

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

TUGAS AKHIR

**Albertus Septian Angkuw
1119002**



**PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2023**

**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



INSTITUT
TEKNOLOGI
HARAPAN
BANGSA

Veritas vos liberabit

**PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2023**

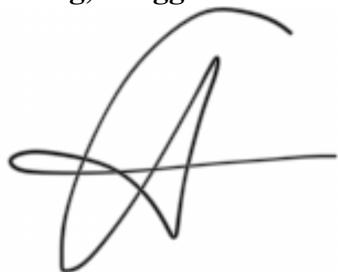
HALAMAN PERNYATAAN ORISINALITAS

**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



**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 bertanda tangan di bawah ini:

Nama : Nama Pengarang

NIM : NIM

Program Studi : Informatika

demi 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:

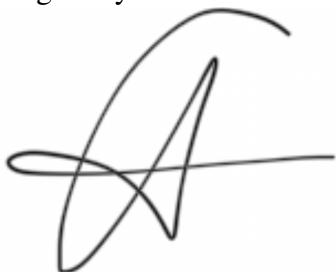
JUDUL TUGAS AKHIR

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 vidiisse 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, tibique sapientem contentiones no vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque similique sed eu, nam facilisis neglegentur ex. Qui ut tibique 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 eruditio 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 vidiisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aeque decore in mei. Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidiisse 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, tibique, 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, tibique sapientem contentiones no vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque similique sed eu, nam facilisis neglegentur ex. Qui ut tibique 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 erudit 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, aeque 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, tibique, 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, tibique sapientem contentiones no
vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque
similique sed eu, nam facilisis neglegentur ex. Qui ut tibique 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 erudit
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, aeque decore in mei.

Bandung, Tanggal Bulan Tahun

Hormat penulis,

A handwritten signature in black ink, appearing to read "Firdaus".

Nama Pengarang

DAFTAR ISI

ABSTRAK	iv
ABSTRACT	v
KATA PENGANTAR	vi
DAFTAR ISI	vii
DAFTAR TABEL	x
DAFTAR GAMBAR	xi
DAFTAR ALGORITMA	xiii
DAFTAR LAMPIRAN	xiv
BAB 1 PENDAHULUAN	1-1
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
BAB 2 LANDASAN TEORI	2-1
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>	2-23
2.1.7.1	<i>Docker</i> [15]	2-23
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]	2-24
2.1.7.5	<i>SciPy</i> [18]	2-24
2.2	Tinjauan Studi	2-25
2.3	Tinjauan Objek	2-28
2.3.1	Odoo	2-28

BAB 3 ANALISIS DAN PERANCANGAN SISTEM	3-1	
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 Model	3-4
3.3.1.4	Pengabungan dan Optimisasi Hasil Ekstraksi	3-5
3.3.1.5	Hierarchical Clustering	3-6
3.3.1.6	Pemilihan Partisi	3-11
3.3.2	Dekomposisi Monolitik ke Microservice	3-14
3.3.2.1	Strategi Pemisahan Kode	3-14
3.3.2.2	Komunikasi antar service	3-16
3.3.2.3	Strategi Pemisahan <i>database</i>	3-16

BAB 4 IMPLEMENTASI DAN PENGUJIAN	4-1
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 Proses Clustering	4-1
4.2.1 Pengambilan Source Code	4-1
4.2.2 Pembuatan Call Graph	4-2
4.2.3 Ekstraksi Dependency Model	4-3
4.2.4 Pengabungan Hasil Ekstraksi	4-7
4.2.5 Optimisasi Hasil Ekstraksi	4-10
4.2.6 Hierarchical Clustering	4-14
4.3 Evaluasi dan Pengujian	4-18
4.3.1 Pemilihan Partisi	4-18
4.3.1.1 Implementasi Pemilihan	4-18
4.3.1.2 Evaluasi Pemilihan	4-19
4.3.2 Dekomposisi Monolitik <i>ke Microservice</i>	4-29
4.3.2.1 Pemilihan Service	4-29
4.3.2.2 <i>Endpoint API</i>	4-32
4.3.2.3 Daftar Class	4-33
4.3.2.4 <i>Docker</i>	4-36
4.3.2.5 <i>Kong</i>	4-37
4.3.2.6 Penerapan JSON Web Token(JWT)	4-38
4.3.2.7 Pola <i>Strangle Fig</i> dan <i>Branch by Abstraction</i>	4-41
4.3.2.8 Analisis Hasil Dekomposisi	4-43
BAB 5 KESIMPULAN DAN SARAN	5-1
5.1 Kesimpulan	5-1
5.2 Saran	5-1
BAB A LAMPIRAN A	A-1
ASJDBAKJSDBKA	A-1
BAB B DATASET HASIL KUISIONER 2	B-3

DAFTAR TABEL

2.1	Daftar Metode dan Fungsi inspect	2-24
2.2	Daftar Metode dan Fungsi SciPy	2-24
2.3	<i>State of the Art</i>	2-25
2.4	Komposisi dari Module pada aplikasi Odoo [14]:	2-30
4.1	Daftar Metode untuk melakukan ekstraksi <i>Dependency Module</i> . . .	4-3
4.2	Daftar Metode untuk pengabungan hasil ekstraksi	4-7
4.3	Daftar Metode untuk optimisasi hasil ekstraksi	4-10
4.4	Daftar Metode untuk proses Clustering	4-15
4.5	Daftar Metode untuk pemilihan Partisi	4-18
4.6	Tabel Nilai maximum, rata-rata, dan minimum untuk setiap linkage	4-21
4.7	Perbandingan Ukuran Service yang dihasilkan oleh Single Linkage	4-22
4.8	Perbandingan Ukuran Service yang dihasilkan oleh Complete Linkage	4-24
4.9	Perbandingan Ukuran Service yang dihasilkan oleh Average Linkage	4-26
4.10	Daftar Module untuk Setiap Partisi	4-29
4.11	Daftar Model untuk masing-masing Module	4-30
4.12	Tabel Endpoint API	4-33
4.13	Tabel class yang digunakan	4-34
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 Stephen C. Johnson <i>Hierarchical Agglomerative Clustering</i> untuk Single Linkage dan Complete Linkage [9]	2-20
2.10	Hasil Partitional Clustering pada Objek dengan n=3	2-21
2.11	Arsitektur Odoo [14]	2-28
2.12	Struktur Repository Odoo	2-29
2.13	Inheritance pada Model Odoo	2-31
2.14	Skema Database Odoo	2-32
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 objek Python lebih mendalam .	3-5
3.6	Graph dan Adjacency List	3-6
3.7	Perubahan dari Adjacency List menjadi Adjacency Matrix	3-6
3.8	Perhitungan Jarak	3-7
3.9	Hasil Akhir Jarak	3-7
3.10	Heatmap yang dihasilkan dari Distance Matrix	3-8
3.11	Proses Perhitungan Hierarchical Clustering	3-9
3.12	Lanjutan Proses Perhitungan Hierarchical Clustering	3-9
3.13	Lanjutan Proses Perhitungan Hierarchical Clustering	3-10
3.14	Lanjutan Proses Perhitungan Hierarchical Clustering	3-10
3.15	Dendogram Single Linkage	3-11
3.16	Dendogram Average Linkage	3-11
3.17	Dendogram Complete Linkage	3-11
3.18	Proses pemotongan tree dan perubahan menjadi Adjacency Matrix dengan linkage single sejumlah 3 partisi	3-12

3.19	Proses perhitungan coupling dan cohesion	3-13
3.20	Perbandingan dari nilai Cohesion dan nilai Coupling dengan jumlah cluster/partisi menggunakan Single Linkage	3-13
3.21	Arsitektur di Monolitik	3-14
3.22	Arsitektur di Microservice	3-14
3.23	Ilustrasi Struktur Module dan Keterhubungannya di Aplikasi Monolitik	3-15
3.24	Penerapan Pola <i>Strangle</i> dan Branch by Abstraction	3-16
3.25	State Diagram pada proses login	3-16
4.1	Waktu pembuatan call graph dengan PyCG	4-2
4.2	Perbandingan hasil PyCG dengan Kode Program Aslinya	4-3
4.3	Implementasi Ekstraksi Model dengan inspect	4-4
4.4	Implementasi Ekstraksi Model dengan inspect Lanjutan	4-5
4.5	Contoh Informasi yang diekstraksi dari Model	4-6
4.6	Hasil ekstraksi dari Model menggunakan inspect	4-7
4.7	Proses pembacaan file JSON dan penambahanan prefix	4-8
4.8	Isi fungsi filterCGNode	4-8
4.9	Hasil akhir node dari yang sudah digabungkan dan dibersihkan . .	4-9
4.10	Isi fungsi updateCGwInspect	4-9
4.11	Ilustrasi Hasil Gabungan Graph	4-10
4.12	class NodeCG untuk menghitung weight	4-12
4.13	Proses perubahan graph di dictionary menjadi class NodeCG . . .	4-12
4.14	Ilustrasi Path dot menjadi bentuk Tree NodeCG	4-13
4.15	Proses Optimisasi Graph dan Pembuatan Adjacency Matrik . . .	4-13
4.16	Ilustrasi Hasil Graph yang dapat dilakukan Proses Pengelompokan .	4-14
4.17	Implementasi Perhitungan Distance Matrix dengan Jaccard dan Struktural Similarity	4-14
4.18	Heatmap yang menunjukan intensitas hubungan antar module/node .	4-15
4.19	Dendogram Single Linkage	4-16
4.20	Dendogram Complete Linkage	4-17
4.21	Dendogram Average Linkage	4-17
4.22	Implementasi untuk melakukan menghitung FOne (coupling dan cohesion)	4-19
4.23	Perbandingan Coupling & Cohesion menurut pendekatan Single Linkage	4-20
4.24	Perbandingan Coupling & Cohesion menurut pendekatan Complete Linkage	4-20

4.25	Perbandingan Coupling & Cohesion menurut pendekatan Average Linkage	4-21
4.26	Konfigurasi kong-declarative di kong.yml	4-38
4.27	Penggalan Kode untuk disisipi proses JWT	4-39
4.28	Potongan Kode untuk memberikan validasi bawah JWT tidak valid .	4-40
4.29	Potongan Kode untuk memberikan validasi bawah JWT tidak valid .	4-40
4.30	Ilustrasi JWT di Cookie	4-41
4.31	Penerapan Class Abstract	4-42
4.32	Penerapan Class Adapter	4-42
4.33	Penerapan di Sisi Service-10 di kasus menambahkan PosCategory .	4-43
4.34	Diagram Database dari Product Tag dan Pos Category	4-44
4.35	Diagram Database dari Calendar Event Type	4-45

DAFTAR ALGORITMA

LAMPIRAN A	A-1
LAMPIRAN B	B-3

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-Oriente 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 sulit 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, dapat dikembangkan secara berkelanjutan, dan memiliki performa yang baik. 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 *cohesive* 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*. Pengenalan komponen 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]. Dalam membentuk partisi ini terdapat beberapa metode yang disebut linkage untuk menentukan partisi terdekat dengan sebuah objek yaitu jarak maksimum (complete linkage), jarak minimum (single linkage) dan nilai rata-rata jarak (average linkage) [13].

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 diuji dengan melihat *cohesion* dan *coupling* kemudian dilakukan pemilihan bagian yang di implementasikan. Dengan ini diharapkan bisa menyelesaikan permasalahan yang terjadi ketika migrasi aplikasi seperti mengenali komponen dan dapat dikembangkan secara berkelanjutan.

1.2 Rumusan Masalah

Berikut adalah rumusan masalah yang dibuat berdasarkan latar belakang diatas.

1. Bagaimana nilai *coupling* dan nilai *cohesion* dari microservice yang dibuat *Hierarchical Clustering*?
2. Bagaimana hasil dekomposisi yang dibuat dengan *linkage* yang berbeda?
3. Bagaimana penerapan microservice dengan menggunakan hasil dekomposisi yang dihasilkan oleh *hierarchical clustering*?

1.3 Tujuan Penelitian

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

1. Menggunakan *Hierarchical Clustering* untuk dekomposisi aplikasi ERP monolitik ke *microservice*
2. Membuat *microservice* yang memiliki nilai *coupling* rendah dan nilai *cohesion* tinggi.
3. Menemukan *linkage* yang cocok untuk dekomposisi *microservice* dengan *Hierarchical Clustering*

1.4 Batasan Masalah

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

1. Aplikasi yang didekomposisi adalah aplikasi yang sudah dibangun sebelumnya dan disebarluaskan dengan arsitektur monolitik.
2. Perubahan arsitektur tidak dapat menjamin secara keseluruhan fungsionalitas dari aplikasi, karena keterbatasan waktu dan pengujian.
3. Microservice yang diterapkan hanya pada bagian tertentu di aplikasi ERP yang dipilih.

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. Menghasilkan kelompok service yang ideal dari hasil dekomposisi.

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 dikelompokkan oleh *Hierarchical Clustering*. Pengujian mempertimbangkan nilai *cohesion*, nilai *coupling*, jumlah partisi, linkage dan melakukan pemilihan bagian untuk diimplementasikan menjadi *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*.

BAB 4: IMPLEMENTASI DAN PENGUJIAN

Implementasi dan Pengujian yang berisi pembangunan aplikasi dan pengujian yang mengevaluasi aplikasi yang 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 disebarluaskan 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, *Remote Method Invocation (RMI)* atau *Common Object Request Broker Architecture (CORBA)* [6].

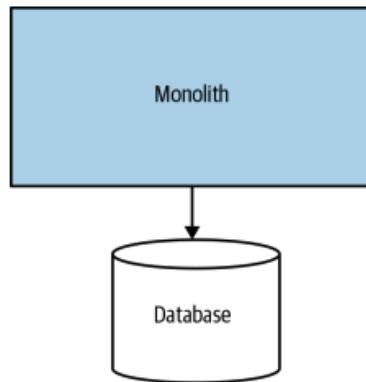
Penggunaan pada bahasa pemrograman seperti *Java, C/C++*, dan *Python* pada pengembangan aplikasi di sisi *server*, memiliki kemampuan dalam melakukan abstraksi untuk memecah kompleksitas program menjadi berupa modul. Namun, bahasa pemrograman ini dirancang untuk membuat *artefacts* monolitik. Di mana abstraksi ini tergantung pada penggunaan berbagai 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 disebarluaskan 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 disebarluaskan 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]:

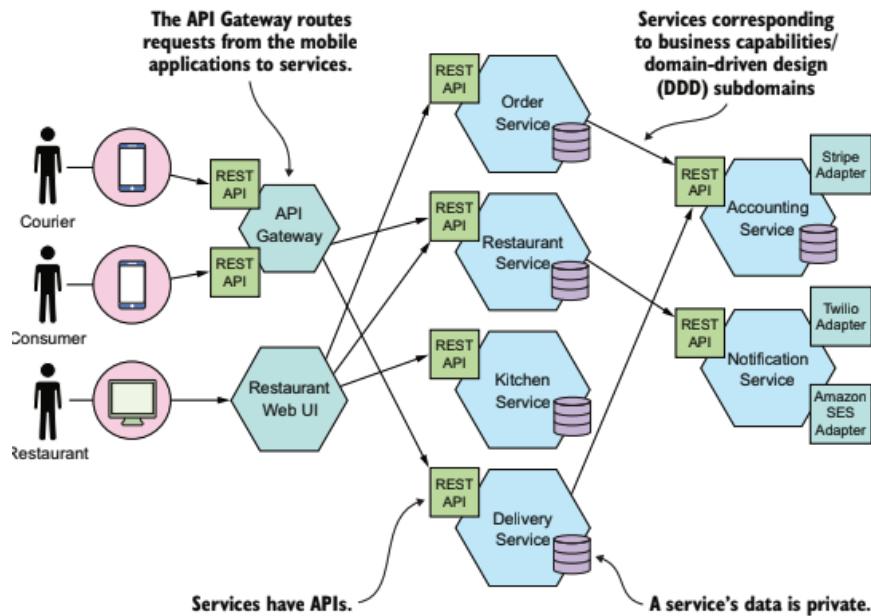
1. Sulit dikembangkan secara berkelanjutan, karena semakin banyak orang yang bekerja pada aplikasi yang sama. Akibatnya setiap pengembang memiliki kepentingan masing-masing dalam mengelola kode yang sama dan membuat pengambilan keputusan sulit serta tidak fleksibel
2. Memiliki reliabilitas yang rendah, 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 tantangan 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

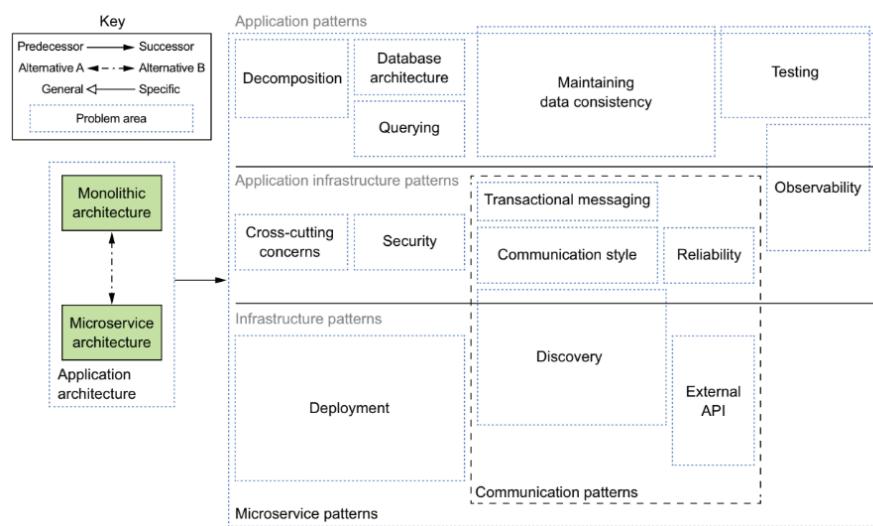
Setiap *service* untuk berkomunikasi antar *service* memiliki tantangan masing-masing seperti latensi, konsistensi data, dan kondisi ketika beberapa *service* mengalami kegagalan. *Microservice* juga meningkatkan kompleksitas operasional oleh karena itu untuk melakukan *deployment* sebaiknya menggunakan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

3. *Deployment* yang melibatkan beberapa service

Untuk melakukan *deployment* ini dibutuhkan koordinasi antara tim pengembang *service* 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 dapat 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 authentikasi pengguna umumnya dilakukan oleh *API Gateway*. Di mana *API Gateway* melanjutkan informasi ke *service* yang bertanggung jawab mengenai authentikasi, 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 perhatikan 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

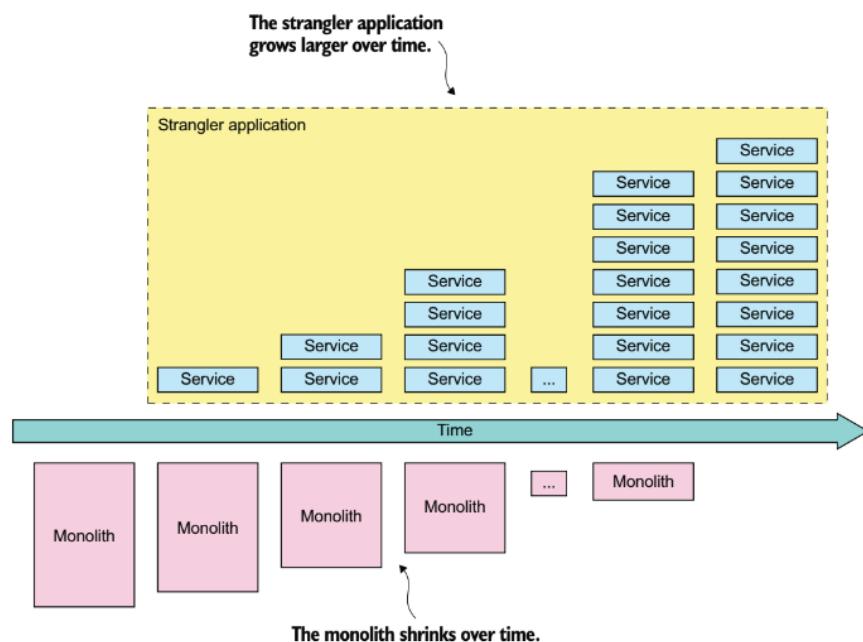
keseluruhan.

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 kelebihannya masing-masing [5]:

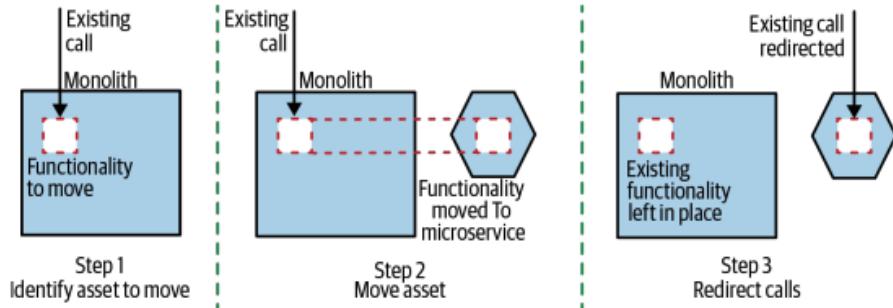
1. Pola *Strangle Fig*

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

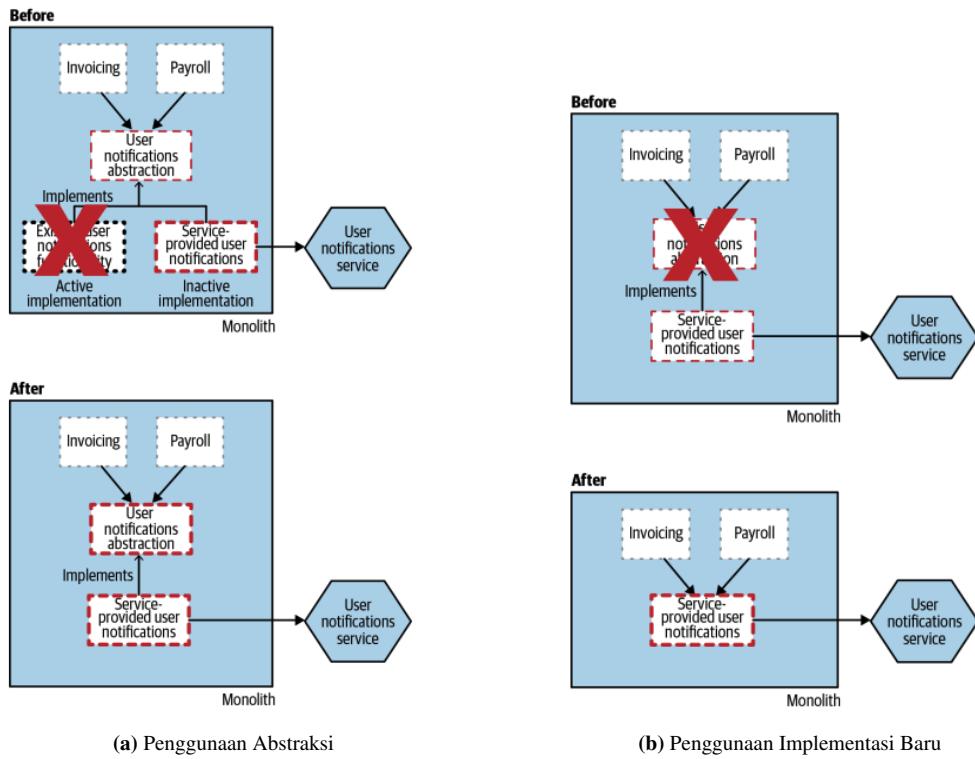
Pola *Strangle Fig* dapat dilakukan untuk memindahkan fungsionalitas namun ketika fungsionalitas itu berada di dalam sistem yang lebih dalam. Maka proses ekstraksi harus dilakukan tanpa mempengaruhi banyak sistem di mana sistem lain bisa berubah tanpa diketahui sistem yang diekstraksi.

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 di aplikasi. *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*(ci,cj) 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]

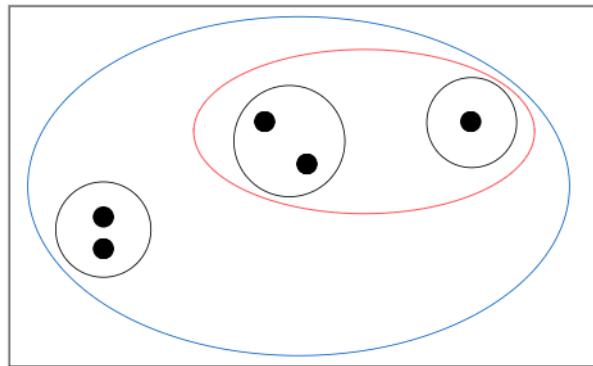
Mirip seperti Jaccard tapi Simple Matching menghitung hal yang tidak sama, jumlah hal yang tidak sama dibandingkan dengan jumlah yang sama. Kemudian dibagi jumlah hal yang tersedia pada objek tersebut.

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]:

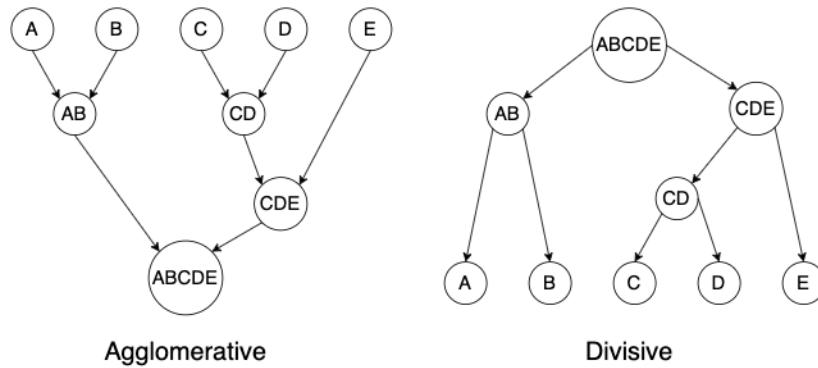
1. Hierarchical Clustering

Metode *Hierarchical Clustering* adalah sebuah prosedur untuk mentransformasi 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 oleh karena itu pendekatan ini berbedanya hanya dalam hal pemilihan prosedur daripada membuat klasifikasi yang berbeda [9].



Gambar 2.8 Perbedaan Agglomerative dan Divisive

Ada beberapa metode untuk menentukan partisi terdekat yaitu dengan menghitung jarak maksimal antara objek partisi (complete lingkage), nilai rata-rata jarak(average lingkage) atau jarak minimum (single lingkage) [13]. Proses untuk menghitung *Hierarchical Agglomerative Clustering*(HAC) menghasilkan stepwise matrix yang dapat digunakan untuk membuat dendogram. Proses ini menggunakan perbandingan jarak antar objek(r,s). Untuk mengabungkan 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 [9].

Step 1. Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.

Step 2. Find the least dissimilar pair of clusters in the current clustering, say pair $\{(r), (s)\}$, according to

$$d[(r), (s)] = \min \{d[(i), (j)]\}$$

where the minimum is over all pairs of clusters in the current clustering.

Step 3. Increment the sequence number: $m \leftarrow m + 1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m . Set the level of this clustering to

$$L(m) = d[(r), (s)]$$

Step 4. Update the proximity matrix, \mathcal{D} , by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster. The proximity between the new cluster, denoted (r, s) and old cluster (k) is defined as follows. For the single-link method,

$$d[(k), (r, s)] = \min \{d[(k), (r)], d[(k), (s)]\}$$

For the complete-link method,

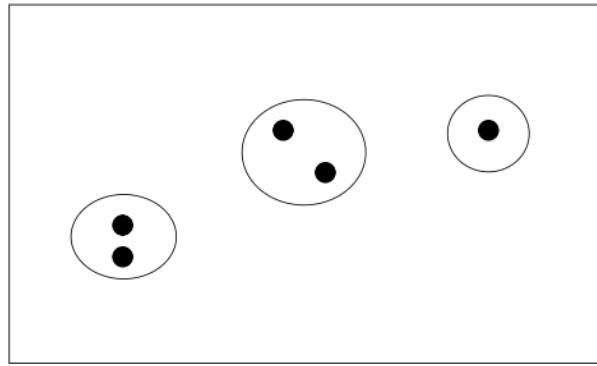
$$d[(k), (r, s)] = \max \{d[(k), (r)], d[(k), (s)]\}$$

Step 5. If all objects are in one cluster, stop. Else, go to step 2.

Gambar 2.9 Algoritma Stephen C. Johnson *Hierarchical Agglomerative Clustering* untuk Single Linkage dan Complete Linkage [9]

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-Partition 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 projek 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 automatisasi, 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 penmantauan serta pengecheckan status *service*. Plugins dapat menambahkan fitur pada Kong seperti

rate limiting untuk mengkontrol jumlah request yang dikirimkan ke suatu servie 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 pengecheckan tipe data, mendapatkan kode program, inpeksi *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	mengelompokan dengan nilai minimal atau nilai terdekat pada matriks kedekatan	y = matriks kedekatan
average	mengelompokan dengan nilai rata-rata jarak pada matriks kedekatan	y = matriks kedekatan

complete	mengelompokan dengan nilai maximal atau nilai terjauh pada matriks kedekatan	$y = \text{matriks kedekatan}$
dendogram	mengilustrasikan dendogram dari <i>cluster</i> yang terbentuk di matriks linkage	$Z = \text{matriks linkage}$

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
A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony (2020). "From monolithic architecture style to Microservice one based on a semi-automatic approach" [12]	Terdapat aplikasi monolitik yang tidak beradaptasi di <i>cloud</i> ataupun <i>DevOps</i> sehingga harus dimigrasi ke <i>microservice</i>	Analisis kode program dan mencari hubungan dalam <i>class</i> -nya	Identifikasi Microservice yang dibuat memiliki hasil yang memuaskan karena mempertimbangkan karakteristik microservice
Chaitanya K. Rudrabhatla. (2020). "Impacts of Decomposition Techniques on Performance and Latency of Microservices." [3]	Bagaimana dampak performa dalam menentukan batasan antar <i>service</i>	Melakukan perbandingan antara pendekatan DDD, Normalized Entity Relationship, dan Hybrid	Teknik DDD lebih baik dalam dekomposisi namun teknik Hybrid dengan mempertimbangkan fungsionalis dan transaksi yang terjadi memiliki performa lebih baik.

A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee (2021). "Mono2micro: A practical and effective tool for decomposing monolithic Java applications to microservices" [11]	Bagaimana pendekatan Mono2Micro dengan cara dekomposisi lainnya dan bagaimana tanggapan praktisi	Menggunakan hierarchical spatio-temporal decomposition yang menyimpan kondisi program secara dinamik berdasarkan eksekusi proses bisnis	Hasil rekomendasi <i>microservice</i> dapat membantu penerapan pola <i>strangle</i> , partisi yang dihasilkan sesuai dengan fungsi bisnis
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 microservice?	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 Pattern(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. Proses 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

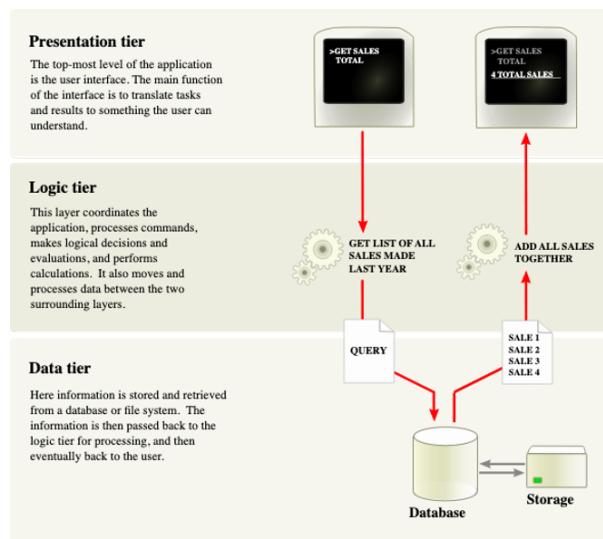
Pada bagian ini akan dijelaskan mengenai objek dan aplikasi terkait yang akan digunakan dalam tugas akhir ini. Object yang digunakan adalah sebuah aplikasi Enterprise Resource Planning yang di deploy secara monolitik, yaitu Odoo.

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].

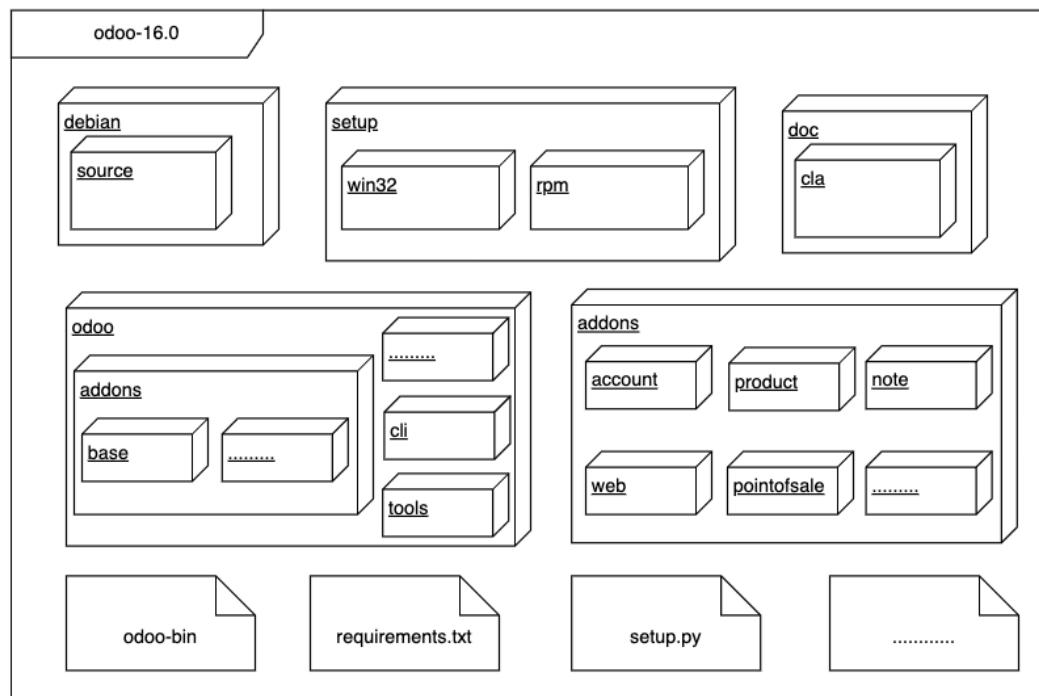


Gambar 2.11 Arsitektur Odoo [14]

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 add-ons [14].

BAB 2 LANDASAN TEORI

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> odoo.http.Controller
Static Web Data	File yang digunakan hanya ditampilkan kepada <i>client</i> di website	File gambar, File CSS, dan File JavaScript

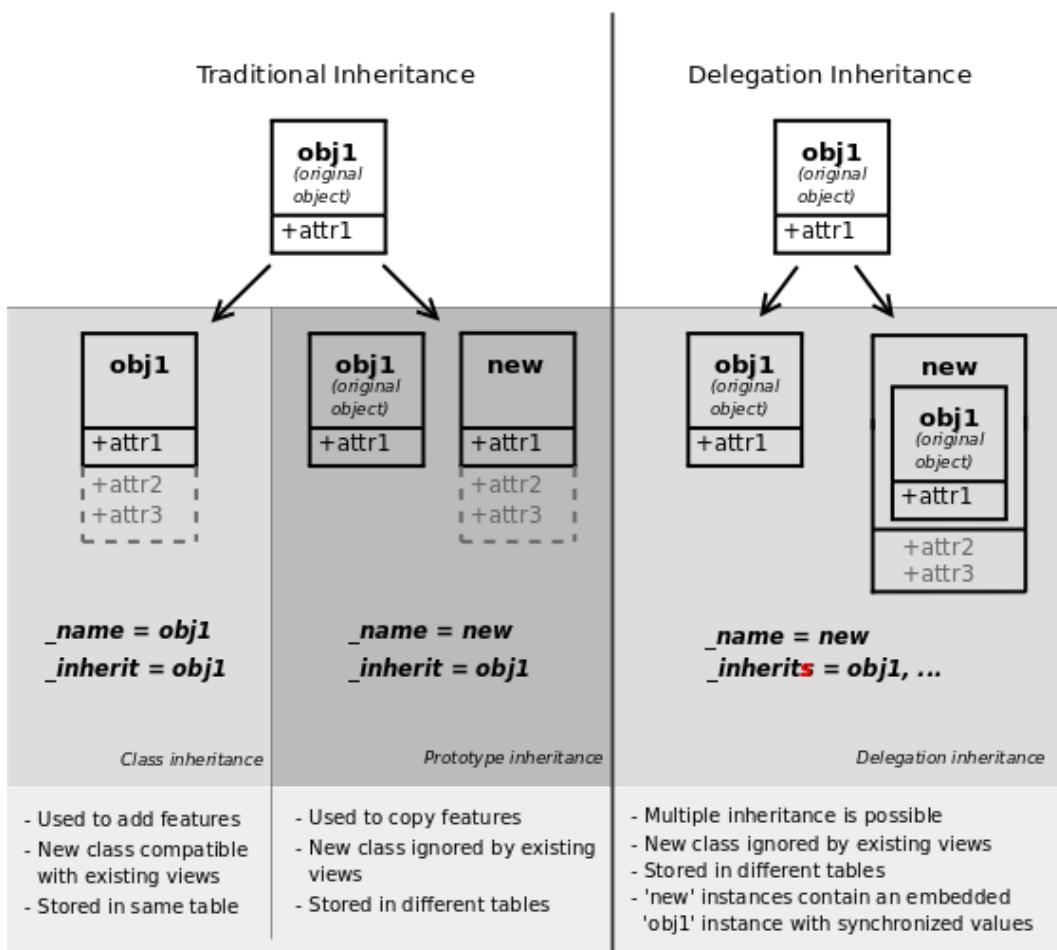
Terdapat 2 jenis addons yaitu addons base dan addons. Yang membedakannya addons base harus ada di setiap aplikasi Odoo bila tidak ada aplikasi tidak dapat berjalan sedangkan addons bisa diganti sesuai keperluan bisnis. Pengelolaan *database* dilakukan secara otomatis oleh Object Relational Mapping (ORM)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

Komponen utama pada Odoo yaitu pada bagian ORM, bagian ini menghindari proses penulisan SQL secara manual dan memudahkan untuk dikembangkan lebih lanjut serta meningkatkan keamanan. Objek bisnis ditulis dalam bentuk class Python yang merupakan turunan dari Model, dimana model ini dapat di integrasikan ke dalam sistem. Model dapat dikonfigurasi berdasarkan pengaturan atributnya di class tersebut. Atribut yang paling utama yaitu *_name* karena *_name* akan diketahui oleh semua model di model Odoo. Terdapat 3 bentuk turunan lainnya pada Model yaitu AbstractModel, Model, dan TransientModel. Perbedaan diantara Model ini adalah 'AbstractModel' tidak menyimpan data di database, 'Model' menyimpan data di database dan 'TransientModel' menyimpan

data di database sementara dalam waktu tertentu.

Model Odoo merupakan turuan dari class BaseModel dimana BaseModel memiliki metaclass yang disebut MetaModel, pada class BaseModel memiliki atribut yaitu `_name` yaitu nama model dari class tersebut dengan format nama path dot(.) contohnya 'product.category', Penamaan ini mempengaruhi nama tabel pada database sehingga nama tabel di database berdasarkan contoh tersebut 'product_category'. Atribut lainnya yaitu seperti `_inherit`, `_inherits`, dan `comodel_name`.

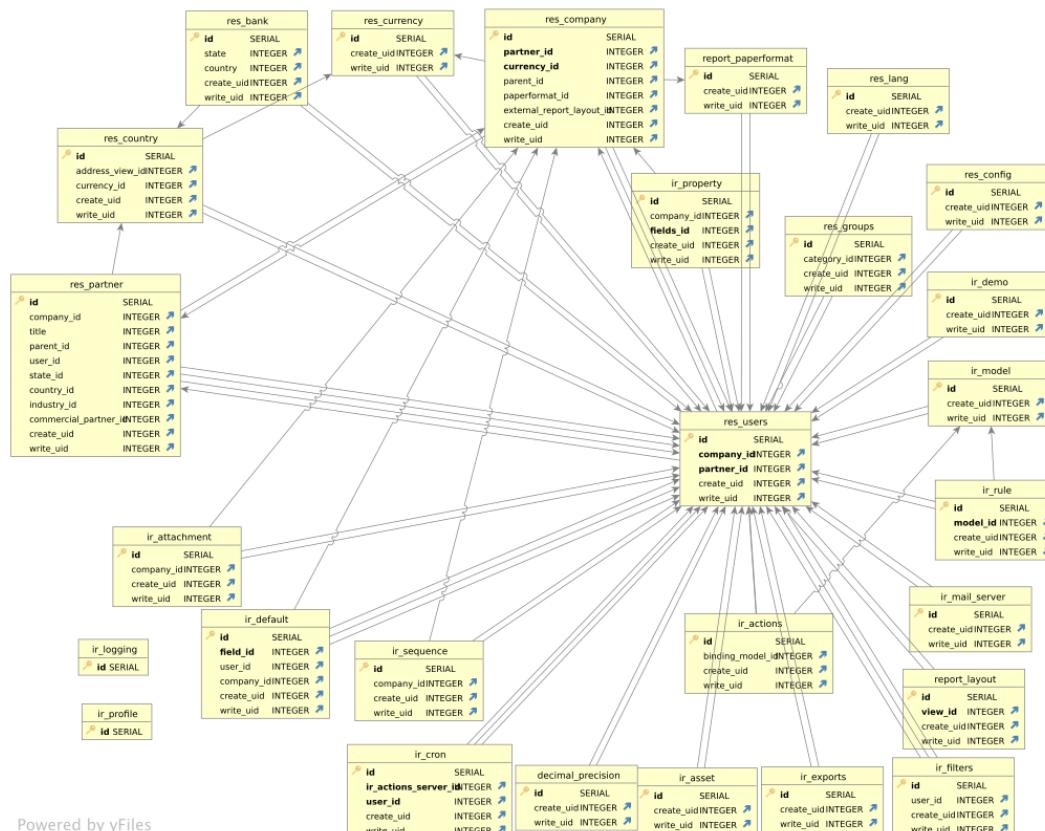
Model Odoo bisa memiliki banyak parent yang diturunkan sehingga tipe atribut `_inherit` bisa berupa str atau list. Apabila atribut nama tidak ditentukan maka model tersebut akan mengupdate parent tersebut (termasuk datanya) sedangkan bila tidak maka model tersebut akan memiliki relasi dengan parent dan data dianggap sebagai data terpisah. Sedangkan attribute `_inherits` menggunakan *inheritance* dengan bentuk *delegation*, dengan ini model memiliki referensi antar model lainnya tetapi model tersebut bukanlah menjadi sebuah model lain tersebut. Pada gambar 2.13 dapat dilihat lebih jelas ilustrasi inheritance pada model Odoo.



Gambar 2.13 Inheritance pada Model Odoo

Atribut comodel_name merupakan atribut yang dimiliki sebuah field relational odoo. Untuk membuat relasi dengan model lainnya seperti one-to-many, many-to-one dan many-to-many, dilakukan dengan menambahkan atribut dengan class field. Di comodel_name merupakan string yang merepresentasikan _name dari model yang ingin direlasikan.

Tabel yang terbentuk pada konfigurasi awal Odoo umumnya bisa mencapai ±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.



Gambar 2.14 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 Analysis Masalah

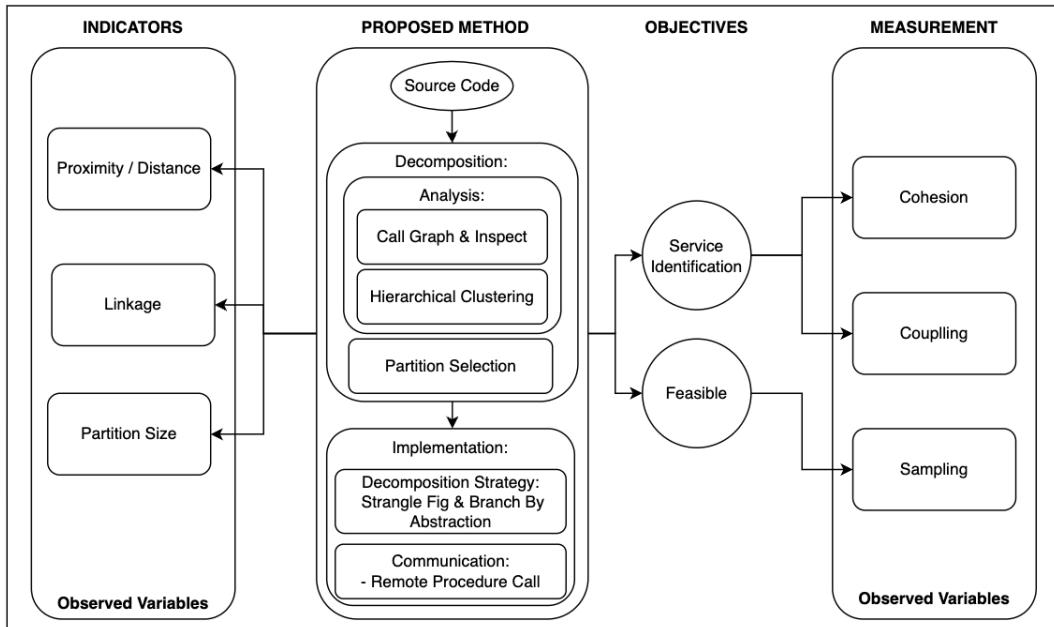
Arsitektur ERP yang digunakan pada Odoo yaitu arsitektur monolitik, seperti yang sudah dijelaskan pada landasan teori. Arsitektur monolitik memiliki kelemahan dan permasalahan yang bisa diselesaikan dengan arsitektur *microservice*.

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.

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. Linkage yang digunakan dalam melakukan Hierarchical Clustering yaitu single, complete, dan average, kemudian Partisi dipilih melalui nilai secara struktural yaitu nilai *coupling* dan *cohesion*. Hasil terbaik dari clustering disampling untuk diimplementasikan menjadi *service*, penelitian ini akan menggunakan strategi dengan pola *strangle fig* dan *branch by abstraction* untuk memecah kode di monolitik.

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.

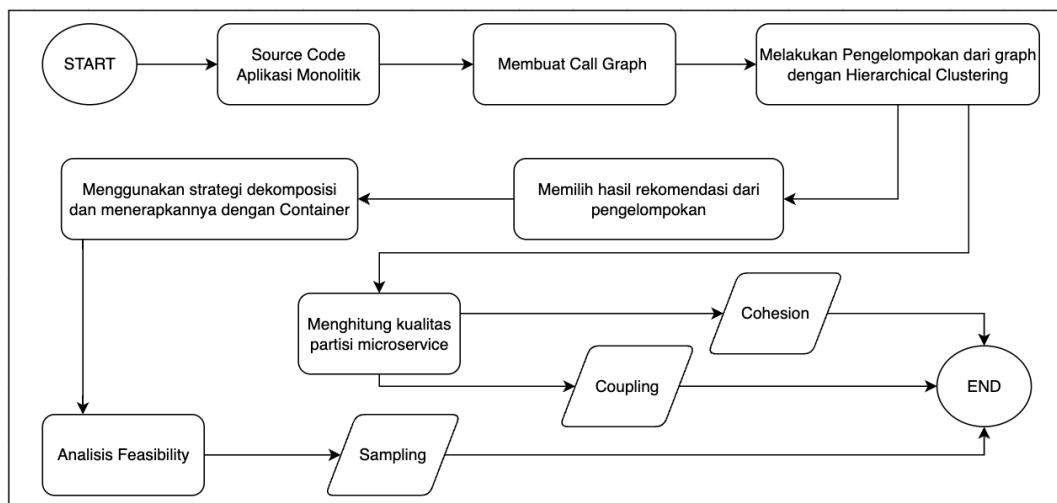
Dari *graph* yang sudah dibuat akan dilakukan pengelompokan dengan algoritma Hierarchical Clustering. Pendekatan algoritma yang digunakan pada penelitian ini yaitu agglomerative, selain itu ditentukan cara menghitung kedekatan antara objek dan pemilihan algoritma Linkage. Metode linkage yaitu menentukan 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 menggunakan metodenya satu sama lain. Dengan membandingkan nilai *coupling*, *cohesion*, linkage dan ukuran partisi maka dapat diketahui kelompok service yang

ideal.

Selain itu, untuk mengetahui apakah hasil clustering relevan dan dapat diimplementasikan maka dilakukan sampling yang berjumlah 2 service dari kelompok service yang ideal. Untuk menerapkannya pemecahan dimulai dari pemecahan kode yang menggunakan strategi dekomposisi *strangle fig* dan *branch by abstraction*. Untuk metode komunikasinya antara *service* yaitu dengan Remote Procedure Call(RPC) dan untuk mengelola datanya, setiap *service* memiliki databasenya masing-masing.

3.3 Urutan Proses Global

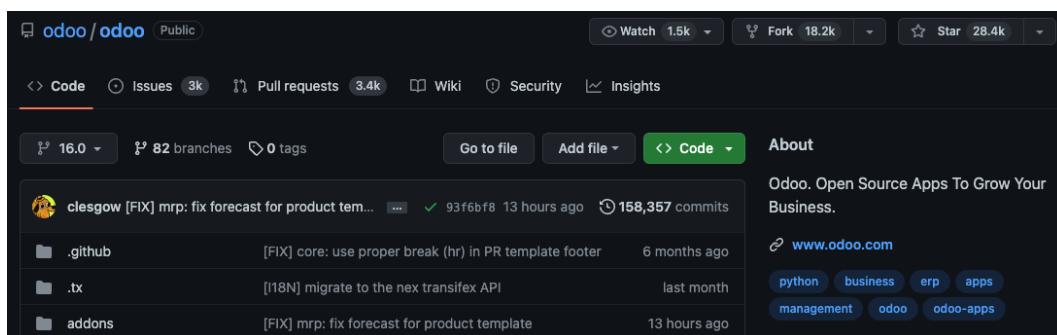


Gambar 3.2 Diagram Flowchart Proses Global

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 instalasi library, module dan Package yang digunakan dari file requirement.txt

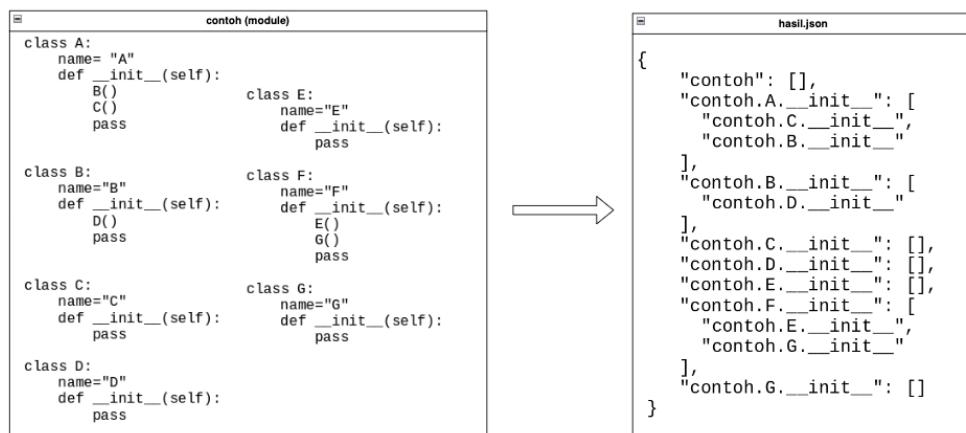


Gambar 3.3 Source Code Aplikasi Odoo pada git repository

3.3.1.2 Pembuatan Call Graph

Pada tugas akhir ini menggunakan tools PyCG untuk menghasilkan *call graph* dalam bentuk format JSON. Terdapat target folder yang harus dianalisis oleh PyCG *call graph* yaitu folder 'odoo/addons' atau package 'odoo.addons'. Pemilihan folder ini dikarenakan folder/package lainnya tidak memiliki hubungan mengenai proses bisnis. Untuk menghemat waktu pembuatan *call graph* maka folder test dan l10n (localization) tidak dianalisis.

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 terhubungan 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.



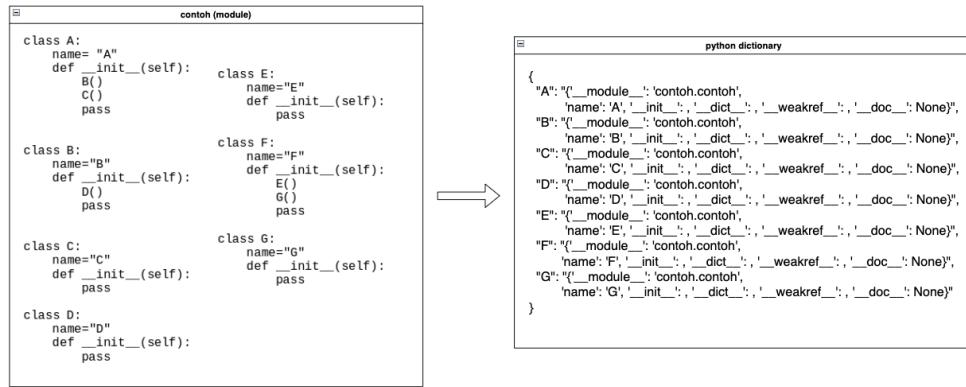
Gambar 3.4 Proses Pembuatan Call Graph dengan PyCG

Proses ekstraksi json yang dihasilkan dari tools PyCG berupa file JSON yang dinamakan berdasarkan argument yang diberikan sebelumnya seperti 'addons.json'. File JSON diubah menjadi *graph* yang direpresentasikan dalam bentuk adjacency list di Python.

3.3.1.3 Ekstraksi Dependency Model

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 pemrograman dinamik di mana pengecekan tipe data dilakukan secara 'run-time'. Ekstrasi ini difokuskan pada module yang memiliki proses bisnis seperti module 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 objek Python lebih mendalam

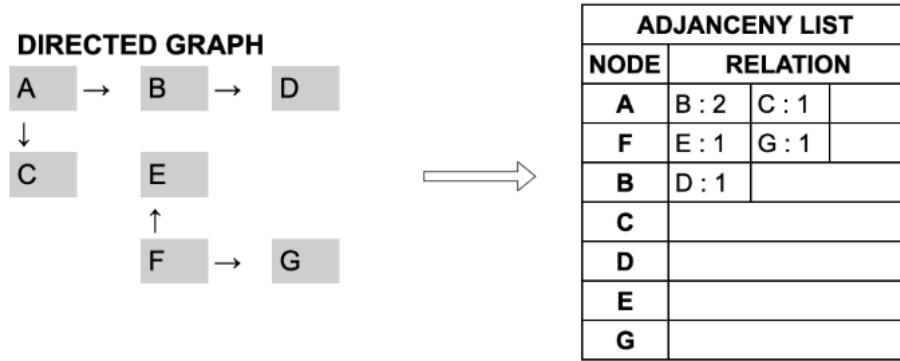
Object yang dianalisis yaitu *class* yang merupakan turunan dari *class* odoo.models.MetaModel, di mana *class* MetaModel memiliki properti seperti *name*, *_inherits*, *_inherit*, dan *comodel_name*. Hasil ekstrasi dependency model digabungkan melalui nama module PyCG.

3.3.1.4 Pengabungan dan Optimisasi Hasil Ekstraksi

Graph yang dihasilkan dari proses ekstrasi 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 campuran yaitu berdasarkan module dan file, yang dikelompokkan menjadi file hanya addons base.

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*.

Pada gambar 3.6 dapat dilihat hasil dari graph yang dihasilkan dari contoh PyCG dan Inspect sebelumnya. Graph yang digunakan merupakan directed sehingga setiap pemanggilan berlaku untuk satu arah. Graph ini dapat diubah menjadi bentuk dictionary sebagai adjacency list, tujuannya agar pengaksesan dan pencarian node lebih cepat dan mudah dilakukan. Setiap relasi memiliki jumlah berapa kali node lain dipanggil dari node tersebut.

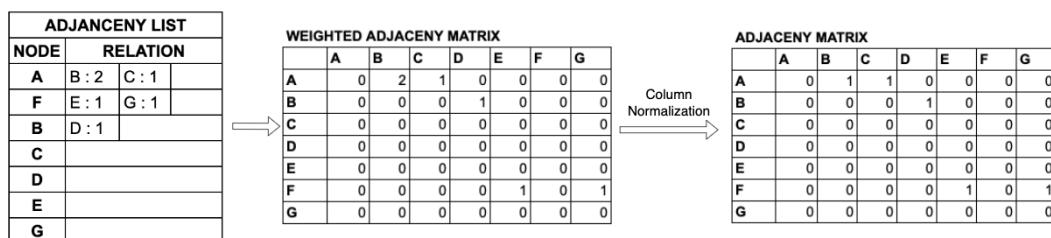


Gambar 3.6 Graph dan Adjacency List

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. Jaccard menghasilkan hasil yang bagus pada remodularisasi perangkat lunak dan Struktural Similarity digunakan untuk melihat kedekatan dari sisi intesitas panggilan antara module.

Berikut pada gambar 3.7 dilakukan proses perubahan dari adjacency list menjadi adjacency matrix yang memiliki bobot. Kemudian dilakukan normalisasi kolom, dimana setiap nilai dibandingkan dengan nilai maximum dan nilai minimum pada kolom yang sama.



Gambar 3.7 Perubahan dari Adjacency List menjadi Adjacency Matrix

Proses perhitungan jarak antara 2 objek dilakukan dari adjacency matrix dengan pola urutan berbentuk matriks segitiga bawah. Dengan ini perbandingan pertama kali dilakukan di baris ke-2(index=1) di kolom ke-1(index=0) dan berakhir pada baris ke-n(jumlah objek) di kolom ke n-1. Pada contoh di 3.8 dijelaskan untuk perhitungan total jarak pada objek B dan A. Sebelum dilakukan perhitungan matriks terlebih dahulu sisi diagonal diberi nilai 1 untuk mengartikan

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

bahwa objek memiliki hubungan dengan dirinya sendiri. Perhitungan dimulai menghitung kemiripan jaccard, jaccard menghasilkan kemiripan dari jumlah interseksi dibagi jumlah union pada objek. Perhitungan selanjutnya Similarity Struktural menghitung dengan mempertimbangkan jumlah sisi panggilan kedua objek yaitu baris ke-1(index=0), baris ke-2(index=1) dan jumlah sisi panggilan keluar kedua objek yaitu kolom ke-1(index=0) dan kolom ke-2(index=1). Hasil kemiripan Similarity Struktural dan Jaccard dirata-ratakan untuk menjadi nilai akhir kemiripan objek.

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

Row: 1 (B) Col : 0 (A) Jaccard (Jaccard Similarity)							
row	0	1	0	1	0	0	0
col	1	1	1	0	0	0	0
intersection (&)	0	1	0	0	0	0	= (SUM) 1
union ()	1	1	1	1	0	0	= (SUM) 4
							Kemiripan A dengan B: 0.25

Row: 1 (B) Col : 0 (A) SimStr (Similarity Structural)							
ci	0	1	0	1	0	0	0
cj	1	1	1	0	0	0	0
i	1						
j	0						
callsinCi	1	1	0	0	0	0	= (SUM) 2
callsinCj	1	0	0	0	0	0	= (SUM) 1

Perhitungan Berdasarkan Kondisi							
if callsinCi > 0 dan callsinCj > 0							
$0.5 \times ((c_i[j] / callsinCj) + (c_j[i] / callsinCi))$							
$0.5 \times ((0/1) + (1/2)) = 0.25$							
elif callsinCi = 0 dan callsinCj > 0							
TIDAK TERPENUHI							
elif callsinCi > 0 dan callsinCj = 0							
TIDAK TERPENUHI							
else							
TIDAK TERPENUHI							
							Kemiripan A dengan B: 0.25

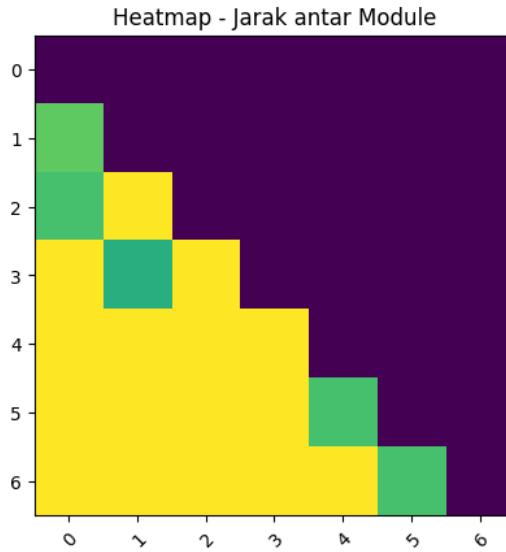
Gambar 3.8 Perhitungan Jarak

Distance Matriks							
	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	1
G	1	1	1	1	1	0.71	0

Gambar 3.9 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 3.10 Heatmap yang dihasilkan dari Distance Matrix

Pemilihan pengelompokan dengan hierarchical agglomerative clustering dibandingkan Parition clustering karena tidak mudah untuk mengetahui jumlah ideal *cluster*. Untuk menentukan metode *linkage* tugas akhir ini menggunakan single lingkage, average linkage, dan complete lingkage. Hasil dari masing-masing lingkage dipilih jumlah partisi yang ideal untuk *microservice*. Penggunaan single lingkage 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 lingkage menghasilkan bentuk partisi di antara complete lingkage dan single lingkage.

Pada gambar 3.11 dijelaskan mengenai proses perhitungan hierarchical clustering dengan distance matriks yang sebelumnya sudah dihitung. Perhitungan hierarchical clustering terdapat 5 bagian besar yaitu pertama memiliki objek pertama (urutan pertama), ke-2 mencari objek lainnya yang terdekat berdasarkan distance matrix, ke-3 mengabungkan objek yang terdekat dengan objek yang pertama dan membuat kelompok baru dari objek tersebut, ke-4 memperbaharui nilai jaraknya pada kelompok baru / partisi menggunakan linkage. Penggunaan linkage akan membandingkan 2 nilai objek yaitu dari nilai objek yang baru dan nilai objek yang sebelumnya (yang terdapat di distance matriks) seperti min memilih nilai yang lebih rendah, complete memilih nilai yang lebih tinggi, dan

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

average melakukan nilai rata-rata dari 2 nilai objek. Langkah terakhir bila sudah tidak ada objek tersisa maka perhitungan berhenti bila masih ada maka dilakukan kembali proses yang ke-2.

Menghitung Clustering dengan Linkage Single

1 Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.

$L = A(0)$

$M = 0$

2 Find the least dissimilar pair of clusters in the current clustering

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

3 Increment the sequence number: $m = m + 1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m .

$M = 1$

	AC	B	D	E	F	G
A	0.75	1	1	1	1	1
C	0.71	1	1	1	1	1

4 Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster.

Single Linkage : $\text{MIN}(AB, CB) \dots\dots$

$B = \text{MIN}(0.75, 1)$

$D = \text{MIN}(1, 1)$

$E = \text{MIN}(1, 1)$

$F = \text{MIN}(1, 1)$

$G = \text{MIN}(1, 1)$

	AC	B	D	E	F	G
AC	0.71	0.75	1	1	1	1

Distance Matrix

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

Complete Linkage : $\text{MAX}(AB, CB) \dots\dots$

$B = \text{MAX}(0.75, 1)$

$D = \text{MAX}(1, 1)$

$E = \text{MAX}(1, 1)$

$F = \text{MAX}(1, 1)$

$G = \text{MAX}(1, 1)$

	AC	B	D	E	F	G
AC	0.71	1	1	1	1	1

Complete Linkage : $\text{MAX}(AB, CB) \dots\dots$

$B = \text{MAX}(0.75, 1)$

$D = \text{MAX}(1, 1)$

$E = \text{MAX}(1, 1)$

$F = \text{MAX}(1, 1)$

$G = \text{MAX}(1, 1)$

	AC	B	D	E	F	G
AC	0.71	0.75	1	1	1	1

5 If all objects are in one cluster, stop. Else, go to step 2.

Gambar 3.11 Proses Perhitungan Hierarchical Clustering

CONTINUE

2 Find the least dissimilar pair of clusters in the current clustering

	B	D	E	F	G
AC	0.75	1	1	1	1

3 Increment the sequence number: $m = m + 1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m .

$M = 2$

	ACB	D	E	F	G
AC	0.75	1	1	1	1
B	0.62	1	1	1	1

4 Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster.

Single Linkage : $\text{MIN}(AB, CB) \dots\dots$

$D = \text{MIN}(0.62, 1)$

$E = \text{MIN}(1, 1)$

$F = \text{MIN}(1, 1)$

$G = \text{MIN}(1, 1)$

	ACB	D	E	F	G
ACB	0.75	0.62	1	1	1

5 If all objects are in one cluster, stop. Else, go to step 2.

Distance Matrix

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

Gambar 3.12 Lanjutan Proses Perhitungan Hierarchical Clustering

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

CONTINUE

	D	E	F	G
ACB	0.62	1	1	1

3 M = 3

	D	E	F	G
ACB		1	1	1
D	0.62	1	1	1

4 Single Linkage : MIN(AB,CB)

E = MIN(1,1)	F = MIN(1,1)
G = MIN(1,1)	
ACBD	0.62

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

5 If all objects are in one cluster, stop. Else, go to step 2.

CONTINUE

	ACBD	E	F	G
ACBD	0.62	1	1	1

3 M = 4

	E	F	G
ACBD	1	1	1
E	0.71	1	1

4 Single Linkage : MIN(AB,CB)

F = MIN(1,0.71)	G = MIN(1,1)
ACBDE	1
ACBDE	0.71

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

5 If all objects are in one cluster, stop. Else, go to step 2.

Gambar 3.13 Lanjutan Proses Perhitungan Hierarchical Clustering

CONTINUE

	ACBDE	F	G
ACBDE	1	0.71	1

CONTINUE

	ACBDEF	G
ACBDEF	1	0.71

3 M = 5

	F	G
ACBDE	0.71	1
F	0.71	0.71

3 M = 6

	G
ACBDEF	0.71
G	0.71

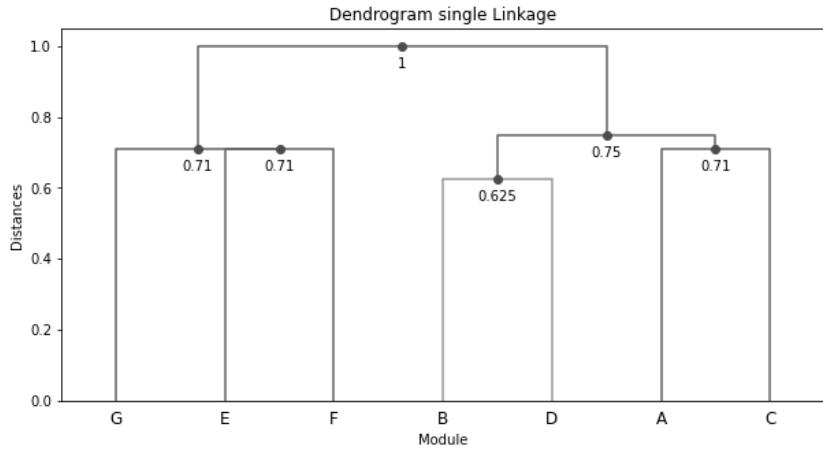
	ACBDEFG
ACBDEFG	0.71

5 If all objects are in one cluster, stop. Else, go to step 2.
STOP

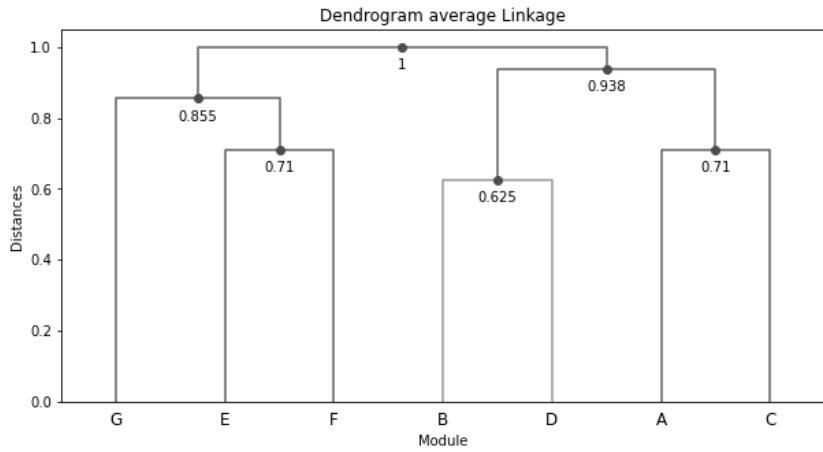
5 If all objects are in one cluster, stop. Else, go to step 2.

Gambar 3.14 Lanjutan Proses Perhitungan Hierarchical Clustering

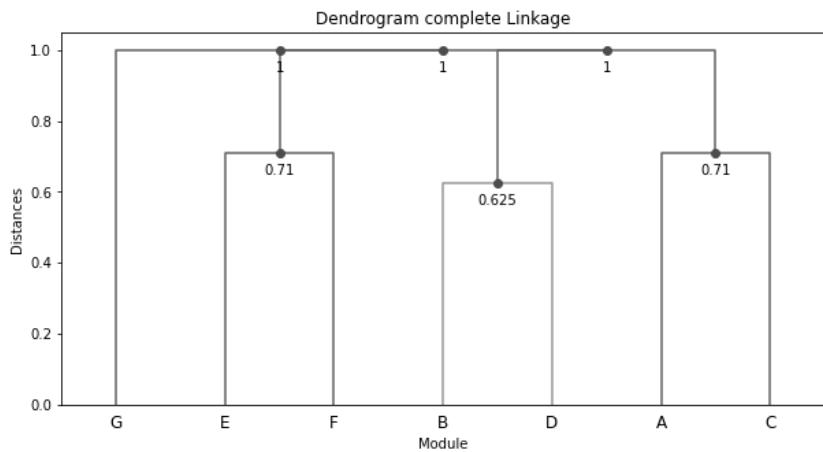
Hasil pengelompokan dapat ditampilkan dalam bentuk dendogram. Di mana pengelompokan dari setiap linkage memiliki dampak berbeda yang bisa dilihat dari bentuk dan nilai kedekatan antar partisi melalui dendogram dan bentuk relasi. Angka gabungan yang ditampilkan pada dendogram merupakan jarak linkage yang dihitung dari proses perhitungan hierarchical clustering sebelumnya.



Gambar 3.15 Dendrogram Single Linkage



Gambar 3.16 Dendrogram Average Linkage



Gambar 3.17 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.

Proses pemilihan partisi dimulai dari pemotongan tree dari hasil hierarchical clustering sebelumnya. Pemotongan dilakukan berdasarkan jumlah cluster / partisi, kemudian dipetakan dalam bentuk directed graph untuk menyusun kelompok partisi yang berisi relasi objek-objek / modul. Didalam node graph setiap hubungan module diluar dari partisinya akan dihitung sebagai panggilan ke luar dan hubungan antar module yang masih didalam dihitung sebagai panggilan internal partisi.

**GRAPH BERUPA DICTIONARY DARI PARTISI
HIERARCHICAL CLUSTERING**

{

 "0": {

 "A": {

 "1": 2,

 "C": 1

 },

 "C": 0

 },

 "1": {

 "B": {

 "D": 1

 },

 "D": 0

 },

 "2": {

 "E": 0,

 "F": {

 "E": 1,

 "G": 1

 },

 "G": 0

 }

}

DIRECTED GRAPH



EFG

ADJACENCY MATRIX

	0	1	2
0	0	1	0
1	0	0	0
2	0	0	0

INTERNAL CALL

A	3
B	2
C	1
D	1
E	1
F	3
G	1

Gambar 3.18 Proses pemotongan tree dan perubahan menjadi Adjacency Matrix dengan linkage single sejumlah 3 partisi

Module juga memiliki panggilannya internal masing-masing sebelumnya karena didalam module bisa berisi banyak sub-module lainnya, data ini disimpan dalam bentuk key-value dictionary. panggilan internal dibutuhkan untuk membuat

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

kohesi yang benar dan panggilan eksternal (diluar partisi) digunakan untuk menghitung nilai coupling. Pada tugas akhir ini menggunakan Structural and Behavioral Dependencies untuk mengevaluasi hasil partisi yang dihasilkan oleh Hierarchical Clustering. Node graph diubah kembali menjadi Adjacency Matrix untuk memudahkan akses seperti komparasi hubungan antar partisi.

$$\text{NbDirectConnections} = 3+1+1+2+1+1+1+3+1+1+1 = 16$$

$$\text{NbPossibleConnection} = 2 + \text{NbDirectConnections} = 18$$

$$\text{Cohesion} = 16 / 18 = \mathbf{0.88}$$

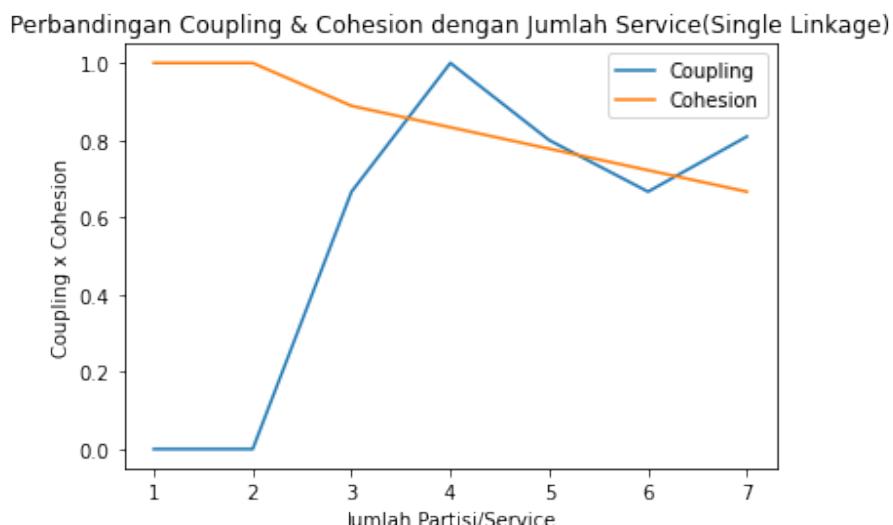
$$\begin{aligned}\text{Sum(CoupP)} &= ((0+0) / 2) + ((0+1) / 1) + ((0+0) / 1) + ((1+0) / 1) + 0 + 0 + ((0+0) / 1) + 0 + 0 + 0 \\ &= (0+1+0+1+0+0+0+0+0) = 2\end{aligned}$$

$$\text{NbPossiblePairs} = 3$$

$$\begin{aligned}\text{Coupling} &= \text{Sum(CoupP)} / \text{NbPossiblePairs} \\ &= 2 / 3 = \mathbf{0.667}\end{aligned}$$

Gambar 3.19 Proses perhitungan coupling dan cohesion

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.

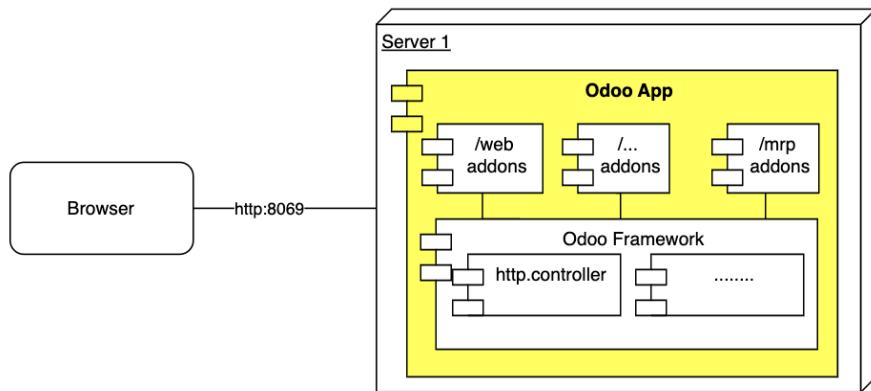


Gambar 3.20 Perbandingan dari nilai Cohesion dan nilai Coupling dengan jumlah *cluster/partisi* menggunakan Single Linkage

3.3.2 Dekomposisi Monolitik ke Microservice

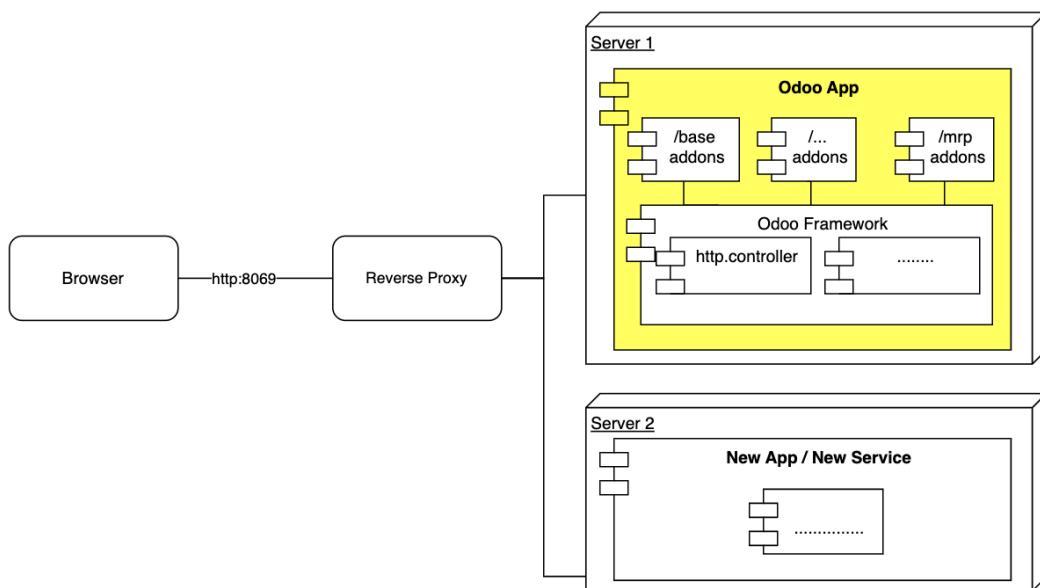
3.3.2.1 Strategi Pemisahan Kode

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 yang dinamik. Pada gambar 3.21 terlihat bahwa aplikasi odoo merupakan aplikasi berbasis client-server.



Gambar 3.21 Arsitektur di Monolitik

Proses pemisahan membutuhkan *reverse proxy* yang dapat menghubungkan client dengan banyak server. 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.

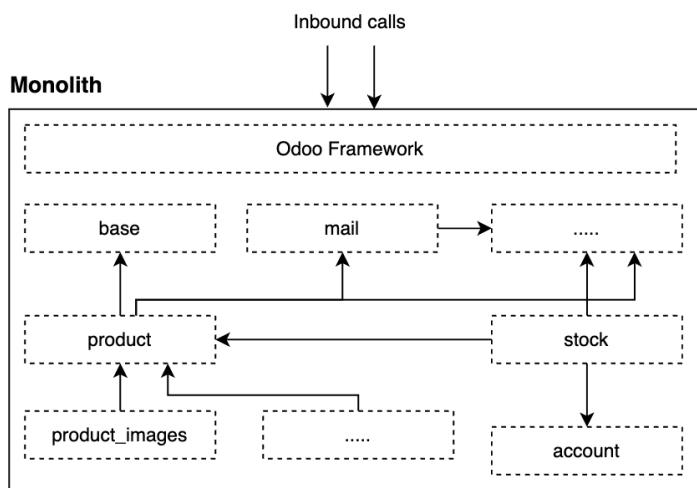


Gambar 3.22 Arsitektur di Microservice

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.

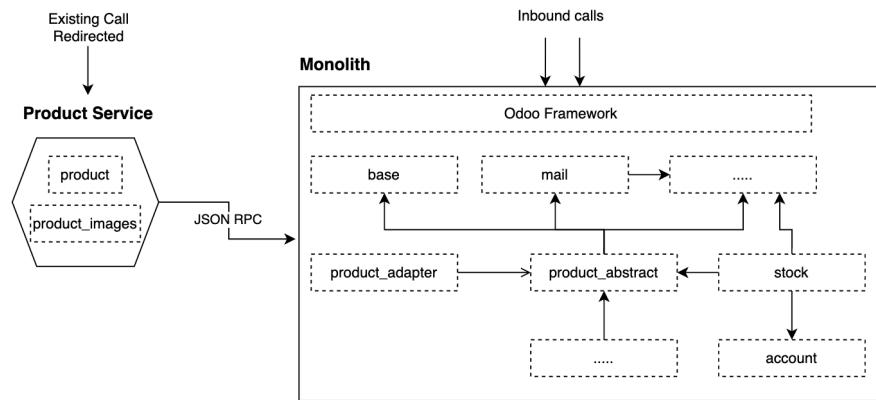
Tugas Akhir ini menggunakan *API Gateway* Kong (*off-the-shelf*) karena Kong sudah memiliki fitur yang lengkap pada kasus migrasi aplikasi monolitik ke *microservice*. Fitur itu berupa kemampuan untuk redireksi url untuk menerapkan proses *strangle*, pemantauan *service*, dan memiliki performa yang baik.



Gambar 3.23 Ilustrasi Struktur Module dan Keterhubungannya di Aplikasi Monolitik

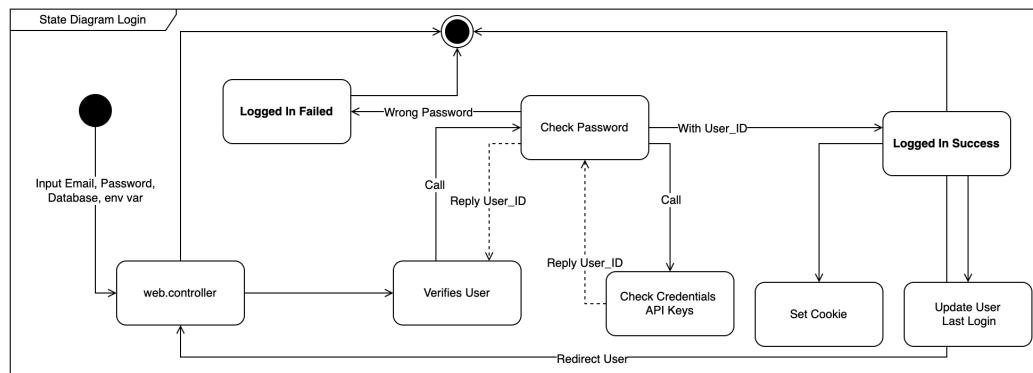
Terdapat 3 langkah utama dalam menerapkan pola *strangle* yaitu memilih bagian yang ingin dipindahkan, memindahkan aplikasi menjadi *service* yang berdiri sendiri, dan yang terakhir mengubah *call* dari monolitik ke *service* yang baru dibuat.

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 3.24 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 JSON Web Token (JWT) tapi bentuk HTTP session. Untuk itu diperlukan modifikasi pada sistem autentikasi yang menggunakan format JWT agar setiap *service* tidak perlu menvalidasi berkali-kali apakah sesi itu valid.



Gambar 3.25 State Diagram pada proses login

3.3.2.2 Komunikasi antar service

Proses komunikasi antar *service* dilakukan melalui JSON-RPC karena Odoo sudah memiliki untuk setiap add-onsnya RPC ini bisa berupa XML atau JSON. Komunikasi ini melalui protokol HTTP agar bisa akses oleh browser.

3.3.2.3 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.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

4.1 Lingkungan Implementasi

Pada bagian ini dijelaskan mengenai spesifikasi perangkat keras dan kondisi lingkungan perangkat lunak yang digunakan selama proses implementasi dekomposisi dan pengujian.

4.1.1 Spesifikasi Perangkat Keras

Berikut spesifikasi perangkat keras untuk proses implementasi dekomposisi:

1. Nama Produk : Macbook Air 2020
2. Processor : Apple M1 8 Core CPU
3. Memori : 8 GB
4. Penyimpanan : 256 GB

4.1.2 Lingkungan Perangkat Lunak

Berikut lingkungan perangkat lunak:

1. Sistem Operasi: macOS Ventura
2. IDE : Visual Studio Code, GoLand
3. Tools : Dbeaver, Postman, Docker, Primate
4. Bahasa Pemrograman: Python 3.8.2rc2, Go v1.20.4

4.2 Implementasi Proses Clustering

Pada bagian ini, dijelaskan mengenai proses, fungsi, dan metode yang digunakan dalam proses clustering untuk menemukan kelompok service di aplikasi monolitik yaitu Odoo.

4.2.1 Pengambilan Source Code

Proses ini pengambilan source code aplikasi Odoo bisa melalui source code repository di github.com. Source code tersebut digunakan untuk proses clustering dan proses dekomposisi dari monolitik ke microservice. Pada tugas akhir ini menggunakan Odoo v16 pada commit yang bisa direferensikan SHA1 375d0db3.

```
$ git init
```

```
$ git remote add origin https://github.com/odoo/odoo
$ git fetch origin 375d0db3694419942a95e58212295b4186085e61:
    refs/remotes/origin/16.0 --depth=1
$ git checkout 375d0db3694419942a95e58212295b4186085e61
```

Listing 4.1: Shell Script Git untuk pengambilan source code

4.2.2 Pembuatan Call Graph

Untuk membuat call graph dari source code aplikasi dapat menggunakan alat PyCG. PyCG harus terlebih dahulu di install bisa melalui pip. PyCG menganalisis kode program Python melalui package yang diberikan, untuk nama folder / package yang mengandung karakter ”.” harus diubah menjadi karakter lain karena PyCG membacanya dalam bentuk package Python.

Proses analisis dilakukan pada package odoo.addons dengan pengecualian folder test(pengujian). Hasil PyCG berupa .json yang kemudian diproses selanjutnya, proses ini membutuhkan waktu yang cukup lama karena besarnya aplikasi.

```
$ pip install pycg
$ py_files=$(find addons -type f -name "*.py" -not -path "
    */tests/*")
$ pycg --package addons $py_files -o odooAddons.json
```

Listing 4.2: Shell Script untuk pembuatan call graph

```
real    561m4.651s
user    544m13.048s
sys     9m11.639s
asa@pop-os:/media/asa/A SSD/ClusteringForDekomposisi/odoo$
```

Gambar 4.1 Waktu pembuatan call graph dengan PyCG

Berikut adalah hasil dari scan yang dibuat PyCG. Dapat dilihat ada panggilan fungsi randint dari package random. Dari data ini tidak dapat diketahui apakah hubungan itu berupa module, class atau metode, untuk itu diproses selanjutnya akan menghilangkan informasi yang tidak penting dalam pengelompokan.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```

Hasil Scan JSON PyCG pada file pos_category.py
{
    "point_of_sale.models.pos_category": [
        "odoor.fields.Boolean",
        "odoor.fields.Many2one",
        "odoor.fields.Image",
        "odoor.api.ondelete",
        "odoor.fields.One2many",
        "odoor.fields.Char",
        "odoor.api.constrains",
        "odoor.api.depends",
        "odoor.fields.Integer"
    ],
    "point_of_sale.models.pos_category.PosCategory._check_category_recursion": [
        "odoor.exceptions.ValidationError",
        "odoor.models.Model._check_recursion",
        "odoor._"
    ],
    "point_of_sale.models.pos_category.PosCategory.name_get": [
        "point_of_sale.models.pos_category.PosCategory.name_get.get_names",
        "<built-in>.reversed"
    ],
    "point_of_sale.models.pos_category.PosCategory.name_get.get_names": [
        "odoor.models.Model.search_count",
        "odoor.exceptions.UserError",
        "odoor._"
    ],
    "point_of_sale.models.pos_category.PosCategory._unlink_except_session_open": [
        "odoor._"
    ],
    "point_of_sale.models.pos_category.PosCategory._compute_has_image": [
        "<built-in>.bool"
    ]
}

```

```

Source Code pos_category.py
class PosCategory(odoo.models.Model):
    _name = "pos.category"
    _description = "Point of Sale Category"
    _order = "sequence, name"

    _api.constrains('parent_id')
    def _check_category_recursion(self):
        if not self._check_recursion():
            raise ValidationError(_('Error ! You cannot create recursive categories.'))

    name = fields.Char(string='Category Name', required=True, translate=True)
    parent_id = fields.Many2one('pos.category', string='Parent Category', index=True)
    child_id = fields.One2many('pos.category', 'parent_id', string='Children Categories')
    sequence = fields.Integer(help='Give the sequence order when displaying a list of product categories.')
    image_128 = fields.Image("Image", max_width=128, max_height=128)

    # During loading of data, the image is not loaded so we expose a lighter
    # field to determine whether a pos.category has an image or not.
    has_image = fields.Boolean(compute='_compute_has_image')

    def name_get(self):
        res = []
        for cat:
            res.append(cat.name)
            cat = cat.parent_id
        return res
    return [(cat.id, " / ".join(reversed(get_names(cat)))) for cat in self]

    @api.ondelete(at_uninstall=False)
    def _unlink_except_session_open(self):
        if self.search_count([('id', 'in', self.ids)]):
            if self.env['pos.session'].sudo().search_count([('state', '=', 'closed')]):
                raise UserError(_('You cannot delete a point of sale category while a session is still opened.'))

    _depends(['has_image'])
    def _compute_has_image(self):
        for category in self:
            category.has_image = bool(category.image_128)

```

Gambar 4.2 Perbandingan hasil PyCG dengan Kode Program Aslinya

4.2.3 Ekstraksi Dependency Model

Pada proses ini dimulai dari menentukan path dimana kode program disimpan, kemudian karena terdapat 2 lokasi addons maka dibuat symbolic link dari addons ke odoo/addons agar inspect bisa mencari addons. Pencarian dimulai mencari module Python yang ada di file *.py, hasilnya module dibuka dengan inspect dan dicari class 'odoo.models.MetaModel'. Dari class tersebut akan memiliki atribut _name, _inherit/_inherits, attribute_rel, dan comodel_name. Atribut ini dapat diekstraksi untuk membuat graph keterhubungan / ketergantungan antara module.

Tabel 4.1 Daftar Metode untuk melakukan ekstraksi *Dependency Module*

Metode / Fungsi	Parameter	Keterangan
walkTroughFolder	folderSC,filterExt	Fungsi rekursif untuk mencari daftar file di dalam folder hingga sub-folder dengan extensi yang tertentu
pathToModule	file, removeFile=True	Mengubah path slash menjadi dot python dan menghapus nama file jika diperlukan
scanModuleWithInspect	modulePath	Membuat daftar class dari nama module dan hasilnya berisi nama model, inherit model, dan relasi attribute,
searchDependency	module	Membuat graph dari daftar class yang dihasilkan scanModuleWithInspect

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```
listModule = [ f'odoo.{s}' for s in pathToModule(walkTroughFolder("odoo/addons")) ]
listModuleName = {}
moduleNameMapping = {}

def searchDependency(module):
    res = scanModuleWithInspect(module)
    for k , m in res.items():
        if len(m) == 0:
            continue
        for _ , c in m.items():
            if c['name'] not in listModuleName:
                moduleNameMapping[c['name']] = f'{module}.{k}'
                listModuleName[c['name']] = []
            if 'inherit' in c and len(c['inherit']) > 0:
                if isinstance(c['inherit'],list):
                    listModuleName[c['name']] += c['inherit']
                elif isinstance(c['inherit'],str):
                    listModuleName[c['name']].append(c['inherit'])
                else:
                    print("Warn: Data Type Not Found: " , c['name'])
            if 'inherits' in c and len(c['inherits']) > 0:
                listModuleName[c['name']] += list(c['inherits'].keys())
            if 'attribute_rel' in c and len(c['attribute_rel']) > 0:
                listModuleName[c['name']] += list(c['attribute_rel'].values())

    for m in listModule:
        try:
            searchDependency(m)
        except Exception as e:
            print("Exception :", e , m)
```

Gambar 4.3 Implementasi Ekstraksi Model dengan inspect

Pada gambar 4.3, proses yang dilakukan yaitu membuat daftar nama module python yang dari path "odoo/addons" dengan bentuk nama module "odoo.nama_model". Terdapat 2 variabel yaitu listModuleName untuk menyimpan relasi sebuah model dengan model lainnya dan variabel moduleNameMapping untuk memetakan nama dari nama model menjadi nama module yang dikenal oleh Python seperti nama model 'sale.order' dapat dipetakan menjadi 'odoo.addons.sale.models.sale_order'.

Proses pencarian dilakukan satu per satu dari daftar module yang ada di variabel listModule. Fungsi searchDependency berfungsi untuk mengubah daftar class dan attributnya yang dihasilkan oleh scanModuleWithInspect untuk bisa dibuat menjadi keterhubungan karena dalam module bisa memiliki banyak model / class dan bisa dipetakan dari nama model menjadi nama module Python.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```
def scanModuleWithInspect(modulePath):
    runcommand = importlib.import_module(modulePath)
    listClass = {}
    for name, obj in inspect.getmembers(runcommand):
        if inspect.ismodule(obj) == False:
            continue
        member = inspect.getmembers(obj)
        tmpClass = {}

        for item in member:
            # item => [ 0 => 'name', 1 => 'value member' ]
            if inspect.isclass(item[1]) == False:
                continue
            if hasattr(item[1], '__class__') and str(item[1].__class__) != "<class 'odoo.models.MetaModel'>":
                continue
            if hasattr(item[1], '_name'):
                tmpClass[item[0]] = { 'name' : item[1]._name}
            if hasattr(item[1], '_inherit'):
                # Case Class Model tidak punya nama
                if tmpClass[item[0]]['name'] in item[1]()._inherit :
                    tmpClass[item[0]]['name'] = item[1]._module_
                    tmpClass[item[0]]['inherit'] = item[1]()._inherit
            if hasattr(item[1], '_inherits'):
                tmpClass[item[0]]['inherits'] = item[1]()._inherits
            classMembers = inspect.getmembers(item[1])

            tmpClass[item[0]]['attribute_rel'] = {}
            for attrClass in classMembers:
                if hasattr(attrClass[1], 'comodel_name'):
                    if attrClass[1].comodel_name != None:
                        tmpClass[item[0]]['attribute_rel'][attrClass[0]] = attrClass[1].comodel_name

            if len(tmpClass) > 0:
                listClass[name] = tmpClass
    return listClass
```

Gambar 4.4 Implementasi Ekstraksi Model dengan inspect Lanjutan

Fungsi `scanModuleWithInspect` yang dapat dilihat pada gambar 4.4 memiliki masukan berupa string untuk alamat module dengan bentuk dot path seperti "odoo.addons.product". Untuk mengetahui isi dari sebuah kode program python harus dilakukan proses pembacaan kode program, pembacaan ini dimulai dari memasukan alamat direktori tempat dimana kode program berada, melalui `import_module`. Import module membantu untuk mengambil sebuah module secara dinamik, hasilnya berupa objek Python yang berisi module dari alamat yang dimasukan. Pada bahasa pemrograman Python hampir semua hal adalah sebuah objek, sehingga dari objek bisa ditentukan tipe data apa yang dimiliki objek tersebut.

Module bisa memiliki banyak module -> sub-modul(file) -> class / def / var /object. Untuk mengetahui apakah sebuah module memilikinya maka digunakan `inspect.getmember`. Hasil keluaran `inspect.getmember` berupa array dimana ['nama_objek', isi_object].

Setiap member bisa berisi bermacam-macam data dan berupa metadata. Apabila module memiliki sub-module maka dicari kembali class-nya dari sub-module tersebut. Fungsi ini hanya berfokus mencari class pada sebuah

BAB 4 IMPLEMENTASI DAN PENGUJIAN

module/sub dengan tipe odoo.models.MetaModel, jika terdapat class yang memiliki tipe MetaModel. Maka dilakukan pengecekan apakah object tersebut memiliki attribute _name, _inherit, _inherits, attribute_rel, dan comodel_name.

Masing-masing atribut ini seperti yang sudah dijelaskan pada tinjauan objek memiliki makna yang penting untuk menentukan relasi antar class. Attribute kemudian simpan di variabel dictionary(key,value) dengan key adalah nama module dan relasi serta namanya, isi datanya berupa string yang merepresentasikan sebuah nama model di Odoo seperti 'res.users'.

```
Hasil Ekstraksi Model dengan inspect
```

```
Source Code
```

```
class Digest(models.Model):
    _inherit = 'digest.digest'
    kpi_pos_total = fields.Boolean('POS Sales')
    kpi_pos_total_value = fields.Monetary(compute='_compute_kpi_pos_total_value')

class Bill(models.Model):
    _name = "pos.bill"
    _order = "value"
    _description = "Coins/Bills"

    name = fields.Char("Name")
    value = fields.Float("Coin/Bill Value", required=True, digits=0)
    pos_config_ids = fields.Many2many("pos.config", string="Point of Sales")

class PosCategory(models.Model):
    _name = "pos.category"
    _description = "Point of Sale Category"
    _order = "sequence, name"

    @api.constrains('parent_id')
    def _check_category_recursion(self):
        if not self._check_recursion():
            raise ValidationError(_('Error ! You cannot create recursive categories.'))

    name = fields.Char(string='Category Name', required=True, translate=True)
    parent_id = fields.Many2one('pos.category', string='Parent Category', index=True)
    child_id = fields.One2many('pos.category', 'parent_id', string='Children Categories')
    sequence = fields.Integer(help="Gives the sequence order when displaying a list of product categories.")
    image_128 = fields.Image("Image", max_width=128, max_height=128)

    # During loading of data, the image is not loaded so we expose a lighter
    # field to determine whether a pos.category has an image or not.
    has_image = fields.Boolean(compute='_compute_has_image')
```

Gambar 4.5 Contoh Informasi yang diekstraksi dari Model

Informasi seperti di gambar 4.5 harus diubah oleh fungsi searchDependency dan hasilnya disimpan pada variabel moduleName dan moduleNameMapping. Hasil dari pembuatan dependency model ini yaitu berupa graph yang disimpan di moduleName contohnya seperti pada gambar 4.6. Graph ini baru dikelompokan berdasarkan nama model yang diketahui Odoo. Name Model ini akan dipetakan menjadi nama module. Sehingga bisa dilakukan penggabungan antara hasil PyCG dan inspect.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```
= Isi variabel listModuleName untuk menampilkan hubungan antar model
{
    "odoo.addons.auth_signup.models.res_users": ["res.users"],
    "coupon.share": ["loyalty.card", "res.users", "loyalty.program",
    |   |   |   |   |   |   "website", "res.users"],
    "pos.category": ["pos.category", "res.users", "pos.category", "res.users"]
}

= Isi variabel moduleNameMapping untuk memetakan nama module dengan nama model
{
    "odoo.addons.auth_signup.models.res_users" : "odoo.addons.auth_signup.models.res_users",
    "coupon.share": "odoo.addons.website_sale_loyalty.wizard.sale_coupon_share",
    "pos.category": "odoo.addons.point_of_sale.models.pos_category"
}
```

Gambar 4.6 Hasil ekstraksi dari Model menggunakan inspect

4.2.4 Pengabungan Hasil Ekstraksi

Hubungan ketergantungan yang sudah dihasilkan dari PyCG dan inspect, diproses dan dijadikan satu kesatuan sebelum dilakukan proses clustering. Hal yang diproses seperti menghapus node yang tidak terpakai atau diluar dari Odoo seperti keterhubungan dengan library ke-3.

Tabel 4.2 Daftar Metode untuk pengabungan hasil ekstraksi

Metode / Fungsi	Parameter	Keterangan
loadJSON	path	Membuka file json dari suatu path
getListRootPackage	path	Membuat daftar module/package dari folder path bila folder tersebut memiliki Python.
addPrefixFolder	cg,root,listPackage	Menambahkan awal root dari call graph yang dihasilkan dari PyCG(JSON) dan disatukan berdasarkan nama modulennya
mergeCGOdooWithAddons	cgOdoo,cgAddons	Mengabungkan call graph dari odoo dan addons yang sudah memiliki prefix
filterCGNode	cgSource	Menghapus node atau hubungan yang tidak di dalam module addons atau base pada graph
updateCGwInspect	listModuleName, moduleNameMapping, callGraph	Mengabungkan hasil relasi yang ditemukan di library inspect ke call graph

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```
cgAddonsSource =loadJSON(addonsJSON)
listPackageAddons = getListRootPackage(f'{currentPath}/odoo/addons')
callGraphRaw = addPrefixFolder(cgAddonsSource, "addons", listPackageAddons)
len(callGraphRaw)
✓ 0.5s
27521
```

Gambar 4.7 Proses pembacaan file JSON dan penambahan prefix

Proses penggabungan dimulai dengan mengimport file JSON yang dihasilkan oleh PyCG. File JSON tersebut diubah menjadi bentuk call graph di dictionary(key,value). Nama node yang dihasilkan PyCG tidak lengkap karena dilakukan scan melalui odoo/addons maka tidak ada prefix atau parentnya yang lengkap, untuk itu dilakukan penambahan nama module didepan nama node "addons" sehingga yang awalannya langsung merupakan nama package / addons seperti "product.models" maka menjadi "addons.product.models". Tujuan dilakukan ini agar node di graph dari PyCG memiliki tingkatan yang sama dengan nama package odoo. Penambahan ini menggunakan fungsi addPrefixFolder dan getListRootPackage.

```
def filterCGNode(cgSource):
    def checkIsBase(key):
        parentKey = key.split('.')
        if parentKey[0] == 'odoo' and parentKey[1] == 'addons':
            # Ganti addons diluar base
            if len(parentKey) >= 3 and parentKey[2] != 'base':
                key = '.'.join(key.split('.')[1:])
        return key

    callGraphFiltered = {}
    listRootFolder = [ 'odoo', 'addons' ]
    edgeGraph = []
    outsideCall = set()
    for key, value in cgSource.items():
        rootSource = key.split('.')[0]
        if rootSource not in listRootFolder:
            outsideCall.add(rootSource)
            continue
        childFilter = {}
        for v in value:
            childSource = v.split('.')[0]
            if childSource not in listRootFolder:
                outsideCall.add(childSource)
                continue
            v = checkIsBase(v)
            childFilter[v] = 1
        if len(childFilter) == 0:
            edgeGraph.append(key)
            continue
        key = checkIsBase(key)
        for c in childFilter:
            if c not in callGraphFiltered:
                callGraphFiltered[c] = {}
            callGraphFiltered[key] = childFilter
    print(f'Total Node Awal: {len(cgSource)} ')
    print(f'Total Node: {len(callGraphFiltered)} ')
    print(f'Total Leaf Node: {len(edgeGraph)} <{edgeGraph[:3]}>')
    print(f'Total Call Diluar package Odoo : {len(outsideCall)} <{list(outsideCall)[:3]}>')
    return callGraphFiltered
```

Gambar 4.8 Isi fungsi filterCGNode

Isi dari graph dilakukan pembuangan pada node yang tidak diinginkan untuk dikelompokan, karena tugas besar ini berfokus pada mengelompokan pada bagian module addons / proses bisnis maka setiap relasi ke package diluar odoo dan addons akan dihilangkan dari graph. Proses ini dilakukan oleh fungsi filterCGNode, fungsi tersebut membuat graph baru dari graph yang memiliki informasi call graph original dan menghapus node yang tidak memiliki hubungan apapun. Fungsi filterCGNode juga memberikan nilai setiap hubungan call sebesar 1, nilai ini akan digunakan sebagai weight atau jumlah call ke suatu node. Call Graph yang dihasilkan dari PyCG tidak memberikan jumlah call sehingga semua call diasumsikan memiliki nilai panggilan sebesar 1. Proses pengubahan dan penyerderhanaan graph dilakukan oleh fungsi filterCGNode.

```
callGraphFiltered = filterCGNode(callGraphRaw)
✓ 0.1s
Total Node Awal: 27521
Total Node: 15953
Total Leaf Node: 14287 <['addons.base', 'odoo.api.Environment', 'addons.base.wizard']>
Total Call Diluar package Odoo : 113 <['PyKCS11', 'imaplib', 'tempfile']>
```

Gambar 4.9 Hasil akhir node dari yang sudah digabungkan dan dibersihkan

Pada gambar 4.9 dapat terlihat jumlah awal node yang awalnya sebesar 27.521 setelah difilter menjadi sebesar 15.953. Node leaf yaitu fungsi yang ditemukan tidak memanggil node lain, untuk itu node tersebut dihilangkan. Proses selanjutnya yaitu mengabungkan call graph dengan hasil inspect, sebelumnya hasil inspect disimpan pada variabel listModuleName dan moduleNameMapping.

```
def updateCGwInspect(callGraphFiltered,listModuleName,moduleNameMapping):
    def levelingPathAddons(path):
        pathSplit = path.split(".")
        if pathSplit[0] == "odoo":
            return ".".join(pathSplit[1:])
        return path
    for r in listModuleName:
        if r not in moduleNameMapping:
            print("Path not Found " , r)
            continue
        if moduleNameMapping[r] not in callGraphFiltered:
            moduleName = levelingPathAddons(moduleNameMapping[r])
            callGraphFiltered[moduleName] = {}
        for c in listModuleName[r]:
            moduleName = levelingPathAddons(moduleNameMapping[r])
            if c not in moduleNameMapping:
                print("Path Call not Found " , c)
                continue
            tmpC = levelingPathAddons(moduleNameMapping[c])
            if tmpC not in callGraphFiltered[moduleName]:
                # print("new key: " , moduleNameMapping[r] , tmpC)
                callGraphFiltered[moduleName][tmpC] = 0
            callGraphFiltered[moduleName][tmpC] += 1
    return callGraphFiltered
```

Gambar 4.10 Isi fungsi updateCGwInspect

Pada inspect dapat diketahui jumlah berapa hubungan antara sebuah module, seperti misalkan ada model yang memiliki 2 attribute yang disebut 'write_uid' untuk menandakan siapa yang mengupdate data dengan relasi res.users dan 'create_uid' untuk menandakan siapa yang membuat data dengan relasi res.users. Selain itu nama model dari inspect akan disesuaikan dengan moduleNameMapping agar pencarian model 'pos.category' dapat menjadi 'odoo.addons.point_of_sale.models.pos_category'. Proses penggabungan dengan inspect dilakukan oleh fungsi updateCGwInspect.



Gambar 4.11 Ilustrasi Hasil Gabungan Graph

4.2.5 Optimisasi Hasil Ekstraksi

Hasil graph yang sudah digabungkan dari hasil ekstraksi harus diproses kembali agar pengelompokan yang dilakukan oleh Hierarchical Clustering terarah. Pada Tugas akhir ini pengelompokan dilakukan berdasarkan proses bisnisnya, proses bisnis tersebut berada pada module yang dinamakan 'addons'. Setiap panggilan kepada suatu addons / node harus diagregasikan untuk dijadikan *weight* yang berkisar 0 (tidak ada relasi) hingga 1 (memiliki relasi yang kuat)

Tabel 4.3 Daftar Metode untuk optimisasi hasil ekstraksi

Metode / Fungsi	Parameter	Keterangan
initKeyCG	-	Membuat daftar key untuk graph addons kecuali module test dan l10n
checkNameKey	k	Mengecek apakah name pada sebuah nama module memiliki module yang tidak perlu dikelompokan seperti 'tests', 'testing', 'test', 'l10n'

updateCGWeight	callGraphFiltered, callGraphWeight	Membuat call graph yang memiliki nilai berbobot (weighted graph) berdasarkan jumlah panggilan(call) dengan cara menjumlahkan call yang dipanggil dari module
cleanUpCG	callGraph	Menghilangkan call yang rendundan dan menyimpan informasi panggilan internal di variabel yang terpisah
removeNotConnectedNode	callGraph	Menghapus node parent yang tidak terhubung ke node lain dan memastikan node graph yang dibentuk lengkap
createAdjacentMatrix	graphSource	Membuat adjacency matrix dari graph

Untuk mengetahui jumlah panggilan keluar atau panggilan didalam module maka graph yang sudah disatukan sebelumnya diubah menjadi bentuk tree. Dengan bentuk tree ini bisa dilakukan perhitungan jumlah panggilan dan relasinya. Tree dibuat dengan class NodeCG,seperti pada gambar 4.12. Proses perubahan graph pada gambar 4.13 dimulai dari membentuk root dengan nama "odoo" kemudian mengubah semua relasi setiap node relasi baru dengan fungsi setRelation pada class NodeCG. Apabila nama dari relasi memiliki module yang berkaitan dengan testing atau lokalisasi seperti 'addons.product.tests' maka relasi 'addons.product.tests' tidak ditambahakan pada odooTree.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```

class NodeCG:
    root = ""
    def __init__(self, name, parent):
        self.name = name
        self.parent = parent
        self.outsideCall = {}
        self.insideCall = {}
        self.relation = {}

    def getRelation(self, name=""):
        pass

    def setRelation(self, name, targetCall):
        pass

    def shiftTargetName(self, strName):
        # 'addons.product.models' => 'addons', 'product.models'
        cutStr = strName.split(".")
        if len(cutStr) > 1:
            return cutStr[0] + ".".join(cutStr[1:])
        return cutStr[0] + ""

    def isValidName(self, name):
        if not isinstance(name, str):
            raise Exception("Nama Node harus str")
        if name.strip() == "":
            raise Exception("Nama Kosong")

    def countCall(self, nodeName):
        r, cName = self.shiftTargetName(nodeName)
        if r in self.relation:
            return self.relation[r].countCall(cName)
        elif cName == "" and r == "":
            oc = self.outsideCall.copy()
            currentAsParent = self.parent.copy()
            currentAsParent.append(self.name)
            currParentStr = '.'.join(currentAsParent)
            oc[currParentStr] = self.insideCall
            return oc
        else:
            return {}

    def getRelation(self, name=""):
        newName, childName = self.shiftTargetName(name)
        if newName in self.relation:
            relCC = self.relation[newName].getRelation(childName)
            return {self.name : relCC}
        listRelation = list(self.relation.keys())
        return {self.name : { 'outsideCall' : self.outsideCall,
                            'relation' : listRelation, 'insideCall' : self.insideCall} }

    def setRelation(self, name, targetCall):
        self.isValidName(name)
        newName, childName = self.shiftTargetName(name)

        currentAsParent = self.parent.copy()
        if self.name == self.root:
            currentAsParent = []
        else:
            currentAsParent.append(self.name)

        if newName not in self.relation:
            # Tambah node dan beri tahu dia memberi tahu node parentnya
            self.relation[newName] = NodeCG(newName, currentAsParent)

        # Kondisi bila bukan paling akhir masih ada child
        currParentStr = '.'.join(currentAsParent)
        for t in targetCall:
            cutStr = t.split(".")
            targetParent = ".".join(cutStr[0:len(self.parent)+1])
            if targetParent == currParentStr:
                # Menambahkan kepada insideCall bila sesuai dengan parent sekarang
                # 'addons.product.models' => 'addons.product.models.B'
                if t not in self.insideCall:
                    self.insideCall[t] = 0
                    self.insideCall[t] += targetCall[t]
            elif self.name != self.root:
                # Menambahkan kepada outsideCall bila tidak sesuai dengan parent sekarang
                # 'addons.product.models' => 'addons.note.models'
                if t not in self.outsideCall:
                    self.outsideCall[t] = 0
                    self.outsideCall[t] += targetCall[t]

        # Kondisi bila sudah paling akhir
        if childName == "" and newName == name:
            self.relation[newName].outsideCall = targetCall
            return True
        return self.relation[newName].setRelation(childName, targetCall)

```

Gambar 4.12 class NodeCG untuk menghitung weight

```

odoоТree = NodeCG("odoo", [])
odoоТree.root = "odoo"
blackListNodeName = ['tests', 'testing', 'test', 'l10n' ]
def checkNameKey(k):
    def checkIfNameContainString(v, t):
        for n in blackListNodeName:
            if checkIfNameContainString(k, n):
                cutStr = k.split(".")
                if cutStr[0] != "addons":
                    return False
                return True

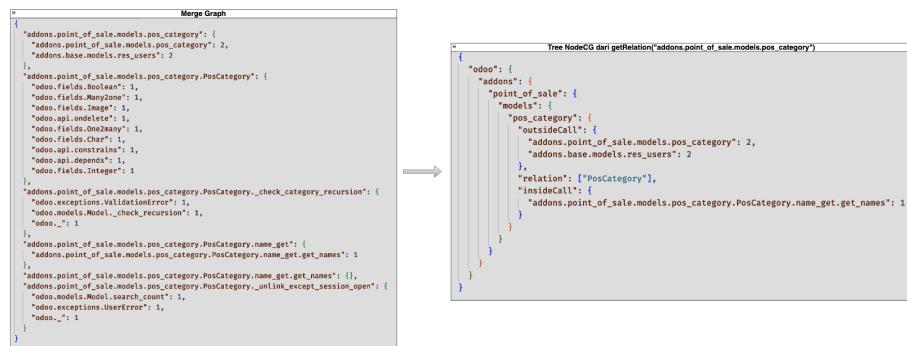
        for k, v in callGraphFiltered.items():
            if not checkNameKey(k):
                continue
            tmpV = {}
            for i in v:
                if not checkNameKey(i):
                    continue
                tmpV[i] = v[i]
            if len(tmpV) == 0:
                continue
            odoоТree.setRelation(k, tmpV)

```

Gambar 4.13 Proses perubahan graph di dictionary menjadi class NodeCG

Penambahan relasi node di tree yang dilakukan oleh fungsi setRelation yaitu berupa key string seperti 'addons.point_of_sale.model' dan isi relasi node. Fungsi setRelation membuat relasi baru didalam tree bila module tersebut belum ada, jika sudah ada relasi sebelumnya maka setRelation akan menggunakan relasi itu dan

memperbarui isi relasi. Fungsi ini berhenti ketika sudah tidak ada child module seperti contohnya 'a.i' make child dari module 'a' adalah 'i' dan module 'i' tidak memiliki memiliki child. Hasil perubahan dari graph yang direpresentasikan dalam bentuk dictionary ke bentuk tree di NodeCG dapat dilihat pada gambar 4.14



Gambar 4.14 Ilustrasi Path dot menjadi bentuk Tree NodeCG

Proses selanjutnya yaitu membuat graph baru yang hanya berisi module addons, proses ini dilakukan oleh fungsi initKeyCG. Fungsi akan membaca direktori dari path odoo/addons dan hanya menambahkan bila module tersebut adalah package (folder). Kemudian graph baru diperbarui isi relasinya dengan fungsi updateCGWeight. Fungsi updateCGWeight menggunakan odooTree dan call graph baru. Kemudian dilakukan pengecekan apakah graph yang dibuat lengkap dan dibuat menjadi Adjacency Matrix. Dari gambar 4.15 dapat dilihat terdapat 335 objek yang akan dilakukan proses hierarchical clustering.

```

callGraphWeight = initKeyCG()
callGraphWeight = updateCGWeight(odooTree,callGraphWeight)
callGraphWeightKey = list(callGraphWeight.keys())

cgFilterInternal, selfCG = cleanUpCG(callGraphWeight)
callGraphFinal = removeNotConnectedNode(cgFilterInternal)
adjMatrix, dictLabel = createdAdjacentMatrix(callGraphFinal)
listLabel = list(dictLabel.keys())
print("Size Label: " ,len(dictLabel))

✓ 0.9s
Info, Relasi dari Target dilewati addons._path_
Info, Relasi dari Target dilewati addons._path_
Info, Relasi dari Target dilewati addons.decimal_precision
Size Label: 335

```

Gambar 4.15 Proses Optimisasi Graph dan Pembuatan Adjacency Matrik

Hasil dari proses ini dapat dilihat pada gambar 4.16, dapat dilihat pengabungan panggilan yang diagregasikan berdasarkan module addons. Proses ini juga menghitung berapa relasi dalam dirinya sendiri, tujuan perhitungan ini untuk mengetahui berapa besar kohesi didalam module. Jumlah relasi kepada diri sendiri tidak ditambahkan pada Adjacency matrix, dan matriks yang dibuat juga dilakukan column-wise normalization seperti yang dijelaskan pada bagian Urutan Proses Global.

BAB 4 IMPLEMENTASI DAN PENGUJIAN



Gambar 4.16 Ilustrasi Hasil Graph yang dapat dilakukan Proses Pengelompokan

4.2.6 Hierarchical Clustering

Proses ini dilakukan untuk menemukan pengelompokan dari adjacency matrix. Matrix adjacency(Matrix keterhubungan) harus dihitung untuk mendapatkan nilai kedekatannya, perhitungan tersebut menghasilkan Distance Matrix(Matrik kedekatan). Pada tugas akhir ini menggunakan nilai kedekatan berdasarkan nilai Jaccard dan Struktural Similarity.

```

def simJaccard(row, col):
    row = np.asarray(row,bool)
    col = np.asarray(col,bool)

    if row.shape != col.shape:
        raise ValueError("Not same size")

    intersection = np.logical_and(row, col)
    union = np.logical_or(row, col)
    return (intersection.sum() / float(union.sum()))

def calculateDistanceMatrix(adjMatrix):
    data = np.array(adjMatrix)
    distanceMatrix = [[0.00 for i in range(data.shape[1])] for j in range(data.shape[0])]

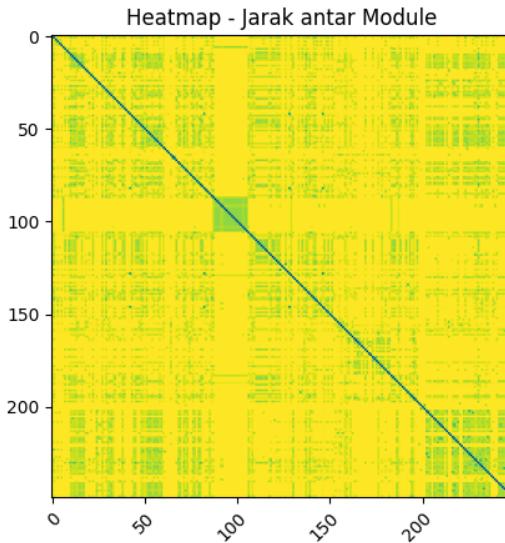
    np.fill_diagonal(data,1)
    for row in range(0,len(data)):
        for col in range(0,row):
            # Semakin jauh mendekati nilai 1, mengambil satu bagian untuk masing masing col
            distanceStructural = round(1- simStr(data[row, :], data[col, :]), row, col,sum(data[:, row]), sum(data[:, col])),2
            distanceJaccard = round(1- simJaccard(data[row, :], data[col, :]),2)
            distanceMatrix[row][col] = ((distanceStructural + distanceJaccard) / 2)
            distanceMatrix[col][row] = distanceMatrix[row][col]
    return distanceMatrix

def simStr(ci, cj, i, j, callsinCi , callsinCj):
    res = 0
    if callsinCi > 0 and callsinCj > 0:
        res = 0.5 * ( ci[j]/callsinCj + cj[i]/callsinCi )
    elif callsinCi == 0 and callsinCj > 0:
        res = ci[j]/callsinCj
    elif callsinCi > 0 and callsinCj == 0:
        res = cj[i]/callsinCi
    return res

```

Gambar 4.17 Implementasi Perhitungan Distance Matrix dengan Jaccard dan Struktural Similarity

Matriks kedekatan bisa ditampilkan dalam bentuk heatmap. Heatmap dapat memberikan bagaimana intensitas kedekatan antar module addons Odoo. Pada gambar 4.18 dapat dilihat semakin kuning atau cerah maka semakin tinggi intensitas hubungannya sedangkan semakin hijau atau redup maka intensitas hubungannya rendah. heatmap ini dihasilkan dari matrix segitiga sehingga hasil dari heatmap sisi kiri sama dengan heatmap sisi kanan.



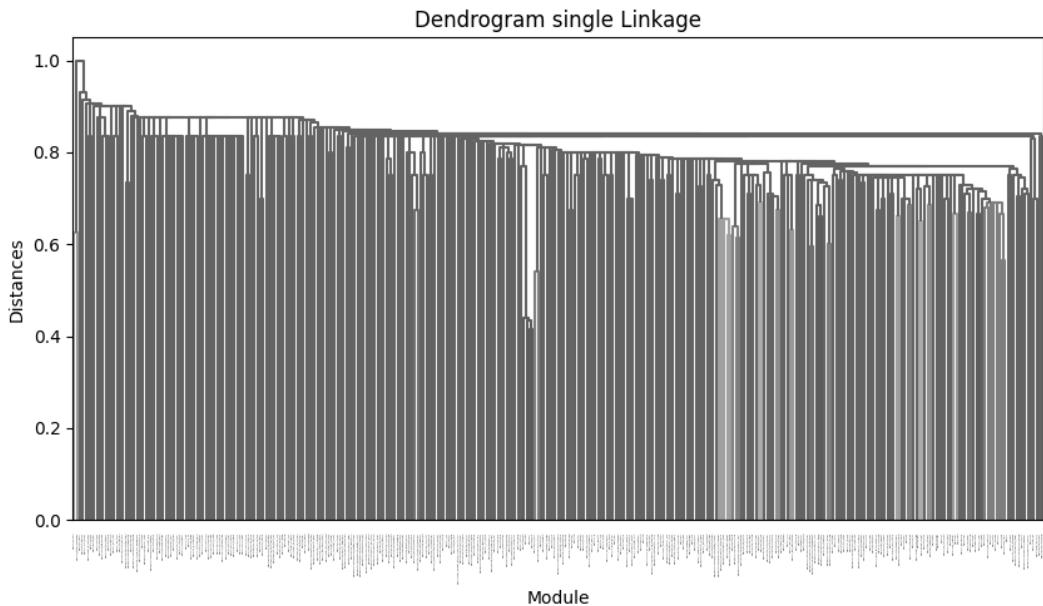
Gambar 4.18 Heatmap yang menunjukkan intensitas hubungan antar module/node

Tabel 4.4 Daftar Metode untuk proses Clustering

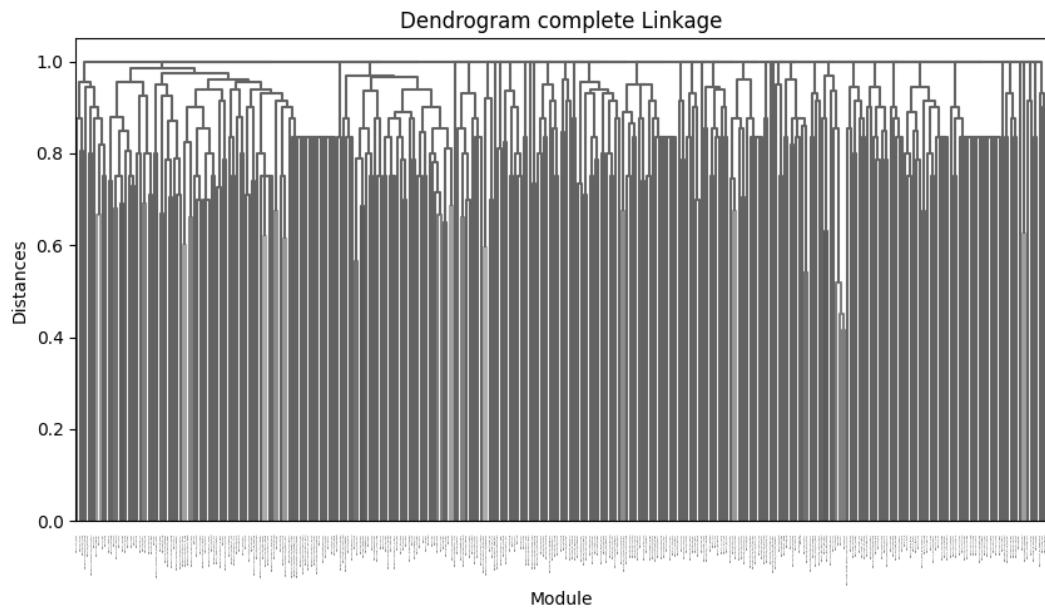
Metode / Fungsi	Parameter	Keterangan
simStr	ci, cj, i, j, callsinCi , callsinC	Implementasi dari rumus 2.2
simJaccard	im1, im2	Implementasi dari rumus 2.1
calculateDistanceMatrix	adjMatrix	Menghitung distance matrix dengan masukan adjacency matrix
visualizeHeatmap	matrix, title	Membuat visualisasi heatmap
calculateCluster	y, _method	Melakukan proses clustering dengan linkage function tertentu (_method) dan distance matrix (y)
visualizeDendogram	z, _method	Menampilkan dendogram dari hierarchical clustering

Distance Matrix dimasukan kedalam fungsi calculateCluster dengan metode ada 3 yaitu average, min, max. Hasil cluster dari hierarchical clustering bisa divisualisasi dalam bentuk dendogram. Hasil dendogram memiliki distance/jarak antar module dan label module. Dari gambar 4.19 module cenderung dikelompokan dengan jumlah yang kecil (1-2) dan ketika dihubungkan memiliki jarak yang jauh. Pada pendekatan 4.20 module dikelompokan dan memiliki ukuran module yang besar untuk setiap partisi/service. Sedangkan pendekatan 4.21 memiliki bentuk campuran antara ukuran module yang besar dan

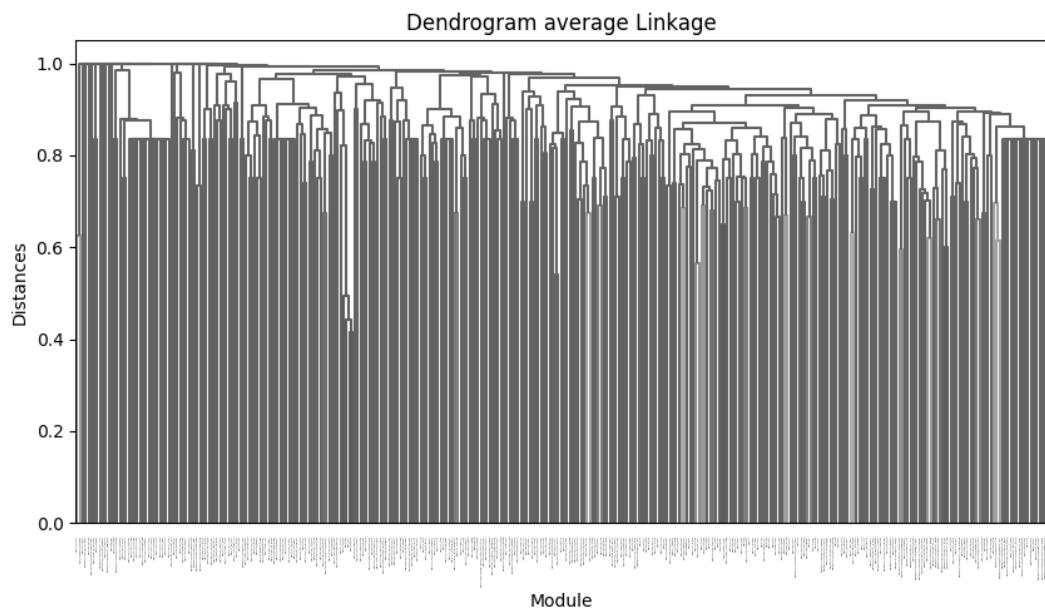
module yang berukuran kecil.



Gambar 4.19 Dendrogram Single Linkage



Gambar 4.20 Dendogram Complete Linkage



Gambar 4.21 Dendrogram Average Linkage

4.3 Evaluasi dan Pengujian

Pada bagian ini, dibahas mengenai evaluasi dan pengujian tentang bagaimana hasil dekomposisi aplikasi monolitik menjadi microservice dengan menggunakan proses Hierarchical Clustering.

4.3.1 Pemilihan Partisi

Banyaknya partisi dan jumlah node, membuat proses pemilihan partisi atau kelompok service terbaik harus dilakukan secara otomatis. Hasil pemilihan dapat menentukan microservice yang memiliki nilai cohesion yang tinggi dan nilai coupling yang rendah.

4.3.1.1 Implementasi Pemilihan

Pemilihan dimulai dengan menggunakan linkage matrix(Z / hasil dari Hierarchical Clustering) untuk setiap metode linkage yang berbeda. Untuk mengetahui jumlah partisi yang tepat dilakukan pengukuran dari nilai coupling dan nilai kohesinya yang dihasilkan dari pengelompokan tersebut. Terdapat fungsi yang akan menghitung dari 1 partisi/service hingga jumlah maximum partisi.

Tabel 4.5 Daftar Metode untuk pemilihan Partisi

Metode / Fungsi	Parameter	Keterangan
mergeClass	ct	Mencari keterhubungan antar partisi dan di dalam partisi berdasarkan tree yang diberikan dari cluster
cutTreeToCluster	ct	Membuat cluster dari tree yang dihasilkan clustering
NbCalls	m, c1, c2	Menghitung jumlah call antar module (Rumus 2.5)
CoupP	m, c1,c2	Implementasi dari rumus 2.5
InterCoup	m	Implementasi dari rumus 2.4
InterCoh	rm	Implementasi dari rumus 2.6
FOne	m , rm	Implementasi dari rumus 2.3 dan perhitungan skor clustering melalui jumlah partisi, nilai coupling dan nilai cohesion
calculateFOne	z, _n_clusters	Menghitung hasil FOne dengan diberikan z dan jumlah cluster/partisi

analystCluster	z	Melakukan perhitungan calculateFOne dari 1 hingga jumlah maximum cluster/partisi
visualizeQualityCluster	qualityCluster, linkageType	Menampilkan plot perbandingan coupling dan cohesion dengan jumlah service

Proses perhitungan dimulai dari fungsi calculateFOne (Rumus 2.3), fungsi ini akan mengubah linkage matrix (Z), menjadi bentuk graph yang menggambarkan keterhubungan antar service dan module didalamnya. Hasil graph tersebut diubah menjadi matriks adjacency dimana isi matriks tersebut merupakan nilai keterhubungan antara service/partisi. Perhitungan Coupling (Rumus 2.4) dan Cohesion (Rumus 2.6) dilakukan dengan evaluasi *Structural and Behavioral Dependencies*. Hasil dari proses ini berupa *dictionary* yang memiliki isi nilai coupling, nilai cohesion dan ukuran service.

```

def CoupP(m, c1,c2):
    def NbCalls(m,c1,c2):
        return m[c2][c1]
    TotalNBCalls = 0
    for x in range(0,len(m)):
        if m[c1][x] > 0:
            TotalNBCalls += m[c1][x]
        if m[c2][x] > 0:
            TotalNBCalls += m[c2][x]
    if TotalNBCalls == 0:
        return 0
    return (NbCalls(m,c1,c2) + NbCalls(m,c2,c1))/TotalNBCalls
def InterCoup(m):
    sumCoup = 0
    NbPossiblePairs = len(m)
    for c1 in range(0,len(m)):
        for c2 in range(0,len(m)):
            tmp = CoupP(m,c1,c2)
            sumCoup += tmp
    return sumCoup/NbPossiblePairs
def FOne(m,rm):
    interCoup = InterCoup(m)
    interCoh = InterCoh(rm)
    return {
        "coupling": round(interCoup,6),
        "cohesion": round(interCoh,6),
        "clusterSize": len(m)
    }
def InterCoh(rm):
    NbDirectConnections = 0
    NbPossibleConnection = 0
    #Menghitung Panggilan Internal
    for _, p in rm.items():
        if len(p) == 0:
            continue
        for m , call in p.items():
            #Pertimbangkan Call dalam Addons/Modules
            if m in selfCG:
                NbDirectConnections += selfCG[m]
            for c, w in call.items():
                #External Call
                if c not in p:
                    #Menghitung Panggilan External
                    NbPossibleConnection += w
                continue
            NbDirectConnections += w
    NbPossibleConnection += NbDirectConnections
    return NbDirectConnections / NbPossibleConnection
def calculate_FOne(z, _n_clusters):
    ct = cut_tree(z, n_clusters=_n_clusters)
    mgCls, inCls = mergeClass(ct)
    adjMerge, _ = createAdjacentMatrix(mgCls)
    result = FOne(adjMerge,inCls)
    return result

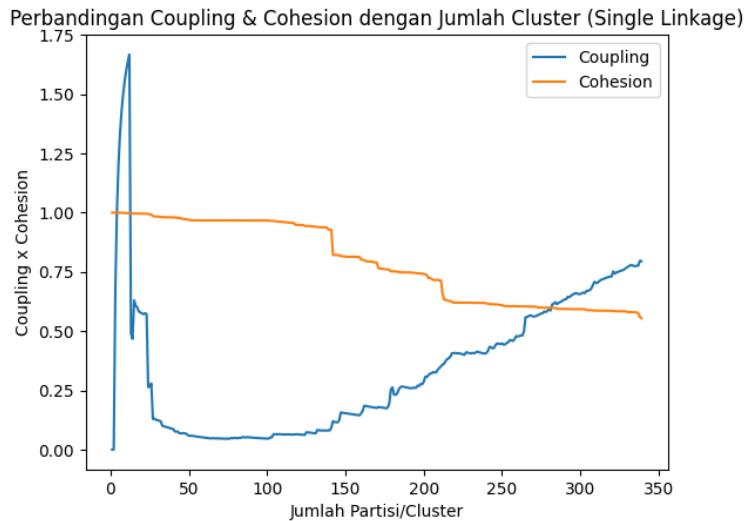
```

Gambar 4.22 Implementasi untuk melakukan menghitung FOne (coupling dan cohesion)

4.3.1.2 Evaluasi Pemilihan

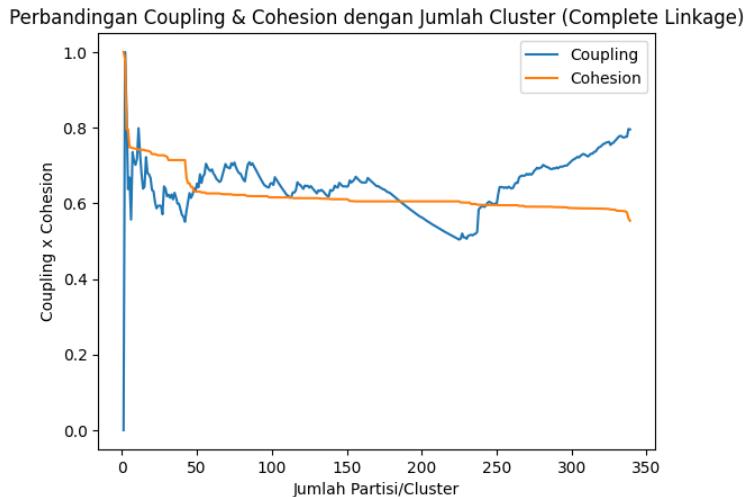
Hasil Implementasi dapat divisualisasikan dan dianalisis lebih lanjut agar dapat menentukan linkage dan jumlah service yang ideal.

Untuk itu dilakukan perbandingan bagaimana nilai coupling dan cohesion dari jumlah service/partisi yang dimulai dari 1 (monolit) hingga 335 (semua partisi menjadi service individu) dengan setiap linkage yang berbeda.



Gambar 4.23 Perbandingan Coupling & Cohesion menurut pendekatan Single Linkage

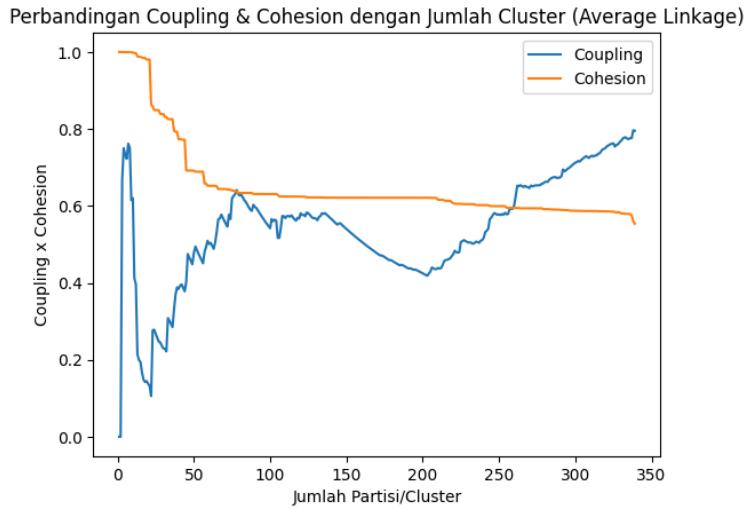
Pada grafik 4.23, hasil dekomposisi menggunakan linkage single memiliki hasil yang buruk untuk coupling tetapi masih memiliki cohesion yang bagus. Nilai coupling menurun drastis ketika jumlah service mencapai 25 service. Jumlah service yang memiliki nilai coupling rendah dan cohesion yang tinggi berkisar dari 25 service ke atas dan dibawah 150 sevice. Ketika jumlah service melebihi 150 service nilai coupling terus meningkat ketika dan nilai cohesion terus memburuk.



Gambar 4.24 Perbandingan Coupling & Cohesion menurut pendekatan Complete Linkage

Pada grafik 4.24, hasil dekomposisi menggunakan linkage complete memiliki coupling yang tinggi sejak jumlah service 1 ke atas hingga sekitar 175 service. Nilai Coupling membaik ketika jumlah service mencapai ± 200 hingga ± 240 , ketika jumlah service melebihi ± 240 nilai menjadi semakin memburuk coupling. Untuk nilai cohesion turun secara perlahan ketika jumlah

service mencapai ± 50 .



Gambar 4.25 Perbandingan Coupling & Cohesion menurut pendekatan Average Linkage

Pada grafik 4.25, hasil dekomposisi menggunakan Average linkage memiliki nilai coupling yang cukup tinggi ketika jumlah service sebesar ± 25 . Nilai coupling meningkat secara tajam ketika jumlah service diatas ± 2 dan mulai menurun saat jumlah service berkisar $\pm 100 - \pm 210$. Ketika jumlah service melebihi ± 210 nilai coupling semakin memburuk. Untuk nilai cohesion, memiliki nilai yang tinggi saat jumlah service berkisar $\pm 2 - \pm 25$, ketika jumlah service diatas ± 25 nilai cohesion menurun drastis hingga jumlah service ± 75 . Nilai cohesion tidak lagi meningkat dan turun perlahan ketika jumlah service bertambah banyak.

Nilai coupling dan cohesion yang dihasilkan bisa dilakukan analisis lebih lanjut seperti melihat nilai maksimum, nilai minimum, dan nilai rata-rata pada nilai coupling dan cohesion untuk setiap linkage.

Linkage	Coupling			Cohesion		
	MIN	AVG	MAX	MIN	AVG	MAX
Single	0.04	0.33	1.6	0.61	0.83	1
Complete	0.40	0.65	1.33	0.61	0.67	0.99
Average	0.14	0.51	0.83	0.61	0.7	1

Tabel 4.6 Tabel Nilai maximum, rata-rata, dan minimum untuk setiap linkage

Dari tabel 4.6 dilihat linkage complete memiliki coupling yang tinggi dan cohesion yang rendah dibandingkan linkage lainnya. Untuk linkage average memiliki nilai coupling dan nilai cohesion yang lebih baik bila dibandingkan

BAB 4 IMPLEMENTASI DAN PENGUJIAN

dengan linkage complete tetapi lebih buruk bila dibandingkan dengan linkage single. Nilai minimum dan maksimum dari cohesion memiliki nilai yang sama tetapi memiliki rata-rata yang berbeda.

Untuk mengetahui lebih detail bagaimana struktur service dikelompokan, maka untuk setiap linkage, jumlah module dari setiap partisi/service dicari nilai rentang, minimum, dan maksimum untuk setiap jumlah ukuran partisi tertentu (1, 2, 5, 10, 15, 25, 50 , 75, 90, 110, 130, 150, 175, 195, 200, 208 , 220, 232, 245, 275, 290, 305, 325, 335). Jumlah Module yang ditampilkan adalah jumlah modulennya diikuti dengan jumlah kemunculannya / Frekuensi yang dilambangkan dengan 'x' .

Tabel 4.7 Perbandingan Ukuran Service yang dihasilkan oleh Single Linkage

Ukuran Service	Jumlah Module	Coupling	Cohesion	Statistik Jumlah Module		
				MIN	MAX	RANGE
1	335(1x)	0.0	1.0	335	335	0
2	333(1x), 2(1x)	0.67	1.0	2	333	331
5	326(1x), 5(1x), 2(1x), 1(2x)	1.33	1.0	1	326	325
10	319(1x), 5(1x), 2(3x), 1(5x)	0.52	1.0	1	319	318
15	315(1x), 3(1x), 2(4x), 1(9x)	0.59	1.0	1	315	314
25	299(1x), 5(1x), 3(1x), 2(6x), 1(16x)	0.19	0.98	1	299	298
50	246(1x), 15(1x), 6(2x), 4(1x), 3(1x), 2(11x), 1(33x)	0.09	0.97	1	246	245
75	246(1x), 7(1x), 3(1x), 2(7x), 1(65x)	0.12	0.97	1	246	245
90	241(1x), 2(5x), 1(84x)	0.2	0.97	1	241	240
110	216(1x), 3(2x), 2(6x), 1(101x)	0.05	0.95	1	216	215
130	190(1x), 7(1x), 3(2x), 2(6x), 1(120x)	0.08	0.94	1	190	189
150	169(1x), 3(3x), 2(11x), 1(135x)	0.1	0.91	1	169	168

BAB 4 IMPLEMENTASI DAN PENGUJIAN

175	133(1x), 6(1x), 3(3x), 2(17x), 1(153x)	0.18	0.85	1	133	132
195	96(1x), 8(1x), 6(1x), 5(1x), 3(5x), 2(19x), 1(167x)	0.22	0.81	1	96	95
200	89(1x), 8(1x), 5(1x), 4(1x), 3(5x), 2(23x), 1(168x)	0.25	0.79	1	89	88
208	66(1x), 11(1x), 8(1x), 6(1x), 5(1x), 4(1x), 3(5x), 2(23x), 1(174x)	0.28	0.76	1	66	65
220	60(1x), 8(1x), 6(1x), 5(2x), 4(1x), 3(5x), 2(23x), 1(186x)	0.31	0.74	1	60	59
232	60(1x), 8(1x), 6(1x), 5(2x), 3(4x), 2(16x), 1(207x)	0.33	0.74	1	60	59
245	54(1x), 8(1x), 5(2x), 3(2x), 2(18x), 1(221x)	0.36	0.72	1	54	53
275	15(1x), 5(2x), 4(2x), 3(6x), 2(20x), 1(244x)	0.65	0.66	1	15	14
290	7(1x), 5(1x), 4(1x), 3(7x), 2(18x), 1(262x)	0.72	0.64	1	7	6
305	6(1x), 4(1x), 3(5x), 2(12x), 1(286x)	0.75	0.64	1	6	5
325	3(1x), 2(8x), 1(316x)	0.8	0.63	1	3	2

Pada tabel 4.7 dapat dilihat bahwa jumlah service yang dibuat oleh single linkage tidak merata dalam membagi modulennya, banyak service yang hanya memiliki jumlah modul kecil sekitar 1-3 module tetapi ada satu service yang sangat besar yang memiliki 100-200 module. Service yang memiliki module sangat banyak itu hilang ketika jumlah service / partisi mencapai ± 275 , namun nilai coupling dan cohesion ketika jumlah service sebesar ± 275 tidak bagus karena lebih tinggi nilai coupling dibandingkan nilai cohesionnya.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Tabel 4.8 Perbandingan Ukuran Service yang dihasilkan oleh Complete Linkage

Ukuran Service	Jumlah Module	Coupling	Cohesion	Statistik Jumlah Module		
				MIN	MAX	RANGE
1	335(1x)	1.0	1.0	335	335	0
2	324(1x), 11(1x)	1.33	1.0	11	324	313
5	212(1x), 88(1x), 17(1x), 11(1x), 7(1x)	0.68	0.96	7	212	205
10	132(1x), 88(1x), 31(1x), 17(2x), 16(1x), 11(1x), 8(2x), 7(1x)	0.41	0.93	7	132	125
15	107(1x), 88(1x), 31(1x), 17(2x), 16(1x), 11(1x), 10(1x), 8(2x), 7(2x), 4(1x), 2(2x)	0.6	0.82	2	107	105
25	88(1x), 46(1x), 31(1x), 17(2x), 16(2x), 11(1x), 10(2x), 8(4x), 7(2x), 5(1x), 4(3x), 2(5x)	0.62	0.75	2	88	86
50	67(1x), 25(1x), 17(2x), 16(1x), 15(1x), 10(2x), 8(6x), 7(1x), 6(4x), 5(3x), 4(4x), 3(6x), 2(12x), 1(6x)	0.65	0.74	1	67	66
75	34(1x), 21(1x), 16(1x), 13(1x), 9(1x), 8(4x), 7(2x), 6(7x), 5(4x), 4(14x), 3(8x), 2(23x), 1(8x)	0.67	0.7	1	34	33
90	19(1x), 16(1x), 9(2x), 8(5x), 7(3x), 6(5x), 5(4x), 4(15x), 3(12x), 2(33x), 1(9x)	0.64	0.68	1	19	18
110	19(1x), 16(1x), 9(1x), 8(2x), 7(3x), 6(3x), 5(6x), 4(11x), 3(17x), 2(46x), 1(19x)	0.67	0.67	1	19	18

BAB 4 IMPLEMENTASI DAN PENGUJIAN

130	16(2x), 7(1x), 6(4x), 5(6x), 4(11x), 3(22x), 2(48x), 1(36x)	0.61	0.66	1	16	15
150	16(1x), 15(1x), 6(3x), 5(3x), 4(5x), 3(26x), 2(62x), 1(49x)	0.68	0.65	1	16	15
175	16(1x), 15(1x), 5(1x), 4(5x), 3(20x), 2(72x), 1(75x)	0.65	0.65	1	16	15
195	14(1x), 8(1x), 5(1x), 4(4x), 3(19x), 2(66x), 1(103x)	0.61	0.65	1	14	13
200	12(1x), 6(1x), 5(1x), 4(4x), 3(19x), 2(65x), 1(109x)	0.59	0.65	1	12	11
208	9(1x), 5(1x), 4(4x), 3(19x), 2(65x), 1(118x)	0.57	0.65	1	9	8
220	5(1x), 4(3x), 3(20x), 2(62x), 1(134x)	0.54	0.65	1	5	4
232	5(1x), 4(3x), 3(17x), 2(56x), 1(155x)	0.55	0.65	1	5	4
245	4(1x), 3(11x), 2(65x), 1(168x)	0.64	0.64	1	4	3
275	3(4x), 2(52x), 1(219x)	0.69	0.64	1	3	2
290	3(2x), 2(41x), 1(247x)	0.71	0.64	1	3	2
305	3(1x), 2(28x), 1(276x)	0.78	0.63	1	3	2
325	2(10x), 1(315x)	0.81	0.62	1	2	1

Pada tabel 4.8 dapat dilihat bahwa jumlah service yang dibuat oleh complete linkage cenderung membentuk service yang besar karena dilihat dari jumlah module yang dimiliki oleh setiap service, namun nilai rentang dari besarnya sebuah service lebih kecil. Di sisi lain nilai coupling dan cohesion yang dihasilkan buruk walaupun jumlah service masih sedikit bila dibandingkan dengan linkage lain.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Tabel 4.9 Perbandingan Ukuran Service yang dihasilkan oleh Average Linkage

Ukuran Service	Jumlah Module	Coupling	Cohesion	Statistik Jumlah Module		
				MIN	MAX	RANGE
1	335(1x)	0.0	1.0	335	335	0
2	333(1x), 2(1x)	0.67	1.0	2	333	331
5	329(1x), 2(2x), 1(2x)	0.72	1.0	1	329	328
10	305(1x), 20(1x), 2(2x), 1(6x)	0.45	1.0	1	305	304
15	293(1x), 20(1x), 6(1x), 2(4x), 1(8x)	0.18	0.98	1	293	292
25	198(1x), 51(1x), 20(1x), 11(1x), 9(1x), 8(1x), 5(3x), 4(1x), 2(4x), 1(11x)	0.25	0.9	1	198	197
50	74(1x), 54(1x), 17(1x), 16(2x), 15(1x), 14(1x), 11(1x), 8(1x), 7(1x), 6(3x), 5(6x), 4(1x), 3(5x), 2(11x), 1(14x)	0.33	0.81	1	74	73
75	29(1x), 28(1x), 17(2x), 16(1x), 12(2x), 11(1x), 10(2x), 8(1x), 7(2x), 6(4x), 5(5x), 4(5x), 3(7x), 2(20x), 1(21x)	0.42	0.72	1	29	28
90	29(1x), 19(1x), 16(1x), 13(1x), 11(2x), 10(1x), 7(2x), 6(9x), 5(3x), 4(11x), 3(7x), 2(27x), 1(24x)	0.45	0.72	1	29	28
110	29(1x), 19(1x), 15(1x), 11(2x), 8(1x), 7(2x), 6(8x), 5(4x), 4(7x), 3(8x), 2(33x), 1(42x)	0.41	0.7	1	29	28

BAB 4 IMPLEMENTASI DAN PENGUJIAN

130	16(1x), 15(1x), 13(1x), 11(1x), 10(1x), 8(1x), 6(7x), 5(5x), 4(2x), 3(14x), 2(49x), 1(47x)	0.47	0.67	1	16	15
150	13(2x), 10(2x), 8(1x), 6(5x), 5(5x), 4(1x), 3(16x), 2(56x), 1(62x)	0.56	0.67	1	13	12
175	13(2x), 10(1x), 8(1x), 6(4x), 5(5x), 4(1x), 3(14x), 2(49x), 1(98x)	0.49	0.67	1	13	12
195	13(1x), 10(2x), 8(1x), 6(3x), 5(4x), 4(1x), 3(11x), 2(47x), 1(125x)	0.46	0.67	1	13	12
200	13(1x), 10(1x), 8(1x), 6(3x), 5(5x), 4(1x), 3(11x), 2(47x), 1(130x)	0.45	0.67	1	13	12
208	13(1x), 10(1x), 8(1x), 6(2x), 5(4x), 4(2x), 3(10x), 2(47x), 1(140x)	0.44	0.67	1	13	12
220	10(1x), 6(1x), 5(5x), 4(3x), 3(11x), 2(50x), 1(149x)	0.49	0.66	1	10	9
232	6(1x), 5(4x), 4(2x), 3(14x), 2(48x), 1(163x)	0.52	0.65	1	6	5
245	5(1x), 4(4x), 3(11x), 2(52x), 1(177x)	0.56	0.65	1	5	4
275	3(8x), 2(44x), 1(223x)	0.68	0.64	1	3	2
290	3(6x), 2(33x), 1(251x)	0.71	0.64	1	3	2
305	3(4x), 2(22x), 1(279x)	0.76	0.63	1	3	2
325	2(10x), 1(315x)	0.81	0.62	1	2	1

Pada tabel 4.9 dapat dilihat bahwa jumlah service dibuat oleh average linkage memiliki service yang tidak terlalu besar bila dilihat dari nilai maksimum jumlah module dan terdapat service yang berukuran kecil (1 modul). Dari nilai coupling dan nilai cohesion, average linkage memiliki nilai yang bagus karena

nilai coupling masih dibawah nilai cohesion.

Dari hasil perbandingan ukuran service yang dihasilkan oleh masing-masing linkage pada kasus dekomposisi aplikasi monolitik Odoo, menunjukan bahwa complete linkage tidak cocok untuk melakukan dekomposisi karena service yang dibentuk besar dan memiliki nilai coupling tinggi. Sedangkan single linkage kurang ideal walaupun memiliki coupling yang rendah dan cohesion yang tinggi namun service yang dibuat tidak dikelompokan secara seimbang dan memiliki nilai rentang yang besar, karena ada service yang memiliki jumlah sangat besar. Untuk Average linkage dapat membentuk service yang ideal karena ukuran service tidak terlalu besar dilihat dari nilai rentangnya dan memiliki nilai couplingnya rendah serta nilai cohesionnya cukup tinggi.

Pemilihan jumlah service yang ideal pada Average linkage, tidak bisa ditentukan secara pasti karena terdapat *trade-off* untuk setiap kenaikan jumlah service. Oleh karena itu rentang jumlah service yang ideal adalah bisa dari 175, 195, 208, 200, 220, 232 , 245. Jumlah service yang tidak ideal adalah dibawah ± 150 / ± 175 service karena nilai rentang isi module service cukup tinggi yaitu diatas 12 module dan diatas ± 245 service karena nilai coupling lebih tinggi dibandingkan nilai cohesionnya.

4.3.2 Dekomposisi Monolitik ke Microservice

Pada bagian ini menjelaskan bagaimana penerapan microservice dengan menggunakan hasil dekomposisi yang dihasilkan oleh hierarchical clustering.

4.3.2.1 Pemilihan Service

Pada tugas akhir ini memilih untuk menggunakan hasil Hierarchical Clustering dengan Average Linkage dengan jumlah service sebesar 200. Hasil evaluasi sebelumnya menunjukkan nilai coupling sebesar 0.67, nilai cohesion sebesar 0.45 dan nilai rentang jumlah modulanya sebesar 12. Detail isi dari setiap module untuk setiap partisi yang dikelompokan dapat dilihat pada Tabel. 4.10

Tabel 4.10 Daftar Module untuk Setiap Partisi

Partisi	Daftar Module
1	auth_totp_mail, base.wizard.base_partner_merge, base.models.ir_sequence, contacts, base.models.ir_asset
2	base.models.res_users, fleet, sales_team, iap, account_check_printing, base_vat, partner_autocomplete, snailmail_account, snailmail_membership
3	auth_signup, portal
4	web
5	hr_work_entry_contract, resource, hr_work_entry, hr_holidays, hr_contract
6	base.models.ir_http, google_recaptcha
7	base.models.res_config, digest, base_geolocalize, auth_ldap, auth_password_policy
8	base.models.res_partner, uom, base_sparse_field
9	website_sale_loyalty, sale_loyalty
10	loyalty, sale, stock, account, product, analytic, point_of_sale, hr_expense, purchase_stock, purchase, mrp, project, lunch
13	delivery, delivery_mondialrelay
14	payment, base.models.res_company
17	calendar, rating, survey, sms, phone_validation, gamification, bus, privacy_lookup
20	http_routing
22	mail, crm
32	hr_recruitment_skills, hr_skills
56	pos_epson_printer_restaurant, pos_restaurant
57	website_event_booth_sale, website_sale_slides
58	base.models.ir_model, base.models.ir_ui_menu, hr_timesheet_attendance

BAB 4 IMPLEMENTASI DAN PENGUJIAN

80	website_event_meet_quiz
101	website_event_booth_exhibitor
133	account_qr_code_sepa
159	mass_mailing_event_track_sms
200	sale_purchase_stock

Service yang dipilih untuk dilakukan implementasi dekomposisi yaitu service ke-10 yang memiliki module addons.product dan addons.point_of_sale, kemudian service ke-17 yang memiliki modul calendar. Di dalam masing-masing module ini terdapat beberapa model yang dapat dilakukan dekomposisi, berikut pada Tabel. 4.11 berisi model dari module yang dipilih dan keterhubungannya yang ditemukan pada proses pengelompokan.

Tabel 4.11 Daftar Model untuk masing-masing Module

Addons/Module - Model	Detail Relasi & Jumlah Panggilannya
product.models	product_product mail.models: 2, product.models: 17, base.models: 2
	product_attribute product.models: 5, base.models: 2
	product_pricelist base.models: 5, product.models: 10
	product_pricelist_item product.models: 8, base.models: 2
	product_template mail.models: 2, base.models: 6, product.models: 51, uom.models: 2
	decimal_precision decimal_precision: 1, product.models: 0
	product_category product.models: 0, base.models: 2
	product_packaging base.models: 3, product.models: 1
	product_supplierinfo base.models: 5, product.models: 2
	product_tag base.models: 2, product.models: 3
	res_company base.models: 1, product.models: 0
	res_config_settings base.models: 1, product.models: 0
	res_country_group base.models: 1, product.models: 1
	res_currency base.models: 1, product.models: 0

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Addons/Module - Model		Detail Relasi & Jumlah Panggilannya
point_of_sale.models	res_partner	base.models: 1, product.models: 1
	uom_uom	uom.models: 1, product.models: 0
	res_config_settings	base.models: 1, product.models: 4, point_of_sale.models: 5, account.models: 2
	pos_payment_method	base.models: 3, point_of_sale.models: 3, account.models: 3
	stock_picking	stock.models: 4, point_of_sale.models: 8
	pos_category	point_of_sale.models: 1, base.models: 2
	res_partner	base.models: 1, point_of_sale.models: 1
	pos_config	product.models: 3, base.models: 9, point_of_sale.models: 31, account.models: 5, stock.models: 3
	pos_session	stock.models: 1, point_of_sale.models: 86
	pos_order	point_of_sale.models: 37
	account_bank_statement	account.models: 1, point_of_sale.models: 1
	account_journal	account.models: 1, point_of_sale.models: 1
	account_move	account.models: 2, point_of_sale.models: 2
	account_payment	account.models: 2, point_of_sale.models: 2
	account_tax	account.models: 1, point_of_sale.models: 0
	barcode_rule	barcodes.models: 1, point_of_sale.models: 0
	chart_template	account.models: 1, point_of_sale.models: 0
	digest	digest.models: 1, point_of_sale.models: 0

Addons/Module - Model	Detail Relasi & Jumlah Panggilannya
	pos_bill base.models: 2, point_of_sale.models: 1
	pos_payment account.models: 1, base.models: 2, point_of_sale.models: 2
	product product.models: 2, point_of_sale.models: 1, uom.models: 2
	res_company base.models: 1, point_of_sale.models: 0
	stock_rule stock.models: 1, point_of_sale.models: 0
	stock_warehouse stock.models: 2, point_of_sale.models: 0
calendar.models	calendar_event mail.models: 3, calendar.models: 43, base.models: 6
	calendar_alarm_manager calendar.models: 4
	res_users base.models: 1, calendar.models: 1
	res_partner base.models: 1, calendar.models: 6
	calendar_recurrence calendar.models: 28, base.models: 2
	calendar_attendee base.models: 6, calendar.models: 3
	calendar_alarm base.models: 2, mail.models: 1, calendar.models: 0
	calendar_event_type base.models: 2, calendar.models: 0
	calendar_filter base.models: 4, calendar.models: 0
	ir_http base.models: 1, calendar.models: 0
	mail_activity mail.models: 1, calendar.models: 1
	mail_activity_mixin mail.models: 1, calendar.models: 1
	mail_activity_type mail.models: 1, calendar.models: 0

4.3.2.2 Endpoint API

Model yang dibuat pada tugas akhir ini yaitu model 'calendar_event_type', 'product_tag', dan 'pos_category', pada Tabel 4.12 dijelaskan endpoint dan routing yang dipakai dalam masing-masing service. Untuk Client berkomunikasi dengan service maka harus melalui API Gateway dan setiap service untuk mencari service lainnya dapat diarahkan oleh API Gateway. API Gateway membuat rute berdasarkan URL path yang sudah dikonfigurasi sebelumnya serta service dan API Gateway berada di jaringan yang sama.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Berikut IP Address dan port yang digunakan setiap service:

1. Kong Gateway → 172.18.0.5:8000
2. Monolitik → 172.18.0.55:8069
3. Service 10 (Product dan Point of Sale) → 172.18.0.51:1323
4. Service 17 (Calendar) → 172.18.0.52:1324

Tabel 4.12 Tabel Endpoint API

Metode	Route	Body	Keterangan
Monolith (/web/session/*)			
POST	/authenticate	JSON, Format: JSON-RPC	Melakukan sesi autentikasi pengguna
POST	/destroy	JSON, Format: JSON-RPC	Menghilangkan sesi autentikasi pengguna
Service 10 (/web/dataset/call_kw/product.tag/* & /web/dataset/call_kw/pos.category/*)			
POST	/read	JSON, Format: JSON-RPC	Mendapatkan data Point Of Sale Category
POST	/create	JSON, Format: JSON-RPC	Menambahkan data Point Of Sale Category
POST	/read	JSON, Format: JSON-RPC	Mendapatkan data Tag pada product
POST	/create	JSON, Format: JSON-RPC	Menambahkan data Tag pada product
Service 17 (/web/dataset/call_kw/calendar.event.type/*)			
POST	/read	JSON, Format: JSON-RPC	Mendapatkan data tipe event
POST	/create	JSON, Format: JSON-RPC	Menambahkan data tipe event baru

4.3.2.3 Daftar Class

Pada bagian ini dijelaskan class yang digunakan Service 10 dan Service 17:

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Tabel 4.13 Tabel class yang digunakan

Class	Atribut	Tipe Data	Keterangan
ResUser	UID	int	Class yang menyimpan informasi mengenai pengguna aplikasi/ Odoo.
	DisplayName	string	Hampir semua aktivitas pengguna harus dicatat dan harus divalidasi melalui GroupID.
	GroupID	int	Informasi ini dimasukan dari nilai JWT yang disimpan client.
AccessControlList	GroupID	int	Class yang menyimpan hak akses suatu model atau resource dan dapat dilihat nilai GroupID.
	ModelID	int	Hak akses ini dikomunikasikan dalam waktu tertentu oleh masing-masing service.
	Read	bool	
	Write	bool	
	Create	bool	
	Unlink	bool	
Service 10 (Product dan Point of Sales)			

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Class	Atribut	Tipe Data	Keterangan
PosCategory	ID	uint	PosCategory atau Point of Sales Category adalah untuk membantu mengelola barang yang yang didagangkan melalui Point of Sale.
	ParentID	uint	Kategori ini bisa memiliki tingkatan kategori dan gambar dari kategori tersebut.
	Name	string	
	Sequence	int	
	Image	PosCategoryImage	
	CreateAt	Timestamp	
	CreatUID	ResUser	
	WriteAt	Timestamp	
PosCategoryImage	WriteUID	ResUser	
	UID	int	Merupakan object yang menyimpan informasi gambar yang dimiliki oleh kategori.
	ImageBase64	string	Gambar disimpan didatabase dan memiliki IdAttachment yang diketahui oleh sistem monolith
	AttachmentId	int	
	ImageSizeByte	int	

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Class	Atribut	Tipe Data	Keterangan
ProductTag	ID	uint	Product Tag adalah model yang mengelola banyak label untuk memudahkan pencarian disuatu produk.
	Color	uint	Product Tag memiliki satu nama yang unik tetapi bisa dimiliki oleh banyak produk template atau produk-produk (varian).
	Name	string	
	CreateAt	Timestamp	
	CreatUID	ResUser	
	WriteAt	Timestamp	
	WriteUID	ResUser	
	ProductTemplate	int[]	
	ProductProduct	int[]	
Service 17 (Calendar)			
MeetingType	ID	uint	Meeting Type adalah model yang mengelola banyak label yang terhubung dengan event di menu Calendar
	Color	uint	
	Name	string	
	CreateAt	Timestamp	
	CreatUID	ResUser	
	WriteAt	Timestamp	
	WriteUID	ResUser	

4.3.2.4 Docker

Pada tugas akhir ini menggunakan teknologi Docker untuk membangun dan mendistribusikan aplikasi dalam bentuk container. Setiap service akan terhubung pada jaringan yang sama. Setiap service termasuk aplikasi monolith memiliki image yang bisa dilakukan deployment antar platform.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```
$ docker network create --subnet=172.18.0.0/16 mono-net
$ docker network create --subnet=172.18.0.0/16 odoo-ms-net
```

Listing 4.3: Shell Script untuk pembuatan jaringan

Berikut adalah inisialisasi untuk membuat container dari masing-masing service, setiap serive berada di jaringan internal odoo yang sudah ditentukan sebelum. Tujuannya agar pemetaan service bisa dilakukan dengan target ip yang konsisten.

```
$ docker run -v "/ubuntu/local":/home/odoo/shared_data --net
          odoo-ms-net --ip 172.18.0.55 --name odoo-mono -p :8069
          -d odoo-mono
$ docker run --net odoo-ms-net --ip 172.18.0.54 --name odoo-
          mariadb -e MYSQL_ROOT_PASSWORD=mariadb -p 3306:3306 -d
          mariadb
$ docker run --net odoo-ms-net --ip 172.18.0.11 --name
          postgres-odoo -e POSTGRES_PASSWORD=postgres -p 5432:5432
          -d postgres
```

Listing 4.4: Shell Script untuk pembuatan container

4.3.2.5 Kong

Proses selanjutnya yaitu membuat Api Gateway, tugas akhir ini menggunakan Kong sebagai api gateway. Konfigurasi Kong yang digunakan tidak menggunakan database tetapi berupa konfigurasi statik berupa file .yml/.json. Tugas akhir ini menggunakan yml.

```
$ docker run -d --name odoo-kong-gateway \
--network=odoo-ms-net --ip 172.18.0.5 -v "$(pwd):/kong/
declarative/" \
-e "KONG_DATABASE=off" \
-e "KONG_DECLARATIVE_CONFIG=/kong/declarative/kong.yml" \
-e "KONG_PROXY_ACCESS_LOG=/dev/stdout" \
-e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" \
-e "KONG_PROXY_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_LISTEN=0.0.0.0:8001" \
-p 8000:8000 -p 8001:8001 -p 8002:8002 -p 8003:8003 \
kong/kong-gateway:latest
```

Listing 4.5: Shell Script untuk pembuatan container

Endpoint yang diset adalah keterangan service (ip address,port,protocol) dan routing (metode, apakah host di rewrite ketika dialihkan). Dengan api gateway client hanya perlu mengetahui endpoint pusat kong yaitu :8000. Ketika pencarian rute url tidak ditemukan maka otomatis diarah service monolith (odoo-mono).

```

1 _format_version: "3.0"
2 _transform: true
3
4 services:
5 - name: odoo-mono
6   port: 8069
7   protocol: http
8   host: 172.18.0.55
9   routes:
10    - name: odoo-mono-route
11      methods:
12        - GET
13        - POST
14        - PUT
15        - DELETE
16        - PATCH
17        - HEAD
18        - OPTION
19      paths:
20      - /
21      strip_path: false
22      preserve_host: true
23
24 - name: service-10
25   port: 1323
26   protocol: http
27   host: 172.18.0.51
28   path: /jsonrpc
29   routes:
30    - name: service-10-route
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
methods:
  - POST
paths:
  - /web/dataset/call_kw/pos.category/create
  - /web/dataset/call_kw/pos.category/write
  - /web/dataset/call_kw/pos.category/unlink
  - /web/dataset/call_kw/pos.category/read
  - /web/dataset/call_kw/product.tag/create
  - /web/dataset/call_kw/product.tag/write
  - /web/dataset/call_kw/product.tag/unlink
  - /web/dataset/call_kw/product.tag/read
strip_path: false
preserve_host: true
- name: service-17
  port: 1324
  protocol: http
  host: 172.18.0.52
  path: /jsonrpc
  routes:
    - name: service-17-route
      methods:
        - POST
      paths:
        - /web/dataset/call_kw/calendar.event.type/create
        - /web/dataset/call_kw/calendar.event.type/write
        - /web/dataset/call_kw/calendar.event.type/unlink
        - /web/dataset/call_kw/calendar.event.type/read
      strip_path: false
      preserve_host: true

```

Gambar 4.26 Konfigurasi kong-declarative di kong.yml

4.3.2.6 Penerapan JSON Web Token(JWT)

Perubahan migrasi arsitektur dari monolith ke microservice, membuat diperlukan sistem pengelolaan hak akses yang stateless. Sehingga service tidak sering harus berkomunikasi satu sama lain terutama pada hal informasi yang sama. Token disimpan didalam cookie client, sehingga baik dari client yang melalui user-interface atau pun secara langsung API, bisa diidentifikasi selama client tersebut memberikan token di cookie ketika meminta data.

```
{
  "uid": 2,
  "user_context": {
    "lang": "en_US",
    "tz": "Europe/Brussels",
    "uid": 2
  },
  "db": "odoo_db",
  "company_id": 1,
  "partner_id": 3,
```

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```
"group_id": [
    2, 22, 7, 34, 48
],
"exp": 1692576855
}
```

Listing 4.6: Isi data berupa JSON di JWT

Untuk menambahkan JWT dapat dituliskan pada proses login sendiri seperti pada 4.27 file py yaitu addons / web/ controlers / session.py . Pada route /web/session/authenticate

```
class Session(http.Controller):
    @http.route('/web/session/get_session_info', type='json', auth="user")
    def get_session_info(self):
        pass

    @http.route('/web/session/authenticate', type='json', auth="none")
    def authenticate(self, db, login, password, base_location=None):
        if not http.db_filter([db]):
            raise AccessError("Database not found.")
        pre_uid = request.session.authenticate(db, login, password)
        if pre_uid != request.session.uid:
            request.session.db = db
            registry = odoo.modules.registry.Registry(db)
            with registry.cursor() as cr:
                env = odoo.api.Environment(cr, request.session.uid, request.session.context)
                if not request.db and not request.session.is_explicit:
                    # request._save_session would not update the session_token
                    # as it lacks an environment, rotating the session myself
                    http.root.session_store.rotate(request.session, env)
                request.future_response.set_cookie(
                    'session_id', request.session.sid,
                    max_age=http.SESSION_LIFETIME, httponly=True,
                )
            jwt_handler = JWTHandler()
            request.future_response.set_cookie(
                'jwt', jwt_handler.encode(pre_uid),
                max_age=http.SESSION_LIFETIME, httponly=True
            )
        return env['ir.http'].session_info()
```

Gambar 4.27 Penggalan Kode untuk disisipi proses JWT

Data yang disimpan oleh JWT penting diketahui secara umum service, seperti pengguna, konteks pengguna(bahasa, zona waktu), perusahaan, dan database(db). Perusahaan dan database diperlukan karena Odoo memiliki fitur untuk memilih database dan untuk memanggil aplikasi monolith(Odoo) harus diketahui databasenya. JWT ini dibuat oleh aplikasi monolith ketika login dan dihapus ketika logout atau sesi sudah habis(exp). Masa hidup JWT sama dengan masa hidup session-token yaitu 3 bulan.

Untuk menghancurkan sesi maka ditambahkan mekanisme berupa cookie yang menunjukkan bahwa jwt sudah tidak valid. Kemudian cookie tersebut akan

BAB 4 IMPLEMENTASI DAN PENGUJIAN

dicek oleh fungsi di odoo / http.py / Request.savesession untuk menghapus cookie dengan nama 'jwt' .

```
@http.route('/web/session/destroy', type='json', auth="user")
def destroy(self):
    request.session.logout()
    request.future_response.set_cookie(f'invalidate_jwt', '1', max_age=0, httponly=True)

@http.route('/web/session/logout', type='http', auth="none")
def logout(self, redirect='/web'):
    request.session.logout(keep_db=True)
    request.future_response.set_cookie(f'invalidate_jwt', '1', max_age=0, httponly=True)
    return request.redirect(redirect, 303)
```

Gambar 4.28 Potongan Kode untuk memberikan validasi bawah JWT tidak valid

```
def _save_session(self):
    """ Save a modified session on disk. """
    sess = self.session

    if not sess.can_save:
        ...

    if sess.should_rotate:
        ...
    elif sess.is_dirty:
        ...

    # We must not set the cookie if the session id was specified
    # using a http header or a GET parameter.
    # There are two reasons to this:
    # - When using one of those two means we consider that we are
    #   overriding the cookie, which means creating a new session on
    #   top of an already existing session and we don't want to
    #   create a mess with the 'normal' session (the one using the
    #   cookie). That is a special feature of the Javascript Session.
    # - It could allow session fixation attacks.
    cookie_sid = self.httprequest.cookies.get('session_id')
    if not sess.is_explicit and (sess.is_dirty or cookie_sid != sess.sid):
        self.future_response.set_cookie('session_id', sess.sid, max_age=SESSION_LIFETIME, httponly=True)

    # Cek Bila Ada Permintaan Revalidate JWT
    for c in request.future_response.headers:
        if c[0] == 'Set-Cookie' and c[1][0:16] == 'invalidate_jwt=1':
            _logger.info("Invalidate JWT")
            self.future_response.set_cookie(f'jwt', f'', max_age=0, httponly=True)
    return
```

Gambar 4.29 Potongan Kode untuk memberikan validasi bawah JWT tidak valid

Pada gambar 4.30, merupakan tangkapan layar pada ResponseHeader melalui DevTools di Browser. Pada gambar bagian kiri terdapat bagian response ketika cookie dengan key JWT berhasil di'set' oleh server, dan pada gambar sisi kanan menunjukan reponse ketika cookie pada JWT dihilangkan.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Kondisi Cookie Ketika Login	
<input type="checkbox"/> General	
Request URL:	http://localhost:8000/web/login
Request Method:	POST
Status Code:	303 SEE OTHER
Remote Address:	(::1):8000
Referrer Policy:	strict-origin-when-cross-origin
<input type="checkbox"/> Response Headers	
Connection:	keep-alive
Content-Length:	215
Content-Type:	text/html; charset=utf-8
Date:	Fri, 02 Jun 2023 13:58:24 GMT
Location:	http://localhost:8000/web
Server:	Werkzeug/0.16.1 Python/3.9.2
Set-Cookie:	jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ1aWQiOiIsInVzZXJY29udGV4dGt6eyJsYW5nIjoiZW5fVVMiLCJ0eiI6IkFzaWEvSmFrYXJ0YSlzInVpZC16Mn0slmRljlbt21vb19Ky1slsmVbXBhbnlfWQfOEslnBhcnRuZXJufaWQfOJMsImdyb3Wx2lkjpBmIxwMlw4LDm1LDUyLDQ5LDE2LD11LD13LDmOsSwlxLDE0LDm2LDE5LDUsNiw1Nw0LDiyDcsNdgsmzsqsNTesNTBdLcJleHAo)E2OTMD0TAzMDR9. KmugCPupznbQk-aWqZ_Wzmbi9Gdgvx8Uldfrd8yg; Expires=Thu, 31-Aug-2023 13:58:24 GMT; Max-Age=7776000; HttpOnly; Path=/; session_Id=292b21d87858c732bf9a70204d44f14c18b15c; Expires=Thu, 31-Aug-2023 13:58:24 GMT; Max-Age=7776000; HttpOnly; Path=/; Via: kong/3.3.0; X-Kong-Proxy-Latency: 2; X-Kong-Upstream-Latency: 435
<input type="checkbox"/> Raw	

Kondisi Cookie Ketika Logout	
<input type="checkbox"/> General	
Request URL:	http://localhost:8000/web/session/logout
Request Method:	GET
Status Code:	303 SEE OTHER
Remote Address:	(::1):8000
Referrer Policy:	strict-origin-when-cross-origin
<input type="checkbox"/> Response Headers	
Connection:	keep-alive
Content-Length:	215
Content-Type:	text/html; charset=utf-8
Date:	Fri, 02 Jun 2023 13:58:53 GMT
Location:	http://localhost:8000/web
Server:	Werkzeug/0.16.1 Python/3.9.2
Set-Cookie:	invalidate_jwt=1; Expires=Fri, 02-Jun-2023 13:58:53 GMT; Max-Age=0; HttpOnly; Path=/; jwt=; Expires=Fri, 02-Jun-2023 13:58:53 GMT; Max-Age=0; HttpOnly; Path=/; session_Id=37ab0fb3027e03149fc7a36748cb16eaef638fe9; Expires=Thu, 31-Aug-2023 13:58:53 GMT; Max-Age=7776000; HttpOnly; Path=/; Via: kong/3.3.0; X-Kong-Proxy-Latency: 3; X-Kong-Upstream-Latency: 35
<input type="checkbox"/> Raw	

Gambar 4.30 Ilustrasi JWT di Cookie

4.3.2.7 Pola *Strangle Fig* dan *Branch by Abstraction*

Perubahan arsitektur dari aplikasi monolith menjadi microservice membutuhkan banyak resiko dan sumber daya untuk itu perubahan sebaiknya tidak mengurangi atau memperburuk aplikasi yang sudah jadi. Setiap proses bisnis odo selalu mengimplementasikan class MetaModel. MetaModel membantu pengembang bisa membuat cepat aplikasi tapi MetaModel memiliki keterkaitan dengan model lainnya tinggi. Model di Odoo bisa mengelola data seperti menyimpan di database, memanggil model lainnya, data untuk tampilan(Frontend), dan hal lainnya.

Di MetaModel terdapat fungsi umum dan fungsi internal yang bisa di'override'. Fungsi tersebut dapat dijadikan sebuah adapter. Fungsi utama yang selalu ada pada model yaitu fungsi create, browse, unlink, _read dan read. Fungsi dan metode tersebut berfungsi untuk mengelola data. Model Adapter dibuat untuk mengabstraksi pengelolaan data sehingga implementasi logic bisa dipindahkan atau dialihkan ke service lainnya dan dikomunikasikan melalui panggilan RPC.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```

class PosCategoryAbstract(models.AbstractModel):
    _name = "pos.category.abstract"
    _description = "Pos Category Abstract RPC"
    _table = "pos_category_abstract"
    _model_name = "pos.category"
    _HOST_RPC = "http://172.18.0.51:1323"
    _PATH_RPC = "/jsonrpc/web/database/call_kw/pos.category/"
    _TARGET_COLUMN = [ '_id', 'name', 'parent_id', 'sequence', 'create_uid',
                      'create_date', 'write_uid', 'write_date', 'display_name', 'has_image' ]
    _TARGET_TYPE_COLUMN = [ int, str, int, int, int, datetime, str, bool ]

    def create_payload(self,modelName,method,args,context,idCall):-
        if _is_same_data(self, old, new):-
            return

    def call_ppc(self, method, headers, context , args=[],keyResult='result'):
        payload = self.create_payload(self._MODEL_NAME, method,args,context,i)
        # print(json.dumps(payload))
        try:
            _logger.info(f"Request RPC: {self._HOST_RPC}{self._PATH_RPC}")
            response = requests.post(f"{self._HOST_RPC}{self._PATH_RPC}", headers=headers, data=json.dumps(payload))
            if response.ok:
                result = response.json()
                if 'error' in result:
                    raise Exception(f"Internal Error In Result RPC : {result['error']}")
                # print(result)
                return result[keyResult]
            else:
                print(response.text)
                raise Exception(f"Internal Error From RPC : {self._HOST_RPC}(self._PATH_RPC) - Response Not OK")
        except Exception as e:
            raise Exception(f"Internal Error By Request RPC: {self._HOST_RPC}{self._PATH_RPC} - {e}")

    def manual_insert(self,env,records):-
        if len(records) == 0:
            return
        try:
            listColumn = ",".join(self._TARGET_COLUMN)
            query = f"INSERT INTO {self._TABLE_NAME} ({listColumn}) VALUES "
            targetValues = f"({', '.join(['%s'] * len(self._TARGET_COLUMN))})"

            values = []
            for record in records:
                values.append(targetValues)
            tmpRV = []
            c = self._TARGET_COLUMN
            ct = self._TARGET_TYPE_COLUMN
            for i in range(0,len(self._TARGET_COLUMN)):
                newData = record.get(c[i])
                tmpRV.append(self.convert_to_odoo_type(newData, ct[i]))
                realValues += tmpRV
            query += ', '.join(values)

            placeholders = tuple(realValues)
            preview_sql_query = query % placeholders
            print(preview_sql_query)
            env.cr.execute(query, realValues)
            env.cr.commit()
            _logger.info("Data inserted successfully into %s", self._TABLE_NAME)
        except Exception as e:
            _logger.error("Error inserting data into %s: %s", self._TABLE_NAME, str(e))
            raise

```

Gambar 4.31 Penerapan Class Abstract

Adapter yang dibuat juga memiliki kemampuan mengelola datanya karena data yang simpan hanyalah data yang diperlukan oleh sistem lainnya, seperti ID yang digunakan sebagai referensi hubungan antar module. Adapter akan mengelola data secara langsung dari database karena adapter harus berkomunikasi dahulu kepada service yang memiliki tanggung jawab data tersebut.

```

class PosCategoryAdapter(models.Model, PosCategoryAbstract):
    _name = "pos.category"
    _description = "Point of Sale Category"
    _order = "sequence, name"
    _rec_name = "display_name"

    name = fields.Char(string='Category Name')
    display_name = fields.Char(string='Display Name')
    parent_id = fields.Many2one('pos.category', string='Parent Category')
    child_ids = fields.One2many('pos.category', 'parent_id', string='Children Categories')
    sequence = fields.Integer(help="Gives the sequence order when displaying a list of product categories.")
    image_128 = fields.Image(max_width=128, max_height=128)
    has_image = fields.Boolean()

    @api.model
    def create(self, records):
        _logger.info("Create Triggered from Mono...")
        newID = self.call_rpc("create", dict(http.request.httprequest.headers), self.env.context, [records])
        return self.browse(newID)

    @api.model
    def browse(self, ids=None):
        if not ids:
            ids = []
        elif ids._class_ is int:
            ids = [ids]
        else:
            ids = tuple(ids)
        _logger.info("Browse Adapter Triggered :" + str(ids))
        currRow = self.manual_count(self.env)
        if currRow == 0:
            self.sync_data()
        return self._class_(self.env, ids, ids)

    def sync_data(self,ids = []):
        _logger.info("Sync Data Started !!")
        # Karena ketika ID's yang diminta semua maka ID request dihilangkan
        if len(ids) == self.manual_count(self.env):
            ids = []
        api_data = self.call_rpc("read", dict(http.request.httprequest.headers), self.env.context, [ids])
        existing_records = self.manual_get(self.env,ids=ids,returnobj=True)
        createList = []
        updateList = []
        sameCounter = 0

        for data in api_data:
            record = {}
            if data['id'] in existing_records:
                existing_record=data['id'][1]['_metadata_found'] = True
                record = existing_records[data['id']]
                if record:
                    if self._is_same_data(record,data) == False:
                        updateList.append(data)
                    else:
                        sameCounter += 1
                else:
                    createList.append(data)
            else:
                createList.append(data)

        deleteList = []

        # #Delete Cek, ketika semua data diambil dari service
        ex_key = list(existing_records.keys())
        for k in ex_key:
            _logger.info("Existing: %s(existing_records); Create: %s(createList); Update: %s(updateList)", len(existing_records), len(createList), len(updateList))

        self.manual_delete(self.env,deleteList)
        self.manual_insert(self.env,createList)
        self.manual_update(self.env,updateList)
        self.env[self._name].clear_caches()

```

Gambar 4.32 Penerapan Class Adapter

Ketika ditemukan bahwa data yang berbeda maka dilakukan Update dan didapatkan bila tidak ada, namun ketika data tersebut sebelumnya ada tetapi ketika dicek ditemukan tidak ada. Maka adapter akan mencoba menghapus data tersebut, hal ini akan membuat efek berkelanjutan dengan sistem yang lain. Bila ternyata data tersebut tidak dapat dihapus maka data tersebut hanya bisa didapatkan dan tidak bisa diubah.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

```
var availableLang = "en_US"
var availableMethod = [...]string{"create", "write", "read", "unlink"}

func ProcessRPC(c echo.Context) error {
    var requestBody dto.CustomJSONRPCContainer

    body, err := io.ReadAll(c.Request).Body
    if err != nil {
        return nil
    }

    err = json.Unmarshal(body, &requestBody)
    if err != nil {
        return nil
    }

    if requestBody.Method != "call" {
        _, modelExist := AvailableModel[requestBody.Params.Model]
        if !modelExist {
            return nil
        }
    }

    if requestBody.Params.Kwargs.Context.Lang != availableLang {
        return nil
    }

    for _, m := range availableMethod {
        if requestBody.Params.Method == m {
            jwtInfo, jwt := c.Get("jwt_data").(*dto.JWTData)
            if !jwt {
                return c.JSON(http.StatusBadRequest, map[string]string{"error": "No JWT Content"})
            }
            return c.JSON(http.StatusOK, modelSelector(requestBody.Params.Model, m, body, jwtInfo))
        }
    }
    return c.JSON(http.StatusMethodNotAllowed, map[string]string{"error": "No method available for"})
}

func createPosCategoryMapper(data dto.CreatePosCategoryDTO, user model.ResUser) {
    for _, p := range data.Params.Args {
        parentId, _ := odoo.Val2Int(p.ParentID)
        image128, _ := odoo.Val2String(p.Image128)
        newPos, err := pos.Create(
            p.Name,
            uint(parentId),
            p.Sequence,
            image128,
            user)

        if err != nil {
            fmt.Println(err.Error())
            return dto.GetResultDTO(data.ID, false)
        }
        return dto.GetResultDTO(data.ID, newPos)
    }
    return dto.GetResultDTO(data.ID, false)
}

func methodPosCategorySelector(method string, body []byte, context *dto.JWTData) interface{} {
    currModel := "pos.category"
    switch method {
    case "create":
        acc, gid := service.isAllowedByACL(service.ModelID[currModel], context.GroupID, "c")
        if acc {
            return map[string]string{"no access": "create"}
        }
        var createDTO dto.CreatePosCategoryDTO
        err := json.Unmarshal(body, &createDTO)
        if err != nil {
            return map[string]string{"error parse dto": err.Error()}
        }
        return createPosCategoryMapper(createDTO, model.ResUser{UID: int(context.Uid), GroupID: gid})
    case "read":
    case "write":
    case "unlink":
    default:
        return nil
    }
}

func Create(name string, parentID uint, sequence int, imageBase64 string, uid model.ResUser) (uint, error) {
    pos := model.PosCategory{
        Name: name,
        ParentID: parentID,
        Sequence: sequence,
        Image: imageBase64,
        User: uid,
    }

    var parentRef model.PosCategory
    if parentID != 0 {
        parentRef = pos
    }

    tx := utils.Database().Begin()
    CreateUserIfNotFound(tx, uid)

    if imageBase64 == "" {
        newImg := model.PosCategoryImage{
            ResImg: resImg,
            ResImgError: resImg.Error,
        }
        pos.Image = newImg
        res := tx.Create(&pos)
        if res.Error != nil {
            tx.Rollback()
            return 0, res.Error
        }
        tx.Commit()
        //Background Task mengirimkan data
        go SaveImageRemote(newImg, int(pos.ID))
        return pos.ID, nil
    }
}
```

Gambar 4.33 Penerapan di Sisi Service-10 di kasus menambahkan PosCategory

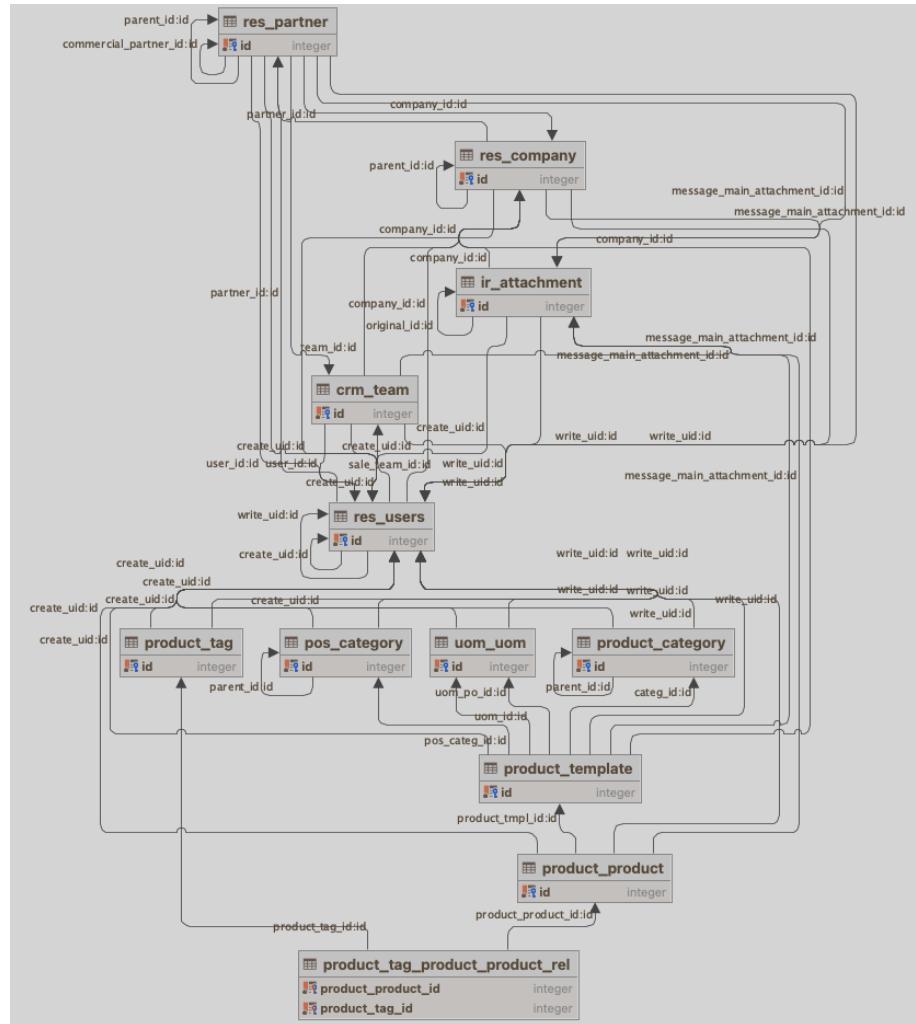
Pada pemisahan model PosCategory, proses penyimpanan data baru dapat memiliki sebuah gambar. Untuk itu ketika ada penyimpanan gambar maka gambar disimpan pada database untuk dicatat dan kemudian dikirimkan kepada aplikasi monolith untuk data melalui model ir.attachment di RPC agar gambar bisa diakses melalui web. Setiap model pada Odoo memiliki hak akses kepada grup tertentu. Hak akses bisa diasumsikan jarang berubah untuk karena itu agar service mengetahui apakah seorang user memiliki hak akses. Service memiliki jadwal melalui cron job untuk mengecheck apakah ada akses grup baru.

Service diminta untuk mencatat siapa dan kapan data berubah. Sehingga ada relasi kepada res.user relasi ini hanya memiliki ID, DisplayName, dan GroupID. Informasi ini didapatkan melalui data JWT yang dikirimkan oleh client sebelumnya.

4.3.2.8 Analisis Hasil Dekomposisi

Hasil Dekomposisi yang sudah diimplementasikan dengan 2 sampel service yaitu service ke-10 dan service ke-17. Didalam microservice yang dibangun terdapat 3 aplikasi yaitu aplikasi monolitik yang sudah dimodifikasi, Service ke-10(Product dan Point of Sale) dan Service ke-17 (Calendar). Untuk mengetahui apakah bagian yang didekomposisi oleh proses Hierarchical Clustering relevan dan dapat diimplementasikan maka dapat dilihat melalui struktur tabel yang terbentuk ketika melakukan implementasi.

Pada service ke-10 terdapat 2 model yang diimplementasikan yaitu product.tag dan pos.category, Odoo yang memiliki ORM sudah otomatis mengubah nama "dot" menjadi "underscore" sehingga bila dilihat dari nama tabel yang terbentuk akan memiliki nama "product_tag" dan "pos_category". Berikut adalah diagram database (4.34) yang menunjukan bagaimana keterhubungan antara model dengan model lainnya.



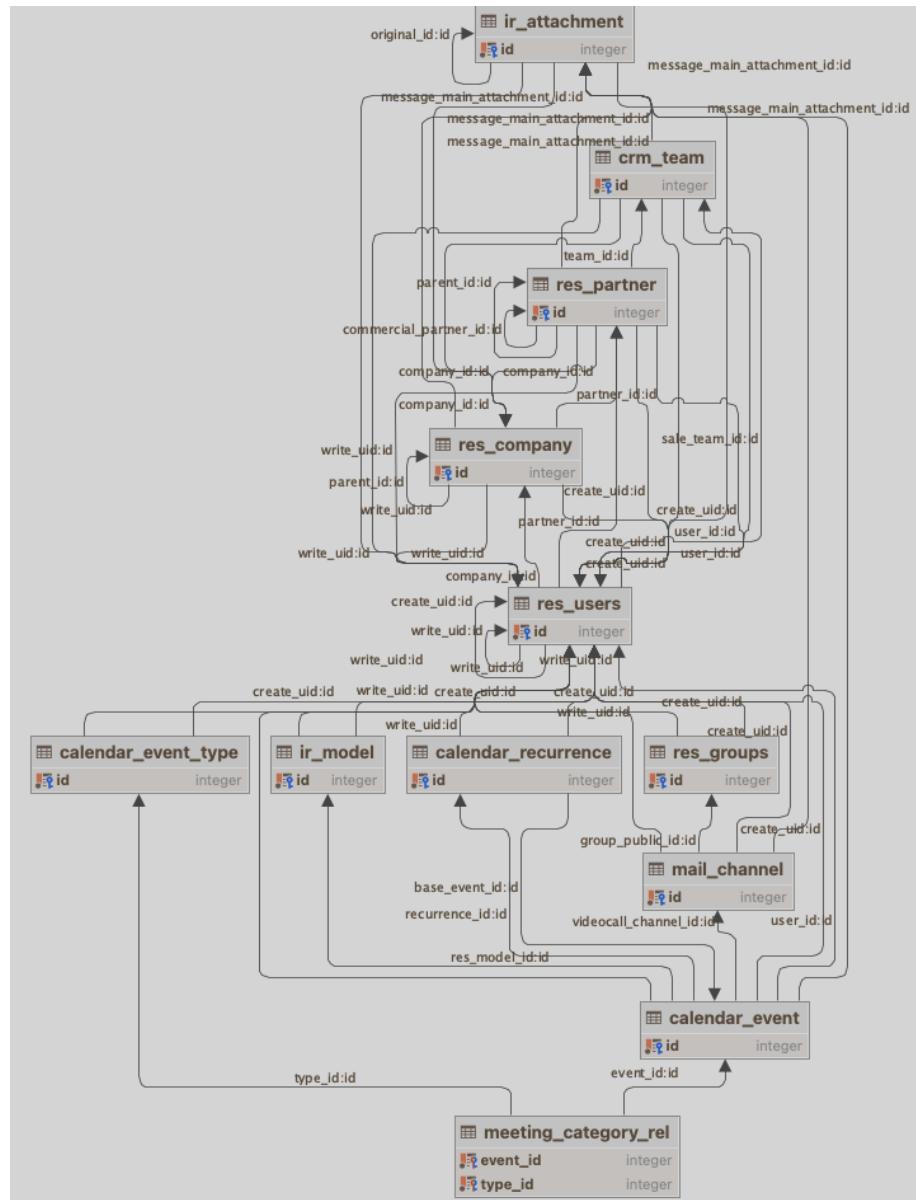
Gambar 4.34 Diagram Database dari Product Tag dan Pos Category

Hasil proses clustering sebelumnya mengelompokan 2 module yaitu product dan point of sale(POS) menjadi service sendiri. Dari diagram database menunjukan bahwa memang benar model product.tag dan pos.category memiliki keterhubungan satu sama lain lain melalui model product.template untuk itulah mengapa 2 modules isi disatukan menjadi service yang sama.

Pada service ke-17 terdapat 1 model yang diimplementasikan yaitu 'calendar.event.type' dan nama yang terbentuk pada tabel adalah

BAB 4 IMPLEMENTASI DAN PENGUJIAN

'calendar_event_type'. Service ke-17 ini memiliki isi module yang berbeda dengan service ke-10, jika dilihat berdasarkan nama modulennya didalamnya. Berikut keterhubungannya yang dapat dilihat melalui diagram database (4.35).



Gambar 4.35 Diagram Database dari Calendar Event Type

Diagram database menunjukkan tidak ada hubungan dengan tabel yang terbentuk dari module product ataupun point of sale(POS), sehingga service ke-17 bisa berdiri sendiri. Dari diagram juga terlihat bahwa ke dua service ini memiliki hubungan yang sama dengan module lain seperti dengan ir.attachment, res.users, res.groups, res.company, dan ir.model.

BAB 5 KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan dan saran dari hasil pengujian yang telah dilakukan.

5.1 Kesimpulan

Berikut kesimpulan dan rangkuman dari penerapan *Microservice* dengan *Hierarchical Clustering* untuk dekomposisi monolitik pada *Enterprise Resource Planning* yang sudah dilakukan.

1. Berdasarkan Tabel 4.7, Tabel 4.8, dan Tabel 4.9 menunjukkan linkage yang cocok untuk membuat kelompok service yang memiliki rentang jumlah module yang kecil tetapi masih memiliki coupling dan cohesion yang baik adalah Average linkage.
2. Berdasarkan Gambar 4.23, Gambar 4.24, dan Gambar 4.25. Menunjukkan Hierarchical Clustering dapat membuat Microservice yang memiliki nilai coupling yang kecil dan nilai cohesion yang tinggi dengan jumlah partisi tertentu. Jumlah partisi yang ditemukan dapat membuat Microservice yang ideal dalam rentang jumlah service yang ideal adalah dari 175-245 service dengan Average Linkage.
3. Berdasarkan Gambar 4.34 dan Gambar 4.35 dapat dilihat dari struktur database yang terbentuk ketika dilakukan implementasi. Struktur tabel menunjukkan Hierarchical Clustering dapat memisahkan module yang tidak terhubung dan service yang memiliki hubungan kuat dikelompokan menjadi satu seperti pada Tabel 4.10 di partisi ke-10 pada Module Product dan Module Point of Sale. Kemudian module yang tidak memiliki hubungan dengan module lain seperti Module Calendar di partisi ke-17 tidak dikelompokan menjadi satu dengan Module Product atau Module Point of Sale, sehingga hasil dari proses Hierarchical Clustering relevan.

5.2 Saran

Saran untuk pengembangan dekomposisi aplikasi monolitik dengan *Hierarchical Clustering* menjadi *Microservice* adalah sebagai berikut:

1. Diperlukan penelitian lebih lanjut dengan aplikasi aplikasi *Enterprise Resource Planning* yang berbeda
2. Mempertimbangkan tipe *coupling* yang berbeda dalam menentukan suatu hubungan antara objek

BAB 5 KESIMPULAN DAN SARAN

3. Menggunakan linkage selain *single*, *complete*, dan *average*

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.

DAFTAR REFERENSI

- [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	Dolor sit amet	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

LAMPIRAN A LAMPIRAN A

Tabel A-1 *Lorem ipsum*

No	Dolor sit amet	At sonet	Vim commune	At quo congue	Cum iisque
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	Dolor sit amet	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