Jurnal Teknik Informatika vol 18 no 1 January-March 2023, pp. 315-324
p-ISSN : 2301-8364, e-ISSN : 2685-6131 , available at : https://ejournal.unsrat.ac.id/index.php/informatika

Development of Microservices Architecture with RESTful API Gateway using Backend-for-frontend Pattern in Higher Education Academic Portal

Pengembangan Arsitektur Microservices dengan RESTful API Gateway menggunakan Backend-for-frontend Pattern pada Portal Akademik Perguruan Tinggi

Fransiscus Xaverius Senduk, Xaverius B. N. Najoan, Sherwin R. U. A. Sompie
Dept. of Electrical Engineering, Sam Ratulangi University Manado, Kampus Bahu St., 95115, Indonesia
e-mails : fxsenduk@gmail.com, xnajoan@unsrat.ac.id, aldo@unsrat.ac.id
Received: 5 January 2023; revised: 17 February 2023; accepted: 21 February 2023

***Abstract* — Applications are typically built with a monolithic architecture, where all application components are combined into a single, indivisible application. As the monolithic application is further updated and developed, the drawbacks of this monolithic architecture begin to outweigh its benefits. At this time, the INSPIRE UNSRAT Portal still uses a monolithic architecture, hence further development will bring about the drawbacks of this architecture. Based on these issues, this study was conducted with the aim of developing a microservices architecture with a RESTful API Gateway using the Backend-for-frontend pattern, using the INSPIRE UNSRAT Portal application as a case study. The software development method used is Scrum, therefore it was carried out in collaboration with three other researchers in the fields of project management, mobile applications, and testing. This research succeeded in designing and creating a microservices architecture that includes 8 services and 39 endpoints, accelerating the development process, and allowing for improved service quality on mobile devices.**
***Key words* — Academic; Microservices; Portal; RESTful API;**

***Abstrak* — Pada umumnya aplikasi dibangun dengan arsitektur *monolith* dimana semua komponen aplikasi dikombinasikan menjadi satu aplikasi tunggal yang tidak terbagi-bagi. Setelah aplikasi monolitik tersebut diperbarui dan dikembangkan lebih lanjut, kerugian dari arsitektur *monolith* ini akan mulai melampaui manfaat yang ditawarkannya. Pada saat ini, Portal INSPIRE UNSRAT masih menggunakan arsitektur *monolith* sehingga pada pengembangan lebih lanjut akan timbul kerugian dari arsitektur ini. Berdasarkan permasalahan tersebut maka dibuatlah penelitian ini yang bertujuan untuk mengembangkan arsitektur *microservices* dengan *RESTful API Gateway* menggunakan *Backend-for-frontend pattern* dengan mengambil studi kasus pada aplikasi Portal INSPIRE UNSRAT. Metode pengembangan perangkat lunak yang digunakan adalah *Scrum* sehingga dikerjakan bersama dengan tiga peneliti lain di bidang manajemen proyek, aplikasi *mobile*, dan pengujian. Penelitian ini berhasil merancang dan membuat arsitektur *microservices* yang di dalamnya terdapat 8 *service* dan 39 *endpoint*, mempercepat proses pengembangan, dan memungkinkan kualitas layanan yang ditingkatkan pada perangkat *mobile*.**
***Kata kunci* — Akademik; Microservices; Portal; RESTful API;**

I. PENDAHULUAN

Selama bertahun-tahun, aplikasi telah dikemas menggunakan arsitektur *monolith* – yaitu tim pengembang akan membuat satu aplikasi besar yang melakukan semua yang diperlukan untuk kebutuhan bisnis. Kelemahan dari sistem ini muncul ketika monolith mulai diperbarui dan dikembangkan, meningkatkan ukuran dan jumlah fitur. Setelah ini terjadi, kerugian arsitektur monolith akan melebihi keuntungannya [1]. Misalnya, jika diperlukan lebih banyak kemampuan komputasi, penskalaan *monolith* besar akan menghasilkan pengeluaran yang lebih besar dibandingkan dengan *monolith* kecil. Solusi yang memungkinkan untuk masalah ini adalah mengubah arsitektur monolith menjadi arsitektur *microservices*.

Ketika konsep *microservices* menjadi populer, *API Gateway* menjadi komponen standar untuk integrasi di lapisan aplikasi atas. Menggunakan layanan *API Gateway* di *microservices* dapat membuat *client* tidak terpengaruh oleh lokasi *instance* layanan dan tidak mengetahui bagaimana aplikasi dipecah menjadi beberapa layanan kecil (*microservices*). Salah satu juga manfaat *API Gateway* adalah merangkum struktur internal aplikasi, sehingga client tidak perlu mengetahui bagaimana kerumitan sistem didalam *microservices* [2]. Pengembangan Portal INSPIRE UNSRAT saat ini menggunakan arsitektur *monolith* [3], dimana seperti dijelaskan sebelumnya arsitektur ini memiliki kelemahan saat ukuran aplikasi sudah semakin besar dan fitur didalamnya sudah semakin banyak dan rumit.

Penelitian ini bertujuan untuk membuat *microservices* dengan *RESTful API Gateway* menggunakan *Backend-for-frontend Pattern* pada Portal INSPIRE UNSRAT. Portal ini merupakan studi kasus dari penelitian ini. Sehingga dari penelitian ini diharapkan memiliki manfaat yaitu memungkinkan implementasi *frontend* Portal INSPIRE pada platform *mobile* dan memungkinkan teknologi pihak ketiga untuk terhubung dengan Portal INSPIRE melalui Web Services.

A. Penelitian Terkait

1) Penelitian Rezaldy dkk berjudul "Desain dan Analisis Arsitektur *Microservices* Pada Sistem Informasi Akademik Perguruan Tinggi Dengan Pendekatan *Architecture Tradeoff Analysis Method* (ATAM) (Studi Kasus: iGracias Universitas Telkom)" [4]. Persamaannya adalah penelitian ini menggunakan arsitektur *microservices*, sedangkan perbedaannya adalah penelitian ini tidak menggunakan *backend-for-frontend pattern* dan hanya merancang dan menganalisis arsitektur *microservices* jika diterapkan pada studi kasus terkait.
2) Penelitian Radhiyan berjudul "Analisis dan Desain Arsitektur *Microservices* dengan *GraphQL* sebagai *API Gateway* untuk Sistem Informasi Akademik AIS UIN Jakarta (Studi Kasus : AIS untuk Mahasiswa)" [5]. Persamaannya adalah penelitian ini menggunakan arsitektur *microservices* dan memiliki *API Gateway*, sedangkan perbedaannya adalah penelitian ini menggunakan *GraphQL* sebagai standar komunikasi API, tidak menggunakan *backendfor-frontend pattern*, dan hanya merancang dan menganalisis arsitektur *microservices* jika diterapkan pada studi kasus terkait.
3) Penelitian Pranata dkk berjudul "Perancangan *Application Programming Interface* (API) berbasis *Web* menggunakan Gaya Arsitektur *Representational State Transfer* (REST) untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit" [6]. Persamaannya adalah penelitian ini mengikuti REST dalam pembuatan *API* berbasis *Web* dan menggunakan *JWT (JSON Web Token)* sebagai cara otentikasi pengguna, sedangkan perbedaannya adalah penelitian ini menggunakan arsitektur *monolith*.
4) Penelitian Belluano dkk berjudul "Sistem Informasi Program Kreativitas Mahasiswa berbasis *Web Service* dan *Microservice*" [7]. Persamaannya adalah penelitian ini menggunakan arsitektur *microservices*, mengikuti *REST* dalam pembuatan *API* berbasis *Web*, dan menggunakan *message broker/queuing*, sedangkan perbedaannya adalah penelitian ini tidak menggunakan *API Gateway* dan komunikasi antar *service* menggunakan *REST protocol*.
5) Penelitian Utomo dkk berjudul "Perancangan *RESTFul Web Service* Pada Sistem Informasi Terintegrasi Menggunakan *FrameWork CodeIgniter*" [8]. Persamaannya adalah penelitian ini mengikuti *REST* dalam pembuatan *API* berbasis *Web* dan menggunakan *framework Codeigniter*, sedangkan perbedaannya adalah penelitian ini tidak menggunakan arsitektur *microservices*.
6) Penelitian Tedyyana dkk berjudul "*Revamp* Keamanan *Web Service* Milik PT XYZ Menggunakan *REST API*" [9]. Persamaannya adalah penelitian ini mengikuti arsitektur *REST* dalam pembuatan *API* berbasis *Web*, menggunakan *JWT (JSON Web Token)* sebagai cara otentikasi pengguna, dan menggunakan bahasa pemrograman PHP, sedangkan perbedaannya adalah penelitian ini menggunakan *framework* Lumen dan arsitektur *monolith*.
7) Penelitian Danil Rafiqi dkk berjudul "Implementasi Arsitektur *Microservice* Pada Aplikasi Online Travel Tourinc" [10]. Persamaannya adalah penelitian ini menggunakan arsitektur *microservices* dan *RESTful API Gateway*, sedangkan perbedaannya adalah penelitian ini tidak menggunakan *Backend-for-frontend pattern*, tidak menjelaskan teknik komunikasi antar-*service* serta memiliki studi kasus yang berbeda.
8) Penelitian Mubariz dkk berjudul "Perancangan *Back-End Server* menggunakan Arsitektur *REST* dan Platform Node.JS (Studi Kasus: Sistem Pendaftaran Ujian Masuk Politeknik Negeri Ujung Pandang)" [11]. Persamaannya adalah penelitian ini mengikuti *REST* dalam pembuatan *API* berbasis *Web*, sedangkan perbedaannya adalah penelitian ini menggunakan platform Node.js dan arsitektur *monolith*.

B. Microservices

Microservices adalah cara mengembangkan dan menyusun sistem perangkat lunak seperti mereka dibangun dari komponen independen kecil yang berinteraksi satu sama lain melalui jaringan [12]. Arsitektur *microservices* dapat mengatasi kompleksitas yang besar karena tiap fitur aplikasi akan dibagi menjadi bagian kecil yang disebut *service*. Service tersebut berjalan masing-masing sesuai dengan fungsinya. Kumpulan dari *service* ini akan saling berkomunikasi sehingga menjadi satu kesatuan dan menjadi sistem yang besar [10].

C. RESTful API

Application Programming Interface (API) adalah antarmuka yang dibangun oleh pengembang sistem agar sebagian atau seluruh fungsi sistem dapat diakses secara terprogram. *Representational State Transfer (REST)* adalah salah satu gaya arsitektur pengembangan API yang menggunakan *Hypertext Transfer Protocol (HTTP)* untuk komunikasi data [6]. *REST* didasarkan pada gaya arsitektur *client/server* dan pertama kali dikemukakan oleh Roy Fielding dalam disertasinya yang berjudul *"Architectural Styles and the Design of Network-based Software Architectures"* pada tahun 2000 [13].

Dalam *REST*, terdapat *request* dan *response* yang terjadi berdasarkan komunikasi sumber daya. Sumber daya atau aplikasi web ini diidentifikasi menggunakan *URI (Uniform Resource Identifier)*. Aplikasi *web* yang mengikuti arsitektur *REST* disebut sebagai *RESTful web service* [9]. Sehingga *API (Application Programming Interface)* yang mengikuti arsitektur *REST* juga disebut sebagai *RESTful API*.

D. API Gateway

API Gateway adalah layanan yang berada di antara *client* dan layanan *backend (server)* yang bertindak sebagai titik masuk tunggal ke sistem. Pertama, *client* mengirim permintaan ke *API Gateway*, dan kemudian *API Gateway* menghubungi *endpoint* yang relevan untuk mengumpulkan informasi *client*. Akhirnya, tanggapan dikirimkan kembali ke *client* [14].

E. Backend-for-frontend

Pola *Backend-For-Frontend*, juga dikenal sebagai *BFF Pattern*, mirip dengan pola *API Gateway*. Namun, setiap jenis

client, misalnya *mobile*, *desktop*, *tablet*, dan *browser*, mendapatkan BFF-nya sendiri sehingga tim *frontend* dapat menyesuaikan BFF agar sesuai dengan kebutuhan jenis *client* mereka [14]. Dengan demikian, BFF merupakan salah satu jenis implementasi *API Gateway* karena dengan BFF setiap jenis *client* memiliki *API Gateway* sendiri.

F. Arsitektur Backend Portal INSPIRE UNSRAT

Dalam penelitian ini, studi kasus untuk Portal Akademik Perguruan Tinggi adalah Portal INSPIRE UNSRAT. Portal ini adalah aplikasi Sistem Informasi Akademik yang memiliki fungsi untuk menampilkan informasi yang berhubungan dengan proses akademik di Universitas Sam Ratulangi Manado [15]. Aplikasi ini masih menggunakan pola desain *MVC (Model, View, Controller)* [3]. Berdasarkan pola desain tersebut, arsitektur aplikasi Portal INSPIRE masih dibuat dalam bentuk monolith dan bukan *microservices*.

G. Scrum

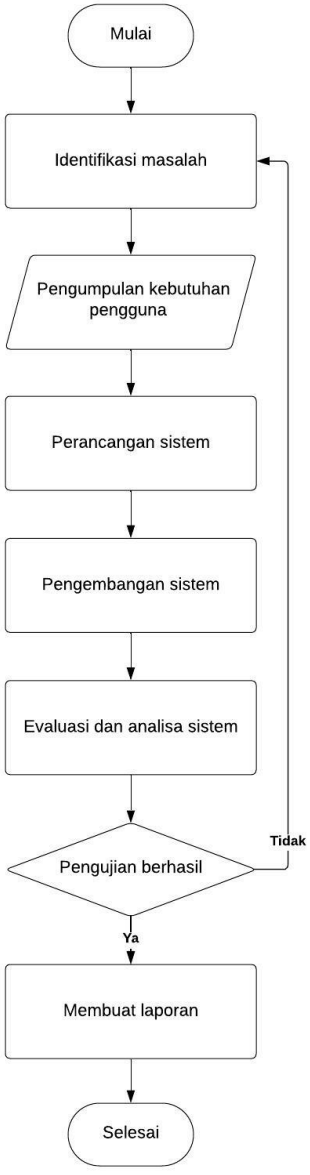
SCRUM mendefinisikan proses pengembangan sistem sebagai serangkaian aktivitas longgar yang menggabungkan alat dan teknik yang diketahui dan dapat diterapkan dengan yang terbaik yang dapat dirancang oleh tim pengembangan untuk membangun sistem. Karena aktivitas ini longgar, kontrol untuk mengelola proses dan risiko bawaan digunakan. SCRUM adalah peningkatan dari siklus pengembangan berorientasi objek iteratif/inkremental yang umum digunakan. Perbedaan utama antara pendekatan *waterfall, spiral* dan *iterative* dengan SCRUM adalah bahwa pendekatan SCRUM mengasumsikan bahwa proses analisis, desain, dan pengembangan dalam fase *Sprint* tidak dapat diprediksi. Mekanisme kontrol digunakan untuk mengelola ketidakpastian dan mengendalikan risiko. Fleksibilitas, daya tanggap, dan keandalan adalah hasilnya [16].

II. METODE

A. Prosedur Penelitian

Langkah-langkah yang akan dilakukan dalam penelitian ini adalah sebagai berikut:

1) Identifikasi masalah. Pada langkah ini, dilakukan analisis untuk mengidentifikasi masalah atau kebutuhan yang ingin dipecahkan dengan pengembangan sistem atau layanan tersebut. Identifikasi dilakukan bersama dengan *Project Manager* yang berkoordinasi dengan *Product Owner* melalui wawancara.
2) Pengumpulan kebutuhan pengguna. Hal ini diperlukan untuk membangun sistem atau layanan berbasis *web*. Ini melibatkan mengidentifikasi fitur, fungsionalitas, dan persyaratan lain yang harus dipenuhi. Tahapan ini menghasilkan daftar kebutuhan pengguna atau dalam *SDLC Scrum* disebut dengan *backlog*.
3) Perancangan sistem. Tahapan ini merancang alur sistem berupa proses yang terjadi didalamnya, model data, dan arsitektur *backend* yang akan dikembangkan. Perancangan ini disesuaikan dengan analisa kebutuhan yang didapat pada langkah sebelumnya.
4) Pengembangan sistem. Setelah sistem telah dirancang, langkah ini melibatkan implementasi atau pembangunan sistem atau layanan berbasis *web* sesuai dengan desain yang telah dibuat. Dilakukan juga pemilihan teknologi yang sesuai dengan kebutuhan sistem. Tahapan ini menghasilkan luaran berupa aplikasi.
5) Evaluasi dan analisa sistem. Tujuannya adalah untuk memastikan bahwa sistem berjalan dengan baik dan memenuhi kebutuhan yang telah ditetapkan sebelumnya. Tahapan ini termasuk dengan pengujian, yang dibahas lebih mendalam pada penelitian yang lain. Jika pengujian berhasil, proses dapat melanjutkan ke langkah selanjutnya. Namun, jika pengujian tidak berhasil, proses harus kembali ke langkah identifikasi masalah untuk memperbaiki masalah yang terdeteksi.
6) Membuat laporan. Setelah sistem telah dievaluasi dan diuji, langkah ini melibatkan pembuatan laporan yang berisi deskripsi lengkap mulai dari perancangan sampai hasil berupa evaluasi, analisis, dan detail tentang sistem yang telah dikembangkan.



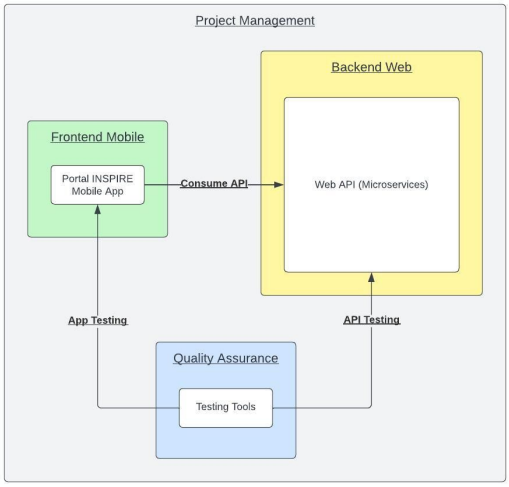
Gambar 1. Prosedur penelitian

B. Metode Pengembangan Perangkat Lunak

Metode Pengembangan Perangkat Lunak atau *Software Development Life Cycle* (SDLC) yang digunakan dalam penelitian ini adalah *Scrum*. Metode ini dipilih karena penelitian ini dilakukan secara bersamaan dengan peneliti lain pada aplikasi yang sama yaitu Portal INSPIRE. Peneliti lain berada pada bidang *Project Manager*, *Frontend Mobile*, dan *Quality Assurance*. Metode ini memungkinkan peneliti bisa melakukan kolaborasi bersama dengan peneliti lain secara *agile*, namun tetap juga berkoordinasi dengan *Product Owner* sebagai pemegang *Product Backlog* / *User Requirements*. *Product Owner* dalam studi kasus penelitian ini adalah Kepala Unit Pelaksana Teknis - Teknologi Informasi dan Komunikasi (UPT-TIK) Universitas Sam Ratulangi Manado. Peneliti juga bisa melakukan perubahan secara *iterative* dan *incremental*.

Untuk menggambarkan dengan jelas bagaimana pembagian bidang pengerjaan penelitian bisa dilihat seperti yang ditunjukkan pada Gambar 2 dan dapat diuraikan bahwa:

1) Bidang *Backend Web* bertugas untuk membuat *Application Programming Interface (API) Web* menggunakan arsitektur *microservices* pada Portal INSPIRE UNSRAT. Inilah bidang yang dikerjakan dalam penelitian ini.
2) Bidang *Frontend Mobile* bertugas mengerjakan tampilan aplikasi mobile dari Portal INSPIRE UNSRAT. Bidang ini akan menggunakan *API (Consume API)* yang disediakan oleh *Backend Web*.
3) Bidang *Quality Assurance* bertugas melakukan pengujian *(testing)* terhadap aplikasi *mobile* yang dibuat oleh *Frontend Mobile (App Testing)* dan melakukan testing terhadap *API* yang dibuat oleh *Backend Web (API Testing)*.
4) Bidang *Project Management* bertugas untuk menjalankan tugas sebagai *Project Manager*. Bidang ini bertugas merumuskan daftar kebutuhan pengguna dalam bentuk *backlog*, menerapkan metodologi pengembangan perangkat lunak *Scrum* pada tiga peneliti lain dan memantau serta memastikan keberhasilan pelaksanaan proyek.



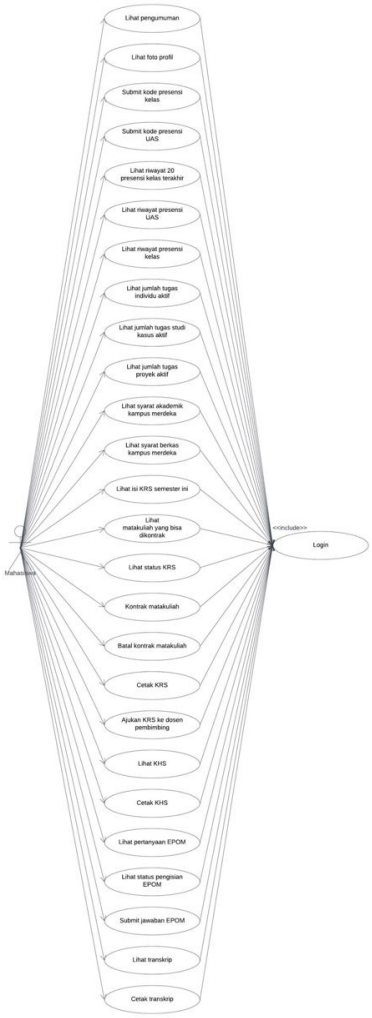
Gambar 2. Pembagian bidang kerja

C. Perancangan sistem

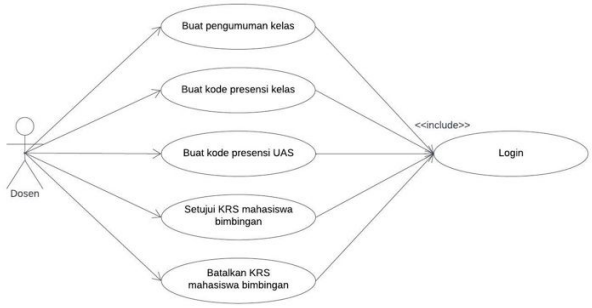
Unified Modeling Language (UML) menjadi alat yang digunakan untuk merancang dan menggambarkan struktur dan proses yang terjadi dalam pengembangan aplikasi backend RESTful API ini. Dibuat diagram UML untuk menjelaskan fitur yang digunakan user dalam bentuk Use Case Diagram, mendefinisikan proses kegiatan yang terjadi dalam bentuk Activity Diagram, dan struktur dari basis data yang digunakan dalam bentuk Entity Relationship Diagram. Use Case Diagram dapat dilihat seperti ditunjukkan pada Gambar 3, 4, dan 5. Beberapa Activity Diagram dapat dilihat seperti ditunjukkan pada Gambar 6, 7, 8, dan 9. Kemudian salah satu Entity Relationship Diagram dapat dilihat pada Gambar 10.

Pada beberapa kasus tertentu, karena bidang Quality Assurance membutuhkan pengujian yang bertindak sebagai user Dosen dan user Admin, maka dibuatlah beberapa Use Case untuk user tersebut.

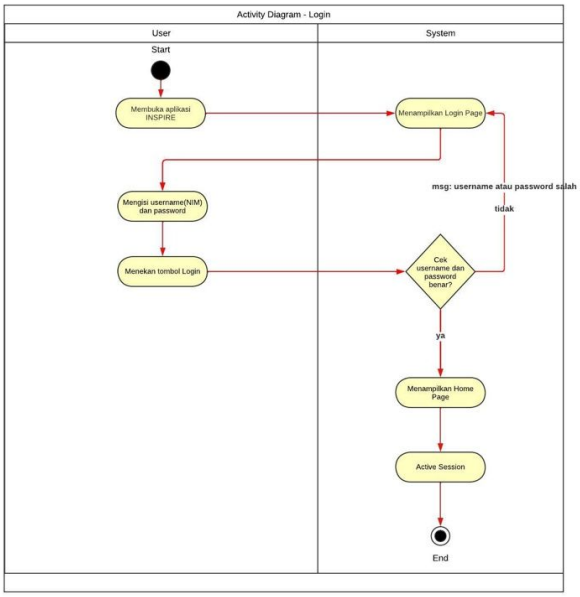
Gambar 3. *Use Case Diagram* admin



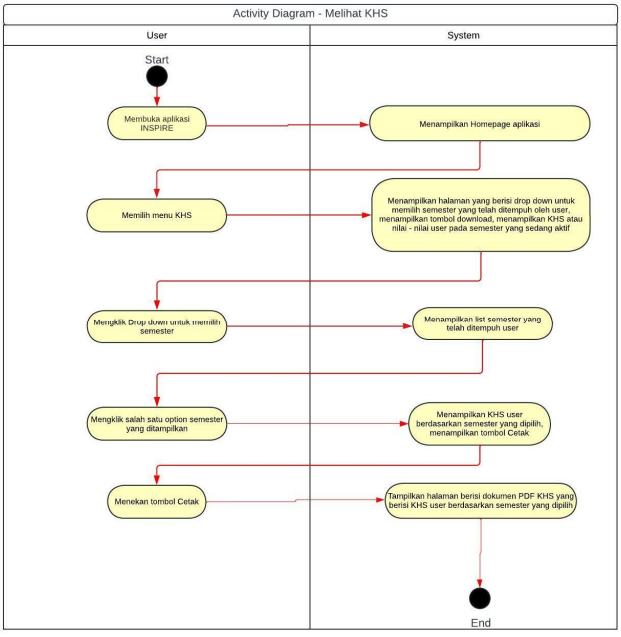
Gambar 4. *Use Case Diagram* mahasiswa



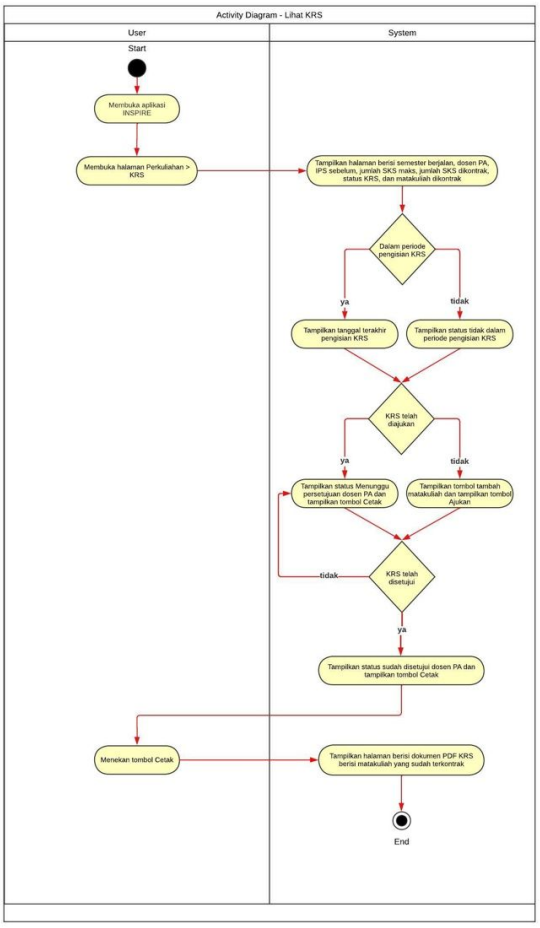
Gambar 5. *Use Case Diagram* dosen



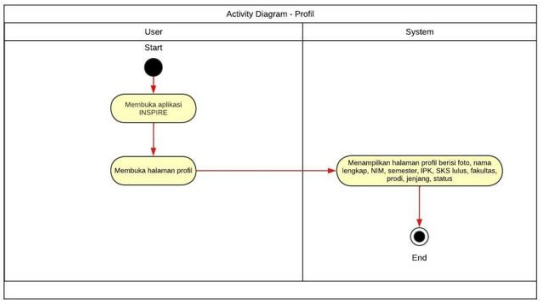
Gambar 6. *Activity Diagram* login



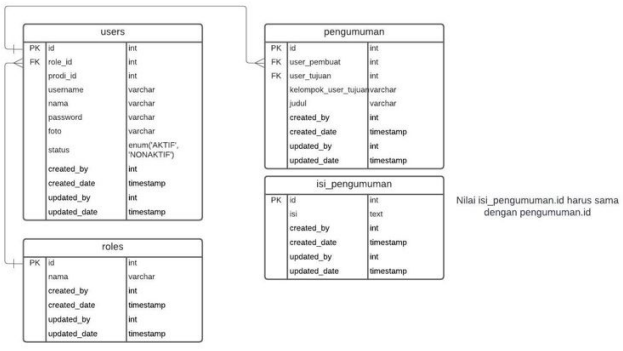
Gambar 7. *Activity Diagram* KHS



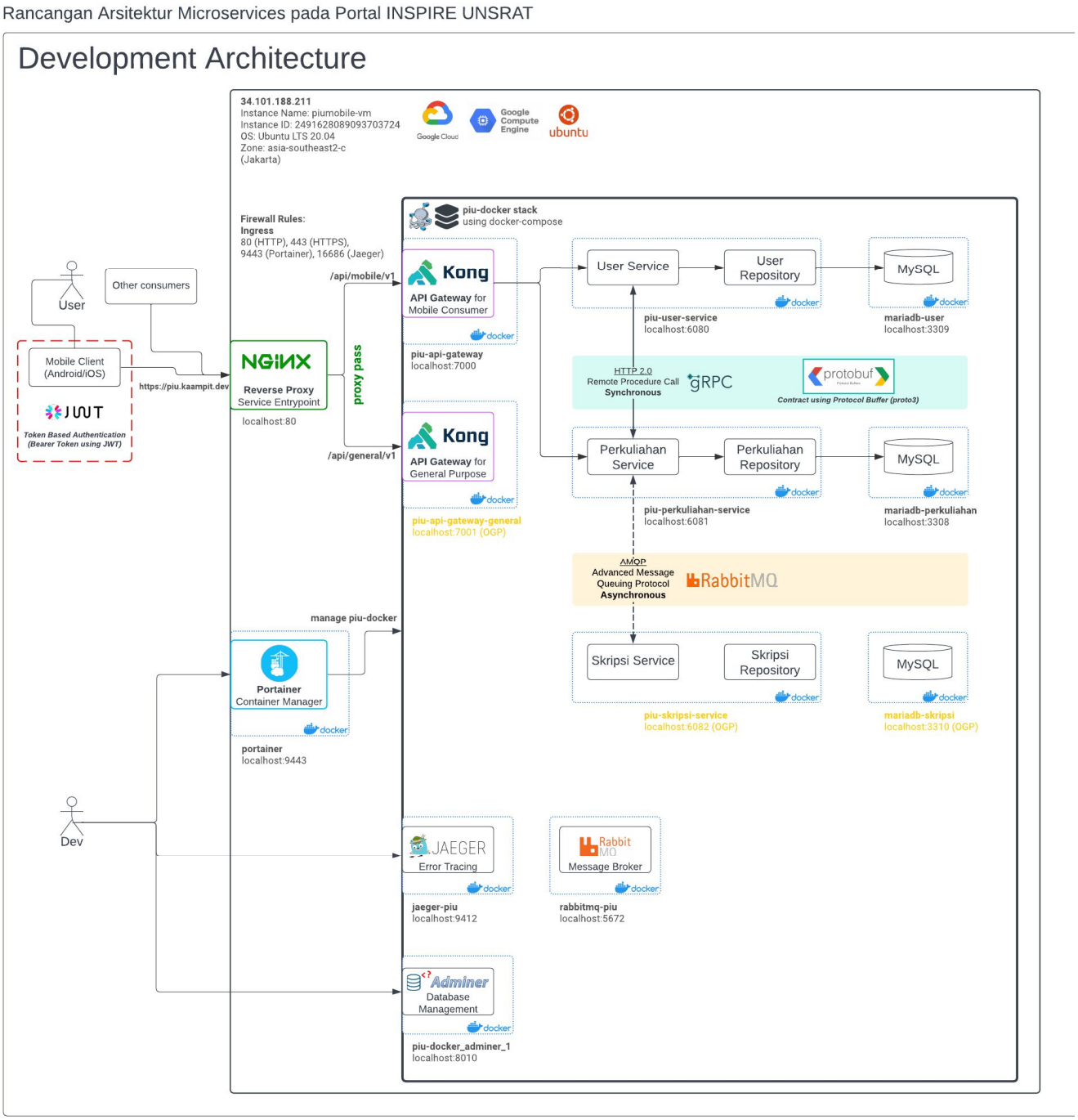
Gambar 8. *Activity Diagram* KRS



Gambar 9. *Activity Diagram* profil pengguna



Gambar 10. *Entity Relationship Diagram* - User Service



Gambar 11. *Development Architecture*

III. HASIL DAN PEMBAHASAN

Arsitektur ini seperti yang bisa dilihat pada Gambar 11 memiliki beberapa konsep dan prinsip yang mendasarinya. Konsep utamanya adalah pembagian sistem menjadi serangkaian *microservice* yang mandiri, dengan setiap *microservice* bertanggung jawab atas fungsionalitas yang spesifik. Pembagian sistem ini didasarkan pada domain yang sesuai. Yaitu, domain pengguna dan domain perkuliahan. Masing-masing domain ini memiliki *service*-nya sendiri. Selain itu, arsitektur ini mengadopsi prinsip-prinsip desain *RESTful API* untuk memastikan komunikasi yang terstandarisasi, skalabel, dan terdokumentasi dengan baik. Penerapan pola *Backend-for-frontend* bertujuan untuk menyediakan antarmuka yang dioptimalkan sesuai kebutuhan *frontend* tertentu, mengurangi beban kerja pada sisi klien, dan memisahkan tugas antara *frontend* dan *backend*.

Implementasi arsitektur ini memanfaatkan lingkungan teknologi modern, yaitu dengan menggunakan Docker untuk mengelola kontainerisasi *microservices*. Arsitektur ini juga

TABEL I
KOMPONEN-KOMPONEN DALAM ARSITEKTUR

No	Nama Service	Deskripsi
1.	piu-user-service	Layanan API yang berkaitan dengan domain User
2.	piu-perkuliahan-service	Layanan API yang berkaitan dengan domain Perkuliahan
3.	mariadb-user	Layanan basis data yang terhubung dengan User Service
4.	mariadb-perkuliahan	Layanan basis data yang terhubung dengan Perkuliahan Service
5.	piu-api-gateway	API Gateway sebagai single entry point dalam arsitektur ini
6.	adminer-piu	Layanan untuk mengelola basis data melalui antarmuka GUI
7.	jaeger-piu	Layanan untuk menampung, melacak, dan menganalisa error yang terjadi pada layanan API
8.	rabbitmq-piu	Layanan yang berfungsi sebagai message broker

menggunakan *framework open-source* Codeigniter untuk mengimplementasikan *microservices*, serta Kong untuk *API Gateway*. Untuk keseluruhan 8 service yang berhasil dibangun dalam arsitektur microservices ini bisa dilihat pada Tabel 1.

A. Pengembangan RESTful API

Pengembangan *RESTful API* dilakukan menggunakan *framework* Codeigniter 4. Codeigniter 4 adalah *framework* PHP yang ringan dan *powerful*, yang menawarkan berbagai fitur untuk memudahkan pengembangan aplikasi web. Dalam pengembangan API, Codeigniter 4 memberikan struktur dan mekanisme yang efisien untuk mengelola rute, validasi data, dan respons API yang konsisten.

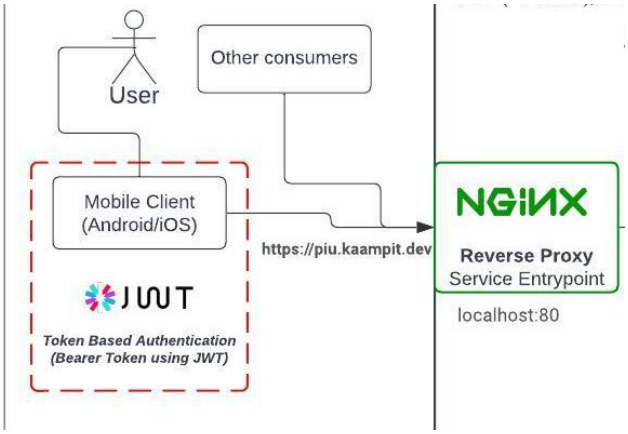
Sebagai standar, response API harus memiliki struktur yang ditetapkan. Hal ini harus dilakukan agar API *consumer* mengetahui apa saja komponen-komponen yang selalu ada dan selalu berubah pada *response body* suatu *API endpoint call*. Dalam pengembangan API versi satu ini, digunakan struktur bahwa setiap *response body* harus memiliki elemen *status*, *message*, dan data. Elemen status berisi *HTTP Response Status Code* terhadap response tersebut, penulisan nilai ini pada *response body* bertujuan agar *API consumer* bisa lebih mudah membaca *HTTP Response Status Code*. Elemen *message* berisi pesan lain yang dikirimkan oleh *backend* untuk lebih menjelaskan *Status Code*, atau jika terjadi *error* akan dicantumkan di elemen ini. Elemen data berisi data yang diharapkan oleh *user* berdasarkan *request* yang dikirimkan, jika respon *backend* adalah *error* atau tidak memiliki data maka elemen data ini akan kosong.

Standar selanjutnya adalah prefix URL yang ditetapkan untuk memudahkan pemversian API. Dalam penelitian ini, struktur URL yang digunakan adalah *prefix* /api/mobile/v1. *Prefix* ini digunakan untuk mengidentifikasi bahwa alamat internet yang diakses adalah API, untuk *platform mobile*, dan versi API adalah versi satu. Pada lingkungan *deployment*, pengembangan dilakukan pada domain piu.kaampit.dev sehingga URL lengkap yang bisa diakses adalah "https://piu.kaampit.dev/api/mobile/v1".

Dalam arsitektur *microservices* ini, dokumentasi API menjadi sangat penting untuk memudahkan penggunaan dan pengembangan layanan yang disediakan oleh setiap *microservice*. Untuk itu, digunakanlah Swagger API Docs dengan standar OpenAPI 3 sebagai alat untuk menghasilkan dokumentasi API yang konsisten dan mudah dibaca. Dokumentasi ini nantinya akan digunakan untuk saling memahami kebutuhan yang sudah diimplementasi oleh *backend*. Tim-tim lain yang berkolaborasi dengan *backend* diharapkan bisa memahami bagaimana interaksi yang

TABEL 2
DAFTAR SELURUH API ENDPOINT YANG DIBUAT

No	Verb	Path
1.	POST	/auth/login
2.	GET	/users/profile
3.	GET	/users/photo
4.	GET	/pemberitahuan
5.	POST	/pemberitahuan
6.	GET	/kampus-merdeka/syarat-akademik
7.	GET	/kampus-merdeka/syarat-berkas
8.	POST	/kampus-merdeka/syarat-berkas
9.	GET	/krs
10.	GET	/krs/matakuliah
11.	GET	/krs/status
12.	PATCH	/krs/status/draft
13.	PATCH	/krs/status/ajukan
14.	PATCH	/krs/status/setujui
15.	PUT	/krs/kontrak/{kelasId}
16.	DELETE	/krs/kontrak/{kelasId}
17.	GET	/krs/cetak
18.	POST	/presensi/kelas
19.	GET	/presensi/kelas/riwayat
20.	POST	/presensi/uas
21.	GET	/presensi/uas/riwayat/{kelasId}
22.	POST	/presensi/uas/generate/{kelasId}
23.	GET	/kelas/{kelasId}/presensi/riwayat
24.	POST	/kelas/{kelasId}/presensi/generate
25.	GET	/kelas/{kelasId}/tugas/individu
26.	GET	/kelas/{kelasId}/tugas/studi-kasus
27.	GET	/kelas/{kelasId}/tugas/proyek
28.	GET	/kelas/{kelasId}/tugas/rincian/{tugasId}
29.	GET	/epom
30	GET	/epom/{kelasId}
31.	POST	/epom/{kelasId}
32.	GET	/khs/semester/{semesterId}
33.	GET	/khs/matakuliah/semester/{semesterId}
34.	GET	/khs/cetak/semester/{semesterId}
35.	GET	/transkrip
36.	GET	/transkrip/matakuliah
37.	GET	/transkrip/cetak
38.	GET	/user-service/log
39.	GET	/perkuliahan-service/log



Gambar 12. Reverse proxy menggunakan Nginx

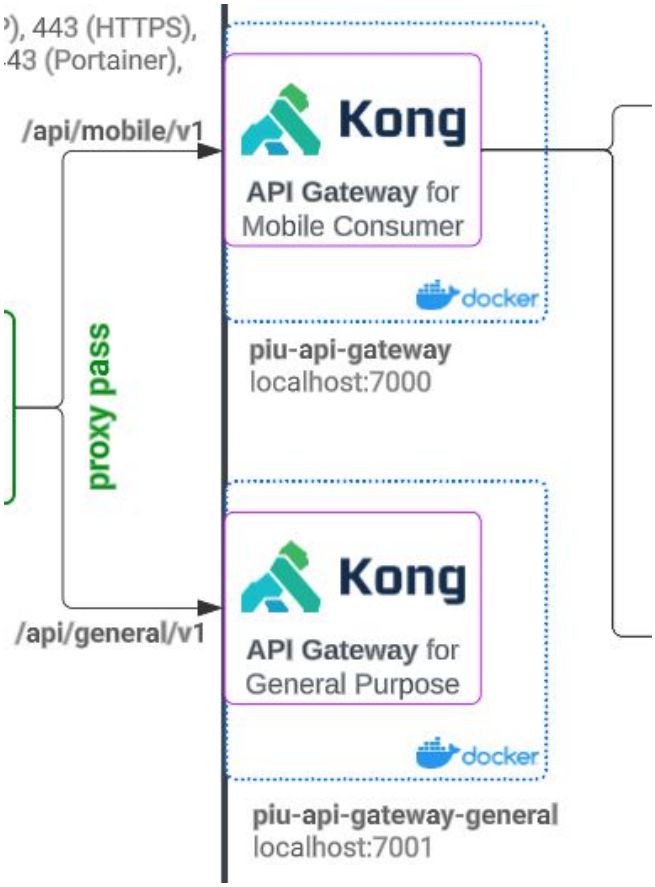
diperlukan agar bisa menggunakan atau melakukan *consume* terhadap API.

Dengan menggabungkan *framework* Codeigniter 4, pola desain *Repository Pattern*, bahasa pemrograman PHP 8, dan penggunaan Docker, pengembangan RESTful API dalam konteks arsitektur *microservices* ini menjadi lebih efisien, terstruktur, dan mudah dijalankan dalam lingkungan yang terkontrol. Sehingga secara keseluruhan terdapat 39 *endpoint* yang dibuat dalam penelitian ini seperti yang ditunjukkan pada Tabel 2.

B. API Gateway

API Gateway menjadi komponen penting yang mengelola lalu lintas antara *client* dan *microservices* yang ada. Dalam penelitian ini, *API Gateway* menggunakan Kong Gateway, sebuah platform yang populer untuk mengelola lalu lintas API. Kong Gateway sebenarnya berjalan di atas Nginx, yang berperan sebagai *reverse proxy server* yang digunakan oleh Kong untuk menangani permintaan HTTP dan mengarahkannya ke *service* yang sesuai. Dalam arsitektur Kong, Nginx bertindak sebagai mesin yang menjalankan fungsi inti dari *API Gateway*. Kong Gateway menggunakan Nginx sebagai mesin penanganan permintaan HTTP karena Nginx telah terbukti sangat cepat, andal, dan tangguh dalam mengelola lalu lintas web. Nginx memiliki kemampuan memuat dan membagi lalu lintas dengan efisien, menangani permintaan bersamaan dengan baik, dan dapat diandalkan untuk menangani volume tinggi dari permintaan yang masuk. Kong Gateway memanfaatkan kemampuan Nginx untuk melakukan penyebaran lalu lintas, penanganan SSL/TLS, *caching*, *logging*, dan manajemen lalu lintas HTTP. Dengan menggabungkan Nginx dengan fitur-fitur khusus yang ditambahkan oleh Kong, seperti manajemen API, autentikasi, otorisasi, dan pemantauan, Kong Gateway menjadi solusi yang cukup lengkap untuk mengelola dan menjembatani API dalam arsitektur *microservices*.

Dengan menggunakan Kong Gateway dalam mode *db-less* dan dijalankan dalam kontainer Docker, *API Gateway* dalam arsitektur *microservices* ini menjadi lebih fleksibel, skalabel, dan mudah dikelola. Konfigurasi *API Gateway* dapat



Gambar 13. Backend-for-frontend Pattern

didefinisikan secara deklaratif, memudahkan pengelolaan dan *version controlling*. Selain itu, penggunaan Docker memastikan konsistensi lingkungan pelaksanaan *API Gateway* di berbagai tahap pengembangan dan operasionalisasi sistem. Dibandingkan dengan *Kong Gateway* dalam mode tradisional atau *database mode*, mode tersebut membutuhkan *database*. Jika menggunakan mode ini, maka konfigurasi API tidak akan masuk dalam Git karena tidak ditulis pada file konfigurasi melainkan langsung tercatat didalam *database*. Meskipun mode ini memungkinkan beberapa *plugin* yang membutuhkan *database* bisa berjalan dan ini merupakan cara yang paling sederhana untuk menjalankan Kong, namun dalam studi kasus ini *plugin-plugin* tersebut tidak dibutuhkan dan pengembangan lebih mengutamakan *versioning* yang terstruktur.

C. Backend-for-frontend Pattern

Dalam arsitektur *microservices* ini, pola yang digunakan untuk *API Gateway* adalah *Backend-for-frontend* (BFF) seperti yang ditunjukkan pada Gambar 13. Pola ini dipilih karena pengembangan API difokuskan pada konsumsi oleh aplikasi *mobile*. *Backend-for-frontend* (BFF) adalah pendekatan di mana setiap jenis klien memiliki *gateway* yang khusus untuk mengakses layanan *backend* yang diperlukan. *Backend-for-frontend* sebenarnya adalah salah satu pola dari *API Gateway*. Dibandingkan dengan *API Gateway* yang hanya menyediakan *single entry point*, BFF menyediakan *multiple*

entry point.

API Gateway yang menggunakan pola *Backend-for-frontend* (BFF) berfungsi sebagai lapisan antara aplikasi mobile dan microservices yang ada di backend. Tujuan utama dari pola ini adalah untuk mengoptimalkan dan menyederhanakan interaksi antara aplikasi mobile dan backend. Sehingga pattern seperti ini cocok untuk pengembangan backend yang memiliki kebutuhan yang berbeda untuk setiap jenis platform client yang berbeda juga.

D.Interaksi antar komponen

Dalam arsitektur yang dikembangkan, *containerization* merupakan pendekatan yang digunakan untuk mengisolasi dan menjalankan setiap komponen dalam wadah yang terpisah, yang dikenal sebagai *container*. Setiap komponen utama, seperti *API Gateway*, dan *service* lainnya, dapat dijalankan dalam kontainer tersendiri. Konsep *containerization* memungkinkan setiap komponen untuk memiliki lingkungan yang terisolasi, termasuk dependensi dan konfigurasi yang diperlukan, sehingga memudahkan dalam pengembangan, *deploy*, dan skalabilitas sistem secara efisien.

Karena sudah berada dalam lingkungan terkontainerisasi, agar setiap komponen bisa saling berinteraksi terdapat dua metode komunikasi yang relevan, yaitu *synchronous communication* dan *asynchronous communication*. *Synchronous communication* terjadi ketika sebuah komponen menunggu respon langsung dari komponen lainnya sebelum melanjutkan eksekusi. Dalam arsitektur ini, metode *synchronous communication* digunakan saat permintaan klien harus menunggu hingga respons dari *service* diterima sebelum dapat melanjutkan ke langkah lain. Pendekatan ini memastikan keberhasilan eksekusi permintaan sebelum melanjutkan proses selanjutnya. Namun, disini terjadi ketergantungan antara dua *service* atau lebih dimana jika salah satu *service* mengalami kegagalan maka seluruh operasi yang melibatkan *service-service* ini tidak akan bisa dilanjutkan. Ketergantungan ini lazim disebut sebagai *Domain Coupling*. Meskipun demikian, dibeberapa kejadian cara berkomunikasi secara *synchronous* sangat dibutuhkan karena timbal balik sesegera mungkin yang harus dipenuhi.

Di sisi lain, *asynchronous communication* terjadi ketika komponen mengirim pesan tanpa harus menunggu respon langsung dari komponen lainnya. Cara berkomunikasi seperti ini lazim disebut *Hit and Forget*. Dalam arsitektur ini, metode *asynchronous communication* mungkin digunakan dalam kasus komunikasi antara-*service* yang tidak membutuhkan timbal balik sesegera mungkin yang harus dipenuhi. Sebagai contoh, ketika *service A* mengirim pesan ke *service B* untuk memicu suatu tindakan atau memperbarui status, pesan tersebut dikirim secara asinkron tanpa harus menunggu respons langsung. Service B bisa saja mendapat pesan dari *service A* setelah beberapa waktu, misalnya dalam 15 menit. Pendekatan ini dapat meningkatkan kinerja dan skalabilitas sistem, karena komponen tidak terikat dengan ketergantungan langsung terhadap respons dari komponen lainnya. Pendekatan seperti ini juga sangat baik diterapkan pada *service* yang rentan mengalami kegagalan fungsi atau hanya dijadwalkan berjalan tiap beberapa waktu. Sebagai contoh apabila saat *service A* mengirim pesan kepada *service B* dan ternyata *service B* sedang *down*, maka ketika *service B* kembali *up* pesan yang dikirim oleh *service A* bisa langsung diterima oleh *service B*. Hal ini tidak akan berlaku pada komunikasi *synchronous*, dimana jika *service A* mengirim pesan kepada *service B* saat sedang *down*, maka pesan akan hilang ditengah jalan dan akan terjadi *error* seketika pada *service A*. Dapat diketahui juga bahwa pada *asynchronous communication*, *service* pengirim pesan tidak dapat memastikan secara langsung jika pesannya sudah sampai pada *service* tujuan, inilah yang dimaksud oleh istilah *Hit and Forget*.

Dengan adanya *containerization* dan pilihan komunikasi yang sesuai, arsitektur ini dapat memanfaatkan kelebihan dari masing-masing konsep tersebut. *Containerization* memungkinkan isolasi dan manajemen yang lebih baik terhadap setiap komponen dalam sistem, sementara *synchronous communication* dan *asynchronous communication* memberikan fleksibilitas dalam interaksi antar komponen, tergantung pada sifat dan kebutuhan fungsionalitas yang diinginkan.

IV. KESIMPULAN DAN SARAN

Berdasarkan penelitian yang telah dilakukan maka dapat ditarik kesimpulan bahwa pengembangan arsitektur *microservices* ini berhasil menghasilkan 8 *service* dan 39 *endpoint* yang bisa digunakan untuk diakses oleh aplikasi Portal INSPIRE UNSRAT pada perangkat *mobile* menggunakan REST API. Pengembangan sistem terdistribusi seperti ini dengan menggunakan *Software Development Lifecycle* Scrum sangat cocok karena mendukung salah satu tujuan utama dari pengembangan secara *microservices*, yaitu kecepatan pengembangan dan perbaikan *service*.

Pola *backend-for-frontend* pada *microservices* ini hanya dibuat untuk *API Consumer* berupa *platform mobile*, untuk pengembangan selanjutnya bisa dilakukan untuk *API Consumer* pada platform desktop dan web. Kemudian dalam arsitektur *microservices* yang lebih besar, *service-service* akan ditempatkan di berbagai tempat dan lingkungan jaringan yang berbeda sehingga perlu diteliti lebih lanjut mengenai interaksi *service* sebesar ini, misalnya implementasi dalam bentuk *service mesh*.

V.KUTIPAN

[1] L. De Lauretis, “From monolithic architecture to microservices architecture,” in *Proceedings - 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019*, Institute of Electrical and Electronics Engineers Inc., Oct. 2019, pp. 93–96. doi: 10.1109/ISSREW.2019.00050.

[2] J. T. Zhao, S. Y. Jing, and L. Z. Jiang, “Management of API Gateway Based on Micro-service Architecture,” in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Oct. 2018. doi: 10.1088/1742-6596/1087/3/032032.

[3] R. Lolong, A. Sambul, and A. Lumenta, "Email Client Development In UNSRAT Inspire Portal With Gmail Service Integration," *Jurnal Teknik Elektro dan Komputer* , vol. 11, no. 1, pp. 35–44, 2022.

[4] M. Rezaldy, I. Asror, and I. L. Sardi, "Desain dan Analisis Arsitektur Microservices Pada Sistem Informasi Akademik Perguruan Tinggi Dengan Pendekatan Architecture Tradeoff Analysis Method (ATAM) (Studi Kasus: iGracias Universitas Telkom)," *e-Proceeding of Engineering*, vol. 4, no. 2, 2017.

[5] M. F. Radhiyan, "Analisis dan Desain Arsitektur Microservices dengan GraphQL Sebagai API Gateway untuk Sistem Informasi Akademik AIS UIN Jakarta (Studi Kasus : AIS untuk Mahasiswa)," 2020.

[6] B. Pranata, A. Hijriani, and A. Junaidi, "Perancangan Application Programming Interface (Api) Berbasis Web Menggunakan Gaya Arsitektur Representational State Transfer (Rest) Untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit," *Jurnal Komputasi*, vol. 6, no. 1, pp. 33–42, 2018.

[7] P. L. L. Belluano, P. Purnawansyah, B. L. E. Panggabean, and H. Herman, "Sistem Informasi Program Kreativitas Mahasiswa berbasis Web Service dan Microservice," *ILKOM Jurnal Ilmiah*, vol. 12, no. 1, pp. 8–16, Apr. 2020, doi: 10.33096/ilkom.v12i1.492.8-16.

[8] S. P. Utomo, N. H. Alfiyah, Z. A. Sani, M. Hanafi, and A. Primadewi, "Perancangan RESTFul Web Service Pada Sistem Informasi Terintegrasi Menggunakan FrameWork CodeIgniter," *Seminar Nasional Dinamika Informatika*, vol. 4, no. 1, 2020.

[9] A. Tedyyana, M. Fauzi, and F. Ratnawati, "Revamp Keamanan Web Service Milik PT XYZ Menggunakan REST API," *Digital Zone: Jurnal Teknologi Informasi & Komunikasi*, vol. 12, no. 1, pp. 1–10, 2021, doi: 10.31849/digitalzone.v12i1.6378ICCS.

[10] M. Danil Rafiqi, E. Subyantoro, and D. W. Kania, "Implementasi Arsitektur Microservice Pada Aplikasi Online Travel Tourinc," *Karya Ilmiah Mahasiswa Manajemen Informatika*, 2019.

[11] A. Mubariz *et al.*, "Perancangan Back-End Server Menggunakan Arsitektur Rest dan Platform Node.JS (Studi Kasus: Sistem Pendaftaran Ujian Masuk Politeknik Negeri Ujung Pandang)," 2020.

[12] H. Suryotrisongko, "Arsitektur Microservice untuk Resiliensi Sistem Informasi," *Jurnal Sisfo*, vol. 06, no. 02, pp. 235–250, 2017.

[13] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University Of California, 2000.

[14] S. Alkhodary, "The Evaluation of Using Backend-For-Frontend in a Microservices Environment," 2022.

[15] M. Blonteng, A. Sambul, and S. Paturusi, "Analysis of User Experience in University Academic Portal Using System Usability Scale (A Case Study in INSPIRE Portal of Sam Ratulangi University)," *Jurnal Teknik Informatika* , vol. 17, no. 3, pp. 213–218, 2022.

[16] K. Schwaber, "SCRUM Development Process," in *Business Object Design and Implementation*, 1997, pp. 117–134. [Online]. Available: http://vai1.al.adzona.edu/rugby/rad/rookie

Fransiscus Xaverius Senduk lahir di Manado pada tanggal 2 Februari 2002. Penulis menempuh pendidikan pada tahun 2007 di SD Katolik Santo Agustinus Warembungan, pada tahun 2013 melanjutkan pendidikan di SMP Katolik Hati Kudus Yesus Karombasan, kemudian pada tahun 2016 melanjutkan pendidikan di SMA Katolik Santo Ignatius Malalayang. Pada tahun 2019, penulis melanjutkan studi di jenjang sarjana di Program Studi Teknik Informatika, Fakultas Teknik, Jurusan Teknik Elektro, Universitas Sam Ratulangi Manado.

Selama perkuliahan penulis tergabung dalam organisasi kemahasiswaan dan aktif berkegiatan dalam Keluarga Mahasiswa Katolik (KMK) Santo Agustinus FT-UNSRAT, Himpunan Mahasiswa Elektro (HME) FT-UNSRAT, dan Unit Kegiatan Mahasiswa Pers dan Penyiaran FT-UNSRAT. Di tingkat program studi penulis juga tergabung dalam komunitas UNSRAT IT Community (UNITY).