

**PENERAPAN MICROSERVICE DENGAN HIERARCHICAL  
CLUSTERING UNTUK DEKOMPOSISI DARI MONOLIT  
PADA ENTERPRISE RESOURCE PLANNING**

**TUGAS AKHIR**

**Albertus Septian Angkuw  
1119002**



**INSTITUT  
TEKNOLOGI  
HARAPAN  
BANGSA**

*Veritas vos liberabit*

**PROGRAM STUDI INFORMATIKA  
INSTITUT TEKNOLOGI HARAPAN BANGSA  
BANDUNG  
2022**

**PENERAPAN MICROSERVICE DENGAN HIERARCHICAL  
CLUSTERING UNTUK DEKOMPOSISI DARI MONOLIT  
PADA ENTERPRISE RESOURCE PLANNING**

**TUGAS AKHIR**

**Diajukan sebagai salah satu syarat untuk memperoleh  
gelar sarjana dalam bidang Informatika**

**Albertus Septian Angkuw  
1119002**



**INSTITUT  
TEKNOLOGI  
HARAPAN  
BANGSA**

*Veritas vos liberabit*

**PROGRAM STUDI INFORMATIKA  
INSTITUT TEKNOLOGI HARAPAN BANGSA  
BANDUNG  
2022**

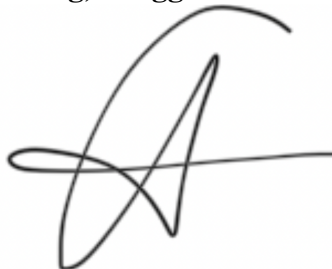
## **HALAMAN PERNYATAAN ORISINALITAS**

**Saya menyatakan bahwa Tugas Akhir yang saya susun ini  
adalah hasil karya saya sendiri.**

**Semua sumber yang dikutip maupun dirujuk  
telah saya nyatakan dengan benar.**

**Saya bersedia menerima sanksi pencabutan gelar akademik  
apabila di kemudian hari Tugas Akhir ini terbukti plagiat.**

**Bandung, Tanggal Bulan Tahun**

A handwritten signature in black ink, featuring a large, stylized capital 'A' with a horizontal line extending to the right and a loop on the left.

**Albertus Septian Angkuw  
1119002**

## **HALAMAN PENGESAHAN TUGAS AKHIR**

Tugas Akhir dengan judul:

**PENERAPAN MICROSERVICE DENGAN HIERARCHICAL CLUSTERING  
UNTUK DEKOMPOSISI DARI MONOLIT PADA ENTERPRISE RESOURCE  
PLANNING**

yang disusun oleh:

**Albertus Septian Angkuw  
1119002**

telah berhasil dipertahankan di hadapan Dewan Penguji Sidang Tugas Akhir yang  
dilaksanakan pada:

Hari / tanggal : Hari, Tanggal Bulan Tahun

Waktu : Jam (24-HOUR FORMAT, contoh 16.00 WIB) WIB

**Menyetujui**

**Pembimbing Utama:**

**Pembimbing Pendamping:**

**Hans Christian Kurniawan, S.T., M.T**

**NIK**

**...  
NIK**

## HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Institut Teknologi Harapan Bangsa, saya yang bertandatangan di bawah ini:

Nama : Nama Pengarang

NIM : NIM

Program Studi : Informatika

demikian pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Institut Teknologi Harapan Bangsa **Hak Bebas Royalti Noneksklusif** (*Non-exclusive Royalty Free Rights*) atas karya ilmiah saya yang berjudul:

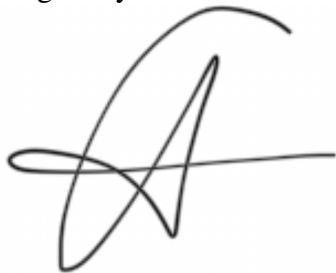
JUDUL TUGAS AKHIR

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Institut Teknologi Harapan Bangsa berhak menyimpan, mengalihmediakan, mengelola dalam pangkalan data, dan memublikasikan karya ilmiah saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Bandung, Tanggal Bulan Tahun

Yang menyatakan



Nama Pengarang

## ABSTRAK

Nama : Nama Pengarang  
Program Studi : Informatika  
Judul : Judul Tugas Akhir dalam Bahasa Indonesia

Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et, ex est mazim libris quaeque. Habeo brute vel id, inani volumus adolescens et mei, solet mediocrem te sit. At sonet dolore atomorum sit, tistique sapientem contentiones no vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque similique sed eu, nam facilisis neglegentur ex. Qui ut tistique voluptua. Ei utroque electram gubergren per. Laudem nonumes an vis, cum veniam eligendi liberavisse eu. Etiam graecis id mel. An quo rebum iracundia definitionem. At quo congrue graeco explicari. Cu eos wisi legimus patrioque. Cum iisque offendit ei. Ei eruditi lobortis pericula sea, te graeco salutatus sed, ne integre insolens mei. Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei. Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et.

Kata kunci: Sonet, dolore, atomorum, tistique, sapientem.

## **ABSTRACT**

*Name* : Nama Pengarang  
*Department* : Informatics  
*Title* : Judul Tugas Akhir dalam Bahasa Inggris

*Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et, ex est mazim libris quaeque. Habeo brute vel id, inani volumus adolescens et mei, solet mediocrem te sit. At sonet dolore atomorum sit, tistique sapientem contentiones no vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque similique sed eu, nam facilisis neglegentur ex. Qui ut tistique voluptua. Ei utroque electram gubergren per. Laudem nonumes an vis, cum veniam eligendi liberavisse eu. Etiam graecis id mel. An quo rebum iracundia definitionem. At quo congue graeco explicari. Cu eos wisi legimus patrioque. Cum iisque offendit ei. Ei eruditi lobortis pericula sea, te graeco salutatus sed, ne integre insolens mei. Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei. Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et.*

*Keywords: Sonet, dolore, atomorum, tistique, sapientem.*

## KATA PENGANTAR

Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et, ex est mazim libris quaeque. Habeo brute vel id, inani volumus adolescens et mei, solet mediocrem te sit. At sonet dolore atomorum sit, tistique sapientem contentiones no vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque similique sed eu, nam facilisis neglegentur ex. Qui ut tistique voluptua. Ei utroque electram gubergren per. Laudem nonumes an vis, cum veniam eligendi liberavisse eu. Etiam graecis id mel. An quo rebum iracundia definitionem. At quo congue graeco explicari. Cu eos wisi legimus patrioque. Cum iisque offendit ei. Ei eruditi lobortis pericula sea, te graeco salutatus sed, ne integre insolens mei. Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

Bandung, Tanggal Bulan Tahun

Hormat penulis,

A stylized, handwritten signature in black ink, consisting of a large, sweeping loop that crosses itself, followed by a horizontal line extending to the right.

Nama Pengarang



## DAFTAR ISI

<b>ABSTRAK</b>	<b>iv</b>
<b>ABSTRACT</b>	<b>v</b>
<b>KATA PENGANTAR</b>	<b>vi</b>
<b>DAFTAR ISI</b>	<b>vii</b>
<b>DAFTAR TABEL</b>	<b>x</b>
<b>DAFTAR GAMBAR</b>	<b>xi</b>
<b>DAFTAR ALGORITMA</b>	<b>xi</b>
<b>DAFTAR LAMPIRAN</b>	<b>xii</b>
<b>BAB 1 PENDAHULUAN</b>	<b>1-1</b>
1.1 Latar Belakang . . . . .	1-1
1.2 Rumusan Masalah . . . . .	1-3
1.3 Tujuan Penelitian . . . . .	1-3
1.4 Batasan Masalah . . . . .	1-3
1.5 Kontribusi Penelitian . . . . .	1-3
1.6 Metodologi Penelitian . . . . .	1-4
1.7 Sistematika Pembahasan . . . . .	1-4
<b>BAB 2 LANDASAN TEORI</b>	<b>2-1</b>
2.1 Tinjauan Pustaka . . . . .	2-1
2.1.1 Monolit . . . . .	2-1
2.1.2 <i>Microservice</i> . . . . .	2-3
2.1.3 <i>Enterprise Resource Planning</i> . . . . .	2-6
2.1.4 Analisis Kode . . . . .	2-7
2.1.5 <i>Clustering</i> . . . . .	2-9
2.1.6 Dekomposisi . . . . .	2-10
2.1.7 Teknologi dan <i>Library</i> . . . . .	2-11
2.1.7.1 <i>Docker</i> . . . . .	2-11
2.1.7.2 <i>gRPC</i> . . . . .	2-11
2.1.7.3 <i>PyCG</i> . . . . .	2-11

2.2	Tinjauan Studi . . . . .	2-11
2.3	Tinjauan Objek . . . . .	2-14
<b>BAB 3</b>	<b>ANALISIS DAN PERANCANGAN SISTEM</b>	<b>3-1</b>
3.1	Analisis Masalah . . . . .	3-1
3.2	Kerangka Pemikiran . . . . .	3-1
3.3	Urutan Proses Global . . . . .	3-3
3.3.1	Proses Clustering . . . . .	3-3
3.3.1.1	Pengambilan Source Code . . . . .	3-3
3.3.1.2	Pembuatan Call Graph . . . . .	3-3
3.3.1.3	Ilustrasi Graph . . . . .	3-3
3.3.1.4	Extrasi Call Graph . . . . .	3-3
3.3.1.5	Extrasi Depedency Module . . . . .	3-3
3.3.1.6	Extrasi Depedency Data . . . . .	3-3
3.3.1.7	Pre Procesinng . . . . .	3-3
3.3.1.8	Hierarchical Clustering . . . . .	3-3
3.3.1.9	Pemilihan Partisi . . . . .	3-4
3.3.2	Dekomposisi Monolitik ke Microservice . . . . .	3-4
3.3.2.1	Pemisahan User Interface . . . . .	3-4
3.3.2.2	Strategi Pemisahan yg dipilih . . . . .	3-4
3.3.2.3	Strategi Pemisahan database . . . . .	3-4
3.3.2.4	Komunikasi antar service . . . . .	3-4
3.3.2.5	Infrastruktur . . . . .	3-4
3.3.3	Deployment . . . . .	3-4
<b>BAB 4</b>	<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>4-1</b>
4.1	Lingkungan Implementasi . . . . .	4-1
4.1.1	Spesifikasi Perangkat Keras . . . . .	4-1
4.1.2	Spesifikasi Perangkat Lunak . . . . .	4-2
4.2	Implementasi Perangkat Lunak . . . . .	4-2
4.2.1	Implementasi <i>Class</i> . . . . .	4-3
4.2.1.1	<i>Class</i> Nama_Class_1 . . . . .	4-3
4.2.1.2	<i>Class</i> Nama_Class_2 . . . . .	4-3
4.2.2	Implementasi Numquam . . . . .	4-3
4.3	Implementasi Nama_Implementasi . . . . .	4-3
4.4	Implementasi Aplikasi . . . . .	4-3
4.5	Pengujian . . . . .	4-4
4.5.1	Pengujian Nama_Pengujian_1 . . . . .	4-4

4.5.2	Pengujian Nama_Pengujian_2 . . . . .	4-6
<b>BAB 5</b>	<b>KESIMPULAN DAN SARAN</b>	<b>5-1</b>
5.1	Kesimpulan . . . . .	5-1
5.2	Saran . . . . .	5-1
<b>BAB A</b>	<b>LAMPIRAN A</b>	<b>A-1</b>
	ASJDBAKJSDBKA . . . . .	A-1
<b>BAB B</b>	<b>DATASET HASIL KUISIONER 2</b>	<b>B-3</b>

## DAFTAR TABEL

2.1	<i>State of the Art</i>	.2-11
2.2	Komposisi dari Module pada aplikasi Odoo	.2-16
4.1	atribut pada <i>class</i> nama_class_1	4-4
4.2	Daftar <i>method</i> pada <i>class helper</i>	4-5
A-1	<i>Lorem ipsum</i>	A-1
A-1	<i>Lorem ipsum</i>	A-2
B-1	<i>Lorem ipsum</i>	B-3
B-1	<i>Lorem ipsum</i>	B-4

## DAFTAR GAMBAR

2.1	Pola dalam menyelesaikan masalah di arsitektur <i>Microservice</i> [8]	2-5
2.2	Ilustrasi Arsitektur Odoo . . . . .	2-15
2.3	Diagram Database Odoo . . . . .	2-17
3.1	Kerangka Pemikiran . . . . .	3-1
3.2	Diagram Flowchart Proses Global . . . . .	3-3
3.3	Diagram Deployment . . . . .	3-4

## **DAFTAR ALGORITMA**

<b>LAMPIRAN A</b>	<b>A-1</b>
<b>LAMPIRAN B</b>	<b>B-3</b>

## **DAFTAR LAMPIRAN**

# BAB 1 PENDAHULUAN

## 1.1 Latar Belakang

Aplikasi *Enterprise Resource Planning* (ERP) merupakan perangkat lunak yang digunakan pada perusahaan untuk menjalankan bisnisnya dimana perusahaan dapat mengotomatisasi dan mengintegrasikan sebagian besar proses bisnisnya dengan ini perusahaan bisa menghasilkan dan mengakses informasi secara langsung [1]. Selain itu diharapkan juga aplikasi memiliki skalabilitas terhadap operasi bisnis, yang diperlukan dalam jangka panjang, misalkan ketika perusahaan tumbuh dari waktu ke waktu, serta dalam waktu singkat, misalnya pada saat volume bertransaksi tinggi seperti belanja Natal [1].

Dalam membangun aplikasi ERP umumnya dibangun dengan beberapa arsitektur seperti *two-tier*, *three-tier/n-tier*, dan *Service Oriented Architecture* (SOA) dimana arsitektur ini disebarkan secara monolit. Saat ini arsitektur yang umum digunakan yaitu SOA karena dapat membantu perancangan ERP menjadi lebih terukur, andal dan fleksibel dengan memecah fungsionalitas menjadi bagian kecil yang dinamakan *service* [1].

SOA memiliki keuntungan dalam melakukan pemeriksaan status aplikasi, melakukan perutean untuk *service backend*, meskipun demikian ditemukan bahwa SOA bisa menjadi rumit dan menyebabkan terjadinya *bottleneck*. Arsitektur *Microservice* (MSA) dapat menangani kekurangan ini dengan terisolasi, independen dan mudah didistribusikan sehingga memudahkan skalabilitas. Keuntungan terbesarnya yaitu aplikasi bisa dibangun dengan berbagai pilihan teknologi dan memungkinkannya untuk digunakan secara independen satu sama lain. Ini sangat menyederhanakan siklus pengembangan, pengujian, pembuatan, dan penerapan aplikasi karena perubahan terbatas pada satu *service* daripada seluruh aplikasi [3].

Hal ini dibuktikan juga dengan penelitian sebelumnya yang melakukan uji performa berupa uji beban dari setiap arsitektur. Dimana MSA memiliki *throughput* yang lebih tinggi pada 1500 pengguna dengan nilai rata-rata 1,1 dibandingkan dengan arsitektur SOA sebesar 0,7 dan monolith sebesar 0,6. Selain itu pada *response time* MSA lebih cepat 5 detik yaitu sebesar 33 detik dibandingkan dengan monolith sebesar 38 detik dan SOA sebesar 43 detik [4].

Pada pengukuran jumlah kode *response* 200 (Berhasil), MSA memiliki



jumlah *response* tertinggi di kode berhasil dengan memiliki jumlah *response* terkecil di kode 302(Pengalihan), 304(Cache), 408(Waktu Habis), 500(Kesalahan Internal Server) dan tidak memiliki jumlah *response* di kode 404(Tidak ditemukan). Dimana pada aspek pemeliharaan aplikasi, MSA lebih unggul daripada SOA dan monolit unggul daripada SOA [4].

Naman manfaat ini hanya dapat dimanfaatkan jika *service* didekomposisi dengan cara yang paling optimal dengan mempertimbangkan gambaran besar dari seluruh cakupan aplikasi. Jika tidak, desain ini mungkin terbukti kontraproduktif dan menyebabkan latensi, kompleksitas, dan inefisiensi. Hal ini diperlukan untuk memisahkan aplikasi menjadi bagian-bagian yang sesuai secara fungsional dan memperoleh *service* kohesif tinggi dan *service* yang digabungkan secara longgar diharapkan sebagai hasil dari dekomposisi [3].

Dalam melakukan dekomposisi bisa dilakukan dengan konsep *Domain Driven Design*(DDD), *Functional*, *Dataflow*, dan *Dependency Capturing* dengan *Clustering*. Pada hasil evaluasi DDD menunjukkan aplikasi berhasil didekomposisi ke *microservice*. Dengan pendekatan *Functional* hasil evaluasi menunjukkan bahwa identifikasi *microservice* dapat dilakukan lebih cepat. Di pendekatan *dataflow* ini menunjukkan dekomposisi bisa ditentukan dari pertimbangan *coupling* dan *kohesi*. Identifikasi *microservice* dengan menganalisis ketergantungan proses bisnis dari control, dengan data dan control, data, dan semantic models. Kemudian untuk metode *Clustering* untuk mengidentifikasi *microservice*, metode *clustering* yang digunakan yaitu *Hierarchical Clustering*. Hasil dari validasi pendekatan ini menunjukkan bahwa pendekatan ini mencapai hasil yang lebih baik daripada pendekatan yang ada dalam hal identifikasi *microservice* [5].

Pada penelitian ini akan melakukan dekomposisi aplikasi ERP yang disebarkan secara monolit menjadi arsitektur *microservice* dengan pendekatan menganalisis *graph* yang dihasilkan dari source code kemudian dilakukan pengelompokan melalui *Hierarchical Clustering*. Hasil dari pengelompokan akan diimplementasikan dan dilakukan uji beban sehingga mengetahui nilai latensi, jumlah *throughput*, dan penggunaan sumber daya komputasi. Dengan ini diharapkan bisa menyelesaikan permasalahan yang terjadi di aplikasi ERP seperti kustomisasi dan skalabilitas.

### 1.2 Rumusan Masalah

Berikut adalah rumusan masalah yang dibuat berdasarkan latar belakang diatas.

1. Bagaimana dekomposisi microservice yang optimal melalui pendekatan Hierarchical Clustering?
2. Bagaimana performa aplikasi ERP antara arsitektur monolit dan arsitektur microservice dalam kondisi beban yang tinggi?
3. Berapa besar penggunaan sumber daya dan skalabilitas aplikasi ERP yang digunakan pada arsitektur monolit dibandingkan arsitektur microservice?

### 1.3 Tujuan Penelitian

Berdasarkan rumusan masalah di atas, maka tujuan tujuan penelitian ini adalah.

1. Menerapkan dekomposisi aplikasi ERP monolit ke microservice dengan pendekatan Hierarchical Clustering.
2. Mencari kelompok service yang memiliki nilai kopel rendah dan nilai kohesi tinggi.
3. Membandingkan performa dan sumber daya aplikasi ERP dengan bentuk pengelompokan yang berbeda di arsitektur microservice.

### 1.4 Batasan Masalah

Agar penelitian ini menjadi lebih terarah, maka penulis membatasi masalah yang akan dibahas sebagai berikut.

1. Penyebaran aplikasi dilakukan dengan framework Docker dengan bahasa pemrograman yang digunakan adalah Python.
2. Aplikasi yang digunakan adalah aplikasi yang sudah dibangun sebelumnya dan disebarkan dengan arsitektur monolit.
3. Perubahan arsitektur tidak dapat menjamin secara keseluruhan fungsionalitas dari aplikasi, karena keterbatasan waktu dan pengujian.
4. Hanya beberapa module pada ERP yang dilakukan dekomposisi.

### 1.5 Kontribusi Penelitian

Kontribusi yang diberikan pada penelitian ini adalah sebagai berikut.

1. Memberikan langkah dalam melakukan dekomposisi aplikasi monolit ke microservice dengan Hierarchical Clustering.
2. Mengetahui pengaruh dari performa aplikasi yang sudah dilakukan dekomposisi

dengan uji beban pada aplikasi.

3. Membuat aplikasi microservice yang memiliki nilai kohesi tinggi dan nilai kopel rendah.

### 1.6 Metodologi Penelitian

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

#### 1. Penelitian Pustaka

Penelitian ini dimulai dengan studi kepustakaan yaitu mengumpulkan referensi baik dari buku, paper, jurnal, atau artikel daring mengenai arsitektur microservice, permasalahan pada aplikasi ERP dan dekomposisi monolit ke microservice.

2. Analisis Dilakukan analisis permasalahan yang ada, batasan-batasan yang ditentukan, dan kebutuhan-kebutuhan yang diperlukan untuk menyelesaikan permasalahan yang ditemukan.

#### 3. Perancangan

Pada tahap ini dilakukan perancangan untuk melakukan dekomposisi dari aplikasi arsitektur monolit ke arsitektur microservice dengan dengan pendekatan Hierarchical Clustering.

#### 4. Implementasi

Pada tahap ini mengimplementasikan hasil perancangan dekomposisi ke aplikasi microservice pada aplikasi yang dibuat dengan arsitektur monolit.

#### 5. Pengujian

Pada tahap ini dilakukan pengujian pada aplikasi yang sudah di dekomposisi. Pengujian melalui uji beban akan dilakukan dengan perbandingan antara aplikasi monolit dan aplikasi microservice.

### 1.7 Sistematika Pembahasan

#### **BAB 1: PENDAHULUAN**

Pendahuluan yang berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, kontribusi penelitian, serta metode penelitian.

#### **BAB 2: LANDASAN TEORI**

Landasan Teori yang berisi penjelasan dasar teori yang mendukung penelitian ini, seperti arsitektur monolit, arsitektur microservice, hierarchical clustering, dan dekomposisi.

#### **BAB 3: ANALISIS DAN PERANCANGAN**

Analisis dan Perancangan yang berisi tahapan penerapan dekomposisi aplikasi monolit ke microservice dengan hierarchical clustering dan penyebaran aplikasi melalui kontainer.

### **BAB 4: IMPLEMENTASI DAN PENGUJIAN**

Implementasi dan Pengujian yang berisi pembangunan aplikasi dan pengujian dengan mensimulasikan dan mengevaluasi aplikasi yang telah didekomposisi.

### **BAB 5: KESIMPULAN DAN SARAN**

Penutup yang berisi kesimpulan dari penelitian dan saran untuk penelitian lebih lanjut di masa mendatang.

## BAB 2 LANDASAN TEORI

Pada bab ini menjelaskan beberapa teori dan jurnal yang berhubungan dengan permasalahan penelitian yang digunakan pada proses penelitian.

### 2.1 Tinjauan Pustaka

Pembahasan mengenai teori-teori tersebut dijelaskan sebagai berikut.

#### 2.1.1 Monolit

Monolit yaitu suatu cara untuk melakukan penyebaran. Ketika semua fungsi dalam sistem harus disebarkan secara bersama-sama, maka itu merupakan sebuah monolit [6]. Monolit merupakan sebuah aplikasi perangkat lunak dimana setiap modulnya tidak bisa dieksekusi secara independen. Hal ini membuat monolit sulit digunakan pada sistem terdistribusi tanpa bantuan penggunaan *frameworks* atau solusi *ad hoc* seperti Objek Jaringan, *RMI* atau *CORBA* [16].

Penggunaan pada bahasa pemrograman seperti *Java*, *C/C++*, dan *Python* pada pengembangan aplikasi di sisi *server*, memiliki kemampuan dalam melakukan abstraksi untuk memecah kompleksitas program menjadi berupa modul. Namun, bahasa pemrograman ini dirancang untuk membuat *artefacts* monolit. Dimana abstraksi ini tergantung pada penggunaan berbagi sumber data pada komputer yang sama (memori, database, file) [16].

Terdapat 3 jenis monolit [6]:

##### 1. *Single Process Monolith*

Dimana sebuah kode disebarkan dengan satu proses. Setiap kode bisa berada di banyak *instances* serta tempat penyimpanan dan mendapatkan data disimpan pada suatu database yang sama. Variasi lainnya yaitu modular monolit dimana setiap kode bisa bekerja secara independen tetapi perlu dijadikan satu kesatuan ketika ingin dilakukan *deployment*.

##### 2. *Distributed Monolith*

Monolit terdistribusi adalah sistem yang terdiri dari beberapa layanan, tetapi untuk apa pun alasannya seluruh sistem harus disebarkan bersama-sama. Sebuah monolit terdistribusi bisa memiliki kesamaan dengan *service-oriented architecture (SOA)*.

Monolit terdistribusi biasanya muncul di kondisi dimana tidak cukup fokus pada konsep *information hiding* dan kohesi dari fungsi bisnis. Akibatnya terbentuklah arsitektur yang memiliki kopel yang tinggi, dimana bisa

perubahan menyebabkan kerusakan pada bagian sistem lain.

### 3. *Sistem Black-Box Pihak Ketiga*

Aplikasi pihak ketiga merupakan sebuah monolit, misalkan sistem penggajian, sistem CRM, dan sistem SDM. Faktor umum yang terjadi yaitu aplikasi ini dibuat dan dikelola oleh orang lain dimana pengembang belum tentu memiliki kemampuan untuk mengubah kode seperti *Software-as-a-Service(SaaS)*.

Keuntungan dari Monolit [6, 8]:

1. Sederhana dalam melakukan pengembangan karena *Integrated Development Environment (IDE)* dan peralatan pengembang berfokus pada membuat satu aplikasi
2. Mudah untuk melakukan perubahan secara radikal di aplikasi. Perubahan ini bisa dari kode hingga skema database serta proses *deployment*.
3. Pengujian dilakukan pada satu aplikasi, pengembang dapat membuat pengujian dari awal hingga akhir dengan lebih mudah dan terintegrasi
4. Deployment dilakukan pada satu aplikasi, pengembang hanya menyalin aplikasi dari komputer ke komputer yang lain. Dengan ini aplikasi relatif mudah dilakukan konfigurasi dan mudah diperbanyak jumlah aplikasi.

Tantangan dari monolit [6, 8]:

1. Sulit dikembangkan secara berkelanjutan, karena semakin banyak orang yang bekerja pada aplikasi yang sama. Akibatnya setiap pengembang memiliki kepentingan masing-masing dalam mengelola kode yang sama dan membuat pengambilan keputusan sulit serta tidak fleksibel
2. Memiliki reliabilitas yang rendah, karena kesalahan pada salah satu module aplikasi bisa menyebabkan kegagalan secara keseluruhan aplikasi. Akibatnya aplikasi tidak dapat digunakan oleh pengguna dan harus dilakukan deployment kembali.
3. Tidak mudah untuk melakukan skalabilitas, setiap modul aplikasi memiliki kebutuhan sumber daya yang berbeda seperti ada module penyediaan data yang membutuhkan banyak memori sedangkan modul pemrosesan gambar membutuhkan banyak CPU, karena module ini berada pada aplikasi yang sama akibatnya pengembang harus melakukan pengorbanan pada salah satu

sisi sumber daya.

4. Terkunci pada teknologi jadul, pengembang terkunci pada teknologi awal yang digunakan untuk membangun aplikasi. Pengembang juga kesulitan ketika ingin mengadopsi teknologi baru pada aplikasi karena sangat berisiko dan sangat mahal untuk menulis kembali seluruh aplikasi antar teknologi.

### 2.1.2 *Microservice*

*Microservice* adalah beberapa *service* yang bisa di deploy secara independen yang dimodelkan berdasarkan bisnis domain. *Service* ini berkomunikasi satu sama lain melalui jaringan komputer dan bisa dibangun dengan berbagai macam teknologi. *Microservice* adalah salah tipe dari *service oriented architecture (SOA)* meskipun ada perbedaan dalam membuat batasan antara *service* dan *deployment* secara independen [8].

*Service* adalah komponen perangkat lunak yang memiliki kegunaannya secara khusus dimana komponen ini bisa berdiri sendiri dan secara independen dilakukan proses deployment. *Service* memiliki *API (Application Programming Interface)* yang memberikan akses kepada *client* untuk melakukan operasi. Terdapat 2 tipe operasi yaitu perintah dan kueri. *API* terdiri dari perintah, kueri dan *event*. Perintah dapat berupa *buatPesanan()* yang melakukan aksi dan memperbarui data. Kueri dapat berupa *cariPesananBerdasarkanID()* yang digunakan untuk mengambil data. *Service* juga dapat membuat suatu *event* seperti *PesananSudahDibuat* dimana *event* ini akan dikonsumsi oleh *client*-nya / *subscriber* [8].

*Service API* akan mengenkapsulasi internal implementasinya, sehingga pengembang aplikasi tidak bisa menuliskan kode yang melewati *API*. Akibatnya arsitektur *microservice* dapat mewajibkan modularitas di aplikasi. Setiap *service* di arsitektur *microservice* memiliki masing-masing arsitektur sendiri dan dimungkinkan dengan teknologi yang berbeda. Tetapi kebanyakan *service* memiliki arsitektur heksagonal. Dimana *API* akan diimplementasi melalui adapter yang berinteraksi dengan logika bisnis [7]

Ciri Khusus *Microservice* [6, 7, 8]:

1. Kecil dan berfokus pada satu hal dengan baik  
*Service* yang dibuat memiliki *encapsulation* dengan pembuatan *service* dimodelkan di sekitar Domain Bisnis, tujuannya agar ketika terjadi

perubahan antar *service* bisa dilakukan dengan lebih mudah dan tidak berdampak pada *service* lain. Oleh karena itu *service* yang dibuat seminimal mungkin untuk tidak berhubungan dengan *service* lain.

### 2. Otonomi / Bisa berdiri sendiri

*Microservice* memiliki *service* yang terisolasi dimana bisa memiliki sistem operasi hingga komputer yang berbeda. Dengan ini sistem terdistribusi lebih sederhana dan nilai kopel yang rendah. Semua komunikasi antar *service* dilakukan melalui jaringan sehingga *service* harus memiliki kemampuan *dideploy* sendiri tanpa harus mempengaruhi *service* lain.

### 3. Data yang dikelola masing-masing *service*

*Service* yang membutuhkan data diluar domainnya harus berkomunikasi melalui *API(application programming interface)*, dengan ini setiap *service* memiliki tanggung jawab terhadap datanya masing-masing sehingga data tersebut hanya bisa diubah oleh *service* itu sendiri. Setiap *service* memiliki data yang pribadi dan data yang bisa dibagikan kepada *service* lain

Keuntungan dari *Microservice* [6, 7, 8]:

#### 1. Memudahkan pengembangan aplikasi kompleks dan flexibel

*Service* berukuran kecil sehingga mudah dikelola, perubahan pada satu *service* bisa diterapkan secara independen dari *service* lainnya. Bila terjadi kegagalan di satu *service* tidak berdampak besar pada *service* lainnya karena *service* masing-masing terisolasi selain itu proses pemulihan bisa dilakukan dengan mudah dan cepat.

#### 2. Bisa dilakukan skaling secara independen

Setiap *service* memiliki fungsi yang berfokus pada satu hal, dimana setiap *service* bisa memiliki kebutuhan sumber daya berbeda. Penggunaan sumber daya ini bisa dikelola dengan mudah dan cepat karena setiap *service* dapat *dideploy* dengan jumlah *service* yang berbeda.

#### 3. Mudah melakukan percobaan dan penggunaan teknologi baru

Arsitektur *microservice* mengeliminasi komitmen penggunaan secara lama pada suatu teknologi. Dengan ini pengembang dapat memilih berbagai teknologi dalam membangun *service* serta *service* yang kecil dan berfokus lebih mudah untuk dilakukan migrasi antara teknologi yang berbeda.

Tantangan dari *Microservice* [6, 7, 8]:



### 1. Menemukan *service* yang tepat itu sulit

Salah satu tantang terbesar dari membuat *microservice* yaitu tidak adanya cara yang pasti bagaimana untuk melakukan dekomposisi dengan baik. Dimana *service* yang didekomposisi dengan tepat tidak mudah ditemukan dan bila dilakukan dengan tidak benar dapat sebaliknya membuat *distributed monolith*.

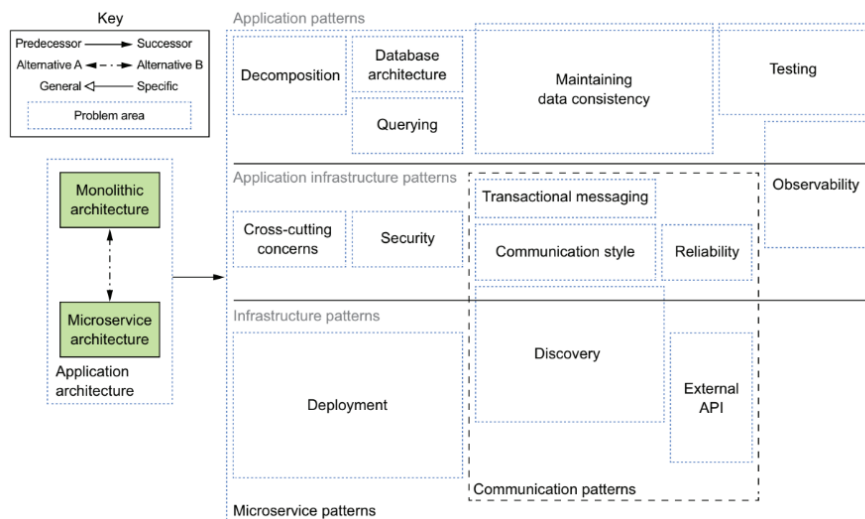
### 2. Memiliki kompleksitas karena merupakan suatu terdistribusi

Setiap *service* untuk berkomunikasi antar *service* memiliki tantangan masing-masing seperti latensi, konsistensi data, dan kondisi ketika beberapa *service* mengalami kegagalan. *Microservice* juga meningkatkan kompleksitas operasional oleh karena itu untuk melakukan *deployment* sebaiknya menggunakan proses otomatisasi.

### 3. *Deployment* yang melibatkan beberapa *service*

Untuk melakukan *deployment* ini dibutuhkan koordinasi antara tim pengembang *servic* ketika menambahkan atau mengubah fitur yang berdampak pada beberapa *service* maka dari itu harus dibuat perencanaan *deployment* berdasarkan ketergantungan antar *service*.

Pola *Microservice* [8]:



**Gambar 2.1** Pola dalam menyelesaikan masalah di arsitektur *Microservice* [8]

1. *Application patterns* Permasalahan yang harus diselesaikan oleh pengembang aplikasi - Proses kueri - Dekomposisi aplikasi menjadi service - Menjaga data konsistensi dan implementasi proses transaksi - Mengotomatiskan proses pengujian service ...

2. *Application infrastructure* Permasalahan infrastruktur yang memiliki pengaruh pada proses pengembangan aplikasi - Pola komunikasi - Pola mengatasi permasalahan antar service ...
3. *Infrastructure patterns* Permasalahan infrastruktur yang muncul diluar dari pengembangan aplikasi - Pola Komunikasi - Pola Service Deployment - Pola observability untuk mengetahui bagaimana aplikasi bekerja ...

### 2.1.3 *Enterprise Resource Planning*

*Enterprise Resource Planning* (ERP) adalah suatu sistem perangkat lunak yang memungkinkan perusahaan untuk mengotomatisasikan dan mengintegrasikan proses bisnisnya dengan komputerisasi. Dengan ini setiap informasi yang diperlukan di proses bisnis dapat dibagikan dan digunakan disemua bagian perusahaan dengan alur terstruktur. Sistem ERP dapat mengeliminasi duplikasi data dan memberikan integrasi data. Sistem ERP memiliki database dimana semua transaksi bisnis dapat direkam, diproses, dipantau dan dilaporkan. Tujuannya agar proses bisnis bisa dilakukan dengan lebih cepat, murah, dan transparan [1].

Sistem ERP dapat memberikan dukungan untuk proses bisnis perusahaan melalui modul yang terpisah. Setiap modul adalah aplikasi perangkat lunak yang dibangun khusus untuk setiap operasi bisnis. Umumnya modul yang ditemukan pada ERP yaitu Modul Produksi, Modul Manajemen Rantai Pasokan, Modul Keuangan, Modul Penjualan & Pemasaran, Modul Sumber Daya Manusia, dan modul pelengkap lainnya seperti *e-commerce* [1].

Arsitektur ERP [1]:

#### 1. *The Tiered*

Arsitektur *tiered* umumnya dirancang dalam bentuk lapisan yang didasarkan dari model *client-server* atau bisa disebut *N-Tier*. Dalam arsitektur ini setiap komponen ERP disusun kedalam masing lapisan seperti lapisan *user interface*, lapisan aplikasi dan lapisan *database / penyimpanan data*.

#### 2. *Web-based*

Arsitektur *Web-based* mengadopsi teknologi berorientasi objek web dimana pengguna yang ingin menggunakan sistem ERP bisa mengakses melalui *browser* dan internet. *Object-oriented technology* diimplementasi untuk mencampur data dan fungsi yang tersedia di berbagai *web service*.

### 3. *Service Oriented*

*SOA*(*Service Oriented Architecture*) adalah sistem yang dimana terdapat fungsi modular yang berkomunikasi melalui jaringan. Satu atau lebih *service* bisa berkordinasi dalam suatu aktivitas fungsi bisnis.

### 4. *Cloud*

*Cloud* dapat memberikan solusi bagi organisasi ketika mengadopsi sistem ERP pada kegiatan bisnisnya. Sistem ERP dengan arsitektur *cloud* bisa dikategorikan sebagai tipe *SaaS*(*Software as a Service*). Organisasi akan membayar pihak ke tiga setiap periode berdasarkan modul yang digunakannya.

#### 2.1.4 Analisis Kode

Analisis Kode adalah suatu proses mengekstraksi informasi mengenai suatu program dari kode atau artifak. Proses ini bisa dilakukan secara manual yaitu dengan melihat kode program atau bahasa mesin namun kompleksitas program yang tinggi membuat proses secara manual sangat sulit dan tidak efektif. Sehingga diperlukan alat otomatisasi yang dapat membantu proses analisis kode. Alat ini dapat memberikan informasi kepada pengembang mengenai program yang dianalisis [9].

Anatomi Analisis Kode [9]:

#### 1. Ekstraksi Data

Proses ini adalah proses pertama kali dilakukan sebelum melakukan analisis kode, data yang diekstraksi berasal dari kode program. Umumnya dilakukan dengan *syntatic analyzer* atau *parser*. Proses *Parser* ini mengkonversi urutan karakter menjadi suatu kata-kata dan mengekstraksi nilai semantik sebenarnya. Tujuannya agar memudahkan proses analisis/transformasi dan penambahan data lainnya.

#### 2. Representasi Informasi

Proses yang merepresentasikan informasi kode menjadi bentuk yang lebih abstrak. Tujuan dari fase ini untuk membentuk beberapa bagian kode agar terhubung pada analisis secara otomatis. Representasi ini kebanyakan berupa graph seperti *Abstract Syntax Trees (AST)*, *Control Flow Graphs (CFG)*, dan *Call Graph*.

#### 3. Eksplorasi Pengetahuan

Setelah informasi direpresentasikan, informasi dibuat menjadi suatu

kesimpulan. Kesimpulan bisa dibuat secara kuantitatif atau kualitatif, proses visualisasi penting dalam proses eksplorasi pengetahuan kode program.

Strategi Analisis Kode [9]:

1. Statik vs Dinamis

Analisis secara statik menganalisis program tanpa dieksekusi untuk mendapatkan semua informasi yang kemungkinan akan dieksekusi. Sedangkan secara Dinamis, program mengumpulkan informasi yang dieksekusi dengan nilai yang diberikan. Beberapa teknik analisis menggabungkan kedua pendekatan ini.

2. *Sound vs Unsound*

*Sound* yaitu analisis yang bisa menjamin secara keseluruhan dan kebenaran eksekusi program. Sedangkan *Unsound* tidak bisa secara keseluruhan menjamin kebenaran hasil analisis program. Namun dalam banyak kasus analisis *Unsound* memiliki hasil yang benar selain itu memiliki kelebihan yaitu mudah diimplementasi dan efisien.

3. *Flow sensitive vs Flow insensitive*

*Flow sensitive* memperhatikan dan menyimpan urutan proses eksekusi sedangkan *Flow insensitive* tidak memperhatikan urutan proses eksekusi sehingga tidak memiliki informasi ketergantungan pada suatu proses dan hanya dapat menyatakan proses tersebut ada.

4. *Context sensitive vs Context insensitive*

*Context in-sensitive* hanya menghasilkan satu hasil yang berhubungan dalam semua konteks. Sedangkan *context sensitive* memiliki hasil berbeda ketika konteks berbeda. Pendekatan ini bertujuan untuk menganalisis proses pembuatan analisis umumnya tanpa adanya informasi mengenai konteks yang akan digunakan. *Context sensitive* penting untuk menganalisis program modern dimana terdapat suatu abstraksi.

Tantangan Kode Analisis [9]:

1. Perbedaan bahasa kode program

Banyak peningkatan dan perubahan pada bahasa pemrograman seperti *dynamic class loading* dan *reflection*. Konsep ini juga terdapat pada proses pengubahan tipe data, *pointer*, *Anonymous types* yang membuat proses *parser* sulit. Fitur pada pemrograman ini meningkatkan fleksibilitas ketika

program berjalan dan membutuhkan analisis secara dinamik yang lebih kuat.

### 2. *Multi-Language*

Banyak aplikasi yang dibuat sekarang dibangun dengan berbagai bahasa pemrograman. Dimana sekarang perlengkapan pembuatan aplikasi masih belum bisa menganalisis secara keseluruhan pada aplikasi yang menggunakan banyak bahasa pemrograman. Seperti aplikasi berbasis web yang memiliki *HTML*, *ASP*, *Java* dan lainnya.

### 3. Analisis secara *Real-Time*

Analisis ini dapat memberikan keuntungan bagi pengembang karena memberikan informasi tambahan selama pengembang membuat aplikasi seperti *code coverage* dan analisis kebocoran memori. Proses analisis juga kerap kali membutuhkan penggunaan sumber daya komputasi yang tinggi dan memori yang banyak.

#### 2.1.5 *Clustering*

*Clustering* yaitu suatu proses untuk melakukan pengelompok atau klasifikasi objek. Objek bisa ditentukan dari pengukuran atau berdasarkan hubungan antar objek lainnya. Tujuan dari clustering yaitu untuk menemukan struktur data yang valid. Cluster terdiri dari sejumlah object serupa yang dikumpulkan / dikelompokkan bersama.[10]

Metode *Clustering* yang umumnya digunakan [10]:

#### 1. *Hierarchical Clustering*

Metode *Hierarchical Clustering* adalah sebuah prosedur untuk mentransformasi sebuah *proximity matrix* menjadi beberapa partisi. Clustering adalah sebuah partisi dimana komponen dari partisi disebut *clusters*. Beberapa partisi memiliki suatu urutan dan tingkatan berdasarkan bagaimana partisi tersebut disatukan. Terdapat 2 pendekatan algoritma dalam membentuk suatu partisi yaitu secara *agglomerative* dan *divisive*. Pendekatan *Agglomerative* dimulai dari setiap objek memiliki partisi masing-masing dan terpisah, kemudian algoritma mengukur nilai *proximity matrix* setiap objek untuk menentukan berapa banyak penggabungan partisi lain yang perlu dilakukan. Proses dilakukan berulang kali dan jumlah partisi akan berkurang hingga tersisa satu partisi, satu partisi ini memiliki keseluruhan objek. Sedangkan pendekatan secara *divisive* melakukan hal

yang sama seperti *Agglomerative* namun prosesnya terbalik yaitu dimulai dari satu partisi.

### 2. *Partitional Clustering*

*Partitional* menggunakan pendekatan dimana diberikan sejumlah  $n$  pola pada data dimensional, kemudian tentukan partisi dari pola menjadi beberapa cluster. Pendekatan *Hierarchical* populer digunakan dibidang biologi, sosial, dan ilmu perilaku karena keperluan untuk membentuk suatu taxonomi. Sedangkan *Partitional* digunakan umumnya di aplikasi teknik.

Dimana satu partisi lebih penting, Metode *Partitional* juga memiliki efisiensi dan kompresi yang cocok untuk data yang besar sehingga pola dalam cluster memiliki kemiripan satu sama lain daripada pola dalam *cluster* yang berbeda. Pemilihan nilai *cluster* bisa ditentukan secara opsional, kriteria *cluster* yang valid harus ditentukan seperti *square-error* untuk menentukan apakah partisi yang dibuat optimal. Kriterianya sendiri bisa dibagi menjadi secara global atau lokal.

Pemilihan Partisi [15]:

1. Secara Struktural dan Perilaku dari *microservice*
2. Nilai *Coupling* dan *Cohesion*
3. Berdasarkan ketergantungan data antar *Class*

#### 2.1.6 Dekomposisi

Pemilihan bagian yang ingin didekomposisi untuk menjadi Service [6]:

1. Proses bisnis
2. Sub-domain (DDD)
3. Analisis Kode

Pola untuk Proses Dekomposisi:

1. Pola Strangle
2. Pola UI Composition
3. Branch By Abstraction
4. Parallel Run

5. Decorating Collaborator

6. Change Data Capture

Tantangan dan Hambatan Dekomposisi:

1. Latensi Jaringan

2. Menjaga konsistensi data antar service

3. Adanya God Class yang mencegah dekomposisi

### 2.1.7 Teknologi dan *Library*

#### 2.1.7.1 *Docker*

...

#### 2.1.7.2 *gRPC*

...

#### 2.1.7.3 *PyCG*

### 2.2 Tinjauan Studi

Pada Tabel 2.1 diberikan penjelasan mengenai studi terkait dalam penelitian:

Tabel 2.1 <i>State of the Art</i>			
Jurnal	Rumusan Masalah	Metode	Hasil
Munezero Immaculée Josélyne, Doreen Tuheirwe-Mukasa, Benjamin Kanagwa, and Joseph Balikuddembe. (2018, May). "Partitioning microservices: a domain engineering approach." [11]	Bagaimana menentukan ukuran yang tepat pada suatu service?	Menggunakan metodologi konseptual Domain Driven Design	Berhasil membuat <i>dissemination domain</i> pada aplikasi cuaca

Tyszberowicz, S., Heinrich, R., Liu, B., Liu, Z. (2018). "Identifying Microservices Using Functional Decomposition." [12]	Bagaimana cara mencari partisi yang tepat pada sistem untuk dijadikan microservice?	Menggunakan spesifikasi use case dan menggunakan functional decomposition	Memiliki hasil yang sebanding dengan teknik manual tapi dengan waktu yang lebih singkat
Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni. (2022). "A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications" [13]	Bagaimana mengotomatisasi proses migrasi aplikasi monolith ke microservice?	Menggunakan DBSCAN(Density-based Clustering) yang menghasilkan rekomentasi microservice	Berhasil membuat microservice yang lebih kohesive dan memiliki interaksi yang lebih sedikit.
Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao, Jidong Ge, Zhihao Shan. (2019). "A dataflow-driven approach to identifying microservices from monolithic applications." [14]	Bagaimana cara menyelesaikan masalah dekomposisi microservice	Menggunakan DFD(Data Flow Diagrams) untuk mengekstraksi ketergantungan antar proses dan data	Menghasilkan coupling dan cohesion yang baik dengan cara yang mudah dan semi-otomatis.



Chaitanya K. Rudrabhatla. (2020). "Impacts of Decomposition Techniques on Performance and Latency of Microservices." [3]	Bagaimana dan dampak performa dalam menentukan batasan antar service	Melakukan perbandingan antara pendekatan DDD, Normalized Entity Relationship, Hybrid	Teknik DDD lebih baik dalam dekomposisi namun teknik Hybrid dengan mempertimbangkan fungsionalis dan transaksi yang terjadi memiliki performa lebih baik.
--	--	--	---

Pada penelitian Munezero Immaculée Josélyne, kendala yang muncul ketika membuat microservice yaitu bagaimana mengetahui batasan antara komponen service. Dengan menggunakan Domain Driven Design bisa memberikan beberapa cara untuk mengetahui batasan pada domain yang besar dan kompleks. Metode Domain Driven Design direkomendasikan dalam proses perancangan microservice.

Kemudian penelitian yang menggunakan pendekatan secara functional dilakukan dari mengidentifikasi microservice dengan spesifikasi use-case dari software requirements. Kemudian melakukan dekomposisi melalui tool visualisasi dimana visualisasi dihasilkan dari relasi antar objek, operasi dan variabel yang berasal dari source code. Selain itu ada pendekatan dengan menggunakan pengelompokan khusus menggunakan hierarchical clustering yang dikombinasi dengan struktur dan analisis semantic. Dengan ini bisa mendeteksi outlier. Hasil dari pengelompokan akan dikomparasi dan dilakukan evaluasi.

Pendekatan dataflow-driven juga dapat membuat identifikasi secara sistematis dan mudah dipahami dalam melakukan dekomposisi dibandingkan dengan cara manual lainnya. Proses yang dilakukan yaitu dimulai dari menggunakan Use Case Specification dan Logic Bisnis dan kemudian diubah menjadi Data Flow diagram. Hasil dari diagram ini akan memberikan kandidat microservice yang bisa dibuat.

Penelitian lainnya berfokus pada dampak dari pendekatan masing-masing dekomposisi karena bila dekomposisi tidak dilakukan dengan baik maka bisa

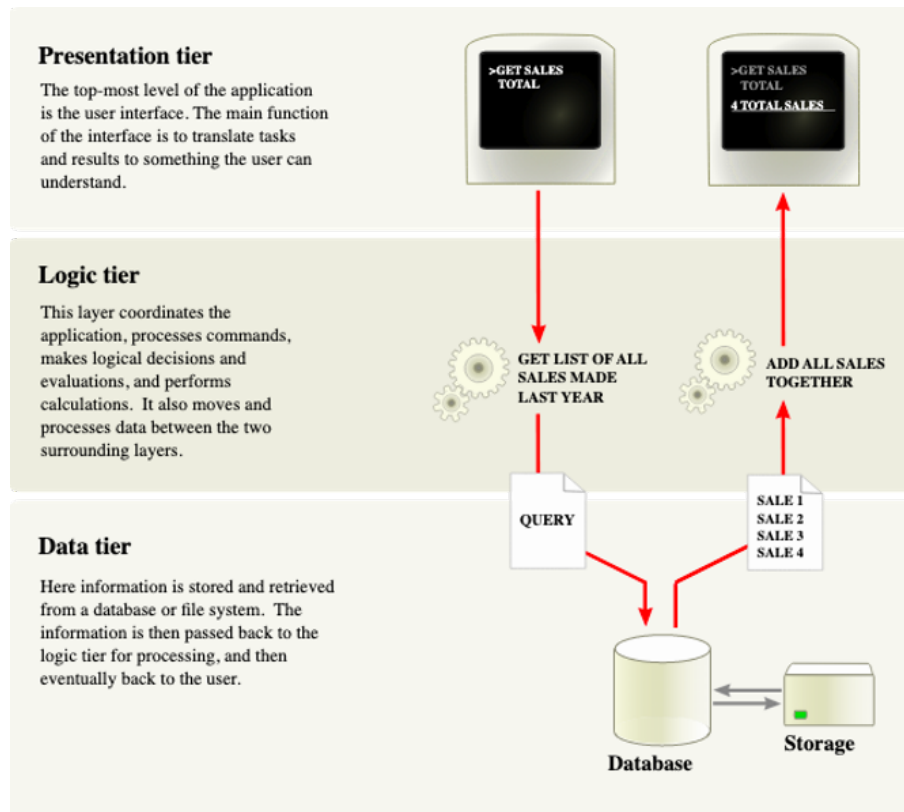
terjadi permasalahan pada latensi, kompleksitas dan ketidakefisienan. Penelitian ini menggunakan skenario e-commerce pada aplikasi microservice yang didekomposisi dengan pendekatan yang berbeda.

### **2.3 Tinjauan Objek**

Pada bagian ini akan dijelaskan mengenai objek dan aplikasi terkait yang akan digunakan dalam tugas akhir ini. Object yang digunakan adalah sebuah aplikasi Enterprise Resource Planning yang di deploy secara monolit, yaitu Odoo.

Odoo merupakan aplikasi bisnis open source yang dapat mencakup semua kebutuhan perusahaan seperti CRM(Customer Relationship Management), eCommerce, akuntansi, inventaris, POS(Point of Sales), manajemen proyek dan lainnya. Aplikasi ini flexibel karena bisa dikembangkan lebih lanjut bila diperlukan dan bisa diubah karena memiliki lisensi source code yang terbuka.

Arsitektur yang digunakan pada Odoo yaitu three-tier arsitektur dimana tampilan, aturan bisnis dan tempat penyimpanan data memiliki lapisan terpisah. Dengan tujuan memudahkan dan mempercepat pengembang untuk melakukan modifikasi aplikasi tanpa harus mengganggu lapisan lainnya.



**Gambar 2.2** Ilustrasi Arsitektur Odoo

Pada tingkatan paling atas yaitu tampilan(presentation tier), tampilan ini yang akan berinteraksi langsung dengan pengguna yang menggunakan aplikasi. Tampilan ini dibangun dengan teknologi web yaitu HTML5, Javascript, dan CSS. Tingkatan dibawahnya yaitu aturan bisnis(logic tier) yang berisi instruksi yang memproses data dan memberikan tanggapan dari interaksi kepada pengguna. Aturan pada Odoo hanya ditulis dalam bahasa pemrograman Python. Sedangkan pada tingkat paling bawah adalah tempat penyimpanan menggunakan DBMS(Database Management System), Odoo hanya bisa mendukung database PostgreSQL.

Odoo memiliki struktur kode yang dibentuk sebagai module untuk setiap fiturnya. Sehingga dari sisi server dan client memiliki hubungan yang disatukan menjadi satu paket tersendiri. Dimana module adalah koleksi dari fungsi dan data untuk menyelesaikan satu tujuan. Modul pada Odoo bisa ditambahkan, diganti, diubah untuk menyesuaikan kebutuhan bisnis. Dimana pada pengguna module dilambangkan dengan nama Apps, tetapi tidak semua module adalah Apps. Modules juga bisa direfrensikan sebagai addons.

**Tabel 2.2** Komposisi dari Module pada aplikasi Odoo

<b>Elemen</b>	<b>Keterangan</b>	<b>Contoh</b>
Business Objects	Object yang akan digunakan di module dimana setiap attribute secara otomatis dipetakan ke kolom database dengan ORM	File python yang memiliki class
Objects Views	Menangani bagaimana data ditampilkan di pengguna. Seperti visualisasi form, list, kanban dan lainnya	Berupa file XML dengan struktur yang sudah ditentukan Odoo
Data Files	Mengelola bagaimana model data seperti laporan, konfigurasi data, data contoh dan lainnya	Berupa file XML atau CSV
Web Controllers	Menangani permintaan dari browser/client	File python yang memiliki class namun merupakan turunan dari class <code>odoo.http.Controller</code>
Static Web Data	File yang digunakan hanya ditampilkan kepada client di website	File gambar, File CSS, dan File JavaScript

Struktur database yang dibentuk pada konfigurasi umum di Odoo memiliki jumlah tabel ±566 tabel, tabel ini merupakan keseluruhan dari aplikasi dimana dapat diidentifikasi 31 tabel utama yang digunakan pada aplikasi. Berikut adalah diagram dari database yang dibuat dengan alat DBeaver Visualize, dengan attribute hanya sebuah key dari tabel.



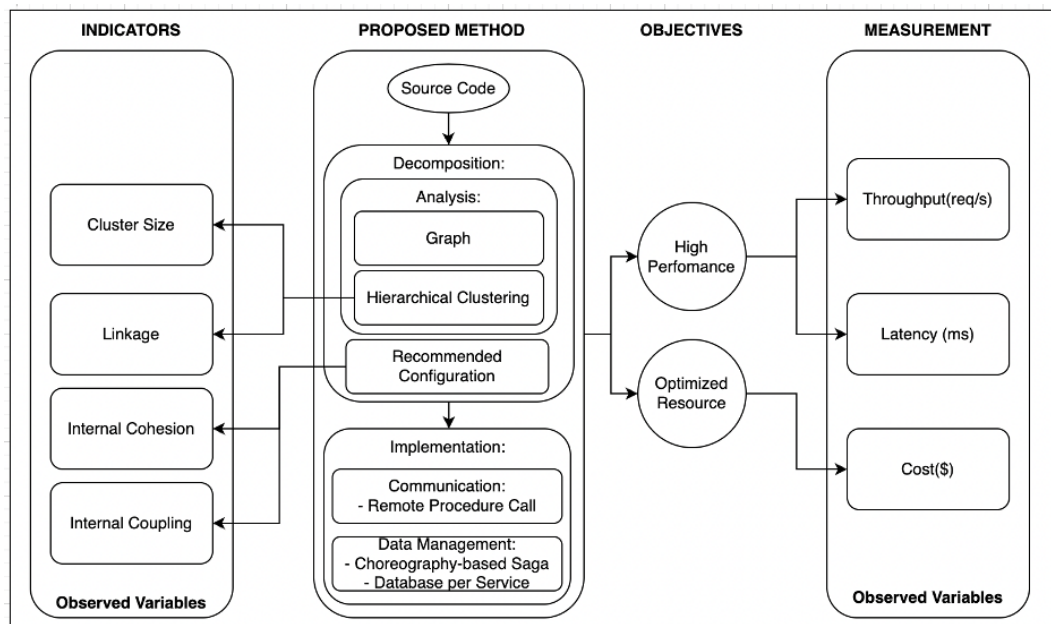
## BAB 3 ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini menjelaskan analisis masalah yang diatasi, alur kerja dari perangkat lunak yang dikembangkan, arsitektur dan metode yang digunakan serta hasil evaluasi

### 3.1 Analisis Masalah

..... - Arsitektur yang digunakan Odoo(ERP) masih berupa monolith sehingga memiliki kelemahan seperti skalabilitas, sulit dikembangkan berkelanjutan, dan terkunci pada teknologi tertentu. - Arsitektur Microservice dapat menyelesaikan masalah tersebut - Namun diperlukan proses dekomposisi dari mono-to-micro. Proses dekomposisi sendiri tidak mudah dan melelahkan karena masih membutuhkan proses manual. - Proses manual ini bisa di"mudahkan" dengan melakukan clustering khususnya dengan metode hierarchical - Sebelum melakukan proses clustering diperlukan analisis kode seperti Call Graph untuk mendapatkan graph dari source code aplikasi Odoo (ERP) - Hasil terbaik dari clustering diimplementasikan menjadi service, service dan dilakukan pemecahan dengan pattern yang sudah ditemukan - Dilakukan evaluasi dari aplikasi yang dibangun untuk menentukan apakah microservice dan clustering bisa melakukan dekomposisi dapat menyelesaikan permasalahan pada kasus ERP ...

### 3.2 Kerangka Pemikiran



Gambar 3.1 Kerangka Pemikiran

Penelitian akan dimulai dengan menggunakan kode sumber aplikasi yang dibuat dengan monolit. Kode sumber ini dilakukan proses dekomposisi yaitu dengan analisis seperti mencari objek beserta atributnya, untuk mencari keterhubungan lebih lanjut tentang objek maka dilakukan pencarian pada fungsi-fungsi sehingga terbentuklah grafik yang menunjukkan bagaimana keterhubungan objek di aplikasi.

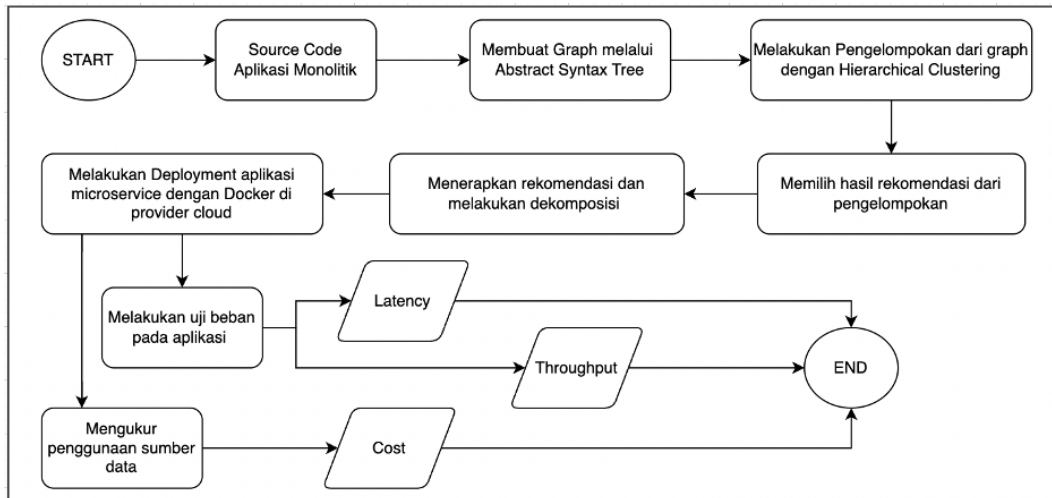
Dari grafik yang sudah dibuat akan dilakukan pengelompokan dengan pendekatan Hierarchical Clustering. Dimana perlu ditentukan jumlah cluster minimum yang harus ditemukan dan pemilihan algoritma Linkage. Metode linkage yaitu menentukan jarak atau kemiripan antara semua objek. Untuk menentukan jarak ini bisa dengan rata-rata, maximum, minimum dan mengecilkan variance.

Pengelompokan dari Hierarchical Clustering akan dipilih dengan mencari nilai cohesion terendah dan nilai coupling tertinggi. Dimana Internal Coupling mengevaluasi tingkat ketergantungan langsung dan tidak langsung antar objek. Semakin banyak dua objek menggunakan metode masing-masing semakin mereka menjadi satu kesatuan. Sedangkan Internal Cohesion akan mengevaluasi kekuatan interaksi antar objek. Biasanya, dua objek atau lebih menjadi interaktif jika metodenya bekerja pada atribut yang sama.

Ketika analisis dekomposisi sudah selesai dilakukan maka akan dilakukan implementasi berdasarkan pengelompokannya masing-masing yang akan menjadi service. Untuk metode komunikasinya antara service yaitu dengan Remote Procedure Call(RPC) dan untuk mengelola data, setiap service memiliki databasenya masing-masing dan untuk menjaga konsistensi data antar service maka digunakan pendekatan SAGA dengan cara choreography.

Untuk mengetahui bagaimana performa dari microservice dibandingkan dengan monolit yaitu dengan test beban. Pengukuran performa dilihat dari throughput, jumlah response, dan latency.

### 3.3 Urutan Proses Global



Gambar 3.2 Diagram Flowchart Proses Global

#### 3.3.1 Proses Clustering

##### 3.X.X Flowchart Proses Clustering

##### 3.3.1.1 Pengambilan Source Code

- Mengunduh dari branch main/master Odoo versi 16 dan semua workflow test pass
- Instalasi requiment Odoo dengan pip ...

##### 3.3.1.2 Pembuatan Call Graph

Menggunakan library call graph (pycg) dengan PyCG yang sudah dimodifikasi ...

##### 3.3.1.3 Ilustrasi Graph

Ilustrasi menggunakan tools graphviz ...

##### 3.3.1.4 Extrasi Call Graph

Proses ekstraksi json yang dihasilkan dari tools PyCG ...

##### 3.3.1.5 Extrasi Depedency Module

- Meparse file manifest.py yang di miliki setiap addons(module) untuk melihat module yang dibutuhkan ...

##### 3.3.1.6 Extrasi Depedency Data

- Meparse XML Data yang berada di setiap addons(module) pada folder 'data' ...

##### 3.3.1.7 Pre Procesinng

- Memisahkan hubungan module external(library) dengan module aplikasi Odoo
- Membuat weight graph dari setip koneksi antar Module
- Membuat adjacency matrix ...

##### 3.3.1.8 Hierarchical Clustering

- Menggunakan python dan library hierarchical clustering SciPy
- Menampilkan dendogram dengan Matplotlib ...



### 3.3.1.9 Pemilihan Partisi

Partisi yang dikelompokkan akan dipilih berdasarkan nilai coupling yang rendah dan cohesion yang tinggi dengan mempertimbangkan ketergantungan data antar module ....

### 3.3.2 Dekomposisi Monolitik ke Microservice

#### 3.3.2.1 Pemisahan User Interface

- Pemisahan web service "werkzeug" odoo dari monolith yang akan merender frontend...

#### 3.3.2.2 Strategi Pemisahan yg dipilih

- Menggunakan Pola Strangle
- Package Diagram / Komponen Diagram monolith dan komponen diagram yang dipilih dari Clustering
- State diagram autentikasi aplikasi
- Sequence Diagram antar service dan module (pada kasus transaksi mis. penjualan barang) ...

#### 3.3.2.3 Strategi Pemisahan database

- Menggunakan pattern monolith as data access layer
- konsistensi data dengan choreography based saga ...

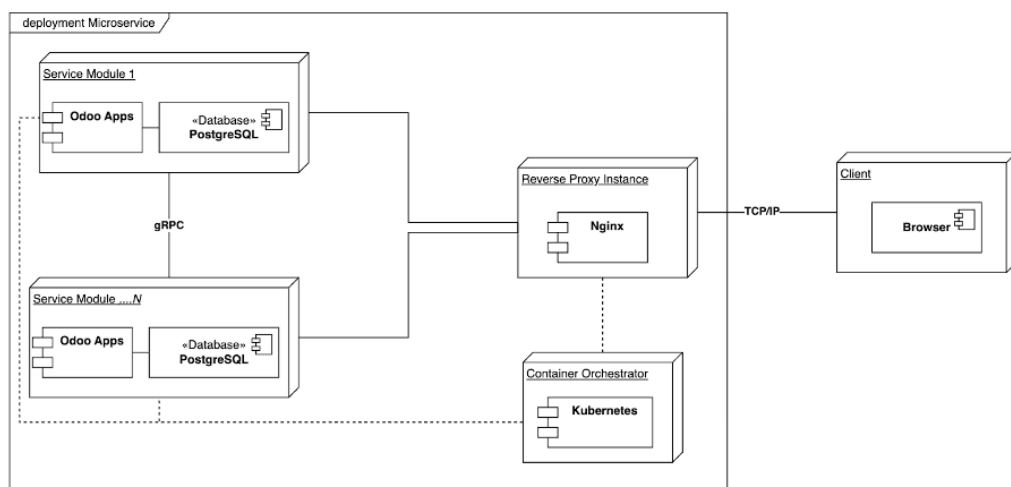
#### 3.3.2.4 Komunikasi antar service

- Menggunakan RPC(gRPC/JSON-RPC) ...

#### 3.3.2.5 Infrastruktur

- Menggunakan API Gateway (Kong) sebagai salah satu cara untuk dalam melakukan pola strangle
- Konfigurasi Docker setiap service ..

### 3.3.3 Deployment



**Gambar 3.3** Diagram Deployment

## DAFTAR REFERENSI

- [1] Amini, Mohammad and Abukari, Arnold. (2020). "ERP Systems Architecture For The Modern Age: A Review of The State of The Art Technologies." Journal of Applied Intelligent Systems and Information Sciences. Volume 1(2), pp.70-90. Available: <https://doi.org/10.22034/jaisis.2020.232506.1009> . [Accessed: 27-Oct-2022].
- [2] Bender, B.; Bertheau, C. and Gronau, N. (2021). "Future ERP Systems: A Research Agenda." In Proceedings of the 23rd International Conference on Enterprise Information Systems. 2, pp.776-783. Available: <http://dx.doi.org/10.5220/0010477307760783>. [Accessed: 27-Oct-2022]
- [3] Chaitanya K. Rudrabhatla. (2020). "Impacts of Decomposition Techniques on Performance and Latency of Microservices." International Journal of Advanced Computer Science and Applications(IJACSA). 11(8). Available: <http://dx.doi.org/10.14569/IJACSA.2020.0110803>. [Accessed: 27-Oct-2022]
- [4] Slamaa, A.A., El-Ghareeb, H.A. , Saleh, A.A. (2021). "A Roadmap for Migration System-Architecture Decision by Neutrosophic-ANP and Benchmark for Enterprise Resource Planning Systems." IEEE Access . 9, pp.48583-48604. Available: <https://doi.org/10.1109/ACCESS.2021.3068837>. [Accessed: 27-Oct-2022]
- [5] Söylemez, M.; Tekinerdogan, B.; Kolukısa Tarhan. (2022). "A. Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review Planning Systems." Applied Science . 12(11), pp.48583-48604. Available: <https://doi.org/10.3390/app12115507>. [Accessed: 27-Oct-2022]
- [6] Sam Newman, *Monolith to Microservices*, Sebastopol, CA: O'Reilly Media, Inc., 2020, pp. 12-15 [[Link GoogleDrive](#) ]
- [7] Sam Newman, Building microservices (2015). Sebastopol: O'Reilly Media, Inc, USA.
- [8] Richardson, C. (2018) Microservice patterns: With examples in Java. Shelter Island, NY: Manning Publications.

- [9] Cruz, D.D., Henriques, P.R. and Pinto, J.S. (2009) Code analysis: past and present [Preprint]. Available at: <https://hdl.handle.net/1822/14352>.
- [10] Jain, A.K. and Dubes, R.C. (1988) Algorithms for clustering data. Englewood Cliffs, NJ: Prentice Hall.
- [11] Munezero Immaculée Josélyne, Doreen Tuheirwe-Mukasa, Benjamin Kanagwa, and Joseph Balikuddembe. (2018, May). "Partitioning microservices: a domain engineering approach." In Proceedings of the 2018 International Conference on Software Engineering in Africa (SEiA '18). May 2018, pp-43-49. Available: <https://doi.org/10.1145/3195528.3195535>. [Accessed: 27-Oct-2022]
- [12] Tyszberowicz, S., Heinrich, R., Liu, B., Liu, Z. (2018). "Identifying Microservices Using Functional Decomposition." Dependable Software Engineering. Theories, Tools, and Applications. SETTA 2018. vol 10998. Springer, Cham. Available: [https://doi.org/10.1007/978-3-319-99933-3\\_4](https://doi.org/10.1007/978-3-319-99933-3_4). [Accessed: 27-Oct-2022]
- [13] Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni. (2022). "A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications" In The International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE 2022). ACM, New York, NY, USA, 11. Available: <https://doi.org/10.1145/3530019.3530040>. [Accessed: 27-Oct-2022]
- [14] Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao, Jidong Ge, Zhihao Shan. (2019). "A dataflow-driven approach to identifying microservices from monolithic applications." Journal of Systems and Software. Volume 157. Available: <https://doi.org/10.1016/j.jss.2019.07.008>. [Accessed: 27-Oct-2022]
- [15] Selmadji, A. et al. (2020) "From monolithic architecture style to Microservice one based on a semi-automatic approach," 2020 IEEE International Conference on Software Architecture (ICSA) [Preprint]. Available at: <https://doi.org/10.1109/icsa47634.2020.00023>.
- [16] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. Safina, L. (2017). "Microservices: Yesterday, Today, and Tomorrow". Present and Ulterior Software Engineering, 195-216.

## LAMPIRAN A LAMPIRAN A

ASJDBAKJSDBKA

Tabel A-1 *Lorem ipsum*

No	<i>Dolor sit amet</i>	At sonet	Vim commune	At quo congue	Cum iisque
1	Ei utroque electram	0	0	10	Laudem
2	Ei utroque electram	3	0	7	Laudem
3	Ei utroque electram	2	0	8	Laudem
4	Ei utroque electram	0	3	7	Laudem
5	Ei utroque electram	10	0	0	Laudem
6	Ei utroque electram	0	0	10	Laudem
7	Ei utroque electram	3	0	7	Laudem
8	Ei utroque electram	2	0	8	Laudem
9	Ei utroque electram	0	3	7	Laudem
10	Ei utroque electram	10	0	0	Laudem
11	Ei utroque electram	0	0	10	Laudem
12	Ei utroque electram	3	0	7	Laudem
13	Ei utroque electram	2	0	8	Laudem
14	Ei utroque electram	0	3	7	Laudem
15	Ei utroque electram	10	0	0	Laudem
16	Ei utroque electram	0	0	10	Laudem
17	Ei utroque electram	3	0	7	Laudem
18	Ei utroque electram	2	0	8	Laudem
19	Ei utroque electram	0	3	7	Laudem
20	Ei utroque electram	10	0	0	Laudem
21	Ei utroque electram	0	0	10	Laudem
22	Ei utroque electram	3	0	7	Laudem
23	Ei utroque electram	2	0	8	Laudem
24	Ei utroque electram	0	3	7	Laudem
25	Ei utroque electram	10	0	0	Laudem

**Tabel A-1** *Lorem ipsum*

<b>No</b>	<b><i>Dolor sit amet</i></b>	<b>At sonet</b>	<b>Vim commune</b>	<b>At quo congue</b>	<b>Cum iisque</b>
26	Ei utroque electram	0	0	10	Laudem
27	Ei utroque electram	3	0	7	Laudem
28	Ei utroque electram	2	0	8	Laudem
29	Ei utroque electram	0	3	7	Laudem
30	Ei utroque electram	10	0	0	Laudem
31	Ei utroque electram	0	0	10	Laudem
32	Ei utroque electram	3	0	7	Laudem
33	Ei utroque electram	2	0	8	Laudem
34	Ei utroque electram	0	3	7	Laudem
35	Ei utroque electram	10	0	0	Laudem
36	Ei utroque electram	0	0	10	Laudem
37	Ei utroque electram	3	0	7	Laudem
38	Ei utroque electram	2	0	8	Laudem
39	Ei utroque electram	0	3	7	Laudem
40	Ei utroque electram	10	0	0	Laudem

## LAMPIRAN B DATASET HASIL KUISIONER 2

**Tabel B-1** *Lorem ipsum*

No	<i>Dolor sit amet</i>	At sonet	Vim commune	At quo congue	Cum iisque
1	Ei utroque electram	0	0	10	Laudem
2	Ei utroque electram	3	0	7	Laudem
3	Ei utroque electram	2	0	8	Laudem
4	Ei utroque electram	0	3	7	Laudem
5	Ei utroque electram	10	0	0	Laudem
6	Ei utroque electram	0	0	10	Laudem
7	Ei utroque electram	3	0	7	Laudem
8	Ei utroque electram	2	0	8	Laudem
9	Ei utroque electram	0	3	7	Laudem
10	Ei utroque electram	10	0	0	Laudem
11	Ei utroque electram	0	0	10	Laudem
12	Ei utroque electram	3	0	7	Laudem
13	Ei utroque electram	2	0	8	Laudem
14	Ei utroque electram	0	3	7	Laudem
15	Ei utroque electram	10	0	0	Laudem
16	Ei utroque electram	0	0	10	Laudem
17	Ei utroque electram	3	0	7	Laudem
18	Ei utroque electram	2	0	8	Laudem
19	Ei utroque electram	0	3	7	Laudem
20	Ei utroque electram	10	0	0	Laudem
21	Ei utroque electram	0	0	10	Laudem
22	Ei utroque electram	3	0	7	Laudem
23	Ei utroque electram	2	0	8	Laudem
24	Ei utroque electram	0	3	7	Laudem
25	Ei utroque electram	10	0	0	Laudem
26	Ei utroque electram	0	0	10	Laudem

**Tabel B-1** *Lorem ipsum*

No	<i>Dolor sit amet</i>	At sonet	Vim commune	At quo congue	Cum iisque
27	Ei utroque electram	3	0	7	Laudem
28	Ei utroque electram	2	0	8	Laudem
29	Ei utroque electram	0	3	7	Laudem
30	Ei utroque electram	10	0	0	Laudem
31	Ei utroque electram	0	0	10	Laudem
32	Ei utroque electram	3	0	7	Laudem
33	Ei utroque electram	2	0	8	Laudem
34	Ei utroque electram	0	3	7	Laudem
35	Ei utroque electram	10	0	0	Laudem
36	Ei utroque electram	0	0	10	Laudem
37	Ei utroque electram	3	0	7	Laudem
38	Ei utroque electram	2	0	8	Laudem
39	Ei utroque electram	0	3	7	Laudem
40	Ei utroque electram	10	0	0	Laudem