikut ini adalah analisis *error* pada fitur *unigram*:

**Tabel 4.17** Analisis Error Fitur *Unigram* pada Klasifikasi Sarkasme

KF	Teks	Fitur	Label	Prediksi
U + T + C	@Budhiheriawan @richard_errik @whytas @riri_hesria @Ria1Kartolo @AdieRinaldi @fuadpuad makin pinter aja anggota DPR #sarkasme	topic 0: 0.055	Sarkasme	Sarkasme
		topic 1: 0.059		
		topic 2: 0.036		
		<b>topic 3: 0.84</b>		
		<b>capitalization: 0.14</b>		
		<b>dpr: 0.26</b>		
		pintar: 0.74		
		<b>angota: 0.44</b>		
T + C	@Budhiheriawan @richard_errik @whytas @riri_hesria @Ria1Kartolo @AdieRinaldi @fuadpuad makin pinter aja anggota DPR #sarkasme	topic 0: 0.055	Sarkasme	Positif
		topic 1: 0.059		
		topic 2: 0.036		
		<b>topic 3: 0.84</b>		
		<b>capitalization: 0.14</b>		

Berdasarkan hasil percobaan klasifikasi di atas, tanpa fitur *unigram*, klasifikasi akan salah, karena fitur yang digunakan tidak memberikan informasi yang cukup untuk mengklasifikasikan sarkasme.

#### 2. Fitur *Topic*

Pada bagian ini akan dilakukan analisis *error* ketika hanya menggunakan fitur *unigram*, tanpa fitur *topic*. Fitur *capitalization* tidak digunakan untuk analisis

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

*error* karena pada pengujian fitur tabel 4.13 menunjukkan tidak ada perubahan akurasi saat menggunakan semua fitur terbaik sarkasme, yaitu *unigram*, *capitalization*, *topic* dan tanpa fitur *topic*. Berikut ini adalah analisis fitur *topic* pada data *training* dengan label sarkasme:

**Tabel 4.18** Analisis Fitur *Topic*

No	Teks	Topic 0	Topic 1	Topic 2	Topic 3
1	Dan besok sekolah, hell yeah ! :D #sarkasme	<b>0.88</b>	0.046	0.028	0.038
2	BPK terbaik. BI terbaik. DPR.....ter..? @Fahrihamzah ... ter...? Mandi dulu Ooom!	<b>0.93</b>	0.025	0.015	0.020
3	astaga, besar sekali hati internet provider satu ini, isinya minta maaf trus #sarkasme	<b>0.92</b>	0.032	0.019	0.027
4	Wow, abis ditinggal sesiangan, jam 8 malam internet @innovateIND @HOMElinksBSD udah nyala lagi #sarkasme	<b>0.93</b>	0.024	0.015	0.020
5	Pantesan namanya dia & papa bs hilang dari list anggota DPR kasus e-KTP Ternyata mahluk paling suci sebagai gedung DPR	0.016	<b>0.95</b>	0.010	0.014
6	Oh @fadlizon betapa beruntungnya DPR RI punya sampeyan. #sarkasme #ILC	0.036	0.038	0.023	<b>0.90</b>
7	Nilai-nilai gue angkanya gede-gede bgt ya wow #sarkasme #edisidepresi	0.036	0.040	0.023	<b>0.89</b>
8	hebat fadlizon ketemu sahabat lama sepaham dan seidiologi dari rusia	0.027	0.028	0.017	<b>0.92</b>

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.18** Analisis Fitur *Topic*

No	Teks	Topic 0	Topic 1	Topic 2	Topic 3
9	jangan salahkan film horor lokal, mereka hanya menuruti KEMAUAN dan KEMALUAN pasar #sarkasme	0.023	<b>0.93</b>	0.015	0.020
10	Owh internet connection-nya keren sekali, saat dibutuhkan sangat bisa diandalkan #sarkasme #sinis #speechless	<b>0.90</b>	0.038	0.023	0.031
11	Hari ini ga ketemu @gistaanindy ckck sekolah segede apa sih? #sarkasme	<b>0.91</b>	0.033	0.020	0.027
12	TERIMA KASIH INTERNET	<b>0.85</b>	0.058	0.036	0.049
13	Salam buat Ketua Dewan-nya ya. Dah sehatkah...?	0.043	<b>0.88</b>	0.028	0.038
14	Semoga FH dan FZ ada di DPR selamanya.. krn sepi dunia kalau gak ada mereka.. :)	0.021	<b>0.94</b>	0.013	0.018
15	film berkualitas "Mr Bean Kesurupan DEPE" jadi bangga dengan negeri ini,, #sarkasme	<b>0.93</b>	0.025	0.015	0.020
16	Sepertinya sudah masuk jam2nya nih, internet di sini luar biasa!!!!!!#sarkasme	0.05	<b>0.85</b>	0.05	0.049
17	Lg istirahat nih di rumah habis sekolah dari pagi sampe sore...!!!	<b>0.92</b>	0.032	0.019	0.027
18	Anggota DPR Pancasilais. Mereka amalkan sila ke-4. Musyawarah dan mufakat untuk membagi2 jarak atas negeri ini. #Sarkasme	0.014	0.018	<b>0.94</b>	0.015
19	Yah ga bisa liat sekolah!!!	<b>0.88</b>	0.047	0.028	0.039

#### IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.18** Analisis Fitur *Topic*

No	Teks	Topic 0	Topic 1	Topic 2	Topic 3
20	besok sekolah. Hatiku gembiraaa! #sarkasme	<b>0.88</b>	0.046	0.028	0.038
21	masuk sekolah/.	0.077	0.081	0.049	<b>0.79</b>
22	RT @fnabila: Asiiik bentar lagi sekolah!!!!!! Ga sabar!!!! #sarkasme	0.036	<b>0.90</b>	0.023	0.032
23	Super sekali internet smartfren utk streaming...muter terusss...sampe ga keliatan... Waakakakkakak #sarkasme	0.024	<b>0.93</b>	0.015	0.020
24	Anda lebih hebat lagi pak @Fahrihamzah tidak punya partai bisa jadi anggota dpr.	0.036	0.038	0.023	<b>0.90</b>
25	Minggu....les : besok...sekolah:— oke,gw suka belajar. Sukaa bgt. #sarkasme	0.021	<b>0.94</b>	0.018	0.013
26	Entar sist kalo anggarannya UDAH NAIK.	<b>0.80</b>	0.080	0.049	0.066
Total		12	8	1	5

Berdasarkan analisis fitur *topic* pada 26 data *training* dengan label sarkasme, dapat disimpulkan setiap teks sarkasme cenderung termasuk pada topik 0, 1 dan 3, dari jumlah topik yang ada adalah 4. Hasil analisis fitur *topic* di atas akan membantu dalam analisis *error* pada fitur *topic* yang akan dilakukan. Berikut ini analisis *error* fitur *topic*:

## IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.19** Analisis *Error* Fitur *Topic* pada Klasifikasi Sarkasme

KF	Teks	Fitur	Label	Prediksi
U + T	Berkaitan akses situs porno anggota dewan Badan Kehormatan DPR adakan investigasi siapa ditonton	topic 0: 0.044	Sarkasme	Sarkasme
		<b>topic 1: 0.88</b>		
		<b>topic 2: 0.02</b>		
		<b>topic 3: 0.03</b>		
		dewan: 0.16		
		angota: 0.11		
		<b>tonton: 0.18</b>		
U	Berkaitan akses situs porno anggota dewan Badan Kehormatan DPR adakan investigasi siapa ditonton	dpr: 0.06	Sarkasme	Positif
		dewan: 0.16		
		angota: 0.11		
		<b>tonton: 0.18</b>		

Berdasarkan hasil percobaan klasifikasi di atas, tanpa fitur *topic*, klasifikasi akan salah, jika terdapat kata yang kemunculannya sering terdapat pada kelas selain kelas sebenarnya. Pada kasus penggunaan di atas kata "tonton" menjadi fitur yang paling menentukan kelas dari sebuah teks, sehingga ketika fitur *topic* tidak digunakan klasifikasi sarkasme menjadi salah. Sebagai contoh teks positif pada data *training* yang menggunakan kata "tonton", yaitu "Jadi support film Indonesia, tapi juga harus jeli apa yang kita tonton ... hmm, aulion bener nih."

### 3. Fitur *Capitalization*

Pada bagian ini akan dilakukan analisis *error* ketika hanya menggunakan fitur *unigram*, *topic*, dan *sentiment score*, tanpa fitur *capitalization*. Berikut ini adalah analisis *error* fitur *capitalization*:

## IV. IMPLEMENTASI DAN PENGUJIAN

---

**Tabel 4.20** Analisis *Error* Fitur *Capitalization* pada Klasifikasi Sarkasme

KF	Teks	Fitur	Label	Prediksi
U + T + C	Hampir 11 jam dekat sekolah I LOVE MY SENIOR LIFE PEEPS!	<b>topic 0: 0.46</b>	Sarkasme	Sarkasme
		topic 1: 0.060		
		topic 2: 0.036		
		topic 3: 0.043		
		<b>capitalization: 4.0</b>		
		sekolah: 0.079		
		life: 0.32		
		jam: 0.23		
		cinta: 0.27		
U + T	Hampir 11 jam dekat sekolah I LOVE MY SENIOR LIFE PEEPS!	<b>topic 0: 0.46</b>	Sarkasme	Positif
		topic 1: 0.060		
		topic 2: 0.036		
		topic 3: 0.043		
		sekolah: 0.079		
		life: 0.32		
		jam: 0.23		
		cinta: 0.27		

Berdasarkan hasil percobaan klasifikasi di atas, tanpa fitur *capitalization*, klasifikasi akan salah, jika terdapat sebuah teks yang memiliki nilai topik yang kecil. Hal ini terjadi karena nilai topik dari teks hanya 0.46, sedangkan kata "cinta" yang cenderung positif pada data *training*, hal ini yang menyebabkan klasifikasi salah ketika tidak ada fitur *capitalization*.

### 4.4.2 Pengujian Parameter pada SMO

Pada bagian ini, akan dilakukan pengujian parameter SMO yaitu, C, *tolerance*, dan *max\_passes*. Nilai parameter C yang akan digunakan untuk pengujian adalah 1, 5 dan 10. Nilai parameter *max\_passes* yang akan digunakan untuk pengujian adalah 5 dan 10. Nilai parameter tol yang akan digunakan untuk pengujian adalah 0.001 dan 0.0001. Berikut ini adalah tabel pengujian parameter SMO:

**Tabel 4.21** Pengujian Parameter SMO

Parameter			Akurasi <i>F-Measure</i>				
C	Tol	MaxPasses	FP.D	FN.D	FT.D	FS.D	DM
1	0.001	5	0.65	0.52	0.89	0.18	0.56
1	0.001	10	0.65	0.52	0.89	0.18	0.56
1	0.0001	5	0.65	0.52	0.89	0.18	0.56
1	0.0001	10	0.65	0.52	0.89	0.18	0.56
5	0.001	5	0.76	0.71	0.83	0.57	0.72
5	0.001	10	0.76	0.71	0.83	0.57	0.72
5	0.0001	5	0.76	0.71	0.83	0.57	0.72
5	0.0001	10	0.76	0.71	0.83	0.57	0.72
10	0.001	5	0.74	0.75	0.81	0.57	0.72
10	0.001	10	0.74	0.75	0.81	0.57	0.72
10	0.0001	5	0.74	0.75	0.81	0.57	0.72
10	0.0001	10	0.74	0.75	0.81	0.57	0.72

Berdasarkan pengujian parameter di atas, parameter yang menghasilkan akurasi terbaik adalah parameter C=5 dan C=10. Parameter C=10 memberikan hasil akurasi yang lebih merata pada setiap kelasnya, sedangkan pada parameter C=5, memberikan akurasi kelas positif yang lebih tinggi, namun akurasi kelas negatif menjadi rendah. Oleh karena itu akan dipilih parameter C=10 dengan *tolerance* 0.001 dan *max passes* 5. Alasan pemilihan parameter tersebut adalah semakin tinggi parameter *tolerance* dan *max passes* tidak terjadi adanya perubahan pada akurasi, selain itu alasan pemilihan parameter *tolerance* dan *max passes* yang lebih rendah adalah supaya *training* yang dilakukan menjadi lebih cepat.

#### 4.4.3 Pengujian Klasifikasi 4 kelas, 3 kelas dan 1 kelas

Pada bagian ini akan dilakukan pengujian untuk membandingkan akurasi *f-measure* jika melakukan klasifikasi 4 kelas, yaitu positif, negatif, netral, sarkasme, kemudian mengklasifikasikan 3 kelas, yaitu positif, negatif dan netral, dan yang terakhir mengklasifikasikan 1 kelas sarkasme atau non-sarkasme. Klasifikasi kelas akan menggunakan fitur yang dihasilkan pada pengujian kombinasi fitur yang telah dilakukan, yaitu *unigram*, *punctuation based* dan *sentiment score* untuk klasifikasi non-sarkasme, dan fitur *unigram*, *capitalization*, *topic* dan *sentiment score* untuk

#### **IV. IMPLEMENTASI DAN PENGUJIAN**

---

klasifikasi sarkasme. Berikut ini adalah tabel pengujian klasifikasi 4 kelas:

**Tabel 4.22** Pengujian Klasifikasi 4 Kelas (Positif, Negatif, Netral, Sarkasme)

<b>Klasifikasi</b>	<b>F-Measure</b>				
	<b>FP.D</b>	<b>FN.L</b>	<b>FT.D</b>	<b>FS.D</b>	<b>DM</b>
Klasifikasi 4 kelas	0.74	0.75	0.81	0.57	0.72

**Tabel 4.23** Pengujian Klasifikasi 3 Kelas (Positif, Negatif, Netral)

<b>Klasifikasi</b>	<b>F-Measure</b>			
	<b>FP.D</b>	<b>FN.L</b>	<b>FT.D</b>	<b>DM</b>
Klasifikasi 3 kelas	0.79	0.81	0.86	0.82

**Tabel 4.24** Pengujian Klasifikasi 1 Kelas (Sarkasme/Non-Sarkasme)

<b>Klasifikasi</b>	<b>F-Measure</b>
Klasifikasi 1 kelas (sarkasme/non-sarkasme)	0.75

Berdasarkan hasil pengujian di atas, hasil akurasi meningkat sebanyak 10% ketika hanya mengklasifikasikan 3 kelas. Dan akurasi deteksi sarkasme meningkat sebanyak 18% ketika hanya melakukan deteksi sarkasme atau non-sarkasme.

## BAB V

### PENUTUP

Bab ini berisi kesimpulan dan saran dari sistem deteksi sarkasme pada analisis sentimen media sosial.

#### 5.1 Kesimpulan

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

1. Kombinasi fitur terbaik untuk model positif, negatif dan netral adalah *unigram*, *punctuation based* dan *sentiment score*. Dan kombinasi fitur terbaik untuk model sarkasme adalah *unigram*, *topic*, dan *capitalization*.
2. Fitur *unigram* sangat berpengaruh dalam klasifikasi teks baik positif, negatif, netral, ataupun sarkasme. Fitur *punctuation based* dapat meningkatkan akurasi klasifikasi teks netral hingga 9%. Fitur *sentiment score* dapat meningkatkan akurasi hingga 7% pada teks positif, dan 10% pada teks negatif.
3. Fitur *topic* dapat meningkatkan akurasi klasifikasi sarkasme hingga 10% saat digabungkan dengan fitur *unigram*. Fitur *capitalization* dapat meningkatkan akurasi klasifikasi sarkasme hingga 7%.
4. Fitur *interjection word* tidak berpengaruh karena kemunculan *interjection word* yang jarang pada data sarkasme yang dilatih. Fitur *part of speech* menurunkan akurasi klasifikasi sarkasme, karena kemunculan kata benda, kata sifat, kata kerja, kata keterangan, dan kata negasi juga banyak terdapat pada teks-teks selain sarkasme. Fitur *sentiment score* menurunkan akurasi klasifikasi sarkasme karena teks sarkasme memiliki nilai sentimen yang selalu berlawanan dari yang terlihat dari teks. Fitur *punctuation based* menurunkan klasifikasi karena tanda baca juga terdapat banyak pada teks selain sarkasme.
5. Fitur *question word* menurunkan akurasi, karena saat digabungkan dengan fitur *punctuation based*, setiap teks positif, negatif atau sarkasme yang memiliki kata tanya dan tanda baca tanya (?) akan otomatis terkласifikasi sebagai teks netral.

6. Kombinasi parameter SMO yang terbaik adalah C=10, tol=0.001, dan max\_passes=5. Akurasi klasifikasi *direct method* lebih tinggi dibanding *levelled method*.
7. Kombinasi klasifikasi pada 4 kelas (positif, negatif, netral, sarkasme) menghasilkan akurasi sebesar 72% dan pada kombinasi klasifikasi 3 kelas (positif, negatif, netral) menghasilkan akurasi sebesar 82%. Akurasi pada klasifikasi 3 kelas tanpa data sarkasme meningkat cukup tinggi. Dan pada klasifikasi 1 kelas, yaitu sarkasme dan non-sarkasme menghasilkan 75%.
8. Klasifikasi pada 4 kelas menghasilkan akurasi yang kecil, karena teks sarkasme pada data yang digunakan terdapat teks yang tidak terlihat positif, sehingga terjadi kesalahan klasifikasi. Sebagai contoh teks sarkasme yang lebih terlihat seperti netral, yaitu: "Apa bedanya anggota DPR dengan sebuah baterai? Baterai punya sisi positif. #Sarkasme".

## 5.2 Saran

Saran dari penulis untuk pengembangan yang dilakukan untuk sistem deteksi sarkasme pada analisis sentimen adalah:

1. Menggunakan data yang lebih banyak dan kategori data yang lebih banyak, sehingga fitur *topic* dapat digunakan untuk memberikan informasi global dari sebuah teks sarkasme dengan lebih baik serta menggunakan jenis kernel RBF untuk klasifikasi.
2. Menggunakan fitur *emoticon* untuk memberikan informasi teks sarkasme.

## DAFTAR PUSTAKA

- [1] Ramadhani, R. A. A., Indriani, F., & Nugrahadi, D. T. *Comparison of Naive Bayes Smoothing Methods for Twitter Sentiment Analysis. Comparison of Naive Bayes Smoothing Methods for Twitter Sentiment Analysis.*
- [2] Tsytsarau, M., & Palpanas, T. (2012). Survey on mining subjective data on the web. *Data Mining and Knowledge Discovery*, 24(3), 478-514
- [3] Ptáček, T., Habernal, I., & Hong, J. (2014). Sarcasm detection on czech and english Twitter. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers* (pp. 213-223).
- [4] Peng, C. C., Lakis, M., & Pan, J. W. Detecting Sarcasm in Text.
- [5] Lunando, E., & Purwarianti, A. (2013, September). *Indonesian social media sentiment analysis with sarcasm detection*. In *Advanced Computer Science and Information Systems (ICACSIS), 2013 International Conference on* (pp. 195-198). IEEE.
- [6] Java, A., Song, X., Finin, T., & Tseng, B. (2007, August). Why we Twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*(pp. 56-65). ACM.
- [7] Karamizadeh, S., Abdullah, S. M., Halimi, M., Shayan, J., & javad Rajabi, M. (2014, September). Advantage and drawback of *Support Vector Machine* functionality. In *Computer, Communications, and Control Technology (I4CT), 2014 International Conference on* (pp. 63-65). IEEE.
- [8] Malouf, R. (2002, August). A comparison of algorithms for *Maximum Entropy* parameter estimation. In *proceedings of the 6th conference on Natural language learning-Volume 20* (pp. 1-7). Association for Computational Linguistics.
- [9] Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, 37(1), 51-89.
- [10] Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1), 1-167.

## DAFTAR PUSTAKA

---

- [11] Baştanlar, Y., & Özysal, M. (2014). Introduction to machine learning. *miRNomics: MicroRNA Biology and Computational Analysis*, 105-128.
- [12] Adriani, M., Asian, J., Nazief, B., Tahaghoghi, S. M., & Williams, H. E. (2007). Stemming Indonesian: A confix-stripping approach. *ACM Transactions on Asian Language Information Processing (TALIP)*, 6(4), 1-33.
- [13] Wicaksono, A. F., & Purwarianti, A. (2010, August). HMM based part-of-speech tagger for Bahasa Indonesia. In *Fourth International MALINDO Workshop, Jakarta*.
- [14] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan), 993-1022.
- [15] Shabtai, A., Moskovitch, R., Elovici, Y., & Glezer, C. (2009). Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *information security technical report*, 14(1), 16-29.
- [16] Berwick, R. (2003). An Idiot's guide to Support Vector Machines (SVMs). *Retrieved on October*, 21, 2011.
- [17] Wang, Z., & Xue, X. (2014). Multi-class *Support Vector Machine*. In *Support Vector Machines Applications* (pp. 23-48). Springer International Publishing.
- [18] CS 229, Autumn 2009 “The Simplified SMO Algorithm”.
- [19] Kwak, H., Lee, C., Park, H., & Moon, S. (2010, April). What is Twitter, a social network or a news media?. In *Proceedings of the 19th international conference on World wide web* (pp. 591-600). ACM.
- [20] Shamay-Tsoory, S. G., Tomer, R., & Aharon-Peretz, J. (2005). The neuroanatomical basis of understanding sarcasm and its relationship to social cognition. *Neuropsychology*, 19(3), 288.

## **LAMPIRAN**