LAPORAN PROJECT UJIAN AKHIR SEMESTER
ARSITEKTUR KOMPUTER DAN SISTEM OPERASI

Disusun oleh :

Alexa Azzahra Prasetyo	**(25031554024)**
Carissa Regina Putri Yania	**(25031554149)**
Nabila Putri Misbahul	**(25031554172)**
Jovan Rio Fernando	**(25031554017)**

UNIVERSITAS NEGERI SURABAYA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

PROGRAM STUDI S1 SAINS DATA

2025-2026

DAFTAR ISI

BAB I
PENDAHULUAN

A. Latar Belakang

Seiring berkembangnya teknologi, sistem informasi menjadi semakin rumit dan membutuhkan arsitektur perangkat lunak yang bisa menyesuaikan diri dengan kebutuhan pengguna yang terus meningkat. Sistem monolitik yang dulunya banyak digunakan mulai menunjukkan kelemahannya, terutama dalam hal skalabilitas, perawatan, dan pengembangan yang berkelanjutan. Ketika ada satu bagian dari sistem yang mengalami masalah atau perlu diperbaharui, biasanya seluruh sistem harus dimatikan terlebih dahulu. Hal inilah yang mendorong munculnya pendekatan arsitektur baru yang lebih fleksibel dan modular, salah satunya adalah arsitektur microservice.[1]

Arsitektur microservice memecah sebuah aplikasi besar menjadi kumpulan layanan kecil yang berdiri sendiri dan saling berkomunikasi melalui antarmuka layanan atau API. Setiap layanan memiliki tanggung jawab yang spesifik sehingga dapat dikembangkan, diuji, dan dideploy secara terpisah. Menurut Dragoni dkk., pendekatan ini mampu meningkatkan keandalan sistem serta mempercepat proses pengembangan karena tim dapat bekerja secara paralel pada layanan yang berbeda. [2]Selain itu, microservice juga mempermudah proses pemeliharaan karena kesalahan pada satu layanan tidak secara langsung memengaruhi layanan lainnya.

Dalam penerapan arsitektur microservice, teknologi container menjadi salah satu komponen penting. Docker merupakan platform containerization yang banyak digunakan karena mampu menyediakan lingkungan eksekusi yang konsisten dan ringan. Dengan Docker, aplikasi dan seluruh dependensinya dapat dikemas dalam sebuah container sehingga dapat dijalankan pada berbagai lingkungan tanpa perbedaan konfigurasi. Pahl menyatakan bahwa penggunaan container seperti Docker sangat mendukung microservice karena

[1] Lewis, J., & Fowler, M. (2014). Microservices: a definition of this new architectural term. https://martinfowler.com/articles/microservices.html
[2] Dragoni, N., et al. (2017). Microservices: Yesterday, Today, and Tomorrow. Lecture Notes in Computer Science, Springer. https://doi.org/10.1007/978-3-319-67425-4_12

mempermudah proses deployment, isolasi layanan, serta pengelolaan sumber daya sistem[3].[3]

Berdasarkan hal tersebut, pemahaman mengenai microservice dan Docker menjadi penting dalam konteks pembelajaran Arsitektur Komputer dan Sistem Operasi. Melalui proyek Ujian Akhir Semester ini, mahasiswa diharapkan mampu menerapkan konsep microservice secara praktis dengan memanfaatkan Docker dan docker-compose. Implementasi ini tidak hanya memperkuat pemahaman teori, tetapi juga memberikan pengalaman langsung dalam mengelola layanan terdistribusi yang mendekati kondisi sistem nyata di dunia industri.

B. Rumusan Masalah

1. Bagaimana menerapkan konsep arsitektur microservice pada sebuah sistem layanan sederhana agar setiap layanan dapat berjalan secara terpisah namun tetap saling terhubung?
2. Bagaimana menggunakan Docker dan docker-compose untuk menjalankan dan mengelola beberapa layanan dalam satu sistem secara terintegrasi?
3. Bagaimana memastikan seluruh layanan yang dibangun dapat berjalan dengan baik sesuai dengan ketentuan dan kebutuhan pada tugas Ujian Akhir Semester?

C. Tujuan

1. Menerapkan konsep arsitektur microservice pada sistem layanan sederhana sesuai dengan materi yang telah dipelajari.
2. Menggunakan Docker dan docker-compose untuk mengonfigurasi, menjalankan, dan mengintegrasikan seluruh layanan dalam satu sistem.
3. Menguji dan memastikan bahwa seluruh layanan dapat berjalan dengan baik sesuai dengan ketentuan yang diberikan pada tugas Ujian Akhir Semester.

[3] Pahl, C. (2015). Containerization and the PaaS Cloud. IEEE Cloud Computing, 2(3), 24–31. https://doi.org/10.1109/MCC.2015.51

BAB II
DESKRIPSI SISTEM & ARSITEKTUR

A. Arsitektur Microservice

Arsitektur microservices merupakan pendekatan pengembangan perangkat lunak yang membagi sistem besar menjadi beberapa layanan kecil yang berdiri sendiri. Setiap layanan memiliki fungsi spesifik dan dijalankan sebagai proses terpisah, namun tetap dapat saling berkomunikasi melalui API. Pendekatan ini dikembangkan untuk mengatasi keterbatasan sistem monolitik, di mana seluruh fungsi digabung dalam satu aplikasi sehingga perubahan atau kesalahan pada satu bagian dapat memengaruhi keseluruhan sistem dan memerlukan proses deployment ulang.

Dengan arsitektur microservices, proses pengembangan dapat dilakukan secara lebih fleksibel dan terstruktur. Setiap layanan dapat dikembangkan, diperbarui, atau diperbaiki secara independen tanpa mempengaruhi layanan lain. Misalnya, ketika ingin update Auth Service, tidak perlu deploy ulang semua layanan—cukup Auth Service saja. Selain itu, jika terjadi kesalahan pada satu layanan, layanan lain tetap berjalan normal (fault isolation), sehingga sistem lebih stabil. Microservices juga memudahkan scaling karena hanya layanan yang membutuhkan resource lebih yang perlu di-scale up, bukan seluruh sistem.

Pada proyek ini, konsep microservices diterapkan dengan memisahkan layanan berdasarkan fungsinya, seperti Auth Service untuk menangani otentikasi dan otorisasi pengguna, serta API Gateway sebagai pintu masuk dan routing layer. Setiap layanan dijalankan dalam container Docker yang terpisah dan dihubungkan melalui jaringan internal Docker (bridge network), sehingga sistem lebih modular, mudah dikelola, dan aman. Dengan pendekatan ini, setiap komponen sistem bisa di-maintain, di-test, dan di-deploy secara independen,

yang sangat membantu dalam proses development dan maintenance jangka panjang.

B. API Gateway

API Gateway merupakan bagian yang memiliki peran penting, yakni sebagai pintu masuk utama bagi klien untuk mengakses berbagai layanan yang ada dalam sistem. Jadi, semua permintaan dari klien akan masuk dulu ke API Gateway sebelum diteruskan ke layanan backend yang dituju. Dengan adanya API Gateway, klien tidak perlu mengetahui detail teknis dari setiap layanan, seperti alamat IP, port, atau protokol yang digunakan. Ini membuat interaksi klien dengan sistem menjadi lebih sederhana dan terstruktur.

Penggunaan API Gateway sangat membantu dalam arsitektur sistem, terutama ketika jumlah layanan yang ada terus bertambah. Dalam kondisi ketika klien harus terhubung langsung ke puluhan layanan yang berbeda—tentu akan sangat merepotkan dan sulit dikelola. API Gateway bertugas mengatur routing request secara otomatis, mengelola endpoint dengan lebih rapi, dan berfungsi sebagai lapisan awal untuk keamanan sistem. Misalnya, validasi token atau mengaktifkan akses bisa diterapkan di level gateway sebelum permintaan sampai ke layanan utama. Pendekatan ini juga membuat sistem pengelolaan menjadi lebih mudah karena komunikasi antar komponen menjadi lebih ringkas dan terorganisir.

Dalam implementasi proyek ini, API Gateway digunakan untuk menghubungkan klien dengan Auth Service dan layanan backend lainnya. API Gateway diimplementasikan menggunakan Nginx karena Nginx terkenal ringan, stabil, dan sudah banyak dipakai dalam sistem microservice di industri. Selain itu, Nginx juga cukup mudah dikonfigurasi dan performanya bagus untuk menangani banyak permintaan secara bersamaan.

C. Docker & Docker Compose

Docker merupakan platform containerization yang digunakan untuk menjalankan aplikasi dalam sebuah lingkungan terisolasi yang disebut

container. Dengan konsep ini, aplikasi tidak lagi bergantung secara langsung pada sistem operasi yang digunakan, karena seluruh kebutuhan aplikasi sudah dikemas di dalam container. dengan begitu, Docker ditujukan untuk membantu mengurangi masalah perbedaan lingkungan antara proses pengembangan dan deployment.

Selain itu, Docker juga mempermudah proses distribusi aplikasi karena container dapat dijalankan di berbagai sistem selama Docker tersedia. Setiap container berjalan secara terpisah sehingga tidak saling mengganggu satu sama lain. Pendekatan ini sangat sesuai dengan arsitektur microservice yang menekankan pemisahan fungsi antar layanan.

Docker Compose digunakan untuk mengelola dan menjalankan beberapa container secara bersamaan. Melalui satu file konfigurasi, pengembang dapat mendefinisikan layanan, jaringan, volume, serta ketergantungan antar container. Pada proyek ini, Docker Compose berperan penting dalam mengintegrasikan API Gateway, Auth Service, dan database agar dapat berjalan sebagai satu kesatuan sistem.

D. RESTful API

RESTful API merupakan pendekatan dalam membangun layanan berbasis web yang menggunakan protokol HTTP sebagai media komunikasi antara client dan server. Setiap layanan menyediakan endpoint yang dapat diakses oleh client menggunakan metode HTTP standar seperti GET (untuk mengambil data), POST (untuk mengirim data baru), PUT (untuk memperbarui data), dan DELETE (untuk menghapus data). Pendekatan ini sangat populer dan banyak digunakan karena sederhana, fleksibel, mudah dipahami, dan bisa diintegrasikan dengan berbagai platform atau bahasa pemrograman.

Salah satu karakteristik utama dari RESTful API adalah sifatnya yang stateless, yang artinya server tidak menyimpan informasi atau context dari sesi client sebelumnya. Setiap request yang masuk akan diproses secara independen berdasarkan data yang dikirimkan saat itu. Ini membuat sistem menjadi lebih ringan dan mudah di-scale. Data yang dikirim atau diterima dalam RESTful API umumnya menggunakan format JSON (JavaScript Object Notation) karena

formatnya ringan, mudah dibaca baik oleh manusia maupun mesin, dan sudah menjadi standar industri.

Dalam arsitektur sistem microservice, RESTful API punya peran penting sebagai penghubung komunikasi antar layanan yang berbeda. Karena setiap microservice berjalan secara independen, mereka perlu cara untuk saling berkomunikasi dan bertukar data. Pada proyek ini, RESTful API digunakan sebagai mekanisme komunikasi antara API Gateway dan Auth Service. Dengan menggunakan RESTful API, alur request dan response dapat berjalan dengan jelas dan terstruktur, serta memudahkan proses testing dan debugging karena setiap endpoint bisa diuji secara terpisah.

E. FastAPI

FastAPI merupakan framework Python modern yang digunakan untuk membangun RESTful API dengan performa tinggi dan struktur kode yang rapi serta mudah dipelihara. Framework ini dirancang dengan fokus pada kemudahan penggunaan dan kecepatan development, terutama dalam pembuatan layanan backend yang membutuhkan validasi data ketat dan response yang konsisten. FastAPI juga terkenal sangat cepat karena dibangun di atas Starlette untuk handling web dan Pydantic untuk validasi data, sehingga performanya bahkan bisa menyaingi framework berbasis Node.js atau Go.

Salah satu keunggulan FastAPI adalah kemampuannya dalam melakukan validasi data secara otomatis berdasarkan *type hints* Python yang didefinisikan pada kode. Dengan menentukan tipe data yang diharapkan, FastAPI akan memeriksa setiap request yang masuk dan secara otomatis memberikan respon error jika data tidak sesuai, tanpa perlu validasi manual. Hal ini membantu mengurangi kesalahan input dan meningkatkan keandalan layanan. Selain itu, FastAPI juga menyediakan dokumentasi API otomatis berbasis OpenAPI seperti Swagger UI dan ReDoc, yang memudahkan proses pengujian, debugging, serta kolaborasi dalam pengembangan.

Pada proyek ini, FastAPI digunakan untuk membangun Auth Service yang menangani proses autentikasi dan otorisasi pengguna. Penggunaan FastAPI membuat proses pembuatan endpoint seperti login, register, dan

validasi token menjadi lebih terstruktur, mudah dipahami, dan cepat dikembangkan. FastAPI juga sangat cocok untuk implementasi microservice pada skala pembelajaran karena sintaksnya yang clean dan dokumentasinya yang lengkap, sehingga memudahkan mahasiswa untuk memahami konsep API development secara praktis.

F. PostgreSQL

PostgreSQL merupakan sistem manajemen basis data relasional (RDBMS) yang digunakan untuk menyimpan dan mengelola data terstruktur. Database ini mendukung penggunaan tabel, relasi antar tabel, transaksi ACID (Atomicity, Consistency, Isolation, Durability), dan query SQL yang kompleks. PostgreSQL banyak digunakan dalam industri karena stabil, open-source, memiliki performa yang baik, dan mendukung berbagai fitur advanced seperti indexing, stored procedures, dan JSON support.

Dalam konteks arsitektur microservice, PostgreSQL biasanya digunakan untuk layanan yang membutuhkan konsistensi data yang tinggi dan integritas referensial yang ketat. Data yang disimpan bersifat terstruktur dengan skema yang jelas dan memiliki hubungan antar tabel yang terdefinisi dengan baik, misalnya relasi one-to-many atau many-to-many. PostgreSQL juga sangat cocok untuk aplikasi yang memerlukan transaksi kompleks dan keamanan data yang terjamin.

Meskipun pada proyek ini PostgreSQL tidak digunakan secara langsung pada layanan utama, pemahaman mengenai PostgreSQL tetap penting sebagai bagian dari pengetahuan tentang sistem backend yang umum digunakan dalam pengembangan aplikasi modern, terutama dalam sistem microservice yang memerlukan persistent storage untuk menyimpan data user, transaksi, atau informasi penting lainnya.

G. MongoDB

MongoDB merupakan basis data NoSQL yang menyimpan data dalam bentuk dokumen berformat JSON-like (BSON). Berbeda dengan database relasional yang menggunakan tabel dan baris, MongoDB menyimpan data

dalam collection dan document yang strukturnya lebih fleksibel. Kita nggak perlu mendefinisikan skema yang ketat dari awal, sehingga sangat cocok untuk data yang strukturnya sering berubah atau bervariasi antar dokumen. MongoDB juga mendukung operasi CRUD (Create, Read, Update, Delete) yang cepat dan mudah, serta bisa melakukan query yang kompleks termasuk aggregation dan indexing.

Penggunaan MongoDB mempermudah pengelolaan data otentikasi karena struktur data pengguna umumnya sederhana dan tidak membutuhkan banyak relasi seperti pada database relasional. Data pengguna dapat disimpan dalam satu dokumen yang mencakup informasi seperti username, password, email, dan role tanpa perlu proses *join*. MongoDB juga mudah diintegrasikan dengan layanan backend berbasis JavaScript maupun Python karena menggunakan format JSON, serta memiliki performa yang baik untuk operasi baca dan tulis.

Pada sistem ini, MongoDB digunakan sebagai Auth Database untuk menyimpan data pengguna seperti username, password yang sudah di-hash, dan informasi akun lainnya. Database ini dijalankan dalam container Docker terpisah dan dihubungkan dengan Auth Service melalui jaringan internal Docker (bridge network), sehingga komunikasi antar container tetap aman dan terisolasi dari jaringan luar. Dengan setup seperti ini, Auth Service bisa mengakses MongoDB tanpa harus expose port database ke public, yang meningkatkan keamanan sistem secara keseluruhan.

H. JSON Web Token (JWT)

JSON Web Token (JWT) merupakan metode otentikasi berbasis token yang banyak digunakan pada sistem modern, terutama pada arsitektur microservices dan RESTful API. JWT digunakan untuk menyimpan informasi pengguna (seperti user ID, role, atau permission) dalam bentuk token yang telah di-encode dan di-sign secara digital. Token ini terdiri dari tiga bagian: header (berisi algoritma enkripsi), payload (berisi data user), dan signature (untuk memastikan token tidak diubah). Token JWT dikirimkan setiap kali client

melakukan request ke endpoint yang memerlukan autentikasi, biasanya melalui HTTP header dengan format Authorization: Bearer <token>.

Penggunaan JWT membuat sistem tidak perlu menyimpan sesi pengguna di server (stateless), karena semua informasi yang diperlukan sudah ada di dalam token itu sendiri. Hal ini membuat sistem lebih ringan, efisien, dan mudah di-scale, terutama dalam sistem microservices yang terdiri dari banyak layanan. Server hanya perlu memverifikasi apakah token valid dan belum expired, tanpa harus query ke database untuk mengecek sesi. Selama token masih valid dan belum kadaluarsa, pengguna dapat mengakses layanan sesuai dengan hak aksesnya tanpa harus login ulang. JWT juga aman karena menggunakan signature yang memastikan token tidak bisa diubah oleh pihak yang tidak berwenang.

Dalam proyek ini, JWT digunakan untuk mengamankan akses pada Auth Service dan layanan backend lainnya. Setelah pengguna berhasil login dengan username dan password yang benar, sistem akan menghasilkan JWT token yang berisi informasi user dan waktu expired. Token ini kemudian dikirimkan ke client dan digunakan untuk proses otorisasi pada setiap request berikutnya. Dengan menggunakan JWT, sistem dapat memastikan bahwa hanya user yang sudah terautentikasi yang bisa mengakses endpoint tertentu.

BAB III
IMPLEMENTASI SISTEM

A. Konfigurasi Docker Compose

Docker Compose merupakan alat yang digunakan untuk menjalankan dan mengelola beberapa container Docker secara bersamaan melalui satu file konfigurasi. Pada sistem ini, Docker Compose berfungsi untuk mengatur seluruh service microservices agar dapat berjalan dalam satu lingkungan yang saling terhubung.

Konfigurasi docker-compose.yml mendefinisikan service API Gateway, Auth Service, Auth Database, Academic Service, dan Academic Database. Setiap service dijalankan dalam satu network internal yang sama sehingga dapat saling berkomunikasi tanpa harus membuka seluruh port ke publik.

Service API Gateway menggunakan Nginx sebagai reverse proxy yang berperan sebagai pintu masuk utama sistem. Port 8081 pada host dipetakan ke port 80 di dalam container agar client dapat mengakses sistem melalui API Gateway.

Auth Service dikonfigurasi sebagai service autentikasi yang dibangun dari source code lokal. Service ini bergantung pada Auth Database (MongoDB) dan menggunakan environment variable untuk mengatur port aplikasi, koneksi database, serta secret key untuk token autentikasi.

Auth Database menggunakan MongoDB dengan volume penyimpanan persisten agar data autentikasi tidak hilang ketika container dihentikan. Database ini hanya diakses oleh Auth Service melalui jaringan internal Docker.

Academic Service dikembangkan menggunakan FastAPI dan berfungsi untuk mengelola data akademik mahasiswa. Service ini terhubung ke Academic Database (PostgreSQL) melalui konfigurasi environment variable yang mendefinisikan host, port, nama database, serta kredensial.

Academic Database menggunakan PostgreSQL sebagai penyimpanan data akademik. Data disimpan secara persisten menggunakan Docker volume sehingga tetap aman meskipun container di-restart.

Seluruh service dihubungkan melalui network microservices-network dengan driver bridge. Penggunaan network ini bertujuan untuk menjaga komunikasi antar service tetap terisolasi dan terstruktur.

B. API Gateway

API Gateway berperan sebagai pintu masuk utama bagi seluruh request dari client. Dalam sistem ini, API Gateway digunakan untuk menerima request HTTP dari browser dan meneruskannya ke Academic Service sesuai dengan endpoint yang diakses.

Fungsi utama API Gateway adalah menyederhanakan akses client terhadap service internal. Client tidak perlu mengetahui detail alamat container atau port service, karena seluruh komunikasi dilakukan melalui API Gateway. Selain itu, API Gateway juga berfungsi sebagai lapisan keamanan awal dengan membatasi akses langsung ke service internal.

Dengan adanya API Gateway, arsitektur sistem menjadi lebih rapi dan mudah dikembangkan. Jika di kemudian hari terdapat penambahan service baru, maka perubahan hanya perlu dilakukan pada konfigurasi API Gateway tanpa memengaruhi sisi client.

C. Auth Service

Academic Service merupakan service utama yang menangani logika bisnis sistem akademik. Service ini dikembangkan menggunakan framework FastAPI dan dijalankan di dalam container Docker. Academic Service bertanggung jawab dalam pengelolaan data mahasiswa, perhitungan IPS, serta penyajian antarmuka dashboard dan halaman login.

Konfigurasi koneksi database pada service ini didefinisikan melalui variabel lingkungan (environment variable), seperti DB_HOST, DB_PORT, DB_NAME, DB_USER, dan DB_PASSWORD. Pendekatan ini memudahkan proses konfigurasi tanpa perlu mengubah kode program secara langsung.

```
 DB_CONFIG = {
"host": os.getenv("DB_HOST", "acad-db"),
"port": os.getenv("DB_PORT", "5432"),
"database": os.getenv("DB_NAME", "products"),
"user": os.getenv("DB_USER", "productuser"),
"password": os.getenv("DB_PASSWORD", "productpass"),
}
```

Potongan kode di atas menunjukkan bahwa service mengambil konfigurasi database dari environment variable. Hal ini bertujuan untuk menjaga fleksibilitas dan keamanan konfigurasi sistem.

Academic Service juga menyediakan beberapa endpoint API penting, di antaranya endpoint untuk menampilkan data mahasiswa, menampilkan nilai mata kuliah, serta menghitung Indeks Prestasi Semester (IPS).

```
@app.get("/api/acad/ips")
async def get_ips(nim: str):
```

Endpoint tersebut digunakan untuk menghitung IPS mahasiswa berdasarkan bobot nilai dan jumlah SKS. Perhitungan dilakukan di sisi backend sehingga hasil yang ditampilkan kepada pengguna bersifat konsisten dan valid.

Selain API, Academic Service juga menyajikan halaman login dan dashboard mahasiswa dalam bentuk HTML. Proses autentikasi dilakukan secara sederhana dengan memanfaatkan cookie session untuk menentukan apakah pengguna sudah login atau belum.

D. Auth Database

Database yang digunakan dalam sistem ini adalah PostgreSQL. Database dijalankan dalam container terpisah untuk menjaga pemisahan antara aplikasi dan penyimpanan data. Struktur database dirancang untuk menyimpan data mahasiswa, mata kuliah, serta nilai akademik.

Struktur tabel mahasiswa digunakan untuk menyimpan identitas dasar mahasiswa, seperti NIM, nama, jurusan, dan angkatan.

```
CREATE TABLE mahasiswa (
nim VARCHAR(10) PRIMARY KEY,
nama VARCHAR(100) NOT NULL,
jurusan VARCHAR(50),
angkatan INT
);
tabel mata_kuliah digunakan untuk menyimpan informasi mata kuliah
beserta jumlah SKS.
CREATE TABLE mata_kuliah (
kode_mk VARCHAR(10) PRIMARY KEY,
nama_mk VARCHAR(100) NOT NULL,
sks INT NOT NULL
);
```

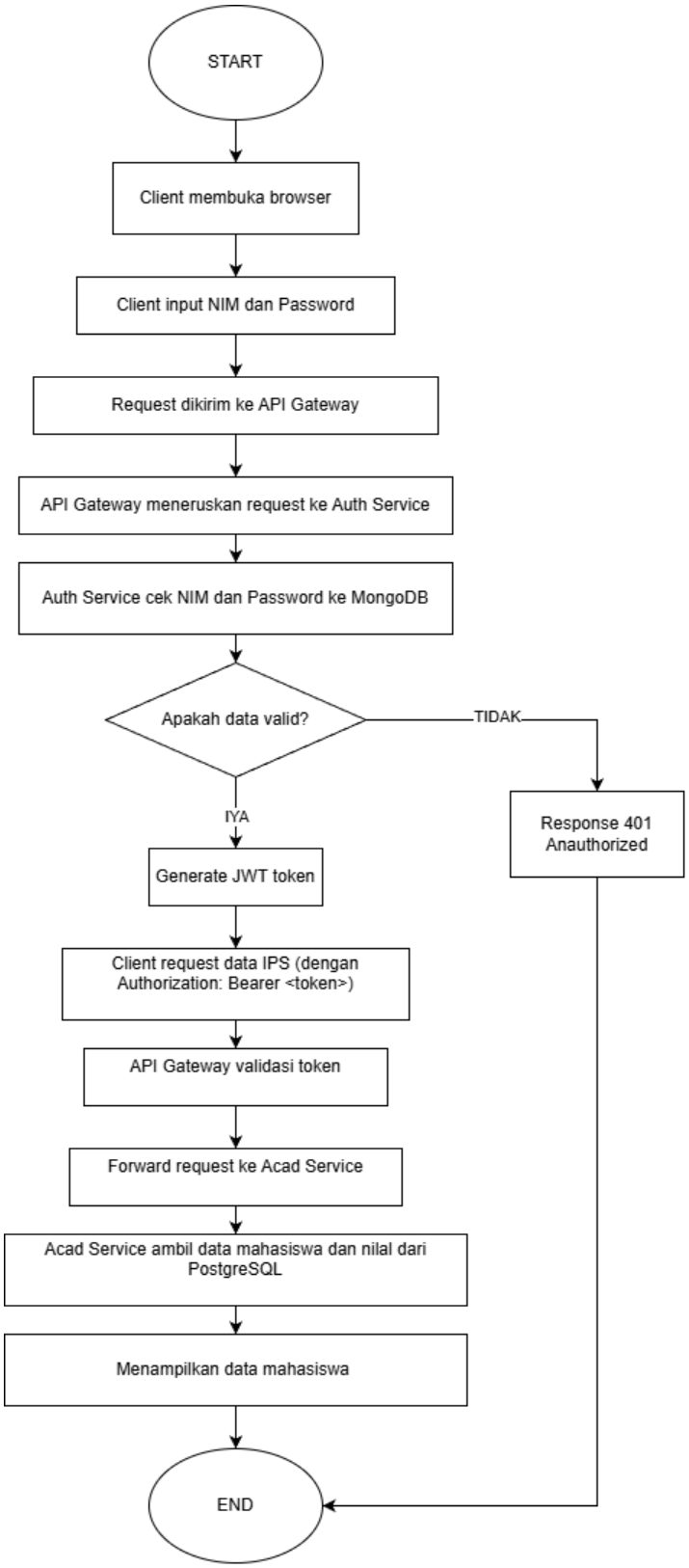
Relasi antara mahasiswa dan mata kuliah disimpan pada tabel krs yang mencatat nilai dan semester pengambilan mata kuliah.

```
CREATE TABLE krs (
id_krs bigserial PRIMARY KEY,
nim VARCHAR(10),
kode_mk VARCHAR(10),
nilai CHAR(2),
semester INT,
FOREIGN KEY (nim) REFERENCES mahasiswa(nim),
FOREIGN KEY (kode_mk) REFERENCES mata_kuliah(kode_mk)
);
```

Dengan struktur tersebut, sistem dapat melakukan query untuk menampilkan data akademik mahasiswa serta menghitung IPS secara akurat. Database hanya dapat diakses oleh Academic Service melalui jaringan internal Docker, sehingga keamanan data tetap terjaga.

Melalui implementasi sistem yang telah dijelaskan pada bab ini, seluruh komponen dapat berjalan secara terintegrasi. Penggunaan Docker, API Gateway, service terpisah, serta database containerized memberikan kemudahan dalam pengelolaan sistem dan mendukung pengembangan aplikasi yang lebih terstruktur dan skalabel.

E. Flowchart


START
Client membuka browser
Client input NIM dan Password
Request dikirim ke API Gateway
API Gateway meneruskan request ke Auth Service
Auth Service cek NIM dan Password ke MongoDB
Apakah data valid?
TIDAK
IYA
Response 401 Anauthorized
Generate JWT token
Client request data IPS (dengan Authorization: Bearer <token>)
API Gateway validasi token
Forward request ke Acad Service
Acad Service ambil data mahasiswa dan nilal dari PostgreSQL
Menampilkan data mahasiswa
END

BAB IV
HASIL IMPLEMENTASI DAN PENGUJIAN

A. Hasil implementasi

Pada tahap ini dilakukan implementasi sistem layanan akademik berbasis arsitektur microservices menggunakan teknologi Docker dan Docker Compose. Sistem terdiri dari beberapa sub-layanan yang berjalan pada container terpisah namun saling terhubung melalui internal network Docker.

Sub-layanan yang berhasil diimplementasikan adalah sebagai berikut:

1. **API Gateway** menggunakan Nginx sebagai pintu masuk utama layanan.
2. **Auth Service** untuk pengelolaan autentikasi pengguna.
3. **Acad Service** untuk pengelolaan data akademik dan perhitungan IPS mahasiswa.
4. **Database Auth** menggunakan MongoDB.
5. **Database Akademik** menggunakan PostgreSQL.

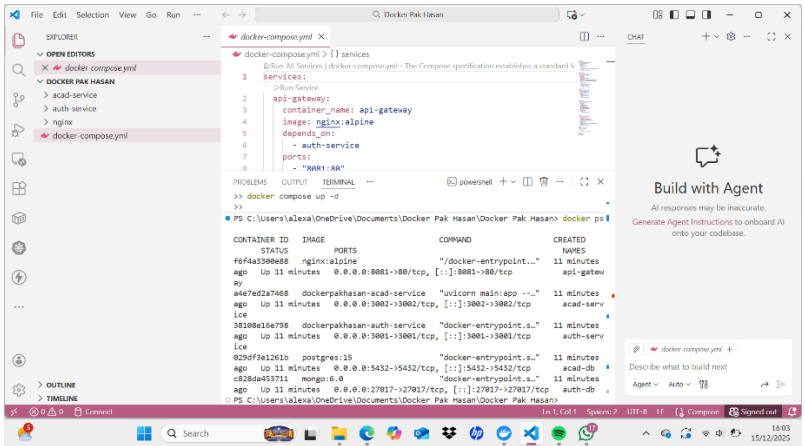
Seluruh layanan dijalankan secara bersamaan menggunakan perintah `docker compose up -d` dan berhasil berjalan tanpa error. API Gateway berjalan pada port **8081**, Auth Service pada port **3001**, dan Acad Service pada port **3002**.

B. Pengujian Sistem

Pengujian sistem dilakukan untuk memastikan bahwa seluruh layanan dapat berjalan sesuai dengan perancangan dan saling terintegrasi dengan baik.

1. Pengujian Container Docker

   Pengujian ini bertujuan untuk memastikan seluruh containers berjalan dengan baik. Pengujian dilakukan dengan menggunakan `docker ps`

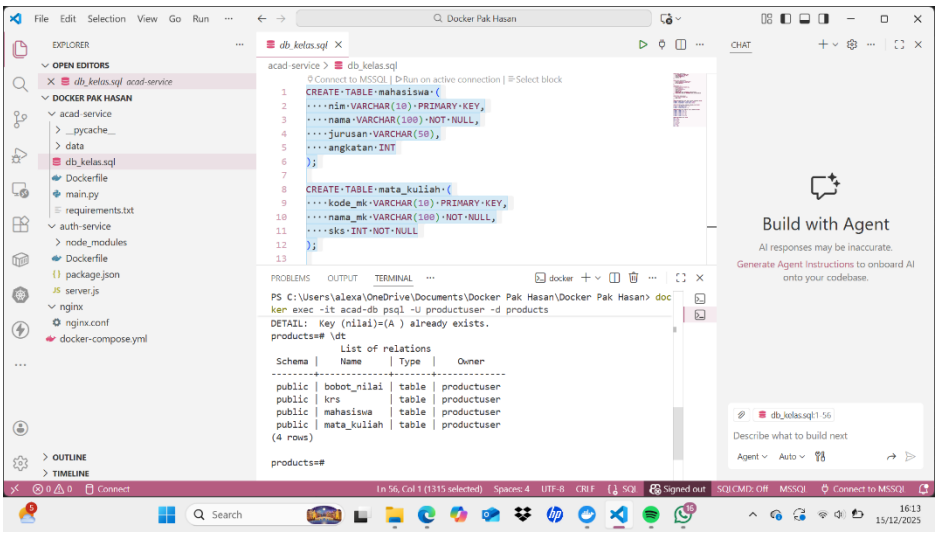


Hasil pengujian menunjukkan bahwa seluruh container berada dalam status running, termasuk API Gateway, Auth Service, Acad Service, MongoDB, dan PostgreSQL. Hal ini menunjukkan bahwa konfigurasi Docker Compose telah berhasil dijalankan.

2. Pengujian Koneksi Database

   Pengujian koneksi database dilakukan dengan melihat log pada masing-masing service. Pada Acad Service, koneksi ke PostgreSQL berhasil ditunjukkan dengan tidak adanya error saat aplikasi dijalankan dan query database dapat dieksekusi dengan baik.

   Selain itu, pengujian juga dilakukan dengan mengakses database PostgreSQL melalui container menggunakan perintah: docker exec -it acad-db psql -U productuser -d products

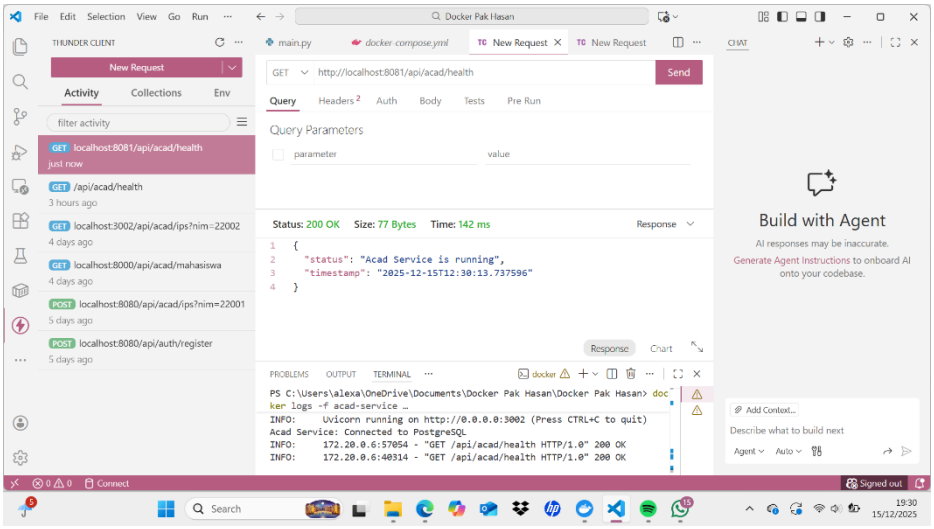


   Hasilnya menunjukkan bahwa tabel mahasiswa, mata_kuliah, bobot_nilai dan krs berhasil dibuat dan dapat diakses.

3. Pengujian endpoint API dilakukan menggunakan **Thunder Client** pada Visual Studio Code. Beberapa endpoint yang diuji antara lain:

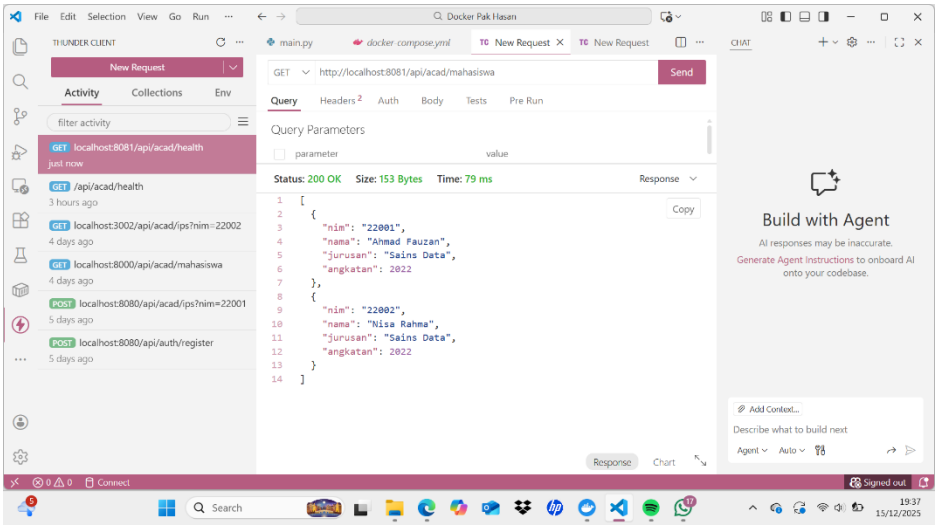
   a. Health Check Acad Service

      1) Endpoint: `/api/acad/health`
      2) Metode: GET
      3) Hasil: Service mengembalikan status aktif.



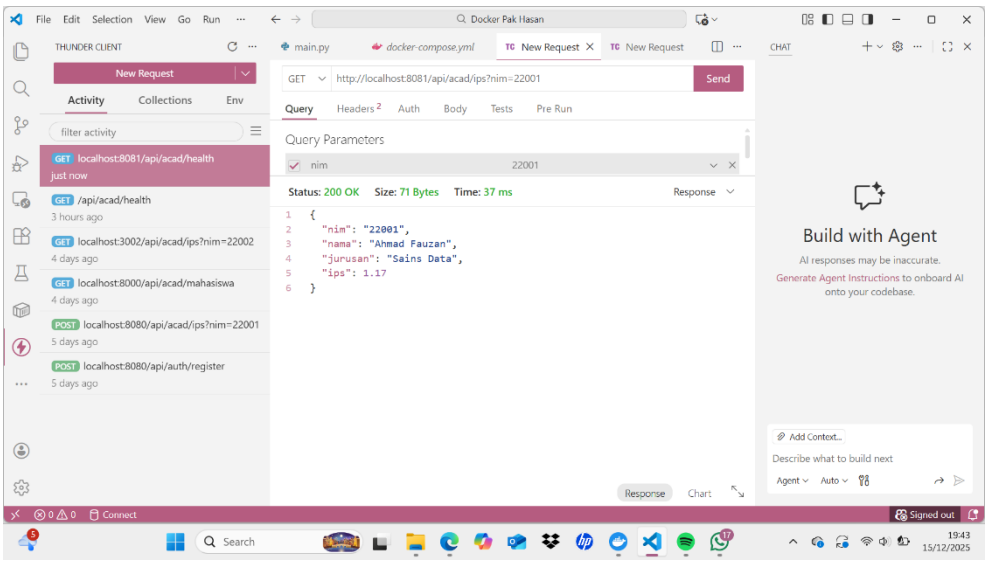
b. Data Mahasiswa

1) Endpoint: `/api/acad/mahasiswa`
2) Metode: GET
3) Hasil: Sistem menampilkan daftar mahasiswa dari database PostgreSQL.



c. Perhitungan IPS Mahasiswa

1) Endpoint: `/api/acad/ips?nim=22002` atau `/api/acad/ips?nim=22001`
2) Metode: GET



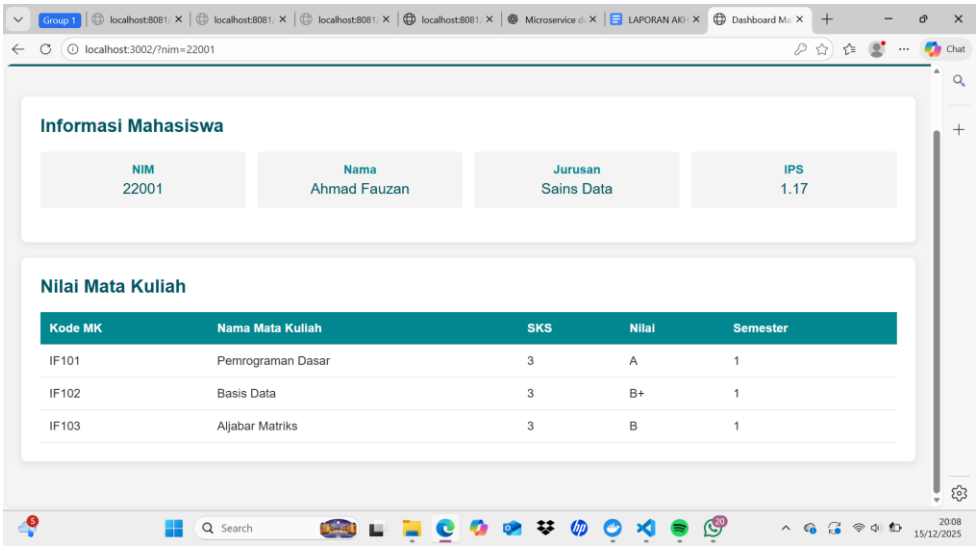
3) Hasil: Sistem berhasil menghitung dan menampilkan nilai IPS mahasiswa sesuai data KRS.

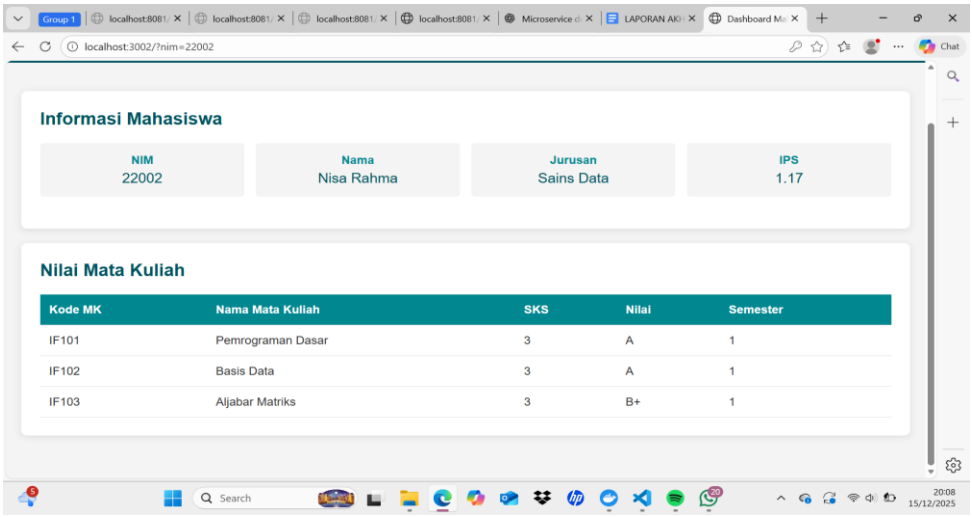
d. Pengujian Perhitungan IPS

Pengujian perhitungan IPS dilakukan dengan menggunakan data nilai mahasiswa yang tersimpan pada tabel KRS. Sistem menghitung IPS berdasarkan bobot nilai dan jumlah SKS setiap mata kuliah.

Bobot nilai yang digunakan antara lain:

- A = 4.0
- A- = 3.75
- B+ = 3.5
- B = 3.0
- B- = 2.75
- C+ = 2.5
- C = 2.0
- D = 1.0
- E = 0.0





Hasil pengujian menunjukkan bahwa nilai IPS yang dihasilkan sistem sesuai dengan perhitungan manual berdasarkan data nilai dan SKS mahasiswa.

e. Ringkasan Hasil Pengujian

No	Pengujian	Hasil
1	Menjalankan container Docker	Berhasil
2	Koneksi database PostgreSQL	Berhasil
3	Endpoint Health Check	Berhasil
4	Endpoint Data Mahasiswa	Berhasil
5	Endpoint Perhitungan IPS	Berhasil

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, dapat disimpulkan bahwa sistem layanan akademik berbasis microservices telah berhasil diimplementasikan dan berjalan sesuai dengan kebutuhan. Seluruh sub-layanan dapat berkomunikasi dengan baik dan menghasilkan output yang sesuai dengan perancangan sistem.

BAB V
PENUTUP

A. Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian sistem yang telah dilakukan, dapat ditarik beberapa kesimpulan sebagai berikut:

1. Sistem backend berhasil dibangun menggunakan arsitektur microservices, di mana setiap layanan memiliki fungsi dan tanggung jawab yang terpisah, sehingga sistem menjadi lebih terstruktur dan mudah dikembangkan.
2. Penggunaan Docker dan Docker Compose terbukti mampu mempermudah proses deployment aplikasi karena setiap service dijalankan dalam container yang terisolasi dan dapat dijalankan secara bersamaan hanya dengan satu perintah.
3. API Gateway berbasis Nginx berhasil diimplementasikan sebagai pintu masuk utama sistem, yang berfungsi untuk meneruskan request dari client ke service yang sesuai, sehingga client tidak perlu mengetahui detail masing-masing service.
4. Sistem terdiri dari beberapa service dengan teknologi yang berbeda, yaitu Auth Service menggunakan Node.js dan Acad Service menggunakan FastAPI (Python), yang menunjukkan bahwa arsitektur microservices mendukung penggunaan multi-teknologi dalam satu sistem.
5. Berdasarkan hasil pengujian endpoint seperti health check, pengambilan data mahasiswa, dan perhitungan IPS, sistem dapat berjalan dengan baik dan sesuai dengan kebutuhan yang telah ditentukan.

B. Saran

Berdasarkan sistem yang telah dibuat masih terdapat beberapa hal yang perlu dikembangkan di masa mendatang agar sistem menjadi lebih baik, oleh karena itu terdapat saran sebagai berikut :

1. Kedepannya, sistem ini dapat ditambahkan fitur keamanan agar hanya pengguna tertentu yang dapat mengakses sistem, sehingga data yang ada menjadi lebih aman.

2. Sistem dapat dikembangkan dengan menambahkan pencatatan aktivitas atau error, sehingga apabila terjadi kesalahan pada sistem, pengembang dapat mengetahui letak permasalahannya dengan lebih mudah.
3. Pengelolaan database pada setiap *service* dapat dipisahkan agar masing-masing *service* dapat berjalan secara lebih mandiri dan tidak saling bergantung.
4. Proses pengembangan sistem dapat dibuat lebih praktis dengan menerapkan otomatisasi, sehingga proses menjalankan dan memperbarui sistem menjadi lebih cepat dan efisien.
5. Sistem ini dapat dikembangkan dengan menambahkan tampilan antarmuka agar pengguna dapat mengakses sistem dengan lebih mudah tanpa harus menggunakan tools khusus.

DAFTAR PUSTAKA

Docker Inc. (2023). Docker Documentation.

Dragoni, N., et al. (2017). Microservices: Yesterday, Today, and Tomorrow. Lecture Notes in Computer Science, Springer. https://doi.org/10.1007/978-3-319-67425-4_12

FastAPI. (2023). FastAPI Documentation.

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. University of California.

Lewis, J., & Fowler, M. (2014). Microservices: a definition of this new architectural term. https://martinfowler.com/articles/microservices.html

IETF. (2015). RFC 7519: JSON Web Token (JWT).

Pahl, C. (2015). Containerization and the PaaS Cloud. IEEE Cloud Computing, 2(3), 24–31. https://doi.org/10.1109/MCC.2015.51