

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

**TUGAS AKHIR**

**Albertus Septian Angkuw  
1119002**



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

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

**TUGAS AKHIR**

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

**Albertus Septian Angkuw  
1119002**



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

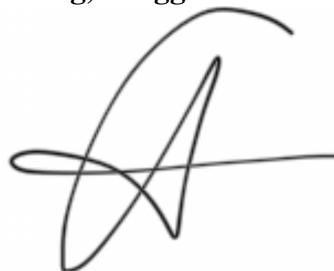
## **HALAMAN PERNYATAAN ORISINALITAS**

**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 MONOLITIK 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:

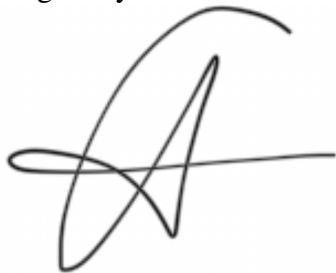
JUDUL TUGAS AKHIR

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 inductum 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 Monolitik . . . . .	2-1
2.1.1.1 Jenis Monolitik . . . . .	2-1
2.1.1.2 Keuntungan . . . . .	2-2
2.1.1.3 Tantangan . . . . .	2-3
2.1.2 <i>Microservice</i> . . . . .	2-3
2.1.2.1 Ciri Khusus <i>Microservice</i> . . . . .	2-4
2.1.2.2 Keuntungan . . . . .	2-5
2.1.2.3 Tantangan . . . . .	2-5
2.1.2.4 Permasalahan dan Pola penyelesaiannya . . . . .	2-6
2.1.3 <i>Enterprise Resource Planning</i> . . . . .	2-8

2.1.3.1	Arsitektur ERP . . . . .	2-9
2.1.4	Analisis Kode . . . . .	2-10
2.1.4.1	Anatomi . . . . .	2-10
2.1.4.2	Strategi Analisis . . . . .	2-10
2.1.4.3	Tantangan . . . . .	2-11
2.1.5	Dekomposisi . . . . .	2-12
2.1.5.1	Pemilihan Bagian yang didekomposisi . . . . .	2-12
2.1.5.2	Permasalahan dan Pola Penyelesaiannya . . . . .	2-14
2.1.5.3	Tantangan dan Hambatan . . . . .	2-16
2.1.6	<i>Clustering</i> . . . . .	2-17
2.1.6.1	<i>Distance</i> . . . . .	2-17
2.1.6.2	<i>Unsupervised Clustering</i> . . . . .	2-18
2.1.6.3	Pemilihan Partisi . . . . .	2-21
2.1.7	Teknologi dan <i>Library</i> . . . . .	2-22
2.1.7.1	<i>Docker</i> [15] . . . . .	2-22
2.1.7.2	<i>PyCG</i> [19] . . . . .	2-23
2.1.7.3	<i>Kong Gateway</i> [16] . . . . .	2-23
2.1.7.4	<i>inspect</i> [17] . . . . .	2-23
2.1.7.5	<i>SciPY</i> [18] . . . . .	2-24
2.2	Tinjauan Studi . . . . .	2-24
2.3	Tinjauan Objek . . . . .	2-27
2.3.1	Odoo . . . . .	2-27
<b>BAB 3</b>	<b>ANALISIS DAN PERANCANGAN SISTEM</b>	<b>3-1</b>
3.1	Analisis Masalah . . . . .	3-1
3.2	Kerangka Pemikiran . . . . .	3-2
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 . . . . .	3-4
3.3.1.3	Ekstraksi Dependency Module . . . . .	3-4
3.3.1.4	Pengabungan dan Optimisasi Hasil Ekstraksi . . . . .	3-5
3.3.1.5	Hierarchical Clustering . . . . .	3-5
3.3.1.6	Pemilihan Partisi . . . . .	3-8
3.3.2	Dekomposisi Monolitik ke Microservice . . . . .	3-9
3.3.2.1	Pemisahan User Interface . . . . .	3-9
3.3.2.2	Strategi Pemisahan Kode . . . . .	3-10
3.3.2.3	Komunikasi antar service . . . . .	3-13

3.3.2.4	Strategi Pemisahan <i>database</i>	3-13
3.3.3	<i>Deployment</i>	3-13
<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	Lingkungan Perangkat Lunak	4-1
4.2	Implementasi Perangkat Lunak	4-1
4.2.1	<i>Endpoint API</i>	4-1
4.2.2	Module <i>ERP</i>	4-1
4.2.3	Daftar <i>Class</i> dan Metode untuk <i>Clustering</i>	4-2
4.2.4	Alat Pendukung	4-3
4.3	Implementasi Migrasi Arsitektur	4-3
4.3.1	Prosess <i>Clustering</i>	4-3
4.3.2	Dekomposisi Monolitik ke <i>Microservice</i>	4-4
4.3.3	<i>Deployment</i>	4-4
4.4	Pengujian	4-4
4.4.1	Perancangan Pengujian	4-4
4.4.2	Performa	4-4
4.4.3	Biaya	4-4
<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	Daftar Metode dan Fungsi inspect . . . . .	2-23
2.2	Daftar Metode dan Fungsi SciPy . . . . .	2-24
2.3	<i>State of the Art</i> . . . . .	2-24
2.4	Komposisi dari Module pada aplikasi Odoo [14]: . . . . .	2-29
4.1	Daftar <i>method</i> pada <i>class helper</i> . . . . .	4-2
4.2	atribut pada <i>class</i> nama_class_1 . . . . .	4-4
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	Arsitektur <i>Single Process Monolith</i> [5]	2-2
2.2	Arsitektur <i>Microservice</i> [7]	2-4
2.3	Pola dalam menyelesaikan masalah di arsitektur <i>Microservice</i> [7]	2-6
2.4	Proses migrasi dari waktu ke waktu	2-14
2.5	Proses melakukan <i>Strangle Fig</i>	2-15
2.6	Ilustrasi Proses Branch By Abstraction	2-16
2.7	Hasil Hierarchical Clustering pada Objek	2-19
2.8	Perbedaan Agglomerative dan Divisive	2-19
2.9	Algoritma <i>Hierarchical Agglomerative Clustering</i> [20]	2-20
2.10	Hasil Partitional Clustering pada Objek dengan $n=3$	2-20
2.11	Arsitektur Odoo [14]	2-28
2.12	Struktur Repository Odoo	2-29
2.13	Skema Database Odoo	2-31
3.1	Kerangka Pemikiran	3-2
3.2	Diagram Flowchart Proses Global	3-3
3.3	Source Code Aplikasi Odoo pada git repository	3-3
3.4	Proses Pembuatan Call Graph dengan PyCG	3-4
3.5	Penggunaan 'inspect' untuk melihat object python lebih mendalam	3-5
3.6	Perhitungan Jarak Jaccard	3-6
3.7	Perhitungan Jarak Struktural	3-6
3.8	Hasil Akhir Jarak	3-6
3.9	Heatmap yang dihasilkan dari Distance Matrix	3-7
3.10	Dendogram Single Linkage	3-8
3.11	Dendogram Average Linkage	3-8
3.12	Dendogram Complete Linkage	3-8
3.13	Perbandingan dari nilai Cohesion dan nilai Coupling dengan jumlah <i>cluster/partisi</i>	3-9
3.14	Arsitektur UI Monolitik	3-10
3.15	Arsitektur UI di Microservice	3-10
3.16	Struktur Module dan Keterhubungannya di Aplikasi Monolitik	3-11
3.17	Penerapan Pola <i>Strangle</i> dan Branch by Abstraction	3-12
3.18	State Diagram pada proses login	3-12
3.19	Sequence Diagram pada kasus pengambilan data Product	3-13
3.20	Diagram <i>Deployment</i>	3-13

## **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) memiliki peran penting dalam industri dan bisnis saat ini. Tujuan dari implementasi ERP di perusahaan yaitu untuk meningkatkan efisiensi dengan cara mengintegrasikan dan mengotomatisasi aktivitas bisnisnya [1]. Selain itu diharapkan juga ERP memiliki skalabilitas dan fleksibilitas terhadap operasi bisnis baik yang diperlukan dalam jangka panjang atau jangka pendek, misalkan ketika perusahaan tumbuh dari waktu ke waktu, serta dalam waktu singkat ketika ada acara tertentu seperti di bulan Natal yang melibatkan volume bertransaksi tinggi [2].

Dalam membangun aplikasi ERP dapat dibangun dengan arsitektur seperti monolitik, *Service-Oriented Architecture* (SOA), dan *Microservice* (MSA) [4]. Implementasi arsitektur ERP mempengaruhi aspek manajemen di perusahaan seperti biaya, kompleksitas perawatan dan cara penggunaan aplikasi [1].

Arsitektur monolitik merupakan arsitektur paling sederhana dalam membangun aplikasi karena pengembangan yang mudah selama aplikasi berbentuk sederhana, walaupun demikian monolitik tidak mudah dilakukan *scaling* dan dikembangkan secara berkelanjutan. SOA bisa membantu menyelesaikan permasalahan tersebut dengan sistem terdistribusi akan tetapi SOA memiliki kekurangan sama seperti monolitik [4]. SOA sendiri dapat disamakan sebagai bentuk monolitik terdistribusi karena sistem memiliki banyak *service* tapi seluruh sistem harus dilakukan *deployment* bersamaan. Monolitik terdistribusi memiliki *coupling* yang tinggi dan bila dilakukan perubahan pada satu bagian dapat menyebabkan kerusakan pada bagian lain [5].

*Microservice* merupakan arsitektur modern yang cocok pada aplikasi perusahaan yang telah tumbuh dengan skala secara vertikal maupun horizontal, terdistribusi, dikembangkan secara berkelanjutan. Pada hal performa aplikasi yang dibangun dengan arsitektur *microservice* memiliki performa yang lebih baik dari arsitektur monolitik karena berdasarkan hasil *load test* yang dilakukan antar arsitektur menunjukkan *microservice* dapat memberikan latensi yang rendah dan *throughput* yang tinggi. Akan tetapi perlu diketahui bahwa tidak semua perusahaan harus menggunakan arsitektur *microservice* bila hanya memiliki sumber daya yang kecil, aplikasi berbentuk sederhana dan tidak memiliki masalah dengan performa yang lambat pada aplikasi [4].



Manfaat dari *microservice* membuat banyak perusahaan melakukan migrasi aplikasi berarsitektur monolitik menjadi arsitektur *microservice* seperti Netflix, eBay, Amazon, IBM, dan lainnya. Namun proses perubahan ini terbukti sulit dan mahal, salah satu tantangan terbesar yang harus dihadapi adalah bagaimana mengidentifikasi dan membagi komponen dari aplikasi monolitik. Komponen aplikasi ini kerap kali sangat *cohesif* dan *coupled* karena sifat desain arsitektur monolitik [10].

Untuk itu ada beberapa pendekatan untuk membagi komponen atau bisa disebut dekomposisi yaitu secara manual atau secara semi-otomatis [12]. Pembagian secara manual dapat menggunakan konsep *Domain Driven Design*, dekomposisi berdasarkan kemampuan bisnis, dan menggunakan pendekatan campuran lainnya [3]. Pendekatan dekomposisi dengan cara manual tidak mudah karena mudah terjadi kesalahan dan membutuhkan banyak waktu [12]. Oleh sebab itu dikembangkan otomatisasi untuk dapat mengenali komponen dari aplikasi monolitik untuk membentuk *microservice*. Dengan demikian masalah ini dapat diselesaikan dengan menggunakan algoritma *clustering*. Sebelum melakukan pengelompokan yaitu membuat *call graph* yang mengkodekan interaksi antara *class* dari kode program. *Graph* tersebut diolah menjadi matriks kemiripan sebelum dimasukkan ke dalam algoritma *clustering* [10].

Algoritma *clustering* yang umum digunakan dan terbukti dapat melakukan modularisasi pada perangkat lunak yaitu *Hierarchical Clustering*. *Hierarchical Clustering* memiliki kompleksitas waktu yang lebih sedikit dibandingkan algoritma lainnya seperti algoritma *hill-climbing* dan algoritma genetik. *Hierarchical Clustering* mengelompokkan objek yang memiliki kesamaan ke dalam suatu partisi(*cluster*) [11].

Pada penelitian ini akan melakukan dekomposisi aplikasi ERP yang memiliki arsitektur monolitik menjadi arsitektur *microservice* dengan pendekatan menganalisis *graph* yang dihasilkan dari kode program kemudian dilakukan pengelompokan secara *Hierarchical Clustering*. Hasil dari pengelompokan akan diimplementasikan dan dilakukan *load test* sehingga dapat diketahui latensi, jumlah *throughput*, dan penggunaan sumber daya. 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 monolitik dan arsitektur *microservice* dalam kondisi beban yang tinggi?
3. Berapa besar penggunaan sumber daya dan skalabilitas aplikasi ERP yang digunakan pada arsitektur monolitik dibandingkan arsitektur *microservice*?

### 1.3 Tujuan Penelitian

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

1. Menerapkan dekomposisi aplikasi ERP monolitik ke *microservice* dengan pendekatan *Hierarchical Clustering*.
2. Membuat *microservice* yang memiliki nilai *coupling* rendah dan nilai *cohesion* tinggi.
3. Membandingkan performa dan sumber daya aplikasi ERP monolitik dengan *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 *Docker*
2. Aplikasi yang didekomposisi adalah aplikasi yang sudah dibangun sebelumnya dan disebarkan dengan arsitektur monolitik.
3. Perubahan arsitektur tidak dapat menjamin secara keseluruhan fungsionalitas dari aplikasi, karena keterbatasan waktu dan pengujian.
4. Hanya beberapa bagian pada aplikasi ERP yang dilakukan dekomposisi yaitu modul produk, modul *point of sale*, dan UI aplikasi (Modul Web)

### 1.5 Kontribusi Penelitian

Kontribusi yang diberikan pada penelitian ini adalah sebagai berikut.

1. Memberikan langkah dalam melakukan dekomposisi aplikasi monolitik 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 *cohesion* tinggi dan nilai *coupling* 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, jurnal, atau artikel daring mengenai arsitektur *microservice*, permasalahan pada aplikasi ERP dan dekomposisi monolitik 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 monolitik ke arsitektur *microservice* dengan pendekatan *Hierarchical Clustering*.

#### 4. Implementasi

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

#### 5. Pengujian

Pada tahap ini dilakukan pengujian pada aplikasi yang sudah di dekomposisi. Pengujian melalui uji beban akan dilakukan dengan perbandingan antara aplikasi monolitik 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 monolitik, arsitektur *microservice*, *hierarchical clustering*, dan dekomposisi.

#### BAB 3: ANALISIS DAN PERANCANGAN

Analisis dan Perancangan yang berisi tahapan penerapan dekomposisi aplikasi monolitik 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 menyimulasikan 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 Monolitik

Monolitik yaitu suatu cara untuk melakukan penyebaran. Ketika semua fungsi dalam sistem harus disebarkan secara bersama-sama, maka itu merupakan sebuah monolitik [5]. Monolitik merupakan sebuah aplikasi perangkat lunak di mana setiap modulnya tidak bisa dieksekusi secara independen. Hal ini membuat monolitik sulit digunakan pada sistem terdistribusi tanpa bantuan penggunaan *frameworks* atau solusi *ad hoc* seperti Objek Jaringan, *RMI* atau *CORBA* [6].

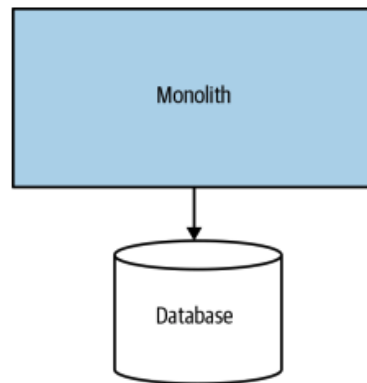
Penggunaan pada bahasa pemrograman seperti *Java*, *C/C++*, dan *Python* pada pengembangan aplikasi di sisi *server*, memiliki kemampuan dalam melakukan abstraksi untuk memecah kompleksitas program menjadi berupa modul. Namun, bahasa pemrograman ini dirancang untuk membuat *artefacts* monolitik. Di mana abstraksi ini tergantung pada penggunaan berbagi sumber data pada komputer yang sama (memori, *database*, *file*) [6].

##### 2.1.1.1 Jenis Monolitik

Setidaknya terdapat 3 jenis monolitik yang memiliki struktur yang berbeda masing-masing namun masih merupakan arsitektur monolitik [5]:

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

Di mana 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 monolitik di mana setiap kode bisa bekerja secara independen tetapi perlu dijadikan satu kesatuan ketika ingin dilakukan *deployment*.



**Gambar 2.1** Arsitektur *Single Process Monolith* [5]

### 2. *Distributed Monolith*

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

Monolitik terdistribusi biasanya muncul di kondisi di mana tidak cukup fokus pada konsep *information hiding* dan *cohesion* dari fungsi bisnis. Akibatnya terbentuklah arsitektur yang memiliki *coupling* yang tinggi, di mana bisa perubahan menyebabkan kerusakan pada bagian sistem lain.

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

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

#### 2.1.1.2 Keuntungan

Masing-masing jenis monolitik memiliki keuntungan yang sama seperti [10, 7]:

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.

### 2.1.1.3 Tantangan

Masing-masing jenis monolitik memiliki tantangan yang sama seperti [10, 7]:

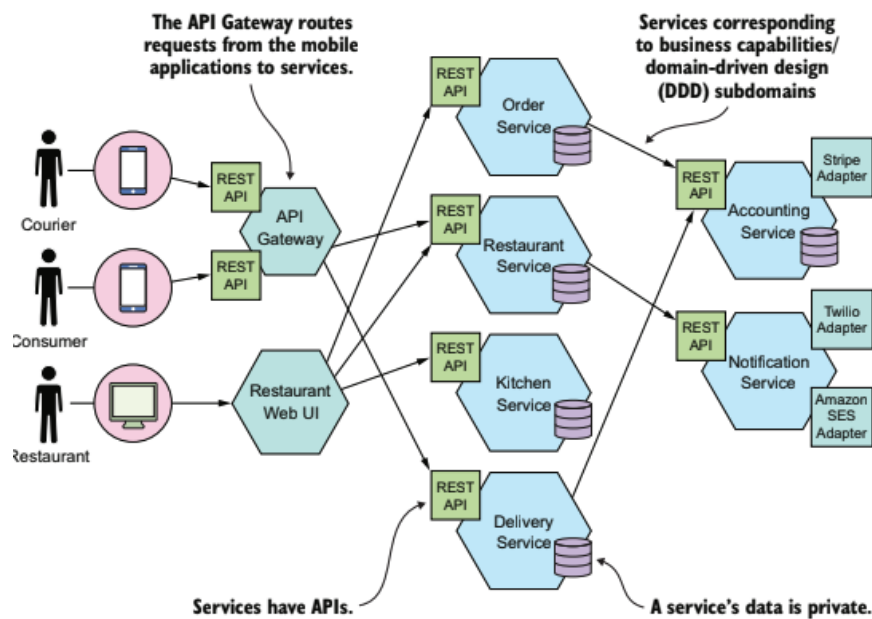
1. Sulit dikembangkan secara berkelanjutan, karena semakin banyak orang yang bekerja pada aplikasi yang sama. Akibatnya setiap pengembang memiliki kepentingan masing-masing dalam mengelola kode yang sama dan membuat pengambilan keputusan sulit serta tidak fleksibel
2. Memiliki reliabilitas yang rendah, karena kesalahan pada salah satu modul 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 modul penyediaan data yang membutuhkan banyak memori sedangkan modul pemrosesan gambar membutuhkan banyak CPU, karena modul ini berada pada aplikasi yang sama akibatnya pengembang harus melakukan pengorbanan pada salah satu sisi sumber daya.
4. Terkunci pada teknologi lama, 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 *dideploy* 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 [5].

*Service* adalah komponen perangkat lunak yang memiliki kegunaannya secara khusus di mana 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 *query*. *API* terdiri dari perintah, *query* dan *event*. Perintah dapat berupa *buatPesanan()* yang melakukan aksi dan memperbarui data. *Query* dapat berupa *cariPesananBerdasarkanID()* yang digunakan untuk mengambil data. *Service* juga dapat membuat suatu *event* seperti *PesananSudahDibuat* di mana *event* ini akan dikonsumsi oleh *client*-nya / *subscriber* [7].



Gambar 2.2 Arsitektur *Microservice* [7]

*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. Di mana *API* akan diimplementasi melalui adapter yang berinteraksi dengan logika bisnis [7]

#### 2.1.2.1 Ciri Khusus *Microservice*

Hal yang membedakan arsitektur *microservice* dengan arsitektur lainnya yaitu [10, 5, 7]:

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 di mana bisa memiliki sistem operasi hingga komputer yang berbeda. Dengan ini sistem terdistribusi lebih sederhana dan nilai *coupling* 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 di luar 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

#### 2.1.2.2 Keuntungan

Keuntungan dari menggunakan arsitektur *Microservice* [10, 5, 7]:

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

*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 *scaling* secara independen

Setiap *service* memiliki fungsi yang berfokus pada satu hal, di mana 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.

#### 2.1.2.3 Tantangan

Tantangan dari menggunakan arsitektur *Microservice* [10, 5, 7]:

#### 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. Di mana *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

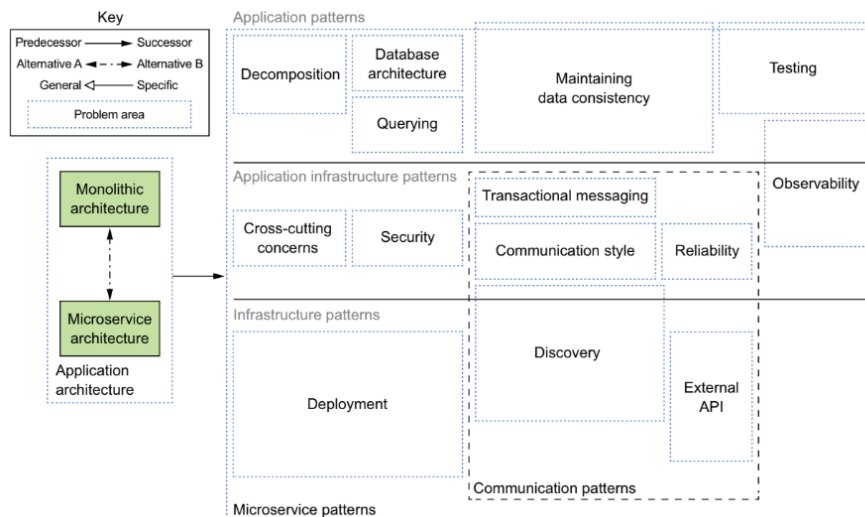
Setiap *service* untuk berkomunikasi antar *service* memiliki tantangan masing-masing seperti latensi, konsistensi data, dan kondisi ketika beberapa *service* mengalami kegagalan. *Microservice* juga meningkatkan kompleksitas operasional oleh karena itu untuk melakukan *deployment* sebaiknya menggunakan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

#### 2.1.2.4 Permasalahan dan Pola penyelesaiannya

Arsitektur *Microservice* memiliki banyak hal yang harus dipertimbangkan oleh karena itu dibutuhkan strategi dan pola untuk menyelesaikan suatu masalah yang dapat terjadi dalam penerapan arsitektur *microservice* [7]:



**Gambar 2.3** Pola dalam menyelesaikan masalah di arsitektur *Microservice* [7]

### 1. *Application patterns*

Permasalahan yang harus diselesaikan oleh pengembang aplikasi, yaitu bagaimana cara dekomposisi sistem menjadi beberapa *service*. Terdapat

beberapa Strategi yang dapat dilakukan seperti berdasarkan sub-domain dan berdasarkan proses bisnis. Service mengelola datanya masing-masing namun ini menyebabkan permasalahan tersendiri karena bisa terjadi data yang tidak konsisten antara *service*. Pendekatan biasa seperti 2PC tidak cocok pada aplikasi modern sehingga konsistensi data dicapai melalui SAGA.

Perbedaan penyimpanan data membuat *query* harus menggabungkan data yang dimiliki oleh beberapa *service* yang terlibat karena data hanya bisa diakses melalui API. Terkadang bisa digunakan komposisi API di mana memanggil API *service* satu dengan yang lain atau dengan *Command Query Responsibility Segregation(CQRS)* yaitu ketika setiap *service* memiliki replika data dari *service* yang dibutuhkan.

Pengujian pada *service* mudah dilakukan karena memiliki lingkup yang kecil dan terpusat namun untuk menguji beberapa *service* tidak mudah, karena banyaknya proses yang harus dilakukan. Sehingga diperlukan proses otomatisasi proses pengujian. Ada beberapa pola yang dapat digunakan untuk pengujian di *microservice* yaitu pengujian dilakukan pada *client*, pengujian pencocokan pada *client*, dan pengujian secara terisolasi.

### 2. Application infrastructure

Permasalahan infrastruktur yang memiliki pengaruh pada proses pengembangan aplikasi. Aplikasi yang dibangun dengan *microservice* merupakan sistem terdistribusi. Sehingga komunikasi antar proses adalah bagian yang penting dalam arsitektur *microservice*. Diperlukan variasi arsitektur dan keputusan desain tentang bagaimana *service* berkomunikasi satu dengan yang lain.

Pola komunikasi dikelompokkan menjadi 5 grup yaitu gaya komunikasi, *discovery* reliabilitas, *Transactional Messaging*, API Eksternal. Terdapat 3 gaya komunikasi yaitu *Messaging* di mana komunikasi dapat dilakukan secara *asynchronous*, *Domain-Specific* di mana komunikasi harus melalui protokol tertentu seperti Email, dan *Remote Procedure Invocation(RPI)* di mana komunikasi dilakukan secara *asynchronous*.

Cara *Messaging* memiliki kelemahan karena transaksi terdistribusi tidak cocok digunakan pada aplikasi modern untuk mengatasi ini ada 2 pendekatan yaitu *polling publisher* di mana menggunakan tabel OUTBOX untuk menyimpan message sementara dan Log Tailing di mana melihat

transaksi terakhir dari message.

Ketika *service* sedang berkomunikasi dan waktu menunggu jawaban dari *service* lain melebihi batas yang ditentukan maka bisa terjadi kemungkinan *service* tersebut mengalami kegagalan. Pola Circuit Breaker dapat diterapkan bila terjadi hal seperti ini tujuannya agar *service* tidak berkomunikasi pada *service* yang gagal.

Pada arsitektur *microservice* untuk proses autentikasi pengguna umumnya dilakukan oleh *API Gateway*. Di mana *API Gateway* melanjutkan informasi ke *service* yang bertanggung jawab mengenai autentikasi, solusi umumnya yaitu menggunakan Access Token seperti JWT(JSON Web Token).

### 3. *Infrastructure patterns*

Permasalahan infrastruktur yang muncul diluar dari pengembangan aplikasi. Infrastruktur menangani proses *deployment*, *discovery*, dan External API. *Discovery* adalah bagaimana *service* bisa ditemukan agar bisa berkomunikasi, terdapat beberapa pendekatan yang bisa dilakukan seperti *service* ditemukan dari *client* atau dari server dan proses registrasi bisa dilakukan secara sendiri atau melalui pihak ke-3.

Eksternal API adalah bagaimana aplikasi pengguna berinteraksi dengan *service*. Ada 2 cara untuk aplikasi berinteraksi yaitu *API Gateway* dan *Backend for Frontend*. Proses *Deployment microservice* memiliki proses yang kompleks karena *service* yang dikelola bisa berjumlah 10 hingga ratusan *service*, sehingga diperlukan proses otomatisasi yang bisa mengelola *service* dan proses *scaling* bisa diatur berdasarkan kebutuhan. Cara melakukan *deployment* bisa dengan *container*, *virtual machine*, *serverless*, atau menggunakan *platform deployment*

### 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* di mana semua transaksi bisnis dapat direkam, diproses, dipantau dan dilaporkan. Tujuannya agar proses bisnis bisa dilakukan dengan lebih cepat, murah, dan

transparan [2].

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* [2].

### 2.1.3.1 Arsitektur ERP

Arsitektur dari sistem ERP dapat didefinisikan menjadi 2 tipe yaitu *logical* dan *physical*. Arsitektur *logical* menggambarkan bagaimana sistem diorganisasikan untuk mendukung kebutuhan bisnis sedangkan arsitektur *physical* mengenai bagian komponen fisik dari keseluruhan sistem untuk melihat performa dan mengurangi biaya. Berikut adalah arsitektur *physical* [2]

#### 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 ke dalam masing-masing lapisan seperti lapisan *user interface*, lapisan aplikasi dan lapisan *database* / penyimpanan data.

#### 2. *Web-based*

Arsitektur *Web-based* mengadopsi teknologi berorientasi objek web di mana 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 di mana 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 artefak. 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 [8].

#### 2.1.4.1 Anatomi

Anatomi Analisis Kode yaitu tahapan dan langkah yang harus dilakukan untuk menemukan hasil analisis kode [8]:

1. Ekstraksi Data

Proses ini adalah proses pertama kali dilakukan sebelum melakukan analisis kode, data yang diekstraksi berasal dari kode program. Umumnya dilakukan dengan *syntactic analyzer* atau *parser*. Proses *Parser* ini mengonversi 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.

#### 2.1.4.2 Strategi Analisis

Terdapat berbagai cara dan pertimbangan dalam melakukan analisis kode [8]:

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 di mana terdapat suatu abstraksi.

#### 2.1.4.3 Tantangan

Untuk melakukan kode analisis terdapat tantangan dan kesulitan untuk mendapatkan hasil analisis yang sempurna [8]:

#### 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. Di mana 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 Dekomposisi

Dekomposisi merupakan proses pemisahan bagian dari aplikasi, proses dekomposisi sendiri dapat menyebabkan masalah dengan latensi, penyelesaian masalah, integritas, kegagalan bersamaan, dan hal lainnya.

#### 2.1.5.1 Pemilihan Bagian yang didekomposisi

Terdapat beberapa cara untuk melakukan dekomposisi aplikasi monolitik, dekomposisi ini menentukan bagaimana bagian aplikasi menjadi *Service*:

#### 1. Kemampuan Bisnis[7]

Service dibuat dari pendekatan proses arsitektur bisnis di mana setiap kegiatan bisnis memiliki ketergantungan terhadap kegiatan bisnis lainnya. Contohnya Toko Online memiliki hubungan dengan proses pemesanan, penyimpanan barang, pengiriman dan lainnya. Untuk menemukan kemampuan bisnis bisa dianalisis dari tujuan organisasi, struktur organisasi dan proses bisnisnya. Setiap kemampuan bisnis bisa dianggap sebagai suatu *service*.

Kemampuan bisnis juga sering berfokus pada objek bisnis, seperti berfokus pada setiap hal masukan, hasil, dan perjanjian. Kemampuan bisnis bisa memiliki bagian kecil lainnya. bagian kecil lainnya terkadang bisa merepresentasikan hal yang sangat berbeda.

#### 2. *Domain Driver Design* (DDD) [7]

DDD mengambil konsep mencari domain di mana domain tersebut dapat digunakan untuk menyelesaikan permasalahan di dalam domain itu sendiri. Model domain hampir mencerminkan antara desain dan implementasi aplikasi. DDD memiliki 2 konsep yang sangat penting saat mengimplementasikan di arsitektur *microservice* yaitu subdomain dan *bounded context*.

DDD memisahkan domain model untuk setiap subdomainnya, subdomain



adalah bagian dari domain. Subdomain diidentifikasi melalui pendekatan yang sama dengan mencari kemampuan bisnis. DDD menggunakan *bounded context* dalam menentukan batasan suatu domain, *bounded context* termasuk kode yang mengimplementasikan model. Pada *microservice bounded context* bisa berupa satu *service* atau beberapa kumpulan *service*.

### 3. Analisis Kode [5, 12]

Transformasi aplikasi monolitik menjadi *microservice* bisa dilakukan dengan strategi analisis statik atau dinamis. Umumnya hal yang diperhatikan adalah ketergantungan / keterhubungan antar module, kemudian menerapkan proses clustering atau algoritma evolusi pada ketergantungan module untuk menghasilkan partisi-partisi module yang sudah ditentukan dari evaluasi tertentu seperti nilai *coupling* yang rendah dan nilai *cohesion* yang tinggi. Analisis ini sendiri bisa berupa campuran antara proses bisnis dekomposisi secara fungsional, *control flow*, *data flow*, dan *semantic model*

Untuk menentukan batasan *microservice* perlu diketahui bagaimana struktur kode mempengaruhi secara *coupling* dan *cohesion*. *Coupling* berfokus pada bagaimana perubahan pada satu hal membuat bagian lain mengalami perubahan. *Cohesion* berfokus pada bagaimana kode dikelompokkan satu sama lain [5].

Struktur *Microservice* yang ideal memiliki *cohesion* yang tinggi dan *coupling* yang rendah, karena dengan *cohesion* yang tinggi setiap *service* memiliki fokusnya masing-masing dan perubahan dilakukan pada spesifik *service* sedangkan dengan *coupling* rendah membuat *service* bisa berdiri sendiri / independen dan setiap *service* mungkin tidak harus sering berinteraksi satu sama lain [5].

*Coupling* merupakan tentang bagaimana bagian yang terhubung memiliki dampak satu sama lain, ketika satu *service* berubah dan *service* itu dihubungkan dengan banyak *service* maka *service* lainnya harus beradaptasi terhadap perubahan tersebut. Terdapat beberapa tipe *coupling* seperti *Logical Coupling*, *Temporal Coupling*, *Deployment Coupling*, dan *Domain Coupling*.

*Cohesion* merupakan tentang bagaimana kode yang dikelompokkan bersama adalah kode yang memiliki keterhubungan kuat. Sehingga ketika terjadi perubahan pada satu bagian, bagian yang lain diubah bersama-sama. Pengelompokan kode yang tepat dapat membantu pengembang untuk melakukan perubahan ketika diperlukan tanpa harus mengganggu stabilitas sistem secara

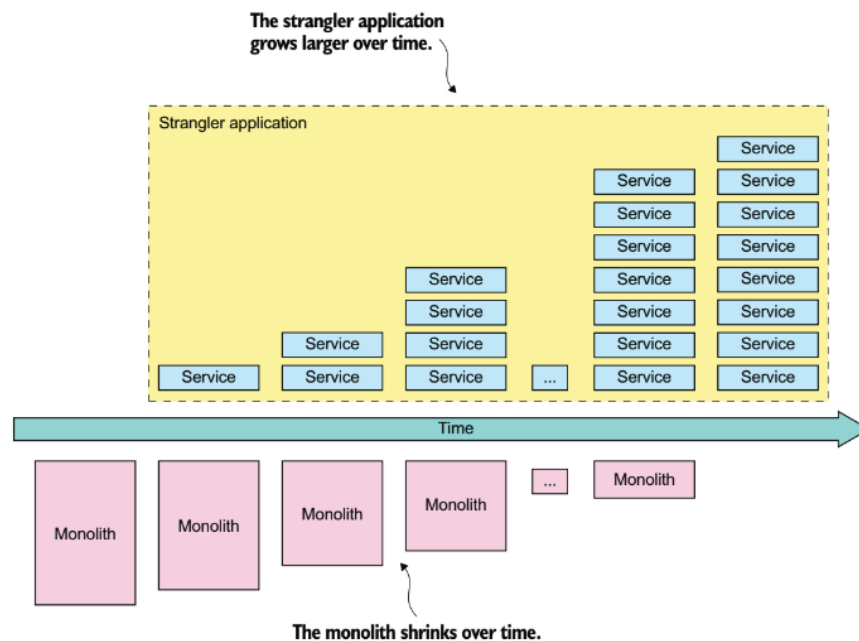
keseluruhan.

### 2.1.5.2 Permasalahan dan Pola Penyelesaiannya

Ketika sudah menentukan *service* yang ingin didekomposisi diperlukan pola dan strategi untuk memisahkan bagian yang ingin didekomposisi. Setiap pola memiliki kekurangan dan kelebihan masing-masing [5]:

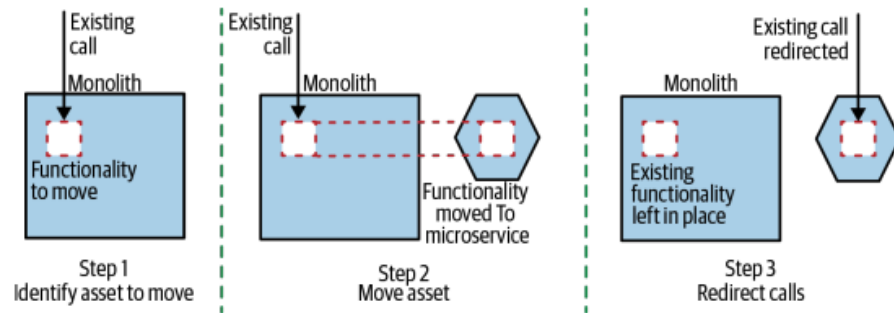
#### 1. Pola *Strangle Fig*

Pola ini terinspirasi cabang yang bisa berdiri sendiri pada pohon, cabang ini awal mulanya adalah bagian besar dari pohon yang lama kelamaan membentuk akarnya sendiri sehingga bisa mendukung kebutuhannya sendiri tanpa harus bergantung pada pohon yang lama. Ide ini pada pengembangan perangkat lunak yaitu bahwa aplikasi dahulu tetap bisa berjalan bersamaan dengan aplikasi baru. Aplikasi baru akan tumbuh dan mengambil alih aplikasi dahulu tersebut.



**Gambar 2.4** Proses migrasi dari waktu ke waktu

Terdapat 3 tahap utama dalam menerapkan *strangle* yaitu memilih bagian yang ingin dipindah, memindahkan bagian tersebut menjadi *service* sendiri, dan mengalihkan panggilan pada bagian itu ke *service* yang baru dibuat. Apabila terjadi kegagalan selama migrasi maka sistem bisa dikembalikan seperti semula. Penerapan *strangle* bisa dilakukan untuk memindahkan fitur lama atau pun menambahkan fitur baru. Untuk mengalihkan panggilan *service* dapat dilakukan melalui *proxy* yang akan merutekan ke *service* yang dapat menangani panggilan tersebut.



**Gambar 2.5** Proses melakukan *Strangle Fig*

## 2. Pola UI Composition

Pola ini digunakan pada User Interface(UI), di mana pemecahan dilakukan pada sisi tampilan aplikasi(UI). User Interface akan memanggil beberapa *service* yang dibutuhkannya. Terdapat beberapa pendekatan dalam pemecahan disisi UI yaitu Page Composition, Widget Composition, dan Micro Frontends. Penggunaan pola membutuhkan kode aplikasi user interface bisa dimodifikasi umumnya teknik ini tergantung dari teknologi yang mengimplementasinya.

## 3. Branch By Abstraction

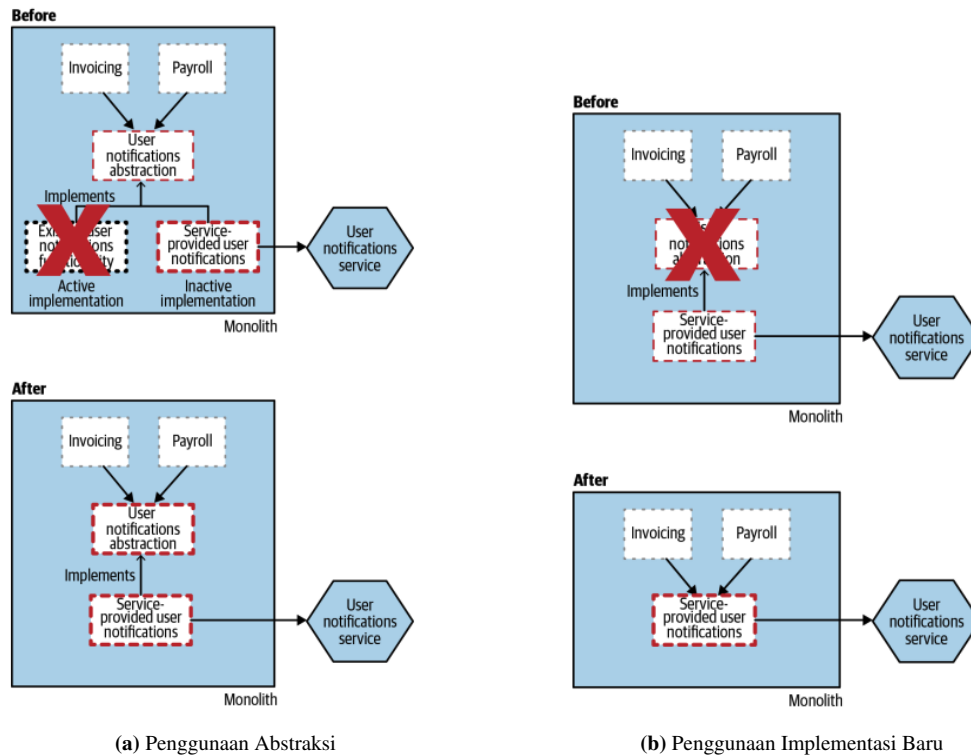
Pola *Strangle Fig* dapat dilakukan untuk memindahkan fungsionalitas namun ketika fungsionalitas itu berada di dalam sistem yang lebih dalam. Maka proses ekstraksi harus dilakukan tanpa mempengaruhi banyak sistem di mana sistem lain bisa berubah tanpa diketahui sistem yang diekstraksi.

Branch By Abstraction memiliki 5 tahap yaitu: membuat abstraksi dari fungsi yang akan diubah, mengubah pengguna yang menggunakan fungsi dengan abstraksi yang baru, membuat implementasi baru dari abstraksi, mengubah abstraksi untuk menggunakan implementasi yang baru, membersihkan abstraksi dan menghapus implementasi yang dahulu.

## 4. Parallel Run

Parallel Run adalah pola bagaimana mengeksekusi sistem yang baru dan sistem yang lama ketika dipisahkan. Teknik ini dapat membuat 2 fungsionalitas di antara sistem baru dan sistem lama dapat berjalan bersama-sama. Hasil dari sistem bisa digunakan sebagai acuan atau verifikasi bahwa sistem baru dapat berjalan dengan benar dan dapat menggantikan sistem lama. Penggunaan metode ini jarang digunakan karena biasanya digunakan pada kasus fungsi yang memiliki risiko tinggi.

## 5. Decorating Collaborator



**Gambar 2.6** Ilustrasi Proses Branch By Abstraction

Pola ini digunakan ketika dibutuhkan proses berdasarkan apa yang terjadi di dalam monolitik, tapi pengembang tidak bisa mengubah monolitik itu sendiri. Pola Decorator dapat menambahkan fungsionalitas pada sistem di mana sistem itu sendiri tidak sadar mengenai fungsionalitas tambahan tersebut. Caranya yaitu panggilan langsung ke aplikasi monolitik dan tidak perlu dialihkan tetapi hasil / *response* aplikasi monolitik dialihkan ke *service* yang dapat mengolaborasi hasil tersebut.

#### 6. Change Data Capture

Change Data Capture tidak menangkap panggilan dan bertindak pada panggilan yang menuju ke monolitik tetapi bereaksi dari perubahan yang terjadi pada penyimpanan data. Untuk mengimplementasikannya yaitu dengan *Database Triggers*, *Transaction Log Pollers*, dan *Batch Delta Copier*. Pola ini dapat digunakan ketika ingin melakukan replikasi data atau proses migrasi data.

#### 2.1.5.3 Tantangan dan Hambatan

Terdapat tantangan dan hambatan ketika melakukan proses dekomposisi. Tantangan dan hambatan dapat terjadi karena proses dekomposisi itu sendiri [7]:

##### 1. Latensi Jaringan

Latensi jaringan merupakan hal yang dikhawatirkan pada sistem terdistribusi. Beberapa dekomposisi pada *service* dapat menimbulkan tinggi latensi antara dua *service*. Solusi untuk mengatasi permasalahan ini yaitu dengan menggabungkan kedua *service* tersebut atau mengganti proses komunikasi antar *service* tersebut.

### 2. Menjaga konsistensi data antar service

Data yang sebelumnya berada di satu sistem, setelah didekomposisi data tersebar di beberapa *service*. Sehingga proses pengaksesan dan perubahan data lebih rumit. Pada kasus transaksi yang membutuhkan ACID (Atomicity, Consistency, Isolation, and Durability) *microservice* tidak memiliki isolasi karena proses transaksi terjadi tidak hanya di satu *service*.

### 3. Adanya God Class yang mencegah dekomposisi

God Class adalah *class* yang berukuran besar dan berdampak secara luas diaplikasi. Class ini biasanya mempunyai hubungan dengan *class* lainnya dan mengelola berbagai aspek di aplikasi. Solusinya yaitu dengan membuat suatu library dari God Class tersebut dan memecah God Class menjadi *service* yang berfokus pada satu hal. Pendekatan lainnya yaitu dengan DDD di mana dibuat banyak domain dan subdomain.

#### 2.1.6 Clustering

*Clustering* yaitu suatu proses untuk melakukan pengelompokan 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 [9].

Metode Clustering membutuhkan adanya indeks kedekatan di antara object. indeks ini bisa dikomputasi dalam bentuk matrix. Hasil matrix kedekatan / distance matrix memiliki nilai kedekatan untuk setiap object terhadap object lainnya. Indeks kedekatan bisa berupa kemiripan atau ketidaksamaan. Semakin jauh nilai antar indeks maka semakin berbeda dua objek tersebut [9].

##### 2.1.6.1 Distance

Untuk membuat kedekatan antar objek perlu diketahui tipe data yang digunakan agar kedekatan yang dihasilkan relevan. Berikut adalah metode mencari nilai kedekatan untuk pengelompokan objek:

#### 1. Jaccard Coefficient [9]

Untuk mendapatkan kedekatan dengan menghitung total hal yang sama ( $a_{11}$ )

atau terhubung di antara object kemudian dibagi dengan jumlah hal yang berbeda( $a_{01}/a_{10}$ ) di antara dua objek( $i,k$ ) tersebut.

$$d(i,k) = \frac{a_{11}}{a_{11} + a_{01} + a_{10}} = \frac{a_{11}}{d - a_{10}} \quad (2.1)$$

## 2. *Structural Similarity* [10]

Kedekatan ditentukan dari jumlah hubungan bersama di antara dua *class*, metode ini melihat ketergantungan antara dua *class*( $c_i, c_j$ ) tersebut. Tujuannya agar pengelompokan yang dihasilkan secara kompak. Structural Similarity mempertimbangkan arah hubungan seperti apakah hubungan itu masuk atau keluar.

$$SimStr_{c_i, c_j} = \begin{cases} \frac{1}{2} \left( \frac{calls(c_i, c_j)}{calls_{in}(c_j)} + \frac{calls(c_j, c_i)}{calls_{in}(c_i)} \right) & \text{if } calls_{in}(c_i) \neq 0 \text{ and } calls_{in}(c_j) \neq 0 \\ \frac{calls(c_i, c_j)}{calls_{in}(c_j)} & \text{if } calls_{in}(c_i) = 0 \text{ and } calls_{in}(c_j) \neq 0 \\ \frac{calls(c_j, c_i)}{calls_{in}(c_i)} & \text{if } calls_{in}(c_i) \neq 0 \text{ and } calls_{in}(c_j) = 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.2)$$

## 3. *Simple matching coefficient* [9]

Mirip seperti Jaccard tapi Simple Matching menghitung hal yang tidak sama, jumlah hal yang tidak sama dibandingkan dengan jumlah yang sama.

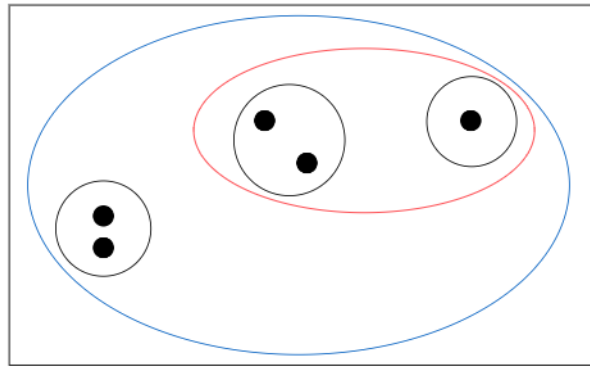
Kemudian dibagi jumlah hal yang tersedia pada objek tersebut.

### 2.1.6.2 *Unsupervised Clustering*

Unsupervise Clustering adalah pengelompokan di mana tidak diberikan sebuah label untuk mengetahui partisi objek sehingga pengelompokan hanya menggunakan nilai kedekatan antar objek. Terdapat 2 bentuk Unsupervised Clustering [9]:

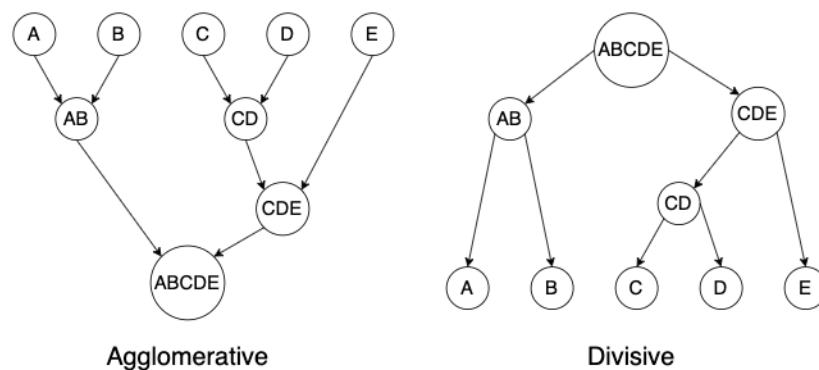
#### 1. *Hierarchical Clustering*

Metode *Hierarchical Clustering* adalah sebuah prosedur untuk mentranformasi sebuah *proximity matrix* menjadi beberapa partisi. Clustering adalah sebuah partisi di mana 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*. [9]



**Gambar 2.7** Hasil Hierarchical Clustering pada Objek

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 [9].



**Gambar 2.8** Perbedaan Agglomerative dan Divisive

Ada beberapa metode untuk menentukan partisi terdekat yaitu dengan menghitung jarak maksimal antara objek partisi (complete linkage), nilai rata-rata jarak (average linkage) atau jarak minimum (single linkage) [13]. Proses untuk menghitung *Hierarchical Agglomerative Clustering* (HAC) menghasilkan stepwise matrix yang dapat digunakan untuk membuat dendrogram. Proses ini menggunakan perbandingan jarak antar objek (X). Untuk menggabungkan hasil kumpulan jarak menggunakan algoritma linkage, hasil dari linkage digabungkan dan disimpan ke dalam array baru. Array yang baru (d) berisi jarak terbaru antar objek lainnya

yang belum digabungkan. Proses berakhir ketika semua objek sudah terhitung keterhubungannya dari individu objek yang memiliki banyak partisi hingga menjadi satu partisi yang memiliki banyak objek [20].

```

1: procedure PRIMITIVE_CLUSTERING( $S, d$ )  ▷  $S$ : node labels,  $d$ : pairwise dissimilarities
2:    $N \leftarrow |S|$   ▷ Number of input nodes
3:    $L \leftarrow []$   ▷ Output list
4:    $size[x] \leftarrow 1$  for all  $x \in S$ 
5:   for  $i \leftarrow 0, \dots, N - 2$  do
6:      $(a, b) \leftarrow \operatorname{argmin}_{(S \times S) \setminus \Delta} d$ 
7:     Append  $(a, b, d[a, b])$  to  $L$ .
8:      $S \leftarrow S \setminus \{a, b\}$ 
9:     Create a new node label  $n \notin S$ .
10:    Update  $d$  with the information
        
$$d[n, x] = d[x, n] = \text{FORMULA}(d[a, x], d[b, x], d[a, b], size[a], size[b], size[x])$$

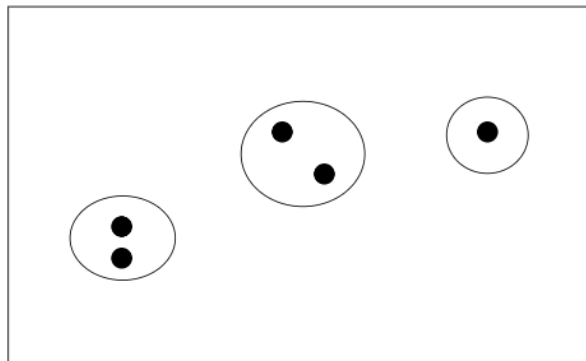
        for all  $x \in S$ .
11:     $size[n] \leftarrow size[a] + size[b]$ 
12:     $S \leftarrow S \cup \{n\}$ 
13:  end for
14:  return  $L$   ▷ the stepwise dendrogram, an  $((N - 1) \times 3)$ -matrix
15: end procedure
(As usual,  $\Delta$  denotes the diagonal in the Cartesian product  $S \times S$ .)

```

**Gambar 2.9** Algoritma *Hierarchical Agglomerative Clustering* [20]

## 2. *Partitional Clustering*

*Partitional* menggunakan pendekatan di mana 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.



**Gambar 2.10** Hasil *Partitional Clustering* pada Objek dengan  $n=3$

Di mana 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.

### 2.1.6.3 Pemilihan Partisi

Untuk mengetahui apakah pengelompokan yang dihasilkan dari proses *clustering* memiliki kualitas yang tepat maka perlu dilakukan evaluasi yang mempertimbangkan aspek *microservice*.

#### 1. *Structural and Behavioral Dependencies* [12]

Microservice dapat dilihat sebagai kumpulan dari suatu *class* yang berkolaborasi satu sama lain untuk memberikan suatu fungsi. Hal ini dapat ditentukan dari kode program melalui nilai internal *coupling*. Kolaborasi bisa ditentukan dengan nilai *cohesion* dari jumlah data seperti atribut yang tidak tetap.

$$FOne(M) = \frac{1}{2}(InterCoup(M) + InterCoh(M)) \quad (2.3)$$

Nilai Internal Coupling dapat dihitung dari jumlah koneksi secara langsung atau tidak langsung melalui ketergantungan di antara *class*(CoupP) dengan jumlah kemungkinan koneksi yang bisa terjadi di antara *class* (NbPossiblePairs). Ketika dua *class* semakin banyak menggunakan fungsi satu sama yang lain, maka semakin tinggi nilai kopelnya.

$$InterCoup(M) = \frac{\sum CoupP(P)}{NbPossiblePairs} \quad (2.4)$$

Perbandingan nilai *coupling* antara dua *class* dihitung oleh fungsi CoupP, fungsi CoupP membagi jumlah *call* yang terjadi(NbCals) antara 2 *class*(C1,C2) dengan total *call* yang dilakukan *class* tersebut(TotalNbCalls).

$$CoupP(C1,C2) = \frac{NbCals(C1,C2) + NbCals(C2,C1)}{TotalNbCalls} \quad (2.5)$$

Perhitungan nilai *coupling* eksternal, dilakukan sama dengan perhitungan nilai *coupling* internal tapi yang membedakannya adalah hanya menghitung jumlah *call* kepada *class* eksternal.

Untuk menghitung nilai *cohesion* dapat dilakukan dengan menghitung jumlah interaksi *class* di dalam partisi. Perhitungan ini dapat dihitung

dengan fungsi *InterCoh*. *InterCoh* membagi antara jumlah *call* yang terjadi di dalam *class* (*NbDirectConnections*) dengan jumlah *call* yang hanya memanggil *class* di dalam partisinya sendiri (*NbPossibleConnections*).

$$InterCoh(M) = \frac{NbDirectConnections}{NbPossibleConnections} \quad (2.6)$$

### 2. *Data Autonomy Class* [12]

Salah satu karakteristik *microservice* adalah memiliki data *Autonomy*. Sebuah *microservice* dapat berdiri sendiri bila tidak memerlukan data dari *service* lainnya. Dengan demikian semakin kecil pertukaran data antar *service* maka semakin baik *microservice*. Perhitungan dilakukan dengan mengukur nilai ketergantungan data antara *class* dan ketergantungannya dengan *class* external.

### 3. *Iter-Paritition Call Percentage(ICP)* [10]

Menghitung jumlah persentase *call* secara runtime antara 2 partisi di *microservice*. Semakin kecil nilai ICP menunjukkan kurangnya interaksi antara partisi di mana merepresentasikan *microservice* yang bagus.

### 4. Jumlah Interface [10]

Jumlah Interface/ Interface Number(IFN) menghitung jumlah interface yang ada di dalam *microservice*. ifni adalah jumlah interface di dalam *microservice* dan N adalah total interface di *microservice*. Semakin kecil nilai IFN mengindikasikan rekomendasi *microservice* yang lebih baik

## 2.1.7 Teknologi dan *Library*

### 2.1.7.1 *Docker* [15]

Docker adalah sebuah platform terbuka untuk *container* yang dapat membuat *container* dan mengelola *container*. Docker dapat memisahkan aplikasi dengan infrastruktur sehingga pengembangan aplikasi bisa berjalan dengan cepat. Dengan Docker, pengguna dapat mengelola infrastruktur sama seperti mengelola aplikasi.

Arsitektur Docker menggunakan arsitektur *client-server*. Docker *client* berkomunikasi dengan Docker daemon, yang melakukan pekerjaan seperti membangun, menjalankan, dan mendistribusikan docker containers. Docker *client* dan daemon dapat berjalan disistem yang sama atau docker *client* berhubungan dengan daemon melalui jaringan seperti REST API.

### 2.1.7.2 PyCG [19]

PyCG adalah tools analisis statik pada bahasa pemrograman Python. PyCG memiliki kemampuan untuk otomatis menemukan module untuk melakukan analisis lebih lanjut. PyCG bisa menghasilkan *call graph* dari suatu kode baik file maupun dalam bentuk package.

PyCG juga memiliki presisi dan *recall* yang tinggi bila dibandingkan dengan alat lainnya seperti Pyan dan Depends. PyCG sudah dievaluasi dengan proyek lainnya baik kecil atau besar. Pendekatan PyCG modular , mudah dikembangkan lebih lanjut dan mencakup sebagian besar fungsionalitas Python.

### 2.1.7.3 Kong Gateway [16]

Kong Gateway adalah *API Gateway* yang ringan, cepat, dan fleksibel serta kompatibel di Cloud. Dengan Kong Gateway pengguna dapat membuat otomatisasi, desentralisasi aplikasi/service dan transisi ke *microservice* dan memiliki kemampuan mengatur dan mengawasi API.

Kong Gateway memiliki Kong Manager yaitu graphical user interface (GUI). Di mana menggunakan Kong Admin API untuk mengatur Kong Gateway. Kong Manager dapat menambahkan rute dan *service* baru, mengaktifkan dan mematikan plugins, membuat grup untuk tim, dan pemantauan serta pengecekan status *service*. Plugins dapat menambahkan fitur pada Kong seperti rate limiting untuk mengontrol jumlah request yang dikirimkan ke suatu service dan pengaturan cache.

### 2.1.7.4 inspect [17]

Inspect adalah library bawaan dari python , di mana library ini memberikan fungsi untuk mengetahui informasi tentang object yang sedang hidup seperti module,class, method, fungsi, traceback, frame object, dan kode object. Terdapat 4 bagian pada library ini yaitu pengecekan tipe data, mendapatkan kode program, inspeksi *class* atau fungsi, dan memeriksa interpreter stack.

Module yang digunakan pada tugas akhir ini:

**Tabel 2.1** Daftar Metode dan Fungsi inspect

Nama	Keterangan	Atribut
inspect.getmembers	untuk mendapatkan member dari sebuah objek seperti <i>class</i> atau module	object = objek yang ingin didapatkan membernya

inspect.ismodule	untuk mengetahui apakah objek adalah sebuah module	object = objek yang ingin dilakukan pengecekan
inspect.isclass	untuk mengetahui apakah objek adalah sebuah <i>class</i>	object = objek yang ingin dilakukan pengecekan

#### 2.1.7.5 SciPY [18]

SciPY adalah koleksi algoritma matematika yang cocok dengan ekstensi NumPY di Python. SciPy memiliki module algoritma clustering yang dapat digunakan dan dapat dilakukan pemilihan metode yang mungkin dibutuhkan. Proses Perhitungan yang dilakukan SciPy sudah di optimalisasi dan efisien.

Module algoritma yang digunakan pada tugas akhir ini:

**Tabel 2.2** Daftar Metode dan Fungsi SciPy

Nama	Keterangan	Atribut
single	melakukan pengelompokan dengan nilai minimal atau nilai terdekat pada matriks kedekatan	y = matriks kedekatan
average	melakukan pengelompokan dengan nilai rata-rata jarak pada matriks kedekatan	y = matriks kedekatan
complete	melakukan pengelompokan dengan nilai maximal atau nilai terjauh pada matriks kedekatan	y = matriks kedekatan
dendogram	mengilustrasikan dendogram dari <i>cluster</i> yang terbentuk di matriks linkage	Z = matriks linkage

## 2.2 Tinjauan Studi

Pada Tabel 2.1 diberikan penjelasan mengenai studi yang terkait dalam tugas akhir:

**Tabel 2.3** *State of the Art*

Jurnal	Rumusan Masalah	Metode	Hasil
--------	-----------------	--------	-------

<p>A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony (2020). <b>"From monolithic architecture style to Microservice one based on a semi-automatic approach"</b> [12]</p>	<p>Terdapat aplikasi monolitik yang tidak beradaptasi di <i>cloud</i> ataupun <i>DevOps</i> sehingga harus dimigrasi ke <i>microservice</i></p>	<p>Analisis kode program dan mencari hubungan dalam <i>class</i>-nya</p>	<p>Identifikasi Microservice yang dibuat memiliki hasil yang memuaskan karena mempertimbangkan karakteristik microservice</p>
<p>Chaitanya K. Rudrabhatla. (2020). <b>"Impacts of Decomposition Techniques on Performance and Latency of Microservices."</b> [3]</p>	<p>Bagaimana dampak performa dalam menentukan batasan antar <i>service</i></p>	<p>Melakukan perbandingan antara pendekatan DDD, Normalized Entity Relationship, dan Hybrid</p>	<p>Teknik DDD lebih baik dalam dekomposisi namun teknik Hybrid dengan mempertimbangkan fungsionalis dan transaksi yang terjadi memiliki performa lebih baik.</p>
<p>A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee (2021). <b>"Mono2micro: A practical and effective tool for decomposing monolithic Java applications to microservices"</b> [11]</p>	<p>Bagaimana pendekatan Mono2Micro dengan cara dekomposisi lainnya dan bagaimana tanggapan praktisi</p>	<p>Menggunakan hierarchical spatio-temporal decomposition yang menyimpan kondisi program secara dinamik berdasarkan eksekusi proses bisnis</p>	<p>Hasil rekomendasi <i>microservice</i> dapat membantu penerapan pola <i>strangle</i> , partisi yang dihasilkan sesuai dengan fungsi bisnis</p>

Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni. (2022). "A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications" [10]	Bagaimana mengotomatisasi proses migrasi aplikasi monolitik ke <i>microservice</i> ?	Menggunakan DBSCAN(Density-based Clustering) yang menghasilkan rekomendasi <i>microservice</i>	Berhasil membuat <i>microservice</i> yang lebih <i>cohesion</i> dan memiliki interaksi yang lebih sedikit.
--	--	--	--

Pada penelitian yang dilakukan oleh A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, dan C. Dony [12], penelitian melakukan identifikasi *microservice* dari aplikasi Monolitik berbasis Object-Oriented(OO). Identifikasi dimulai dari membuat partisi dari proses pengelompokan untuk menemukan *microservice* yang bagus dengan memperhitungkan karakteristik dari *microservice* seperti berdasarkan struktural dan perilaku ketergantungan *microservice* serta *Data Autonomy*. Untuk mengevaluasi hasil identifikasi yang dibuat yaitu dengan menggunakan kode program yang sudah menjadi *microservice* dan kemudian membandingkannya antara rekomendasi dan yang sebenarnya.

Hasil dari penelitian tersebut adalah identifikasi *microservice* terbaik bisa dilakukan melalui algoritma gravity center dengan keseluruhan kode program. Penggunaan Fungsi untuk mengetahui *microservice* terbaik dapat dilakukan hanya dengan cara struktural tanpa harus mempertimbangkan *data autonomy*.

Pada penelitian yang dilakukan oleh Chaitanya K. Rudrabhatla [3], penelitian melakukan perbandingan latensi antara pemilihan dekomposisi. Peneliti menggunakan aplikasi yang dibuat dengan Spring Boot Java. Dari hasil evaluasi diketahui dekomposisi dengan domain driven desain(DDD) memiliki performa lebih baik daripada pendekatan melalui entitas. Namun dengan pendekatan hybrid/campuran performa antara domain driven memiliki kesamaan sehingga diperlukan transaksi yang kompleks untuk melihat perbedaan

Pada penelitian yang dilakukan oleh A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, dan D. Banerjee [11], peneliti menggunakan Mono2Micro untuk melakukan dekomposisi aplikasi monolitik Java menjadi *microservice*. Kemudian menggunakan metrik untuk mengevaluasi apakah *microservice* yang

dibuat oleh Mono2Micro efektif. Peneliti menggunakan 5 metrik untuk mengukur efektifitas yaitu secara Structural Modularity(SM), Indirect Call Patern(ICP), Business Context Purity(BCP), jumlah Interface(IFN), dan Non-Extreme Distribution(NED) serta ada survey kepada praktisi mengenai penggunaan Mono2Micro

Hasil dari penelitian menunjukkan bahwa penggunaan Mono2Micro efektif dalam melakukan dekomposisi dan dapat memberikan keuntungan bagi pengembang karena bisa membantu pengembang untuk melihat partisi lainnya. Metrik yang dihasilkan dari Mono2Micro memiliki hasil yang bagus di antara 5 metrik tersebut, namun diperlukan penelitian lebih lanjut pada kasus tingginya nilai SM menyebabkan tingginya nilai NED.

Pada penelitian yang dilakukan oleh Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni [10]. Penelitian menggunakan Algoritma Hierarchical DBSCAN dengan analisis statik kode program untuk mendapatkan sekumpulan *service* yang bisa kandidat *microservice*. Peneliti menggabungkan nilai struktural dan nilai semantik analisis untuk menentukan kedekatan suatu *class* dengan *class* lainnya. Hierarchical DBSCAN. Prose evaluasi menggunakan perbandingan antara *microservice service* yang sudah diekstraksi sebelumnya oleh pengembang.

Hasil dari penelitian menunjukkan pendekatan hierarchical clustering dapat melakukan dekomposisi aplikasi monolitik dengan analisis statik. Microservice yang didekomposisi memiliki nilai *cohesion* yang lebih baik di dalam *microservice* dan jumlah interaksi antar *microservice* lebih sedikit.

### 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 monolitik, yaitu Odoo.

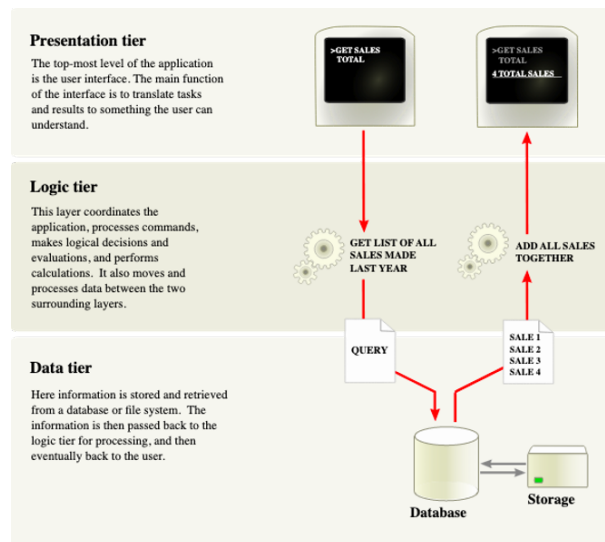
#### 2.3.1 Odoo

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 [14].

Sebelum Odoo terdapat OpenERP, di mana OpenERP memiliki arsitektur monolitik. Pada versi OpenERP ke 7, karena banyaknya pengembangan fitur yang

membuat waktu pengembangan menjadi lama dan sulit. OpenERP melakukan migrasi menjadi arsitektur SOA dan berganti nama menjadi Odoo [4].

Arsitektur yang digunakan pada Odoo yaitu three-tier arsitektur di mana 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 [14].

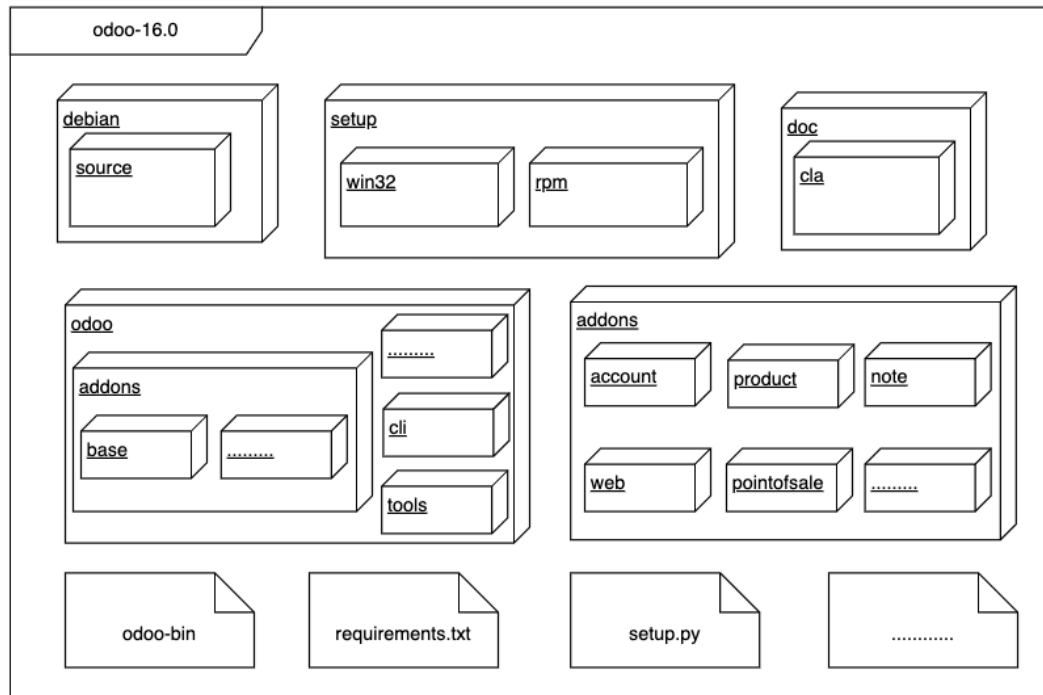


**Gambar 2.11** Arsitektur Odoo [14]

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 [14].

Struktur *module* pada kode program di Odoo versi 16 yaitu ada folder *debian* dan *setup* yang menangani bagaimana aplikasi Odoo diinstall di berbagai platform. Folder *doc* berisi mengenai dokumentasi dan hak cipta. Folder *odoo* merupakan module utama aplikasi odoo tanpa ada module ini maka aplikasi tidak dapat dijalankan. Folder *addons* berisi module yang memiliki aset serta bisa diganti atau dihilangkan sesuai kebutuhan. File seperti *odoo-bin* untuk memulai aplikasi odoo, *requirements.txt* berisi library python yang dibutuhkan aplikasi odoo.





**Gambar 2.12** Struktur Repository Odoo

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. Di mana module adalah koleksi dari fungsi dan data untuk menyelesaikan satu tujuan. Modul pada Odoo bisa ditambahkan, diganti, diubah untuk menyesuaikan kebutuhan bisnis. Di mana pada pengguna module dilambangkan dengan nama Apps, tetapi tidak semua module adalah Apps. Modules juga bisa direfrensikan sebagai addons [14].

**Tabel 2.4** Komposisi dari Module pada aplikasi Odoo [14]:

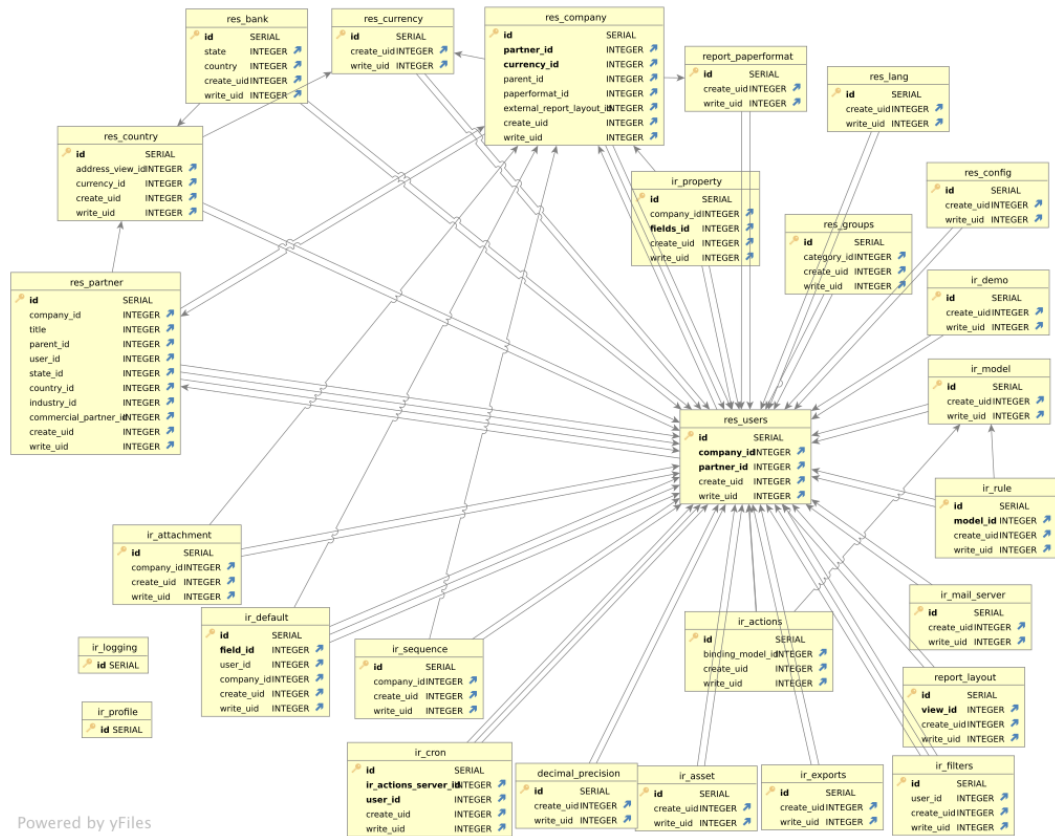
Elemen	Keterangan	Contoh
Business Objects	Object yang akan digunakan di module di mana setiap attribute secara otomatis dipetakan ke kolom <i>database</i> dengan ORM	File python yang memiliki <i>class</i>
Objects Views	Menangani bagaimana data ditampilkan di pengguna. Seperti visualisasi form, list, 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 <i>class</i> namun merupakan turunan dari <i>class</i> <code>odoo.http.Controller</code>
Static Web Data	File yang digunakan hanya ditampilkan kepada <i>client</i> di website	File gambar, File CSS, dan File JavaScript

Terdapat 2 jenis addons yaitu addons base dan addons. Yang membedakannya addons base harus ada di setiap aplikasi Odoo bila tidak ada aplikasi tidak dapat berjalan sedangkan addons bisa diganti sesuai keperluan bisnis. Pengelolaan *database* dilakukan secara otomatis oleh ORM internal Odoo, Odoo memiliki framework yang bisa digunakan untuk menambahkan atau mengelola fitur atau addons.

Tabel yang terbentuk pada konfigurasi umumnya bisa mencapai  $\pm 566$  tabel bila semua module di install sementara itu jumlah tabel tanpa ada module terinstall adalah 99 tabel. Dari tabel tanpa ada module bisa diidentifikasi 27 tabel utama yang digunakan pada aplikasi. Berikut adalah diagram dari *database* aplikasi Odoo, dengan attribute yang ditampilkan hanya sebuah key dari tabel. Nama depan tabel yang berinisial 'ir' artinya information repository dan 'res' artinya resource. Perbedaannya adalah 'res' menyimpan informasi yang digunakan dalam proses bisnis sedangkan 'ir' menyimpan informasi mengenai kebutuhan internal aplikasi.

## BAB 2 LANDASAN TEORI



Gambar 2.13 Skema Database Odoo

## 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

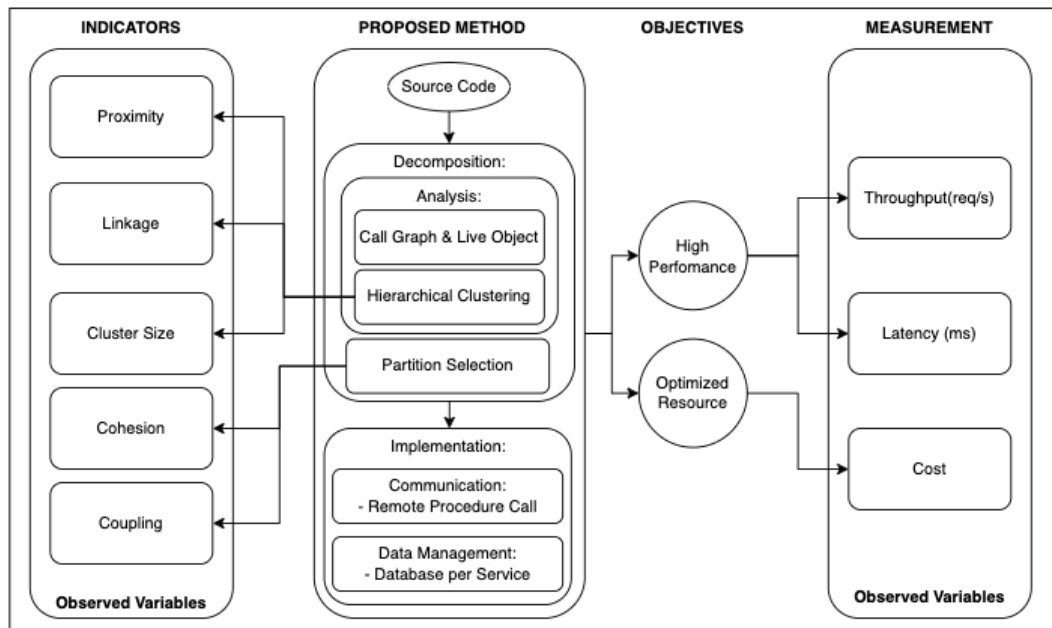
Arsitektur ERP yang digunakan pada Odoo yaitu arsitektur Service Oriented Architecture (SOA) dan masih dideploy secara monolitik, seperti yang sudah dijelaskan pada landasan teori. Arsitektur monolitik memiliki kelemahan dan permasalahan yang bisa diselesaikan dengan arsitektur *microservice*. Aplikasi ERP juga diperlukan untuk memiliki skalabilitas yang baik dan kustomisasi.

Namun untuk melakukan perubahan arsitektur harus dilakukan dekomposisi, proses dekomposisi sendiri tidak mudah karena proses dekomposisi masih membutuhkan analisis secara manual dan untuk mengidentifikasi *service* sulit karena banyaknya pendekatan dan pertimbangan. Pada penelitian ini menggunakan pendekatan Hierarchical clustering untuk membantu menemukan *service* yang tepat, di mana hierarchical clustering memberikan rekomendasi bagaimana pengelompokan *service* berdasarkan pemilihan partisi terbaik. Proses dimulai dari melakukan analisis kode seperti Call Graph yang dihasilkan dari kode aplikasi Odoo. Hasil analisis kode di ekstraksi menjadi matrix untuk dilakukan hierarchical clustering. Cluster terbaik dipilih melalui nilai secara struktural yaitu nilai *coupling* dan *cohesion*.

Hasil terbaik dari clustering diimplementasikan menjadi *service*, penelitian ini akan menggunakan strategi dengan pola *strangle* untuk memecah kode di monolitik. Untuk dekomposisi pada data akan menggunakan strategi *Aggregate Exposing Monolith*.

Microservice yang dibentuk akan dievaluasi melalui uji beban dan penggunaan sumber daya aplikasi untuk menentukan apakah dengan arsitektur *microservice* dapat menyelesaikan permasalahan yang muncul pada arsitektur monolitik di aplikasi ERP.

### 3.2 Kerangka Pemikiran



Gambar 3.1 Kerangka Pemikiran

Penelitian akan dimulai dengan menggunakan kode sumber aplikasi yang dibuat dengan monolitik. Kode sumber 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 *call graph* yang menunjukkan bagaimana keterhubungan masing-masing objek di aplikasi.

Dari *graph* yang sudah dibuat akan dilakukan pengelompokan dengan pendekatan Hierarchical Clustering. Di mana perlu ditentukan cara menghitung kedekatan antara objek dan pemilihan algoritma Linkage. Metode linkage yaitu menentukan jarak atau kemiripan antara semua objek. Untuk menentukan jarak ini bisa dengan rata-rata, maximum, dan minimum.

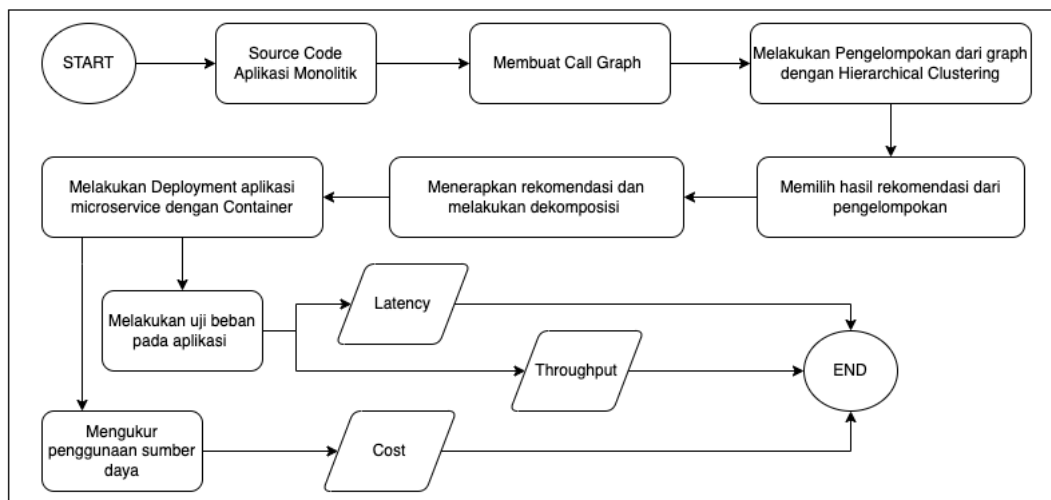
Pengelompokan dari Hierarchical Clustering akan dipilih dengan mencari nilai *cohesion* terendah dan nilai *coupling* tertinggi. Di mana *coupling* mengevaluasi tingkat ketergantungan langsung dan tidak langsung antar objek. Semakin banyak dua objek menggunakan metode masing-masing semakin mereka menjadi satu kesatuan. Sedangkan *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.

Untuk mengetahui bagaimana performa dari *microservice* dibandingkan dengan monolitik yaitu dengan test beban. Pengukuran performa dilihat dari throughput, jumlah *response*, dan latency. Bagi pengukuran biaya dilakukan melalui menghitung perbandingan antara jumlah sumber daya yang digunakan seperti memori (RAM) dan penggunaan CPU(jumlah core) pada *virtual machine* di masing-masing arsitektur.

### 3.3 Urutan Proses Global

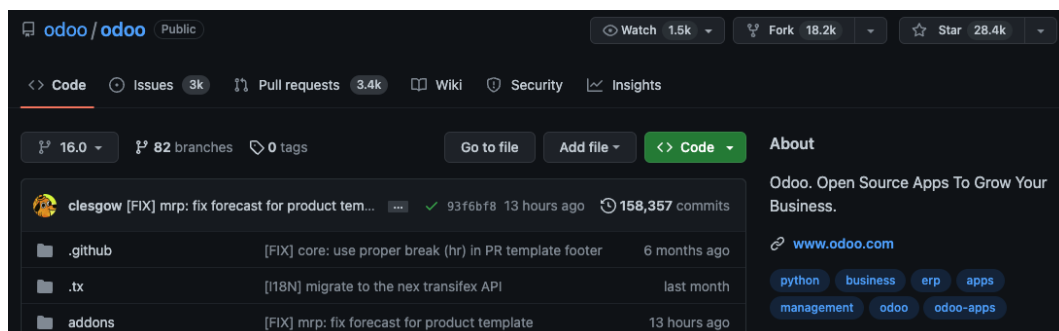


Gambar 3.2 Diagram Flowchart Proses Global

#### 3.3.1 Proses Clustering

##### 3.3.1.1 Pengambilan Source Code

Aplikasi ERP Odoo merupakan aplikasi berlisensi open source, kode program dapat diunduh melalui situs repository Odoo. Pada tugas akhir ini menggunakan Odoo versi 16 dengan status pengujian lulus. Agar kode program dapat berjalan dengan lancar maka diperlukan proses installasi library, module dan Package yang digunakan dari file requirement.txt

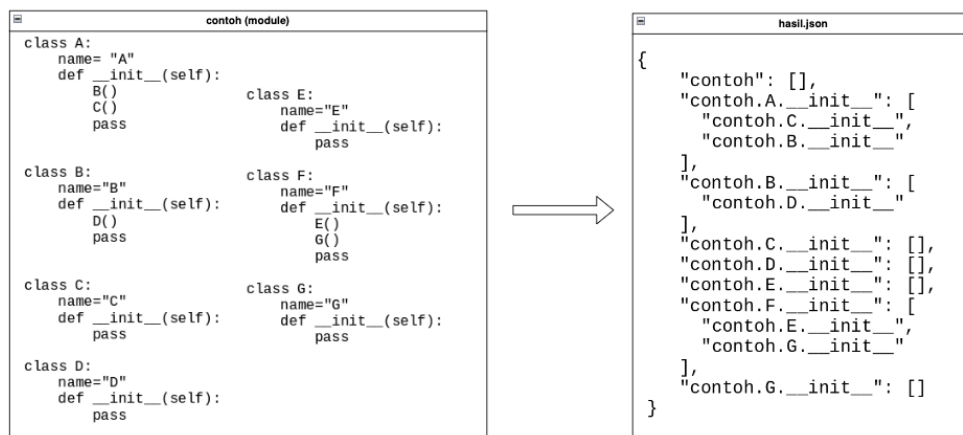


Gambar 3.3 Source Code Aplikasi Odoo pada git repository

### 3.3.1.2 Pembuatan Call Graph

Pada tugas akhir ini menggunakan tools PyCG untuk menghasilkan *call graph* dalam bentuk format JSON. Terdapat 2 target folder yang dibuat *call graph* yaitu folder `odoo.addons` dan folder `addons` karena folder lainnya tidak memiliki hubungan mengenai proses bisnis. Untuk menghemat waktu pembuatan *call graph* maka folder `test` tidak dibuat.

*Entry point* untuk tools PyCG adalah semua file di target folder dengan ekstensi file `.py` serta ditentukan package yang ingin diolah menjadi *call graph*. Proses eksekusi dilakukan melalui terminal. Call graph yang dihasilkan berisi *call* yang berasal dari file `.py` yang ditentukan sebelumnya dan semua module yang terhubung dari target. Semua module ini bisa diluar dari target module apabila keterhubungan itu terus berlanjut. PyCG hanya bisa menghasilkan *call graph* tanpa informasi mengenai jumlah pemanggilan dan urutan pemanggilan.

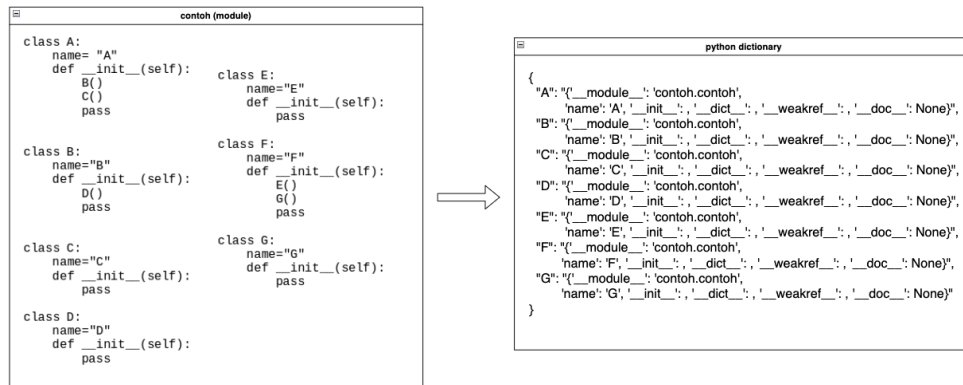


Gambar 3.4 Proses Pembuatan Call Graph dengan PyCG

Proses ekstraksi json yang dihasilkan dari tools PyCG berupa 2 file JSON masing masing adalah `odoo` dan `addons`. Kedua file JSON digabungkan dan menjadi satu *graph* yang direpresentasikan dalam bentuk adjacency list di python.

### 3.3.1.3 Ekstraksi Dependency Module

Keterbatasannya informasi *call graph* yang dihasilkan dari PyCG, sehingga tugas akhir ini menggunakan library python yaitu `'inspect'` untuk menganalisis object secara run-time. Hal ini disebabkan python adalah bahasa pemrogram dinamik di mana pengecekan tipe data dilakukan secara `'run-time'`. Ekstraksi ini difokuskan pada module yang memiliki proses bisnis seperti module `addons` dan `odoo/addons`. Dari gambar 3.5 bisa diketahui penggunaan `inspect` bisa menemukan atribut apa saja dan nilainya dari atribut pada objek.



**Gambar 3.5** Penggunaan 'inspect' untuk melihat object python lebih mendalam

Object yang dianalisis yaitu *class* yang merupakan turunan dari *class* `odoo.models.MetaModel`, di mana *class* `MetaModel` memiliki properti seperti `name`, `_inherits`, `_inherit`, dan `attribute_rel`. Hasil ekstraksi dependency module digabungkan melalui nama module `PyCG`

### 3.3.1.4 Penggabungan dan Optimisasi Hasil Ekstraksi

Graph yang dihasilkan dari proses ekstraksi dipisahkan antara module eksternal dan module internal. Module yang digunakan untuk pengelompokan adalah module internal, setiap *call* yang dilakukan memiliki nama *call* yang berupa gabungan antara nama fungsi / *class* / module / file di kode program. `PyCG` tidak memberikan informasi apakah nama *call* tersebut berupa tipe apa, untuk itu pengelompokan dilakukan secara hybrid yaitu berdasarkan module dan file.

Nama *call* yang disatukan menjadi module adalah *call* yang memiliki awalan(root) `addons` atau `odoo/addons` dan nama *call* yang disatukan dengan file adalah nama *call* selain `addons`. *Call* yang dikelompokkan memiliki nilai agregasi dari jumlah *call*. Jumlah *call* dapat digunakan sebagai *weight* yang dapat menunjukkan kekuatan antara *call* satu sama lain, proses ini membentuk *call* baru yang lebih ringkas dan relevan dalam bentuk *graph*.

### 3.3.1.5 Hierarchical Clustering

Graph yang berbentuk Adjacency list diubah menjadi adjacency matrix, proses normalisasi data dilakukan pada matrix. Tujuan normalisasi data agar nilai *weight*(jumlah *call*) dari relasi berkisar dari 1 hingga 0. Semakin banyak jumlah *call* dilakukan maka nilai mendekati 1, kemudian matrix tersebut dibuat menjadi Distance Matrix, rumus jarak yang digunakan ada 2 yaitu Jaccard dan Struktural Similarity. Masing-masing hasil jarak disatukan dengan rata-rata. Jaccard menghasilkan hasil yang bagus pada remodularisasi perangkat lunak dan Struktural Similarity digunakan untuk melihat kedekatan dari sisi intensitas



panggilan antara module.

	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	0	0	0	1	0	0	0
C	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0
F	0	0	0	0	1	0	1
G	0	0	0	0	0	0	0

→

	A	B	C	D	E	F	G
A	0	0	0	0	0	0	0
B	0.75	0	0	0	0	0	0
C	0.67	1	0	0	0	0	0
D	1	0.5	1	0	0	0	0
E	1	1	1	1	0	0	0
F	1	1	1	1	0.67	0	0
G	1	1	1	1	1	0.67	0

**Gambar 3.6** Perhitungan Jarak Jaccard

	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	0	0	0	1	0	0	0
C	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0
F	0	0	0	0	1	0	1
G	0	0	0	0	0	0	0

→

	A	B	C	D	E	F	G
A	0	0	0	0	0	0	0
B	0.75	0	0	0	0	0	0
C	0.75	1	0	0	0	0	0
D	1	0.75	1	0	0	0	0
E	1	1	1	1	0	0	0
F	1	1	1	1	0.75	0	0
G	1	1	1	1	1	0.75	0

**Gambar 3.7** Perhitungan Jarak Struktural

	A	B	C	D	E	F	G
A	0	0	0	0	0	0	0
B	0.75	0	0	0	0	0	0
C	0.75	1	0	0	0	0	0
D	1	0.75	1	0	0	0	0
E	1	1	1	1	0	0	0
F	1	1	1	1	0.75	0	0
G	1	1	1	1	1	0.75	0

+

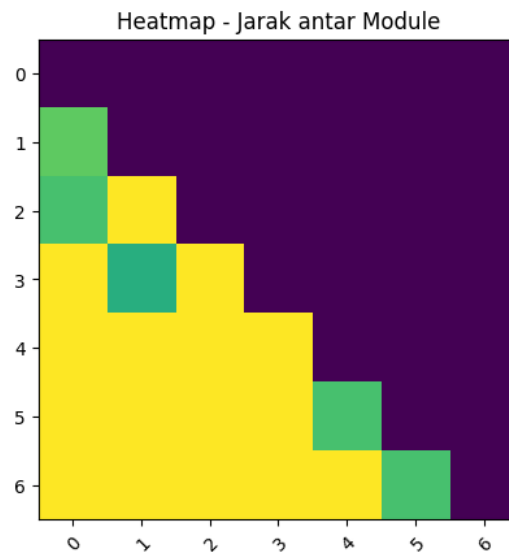
	A	B	C	D	E	F	G
A	0	0	0	0	0	0	0
B	0.75	0	0	0	0	0	0
C	0.67	1	0	0	0	0	0
D	1	0.5	1	0	0	0	0
E	1	1	1	1	0	0	0
F	1	1	1	1	0.67	0	0
G	1	1	1	1	1	0.67	0

=

	A	B	C	D	E	F	G
A	0	0	0	0	0	0	0
B	0.75	0	0	0	0	0	0
C	0.71	1	0	0	0	0	0
D	1	0.62	1	0	0	0	0
E	1	1	1	1	0	0	0
F	1	1	1	1	0.71	0	0
G	1	1	1	1	1	0.71	0

**Gambar 3.8** Hasil Akhir Jarak

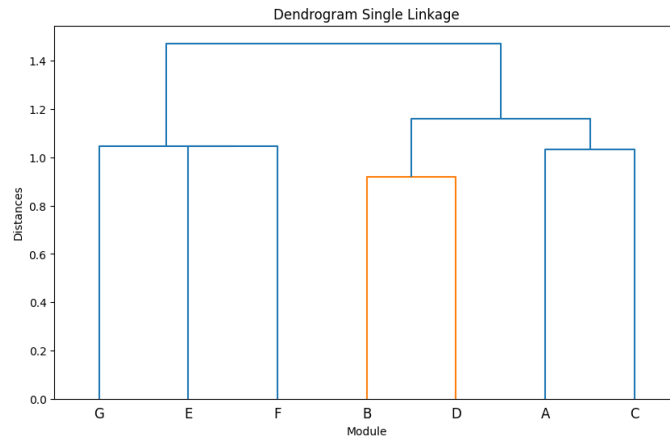
Distance Matrix / Matrix kedekatan dapat dilihat nilai dengan ilustrasi Heatmap, di mana sumbu x dan y adalah semua module dan nilai kedekatannya dengan module lainnya. Semakin terang warna menunjukkan hubungan yang kuat antara module, perlu diketahui bahwa distance matrix merupakan matrix segitiga. Pada tugas akhir ini menggunakan library SciPy untuk melakukan proses *clustering* yang memiliki fungsi Hierarchical Clustering.



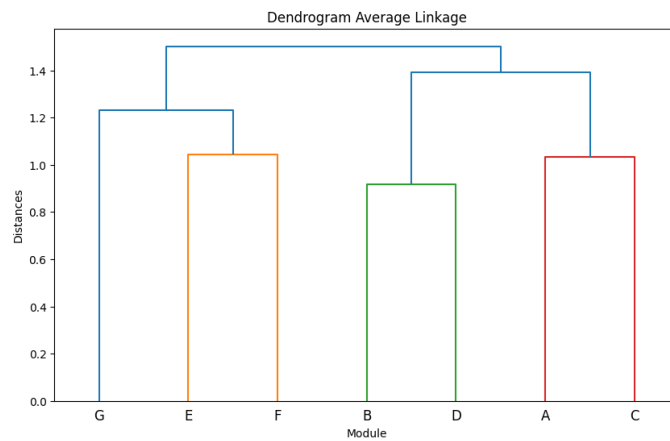
**Gambar 3.9** Heatmap yang dihasilkan dari Distance Matrix

Pemilihan pengelompokan dengan hierarchical agglomerative clustering dibandingkan Partition clustering karena tidak mudah untuk mengetahui jumlah ideal *cluster*. Untuk menentukan metode *linkage* tugas akhir ini menggunakan single linkage, average linkage, dan complete linkage. Hasil dari masing-masing linkage dipilih jumlah partisi yang ideal untuk *microservice*. Penggunaan single linkage memiliki kecenderungan menghasilkan banyak partisi yang berisi modul sedikit tetapi ada satu partisi memiliki banyak module sedangkan complete linkage menghasilkan partisi yang memiliki jumlah modul yang sama dengan partisi lainnya. Untuk Average linkage menghasilkan bentuk partisi di antara complete linkage dan single linkage.

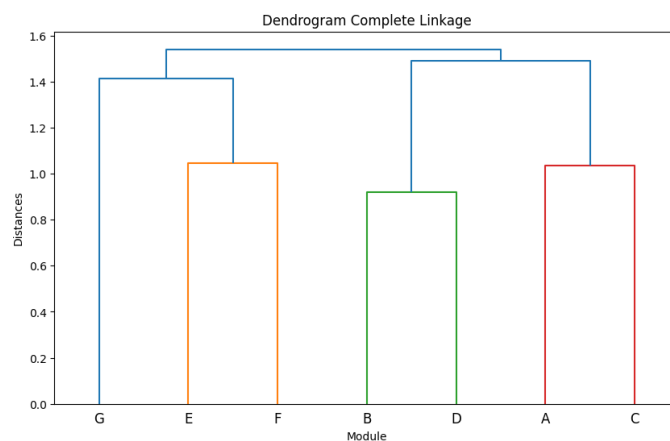
Hasil pengelompokan dapat ditampilkan dalam bentuk dendogram. Di mana pengelompokan dari setiap linkage memiliki dampak berbeda yang bisa dilihat dari bentuk dan nilai kedekatan antar partisi melalui dendogram dan bentuk relasi. Warna pada garis keterhubungan module memiliki arti sebagai partisi tersendiri karena memiliki jarak yang jauh dari partisi lainnya.



**Gambar 3.10** Dendrogram Single Linkage



**Gambar 3.11** Dendrogram Average Linkage



**Gambar 3.12** Dendrogram Complete Linkage

### 3.3.1.6 Pemilihan Partisi

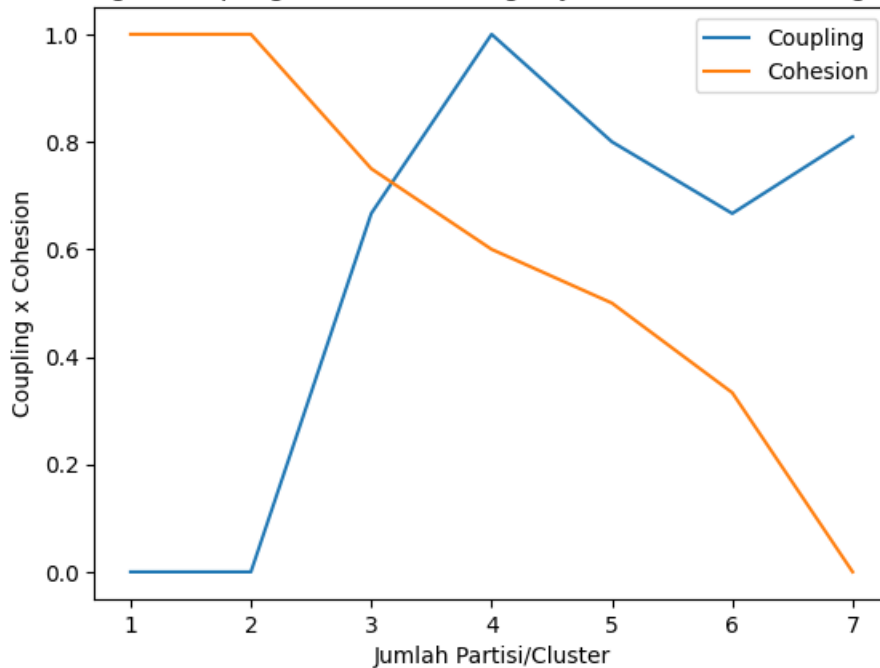
Pemilihan jumlah partisi perlu dilakukan dengan perhitungan yang dapat menentukan jumlah *service* yang ideal. Microservice yang ideal memiliki nilai *coupling* yang rendah dan nilai *cohesion* yang tinggi. Untuk itu tugas akhir ini menentukan partisi dengan nilai struktural yang menggunakan persamaan 2.2 dan

tidak memperhitungkan nilai *coupling* external karena addons pada Odoo dibuat dengan framework Odoo. Hubungan module luar seperti library umumnya dilakukan oleh framework Odoo sendiri bukan oleh addons.

Untuk pemilihan partisi harus mempertimbangkan nilai *cohesion*, nilai *coupling*, jumlah *service*, dan apakah *service* tersebut seimbang. Partisi yang akan menjadi *service* diharapkan bisa independen.

Berikut adalah hasil nilai *coupling* dan *cohesion* masing-masing jumlah *cluster*. Semakin tinggi jumlah *cluster* maka nilai *coupling* akan meningkat dan begitu pula sebaliknya untuk nilai *cohesion*. Dari contoh data ditemukan bahwa *cluster* yang ideal berjumlah 2 *service*, karena ketika semakin banyak jumlah *service* maka nilai *coupling* meningkat dan sebaliknya nilai *cohesion* menurun.

Perbandingan Coupling & Cohesion dengan Jumlah Cluster (Average Linkage)

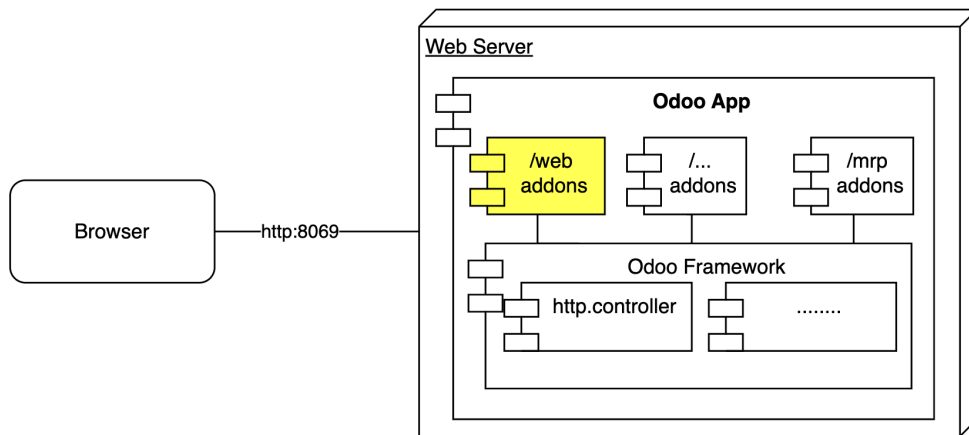


**Gambar 3.13** Perbandingan dari nilai Cohesion dan nilai Coupling dengan jumlah *cluster*/partisi

### 3.3.2 Dekomposisi Monolitik ke Microservice

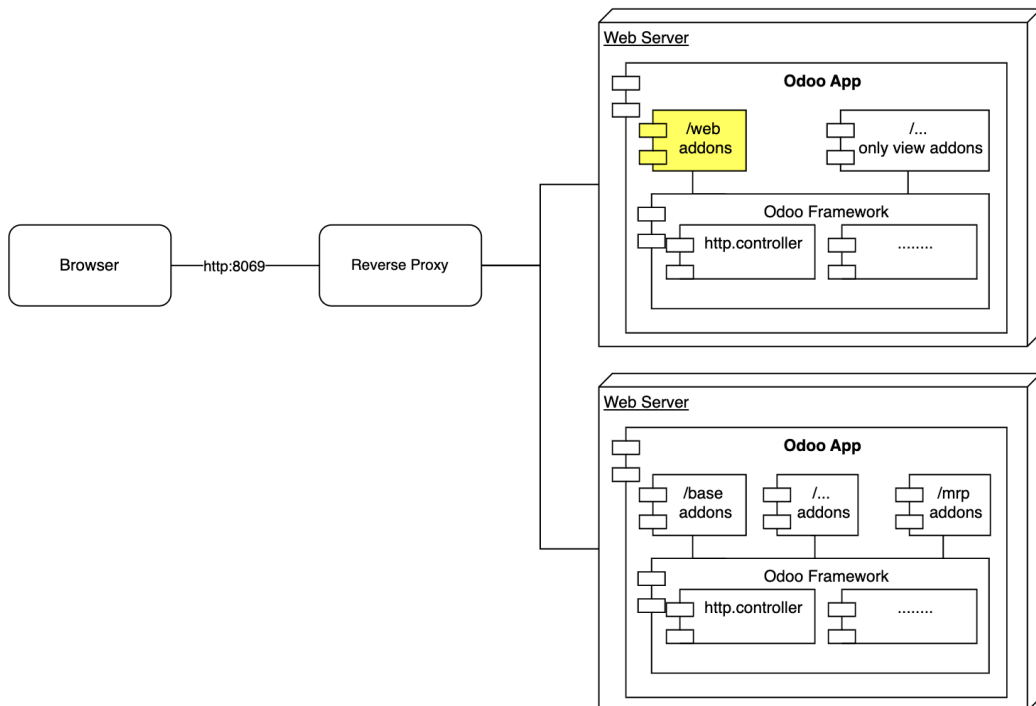
#### 3.3.2.1 Pemisahan User Interface

Odoo adalah aplikasi ERP berbasis web, tampilan pada Odoo bisa dibuka pada broser yang kompatibel. Odoo menggunakan pendekatan SPA (Single Page Application) dan adanya server rendering untuk menghasilkan HTML. UI dapat memanggil API yang berada pada server yang sama, sehingga diperlukan pemisahan kepentingan. Benefit dari proses ini dapat meningkatkan skalabilitas dan pengembangan bisa dilakukan secara independen.



**Gambar 3.14** Arsitektur UI Monolitik

Proses pemisahan membutuhkan *reverse proxy* yang dapat menghubungkan web server di mana bertanggung jawab pada hal seperti file static(css,js,gambar) dan UI dengan web server lainnya. Tujuan adanya *reverse proxy* agar *client* hanya perlu mengetahui satu pintu masuk aplikasi yaitu *reverse proxy* itu sendiri dan tidak perlu mengetahui seluruh server yang ada di aplikasi.

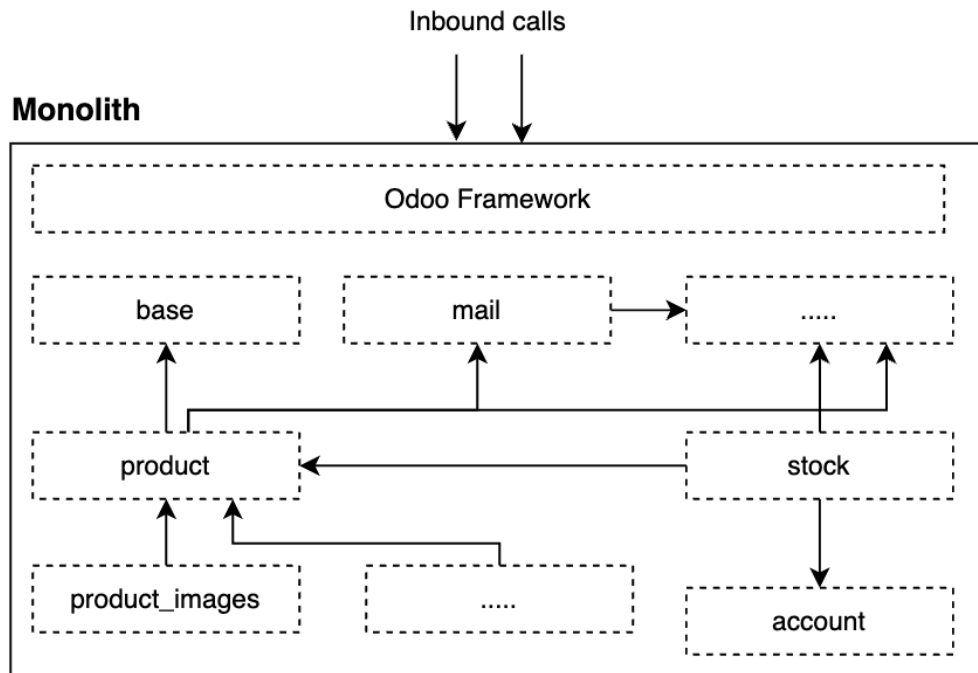


**Gambar 3.15** Arsitektur UI di Microservice

### 3.3.2.2 Strategi Pemisahan Kode

Berdasarkan landasan teori terdapat beberapa strategi pemisahan kode aplikasi monolitik, pada tugas akhir ini akan menggunakan 2 strategi yaitu pola

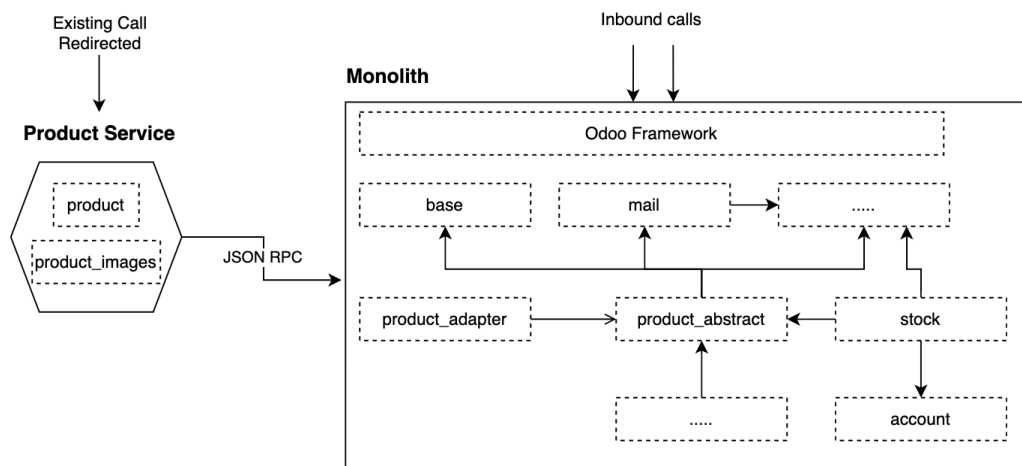
*Strangle* dan pola Branch by Abstraction. Pola *Strangle* diterapkan karena pendekatan ini umum diterapkan dan lebih mudah pada suatu aplikasi yang sudah besar, dengan pola ini aplikasi monolitik bisa berdiri bersamaan dengan *service* yang ingin dibangun atau dimigrasi.



**Gambar 3.16** Struktur Module dan Keterhubungannya di Aplikasi Monolitik

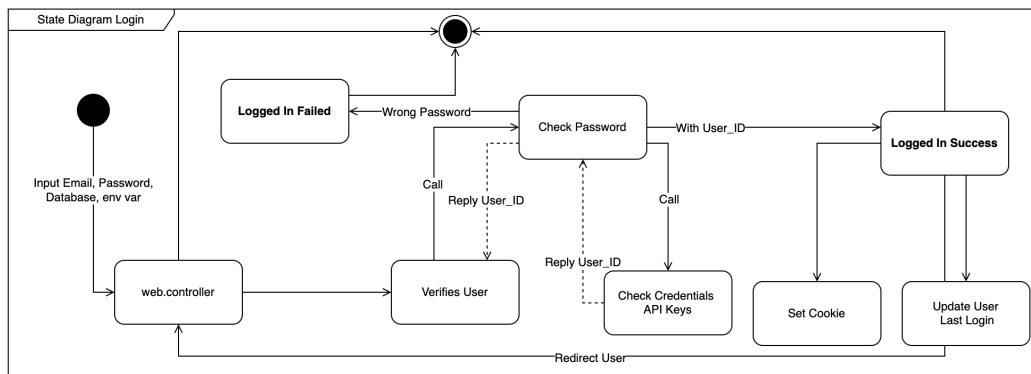
Terdapat 3 langkah utama dalam menerapkan pola *strangle* yaitu memilih bagian yang ingin dipindahkan, memindahkan aplikasi menjadi *service* yang berdiri sendiri, dan yang terakhir mengubah *call* dari monolitik ke *service* yang baru dibuat. Tugas akhir ini milih bagian yang dipindahkan pada kasus bisnis yaitu 'Product', Product bisa dilakukan penambahan Product baru, perubahan atributnya, pencarian atau mendapatkan product dan penghapusan product.

Proses pemindahan module dibantu dengan hasil clustering yang sudah diproses sebelumnya sehingga dibuat untuk membuat *microservice*. Untuk menghubungkan antara bagian yang sudah dipisah dari monolitik dengan bagian yang masih di monolitik maka diperlukan penerapan pola Branch by Abstraction. Terdapat dua bagian utama yaitu abstract dan adapter. Abstract berperan menggantikan bagian yang sudah pisah menjadi *service* sehingga bagian lain di monolitik tidak terdampak dan Adapter adalah implementasi sesungguhnya yang menghubungkan antara *service* dan aplikasi monolitik.



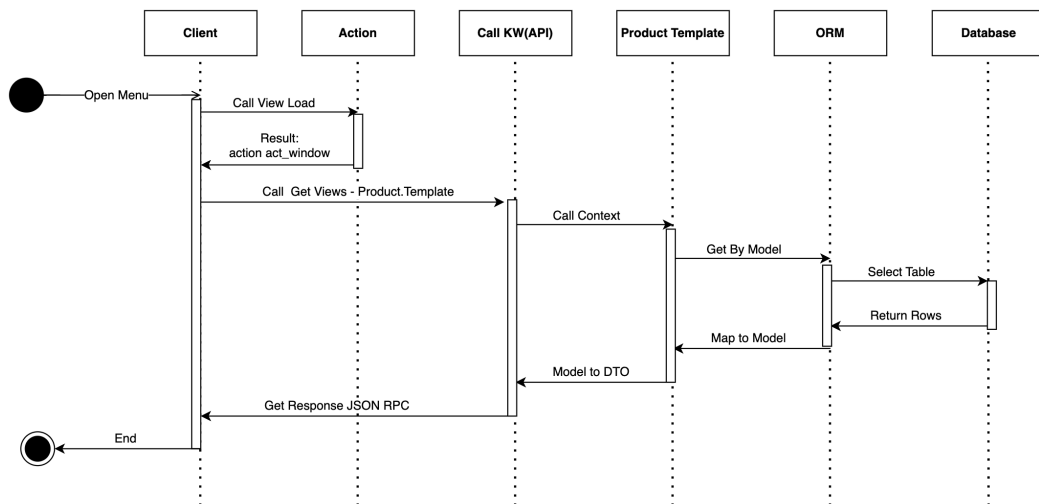
**Gambar 3.17** Penerapan Pola *Strangle* dan Branch by Abstraction

Pemisahan kode mempengaruhi proses autentikasi, proses autentikasi pada aplikasi Odoo terdapat 2 cara yaitu melalui password atau API-Key. Odoo menyimpan sesi autentikasi di cookie namun bukan dalam format JWT tapi bentuk HTTP session. Untuk itu diperlukan modifikasi pada sistem autentikasi yang menggunakan format JWT agar setiap *service* tidak perlu memvalidasi berkali-kali apakah sesi itu valid.



**Gambar 3.18** State Diagram pada proses login

Pada kasus bisnis untuk mendapatkan product, Odoo memiliki model framework yang memiliki fitur Object Relational Mapping(ORM), fitur controller dan fitur lainnya. Model framework diterapkan pada module addons ataupun odoo.addons, penggunaan model framework masih menjadi satu pada repo sehingga perubahan model dapat menyebabkan kerusakan di bagian addons lainnya. Model framework ini bisa diubah menjadi *shared library* dan menjadi bentuk *microservice chasis*.



**Gambar 3.19** Sequence Diagram pada kasus pengambilan data Product

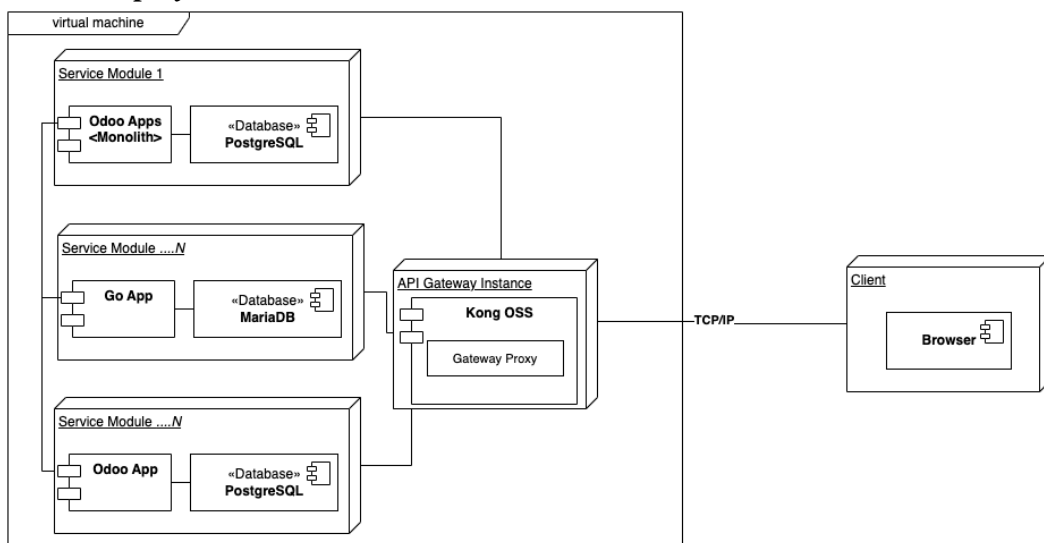
### 3.3.2.3 Komunikasi antar service

Proses komunikasi antar *service* dilakukan melalui JSON-RPC karena Odoo sudah memiliki untuk setiap addonsnya RPC ini bisa berupa XML atau JSON. Apabila diperlukan penambahan metode komunikasi lainnya seperti gRPC maka harus dibangun *service* yang mengubah gRPC menjadi JSON-RPC atau sebaliknya.

### 3.3.2.4 Strategi Pemisahan database

Pemisahan *database* dilakukan setelah dilakukan pemisahan kode karena pada Odoo sudah terdapat ORM yang mengelola *database*. Ketika *database* ingin dipisahkan maka pengaksesan *database* monolitik digunakan sebagai data access layer melalui API yang bisa berupa JSON-RPC.

### 3.3.3 Deployment





**Gambar 3.20** Diagram *Deployment*

Pendekatan *Deployment Service* yang digunakan adalah pola *container* karena *container* memiliki proses *deployment* yang ringan dan modern. Container ini bisa dijalankan melalui Virtual Machine atau langsung dari host, selain itu *container* memiliki banyak keuntungan seperti mengenkapsulasi teknologi, setiap server instance terisolasi, performa juga tidak buruk.

Tugas Akhir ini menggunakan *API Gateway Kong (off-the-shelf)* karena Kong sudah memiliki fitur yang lengkap pada kasus migrasi aplikasi monolitik ke *microservice*. Fitur itu berupa kemampuan untuk redireksi url untuk menerapkan proses *strangle*, pemantauan *service*, dan memiliki performa yang baik.

## **BAB 4 IMPLEMENTASI DAN PENGUJIAN**

### **4.1 Lingkungan Implementasi**

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. Aliquam tincidunt a nulla ac posuere. Maecenas sapien mi, feugiat sit amet tellus at, dictum varius ante. Cras rutrum facilisis felis at hendrerit. Nullam eleifend sed lorem a iaculis. Donec ut odio at nisl molestie euismod quis et purus. Curabitur eu ex turpis. Etiam maximus metus non iaculis placerat. Sed in risus sodales, posuere elit in, eleifend tellus. Mauris at consectetur arcu. Integer fringilla eros mi, vel volutpat enim commodo ac.

#### **4.1.1 Spesifikasi Perangkat Keras**

Suspendisse ac porta diam, ut viverra ante. Aliquam mattis tincidunt diam in

#### **4.1.2 Lingkungan Perangkat Lunak**

Suspendisse ac porta diam, ut viverra ante. Aliquam mattis tincidunt diam in molestie.

### **4.2 Implementasi Perangkat Lunak**

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.

#### **4.2.1 Endpoint API**

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

#### **4.2.2 Module ERP**

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

### 4.2.3 Daftar *Class* dan Metode untuk *Clustering*

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

**Tabel 4.1** Daftar *method* pada *class helper*

No.	Method	Masukan	Luaran	Keterangan
1.	<code>__init__</code>	-	-	Konstruktor yang menginisialisasi objek Training di mana proses inisialisasi parameter CNN juga dilakukan.
2.	<code>auto_training</code>	-	float[] float[]	Menjalankan alur proses <i>training</i> secara keseluruhan dimulai dari pengambilan citra <i>host</i> dan <i>watermark</i> dari direktori <i>local</i> , penyisipan <i>watermark</i> , ekstraksi <i>embedding map</i> , hingga pemrosesan <i>embedding map</i> dengan CNN. Fungsi mengembalikan nilai <i>loss</i> dari akurasi.
3.	<code>normalize</code> <code>_watermark</code>	image : float[][]	float[][]	Memroses citra watermark agar dapat digunakan untuk <i>training</i> .
4.	<code>apply</code> <code>_transformations</code>	image : float[][]  image : float[][] iswatermark : boolean	float[][]	Menjalankan seluruh transformasi digital pada citra dan menyimpannya sebagai <i>array</i> .

5.	get _embedding_maps	images : float[][] key : string	float[][]	Mengambil <i>embedding map</i> dari setiap citra yang telah disisipi watermark.
6.	divide _training_images	images : float[][] ground_truth : float[]	-	Membagi <i>embedding map</i> dan citra <i>ground truth</i> ke dalam <i>batch</i> sesuai <i>batch size</i> yang telah ditentukan.
7.	cross_entropy _per_batch	images : float[][] ground_truth : float[]	float[]	menghitung nilai <i>loss</i> setiap citra dalam satu <i>batch</i> terhadap citra <i>ground truth</i> .
8.	run	-	float[] float[]	Menjalankan proses <i>training CNN</i> . Fungsi mengembalikan hasil <i>training</i> dan <i>loss</i> terakhir.
9.	store_params	-	-	Menyimpan seluruh parameter CNN ke dalam direktori <i>local</i> .
10.	normalize _watermark	images : float[][]	float[]	Menyamakan ukuran dan tipe data watermark.

#### 4.2.4 Alat Pendukung

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

### 4.3 Implementasi Migrasi Arsitektur

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ac felis dignissim, iaculis odio ut, euismod quam.

#### 4.3.1 Proses Clustering

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ac felis dignissim

### 4.3.2 Dekomposisi Monolitik ke Microservice

Quisque dictum auctor tempor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

### 4.3.3 Deployment

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

**Tabel 4.2** atribut pada *class* nama\_class\_1

<b>atribut:</b>					
Float	C	Float	tol	Float	gamma
Float	a	Float	r	Integer	pos_true
Integer	pos_pred	Integer	net_true	Integer	net_pred
Integer	neg_true	Integer	neg_pred	Float	accuracy_score
Float	precision_score	Float	recall_score	Float	f_score

## 4.4 Pengujian

### 4.4.1 Perancangan Pengujian

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

### 4.4.2 Performa

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

### 4.4.3 Biaya

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

## **BAB 5 KESIMPULAN DAN SARAN**

### **5.1 Kesimpulan**

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 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, consectetur adipiscing elit. Donec ac felis dignissim, iaculis odio ut, euismod quam. Donec vestibulum pellentesque sem, eu aliquet purus lacinia ac. Nam porttitor auctor justo et lobortis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed et gravida neque. Praesent commodo aliquam vestibulum. Vivamus blandit mattis mi ut euismod. Proin vitae vestibulum orci, eget elementum tellus. Suspendisse potenti.

Integer non diam a sem venenatis iaculis. Suspendisse quam leo, ultrices sed mollis sit amet, sagittis sit amet nulla. Nam placerat enim in tellus convallis gravida nec quis ipsum. Sed a dapibus erat. Maecenas suscipit maximus turpis vel tempor. In cursus aliquet tellus id viverra. Aenean venenatis augue magna, at ullamcorper erat tincidunt nec. Etiam nec dolor efficitur, iaculis nulla in, semper mi. Ut consectetur aliquet ex, a tincidunt nisi vulputate non. Proin mauris sapien, ultricies sit amet arcu bibendum, molestie suscipit mi. Mauris laoreet facilisis augue, et interdum purus vehicula sit amet. Fusce porta condimentum cursus.

### **5.2 Saran**

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 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, consectetur adipiscing elit. Donec ac felis dignissim, iaculis odio ut, euismod quam. Donec vestibulum pellentesque sem, eu aliquet purus lacinia ac. Nam porttitor auctor justo et lobortis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed et gravida neque. Praesent commodo aliquam vestibulum. Vivamus blandit mattis mi ut euismod. Proin vitae vestibulum orci, eget elementum tellus. Suspendisse potenti.

## 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] Sam Newman, *Monolith to Microservices*, Sebastopol, CA: O'Reilly Media, Inc., 2020, pp. 12-15
- [6] 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.
- [7] Richardson, C. (2018) Microservice patterns: With examples in Java. Shelter Island, NY: Manning Publications.
- [8] 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>.
- [9] Jain, A.K. and Dubes, R.C. (1988) Algorithms for clustering data. Englewood Cliffs, NJ: Prentice Hall.



- [10] 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]
- [11] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, "Mono2micro: A practical and effective tool for decomposing monolithic Java applications to microservices," Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021.
- [12] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From monolithic architecture style to Microservice one based on a semi-automatic approach," 2020 IEEE International Conference on Software Architecture (ICSA), 2020.
- [13] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing object-oriented class modules using an agglomerative clustering technique," 2009 IEEE International Conference on Software Maintenance, 2009.
- [14] "Odoo documentation," Odoo. [Online]. Available: <https://www.odoo.com/documentation/16.0/>. [Accessed: 21-Mar-2023].
- [15] "Docker docs," Docker. [Online]. Available: <https://docs.docker.com/>. [Accessed: 21-Mar-2023].
- [16] "Kong documentation," Kong. [Online]. Available: <https://docs.konghq.com/>. [Accessed: 21-Mar-2023].
- [17] "inspect — Inspect live object," inspect. [Online]. Available: <https://docs.python.org/3/library/inspect.html> . [Accessed: 21-Mar-2023].
- [18] "Scipy documentation," SciPY. [Online]. Available: <https://docs.scipy.org/doc/scipy/>. [Accessed: 21-Mar-2023].
- [19] V. Salis, T. Sotiropoulos, P. Louridas, D. Spinellis, and D. Mitropoulos, "PYCG: Practical *call graph* generation in python," 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021.

## DAFTAR REFERENSI

---

- [20] D. Müllner, “Modern hierarchical, agglomerative clustering algorithms,” arXiv.org, 12-Sep-2011. [Online]. Available: <https://arxiv.org/abs/1109.2378>. [Accessed: 21-Mar-2023].

## 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