

**ANALISIS PERFORMA APLIKASI OWL ASSISTANT MENGGUNAKAN  
*MONOLITHIC DAN MICROSERVICES ARCHITECTURE***

**SKRIPSI**

Diajukan untuk menempuh Ujian Sarjana pada  
Fakultas Matematika dan Ilmu Pengetahuan Alam  
Universitas Padjadjaran

**ASEP NUR MUHAMMAD ISKANDAR YUSUF  
140810140070**



**UNIVERSITAS PADJADJARAN  
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM  
PROGRAM STUDI TEKNIK INFORMATIKA  
JATINANGOR**

**2018**

## SKRIPSI

### ANALISIS PERFORMA APLIKASI OWL ASSISTANT MENGUNAKAN MONOLITHIC DAN MICROSERVICES ARCHITECTURE

#### *PERFORMANCE ANALYSIS OWL ASSISTANT APPLICATION USING MONOLITHIC AND MICROSERVICES ARCHITECTURE*

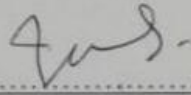

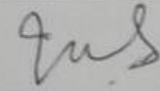
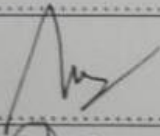
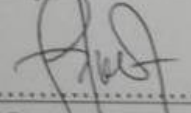
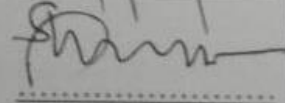
Dipersiapkan dan disusun oleh

**Asep Nur Muhammad Iskandar Yusuf**  
**140810140070**

telah dipertahankan dihadapan Tim Penguji Program Sarjana (S-1) Teknik  
Informatika  
Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Padjadjaran  
Pada tanggal 08 Februari 2018

#### Tim Penguji

1. Dr. Juli Rejito, Drs., M.Kom. Ketua Tim Penguji  
NIP. 19680717 199303 1 003
2. Erick Paulus, S.Si, M.Kom. Pembimbing  
NIP. 198210318 200604 1 001
3. Dr. Juli Rejito, Drs., M.Kom. Co-Pembimbing  
NIP. 19680717 199303 1 003
4. Drs. Ino Suryana, M.Kom Penguji  
NIP. 19600115 198701 1 002
5. Rudi Rosadi, S.Si., M.Kom. Penguji  
NIP. 19760723 200812 1 001
6. Dr. Asep Sholahuddin, MT. Penguji  
NIP. 19670403 199303 1 002

  
.....  
  
.....  
  
.....  
  
.....  
  
.....  
  
.....

## KATA PENGANTAR

Segala puji dan syukur penulis panjatkan ke hadirat Allah SWT yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan penyusunan skripsi yang berjudul “**Analisis Performa Aplikasi Owl Assistant Menggunakan *Monolithic Dan Microservices Architecture***” sebagai salah satu syarat dalam menempuh ujian sarjana pada Program Studi S-1 Teknik Informatika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Padjadjaran.

Dalam proses penyusunan dan penulisan skripsi ini tidak terlepas dari bantuan, bimbingan, serta dukungan dari berbagai pihak. Oleh karena itu dalam kesempatan ini penulis mengucapkan terima kasih kepada Bapak Erick Paulus, S.Si., M.Kom sebagai pembimbing utama, Bapak Dr. Juli Rejito, M.Kom, sebagai Ketua Program Studi Teknik Informatika sekaligus pembimbing pendamping yang telah meluangkan waktu dan pikirannya sehingga penulis dapat menyelesaikan skripsi ini. Ucapan terima kasih juga diberikan kepada keluarga penulis yang selalu memberikan motivasi dan doa yang menjadi pendorong dalam penyelesaian skripsi ini. Penulis juga mengucapkan terima kasih sebanyak-banyaknya kepada:

1. Prof. Dr. Sudradjat, MS, selaku Dekan Fakultas Matematika dan Ilmu Pengetahuan Alam
  2. Dr. Setiawan Hadi, M.Sc.Cs., selaku Kepala Departemen Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Padjadjaran
- Ino Suryana , Drs., M.Komp., Rudi Rosadi, S.Si., M.Kom., dan Dr. Asep Sholahuddin ., MT. selaku dosen penguji

3. Akik Hidayat , Drs., M.Kom., selaku dosen wali yang telah meluangkan waktu, membimbing, dan mendidik penulis dari sejak mahasiswa baru hingga menyelesaikan skripsi
4. Seluruh staf pengajar dan tata usaha Departemen Ilmu Komputer FMIPA Unpad
5. Risal Falah, Rifki Muhammad, Hidayaturrahman, dan Febriyani Pertiwi yang telah membantu dalam memberikan bekerja sama dan memberikan banyak ilmu dalam pembuatan portal praktikum dan skripsi ini
6. Teman-teman Program Studi S-1 Teknik Informatika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Padjadjaran
7. Seluruh pihak yang tidak dapat penulis sebutkan, yang telah memberikan motivasi dan bantuan kepada penulis.

Akhir kata, penulis berharap semoga skripsi ini dapat bermanfaat bagi pengembangan ilmu informatika.

Jatinangor, 08 Februari 2018

Penulis

## ABSTRAK

Perancangan arsitektur aplikasi merupakan peran penting untuk meningkatkan kinerja aplikasi. Arsitektur Microservices adalah pola arsitektur untuk pengembangan aplikasi terdistribusi, di mana aplikasi terdiri dari komponen independen kecil sebagai layanan dan sebagian besar diimplementasikan pada tingkat Enterprise. Berbeda dengan microservices, komponen fungsi besar dalam arsitektur Monolitik disimpan dalam bundel atau folder dan berdiri sebagai satu layanan. Skripsi ini menampilkan perbandingan kinerja arsitektur Monolitik dan Microservices – Studi kasus pada skripsi ini adalah aplikasi Owl Assistant. Dengan *request time* dan *resource utilization* sebagai indeks parameter yang diuji, penelitian ini menggunakan Alibaba *cloud* dengan spesifikasi 1 Core *CPU*, 1 GB *Memory*, dan peak bandwidth 200MB sebagai layanan *cloud* untuk menyebarkan aplikasi ke lingkungan pengujian. Menggunakan sejumlah permintaan percobaan dari 1-2000 *request* dengan 15 kali pengulangan dan sumber daya yang disediakan pada kedua arsitektur sama, Monolitik mendapat waktu 76 - 21020 *millisecond* sementara microservices 261 - 51331 *millisecond*, pemanfaatan sumber daya menggunakan monolitik memiliki *peak CPU* dari permintaan 2 - 71%, sedangkan microservices 74%, dan penggunaan memori keduanya mencapai 100% pada 2000 *request*. Hasil penelitian menunjukkan Owl Assistant menggunakan Monolitik memiliki performa yang lebih baik daripada menggunakan microservices yang secara teknis memiliki jumlah data yang sama yaitu 10.809.115 data baris..

**Kata Kunci:** Performa, arsitektur, *microservices architecture*, *monolithic architecture*, *request time*, *resource utilization*

## **ABSTRACT**

*The design of application architecture is an important role to improve performance of application. Microservices architecture is an architectural pattern for distributed application development, where application consists of small independent components as services and mostly implemented at Enterprise level. Different with microservices, components of large functions in Monolithic architecture are stored in a bundle or folder and stand as one service. This paper performs performance comparison Monolithic and Microservices architecture – Case study for this paper is Portal Laboratory application. With request time and resource utilization as index parameters which are tested, this research use Alibaba cloud with specification 1 Core CPU, 1 GB Memory, and peak bandwidth 200MB as cloud service to deploy the application to testing environment. Using number of requests experiment from 1-2000 requests with 15 loop times and resources provided on both architectures are the same, Monolithic got time 76 - 21020 millisecond while microservices 261 - 51331 millisecond, resource utilization using monolithic has peak CPU of the requests 2 - 71%, while microservices 74%, and memory usages in both reaches 100% in 2000 requests. Result show Portal Laboratory using monolithic has better performance than using microservices which technically has the same amount of data that is 10.809.115 rows data.*

**Keyword:** *Performance, architecture, microservices architecture, monolithic architecture, request time, resource utilization*

## DAFTAR ISI

<b>KATA PENGANTAR .....</b>	<b>iii</b>
<b>ABSTRAK.....</b>	<b>v</b>
<b>ABSTRACT .....</b>	<b>vi</b>
<b>DAFTAR ISI .....</b>	<b>vii</b>
<b>DAFTAR TABEL .....</b>	<b>x</b>
<b>DAFTAR GAMBAR .....</b>	<b>xii</b>
<b>BAB I PENDAHULUAN .....</b>	<b>1</b>
1.1    Latar Belakang.....	1
1.2    Identifikasi Masalah .....	2
1.3    Batasan Masalah .....	3
1.4    Tujuan.....	3
1.5    Metodologi Penelitian .....	3
1.6    Manfaat Penelitian.....	4
1.6.1    Manfaat Teoretis .....	4
1.6.2    Manfaat Praktis .....	5
1.7    Sistematika Penulisan.....	5
<b>BAB II TINJAUAN PUSTAKA .....</b>	<b>7</b>
2.1    Owl Assistant.....	7
2.2    Client Server .....	7
2.3    Monolithic Architecture .....	8

2.4	Microservices Architecture .....	9
2.5	Granularity Service.....	10
2.6	MySQL & Redis .....	11
2.7	Docker .....	11
2.8	Golang .....	12
2.9	Jmeter & Aliyun Alibaba Cloud.....	12
2.10	Diagram Use Case .....	13
2.11	Activity Diagram .....	14
2.12	<i>Entity Relationship Diagram</i> .....	15
<b>BAB III ANALISIS DAN DESAIN.....</b>		<b>17</b>
3.1	Desain Sistematisa Penelitian .....	17
3.1.1	Studi Pustaka.....	18
3.1.2	Analisis Kebutuhan Aplikasi .....	18
3.1.3	Perencanaan Arsitektur .....	21
3.1.4	Pengujian Arsitektur .....	37
3.1.5	Alat dan Bahan Penelitian.....	37
3.1.6	Use Case Diagram.....	39
3.1.7	Activity Diagram.....	40
<b>BAB IV HASIL DAN PEMBAHASAN .....</b>		<b>44</b>
4.1	Spesifikasi Arsitektur .....	44
4.1.1	Spesifikasi <i>Monolithic architecture</i> .....	44
4.1.2	Spesifikasi <i>Microservices architecture</i> .....	44



4.2	Rencana Uji .....	45
4.2.1	Web Service Utama.....	46
4.2.2	API dan Antarmuka <i>Detail Profile</i> .....	48
4.2.3	API dan Antarmuka <i>Detail Course</i> .....	50
4.2.4	API dan Antarmuka <i>Detail Assignment</i> .....	52
4.2.5	API dan Antarmuka <i>detail information</i> .....	57
4.2.6	API Bot dan Antarmuka Bot .....	59
4.3	Hasil Penelitian.....	62
4.4	Analisis .....	66
<b>BAB V SIMPULAN DAN SARAN .....</b>		<b>68</b>
5.1	Simpulan.....	68
5.2	Saran .....	69
<b>DAFTAR PUSTAKA .....</b>		<b>70</b>
<b>LAMPIRAN .....</b>		<b>73</b>

## DAFTAR TABEL

Tabel 2.1 Simbol Diagram <i>Use Case</i> .....	13
Tabel 2.2 Simbol <i>Activity Diagram</i> .....	14
Tabel 2.3 Simbol Relationship Diagram Database .....	15
Tabel 3.1 Tabel Kebutuhan fungsional Owl Assistant.....	20
Tabel 3.2 Kebutuhan Non Fungsional Owl Assistant.....	21
Tabel 3.3 Kamus Data Tabel <i>Assignment</i> .....	24
Tabel 3.4 Kamus Data Tabel <i>Attendance</i> .....	24
Tabel 3.5 Kamus Data Tabel <i>Bot_Logs</i> .....	25
Tabel 3.6 Kamus Data Tabel <i>Course</i> .....	25
Tabel 3.7 Kamus Data Tabel <i>Files</i> .....	26
Tabel 3.8 Kamus Data Tabel <i>Grade</i> .....	27
Tabel 3.9 Kamus Data Tabel <i>Information</i> .....	27
Tabel 3.10 Kamus Data Tabel <i>Meetings</i> .....	28
Tabel 3.11 Kamus Data Tabel <i>Users Schedule</i> .....	28
Tabel 3.12 Kamus Data Tabel <i>Places</i> .....	29
Tabel 3.13 Kamus Data Tabel <i>Role Groups</i> .....	29
Tabel 3.14 Kamus Data Tabel <i>Role Group Modules</i> .....	30
Tabel 3.15 Kamus Data Tabel <i>Tutorials</i> .....	30
Tabel 3.16 Kamus Data Tabel <i>Schedules</i> .....	31
Tabel 3.17 Kamus Data Tabel <i>Users</i> .....	32
Tabel 3.18 Jumlah data yang tersedia pada <i>database</i> .....	39

Tabel 4.1 Pembagian <i>resource</i> untuk setiap layanan .....	45
Tabel 4.2 Properti rencana uji .....	45
Tabel 4.3 API yang digunakan untuk pengujian .....	46
Tabel 4.4 Hasil Uji Performa <i>Complete Request Time</i> Aplikasi Owl Assistant ...	62
Tabel 4.5 Hasil Uji <i>CPU Usages</i> Aplikasi Owl Assistant .....	63
Tabel 4.6 Hasil Uji <i>Memory Usages</i> Aplikasi Owl Assistant .....	65
Tabel 4.7 Perbedaan kedua arsitektur dengan jumlah 1000 <i>request</i> .....	66

## DAFTAR GAMBAR

Gambar 2.1 Ilustrasi <i>Client-Server</i> .....	8
Gambar 2.2 Ilustrasi komunikasi pada <i>monolithic architecture</i> .....	9
Gambar 2.3 Ilustrasi komunikasi <i>microservices architecture</i> .....	10
Gambar 2.4 Rancangan Biaya <i>Granularity</i> .....	10
Gambar 3.1 Desain Sistematika Penelitian .....	17
Gambar 3.2 Desain <i>Monolithic Architecture</i> Owl Assistant.....	22
Gambar 3.3 Desain Database Owl Assistant - <i>Monolithic Architecture</i> .....	23
Gambar 3.4 Desain <i>Microservices Architecture</i> Owl Assistant.....	33
Gambar 3.5 Desain <i>Database Service User</i> .....	34
Gambar 3.6 Desain <i>Database Service Course</i> .....	35
Gambar 3.7 Desain <i>Database Service Information</i> .....	36
Gambar 3.8 Desain <i>Database Service Bot</i> .....	36
Gambar 3.9 Use Case Diagram Owl Assistant .....	39
Gambar 3.10 <i>Activity Diagram - Assignment and Grade</i> .....	40
Gambar 3.11 <i>Activity Diagram - Course and Schedule Information</i> .....	41
Gambar 3.12 <i>Activity Diagram – Edit Profile</i> .....	42
Gambar 3.13 <i>Activity Diagram – Information List</i> .....	42
Gambar 3.14 <i>Activity Diagram - Chatting Bot</i> .....	43
Gambar 4.1 Antarmuka Jmeter untuk pengujian performa.....	46
Gambar 4.2 Antarmuka dari API <i>/api/v1/user/profile</i> .....	50
Gambar 4.3 Antarmuka aplikasi API <i>/api/admin/v1/course/:id</i> .....	52

Gambar 4.4 Antarmuka API /api/v1/assignment/:id .....	57
Gambar 4.5 Antarmuka Aplikasi API /api/v1/information/:id .....	59
Gambar 4.6 Antarmuka Aplikasi API /api/v1/bot .....	62
Gambar 4.7 Grafik Hasil Uji <i>complete request time</i> kedua arsitektur .....	63
Gambar 4.8 Hasil CPU <i>Utilization</i> .....	64
Gambar 4.9 Hasil <i>Memory Utilization</i> .....	65

# **BAB I**

## **PENDAHULUAN**

Bab ini menguraikan tentang latar belakang penelitian, identifikasi masalah, batasan masalah, maksud dan tujuan penelitian, kegunaan penelitian, metodologi penelitian, dan sistematika penulisan.

### **1.1 Latar Belakang**

Performa aplikasi menjadi perhatian utama setelah fase pengembangan (Zhou et al., 2012). Keputusan pemilihan arsitektur menentukan kemampuan sistem yang diterapkan untuk memenuhi persyaratan atribut fungsional dan kualitas. Memilih dan merancang arsitektur yang memenuhi persyaratan atribut fungsional dan kualitas (misalnya, ketersediaan, keamanan, dan kinerja) sangat penting bagi keberhasilan sistem (Bianco et al., 2007).

*Microservices architecture* merupakan pendekatan pengembangan aplikasi di mana aplikasi besar dibangun sebagai rangkaian layanan kecil modular dan setiap layanan berdiri sebagai layanan mandiri (Yu et al., 2016). *Microservices* pada umumnya diimplementasi di level Enterprise. Contoh seperti MGDIS SA – perusahaan *software editing* yang menggunakan arsitektur *microservices* – dan Hazmat Environmental – aplikasi pelaporan pengiriman barang kimia – merupakan aplikasi yang menerapkan arsitektur *microservices* dan berhasil memperoleh keuntungan yang didapatkan dari *microservices architecture* seperti penggunaan kembali fungsi, pengembangan aplikasi, dan kemudahan skalabilitas dari setiap komponen (Gouigoux & Tamzalit, 2017) (Cherradi et al., 2017)

*Monolithic architecture* merupakan pendekatan pengembangan aplikasi di mana komponen atau layanan dari fungsi besar disimpan dalam bundel atau folder. (Cagla Okutan, 2010). *Monolithic architecture* mempunyai beberapa keuntungan - keuntungan seperti *Microservices* yaitu kemudahan pengembangan, kinerja aplikasi, dan kecepatan penyebaran. Salah satu yang sukses mendapatkan keuntungan dari monolitik yaitu aplikasi ACM News (Zhou et al., 2012)

Berdasarkan keuntungan kedua arsitektur di atas, pada penelitian kali ini penulis akan menganalisis performa web aplikasi Owl Assistant – Sistem informasi laboratorium yang akan diimplementasikan di Teknik Informatika Universitas Padjadjaran – menggunakan *monolithic architecture* dan *microservices architecture*. Indeks parameter yang digunakan yaitu *request time* dan *resource utilization* ( *CPU* dan *memory*). dengan harapan dapat menghasilkan arsitektur yang optimal untuk aplikasi Owl Assistant.

## 1.2 Identifikasi Masalah

Mengacu pada latar belakang tersebut, didapatkan identifikasi masalah yang menjadi pokok pada penelitian ini, yaitu:

- a. Bagaimana perbedaan performa Owl Assistant menggunakan *Monolithic* dan *Microservices architecture*?
- b. Bagaimana perbedaan *resource utilization* Owl Assistant menggunakan *monolithic* dan *microservices architecture*?
- c. Arsitektur apa yang optimal untuk aplikasi Owl Assistant?

### 1.3 Batasan Masalah

Berdasarkan permasalahan yang telah diuraikan di atas, akan dilakukan pembatasan masalah pada penelitian ini sebagai berikut:

- a. Jumlah data yang pada tabel digunakan pengujian yaitu antara 100.000 - 10.000.000 baris data
- b. Indeks parameter pengukuran yang digunakan yaitu *request time* dan *resource utilization*
- c. *Method* yang di uji yaitu hanya *GET method*
- d. *Resource* yang digunakan masing-masing arsitektur mempunyai spesifikasi yang sama

### 1.4 Tujuan

Adapun tujuan yang ingin dicapai oleh penulis dari penelitian ini adalah:

1. Mengetahui performa aplikasi Owl Assistant menggunakan *monolithic* dan *microservices architecture*
2. Mendapatkan arsitektur yang optimal untuk aplikasi Owl Assistant

### 1.5 Metodologi Penelitian

Penelitian ini dilakukan dengan metode berikut ini:

1. Studi pustaka, pada tahap ini akan dilakukan studi pustaka terhadap penelitian-penelitian yang terkait dengan *monolithic* dan *microservices architecture*
2. Analisis kebutuhan, pada tahap ini disusun kebutuhan aplikasi yang akan diuji secara fungsional dan non-fungsional
3. Perencanaan arsitektur sistem yang berisi penggambaran rancangan arsitektur yang akan diuji



4. Pengumpulan data, data yang menjadi parameter pengukuran performa arsitektur yaitu *dummy data*. Pengumpulan data dilakukan dengan memasukkan data secara otomatis menggunakan aplikasi bantuan.
5. Pengujian arsitektur, pengujian dilakukan di tahap *testing*, dengan menggunakan Apache Jmeter, adapun indeks pengukuran performa dari setiap arsitektur yaitu *request time* dan *resource utilization*
6. Analisis hasil pengujian, Hasil dari pengujian performa arsitektur akan di ambil untuk kemudian di ambil kesimpulannya, parameter yang di jadikan acuan untuk perbandingan yaitu *request time*, dan *resource utilization* dari empat fungsi yaitu *assignment*, *users*, *course* dan *information*.
7. Penulisan Laporan, Pada tahap ini dilakukan penyusunan laporan mengenai latar belakang yang merupakan alasan dilakukannya penelitian ini. Selanjutnya, terdapat rancangan penelitian yang berdasarkan tinjauan pustaka. Ada pula hasil penelitian, implementasi sistem, analisis hasil dan kesimpulan penelitian.

## **1.6 Manfaat Penelitian**

### **1.6.1 Manfaat Teoretis**

Hasil penelitian ini diharapkan dapat memperoleh manfaat teoretis sebagai berikut:

1. Hasil dari penelitian ini diharapkan dapat menambah khazanah keilmuan tentang pemilihan arsitektur aplikasi sistem informasi.
2. Menjadi alternatif dalam pemilihan arsitektur aplikasi khususnya aplikasi pada sistem informasi.

### 1.6.2 Manfaat Praktis

1. Diharapkan hasil penelitian ini dapat menghasilkan arsitektur yang optimal untuk aplikasi Owl Assistant
2. Dapat digunakan sebagai referensi untuk penelitian selanjutnya

### 1.7 Sistematika Penulisan

Sistematika penulisan dalam skripsi ini adalah sebagai berikut:

#### **BAB I PENDAHULUAN**

Bab ini menguraikan latar belakang penelitian, identifikasi masalah, batasan masalah, maksud dan tujuan penelitian, kegunaan penelitian, metodologi penelitian, dan sistematika penulisan.

#### **BAB II TINJAUAN PUSTAKA**

Bab ini berisi uraian mengenai penelitian terkait yang pernah dilakukan sebelumnya mengenai analisis performa aplikasi. Pada Bagian ini juga dipaparkan mengenai beberapa teori serta tulisan yang berkaitan dengan penelitian yang dilakukan.

#### **BAB III METODE PENELITIAN**

Pada bab ini diuraikan mengenai proses penelitian yang dilakukan. Bab ini juga menjelaskan alat serta bahan yang digunakan selama proses penelitian. Bab ini menguraikan setiap langkah yang dilakukan selama proses penelitian.

#### **BAB IV HASIL DAN PEMBAHASAN**

Bab ini membahas mengenai hasil dari perbandingan performa aplikasi Owl Assistant menggunakan *monolithic* dan *microservices architecture*. Pada bab ini akan dikaji mengenai tingkat optimal performa aplikasi, ditinjau dari indeks parameter pengukuran dan jumlah data pada tabel yang terkait.

## **BAB V SIMPULAN DAN SARAN**

Bab ini merupakan penutup dari skripsi yang berisi simpulan dan saran yang diambil dari pembahasan pada skripsi ini.

## **BAB II**

### **TINJAUAN PUSTAKA**

Bab ini berisi uraian dipaparkan mengenai beberapa teori yang berkaitan dengan penelitian yang dilakukan. Berikut adalah beberapa landasan teori yang digunakan dalam analisis performa aplikasi menggunakan *monolithic* dan *microservices architecture*.

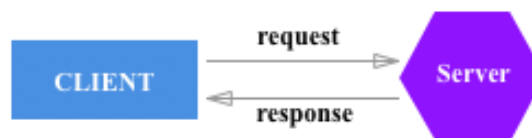
#### **2.1 Owl Assistant**

Owl Assistant merupakan aplikasi *multiplatform* sistem informasi praktikum Teknik Informatika Universitas Padjadjaran. Owl Assistant memiliki fitur utama yaitu mendapatkan informasi-informasi penting mengenai praktikum, sistem penilaian, pengumpulan tugas, dan *Chabot* yang digunakan untuk mengakses informasi praktikum dengan cepat.

#### **2.2 Client Server**

Model *Client-Server* adalah salah satu paradigma yang paling populer yang diandalkan dalam sistem terdistribusi (Jain et al., 2000). Konsep ini merupakan cara untuk mewakili hubungan antara dua program yang berkomunikasi satu sama lain, satu disebut klien dan yang lainnya adalah server. Biasanya, klien adalah orang yang memulai komunikasi dengan sebuah permintaan dan server adalah orang yang menerimanya. Baik klien dan server dapat ditempatkan pada mesin yang sama atau masing-masing pada mesin yang berbeda. Server adalah salah satu sumber yang diwakili sebagai *database* atau aplikasi untuk dibagi dengan pemohon (klien) (Wietze A. de Vries, 1997). Tipe *request-response* dari protokol model ini

direferensikan sebagai *remote procedure calls* (RPC). Paradigma *Client-Server* terutama digunakan di lingkungan Java yang mengandalkan mekanisme pemanggilan Objek Jarak Jauh. Teknologi ini memiliki dampak besar pada model *Client-Server* dan membuatnya berkembang dari dua arsitektur *tier* menjadi tiga dan arsitektur *n-tier*. Gambar 2.1 memberikan penjelasan kasar tentang struktur dasar model *Client-Server*.



Gambar 2.1 Ilustrasi *Client-Server*

### 2.3 Monolithic Architecture

*Monolithic architecture* adalah pola arsitektur perangkat lunak di mana komponen-komponen atau layanan fungsi besar disimpan dalam satu bundel atau folder contohnya dalam berkas WAR (*Web Application Archive*) dan Kopling antar komponen bersifat masif. (Gouigoux & Tamzalit, 2017) Hal itu sangat mudah untuk memetakan objek menjadi *database relational* tunggal. Salah satu keuntungan terbesar dari arsitektur ini adalah transaksional, sinkron, mudah diterapkan, dan mudah untuk divisualisasikan dan dipantau. Proses komunikasi sangat simpel di mana *client* mengakses satu API dan server hanya akan berkomunikasi di dalam satu *service* tersebut memanggil komponen-komponen yang dibutuhkan sebagai *response*. Ilustrasi proses komunikasi *monolithic architecture* diilustrasikan pada Gambar 2.2 (Dzo & Gui, 2017)

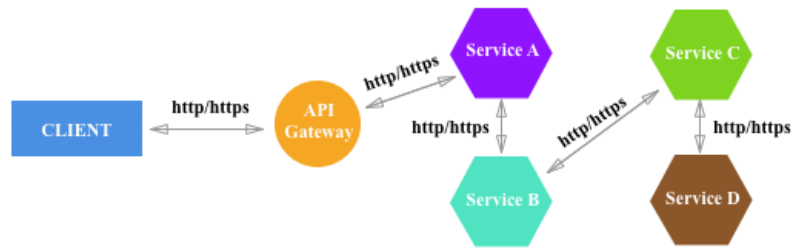


Gambar 2.2 Ilustrasi komunikasi pada *monolithic architecture*

## 2.4 Microservices Architecture

*Microservices architecture* adalah pola arsitektur perangkat lunak untuk pengembangan aplikasi terdistribusi, di mana aplikasi terdiri dari sejumlah layanan-layanan kecil independen. Layanan-layanan kecil atau microservices ini mempunyai arsitekturnya tersendiri dan sumber daya yang dimiliki oleh masing-masing layanan seperti kontainer, *cache*, *datastore*, dan yang lain-lain tidak di bagikan ke layanan yang lainnya. Independen merupakan kata kunci dari *microservices architecture*, pengelolaan data digunakan dalam implementasinya di mana microservices lainnya di izinkan untuk membaca datanya, namun untuk menulis data akan dilakukan melalui layanan yang terkait. (Yu et al., 2016)

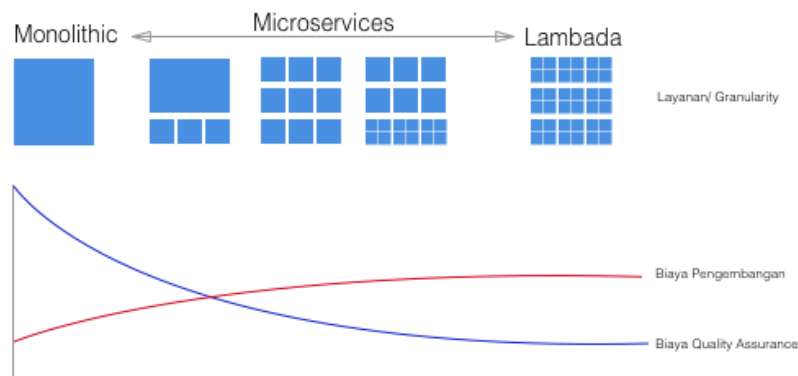
Hal yang penting dari membangun aplikasi menggunakan microservices yaitu komunikasi antar internal *services*. Di mana satu *service* akan berkomunikasi dengan *service* lainnya ketika membutuhkan data dari *service* yang berkaitan. Pada Gambar 2.3 merupakan ilustrasi komunikasi internal antar *service*, di mana *client* hanya akan mengakses satu API kemudian *service* A membutuhkan data dari *service* B sehingga *service* A mengakses *service* B, kemudian *service* B berkomunikasi dengan *service* C dan seterusnya. (Dzo & Gui, 2017)



Gambar 2.3 Ilustrasi komunikasi *microservices architecture*

## 2.5 Granularity Service

*Granularity* atau pembagian layanan merupakan bagian *planning* yang harus dilakukan sebelum migrasi dari *monolithic architecture* ke *microservices architecture*. Pemilihan *granularity* harus mempertimbangkan biaya dari *Quality Assurance* dan biaya pengembangan. Berikut merupakan grafik penentuan *granularity* berdasarkan biaya. (Gouigoux & Tamzalit, 2017)



Gambar 2.4 Rancangan Biaya *Granularity*

*Monolithic architecture* mempunyai biaya pengembangan yang sangat murah, dengan alasan untuk mengembangkan aplikasi tidak perlu banyak pengembang, dan fokus pengembangan hanya pada satu bundel aplikasi, namun untuk biaya *Quality Assurance* mempunyai biaya yang tinggi daripada *microservices architecture*. *Microservices architecture* mempunyai biaya

bervariasi, semakin banyak layanan yang dibuat independen, biaya pengembangan semakin besar namun semakin kecil biaya untuk *Quality Assurance*. Dan jika *service* semakin kecil sampai detail, *microservices architecture* disebut dengan *Lambda*. (Gouigoux & Tamzalit, 2017)

## 2.6 MySQL & Redis

*Database* yang digunakan untuk pengembangan dan penyebaran adalah MySQL dan Redis. MySQL adalah *database* yang biasa digunakan untuk menyimpan data yang berelasi (Letkowski, 2014) sedangkan Redis adalah NoSQL yang mengelola data berupa pasangan kunci dan nilai, yang tersimpan dalam memori utama (RAM). Idealnya Redis lebih suka menyimpan informasi penting yang perlu diakses, dimodifikasi dan disisipkan pada tingkat yang sangat cepat (Baron, 2015)

## 2.7 Docker

Penyebaran aplikasi monolitik dengan semua ketergantungan dan prasyaratnya akan membutuhkan waktu yang lama. Ketika proses penyebaran terjadi pada *microservices architecture* yang terdiri dari beberapa *service* yang harus disebar satu per satu akan membutuhkan waktu yang berkali lipat berdasarkan banyak *services* yang tersedia (Gouigoux & Tamzalit, 2017) Untuk mempercepat proses penyebaran setiap *service*, teknologi yang digunakan yaitu teknologi berbasis kontainer yang merupakan implementasi populer di mana *service* dan kebutuhannya di simpan satu paket di dalam kontainer. *Software* yang sering



digunakan dalam implementasi teknologi berbasis kontainer ini yaitu Docker di samping banyak penyedia layanan yang lainnya seperti Kubertenes.io, Vagrant dan lain-lain. Namun untuk penelitian ini, penulis memilih Docker dengan alasan docker merupakan teknologi kontainer yang ringan dan ideal serta docker mempunyai fitur Docker Swarm yang merupakan *scheduler* yang paling sederhana di antara yang lain. Dan menurut referensi (Zhou, Ye, & Ding, 2012), docker mempercepat pengembangan, penyebaran dan penggulungan puluhan atau ratusan kontainer yang disusun sebagai satu aplikasi. (Gouigoux & Tamzalit, 2017)

## 2.8 Golang

Golang adalah bahasa pemrograman yang dikembangkan oleh Robert Griesemer, Rob Pike, dan Ken Thompson. Bahasa pemrograman ini diumumkan pada November 2009 lalu dan saat ini dipakai di beberapa produk Google. Golang merupakan bahasa pemrograman yang cukup sederhana, handal dan *open source* atau gratis. Golang bisa digunakan untuk *web service*, *artificial intelligence*, dan yang lainnya. (Schmager, 2010).

## 2.9 Jmeter & Aliyun Alibaba Cloud

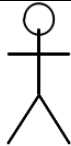
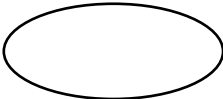

Untuk mengukur perbandingan *monolithic* dan *microservices architecture* berdasarkan *request time* dan *resource utilization*, penulis menggunakan aplikasi standar untuk pengujian yaitu Apache JMeter yang merupakan aplikasi *open source* yang dirancang untuk mengukur kinerja terutama untuk mencatat waktu permintaan (Kumar, 2015). Untuk mencatat *resource utilization*, penulis menggunakan teknik

dari referensi ini (Zhou, Ye, & Ding, 2012) menggunakan log dari server untuk mencatat semua penggunaan sumber saat pengujian dilakukan. Pengujian dilakukan pada tingkat testing. penulis menggunakan layanan awan standar dan populer Aliyun Alibaba untuk menyebarkan aplikasi ke tahap pengujian (Burghouwt & Veldhuis, 2006).

## 2.10 Diagram Use Case

*Diagram Use case* adalah penggambaran interaksi antara sistem dan pengguna atau sistem eksternal lainnya. Diagram ini juga biasa digunakan sebagai visualisasi pengguna yang terdapat pada *requirement* sehingga terlihat lebih mudah. Tabel 2.1 menjelaskan simbol-simbol yang terdapat pada diagram *use case*. Namun, biasanya simbol yang sering digunakan adalah simbol aktor, *use cases*, dan *association relationship* (Hamilton & Miles, 2006).






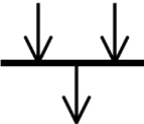
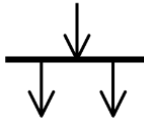
Tabel 2.1 Simbol Diagram *Use Case*

Simbol	Nama	Keterangan
	Aktor	Pengguna yang menggunakan sistem
	<i>Use Cases</i>	Interaksi yang dapat dilakukan pada sistem
	<i>Association Relationship</i>	Hubungan aktor dengan sistem

## 2.11 Activity Diagram

Diagram aktivitas berfungsi untuk menggambarkan alur kerja secara berurutan dan dikategorikan berdasarkan aktor. Diagram ini digunakan setelah diagram *use case* didefinisikan. Adapun aturan penggunaan diagram aktivitas seperti pada Tabel 2.2

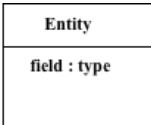





Tabel 2.2 Simbol *Activity Diagram*



Simbol	Nama	Keterangan
	Aktivitas	Representasi dari aksi
	<i>Initial State</i>	Awal dari sebuah aktivitas
	<i>Final State</i>	Akhir dari sebuah aktivitas
	<i>Control flow</i>	Menghubungkan aktivitas sebelum dan sesudah
	<i>Swimlane</i>	Kumpulan aktivitas yang dilakukan oleh seorang aktor
	<i>Join Node</i>	Menggabungkan aksi yang berjalan secara <i>concurrent</i>
	<i>Fork Node</i>	Membagi aksi menjadi satu set paralel yang dapat berjalan secara <i>concurrent</i>

## 2.12 Entity Relationship Diagram

*Entity-Relationship* (ER) adalah entitas yang model dasar dan model hubungan entitas yang diperluas atau ditingkatkan (*Enhanced Entity Relational*) sebagai model yang mencakup konsep tambahan. Model ER dan EER sering digunakan untuk membantu komunikasi antara perancang dan pengguna pada tahap analisis kebutuhan. Berikut merupakan simbol dari ER diagram. Adapun aturan penggunaan *Entity Relationship* seperti pada Tabel 2.3 (Byrne & Qureshi, 2013)

Tabel 2.3 Simbol Relationship Diagram Database

Simbol	Nama	Keterangan
	<i>Entity, Field and Type</i>	1. <i>Entity</i> : entitas tabel 2. <i>Field</i> : mewakili bagian dari tabel 3. <i>Type</i> : tipe dari <i>field</i>
	<i>Primary key</i>	<i>Field</i> yang unik yang menjadi kunci dari sebuah tabel
	<i>One</i>	Kardinalitas hubungan ke satu
	<i>Many</i>	Kardinalitas hubungan ke banyak
	<i>One (and only one)</i>	Kardinalitas hubungan ke satu dan hanya satu
	<i>Zero or one</i>	Kardinalitas hubungan ke satu atau kosong (opsional)

Simbol	Nama	Keterangan
	<i>One or many</i>	Kardinalitas hubungan ke satu ( <i>absolute</i> ) atau ke banyak
	<i>Zero or many</i>	Kardinalitas hubungan ke kosong (opsional) atau ke banyak

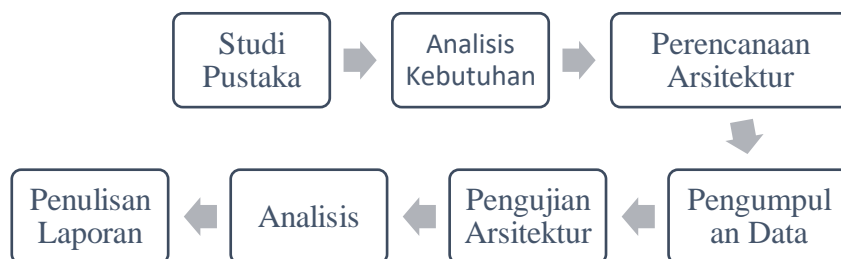
## BAB III

### ANALISIS DAN DESAIN

Pada bab ini diuraikan mengenai desain sistematika penelitian mengenai analisis performa *monolithic* dan *microservices architecture*. Pada bab ini juga akan diuraikan mengenai alat dan bahan yang digunakan selama proses penelitian.

#### 3.1 Desain Sistematika Penelitian

Dalam pelaksanaan penelitian diperlukan acuan dasar pelaksanaan agar penelitian dapat berjalan dengan baik, sehingga dapat diimplementasikan dan memberikan hasil yang diharapkan. Bagan pada Gambar 3.1 merupakan desain penelitian yang menggambarkan tahap dari kegiatan penelitian yang akan dilakukan.



Gambar 3.1 Desain Sistematika Penelitian

Penelitian ini dilakukan secara sistematis melalui beberapa tahap yaitu: studi pustaka, analisis kebutuhan aplikasi yang diuji, perencanaan arsitektur, pengumpulan data, pengujian arsitektur, analisis, dan penyusunan laporan.

### 3.1.1 Studi Pustaka

Pada tahap ini akan dilakukan studi pustaka terhadap penelitian-penelitian yang terkait dengan *monolithic* dan *microservices architecture*. Pada tahap ini dipelajari konsep arsitektur perangkat lunak yang telah mencapai *state-of-the-art* sehingga dapat menjadi rujukan dalam merancang dan membangun arsitektur dan analisa performa *monolithic* dan *microservices architecture*.

Pada tahap ini juga dipelajari bagaimana cara melakukan *granularity* layanan dari *monolithic* ke *microservices architecture* sehingga diharapkan mendapatkan *granularity* yang optimal.

### 3.1.2 Analisis Kebutuhan Aplikasi

#### 3.1.2.1 Kebutuhan aplikasi

Kebutuhan aplikasi yang akan di uji yaitu sebagai berikut:

##### 1. *Profile*

Fitur ini adalah fitur untuk melihat *profile* dari pengguna aplikasi. Data dari fitur ini juga digunakan untuk mendapatkan informasi dari halaman lain sebagai *authentication*.

##### 2. *Course*

Fitur ini adalah fitur untuk melihat deskripsi mata kuliah. Pada fitur ini pengguna akan mendapatkan informasi tentang mata kuliah, asisten, kehadiran, informasi tugas, dan fail-fail praktikum. Pengguna akan mendapatkan informasi ini pada laman *course*.

### **3. *Information***

Fitur ini adalah fitur untuk mendapatkan informasi dari semua aktivitas praktikum. Informasi yang akan didapat meliputi informasi pindah ruangan, informasi libur praktikum dan informasi umum yang lainnya.

### **4. *Assignment***

Fitur ini adalah fitur untuk mendapatkan informasi *assignment* pada praktikum. Pada fitur ini pengguna bisa mengunggah dan mengunduh fail yang diberikan oleh asisten maupun mengunduh fail yang diunggah oleh mahasiswa tersebut.

### **5. *Chatbot***

Fitur ini adalah fitur untuk mendapatkan semua informasi seperti *assignment*, nilai, asisten yang mengajar, dan informasi umum lainnya yang berkaitan dengan praktikum.

#### **3.1.2.2 *Kebutuhan fungsional***

Kebutuhan fungsional berisi proses-proses layanan yang harus disediakan oleh sistem, mencakup bagaimana sistem harus bereaksi pada *input* tertentu dan bagaimana perilaku sistem pada situasi tertentu. Tabel 3.1 merupakan *list* dari kebutuhan Fungsional Owl Assistant.



Tabel 3.1 Tabel Kebutuhan fungsional Owl Assistant

No	Nama	Kode	Deskripsi	Prioritas	Risiko
1	<i>View Assignment</i>	Owl-FR/01	Pengguna melihat seluruh daftar tugas	<i>High</i>	Data tidak terbaca karena kendala koneksi internet, atau <i>server down</i>
2	<i>View Detail Course</i>	Owl-FR/02	Pengguna melihat daftar praktikum yang sudah, belum, dan akan diambil	<i>High</i>	Data tidak terbaca karena kendala koneksi internet, atau <i>server down</i>
3	<i>View Assignment Course</i>	Owl-FR/03	Pengguna melihat daftar tugas pada <i>course</i> tertentu	<i>High</i>	Data tidak terbaca karena kendala koneksi internet, atau <i>server down</i>
4	<i>View Detail Information</i>	Owl-FR/04	Pengguna melihat informasi tertentu	<i>High</i>	Data tidak terbaca karena kendala koneksi internet, atau <i>server down</i>
5	<i>Setting Profile</i>	Owl-FR/05	Pengguna mengubah data <i>profile</i>	<i>High</i>	Data tidak terkirim karena kendala koneksi internet, atau <i>server down</i>
6	<i>Bot</i>	Owl – FR/ 06	Pengguna menanyakan informasi umum kepada bot	<i>High</i>	Data tidak terkirim karena kendala koneksi internet, atau <i>server down</i>

### 3.1.2.3 Kebutuhan non fungsional

Kebutuhan non fungsional menitikberatkan pada properti perilaku yang dimiliki oleh sistem. Kebutuhan non fungsional ini sebagai batasan layanan atau fungsi yang ditawarkan sistem seperti batasan waktu, batasan pengembangan proses, standardisasi dan lain-lain. Tabel 3.2 merupakan Kebutuhan Non Fungsional Owl Assistant

Tabel 3.2 Kebutuhan Non Fungsional Owl Assistant

No.	Kode	Parameter	Deskripsi
1	Owl-NFR/01	<i>Availability</i>	Ketersediaan aplikasi untuk dapat diakses pengguna
2	Owl-NFR/02	<i>Reliability</i>	Keandalan aplikasi, termasuk aspek seperti koneksi
3	Owl-NFR/02	<i>Portability</i>	Aplikasi mudah diakses di berbagai <i>platform</i>
4	Owl-NFR/02	<i>Response time</i>	Waktu aplikasi untuk merespons <i>request</i> dari <i>user</i>
5	Owl-NFR/02	<i>Security</i>	Enkripsi data pada komunikasi aplikasi

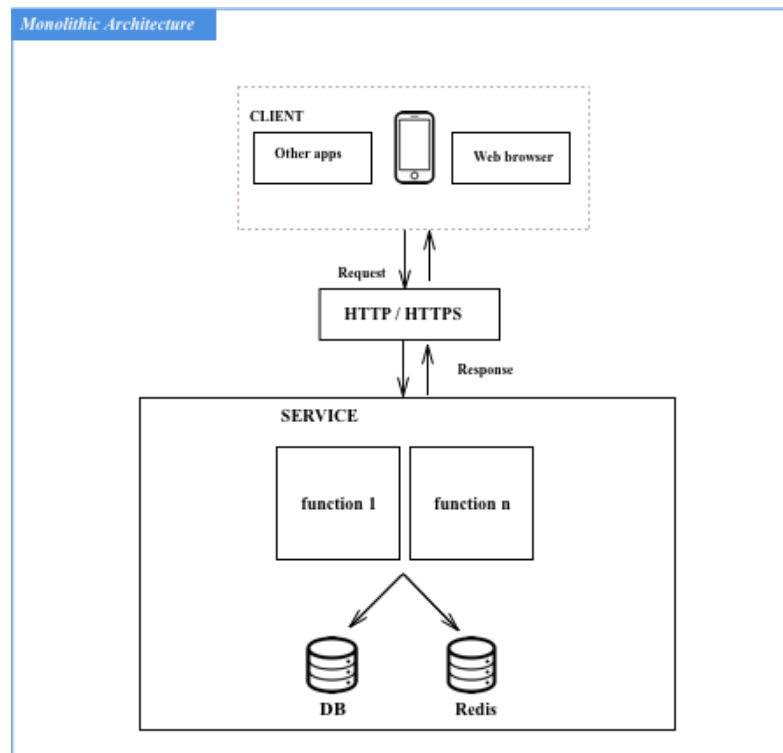
### 3.1.3 Perencanaan Arsitektur

Perancangan Arsitektur memperhatikan studi pustaka yang telah dilakukan penulis, berikut merupakan perencanaan arsitektur yang dilakukan di kedua arsitektur:

#### 3.1.3.1 Desain *Monolithic Architecture*

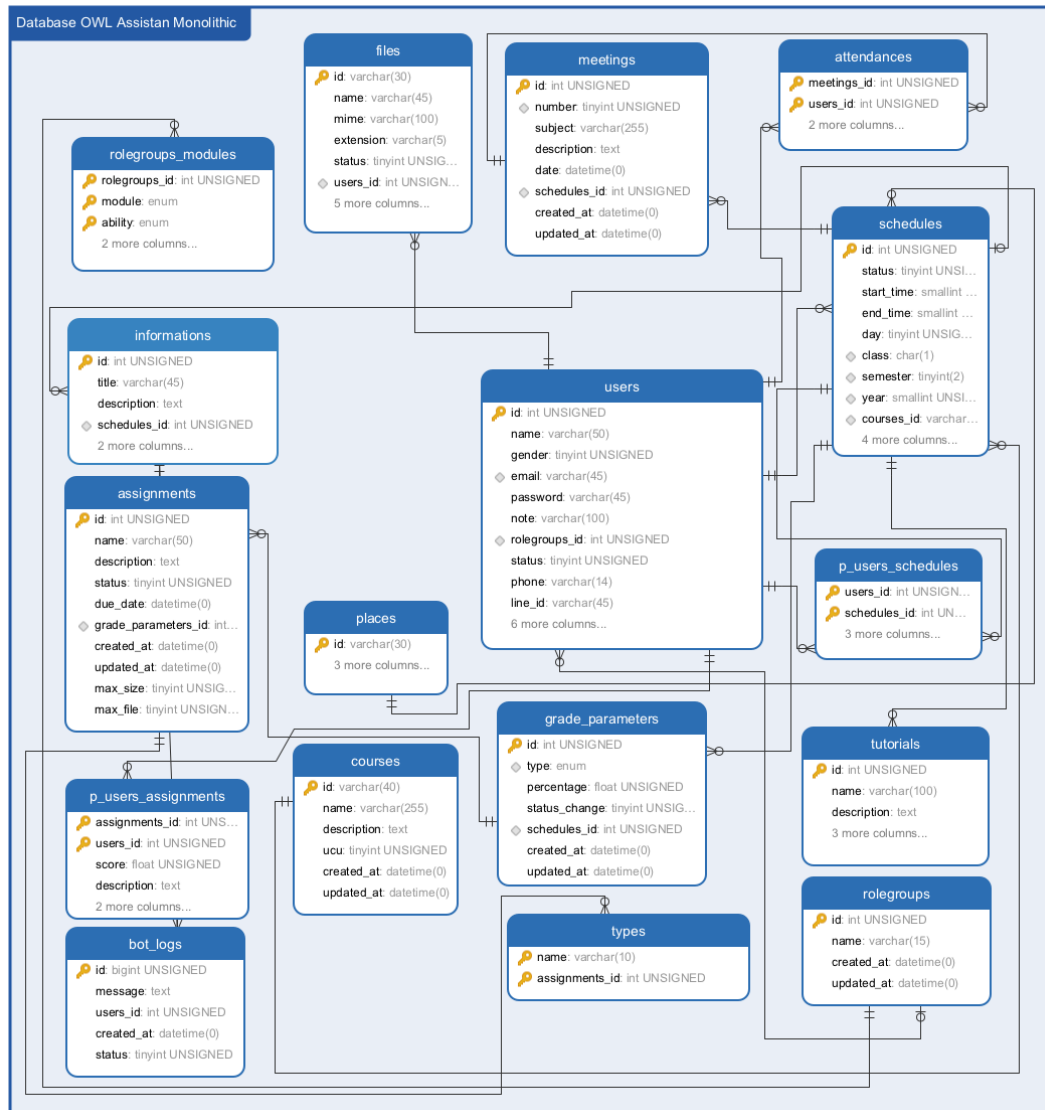
Pada *monolithic architecture*, tidak banyak hal yang harus di perhatikan, karena struktur *monolithic* bersifat masif dan pasti, artinya semua komponen yang

disebut fungsi disimpan dalam satu bundel fail. Komponen atau fitur yang terdapat pada kebutuhan fungsional Tabel 3.1 dikembangkan dan disebar dalam satu fail.



Gambar 3.2 Desain *Monolithic Architecture* Owl Assistant

Owl Assistant merupakan aplikasi *multiplatform* di mana klien bisa mengakses aplikasi dari berbagai *platform*, dari bagan Gambar 3.2 klien bisa mengakses aplikasi dari *web browser*, *mobile phone*, dan aplikasi lainnya seperti Postman. Kemudian klien melakukan *request* atau mengakses aplikasi melalui *http/https* ke *server*, dan *service* akan menerima request tersebut kemudian memberikan *response* melalui *http/https gateway*. Pada *service* terdapat fungsi 1 dan fungsi n, yang merupakan *instance* dari fitur-fitur yang tersedia, semua fitur dan fungsi pendukung lainnya dijadikan menjadi satu layanan. Kemudian *database* aplikasi mempunyai struktur yang lengkap untuk memenuhi fitur-fitur yang tersedia.



Gambar 3.3 Desain Database Owl Assistant - *Monolithic Architecture*

Gambar 3.3 merupakan desain *database* Owl Assistant, *database* tersebut berdiri sendiri sebagai satu *database* yang utuh. Jumlah tabel pada *database* tersebut yaitu 17 tabel. Semu tabel mempunyai relasi dengan tabel lainnya kecuali tabel *bot\_logs*. Informasi lengkap dari Gambar 3.3 ditunjukkan dengan kamus data dari pada Tabel 3.3 – Tabel 3.17

Tabel 3.3 Kamus Data Tabel *Assignment*

<b>Tabel <i>assignments</i></b>		
No.	Kolom	Tipe Data
1	<i>id</i>	int(10), <i>primary key</i> , not null
2	status	tinyint(3), not null
3	due_date	datetime
4	grade_paramters_id	int(11), referenced grade_parameters(id), not null
5	description	text
6	created_at	datetime
7	updated_at	datetime
8	name	varchar(50), not null
9	size	int(11)
10	type	varchar

Tabel 3.3 merupakan kamus data dari tabel *assignments* yang digunakan untuk menyimpan tugas-tugas yang dibuat oleh asisten. Tabel 3.3 mempunyai relasi dengan tabel *grade\_parameters\_id* untuk perhitungan nilai dari setiap *course* yang diambil.

Tabel 3.4 Kamus Data Tabel *Attendance*

<b>Tabel <i>attendances</i></b>		
No.	Kolom	Tipe Data
1	meetings_id	int(10), primary key 1, referenced meetings_id(id), not null
2	users_id	tinyint(3), primary key 2, referenced users(id) , not null
3	created_at	datetime
4	updated_at	datetime

Tabel 3.4 merupakan kamus data dari tabel *attendances* yang digunakan untuk menyimpan daftar hadir mahasiswa yang dibuat oleh asisten. Tabel ini mempunyai relasi dengan tabel *users* dan tabel *meetings* untuk mengisi entitas user dan pertemuan dari setiap kehadiran mahasiswa.

Tabel 3.5 Kamus Data Tabel *Bot\_Logs*

<b>Tabel bot_logs</b>		
No.	Kolom	Tipe Data
1	id	bigint(20), primary key, not null
2	message	text, not null
3	users_id	int(10), not null
4	created_at	datetime
5	status	tinyint, not null

Tabel 3.5 merupakan kamus data dari tabel *bot\_logs* yang digunakan untuk menyimpan log percakapan antara pengguna baik itu asisten ataupun mahasiswa dengan bot. Tabel ini mempunyai relasi dengan tabel *users* untuk mengidentifikasi percakapan siapa yang masuk jadi dalam log.

Tabel 3.6 Kamus Data Tabel *Course*

<b>Tabel courses</b>		
No.	Kolom	Tipe Data
1	id	varchar(40), not null
2	name	varchar(255), not null
3	description	text
4	ucu	tinyint(2), not null
6	created_at	datetime
7	updated_at	datetime

Tabel 3.6 merupakan kamus data dari tabel *courses* yang digunakan untuk menyimpan *course* yang tersedia atau di tambahkan oleh asisten. Tabel ini tidak mempunyai relasi dengan tabel lain karena merupakan entitas yang unik.

Tabel 3.7 Kamus Data Tabel *Files*

Tabel <i>files</i>		
No.	Kolom	Tipe Data
1	id	int(30), not null, primary key
2	name	varchar(45), not null
3	mime	varchar(100), not null
4	extension	varchar(5), not null
5	status	tinyint(1), not null
6	users_id	int(10), referenced users(id), not null
7	type	varchar(10) (0=deleted, 1=exist)
8	table_name	varchar(45),
9	table_id	varchar(20)
10	created_at	datetime
11	updated_at	datetime

Tabel 3.7 merupakan kamus data dari tabel *files* yang digunakan untuk menyimpan data *file* yang ditambahkan oleh mahasiswa ataupun asisten. Tabel ini mempunyai relasi dengan tabel *users* untuk mengidentifikasi siapa yang mempunyai informasi *file* tersebut.

Tabel 3.8 Kamus Data Tabel Grade

<b>Tabel grade_parameters</b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key, not null
2	type	enum, not null
3	percentage	float(5,2), not null
4	status_change	tinyint(1), not null
5	schedule_id	int(10), referenced schedules(id), unique, not null
6	created_at	datetime
7	updated_at	datetime

Tabel 3.8 merupakan kamus data dari tabel *grade\_paramters* yang digunakan untuk menyimpan bobot parameter dari *course* seperti UAS, QUIZ, dan UTS. Tabel ini mempunyai relasi dengan tabel *course* untuk mengidentifikasi *grade parameter* yang dibuat adalah untuk *course* yang berelasi.

Tabel 3.9 Kamus Data Tabel Information

<b>Tabel informations</b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key, not null
2	title	varchar(45)
3	description	text
4	schedule_id	int(10), referenced schedules(id)
5	created_at	datetime
6	updated_at	datetime

Tabel 3.9 merupakan kamus data dari tabel *Information* yang digunakan untuk menyimpan informasi umum ataupun dari *course*. Tabel ini mempunyai



relasi dengan tabel *schedule* untuk mengidentifikasi mata kuliah apa yang terkait dengan informasi tersebut namun relasi ini tidak bersifat wajib atau bisa NULL untuk menandakan informasi tersebut bersifat menyeluruh.

Tabel 3.10 Kamus Data Tabel *Meetings*

<b>Tabel <i>meetings</i></b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key, not null
2	number	tinyint(3), not null
3	subject	varchar(255), not null
4	description	text
5	date	datetime
6	schedule_id	int(10), referenced schedules(id), not null
7	created_at	datetime
8	updated_at	datetime

Tabel 3.10 merupakan kamus data dari tabel *meetings* yang digunakan untuk menyimpan informasi pertemuan kehadiran. Tabel ini mempunyai relasi dengan tabel *schedule* untuk mengidentifikasi mata kuliah apa yang terkait dengan informasi pertemuan tersebut.

Tabel 3.11 Kamus Data Tabel *Users Schedule*

<b>Tabel <i>p_users_schedules</i></b>		
No.	Kolom	Tipe Data
1	users_id	int(10), primary key 1, referenced users(id), not null
2	schedules_id	int(10), primary key 2, referenced schedules(id), not null
3	status	tinyint(3), not null
4	created_at	datetime
5	updated_at	datetime

Tabel 3.11 merupakan kamus data dari tabel *p\_users\_schedules* yang digunakan untuk menyimpan informasi *user* yang mengambil mata kuliah dari mata kuliah yang tersedia pada tabel *course*. Tabel ini merupakan tabel pivot antara *users* dan *schedules* yang mempunyai relasi *many to many*.

Tabel 3.12 Kamus Data Tabel *Places*

<b>Tabel <i>places</i></b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key, referenced users(id), not null
2	descriptions	text
3	created_at	datetime
4	updated_at	datetime

Tabel 3.12 merupakan kamus data dari tabel *places* yang digunakan untuk menyimpan informasi tempat mata kuliah. Tabel ini tidak tergantung dengan tabel lainnya karena merupakan entitas yang unik.

Tabel 3.13 Kamus Data Tabel *Role Groups*

<b>Tabel <i>rolegroups</i></b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key , not null
2	name	varchar(15), not null
3	created_at	datetime
4	updated_at	datetime

Tabel 3.13 merupakan kamus data dari tabel *rolegroups* yang digunakan untuk menyimpan informasi *roles* yang di tambahkan oleh *admin*. Tabel ini tidak tergantung dengan tabel lainnya karena merupakan entitas yang unik.

Tabel 3.14 Kamus Data Tabel *Role Group Modules*

<b>Tabel rolegroups_modules</b>		
No.	Kolom	Tipe Data
1	rolegroups_id	int(10), primary key 1, referenced rolegroups(id), not null
2	module	enum, primary key 2, not null
3	ability	enum, primary key 3, not null
4	created_at	datetime
5	updated_at	datetime

Tabel 3.14 merupakan kamus data dari tabel *rolegroups\_modules* yang digunakan untuk menyimpan informasi *module* apa saja yang bisa diakses oleh *role* tersebut. Tabel ini mempunyai relasi dengan tabel *role groups* untuk mengidentifikasi *module* ada terkait dengan *role group* yang berelasi.

Tabel 3.15 Kamus Data Tabel *Tutorials*

<b>Tabel tutorials</b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key, not null
2	name	varchar(100), not null
3	description	text
4	schedules_id	int(10), referenced schedules(id), not null
5	created_at	datetime
6	updated_at	datetime

Tabel 3.15 merupakan kamus data dari tabel *tutorials* yang digunakan untuk menyimpan informasi *file* tutorial yang di *input* oleh *admin*. Tabel ini mempunyai relasi dengan tabel *schedules* untuk mengidentifikasi tutorial tersebut berkaitan dengan *schedules* yang berkaitan.

Tabel 3.16 Kamus Data Tabel *Schedules*

<b>Tabel <i>schedules</i></b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key, not null
2	status	tinyint(4), not null
3	start_time	smallint(5), not null
4	end_time	smallint(5), not null
5	day	tinyint(1), not null
6	class	char(1), unique, not null
7	semester	tinyint(2), unique, not null
8	year	smallint(4), unique, not null
9	courses_id	varchar(40), unique, referenced courses(id), not null
10	places_id	varchar(30), referenced places(id), not null
11	created_by	int(10), referenced users(id), not null
12	created_at	datetime
13	updated_at	datetime

Tabel 3.16 merupakan kamus data dari tabel *schedules* yang digunakan untuk menyimpan informasi *schedule* yang bisa di ambil oleh mahasiswa. Tabel ini

mempunyai relasi dengan tabel *courses* dan *places* mendapatkan informasi lengkap terkait dengan *course* dan tempat praktikum.

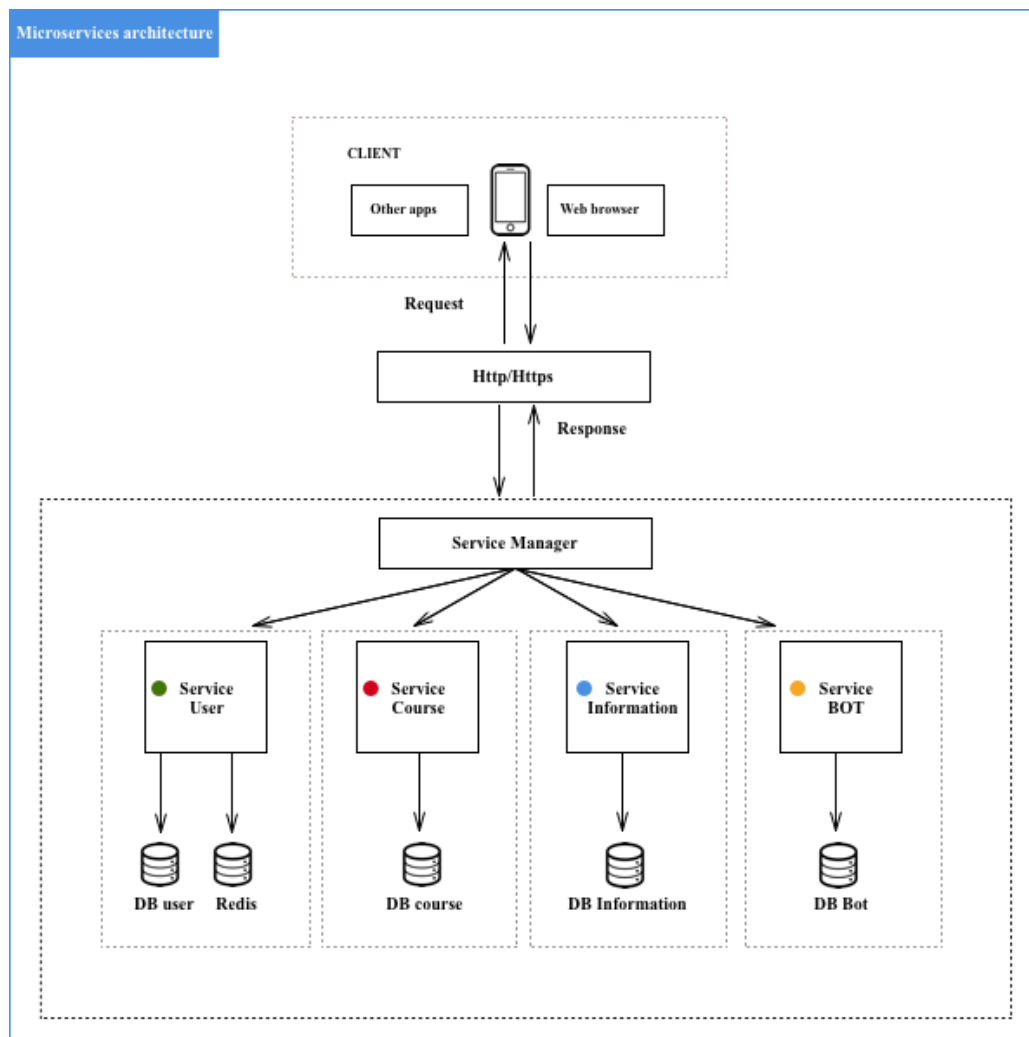
Tabel 3.17 Kamus Data Tabel *Users*

<b>Tabel <i>users</i></b>		
No.	Kolom	Tipe Data
1	id	int(10), primary key, not null
2	name	varchar(50), not null
3	gender	texttinyint(3), not null
4	email	varchar(45), unique, not null
5	password	varchar(45), not null
6	note	varhchar(100), not null
7	rolegroups_id	int(11), referenced rolegroups(id)
8	status	tinyint(3), not null
9	phone	varchar(14)
10	line_id	varchar(45)
11	identity_code	varchar(18), unique, not null
12	email_verification_code	smallint(4)
13	email_verification_expire_date	datetime
14	email_verification_attempt	tinyint(10)
15	created_at	datetime
16	updated_at	datetime

Tabel 3.17 merupakan kamus data dari tabel *uses* yang digunakan untuk menyimpan informasi *users* baik itu asisten ataupun mahasiswa. Tabel ini mempunyai relasi dengan tabel *role groups* untuk mengidentifikasi *privilege user* terhadap sistem.

### 3.1.3.2 Desain *Microservices Architecture*

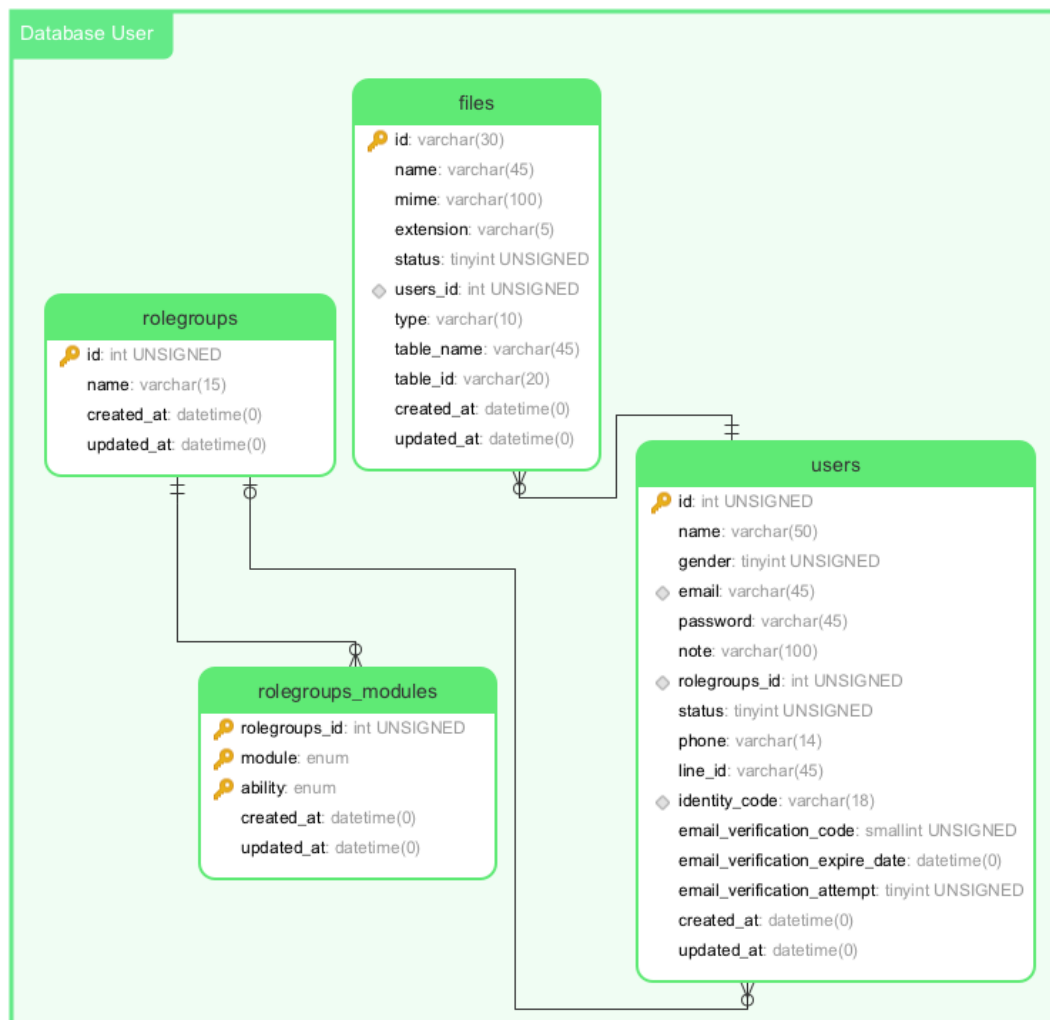
Bagian terpenting dari *microservices* yaitu *Granularity* layanan dari fitur yang ada. Berdasarkan *granularity definition* yang berada pada studi pustaka penulis memartisi layanan menjadi 4 layanan, yaitu layanan *user*, layanan informasi, layanan *course* atau mata kuliah, dan layanan bot.



Gambar 3.4 Desain *Microservices Architecture* Owl Assistant

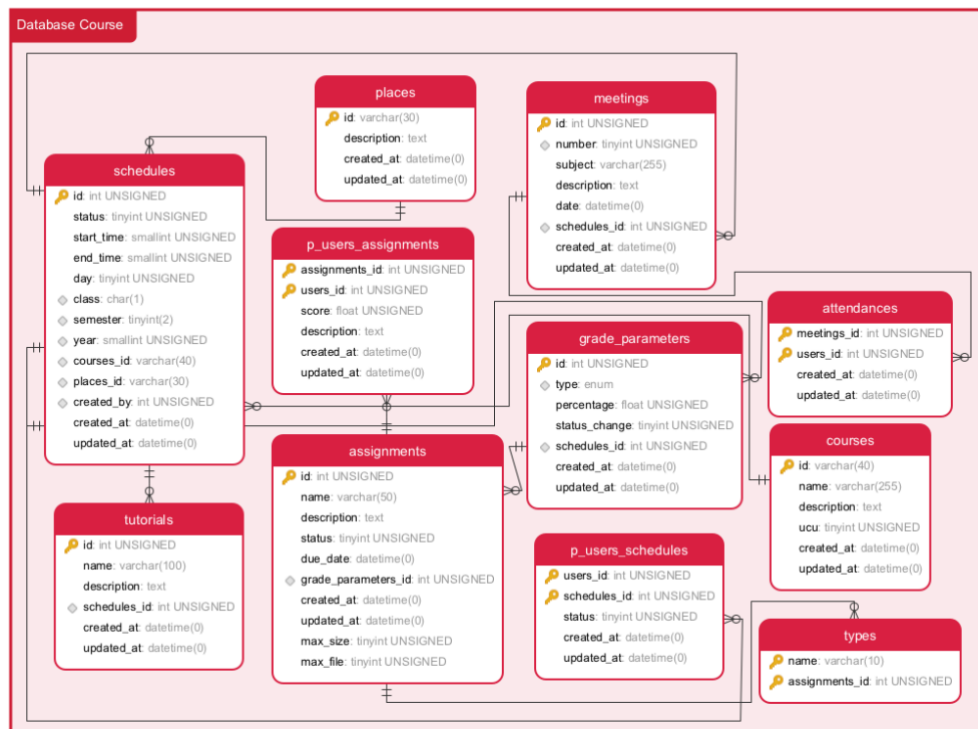
Seperti halnya *monolithic*, klien hanya akan *request* pada satu URL melalui *http/https gateway*, kemudian di dalam satu layanan, jika satu layanan membutuhkan data dari layanan lainnya, maka layanan tersebut akan membuat

*request* ke layanan yang lainnya. Contohnya, pada layanan *course* ketika membutuhkan daftar *user* yang mengambil *course* tersebut maka layanan *course* akan meminta data dengan mengakses layanan *user* melalui *API Gateway*. *Database* yang tadinya merupakan satu kesatuan sekarang di bagi menjadi 4 bagian, yaitu *database* untuk layanan *user*, *course*, *information*, dan *bot*. Berikut merupakan desain *database* dari masing-masing layanan:



Gambar 3.5 Desain Database Service User

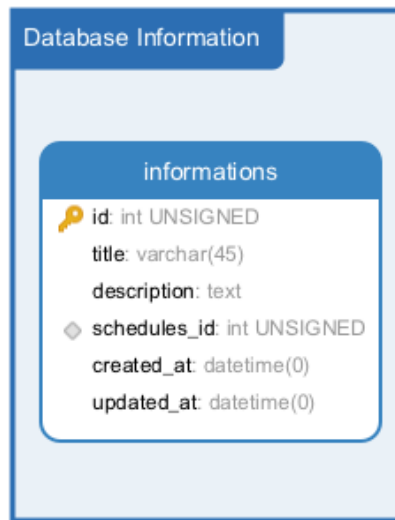
Pada gambar 3.5 merupakan desain *database* untuk *service user*, dari *database* tersebut tidak ada relasi sama sekali dengan *database* yang lainnya atau bersifat independen. Jumlah tabel pada *database user* ini yaitu 9 tabel dan masing-masing tabel mempunyai relasi dengan tabel lainnya.



Gambar 3.6 Desain *Database Service Course*

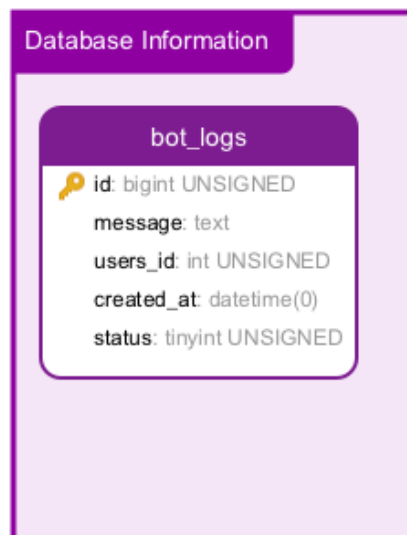
Gambar 3.6 merupakan desain *database* dari *service course*, *database course* juga tidak mempunyai relasi dengan tabel dari *database service* lain atau bersifat *independent*. Jumlah tabel pada *database course* 11 tabel dan masing-masing tabel berelasi satu sama lain dalam satu *database*.





Gambar 3.7 Desain *Database Service Information*

Gambar 3.7 merupakan desain dari *database service information* dan tidak mempunyai relasi dengan *database* yang lainnya atau independen. *Database* ini digunakan untuk menyimpan informasi yang telah dibuat oleh *admin*.



Gambar 3.8 Desain *Database Service Bot*

Gambar 3.8 merupakan desain dari *database* bot *service* dan tidak mempunyai relasi dengan *database* yang lainnya atau independen. *Database* ini digunakan untuk log dari setiap percakapan antara bot dengan pengguna.

### 3.1.4 Pengujian Arsitektur

Desain yang sudah di bangun, kemudian diimplementasi dan di uji, pengujian dilakukan di tahap *testing*, dengan menggunakan Apache Jmeter, adapun indeks pengukuran performa dari setiap arsitektur yaitu *request time* dan *resource utilization*. Fungsi yang akan di testing hanya beberapa fungsi saja yaitu fungsi *assignment* yang mempunyai data sebanyak 10.000.000 baris, *users* yang mempunyai data sebanyak 100.000 baris, *course* yang mempunyai data sebanyak 100.000 baris dan kemudian informasi yang mempunyai data sebanyak 10.000.000 baris.

### 3.1.5 Alat dan Bahan Penelitian

Dalam pengujian performa *monolithic* dan *microservices architecture* dibutuhkan perangkat keras dan perangkat lunak sebagai berikut:

#### 3.1.5.1 Alat Penelitian

Pada penelitian ini digunakan alat penelitian berupa perangkat keras dan perangkat lunak untuk pengujian performa *monolithic* dan *microservices architecture*.

#### 1. Perangkat keras

Adapun spesifikasi perangkat keras yang digunakan pada penelitian ini adalah sebagai berikut:

- *Processor* : Intel Core i5 1,6 GHz

- GPU : Intel HD *Graphics* 6000
- RAM : 4GB DDR3L
- *Storage* : 128 GB

## 2. Perangkat lunak

Perangkat lunak yang digunakan pada penelitian ini adalah sebagai berikut:

- Sistem Operasi : MacOS High Sierra 10.13.2
- Aplikasi : Visual Studio Code 1.18.1, Sketch, Navicat
- Bahasa Pemrograman : Go 1.8.3
- *Database* : MySQL 5.7.20
- *Cloud Service* Dengan spesifikasi :
  - Prosesor: 1 Core(s) Generation II General Type n1
  - *Memory*: 1 GB
  - *Disk*: 1
  - *Network Type*: VPC (*Virtual Private Cloud*)
  - *Bandwidth Peak*: 200Mbps *Data Transfer*
  - *Region*: Asia Pacific SE 1 (*Singapore*)
  - *Operating System*: Centos 7

### 3.1.5.2 Bahan Penelitian

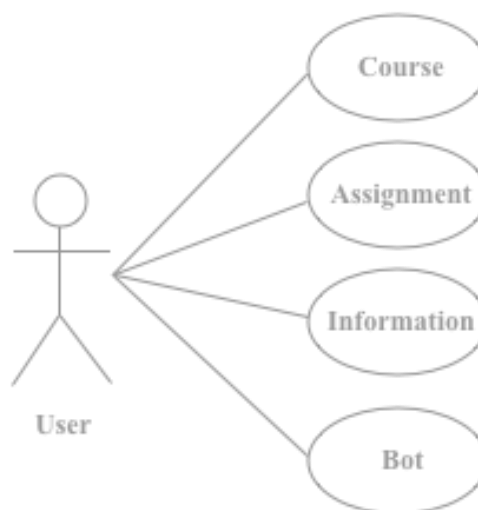
Bahan penelitian didapatkan dari hasil pengisian *database* secara otomatis menggunakan *tools* bantuan, data yang dihasilkan merupakan *dummy data*. Pada tabel 3.18 merupakan jumlah data yang tersedia pada *database*.

Tabel 3.18 Jumlah data yang tersedia pada *database*

No.	Nama Tabel	Nama DB	Jumlah Data (baris)
.1.	<i>Assignments</i>	<i>courses</i>	100.000
.2.	<i>Users</i>	<i>users</i>	100.000
.3.	<i>Schedules</i>	<i>courses</i>	100.000
.4.	<i>Courses</i>	<i>courses</i>	100.000
.5.	<i>P_users_assignments</i>	<i>courses</i>	10.000.000
.6.	<i>Informations</i>	<i>informations</i>	100.000

### 3.1.6 Use Case Diagram

Diagram *use case* digunakan untuk mendeskripsikan hubungan antara aktor dengan aktivitas yang dapat dilakukan pada sistem aplikasi. Sasaran *use case diagram* ini yaitu sebagai referensi dalam perancangan kebutuhan fungsional ataupun perancangan antarmuka.



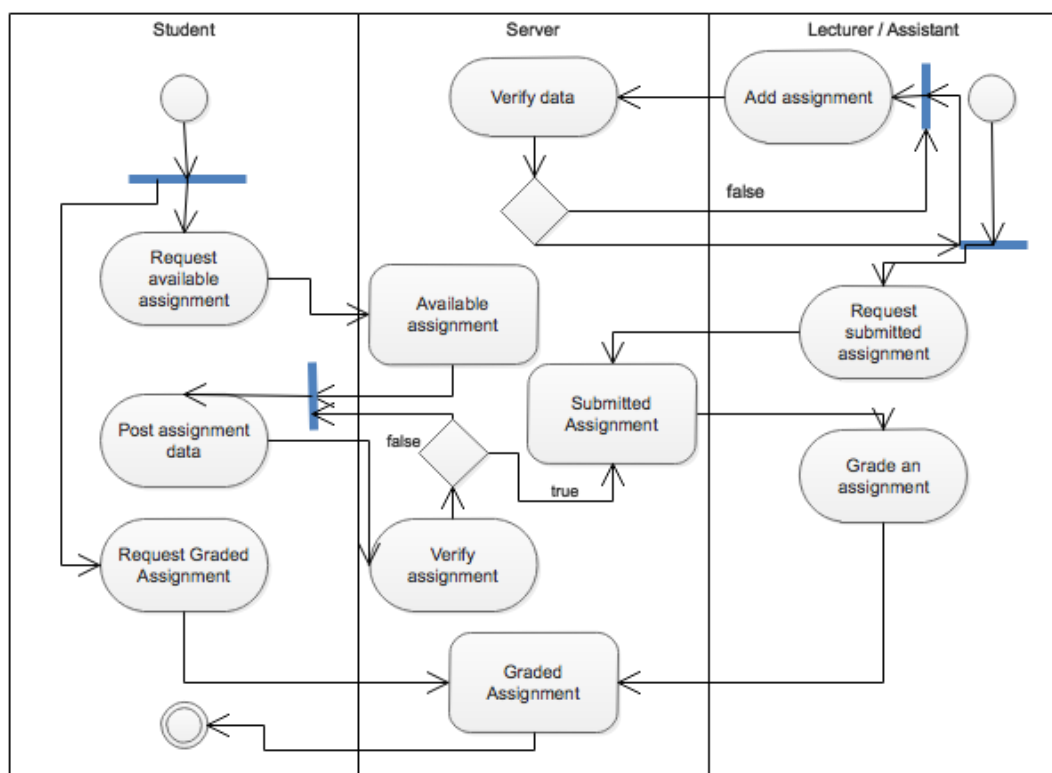
Gambar 3.9 Use Case Diagram Owl Assistant

Berdasarkan diagram *use case* pada Gambar 3.9, hanya terdapat aktor pengguna, yaitu mahasiswa dan asisten. Semuanya mempunyai akses untuk *Course, Assignment, Information, dan Bot*. Namun untuk manajemen *course, assignment, information, dan user* hanya asisten.

### 3.1.7 Activity Diagram

Diagram aktivitas berfungsi untuk mendefinisikan bisnis proses secara rinci mengenai aktivitas yang dapat dilakukan pada aplikasi. Aktivitas yang dapat dilakukan pada aplikasi yang uji ini yaitu *Get assignment, get information, get user profile, get course, dan get bot history*.

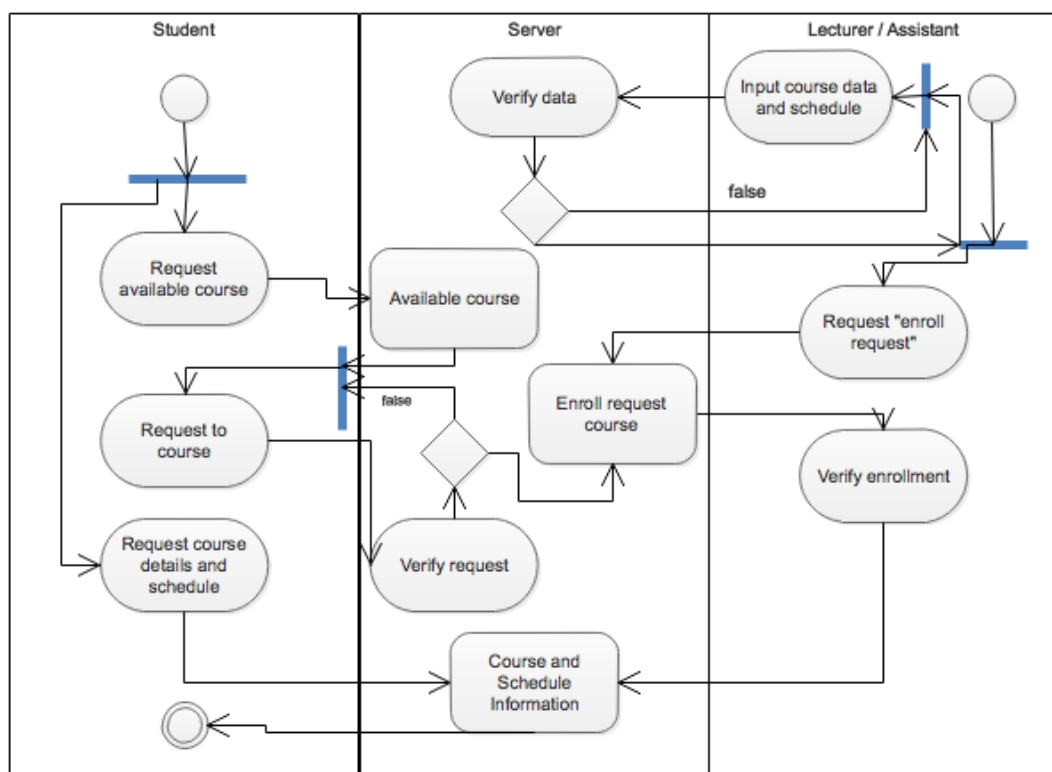
#### 3.1.7.1 Activity Assignment



Gambar 3.10 Activity Diagram - Assignment and Grade

Gambar 3.10 merupakan proses untuk mengakses *assignment*. Proses dimulai dari mahasiswa mengakses *assignment* yang tersedia, kemudian mahasiswa tersebut akan mengakses *assignment detail* untuk kemudian mengunggah *assignment* yang sudah dikerjakan.

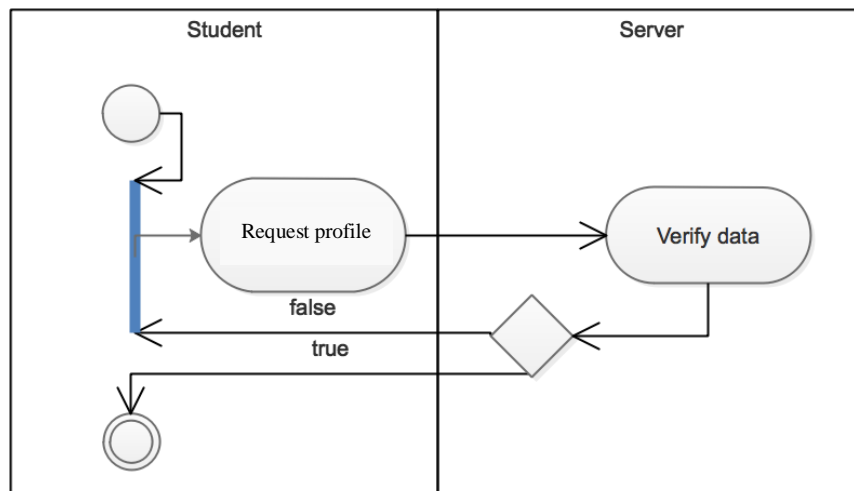
### 3.1.7.2 Activity Course Enroll dan Shedule Information



Gambar 3.11 Activity Diagram - Course and Schedule Information

Gambar 3.11 merupakan proses *get profile*. Proses dimulai dari mahasiswa melakukan *request* ke *course* yang tersedia ke server, setelah data *list course* diberikan oleh server, mahasiswa mengakses detail informasi *course* untuk melihat perkembangan informasi dari *course* yang dipilih atau di ikuti.

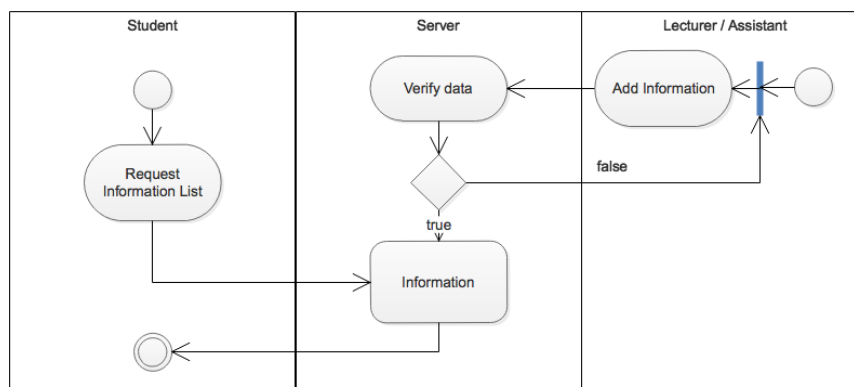
### 3.1.7.3 Activity Profile



Gambar 3.12 Activity Diagram – Edit Profile

Gambar 3.12 merupakan proses *get* profil. Proses dimulai dari mahasiswa melakukan *request* ke server, jika *session* benar pada verifikasi yang dilakukan server, kemudian server akan memberikan respons data lengkap profil dari mahasiswa tersebut.

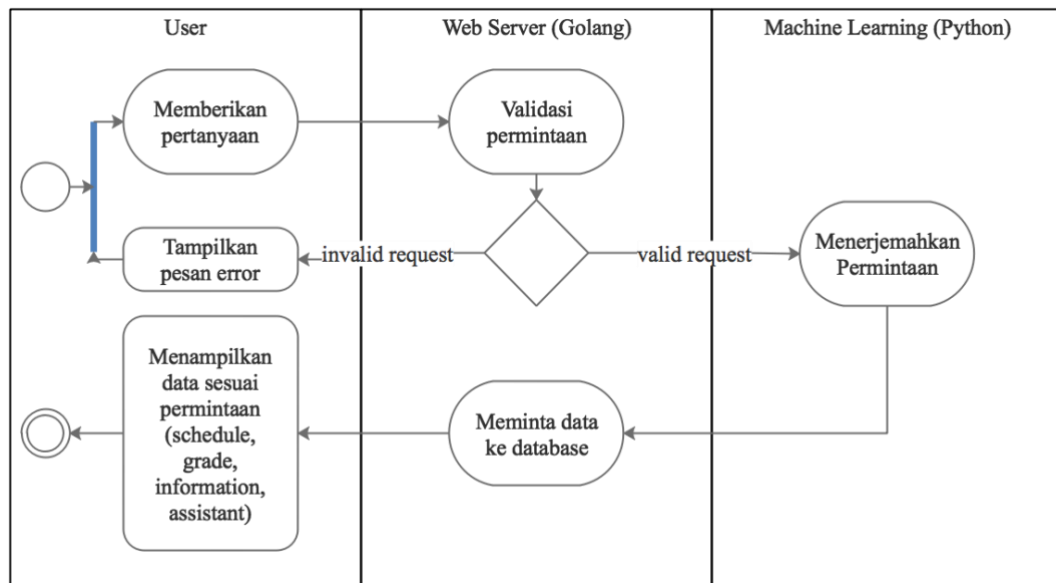
### 3.1.7.4 Activity Information List



Gambar 3.13 Activity Diagram – Information List

Gambar 3.13 merupakan proses mengakses informasi yaitu mahasiswa mengakses informasi yang sudah di tambahkan oleh asisten. Kemudian data di verifikasi, setelah di verifikasi informasi yang dibuat bisa di akses oleh mahasiswa.

### 3.1.7.5 Activity ChatBot



Gambar 3.14 Activity Diagram - Chatting Bot

Gambar 3.14 merupakan proses mengakses bot yaitu mahasiswa mengakses informasi seperti *assignment*, *course*, atau yang lainnya dengan memberikan pertanyaan kemudian pertanyaan tersebut akan di validasi, jika informasi yang ditanyakan valid, maka akan disampaikan ke penerjemah dan meminta data ke *database* untuk kemudian ditampilkan kepada mahasiswa sebagai respons



## **BAB IV**

### **HASIL DAN PEMBAHASAN**

Pada bab ini dibahas mengenai hasil penelitian, yaitu perbandingan performa aplikasi Owl Assistant menggunakan *monolithic* dan *microservices architecture* berdasarkan *request time* dan *resource utilization* (*CPU* dan *memory*).

#### **4.1 Spesifikasi Arsitektur**

Pada penelitian ini digunakan dua arsitektur yaitu *monolithic* dan *microservices architecture*. Spesifikasi keduanya menggunakan *resource* yang seimbang sehingga didapatkan hasil yang objektif.

##### **4.1.1 Spesifikasi *Monolithic architecture***

Pada *monolithic architecture* semua *resource* yang tersedia dipakai penuh untuk layanan tunggal aplikasi Owl Assistant, berikut merupakan spesifikasinya:

*CPU*: 1 Core(s)

*Memory*: 1 GB

*Bandwidth Peak*: 200 Mbps

##### **4.1.2 Spesifikasi *Microservices architecture***

Pada *microservices resource* yang tersedia dibagi menjadi 4 bagian untuk 4 layanan *microservices* yaitu layanan *user*, *course*, *information*, dan *bot*. *Granularity* yang dipilih yaitu berdasarkan definisi pembagian *service* yang terlampir pada tinjauan pustaka. Tabel 4.1 merupakan informasi pembagian *resource* setiap servisnya.

Tabel 4.1 Pembagian *resource* untuk setiap layanan

<b>Layanan</b>	<b>CPU (<i>Core</i>)</b>	<b>Memory (Mb)</b>	<b>Bandwidth (Mbps)</b>
<i>User</i>	0.3	300	50
<i>Course</i>	0.3	300	50
<i>Information</i>	0.2	200	50
<i>Bot</i>	0.2	300	50

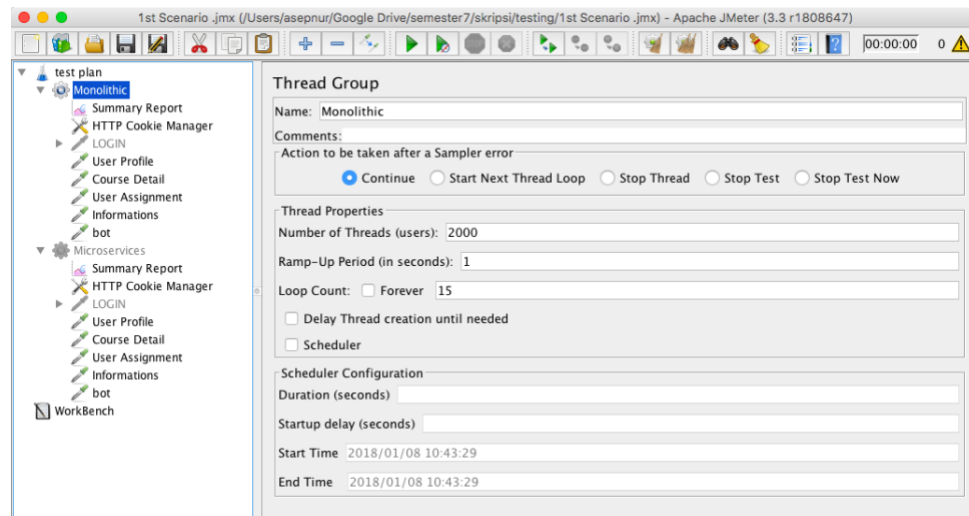
## 4.2 Rencana Uji

Pada penelitian ini digunakan rencana uji untuk pengujian performa aplikasi Owl Assistant menggunakan *monolithic* dan *microservices architecture* menggunakan aplikasi JMeter. properti rencana uji pengujian di informasikan pada Tabel 4.2

Tabel 4.2 Properti rencana uji

<b>Properti</b>	<b>Pengujian ke-</b>						
	1	2	3	4	5	6	7
Jumlah <i>user / request</i>	1	10	100	500	1000	1500	2000
Pengulangan	15	15	15	15	15	15	15

Properti jumlah *user* dan pengulangan yang di definisikan telah ada di aplikasi JMeter secara otomatis. Gambar 4.1 menunjukan antarmuka dari JMeter



Gambar 4.1 Antarmuka Jmeter untuk pengujian performa

Pengetesan dengan properti rencana uji pada Tabel 4.3 di kombinasikan dengan mengakses API. API yang tersedia di uji secara bersamaan baik itu pada *monolithic* ataupun *microservices architecture*. Berikut merupakan API yang di uji pada kedua arsitektur dan implementasi aplikasi API yang dipakai

Tabel 4.3 API yang digunakan untuk pengujian

<b>Method</b>	<b>API</b>	<b>Layanan</b>	<b>Aksi Database</b>
GET	/api/v1/user/profile	<i>Monolithic / user</i>	READ
GET	/api/admin/v1/course/:id	<i>Monolithic/ course</i>	READ
GET	/api/v1/assignment/:id	<i>Monolithic/ course</i>	READ
GET	/api/v1/information/:id	<i>Monolithic/ information</i>	READ
GET	/api/v1/bot	<i>Monolithic/ bot</i>	READ

#### 4.2.1 Web Service Utama

Kode dari aplikasi Owl Assistant pada *monolithic* dan *microservices* untuk kode utamanya hampir sama, perbedaannya yaitu jika pada *monolithic* Database,

Redis, Auth dan Email diinisialisasi sedangkan pada microservices hanya layanan user saja yang menginisialisasi semuanya. Berikut merupakan kode utama web servis

```
package main

import (
    "flag"
    "log"
    "runtime"

    "github.com/melodiez14/meiko/src/cron"
    "github.com/melodiez14/meiko/src/email"
    "github.com/melodiez14/meiko/src/util/alias"
    "github.com/melodiez14/meiko/src/util/auth"
    "github.com/melodiez14/meiko/src/util/conn"
    "github.com/melodiez14/meiko/src/util/env"
    "github.com/melodiez14/meiko/src/util/jsonconfig"
    "github.com/melodiez14/meiko/src/webserver"
    "github.com/melodiez14/meiko/src/webserver/handler/bot"
)

type configuration struct {
    Directory alias.DirectoryConfig `json:"directory"`
    Database  conn.DatabaseConfig `json:"database"`
    Redis     conn.RedisConfig    `json:"redis"`
    Webserver webserver.Config `json:"webserver"`
    Email     email.Config         `json:"email"`
    Auth      auth.Config          `json:"auth"`
}

func init() {
    runtime.GOMAXPROCS(runtime.NumCPU())
}

func main() {
```

```

flag.Parse()

// load configuration
cfgenv := env.Get()
config := &configuration{}
isLoaded := jsonconfig.Load(&config, "/etc/meiko", cfgenv) || jsonconfig.Load(&config,
"./files/etc/meiko", cfgenv)
if !isLoaded {
    log.Fatal("Failed to load configuration")
}

// initiate instance
alias.InitDirectory(config.Directory)
conn.InitDB(config.Database)
conn.InitRedis(config.Redis)
bot.Init()
cron.Init()
auth.Init(config.Auth)
email.Init(config.Email)
webserver.Start(config.Webserver)
}

```

#### 4.2.2 API dan Antarmuka *Detail Profile*

Kode dari API detail *profile* untuk *monolithic* dan *microservices* pada fungsi utamanya sama, namun perbedaannya jika pada *microservices* fungsi tersebut berdiri sendiri, namun pada *monolithic* fungsi tersebut bergabung dengan fungsi lainnya seperti *course*, *assignment*, dan *module* lainnya dalam satu folder. Tabel 4.5 merupakan kode API dari *get profile*.

Kode API `/api/v1/user/profile`

```

type getProfileResponse struct {
    Name      string `json:"name"`
    Email     string `json:"email"`
}

```

```

Gender      string `json:"gender"`
Phone       string `json:"phone"`
IdentityCode int64  `json:"id"`
LineID      string `json:"line_id"`
Note        string `json:"about_me"`
ImageProfile string `json:"img"`
ImageProfileThumbnail string `json:"img_t"`
}

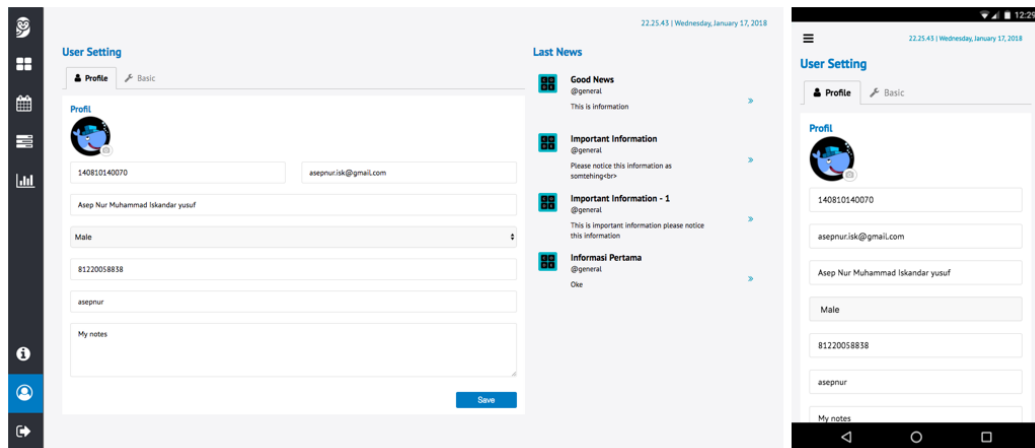
func GetProfileHandler(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {

    sess := r.Context().Value("User").(*auth.User)
    var gender string
    switch sess.Gender {
    case user.GenderMale:
        gender = "male"
    case user.GenderFemale:
        gender = "female"
    }

    res := getProfileResponse{
        Name:      sess.Name,
        Email:      sess.Email,
        Gender:     gender,
        Phone:      sess.Phone,
        IdentityCode: sess.IdentityCode,
        LineID:     sess.LineID,
        Note:       sess.Note,
        ImageProfile: alias.URLProfile,
        ImageProfileThumbnail: alias.URLProfileThumbnail,
    }

    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusOK).
        SetData(res))
    return
}

```



Gambar 4.2 Antarmuka dari API `/api/v1/user/profile`

Gambar 4.2 memperlihatkan antar muka dari API *profile*, ini menjadi sangat penting untuk di uji karena informasi dari *user* akan dipakai terus untuk kebutuhan *request* halaman yang lainnya.

#### 4.2.3 API dan Antarmuka *Detail Course*

Kode API detail *course* untuk *microservices* dan *monolithic* pada fungsi utamanya sama, namun relasi di dalam fungsinya berbeda, pada *monolithic* untuk mendapatkan informasi *user* fungsi akan langsung memanggil data *user* dari redis yang merupakan satu kesatuan dari *service* tersebut, namun pada *microservices* untuk mendapatkan data *user* atau *authentication user*, fungsi akan meminta *request* ke *service user* yang berdiri sendiri sebagai *service* yang independen. berikut merupakan kode untuk API *detail course* dengan path API `/api/admin/v1/course/:id`

```

if ps.ByName("schedule_id") == "today" {
    GetTodayHandler(w, r, ps)
    return
}

sess := r.Context().Value("User").(*auth.User)

params := getDetailParams{
    scheduleID: ps.ByName("schedule_id"),
}

args, err := params.validate()
if err != nil {
    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusBadRequest).
        AddError("Invalid Request"))
    return
}

if !cs.IsEnrolled(sess.ID, args.scheduleID) {
    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusBadRequest).
        AddError("Invalid Request"))
    return
}

c, err := cs.GetByScheduleID(args.scheduleID)
if err != nil {
    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusInternalServerError))
    return
}

resp := getDetailResponse{
    ID:      c.Schedule.ID,
    Name:    c.Course.Name,

```

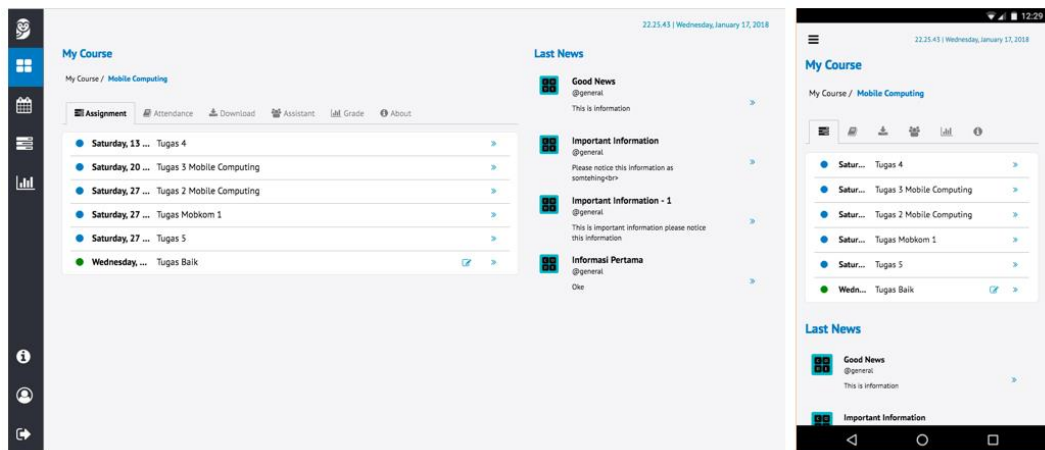


```

        Description: c.Course.Description.String,
    }

    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusOK).
        SetData(resp))
    return

```



Gambar 4.3 Antarmuka aplikasi API `/api/admin/v1/course/:id`

Gambar 4.3 memperlihatkan antarmuka aplikasi API *course detail*, antarmuka ini penting di uji karena akan menjadi halaman yang sering diakses untuk manajemen mata kuliah oleh setiap mahasiswa seperti melihat jadwal, pengajar, dan fitur yang lainnya yang berkaitan dengan course.

#### 4.2.4 API dan Antarmuka *Detail Assignment*

Seperti halnya API *detail course*, kode API detail assignment untuk microservices dan monolithic pada fungsi utamanya sama, namun relasi di dalam

fungsinya berbeda, pada monolithic untuk mendapatkan informasi user fungsi akan langsung memanggil data user dari Redis yang merupakan satu kesatuan dari service tersebut, namun pada microservices untuk mendapatkan data user atau *user authentication*, fungsi akan meminta request ke service user yang berdiri sendiri sebagai service yang independen. berikut merupakan kode untuk API *detail assignment* dengan path API `/api/v1/assignment/:id`

```
func GetDetailHandler(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {

    sess := r.Context().Value("User").(*auth.User)
    params := getDetailParams{
        id: ps.ByName("id"),
    }

    args, err := params.validate()
    if err != nil {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusBadRequest).
            AddError("Invalid Request"))
        return
    }

    assignment, err := asg.GetByID(args.id)
    if err != nil {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusNoContent))
        return
    }

    scheduleID, err := cs.GetScheduleIDByGP(assignment.GradeParameterID)
    if err != nil {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusInternalServerError))
        return
    }
}
```

```

}

if !cs.IsEnrolled(sess.ID, scheduleID) {
    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusNoContent))
    return
}

submitted, err := asg.GetSubmittedByUser(args.id, sess.ID)
if err != nil {
    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusInternalServerError))
    return
}

// response data preparation
status := "unsubmitted"
score := "-"
submittedDate := "-"
submittedDesc := ""
isAllowUpload := true
if assignment.DueDate.Before(time.Now()) {
    status = "overdue"
    isAllowUpload = false
}
if submitted != nil {
    status = "submitted"
    submittedDesc = submitted.Description.String
    submittedDate = submitted.UpdatedAt.Format("Monday, 2 January 2006 15:04:05")
    if submitted.Score.Valid {
        isAllowUpload = false
        status = "done"
        score = fmt.Sprintf("%.3g", submitted.Score.Float64)
    }
}

if assignment.Status == asg.StatusUploadNotRequired {

```

```

    status = "notrequired"
    isAllowUpload = false
}

tableID := []string{strconv.FormatInt(args.id, 10)}
asgFile, err := fl.SelectByRelation(fl.TypeAssignment, tableID, nil)
if err != nil {
    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusInternalServerError))
    return
}

submittedFile, err := fl.SelectByRelation(fl.TypeAssignmentUpload, tableID, &sess.ID)
if err != nil {
    template.RenderJSONResponse(w, new(template.Response).
        SetCode(http.StatusInternalServerError))
    return
}

// file from assistant
rAsgFile := []file{}
for _, val := range asgFile {
    rAsgFile = append(rAsgFile, file{
        ID:      val.ID,
        Name:     fmt.Sprintf("%s.%s", val.Name, val.Extension),
        URL:      fmt.Sprintf("/api/v1/file/assignment/%s.%s", val.ID, val.Extension),
        URLThumbnail: helper.MimeToThumbnail(val.Mime),
    })
}

// file from student
rSubmittedFile := []file{}
for _, val := range submittedFile {
    rSubmittedFile = append(rSubmittedFile, file{
        ID:      val.ID,
        Name:     fmt.Sprintf("%s.%s", val.Name, val.Extension),

```

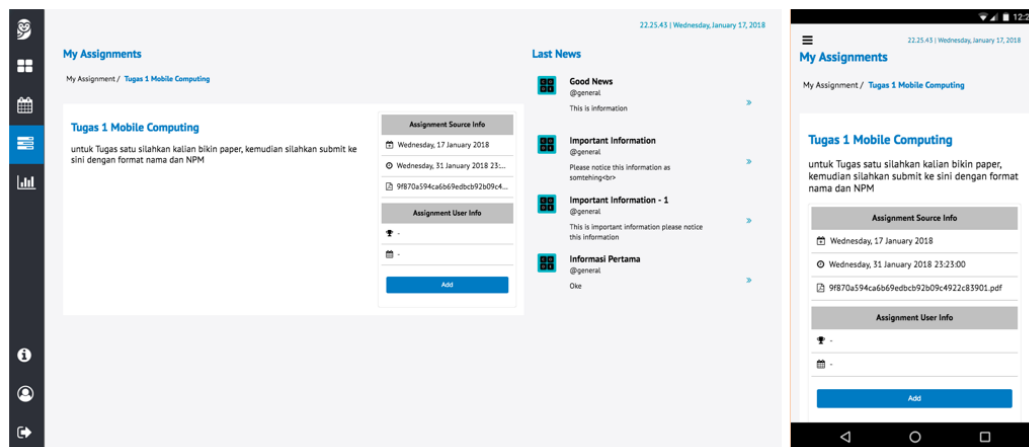
```

        URL:      fmt.Sprintf("/api/v1/file/assignment/%s.%s", val.ID, val.Extension),
        URLThumbnail: helper.MimeToThumbnail(val.Mime),
    })
}

resp := getDetailResponse{
    ID:      assignment.ID,
    Name:    assignment.Name,
    Status:   status,
    Description: assignment.Description.String,
    DueDate:  assignment.DueDate.Format("Monday, 2 January 2006 15:04:05"),
    Score:    score,
    CreatedAt: assignment.CreatedAt.Format("Monday, 2 January 2006"),
    UpdatedAt: assignment.UpdatedAt.Format("Monday, 2 January 2006"),
    AssignmentFile: rAsgFile,
    IsAllowUpload:  isAllowUpload,
    SubmittedDescription: submittedDesc,
    SubmittedFile:  rSubmittedFile,
    SubmittedDate:  submittedDate,
}

template.RenderJSONResponse(w, new(template.Response).
    SetCode(http.StatusOK).
    SetData(resp))
return
}

```



Gambar 4.4 Antarmuka API `/api/v1/assignment/:id`

Gambar 4.4 menunjukkan antarmuka API *assignment detail*, API ini sangat penting untuk di uji karena halaman ini akan sering di akses untuk kebutuhan upload tugas ataupun melihat informasi tugas yang diberikan oleh asisten kepada user.

#### 4.2.5 API dan Antarmuka *detail information*

Kode API *detail information* untuk *microservices* dan *monolithic* pada fungsi utamanya sama, namun relasi di dalam fungsinya berbeda, pada *monolithic* untuk mendapatkan informasi user fungsi akan langsung memanggil data user dari Redis, dan data informasi *course* dari *database* yang merupakan satu kesatuan dari service tersebut, namun pada *microservices* untuk mendapatkan data user atau *user authentication*, fungsi akan meminta request ke service user yang berdiri sendiri sebagai service yang independen sedangkan untuk informasi *course* akan meminta *request* ke *service course* yang juga independen sebagai service kecil. Berikut merupakan kode untuk API *detail assignment* dengan *path* API `/api/v1/information/:id`

```

func GetDetailHandler(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {

    sess := r.Context().Value("User").(*auth.User)
    params := getDetailParams{
        id: ps.ByName("id"),
    }

    args, err := params.validate()
    if err != nil {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusBadRequest).AddError(err.Error()))
        return
    }

    scheduleID := inf.GetScheduleIDByID(args.id)
    if scheduleID != 0 && !course.IsEnrolled(sess.ID, scheduleID) {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusBadRequest).
            AddError("you do not have permission to this informations"))
        return
    }

    information, err := inf.GetByID(args.id)
    if err != nil {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusBadRequest).
            AddError("Information does not exist"))
        return
    }

    desc := "-"
    if information.Description.Valid {
        desc = information.Description.String
    }

    resp := getDetailResponse{

```

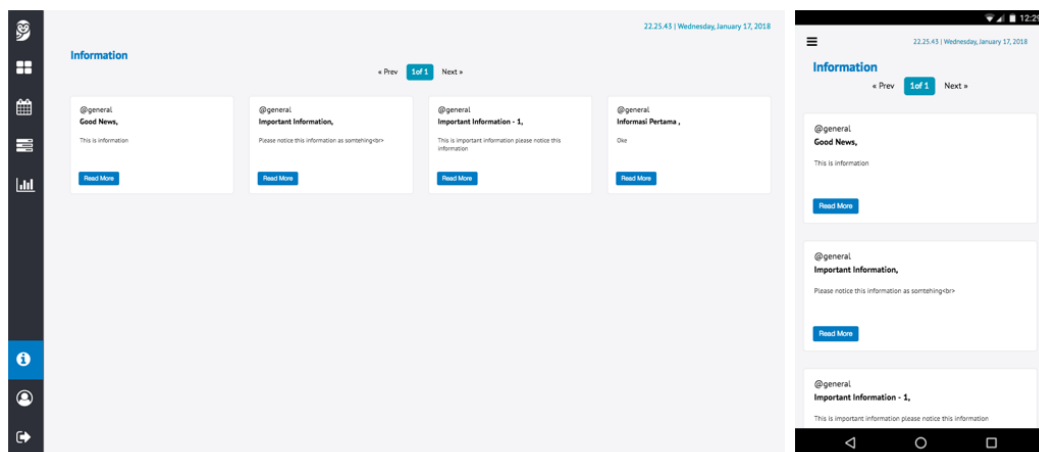
```

    ID:    information.ID,
    Title: information.Title,
    Description: desc,
    Date:    information.CreatedAt.Format("Monday, 2 January 2006"),
  }

  template.RenderJSONResponse(w, new(template.Response).
    SetCode(http.StatusOK).
    SetData(resp))

  return
}

```



Gambar 4.5 Antarmuka Aplikasi API /api/v1/information/:id

Antarmuka aplikasi untuk API informasi sangat penting untuk diuji karena informasi terbaru dari asisten untuk mahasiswa akan di sampaikan lewat fitur *information* ini. Antarmuka aplikasi di tunjukan oleh Gambar 4.6.

#### 4.2.6 API Bot dan Antarmuka Bot

Kode API detail *Bot* untuk *microservices* dan *monolithic* pada fungsi utamanya sama, namun relasi di dalam fungsinya berbeda, pada *monolithic* untuk



mendapatkan informasi *user* fungsi akan langsung memanggil data *user* dari Redis, data informasi *course*, dan *information* dari *database* yang merupakan satu kesatuan dari *service* tersebut, namun pada *microservices* untuk mendapatkan data *user* atau *user authentication*, fungsi akan meminta *request* ke *service user* yang berdiri sendiri sebagai *service* yang independen, untuk informasi *course* akan meminta *request* ke *service course*, dan untuk informasi *information* akan meminta *request* ke *service information* yang juga independen sebagai *service* kecil. Berikut merupakan kode untuk API *detail assignment* dengan API `/api/v1/bot`

```
func LoadHistoryHandler(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
    sess := r.Context().Value("User").(*auth.User)
    params := loadHistoryParams{
        id: r.FormValue("id"),
    }args, err := params.validate()
    if err != nil {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusBadRequest).
            AddError("Invalid Request"))
        return
    }
    var log []bot.Log
    if args.id.Valid {
        log, err = bot.LoadByID(args.id.Int64, sess.ID)
    } else {
        log, err = bot.LoadByUserID(sess.ID)
    }
    if err != nil {
        template.RenderJSONResponse(w, new(template.Response).
            SetCode(http.StatusInternalServerError))
        return
    }
    resp := []map[string]interface{}{}
    for _, val := range log {
```

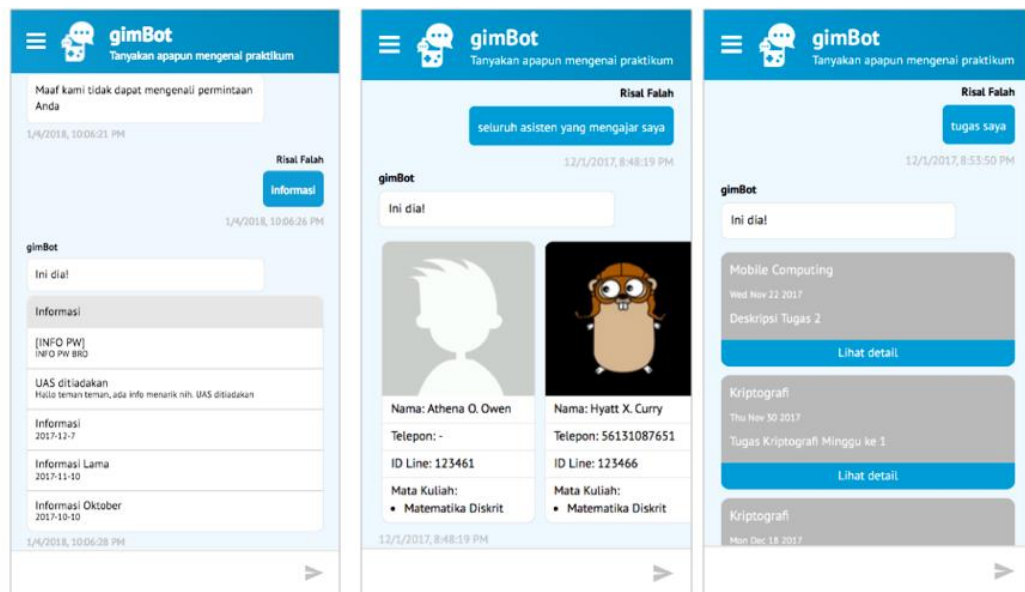
```

if val.Status == bot.StatusUser {
    resp = append(resp, map[string]interface{}{
        "status": bot.StatusUser,
        "time":   val.CreatedAt.Unix(),
        "message": map[string]interface{}{
            "id":   val.ID, "text": val.Message,
        },
    })
    continue
}

jsnMap := map[string]interface{}{}
json.Unmarshal([]byte(val.Message), &jsnMap)
jsnMap["id"] = val.ID
resp = append(resp, map[string]interface{}{
    "status": bot.StatusBot, "time":   val.CreatedAt.Unix(),
    "message": jsnMap,
})
}

template.RenderJSONResponse(w, new(template.Response).
    SetCode(http.StatusOK).
    SetData(resp))
return
}

```



Gambar 4.6 Antarmuka Aplikasi API /api/v1/bot

Gambar 4.6 menunjukkan antarmuka aplikasi fitur Bot dari Owl Assistant.

API ini penting di uji karena untuk mempermudah pengambilan informasi pengguna akan lebih sering memanfaatkan fitur ini.

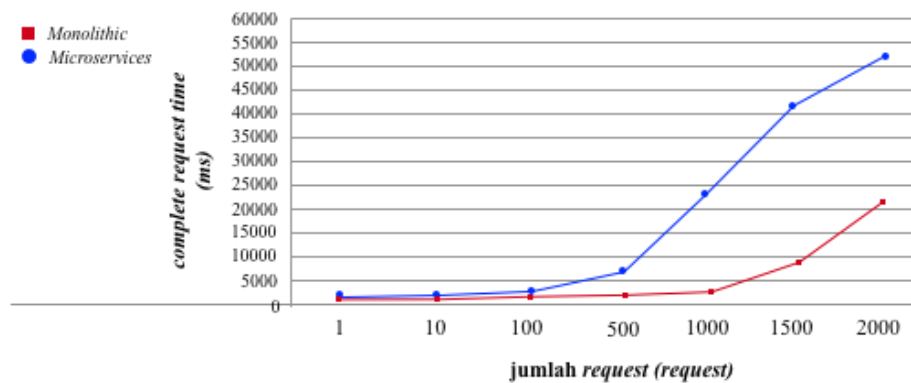
### 4.3 Hasil Penelitian

Berdasarkan skenario yang telah ditentukan, berikut merupakan hasil dari pengujian performa aplikasi Owl Assistant menggunakan *monolithic* dan *microservices architecture*. Nilai ini merupakan nilai rata-rata dari 15 kali percobaan yang dilakukan pada setiap requestnya.

Tabel 4.4 Hasil Uji Performa *Complete Request Time* Aplikasi Owl Assistant

<i>Architecture</i>	<b>Jumlah Request (request)</b>						
	1	10	100	500	1000	1500	2000
<i>Monolithic</i>	176,3	177,3	192,6	1166,6	3234,3	9970	21020
<i>Microservices</i>	261	258,6	278,6	5324	24351,6	42057	51331,3

Tabel 4.4 merupakan hasil Uji *complete request time* atau waktu yang dibutuhkan selama jumlah *request*. Jumlah *request* dimulai dari 1 *request* sampai dengan 2000 *request*. Dari hasil tabel tersebut didapatkan grafik pada Gambar 4.7



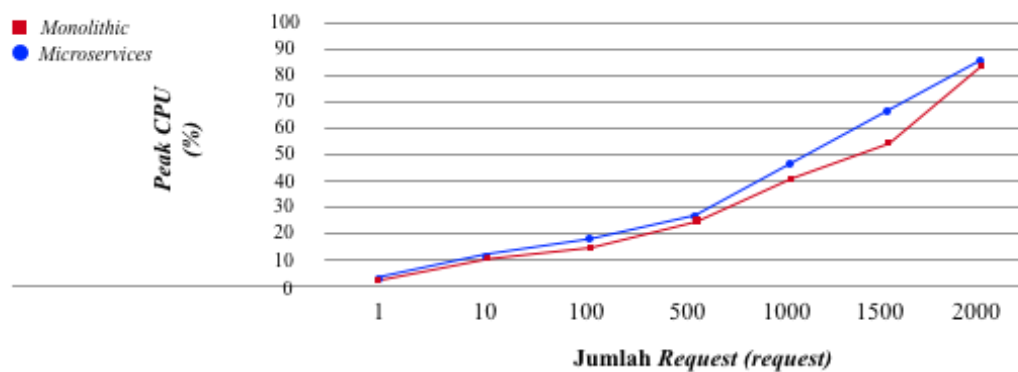
Gambar 4.7 Grafik Hasil Uji *complete request time* kedua arsitektur

Gambar 4.7 menunjukkan hasil waktu *request* dari kedua arsitektur. Garis yang berwarna merah menunjukkan *complete time request* dari *Monolithic architecture*. Dengan *request range request* dari 1 – 2000 *request*, *monolithic architecture* mampu menyelesaikan *request* dari 176 – 21020 *millisecond*. Garis berwarna biru merupakan *complete time request microservices architecture*. *Microservices architecture* mampu menyelesaikan jumlah *request* dari 1 – 2000 dengan *range waktu* 261– 51331,3 *millisecond*. Pada *range* 1 – 100 *request* perbedaan *complete request time* kedua arsitektur tidak terlalu signifikan, namun dari 500 *request* sampai 2000 *request* perbedaan dari kedua arsitektur sangat jauh. Puncaknya terjadi pada titik 2000 *request* di mana selisih *complete time request* keduanya yaitu 30311 *millisecond* atau 30 detik.

Tabel 4.5 Hasil Uji *CPU Usages* Aplikasi Owl Assistant

<i>Architecture</i>	<b>Peak CPU (%)</b>						
	1	10	100	500	1000	1500	2000
<i>Monolithic</i>	2	11	15	23	40	54	71
<i>Microservices</i>	1	11	17	25	47	67	74

Tabel 4.5 Menunjukkan hasil uji dari CPU *usages* atau penggunaan CPU selama proses pengujian dengan jumlah *request* tertentu. Dari tabel tersebut didapatkan grafik yang di ilustrasikan pada Gambar 4.8.



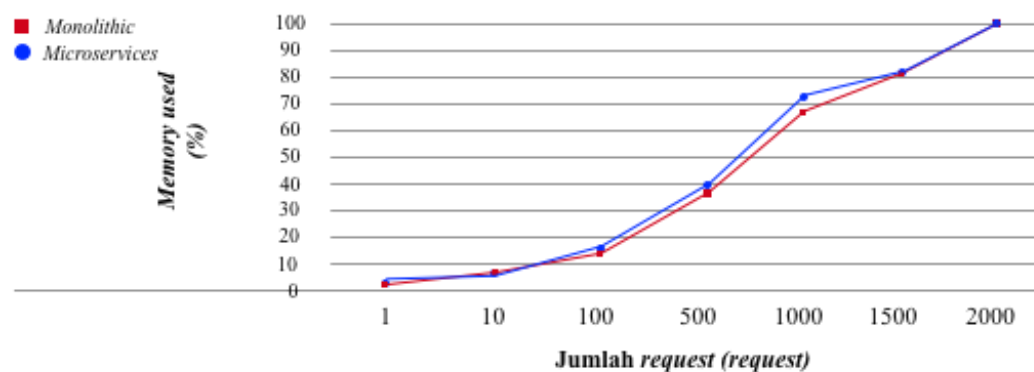
Gambar 4.8 Hasil CPU *Utilization*

Gambar 4.8 menunjukkan hasil *CPU utilization* kedua arsitektur. Garis berwarna merah merupakan penggunaan *peak CPU* menggunakan *monolithic architecture* dari 1 – 2000 *request* yang di uji, *peak CPU* yang dihasilkan *monolithic architecture* yaitu 2 – 71 %. Garis berwarna biru merupakan penggunaan *peak CPU* menggunakan *microservices architecture*. Nilai *peak CPU* menggunakan *microservices* dari 1 – 2000 *request* yaitu 1 – 74 %. Dari grafik 4.3 *resource utilization CPU* aplikasi Owl Assistant menggunakan *monolithic architecture* lebih kecil daripada *microservices architecture* meskipun perbedaan yang dihasilkan tidak terlalu jauh seperti pada *complete request time*.

Tabel 4.6 Hasil Uji *Memory Usages* Aplikasi Owl Assistant

<i>Architecture</i>	<i>Memory Peak (%)</i>						
	1	10	100	500	1000	1500	2000
<i>Monolithic</i>	3	8	12	28	69	81	100
<i>Microservices</i>	3	7	15	40	73	80	100

Tabel 4.6 Menunjukkan hasil uji dari *Memory usages* atau penggunaan *memory* selama proses pengujian dengan jumlah *request* tertentu. Dari tabel tersebut didapatkan grafik yang di ilustrasikan pada Gambar 4.9.

Gambar 4.9 Hasil *Memory Utilization*

Gambar 4.9 menunjukkan hasil *Memory utilization* kedua arsitektur. Garis berwarna merah merupakan penggunaan *memory* menggunakan *monolithic architecture* dari 1 – 2000 *request* yang di uji, *memory* yang digunakan *monolithic architecture* yaitu 3 – 100 %. Garis berwarna biru merupakan penggunaan *memory* menggunakan *microservices architecture*. Penggunaan *memory* menggunakan *microservices* dari 1 – 2000 *request* yaitu 3 – 100 %. Dari grafik 4.9 *Memory*

*utilization* perbedaan aplikasi Owl Assistant menggunakan *monolithic* dan *microservices architecture* terletak pada titik 1000 *request*.

#### 4.4 Analisis

Berdasarkan hasil pada sub bab Hasil Penelitian, dengan *error request* di bawah 3% pada pengujian yang dilakukan pada dua arsitektur, waktu *request* dengan jumlah *request* 1 – 2000 *request* dengan 15 kali pengulangan dan jumlah API uji pada tabel 4.2 didapatkan bahwa waktu *request* menggunakan *monolithic architecture* lebih cepat dari pada *microservices* dengan rentang waktu 176 – 21020 (*ms*) sedangkan *microservices architecture* 261- 51331 (*ms*).

CPU dan *memory utilization* yang digunakan pada *monolithic* tidak memiliki perbedaan yang signifikan namun penggunaan *resource* pada *monolithic* lebih kecil daripada *microservices*. *Monolithic* mempunyai *peak CPU* dari *request* yang dilakukan dari 1 – 2000 *request* yaitu 2 – 71 %, sedangkan *microservices* mempunyai *peak CPU* yaitu 74 %. *Memory* yang digunakan pada keduanya mencapai 100% dengan jumlah *request* 2000 *request*. Perbedaan signifikan dari hasil keduanya di tunjukan pada Tabel 4.13 dengan jumlah *request* yaitu 1000 *request*

Tabel 4.7 Perbedaan kedua arsitektur dengan jumlah 1000 *request*

Indeks parameter	Arsitektur	
	<i>Microservices</i>	<i>Monolithic</i>
Waktu penyelesaian <i>request</i> (ms)	24351,6	3234,3
<i>Peak CPU</i> (%)	40	47
<i>Memory utilization</i> (%)	73	69

Dari Tabel 4.7, Performa aplikasi Owl Assistant menggunakan *monolithic architecture* secara signifikan lebih baik dibandingkan *microservices* dengan indeks parameter *request time* dan *resource utilization* (*CPU* dan *Memory*) dengan *resource* yang seimbang, di mana untuk *complete request time* pada *microservices* 24351,6 *millisecond*, Peak CPU 47%, dan *memory utilization* 73% sedangkan *monolithic* lebih baik yaitu *complete request time* 3234,3 *millisecond*, *peak CPU* 40%, dan *memory utilization* 69%.



## BAB V

### SIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang diperoleh setelah proses penelitian selesai, serta sejumlah saran terkait penelitian dan implementasi perangkat lunak.

#### 5.1 Simpulan

Berdasarkan penelitian yang dilakukan, disimpulkan bahwa:

- a. Pengujian aplikasi Owl Assistant dengan **resource yang sama** menggunakan *monolithic* dan *microservices architecture*, performa aplikasi Owl Assistant menggunakan *monolithic architecture* mendapatkan performa lebih baik daripada menggunakan *microservices architecture* dengan indeks parameter pengukuran waktu penyelesaian *request* ( *request time*), waktu yang dihasilkan aplikasi Owl Assistant menggunakan *monolithic architecture* untuk 1 – 2000 *request* yaitu 176 - 21020 *millisecond*, sedangkan *microservices* 261 – 51331,3 *millisecond*.
- b. Penggunaan *resource* yang digunakan aplikasi Owl Assistant menggunakan *monolithic architecture* lebih optimal daripada menggunakan *microservices architecture*. Dari 1 - 2000 *request*, *microservices peak* CPU sampai 74 % sedangkan *monolithic* hanya mencapai 71 %.
- c. Aplikasi Owl Assistant menggunakan *microservices* lebih *scalable* daripada menggunakan *monolithic architecture* untuk pengembangan lebih lanjut.

- d. Dari segi bahasa pemrograman, aplikasi Owl Assistant menggunakan *microservices architecture* lebih fleksibel daripada *monolithic architecture* karena setiap layanan bisa memiliki bahasa pemrograman yang berbeda.

## 5.2 Saran

Adapun saran yang dapat diimplementasikan pada penelitian selanjutnya antara lain:

- a. Lakukan penelitian dengan menggunakan *method* yang lain seperti POST, DELETE, dan PUT
- b. Lakukan penelitian dengan *resource* dari *microservices* yang disesuaikan dengan kebutuhan *service* tersebut.

## DAFTAR PUSTAKA

- Baron, C. A. (2015). NoSQL Key-Value DBs Riak and Redis. *Database Systems Journal*, VI(4), 3–10. Retrieved from [http://www.dbjournal.ro/archive/22/22\\_1.pdf](http://www.dbjournal.ro/archive/22/22_1.pdf)
- Bianco, P., Kotermanski, R., & Merson, P. F. (2007). Evaluating a Service-Oriented Architecture. *Research Showcase @ CMU*, (September). Retrieved from <http://repository.cmu.edu/sei>
- Burghouwt, G., & Veldhuis, J. (2006). A Competitive Penalty Model For Availability Based Cloud SLA. *Journal of Air Transportation*, 11(1).
- Byrne, B. M., & Qureshi, Y. S. (2013). UML Class Diagram or Entity Relationship Diagram? An Object-relational Conceptual Impedance Mismatch. *Proceedings of ICERI2013 Conference 18th-20th November 2013*, (November), 3594–3604.
- Cagla Okutan, N. K. C. (2010). A monolithic approach to automated composition of semantic web services with the Event Calculus. *Knowledge-Based Systems*, 23, 440–454. <https://doi.org/https://doi.org/10.1016/j.knosys.2010.02.006>
- Cherradi, G., El Bouziri, A., Boulmakoul, A., & Zeitouni, K. (2017). Real-Time HazMat Environmental Information System: A micro-service based architecture. *Procedia Computer Science*, 109, 982–987. <https://doi.org/10.1016/j.procs.2017.05.457>
- Dzo, T. H. E., & Gui, N. E. (2017). Microservices Breaking Down the Monolith.
- Gouigoux, J. P., & Tamzalit, D. (2017). From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. *Proceedings*

- 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings, 62–65.  
<https://doi.org/10.1109/ICSAW.2017.35>

Hamilton, B. K., & Miles, R. (2006). *Learning UML 2.0*. O'Reilly.

Jain, R., Anjum, F., & Umar, A. (2000). A comparison of mobile agent and client-server paradigms for information retrieval tasks in virtual enterprises. *Proceedings - Academia/Industry Working Conference on Research Challenges 2000: Next Generation Enterprises: Virtual Organizations and Mobile/Pervasive Technologies, AIWORC 2000*, 209–213.  
<https://doi.org/10.1109/AIWORC.2000.843295>

Kumar, R. (2015). A Comparative Study and Analysis of Web Service Testing Tools, 4(1), 433–442.

Letkowski, J. (2014). Doing database design with MySQL. *Western New England University*, 6(I), 1–15.

Schmager, F. (2010). Evaluating the G O Programming Language with Design Patterns by, 175. Retrieved from  
<http://ecs.victoria.ac.nz/foswiki/pub/Main/TechnicalReportSeries/ECSTR11-01.pdf>

Wietze A. de Vries, R. A. F. (1997). Client/server infrastructure: a case study in planning and conversion, 97(6), 222–232.  
<https://doi.org/https://doi.org/10.1108/02635579710176795>

Yu, Y., Silveira, H., Sundaram, M., Yale, Y., Silveira, H., & Sundaram, M. (2016). A microservice based reference architecture model in the context of enterprise

architecture. *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 1856–1860.  
<https://doi.org/10.1109/IMCEC.2016.7867539>

Zhou, Q., Ye, H., & Ding, Z. (2012). Performance Analysis of Web Applications Based on User Navigation. *Physics Procedia*, 24, 1319–1328.  
<https://doi.org/10.1016/j.phpro.2012.02.197>

## **LAMPIRAN**



## **RIWAYAT HIDUP**

### **DATA PRIBADI**

- a. Nama Lengkap : Asep Nur Muhammad Iskandar Yusuf
- 1. NPM : 140810140070
- 2. Jenis Kelamin : Laki-laki
- 3. Agama : Islam
- 4. Alamat : Jl. Aryakiban Ds. Rajagaluh Kidul Kec. Rajagaluh  
Kab. Majalengka
- 5. Telepon : 081220058838
- 6. Email : asepnur.isk@gmail.com  
asep14010@mail.unpad.ac.id

### **RIWAYAT PENDIDIKAN**

- 2001 – 2007 : SD Negeri Kumbung 1
- 2007 – 2010 : MTs Negeri Rajagaluh 1
- 2010 – 2013 : SMAN Rajagaluh 1
- 2014 – 2018 : Teknik Informatika FMIPA Unpad

### **RIWAYAT ORGANISASI**

- 2015 – 2017 Asisten Laboratorium Teknik Informatik FMIPA Unpad
- 2015 – 2016 Kepala Divisi Film Gelanggang Seni Sastra Teater dan Film Unit  
Kegiatan Mahasiswa Unpad

### **RIWAYAT KEPANITIAAN**

- 2014 Staf Divisi Kreatif Informatics For Study Himatif Unpad
- 2015 Staf Divisi Publikasi dan Dokumentasi CBS Himatif Unpad
- 2015 Staf Divisi Marketing Informatics Festival Himatif Unpad