

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

TUGAS AKHIR

**Albertus Septian Angkuw
1119002**



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

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

TUGAS AKHIR

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

**Albertus Septian Angkuw
1119002**



**INSTITUT
TEKNOLOGI
HARAPAN
BANGSA**

Veritas vos liberabit

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

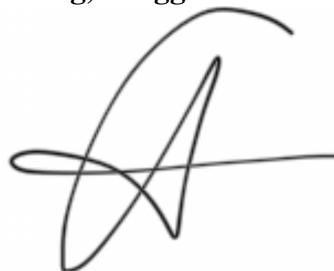
HALAMAN PERNYATAAN ORISINALITAS

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

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

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

Bandung, Tanggal Bulan Tahun

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

**Albertus Septian Angkuw
1119002**

HALAMAN PENGESAHAN TUGAS AKHIR

Tugas Akhir dengan judul:

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

yang disusun oleh:

**Albertus Septian Angkuw
1119002**

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

Hari / tanggal : Hari, Tanggal Bulan Tahun

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

Menyetujui

Pembimbing Utama:

Pembimbing Pendamping:

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

NIK

**...
NIK**

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

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

Nama : Nama Pengarang

NIM : NIM

Program Studi : Informatika

demikian 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:

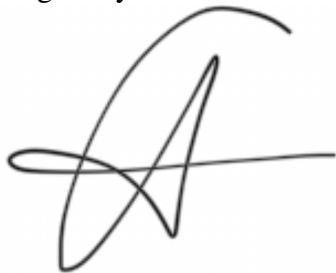
JUDUL TUGAS AKHIR

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

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

ABSTRACT

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

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

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

KATA PENGANTAR

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

Bandung, Tanggal Bulan Tahun

Hormat penulis,

A stylized, handwritten signature in black ink, consisting of a large, sweeping loop followed by a horizontal line and a small upward stroke.

Nama Pengarang

DAFTAR ISI

ABSTRAK	iv
ABSTRACT	v
KATA PENGANTAR	vi
DAFTAR ISI	vii
DAFTAR TABEL	x
DAFTAR GAMBAR	xi
DAFTAR ALGORITMA	xi
DAFTAR LAMPIRAN	xii
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 Monolit	2-1
2.1.2 <i>Microservice</i>	2-3
2.1.3 <i>Enterprise Resource Planning</i>	2-7
2.1.4 Analisis Kode	2-8
2.1.5 <i>Clustering</i>	2-11
2.1.6 Dekomposisi	2-14
2.1.7 Teknologi dan <i>Library</i>	2-20
2.1.7.1 <i>Docker</i>	2-20
2.1.7.2 <i>jsonRPC</i>	2-20
2.1.7.3 <i>PyCG</i>	2-20

2.1.7.4	<i>Kong</i>	2-20
2.1.7.5	<i>inspect</i>	2-20
2.1.7.6	<i>SciPY</i>	2-20
2.2	Tinjauan Studi	2-20
2.3	Tinjauan Objek	2-23
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 Call Graph	3-5
3.3.1.4	Ekstraksi Depedency Module	3-6
3.3.1.5	Pre Procesinng	3-7
3.3.1.6	Hierarchical Clustering	3-8
3.3.1.7	Pemilihan Partisi	3-9
3.3.2	Dekomposisi Monolitik ke Microservice	3-13
3.3.2.1	Pemisahan User Interface	3-13
3.3.2.2	Strategi Pemisahan Kode	3-15
3.3.2.3	Komunikasi antar service	3-17
3.3.2.4	Strategi Pemisahan database	3-17
3.3.3	Deployment	3-18
BAB 4	IMPLEMENTASI DAN PENGUJIAN	4-1
4.1	Lingkungan Implementasi	4-1
4.1.1	Spesifikasi Perangkat Keras	4-1
4.1.2	Spesifikasi Perangkat Lunak	4-2
4.2	Implementasi Perangkat Lunak	4-2
4.2.1	Implementasi <i>Class</i>	4-3
4.2.1.1	<i>Class</i> Nama_Class_1	4-3
4.2.1.2	<i>Class</i> Nama_Class_2	4-3
4.2.2	Implementasi Numquam	4-3
4.3	Implementasi Nama_Implementasi	4-3
4.4	Implementasi Aplikasi	4-3
4.5	Pengujian	4-4
4.5.1	Pengujian Nama_Pengujian_1	4-4

4.5.2	Pengujian Nama_Pengujian_2	4-6
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	<i>State of the Art</i>	.2-20
2.2	Komposisi dari Module pada aplikasi Odoo [16]:	.2-25
4.1	atribut pada <i>class</i> nama_class_1	4-4
4.2	Daftar <i>method</i> pada <i>class helper</i>	4-5
A-1	<i>Lorem ipsum</i>	A-1
A-1	<i>Lorem ipsum</i>	A-2
B-1	<i>Lorem ipsum</i>	B-3
B-1	<i>Lorem ipsum</i>	B-4

DAFTAR GAMBAR

2.1	Pola dalam menyelesaikan masalah di arsitektur <i>Microservice</i> [9]	2-5
2.2	Proses migrasi dari waktu ke waktu	2-17
2.3	Proses melakukan Strangle Fig	2-17
2.4	Ilustrasi Proses Branch By Abstraction	2-18
2.5	Arsitektur Odoo [16]	2-24
2.6	Skema Database Odoo	2-26
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	Class yang dimodifikasi pada PyCG	3-4
3.5	Kasus Monkey Patching di Odoo	3-4
3.6	Isi folder Odoo	3-5
3.7	Isi folder Addons	3-5
3.8	Ilustrasi JSON yang dihasilkan PyCG	3-6
3.9	Implementasi inspect untuk mengekstraksi object di addons Odoo	3-7
3.10	Menghilangkan Node diluar dari target	3-8
3.11	Jarak Jaccard	3-8
3.12	Jarak Struktural	3-8
3.13	Pembuatan Distance Matrix	3-8
3.14	Ilustrasi Heatmap	3-9
3.15	Proses Pengelompokan	3-9
3.16	Dendogram yang dihasilkan dari Clustering	3-10
3.17	Implementasi FOne	3-10
3.18	Implementasi InterCoup	3-11
3.19	Implementasi Coup	3-11
3.20	Implementasi InterCoh	3-12
3.21	Penerapan Pemilihan Jumlah Cluster terbaik	3-13
3.22	Nilai Coupling dan Cohesion setiap jumlah cluster	3-13
3.23	Arsitektur UI Monolit	3-14
3.24	Arsitektur UI di Microservice	3-14
3.25	Struktur Module dan Keterhubungannya di Aplikasi Monolit	3-15
3.26	Penerapan Pola Strangle dan Branch by Abstraction	3-16
3.27	State Diagram pada proses login	3-16
3.28	Sequence Diagram pada kasus pengambilan data Product	3-17
3.29	Diagram Deployment	3-18

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 umumnya dibangun dengan beberapa arsitektur seperti arsitektur *n-tier*, *web-based*, *service oriented*, dan *cloud*. Implementasi arsitektur ERP mempengaruhi aspek manajemen di perusahaan seperti biaya, kompleksitas perawatan dan cara penggunaan aplikasi [1].

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

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

Pada pengukuran jumlah kode *response* 200(Berhasil), MSA memiliki jumlah *response* tertinggi di kode berhasil dengan memiliki jumlah *response* terkecil di kode 302(Pengalihan), 304(Cache) ,408(Waktu Habis), 500(Kesalahan

Internal Server) dan tidak memiliki jumlah *response* di kode 404(Tidak ditemukan). Dimana pada aspek pemeliharaan aplikasi, MSA lebih unggul daripada SOA dan monolit unggul daripada SOA [4].

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

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

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

1.2 Rumusan Masalah

Berikut adalah rumusan masalah yang dibuat berdasarkan latar belakang diatas.

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

1.3 Tujuan Penelitian

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

1. Menerapkan dekomposisi aplikasi ERP monolit ke *microservice* dengan pendekatan *Hierarchical Clustering*.
2. Membuat *microservice* yang memiliki nilai kopel rendah dan nilai kohesi tinggi.
3. Membandingkan performa dan sumber daya aplikasi ERP monolit dengan *microservice*.

1.4 Batasan Masalah

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

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

1.5 Kontribusi Penelitian

Kontribusi yang diberikan pada penelitian ini adalah sebagai berikut.

1. Memberikan langkah dalam melakukan dekomposisi aplikasi monolit ke *microservice* dengan *Hierarchical Clustering*.
2. Mengetahui pengaruh dari performa aplikasi yang sudah dilakukan dekomposisi dengan uji beban pada aplikasi.
3. Membuat aplikasi *microservice* yang memiliki nilai kohesi tinggi dan nilai kopel rendah.

1.6 Metodologi Penelitian

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

1. Penelitian Pustaka

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

2. Analisis

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

3. Perancangan

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

4. Implementasi

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

5. Pengujian

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

1.7 Sistematika Pembahasan

BAB 1: PENDAHULUAN

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

BAB 2: LANDASAN TEORI

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

BAB 3: ANALISIS DAN PERANCANGAN

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

BAB 4: IMPLEMENTASI DAN PENGUJIAN

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

BAB 5: KESIMPULAN DAN SARAN

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

BAB 2 LANDASAN TEORI

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

2.1 Tinjauan Pustaka

Pembahasan mengenai teori-teori tersebut dijelaskan sebagai berikut.

2.1.1 Monolit

Monolit yaitu suatu cara untuk melakukan penyebaran. Ketika semua fungsi dalam sistem harus disebarakan secara bersama-sama, maka itu merupakan sebuah monolit [6]. Monolit merupakan sebuah aplikasi perangkat lunak dimana setiap modulnya tidak bisa dieksekusi secara independen. Hal ini membuat monolit sulit digunakan pada sistem terdistribusi tanpa bantuan penggunaan *frameworks* atau solusi *ad hoc* seperti Objek Jaringan, *RMI* atau *CORBA* [8].

Penggunaan pada bahasa pemrograman seperti *Java*, *C/C++*, dan *Python* pada pengembangan aplikasi di sisi *server*, memiliki kemampuan dalam melakukan abstraksi untuk memecah kompleksitas program menjadi berupa modul. Namun, bahasa pemrograman ini dirancang untuk membuat *artefacts* monolit. Dimana abstraksi ini tergantung pada penggunaan berbagi sumber data pada komputer yang sama (memori, database, file) [8].

Terdapat 3 jenis monolit [6]:

1. *Single Process Monolith*

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

2. *Distributed Monolith*

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

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

perubahan menyebabkan kerusakan pada bagian sistem lain.

3. *Sistem Black-Box Pihak Ketiga*

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

Keuntungan dari Monolit [12, 9]:

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

Tantangan dari monolit [12, 9]:

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

sisi sumber daya.

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

2.1.2 *Microservice*

Microservice adalah beberapa *service* yang bisa di deploy secara independen yang dimodelkan berdasarkan bisnis domain. *Service* ini berkomunikasi satu sama lain melalui jaringan komputer dan bisa dibangun dengan berbagai macam teknologi. *Microservice* adalah salah tipe dari *service oriented architecture (SOA)* meskipun ada perbedaan dalam membuat batasan antara *service* dan *deployment* secara independen [9].

Service adalah komponen perangkat lunak yang memiliki kegunaannya secara khusus dimana komponen ini bisa berdiri sendiri dan secara independen dilakukan proses deployment. *Service* memiliki *API (Application Programming Interface)* yang memberikan akses kepada *client* untuk melakukan operasi. Terdapat 2 tipe operasi yaitu perintah dan kueri. *API* terdiri dari perintah, kueri dan *event*. Perintah dapat berupa *buatPesanan()* yang melakukan aksi dan memperbarui data. Kueri dapat berupa *cariPesananBerdasarkanID()* yang digunakan untuk mengambil data. *Service* juga dapat membuat suatu *event* seperti *PesananSudahDibuat* dimana *event* ini akan dikonsumsi oleh *client*-nya / *subscriber* [9].

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

Ciri Khusus *Microservice* [12, 7, 9]:

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

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

2. Otonomi / Bisa berdiri sendiri

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

3. Data yang dikelola masing-masing *service*

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

Keuntungan dari *Microservice* [12, 7, 9]:

1. Memudahkan pengembangan aplikasi kompleks dan flexibel

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

2. Bisa dilakukan skaling secara independen

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

3. Mudah melakukan percobaan dan penggunaan teknologi baru

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

Tantangan dari *Microservice* [12, 7, 9]:

1. Menemukan *service* yang tepat itu sulit

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

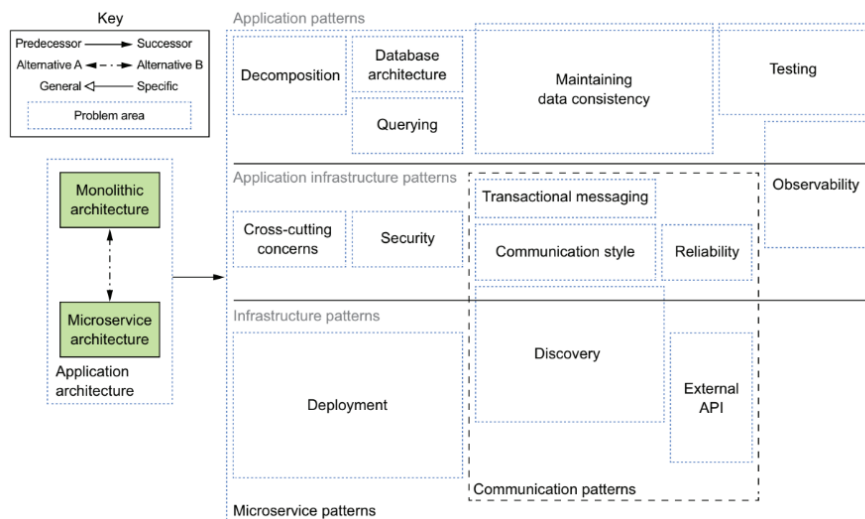
2. Memiliki kompleksitas karena merupakan suatu terdistribusi

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

3. *Deployment* yang melibatkan beberapa *service*

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

Pola Arsitektur *Microservice* [9]:



Gambar 2.1 Pola dalam menyelesaikan masalah di arsitektur *Microservice* [9]

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 kueri harus menggabungkan data yang dimiliki oleh beberapa service yang terlibat karena data hanya bisa diakses melalui API. Terkadang bisa digunakan komposisi API dimana 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 test dilakukan pada client/konsumen, test pencocokan pada client/konsume, dan pengujian secara terisolasi.

2. *Application infrastructure*

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

Pola komunikasi dikelompokkan menjadi 5 grup yaitu gaya komunikasi, discovery, reliabilitas, Transactional Messaging, API Eksternal. Terdapat 3 gaya komunikasi yaitu Messaging dimana komunikasi dapat dilakukan secara asynchronous, Domain-Specific dimana komunikasi harus melalui protokol tertentu seperti Email, dan Remote Procedure Invocation dimana 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 dimana menggunakan tabel OUTBOX untuk menyimpan message sementara dan Log Tailing dimana melihat

transaksi terakhir dari message.

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

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

3. *Infrastructure patterns*

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

Eksternal API adalah bagaimana aplikasi pengguna berinteraksi dengan service. Ada 2 cara untuk aplikasi berinteraksi yaitu API Gateway dan Backend for Frontend. Proses Deployment microservice memiliki proses yang kompleks karena service yang dikelola bisa berjumlah 10 hingga ratusan service, sehingga diperlukan proses otomatisasi yang bisa mengelola service dan proses skaling 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 dimana 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].

Arsitektur ERP [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 kedalam masing lapisan seperti lapisan *user interface*, lapisan aplikasi dan lapisan *database / penyimpanan data*.

2. *Web-based*

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

3. *Service Oriented*

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

4. *Cloud*

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

2.1.4 Analisis Kode

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

dapat memberikan informasi kepada pengembang mengenai program yang dianalisis [10].

Anatomi Analisis Kode [10]:

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 mengkonversi urutan karakter menjadi suatu kata-kata dan mengekstraksi nilai semantik sebenarnya. Tujuannya agar memudahkan proses analisis/transformasi dan penambahan data lainnya.

2. Representasi Informasi

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

3. Eksplorasi Pengetahuan

Setelah informasi direpresentasikan, informasi dibuat menjadi suatu kesimpulan. Kesimpulan bisa dibuat secara kuantitatif atau kualitatif, proses visualisasi penting dalam proses eksplorasi pengetahuan kode program.

Strategi Analisis Kode [10]:

1. Statik vs Dinamis

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

2. *Sound vs Unsound*

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

3. *Flow sensitive vs Flow insensitive*

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

4. *Context sensitive vs Context insensitive*

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

Tantangan Kode Analisis [10]:

1. Perbedaan bahasa kode program

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

2. *Multi-Language*

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

3. Analisis secara *Real-Time*

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

2.1.5 Clustering

Clustering yaitu suatu proses untuk melakukan pengelompok atau klasifikasi objek. Objek bisa ditentukan dari pengukuran atau berdasarkan hubungan antar objek lainnya. Tujuan dari clustering yaitu untuk menemukan struktur data yang valid. Cluster terdiri dari sejumlah object serupa yang dikumpulkan / dikelompokan bersama [11].

Metode Clustering membutuhkan adanya indeks kedekatan diantara object. indeks ini bisa dikomputasi dalam bentuk matrix. Hasil matrix kedekatan / proximity 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[11].

Berikut adalah metode untuk mencari indeks kedekatan pada object:

1. Jaccard Coefficient [11]

Untuk mendapatkan kedekatan dengan menghitung total hal yang sama atau terhubung diantara object kemudian dibagi dengan jumlah hal yang berbeda diantara dua objek 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 [12]

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

$$Sim_{str}(c_i, c_j) = \begin{cases} \frac{1}{2} \left(\frac{calls(c_i, c_j)}{calls(c_i)} + \frac{calls(c_j)}{calls(c_i)} + \frac{calls(c_i)}{callsin(c_i)} \right) & \text{if } callsin(c_j) \neq 0 \text{ and } calls(c_i) \neq 0 \\ \frac{calls(c_i)}{calls(c_i)} & \text{if } callsin(c_i) \\ \frac{calls(c_i)}{callsin(c_j)} & \text{if } calls(c_i) \neq 0, \end{cases}$$

3. Simple matching coefficient [11]

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.

$$d(i,k) = \frac{a_{00} + a_{11}}{a_{un} + a_{11} + a_{01} + a_{10}} = \frac{a_{00} + a_{11}}{d} \quad (2.2)$$

Metode *Clustering* yang umumnya digunakan [11]:

1. *Hierarchical Clustering*

Metode *Hierarchical Clustering* adalah sebuah prosedur untuk mentransformasi sebuah *proximity matrix* menjadi beberapa partisi. Clustering adalah sebuah partisi dimana komponen dari partisi disebut *clusters*. Beberapa partisi memiliki suatu urutan dan tingkatan berdasarkan bagaimana partisi tersebut disatukan. Terdapat 2 pendekatan algoritma dalam membentuk suatu partisi yaitu secara *agglomerative* dan *divisive*. [11]

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 [11].

Ada beberapa metode untuk menentukan partisi terdekat yaitu dengan menghitung jarak maksimal antara objek partisi (complete linkage), nilai rata-rata jarak (average linkage) atau jarak minimum (single linkage) [15].

2. *Partitional Clustering*

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

Dimana satu partisi lebih penting, Metode *Partitional* juga memiliki efisiensi dan kompresi yang cocok untuk data yang besar sehingga pola dalam cluster memiliki kemiripan satu sama lain daripada pola dalam

cluster yang berbeda. Pemilihan nilai *cluster* bisa ditentukan secara opsional, kriteria *cluster* yang valid harus ditentukan seperti *square-error* untuk menentukan apakah partisi yang dibuat optimal. Kriterianya sendiri bisa dibagi menjadi secara global atau lokal.

Pemilihan Partisi:

1. *Structural and Behavioral Dependencies* [14]

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 kopel. Kolaborasi bisa ditentukan dengan nilai kohesi dari jumlah data seperti atribut yang tidak tetap.

$$FStructBeh(M) = \frac{1}{n}(\alpha FOne(M) - \beta FAutonomy(M)) \quad (2.3)$$

Nilai Internal Coupling dapat dihitung dari jumlah koneksi secara langsung atau tidak langsung melalui ketergantungan di antara class. 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 kopling antara dua class dihitung oleh fungsi CoupP, fungsi CoupP membagi jumlah call yang terjadi antara 2 class dengan total call yang dilakukan class tersebut.

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

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

$$ExterCoup(M) = \frac{\sum CoupP(P) - \sum_{PV \in PairsV} \sigma(PV)}{NbPossibleExternalPairs}. \quad (2.6)$$

Untuk menghitung nilai kohesi dapat dilakukan dengan menghitung jumlah interaksi class didalam partisi. Perhitungan ini dapat dihitung dengan fungsi

InterCoh. InterCoh membagi antara jumlah call yang terjadi didalam class dengan jumlah call yang hanya memanggil class didalam partisinya sendiri.

$$InterCoh(M) = \frac{NbDirectConnections}{NbPossibleConnections} \quad (2.7)$$

2. *Data Autonomy Class* [14]

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

3. *Iter-Paritition Call Percentage(ICP)* [12]

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

$$icp_{i,j} = c_{i,j} / \sum_{i=1, j=1, i \neq j}^M \quad (2.8)$$

4. *Jumlah Interface* [12]

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

$$\frac{1}{N} \sum_{i=1}^N ifn_i \quad (2.9)$$

2.1.6 Dekomposisi

Terdapat beberapa pendekatan dalam menentukan bagian aplikasi untuk menjadi Service:

1. *Kemampuan Bisnis*[9]

Service dibuat dari pendekatan proses arsitektur bisnis dimana 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) [9]

DDD mengambil konsep mencari domain dimana domain tersebut dapat digunakan untuk menyelesaikan permasalahan didalam domain itu sendiri. Model domain hampir mencerminkan antara design 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 [6, 14]

Transformasi aplikasi monolit menjadi microservice bisa dilakukan dengan strategi analisis statik atau dinamis. Umumnya hal yang diperhatikan adalah ketergantungan / keterhubungan antar module, kemudian menerapkan proses clustering atau algoritma evolusi pada ketergantungan module untuk menghasilkan partisi-partisi module yang sudah ditentukan dari evaluasi tertentu seperti nilai kopel yang rendah dan nilai kohesi 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 kopel dan kohesi. Kopel(*Coupling*) berfokus pada bagaimana perubahan pada satu hal membuat bagian lain mengalami perubahan. Kohesi(*Cohesion*) berfokus pada bagaimana kode dikelompokkan satu

sama lain [6].

Struktur Microservice yang ideal memiliki kohesi yang tinggi dan kopel yang rendah, karena dengan kohesi yang tinggi setiap service memiliki fokusnya masing-masing dan perubahan dilakukan pada spesifik service sedangkan dengan kopel rendah membuat service bisa berdiri sendiri / independen dan setiap service mungkin tidak harus sering berinteraksi satu sama lain [6].

1. Kopel [6]

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 kopel seperti *Logical Coupling*, *Temporal Coupling*, *Deployment Coupling*, dan *Domain Coupling*.

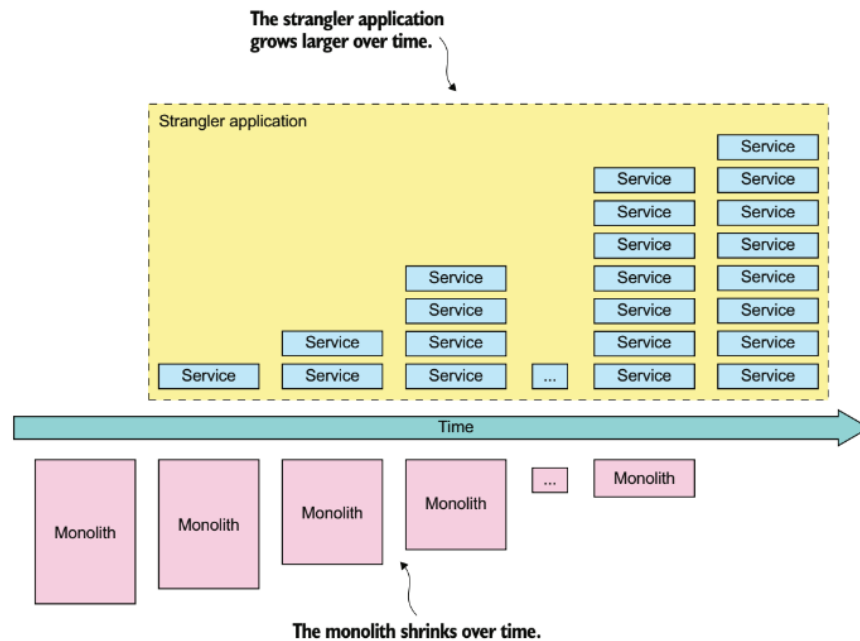
2. Kohesi [6]

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.

Pola untuk Proses Dekomposisi [6]:

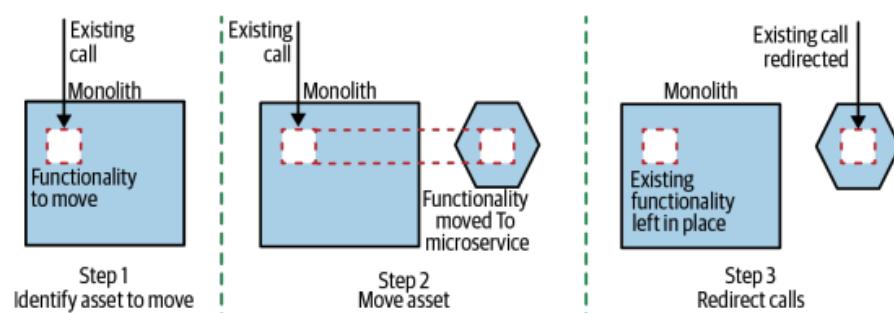
1. Pola Strangle Fig

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 2.2 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 2.3 Proses melakukan Strangle Fig

2. Pola UI Composition

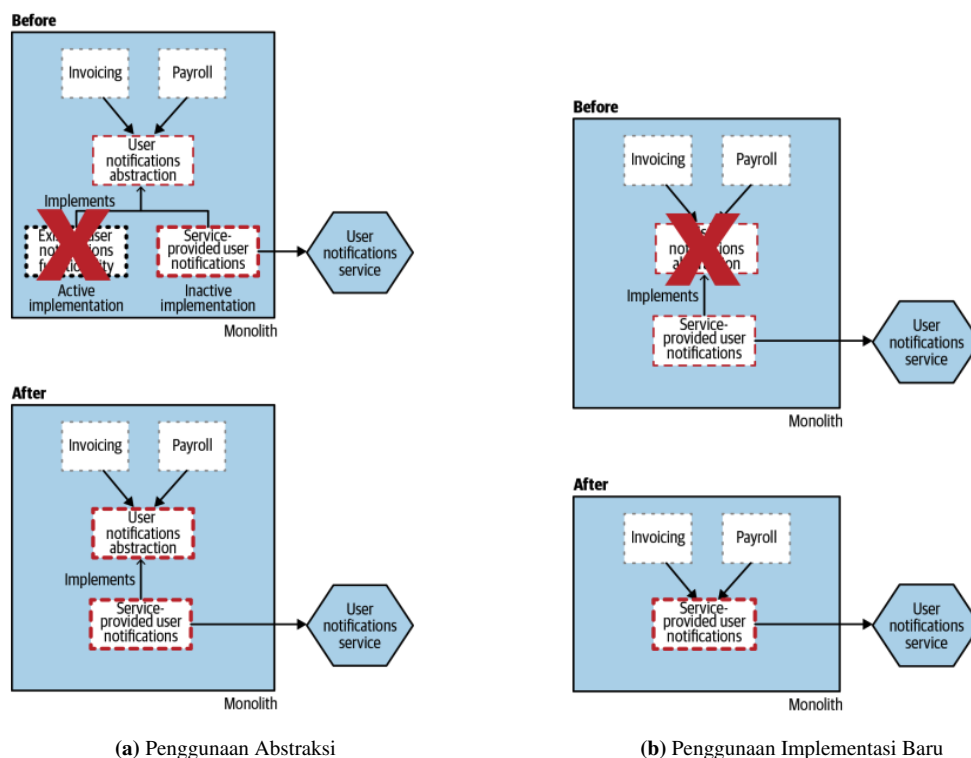
Pola ini digunakan pada User Interface(UI), dimana pemecahan dilakukan pada sisi tampilan aplikasi(UI). User Interface akan memanggil beberapa service yang dibutuhkannya. Terdapat beberapa pendekatan dalam pemecahan disisi UI yaitu Page Composition, Widget Composition, dan

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 didalam sistem yang lebih dalam. Maka proses ekstraksi harus dilakukan tanpa mempengaruhi banyak sistem dimana 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.



Gambar 2.4 Ilustrasi Proses Branch By Abstraction

4. Parallel Run

Parallel Run adalah pola bagaimana mengeksekusi sistem yang baru dan sistem yang lama ketika dipisahkan. Teknik ini dapat membuat 2 fungsionalitas diantara 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 resiko tinggi.

5. Decorating Collaborator

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

6. Change Data Capture

Change Data Capture tidak menangkap panggilan dan bertindak pada panggilan yang menuju ke monolit 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.

Tantangan dan Hambatan Dekomposisi [9]:

1. Latensi Jaringan

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

2. Menjaga konsistensi data antar service

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

3. Adanya God Class yang mencegah dekomposisi

God Class adalah class yang berukuran besar dan berdampak secara luas

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

2.1.7 Teknologi dan *Library*

2.1.7.1 *Docker*

...

2.1.7.2 *jsonRPC*

...

2.1.7.3 *PyCG*

...

2.1.7.4 *Kong*

...

2.1.7.5 *inspect*

...

2.1.7.6 *SciPY*

...

2.2 Tinjauan Studi

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

Tabel 2.1 *State of the Art*

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

<p>A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony (2020). "From monolithic architecture style to Microservice one based on a semi-automatic approach" [14]</p>	<p>Terdapat aplikasi monolit yang tidak beradaptasi di <i>cloud</i> ataupun <i>DevOps</i> sehingga harus dimigrasi ke microservice</p>	<p>Analisis kode program dan mencari hubungan dalam class-nya</p>	<p>Identifikasi Microservice yang dibuat memiliki hasil yang memuaskan karena mempertimbangkan karakteristik microservice</p>
<p>Chaitanya K. Rudrabhatla. (2020). "Impacts of Decomposition Techniques on Performance and Latency of Microservices." [3]</p>	<p>Bagaimana dampak perfoma dalam menentukan batasan antar service</p>	<p>Melakukan perbandingan antara pendekatan DDD, Normalized Entity Relationship, dan Hybrid</p>	<p>Teknik DDD lebih baik dalam dekomposisi namun teknik Hybrid dengan mempertimbangkan fungsionalis dan transaksi yang terjadi memiliki performa lebih baik.</p>
<p>A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee (2021). "Mono2micro: A practical and effective tool for decomposing monolithic Java applications to microservices" [13]</p>	<p>Bagaimana pendekatan Mono2Micro dengan cara dekomposisi lainnya dan bagaimanan tanggapan praktisi</p>	<p>Menggunakan hierarchical spatio-temporal decomposition yang menyimpan kondisi program secara dinamik berdasarkan eksekusi proses bisnis</p>	<p>Hasil rekomendasi microservice dapat membantu penerapan pola strangle , partisi yang dihasilkan sesuai dengan fungsi bisnis</p>

Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni. (2022). "A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications" [12]	Bagaimana mengotomatisasi proses migrasi aplikasi monolith ke microservice?	Menggunakan DBSCAN(Density-based Clustering) yang menghasilkan rekomentasi microservice	Berhasil membuat microservice yang lebih kohesive dan memiliki interaksi yang lebih sedikit.
---	---	---	--

Pada penelitian yang dilakukan oleh A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, dan C. Dony [14], 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 design(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 [13], peneliti menggunakan Mono2Micro untuk melakukan dekomposisi aplikasi monolit Java menjadi microservice. Kemudian menggunakan metrik untuk mengevaluasi apakah microservice yang

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

Hasil dari penelitian menunjukkan bahwa penggunaan Mono2Micro efektif dalam melakukan dekomposisi dan dapat memberikan benefit bagi pengembang karena bisa membantu pengembang untuk melihat partisi lainnya. Metrik yang dihasilkan dari Mono2Micro memiliki hasil yang bagus diantara 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 [12]. 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 monolit dengan analisis statik. Microservice yang didekomposisi memiliki nilai kohesi yang lebih baik di dalam microservice dan jumlah interaksi antar microservice lebih sedikit.

2.3 Tinjauan Objek

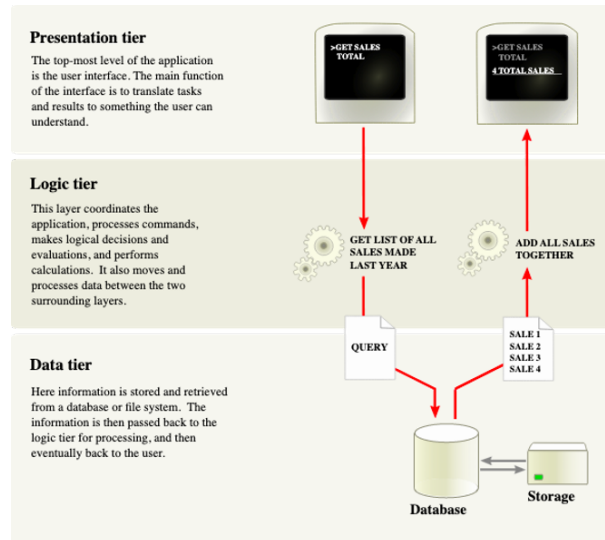
Pada bagian ini akan dijelaskan mengenai objek dan aplikasi terkait yang akan digunakan dalam tugas akhir ini. Object yang digunakan adalah sebuah aplikasi Enterprise Resource Planning yang di deploy secara monolit, yaitu Odoo.

Odoo merupakan aplikasi bisnis open source yang dapat mencakup semua kebutuhan perusahaan seperti CRM(Customer Relationship Management), eCommerce, akuntansi, inventaris, POS(Point of Sales), manajemen proyek dan lainnya. Aplikasi ini flexibel karena bisa dikembangkan lebih lanjut bila diperlukan dan bisa diubah karena memiliki lisensi source code yang terbuka [16].

Sebelum Odoo terdapat OpenERP, dimana OpenERP memiliki arsitektur monolit. 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 dimana tampilan, aturan bisnis dan tempat penyimpanan data memiliki lapisan terpisah. Dengan tujuan memudahkan dan mempercepat pengembang untuk melakukan modifikasi aplikasi tanpa harus mengganggu lapisan lainnya [16].



Gambar 2.5 Arsitektur Odoo [16]

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 [16].

Odoo memiliki struktur kode yang dibentuk sebagai module untuk setiap fiturnya. Sehingga dari sisi server dan client memiliki hubungan yang disatukan menjadi satu paket tersendiri. Dimana module adalah koleksi dari fungsi dan data untuk menyelesaikan satu tujuan. Modul pada Odoo bisa ditambahkan, diganti, diubah untuk menyesuaikan kebutuhan bisnis. Dimana pada pengguna module dilambangkan dengan nama Apps, tetapi tidak semua module adalah Apps. Modules juga bisa direfrensikan sebagai addons [16].

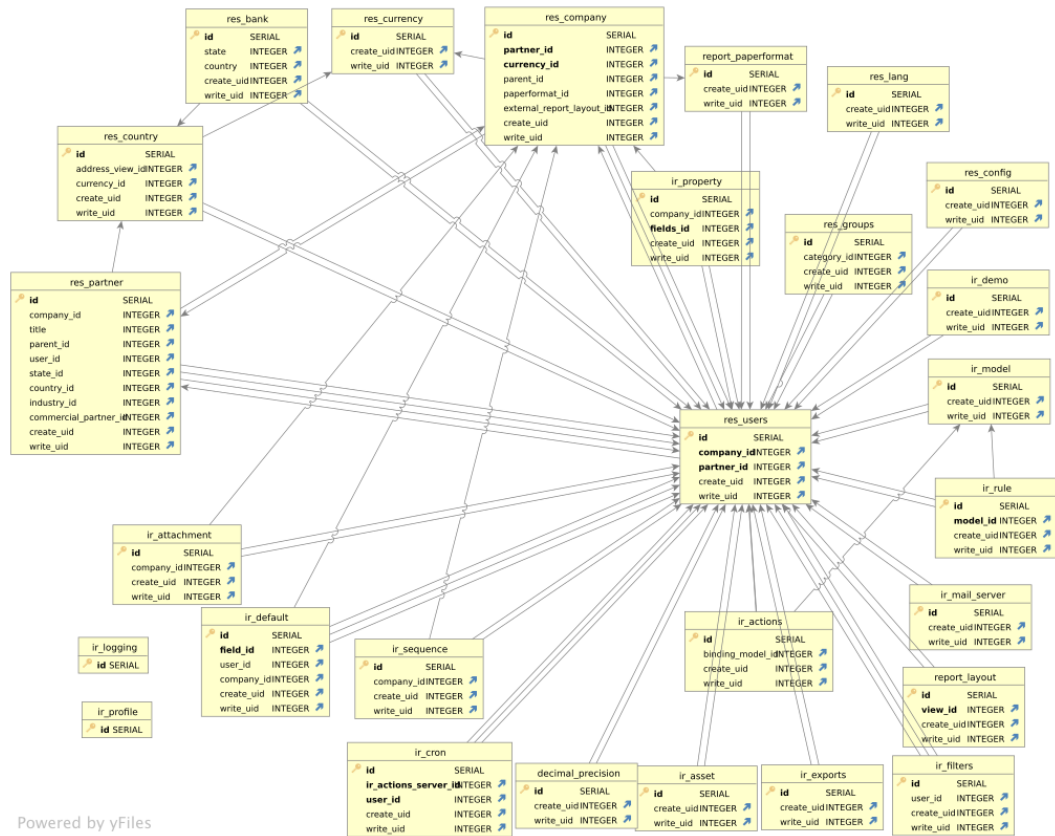
Tabel 2.2 Komposisi dari Module pada aplikasi Odoo [16]:

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

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

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

BAB 2 LANDASAN TEORI



Powered by yFiles

Gambar 2.6 Skema Database Odoo

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

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

3.1 Analisis Masalah

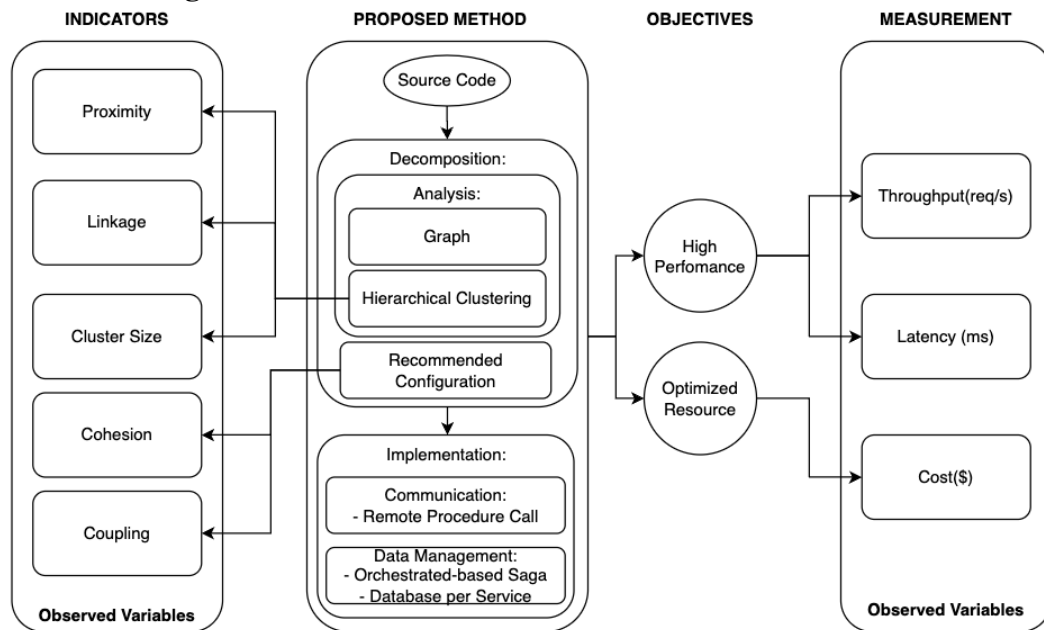
Arsitektur ERP yang digunakan pada Odoo yaitu arsitektur Service Oriented Architecture (SOA) dan masih dideploy secara monolit, seperti yang sudah dijelaskan pada landasan teori. Arsitektur monolit memiliki kelemahan dan permasalahan yang bisa diselesaikan dengan arsitektur microservice. Aplikasi ERP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

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

Hasil terbaik dari clustering diimplementasikan menjadi service, penelitian ini akan menggunakan strategi dengan pola strangle untuk memecah kode di monolit. Untuk dekomposisi pada data akan menggunakan strategi Aggregate Exposing Monolith. Agar data antar service tetap terintegrasi maka dilakukan penerapan SAGA dengan pendekatan Orchestrated .

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

3.2 Kerangka Pemikiran



Gambar 3.1 Kerangka Pemikiran

Penelitian akan dimulai dengan menggunakan kode sumber aplikasi yang dibuat dengan monolit. 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 graph yang menunjukkan bagaimana keterhubungan masing-masing objek di aplikasi.

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

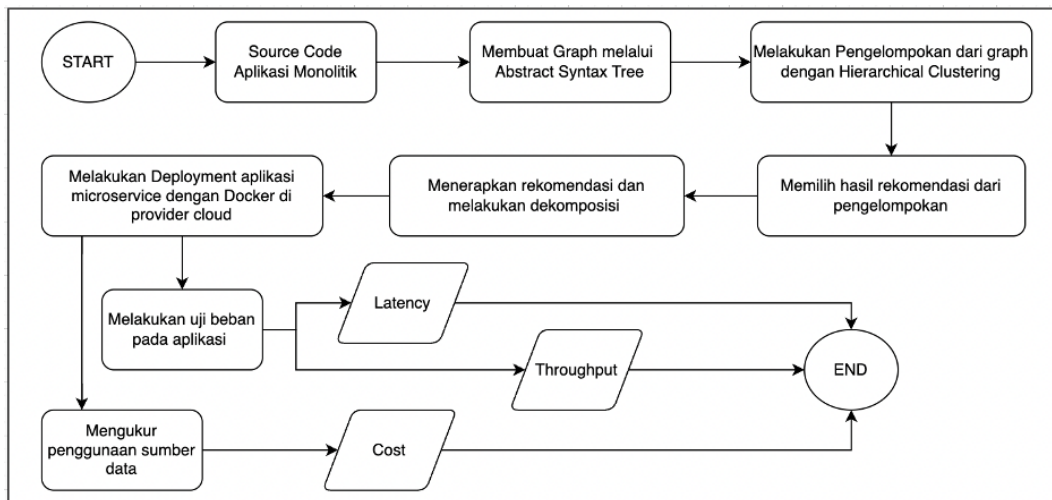
Pengelompokan dari Hierarchical Clustering akan dipilih dengan mencari nilai cohesion terendah dan nilai coupling tertinggi. Dimana Internal Coupling mengevaluasi tingkat ketergantungan langsung dan tidak langsung antar objek. Semakin banyak dua objek menggunakan metode masing-masing semakin mereka menjadi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

Ketika analisis dekomposisi sudah selesai dilakukan maka akan dilakukan implementasi berdasarkan pengelompokannya masing-masing yang akan menjadi

service. Untuk metode komunikasinya antara service yaitu dengan Remote Procedure Call(RPC) dan untuk mengelola data, setiap service memiliki databasenya masing-masing dan untuk menjaga konsistensi data antar service maka digunakan pendekatan SAGA dengan cara choreography.

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

3.3 Urutan Proses Global

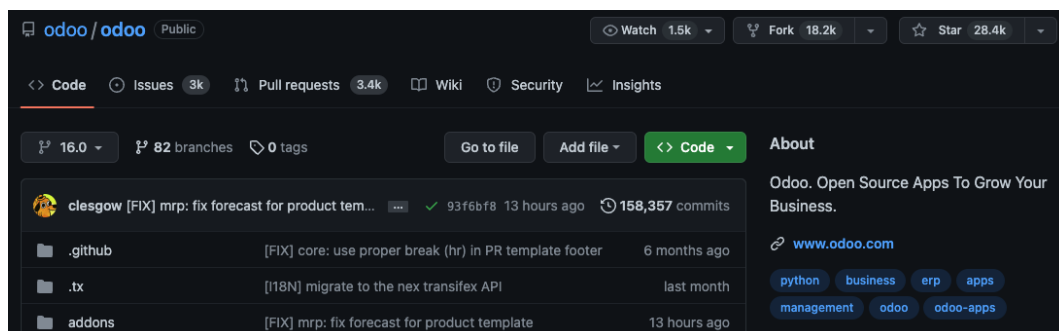


Gambar 3.2 Diagram Flowchart Proses Global

3.3.1 Proses Clustering

3.3.1.1 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. Adapun beberapa modifikasi perlu dilakukan pada tools PyCG yaitu pada module classes di class ClassNode dengan tujuan mencegah terjadinya pengulangan tak terbatas pada kasus *Monkey Patching*.

```
class ClassNode:
    def __init__(self, ns, module):
        self.ns = ns
        self.module = module
        self.mro = [ns]

    def add_parent(self, parent):
        if isinstance(parent, str):
            self.mro.append(parent)

        elif isinstance(parent, list):
            for item in parent:
                if self.mro.count(item) <= 2:
                    self.mro.append(item)
            else:
                print("Skipped Add Parent !!:" , item)

        self.fix_mro()
```

Gambar 3.4 Class yang dimodifikasi pada PyCG

```
odoo-16 > odoo > tools > pdf.py > Jupyter > merge_pdf
62 DictionaryObject.get - _unwrapping_get
63
64
65 class BrandedFileWriter(PdfFileWriter):
66     def __init__(self):
67         super().__init__()
68         self.addMetadata({
69             '/Creator': "Odoo",
70             '/Producer': "Odoo",
71         })
72
73
74 PdfFileWriter = BrandedFileWriter
75
```

Gambar 3.5 Kasus Monkey Patching di Odoo

Terdapat 2 target folder yang dibuat call graph yaitu folder odoo dan folder addons karena folder lainnya tidak memiliki hubungan mengenai proses bisnis. Untuk menghemat waktu pembuatan call graph pada folder test tidak diikuti. Entry point untuk tools PyCG adalah semua file di target folder dengan ekstensi file .py serta ditentukan package yang ingin diolah menjadi call graph. Proses eksekusi dilakukan melalui terminal. Call graph yang dihasilkan berisi call yang berasal

dari file .py yang ditentukan sebelumnya dan semua module yang terhubung dari target. Semua module ini bisa diluar dari target module apabila keterhubungan itu terus berlanjut. PyCG hanya bisa menghasilkan call graph tanpa informasi mengenai jumlah pemanggilan dan urutan pemanggilan.

```
perancangan-dekomposisi/odoo16$ ls odoo
addons          import_xml.rng  __pycache__
api.py          __init__.py     release.py
cli            loglevels.py    service
conf          models.py       sql_db.py
exceptions.py  modules         tests
fields.py     netsvc.py       tools
http.py       osv            upgrade
```

Gambar 3.6 Isi folder Odoo

```
perancangan-dekomposisi/odoo16$ ls addons
account
account_check_printing
account_debit_note
account_edi
account_edi_proxy_client
account_edi_ubl_cii
account_fleet
account_lock
account_payment
account_payment_invoice_online_payment
```

Gambar 3.7 Isi folder Addons

3.3.1.3 Ekstraksi Call Graph

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

```

"addons.base.models.res_company.Company._default_currency_id": [],
"addons.base.models.res_company.Company._get_default_favicon": [
    "odoo.tools.file_open",
    "random.randrange",
    "odoo.modules.module.get_resource_path",
    "io.BytesIO",
    "PIL.Image.open",
    "PIL.Image.new",
    "<builtins>.range",
    "base64.b64encode"
],
"random.randrange": [],
"PIL.Image.new": [],
"addons.base.models.res_company.Company": [
    "odoo.api.constrains",
    "odoo.fields.Many2many",
    "odoo.fields.Selection",
    "odoo.api.model_create_multi",
    "odoo.fields.Binary",
    "odoo.api.model",
    "odoo.fields.Boolean",
    "odoo.fields.Html",
    "odoo.api.onchange",
    "odoo.api.returns",
    "odoo.fields.Integer",
    "odoo.fields.Char",
    "odoo.api.depends",
    "odoo.fields.Many2one",
    "odoo.fields.One2many"
],
"addons.base.models.res_company.Company.<lambda1>": [],
"addons.base.models.res_company.Company.<lambda2>": [],
"addons.base.models.res_company.Company.init": [
    "odoo.models.Model.env.ref",
    "odoo.models.Model.search",
    "<builtins>.super",
    "<builtins>.hasattr"
],
"addons.base.models.res_company.Company._get_company_address_field_names"
```

Gambar 3.8 Ilustrasi JSON yang dihasilkan PyCG

3.3.1.4 Ekstraksi Dependency Module

Keterbatasannya informasi call graph yang dihasilkan dari PyCG, sehingga tugas akhir ini menggunakan library python yaitu 'inspect' untuk menganalisis object secara run-time. Hal ini disebabkan python adalah bahasa pemrograman dynamic dimana pengecekan tipe data dilakukan secara 'run-time'. Ekstrasi ini difokuskan pada module yang memiliki proses bisnis baik seperti module addons dan odoo/addons.

```
import inspect, importlib.util
def scanModule(modulePath):
    runcommand = importlib.import_module(modulePath)
    listClass = {}
    for name, obj in inspect.getmembers(runcommand):
        if inspect.ismodule(obj):
            member = inspect.getmembers(obj)
            tmpClass = {}
            for item in member:
                if inspect.isclass(item[1]):
                    print(item[1])
                    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'):
                        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 3.9 Implementasi inspect untuk mengekstraksi object di addons Odoo

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

3.3.1.5 Pre Procesinnng

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

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

```

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
        childFilter[v] = 1
    if len(childFilter) == 0:
        edgeGraph.append(key)
        continue
    for c in childFilter:
        if c not in callGraphFiltered:
            callGraphFiltered[c] = {}
        callGraphFiltered[key] = childFilter

print(f'Total Top Node: {len(callGraphFiltered)} ')
print(f'Total Edge: {len(edgeGraph)} <{edgeGraph[:3]}>')
print(f'Total OutsideEdge: {len(outsideCall)} <{list(outsideCall)[:3]}>')

Total Top Node: 17282
Total Edge: 15457 <['odoo.addons.test_inherits_depends.__manifest__', 'odoo.addons.test_inherits_depends.models', 'odoo.fields.Char']>
Total OutsideEdge: 157 <['slugify', 'ctypes', 'json']>

```

Gambar 3.10 Menghilangkan Node diluar dari target

3.3.1.6 Hierarchical Clustering

Graph yang berbentuk Adjacency list diubah menjadi adjacency matrix, proses normalisasi data dilakukan pada matrix. Yang kemudian matrix tersebut dibuat menjadi Distance Matrix, rumus jarak yang digunakan ada 3 yaitu Jaccard, Hamming dan Struktural. Masing-masing hasil jarak disatukan dengan rata-rata.

```

def simJaccard(im1, im2):
    im1 = np.asarray(im1, bool)
    im2 = np.asarray(im2, bool)

    intersection = np.logical_and(im1, im2)
    union = np.logical_or(im1, im2)
    return (intersection.sum() / float(union.sum()))

```

Gambar 3.11 Jarak Jaccard

```

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 3.12 Jarak Struktural

```

distanceMatrix = [[0.00 for i in range(data.shape[1])] for j in range(data.shape[0])]

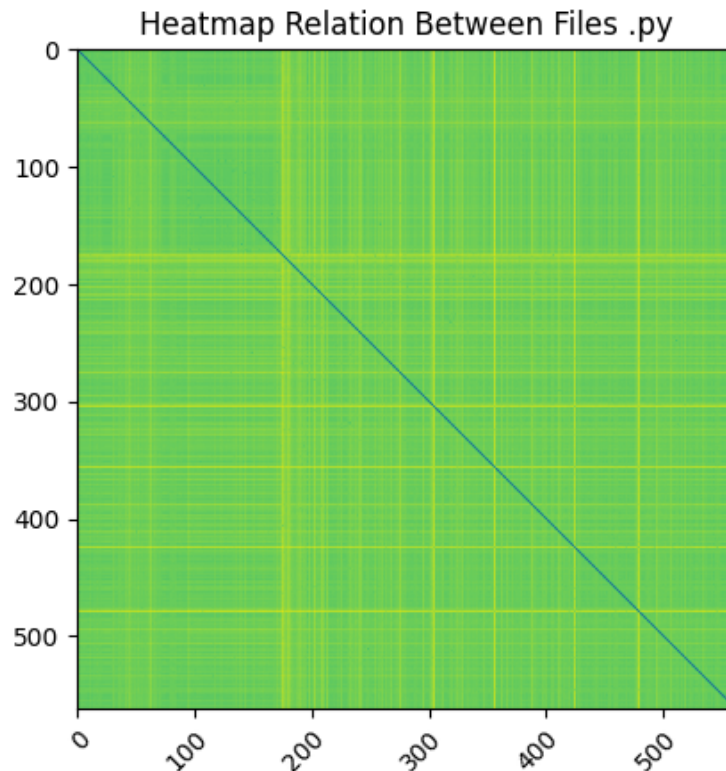
np.fill_diagonal(data, 1)
for i in range(0, len(data)):
    for j in range(0, i):
        distanceStructual = round(1- simStr(data[i, :], data[j, :], i, j, sum(data[:, i]), sum(data[:, j])), 2)
        distanceJaccard = round(1- simJaccard(data[i, :], data[j, :]), 2)
        distanceHamming = round(distance.hamming(data[i, :], data[j, :]), 2)
        distanceMatrix[i][j] = (distanceStructual + distanceJaccard + distanceHamming) / 3

    distanceMatrix[j][i] = distanceMatrix[i][j]

```

Gambar 3.13 Pembuatan Distance Matrix

Distance Matrix dapat dilihat nilai dengan ilustrasi Heatmap, dimana sumbu x dan y adalah semua module dan nilai kedekatannya dengan module lainnya. Pada tugas akhir ini menggunakan library SciPy yang memiliki fungsi Hierarchical Clustering dengan algoritma linkage. Hasil pengelompokan dapat ditampilkan dalam bentuk dendrogram.



Gambar 3.14 Ilustrasi Heatmap

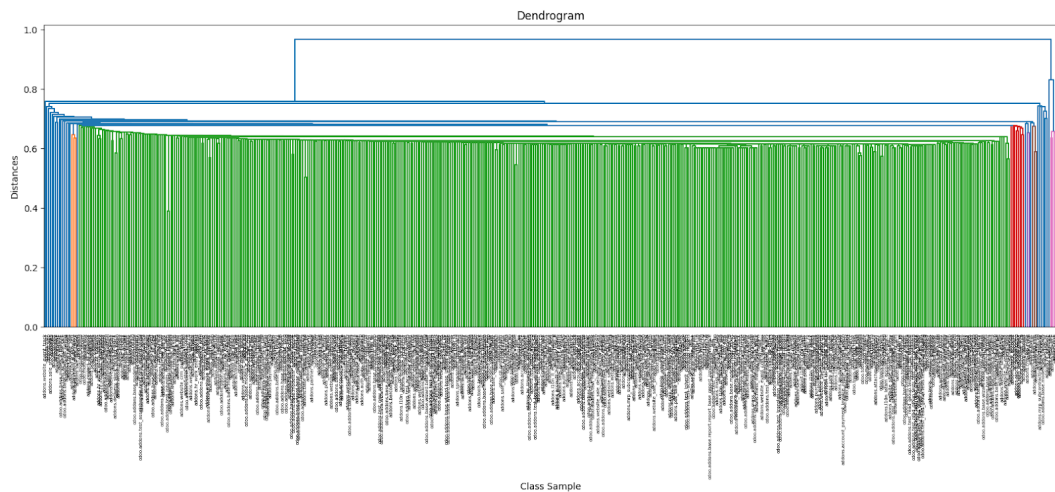
```
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import linkage
tmpz = linkage(distanceMatrix)
plt.figure(figsize=(20,6))
dend = sch.dendrogram(tmpz,labels=listLabel)
plt.title('Dendrogram')
plt.xlabel('Class Sample')
plt.ylabel('Distances')
plt.show()
```

Gambar 3.15 Proses Pengelompokan

3.3.1.7 Pemilihan Partisi

Pemilihan jumlah cluster/partisi perlu dilakukan dengan perhitungan yang dapat menentukan jumlah service yang ideal. Microservice yang ideal memiliki nilai coupling yang rendah dan nilai kohesi yang tinggi. Untuk itu tugas akhir ini

menentukan partisi secara nilai struktural.



Gambar 3.16 Dendrogram yang dihasilkan dari Clustering

Terdapat implementasi rumus yang digunakan FOne, InterCoup, Inter Coh dan NBCalls. Nilai FOne yang menentukan apakah partisi dengan jumlah cluster tertentu memiliki nilai coupling dan cohesion yang ideal. Terdapat modifikasi rumus pada FOne yaitu adanya pembagian jumlah cluster pada nilai cohesion agar microservice yang memiliki banyak service lebih baik dari pada service yang sedikit sehingga microservice yang dibentuk bisa independen.

```
def FOne(m,rm):  
    interCoup = InterCoup(m)  
    interCoh = InterCoh(m,rm)  
    return {  
        "coupling": interCoup,  
        "cohesion": interCoh,  
        "value": (interCoup+(interCoh/len(m)))/2,  
        "clusterSize": len(m)  
    }
```

Gambar 3.17 Implementasi FOne

Pada InterCoup yaitu menghitung coupling antara partisi. Perhitungan call antar partisi dihasilkan dari nama call, nama call sebelumnya yang mungkin mengarah ke nama call dimana nama call tersebut sudah menjadi bagian besar dari partisi lain. Maka nama call tersebut diganti menjadi nama call partisi itu, jumlah call yang terjadi tetap disimpan dan dibuat adjacency matrix baru untuk menghitung call dengan bantuan fungsi CoupP.


```
def InterCoup(m):  
    sumCoup = 0  
    NbPossiblePairs = len(m)  
    for c1 in range(0, len(m)):  
        for c2 in range(0, len(m)):  
            sumCoup += CoupP(m, c1, c2)  
    return sumCoup/NbPossiblePairs
```

Gambar 3.18 Implementasi InterCoup

Fungsi CoupP menghitung berapa nilai coupling antara kedua class dengan class lainnya. Coup menggunakan fungsi NBCalls untuk mengetahui berapa jumlah relasi antara kedua class tersebut.

```
def CoupP(m, c1, c2):  
    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:  
        # print("NB Calls 0")  
        return 0  
    return (NbCalls(m, c1, c2) + NbCalls(m, c2, c1))/TotalNBCalls
```

Gambar 3.19 Implementasi Coup

InterCoh menghitung cohesi dari setiap partisi di microservice. Setiap partisi yang dibentuk bila memiliki call maka call tersebut dihitung sebagai koneksi langsung dan koneksi tidak langsung ketika call diluar dari partisinya sendiri.

```
def InterCoh(m, rm):
    NbDirectConnections = 0
    NbPossibleConnection = 0
    for _, p in rm.items():
        if len(p) == 0:
            continue
        adj, _ = createAdjacentMatrix(p)
        for c1 in range(0, len(adj)):
            for c2 in range(0, len(adj)):
                NbDirectConnections += CoupP(adj, c1, c2)
    for c1 in range(0, len(m)):
        for c2 in range(0, len(m)):
            NbPossibleConnection += CoupP(m, c1, c2)
    NbPossibleConnection += NbDirectConnections
    return NbDirectConnections / NbPossibleConnection
```

Gambar 3.20 Implementasi InterCoh

Proses pencarian jumlah cluster dilakukan dari 1 cluster hingga semua cluster hanya memiliki 1 objek. dimulai dari pemotongan *tree* dan menggabungkan class yang bersatu pada partisinya masing-masing dan kemudian memanggil fungsi FOne. FOne menghasilkan nilai terendah (mendekati 0) adalah jumlah cluster yang terbaik untuk microservice.

```
def calculate_FOne(_n_clusters):
    try :
        ct = cut_tree(tmpz, n_clusters=_n_clusters)
        clusterCT = cutTreeToCluster(ct)
        mgCls, inCls = mergeClass(clusterCT)
        adjMerge, _ = createAdjacentMatrix(mgCls)
        result = FOne(adjMerge, inCls)
        print(result)
        return result
    except Exception as e:
        return {'value' : 1 , "Error" : e}
```

```

from multiprocessing import Process

qualityCluster = []
procs = []
MAX_PROCESSING = 4

for b in range(1, len(listLabel), MAX_PROCESSING):
    t = b
    procs = []
    while t < b+MAX_PROCESSING and t ≤ len(listLabel):
        proc = Process(target=calculate_FOne, args=(t,))
        procs.append(proc)
        proc.start()
        t+=1
    for proc in procs:
        qualityCluster.append(proc.join())

```

Gambar 3.21 Penerapan Pemilihan Jumlah Cluster terbaik

Berikut adalah nilai coupling dan cohes masing-masing jumlah cluster. Semakin tinggi jumlah cluster maka nilai coupling akan meningkat dan begitupula sebaliknya untuk nilai cohesion. Dari hasil yang ditemukan bahwa cluster yang ideal berjumlah 8-15 service.

```

{'coupling': 0.6839880216997086, 'cohesion': 0.9950231686813121, 'value': 0.507831205630073, 'clusterSize': 3}
{'coupling': 0.5917319609409468, 'cohesion': 0.9942610775947084, 'value': 0.42014861516981195, 'clusterSize': 4}
{'coupling': 0.7290095525389643, 'cohesion': 0.9964600354647024, 'value': 0.6136197851356803, 'clusterSize': 2}
{'coupling': 0.0, 'cohesion': 1.0, 'value': 0.5, 'clusterSize': 1}
{'coupling': 0.6088190355239034, 'cohesion': 0.9925958915356443, 'value': 0.4036691069155161, 'clusterSize': 5}
{'coupling': 0.5436194664131861, 'cohesion': 0.9894475498661741, 'value': 0.33365020507322896, 'clusterSize': 8}
{'coupling': 0.5566289335618729, 'cohesion': 0.9905484993023813, 'value': 0.3490679310168209, 'clusterSize': 7}
{'coupling': 0.5543988701198025, 'cohesion': 0.9919185738346786, 'value': 0.3598593162127911, 'clusterSize': 6}
{'coupling': 0.6089537990200921, 'cohesion': 0.9824157112368644, 'value': 0.3454108874782487, 'clusterSize': 12}
{'coupling': 0.6410965595270569, 'cohesion': 0.9830096174192845, 'value': 0.36523053510076864, 'clusterSize': 11}
{'coupling': 0.67741359512988, 'cohesion': 0.9852859556455444, 'value': 0.3934587884034687, 'clusterSize': 9}
{'coupling': 0.644866979261845, 'cohesion': 0.9844333133628071, 'value': 0.37165515529906284, 'clusterSize': 10}
{'coupling': 0.7044490278937455, 'cohesion': 0.9754046496725745, 'value': 0.3847380022692919, 'clusterSize': 15}
{'coupling': 0.704330297151238, 'cohesion': 0.9738162619021742, 'value': 0.3825969067600619, 'clusterSize': 16}
{'coupling': 0.689174654142646, 'cohesion': 0.9775424539016395, 'value': 0.3794995575678101, 'clusterSize': 14}
{'coupling': 0.7004657602120469, 'cohesion': 0.97879466522707, 'value': 0.387878828768603, 'clusterSize': 13}
{'coupling': 0.7715720404045149, 'cohesion': 0.9643837718215246, 'value': 0.40989561449779555, 'clusterSize': 20}
{'coupling': 0.8557055940135114, 'cohesion': 0.4393700048269287, 'value': 0.42825005740895006, 'clusterSize': 553}
{'coupling': 0.8608531227202318, 'cohesion': 0.43735653924320095, 'value': 0.43082128747856646, 'clusterSize': 554}
{'coupling': 0.863315698954948, 'cohesion': 0.43507780074231905, 'value': 0.4320484040562373, 'clusterSize': 557}
{'coupling': 0.8609488560079228, 'cohesion': 0.4345371489457951, 'value': 0.43086240760123445, 'clusterSize': 560}
{'coupling': 0.8629300564334684, 'cohesion': 0.4347214966577956, 'value': 0.4318545636080046, 'clusterSize': 558}
{'coupling': 0.8628734998271561, 'cohesion': 0.43446254153487546, 'value': 0.43182535683802786, 'clusterSize': 559}
{'coupling': 0.8601410818085943, 'cohesion': 0.4339976660869206, 'value': 0.4304573480933229, 'clusterSize': 561}
{'coupling': 0.8589577795424186, 'cohesion': 0.43256750323503496, 'value': 0.42986373630433655, 'clusterSize': 562}

```

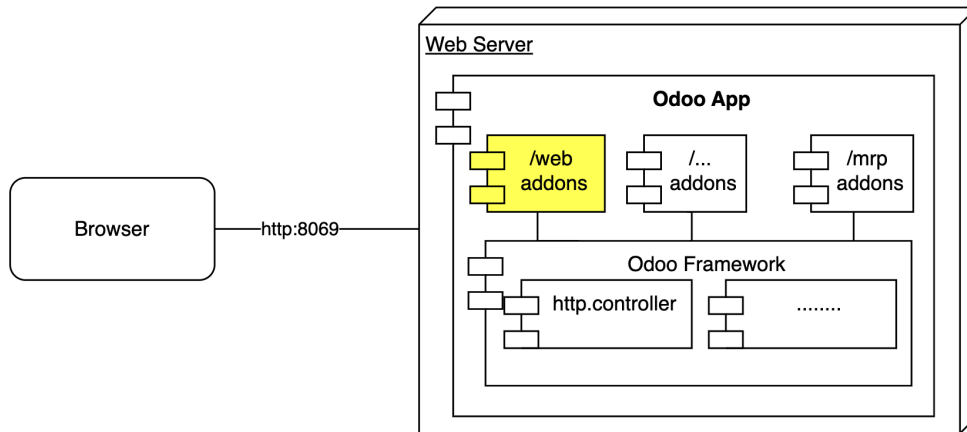
Gambar 3.22 Nilai Coupling dan Cohesion setiap jumlah cluster

3.3.2 Dekomposisi Monolitik ke Microservice

3.3.2.1 Pemisahan User Interface

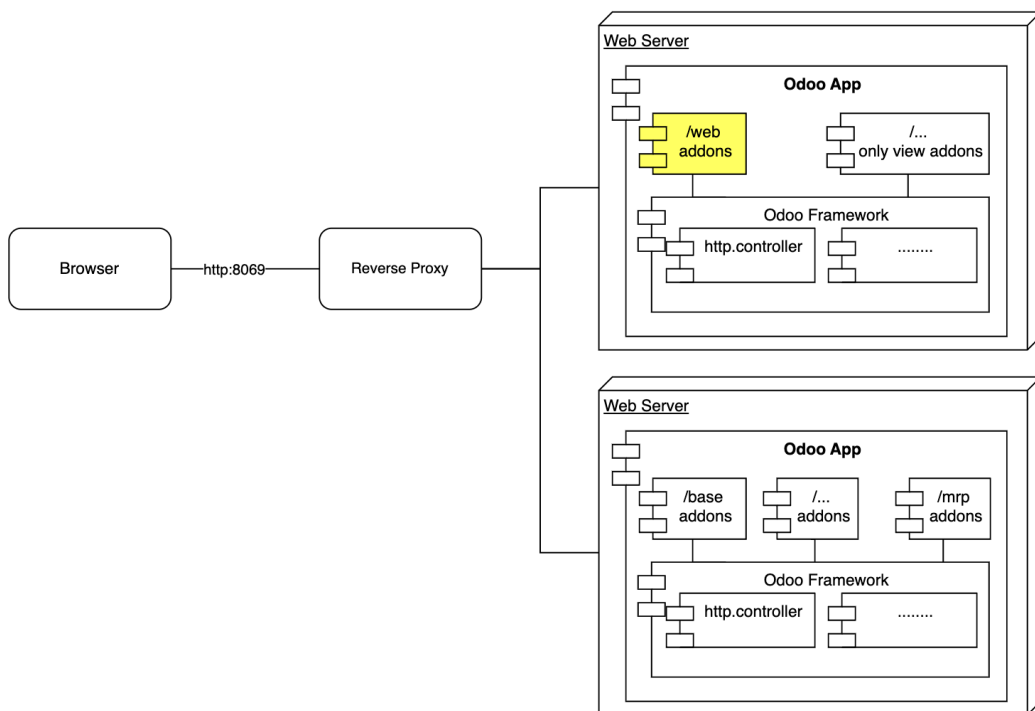
Odoo adalah aplikasi ERP berbasis web, tampilan pada Odoo bisa dibuka pada broser yang kompatibel. Odoo menggunakan pendekatan SPA (Single Page Application) dan adanya server rendering untuk menghasilkan HTML. UI dapat memanggil API yang berada pada server yang sama, sehingga diperlukan

pemisahan kepentingan. Benefit dari proses ini dapat meningkatkan skalabilitas dan pengembangan bisa dilakukan secara independen.



Gambar 3.23 Arsitektur UI Monolit

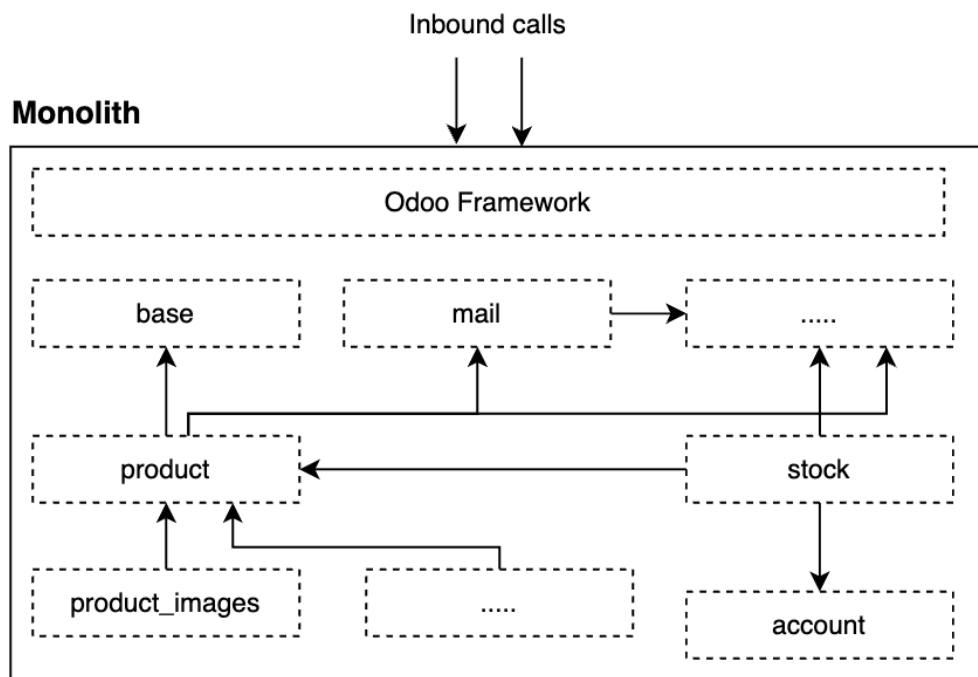
Proses pemisahan membutuhkan reverse proxy yang dapat menghubungkan web server dimana bertanggung jawab pada hal seperti file static(css,js,gambar) dan UI dengan web server lainnya. Tujuan adanya reverse proxy agar client hanya perlu mengetahui satu pintu masuk aplikasi yaitu reverse proxy itu sendiri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



Gambar 3.24 Arsitektur UI di Microservice

3.3.2.2 Strategi Pemisahan Kode

Berdasarkan landasan teori terdapat beberapa strategi pemisahan kode aplikasi monolit, 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 monolit bisa berdiri bersamaan dengan service yang ingin dibangun atau dimigrasi.

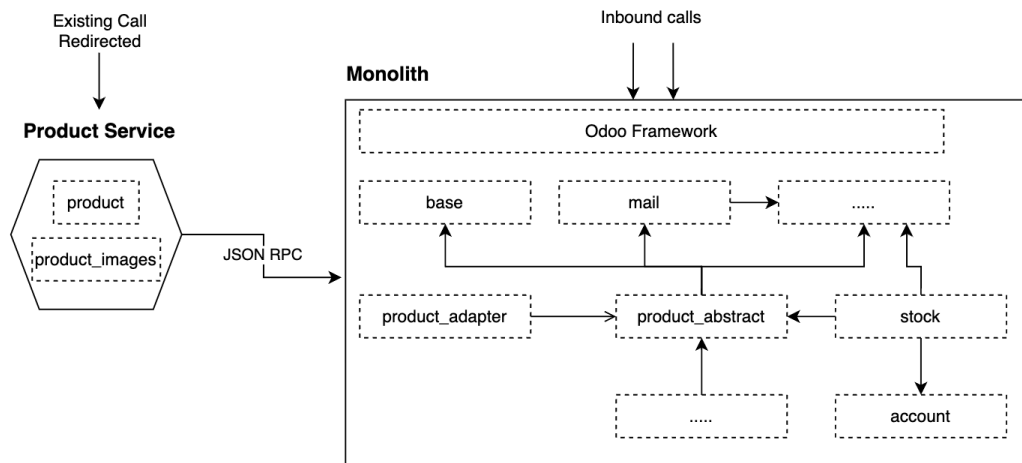


Gambar 3.25 Struktur Module dan Keterhubungannya di Aplikasi Monolit

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 monolith ke service yang baru dibuat. Tugas akhir ini milih bagian yang dipindahkan pada kasus bisnis yaitu 'Product', Product bisa dilakukan penambahan Product baru, perubahan atributnya, pencarian atau mendapatkan product dan penghapusan product.

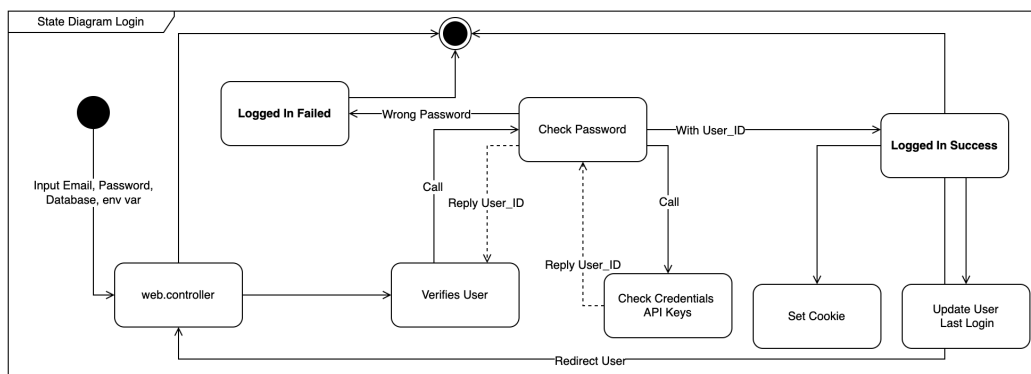
Proses pemindahan module dibantu dengan hasil clustering yang sudah diproses sebelumnya sehingga dibuat untuk membuat microservice. Untuk menghubungkan antara bagian yang sudah dipisah dari monolit dengan bagian yang masih di monolit maka diperlukan penerapan pola Branch by Abstraction. Terdapat dua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

monolit tidak terdampak dan Adapter adalah implementasi sesungguhnya yang menghubungkan antara service dan aplikasi monolith.



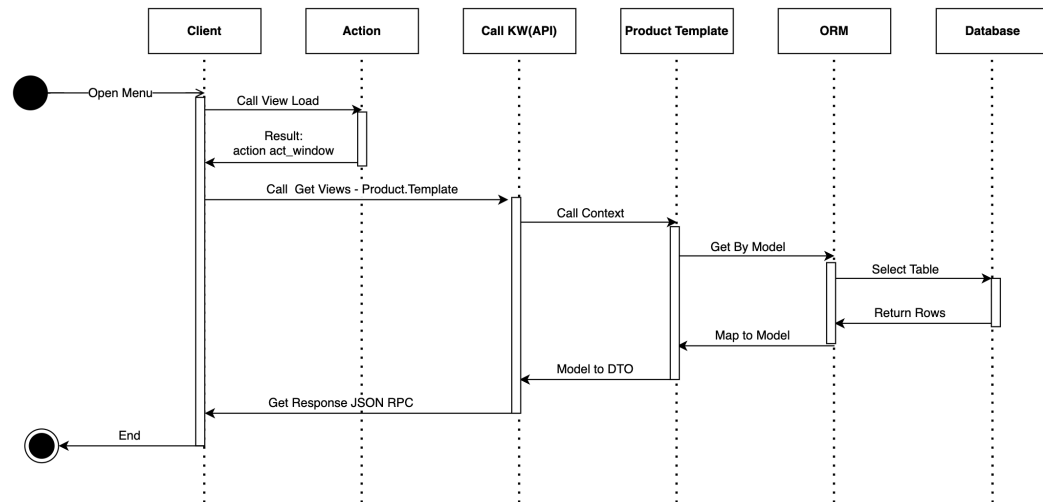
Gambar 3.26 Penerapan Pola Strangle dan Branch by Abstraction

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



Gambar 3.27 State Diagram pada proses login

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



Gambar 3.28 Sequence Diagram pada kasus pengambilan data Product

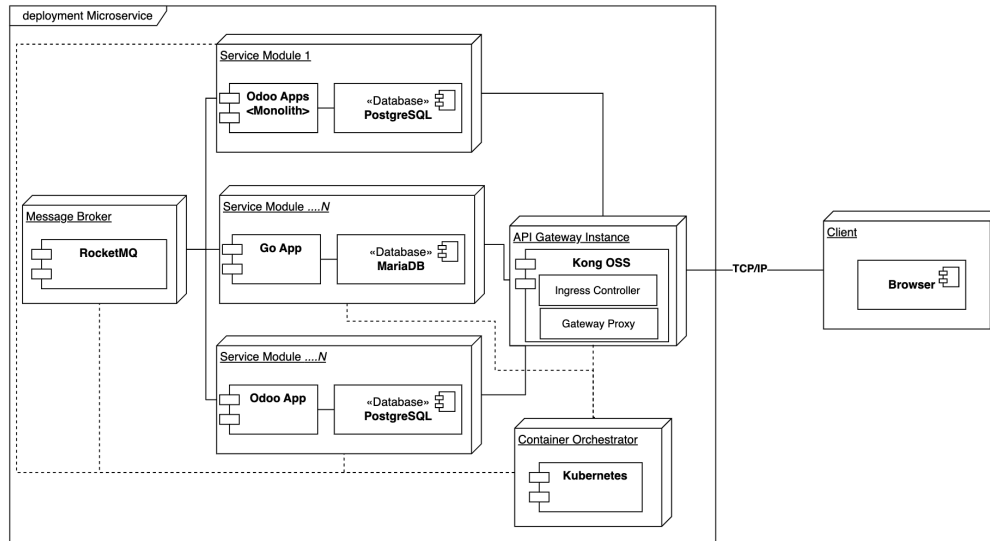
3.3.2.3 Komunikasi antar service

Proses komunikasi antar service dilakukan melalui JSON-RPC karena RPC ini sudah kompatibel dengan aplikasi monolit yang dibangun. Apabila diperlukan penambahan metode komunikasi lainnya seperti gRPC maka harus dibangun service yang mengubah gRPC menjad JSON-RPC atau sebaliknya.

3.3.2.4 Strategi Pemisahan database

Pemisahan database dilakukan bersama-sama dengan pemisahan kode karena pada Odoo sudah terdapat ORM yang mengelola database. Untuk menjaga konsistensi database antara service maka diterapkan SAGA dengan pendekatan Choreography. Untuk mengakses database monolit akan digunakan sebagai data access layer melalui API yang bisa berupa JSON-RPC. Proses seperti Insert, Update, Delete atau proses yang mengubah data maka harus melalui message broker agar pesan tidak hilang bila terjadi kegagalan disalah satu pihak. Untuk mengakses data seperti mencari dan mengambil data dapat melalui RPC.

3.3.3 Deployment



Gambar 3.29 Diagram Deployment

Tugas Akhir ini menggunakan API Gateway Kong (*off-the-shelf*) karena Kong sudah memiliki fitur yang lengkap pada kasus migrasi aplikasi monolit ke microservice. Fitur itu berupa kemampuan untuk redireksi url untuk menerapkan proses strangle, adanya integrasi dengan Kubernetes untuk meningkatkan skalabilitas dan penemuan service, dan memiliki perfoma yang baik.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

4.1 Lingkungan Implementasi

Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei. Aliquam tincidunt a nulla ac posuere. Maecenas sapien mi, feugiat sit amet tellus at, dictum varius ante. Cras rutrum facilisis felis at hendrerit. Nullam eleifend sed lorem a iaculis. Donec ut odio at nisl molestie euismod quis et purus. Curabitur eu ex turpis. Etiam maximus metus non iaculis placerat. Sed in risus sodales, posuere elit in, eleifend tellus. Mauris at consectetur arcu. Integer fringilla eros mi, vel volutpat enim commodo ac.

4.1.1 Spesifikasi Perangkat Keras

Suspendisse ac porta diam, ut viverra ante. Aliquam mattis tincidunt diam in molestie. Sed auctor fermentum turpis, sed varius ante. Nulla rutrum, enim et efficitur dignissim, urna diam consequat purus, sit amet elementum nibh mauris ut tellus. Quisque interdum leo ligula, a volutpat mauris viverra ut. Fusce ac felis finibus, convallis ligula a, aliquam nunc. Quisque faucibus ligula et ornare finibus. Morbi maximus dolor vitae dolor tristique, eu sagittis metus auctor. Pellentesque quam lacus, ornare ut est ut, egestas auctor leo. Duis eros neque, mollis quis elit id, cursus egestas neque. Pellentesque ac sapien vitae nulla varius rhoncus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam pharetra nisl et massa facilisis aliquet. Nulla sit amet quam enim. Nunc dictum pellentesque orci, at sollicitudin erat condimentum eu. Nullam mi dolor, vestibulum at lacinia quis, feugiat faucibus felis.

Suspendisse ac porta diam, ut viverra ante. Aliquam mattis tincidunt diam in molestie. Sed auctor fermentum turpis, sed varius ante. Nulla rutrum, enim et efficitur dignissim, urna diam consequat purus, sit amet elementum nibh mauris ut tellus. Quisque interdum leo ligula, a volutpat mauris viverra ut. Fusce ac felis finibus, convallis ligula a, aliquam nunc. Quisque faucibus ligula et ornare finibus. Morbi maximus dolor vitae dolor tristique, eu sagittis metus auctor. Pellentesque

quam lacus, ornare ut est ut, egestas auctor leo. Duis eros neque, mollis quis elit id, cursus egestas neque. Pellentesque ac sapien vitae nulla varius rhoncus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam pharetra nisl et massa facilisis aliquet. Nulla sit amet quam enim. Nunc dictum pellentesque orci, at sollicitudin erat condimentum eu. Nullam mi dolor, vestibulum at lacinia quis, feugiat faucibus felis.

4.1.2 Spesifikasi Perangkat Lunak

Suspendisse ac porta diam, ut viverra ante. Aliquam mattis tincidunt diam in molestie. Sed auctor fermentum turpis, sed varius ante. Nulla rutrum, enim et efficitur dignissim, urna diam consequat purus, sit amet elementum nibh mauris ut tellus. Quisque interdum leo ligula, a volutpat mauris viverra ut. Fusce ac felis finibus, convallis ligula a, aliquam nunc. Quisque faucibus ligula et ornare finibus. Morbi maximus dolor vitae dolor tristique, eu sagittis metus auctor. Pellentesque quam lacus, ornare ut est ut, egestas auctor leo. Duis eros neque, mollis quis elit id, cursus egestas neque. Pellentesque ac sapien vitae nulla varius rhoncus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam pharetra nisl et massa facilisis aliquet. Nulla sit amet quam enim. Nunc dictum pellentesque orci, at sollicitudin erat condimentum eu. Nullam mi dolor, vestibulum at lacinia quis, feugiat faucibus felis.

4.2 Implementasi Perangkat Lunak

Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet inductum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire.

Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet inductum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

4.2.1 Implementasi *Class*

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

4.2.1.1 *Class Nama_Class_1*

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

4.2.1.2 *Class Nama_Class_2*

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

4.2.2 Implementasi *Numquam*

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

4.3 Implementasi *Nama_Implementasi*

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

4.4 Implementasi *Aplikasi*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ac felis dignissim, iaculis odio ut, euismod quam. Donec vestibulum pellentesque sem, eu aliquet purus lacinia ac. Nam porttitor auctor justo et lobortis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed et gravida neque. Praesent commodo aliquam vestibulum. Vivamus blandit mattis mi ut

euismod. Proin vitae vestibulum orci, eget elementum tellus. Suspendisse potenti.

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

4.5 Pengujian

Quisque dictum auctor tempor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vestibulum ultricies justo elit, sed tincidunt tellus congue quis. Suspendisse potenti. In iaculis volutpat odio sed placerat. Nullam est purus, egestas sit amet sagittis sit amet, eleifend in nisl. Nullam vitae auctor dolor. Nulla non laoreet dolor. Quisque nibh enim, bibendum sit amet tristique sit amet, efficitur nec tellus. Nullam congue ex felis, quis aliquam purus vulputate in. Aliquam in euismod neque. Sed quis odio non ex molestie posuere. Aenean efficitur id ex ut faucibus. Suspendisse imperdiet mattis ipsum, viverra efficitur ligula. Nulla varius lacus massa, ut egestas turpis consequat in. Sed et finibus orci, id tincidunt velit.

4.5.1 Pengujian Nama Pengujian_1

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

Tabel 4.1 atribut pada *class* nama_class_1

atribut:					
Float	C	Float	tol	Float	gamma
Float	a	Float	r	Integer	pos_true
Integer	pos_pred	Integer	net_true	Integer	net_pred
Integer	neg_true	Integer	neg_pred	Float	accuracy_score

Float	precision_score	Float	recall_score	Float	f_score
-------	-----------------	-------	--------------	-------	---------

Tabel 4.2 Daftar *method* pada *class helper*

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

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

4.5.2 Pengujian Nama Pengujian 2

Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

BAB 5 KESIMPULAN DAN SARAN

5.1 Kesimpulan

Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ac felis dignissim, iaculis odio ut, euismod quam. Donec vestibulum pellentesque sem, eu aliquet purus lacinia ac. Nam porttitor auctor justo et lobortis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed et gravida neque. Praesent commodo aliquam vestibulum. Vivamus blandit mattis mi ut euismod. Proin vitae vestibulum orci, eget elementum tellus. Suspendisse potenti.

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

5.2 Saran

Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per

vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ac felis dignissim, iaculis odio ut, euismod quam. Donec vestibulum pellentesque sem, eu aliquet purus lacinia ac. Nam porttitor auctor justo et lobortis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed et gravida neque. Praesent commodo aliquam vestibulum. Vivamus blandit mattis mi ut euismod. Proin vitae vestibulum orci, eget elementum tellus. Suspendisse potenti.

DAFTAR REFERENSI

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

- [9] Richardson, C. (2018) *Microservice patterns: With examples in Java*. Shelter Island, NY: Manning Publications.
- [10] 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>.
- [11] Jain, A.K. and Dubes, R.C. (1988) *Algorithms for clustering data*. Englewood Cliffs, NJ: Prentice Hall.
- [12] 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]
- [13] 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.
- [14] 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.
- [15] 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.
- [16] "Odoo documentation," Odoo. [Online]. Available: <https://www.odoo.com/documentation/16.0/>. [Accessed: 21-Mar-2023].

LAMPIRAN A LAMPIRAN A

ASJDBAKJSDBKA

Tabel A-1 *Lorem ipsum*

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

Tabel A-1 *Lorem ipsum*

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