LAPORAN AKHIR PROYEK (DOCKER) ARSITEKTUR KOMPUTER DAN SISTEM OPERASI

Disusun Oleh:

Faris Riky Pratama	: 25031554176
Mochammad Fawwaz Muslih	: 25031554126
Adinda Atsila Salsabilah	: 25031554262

Dosen Pengampu:
Hasanudin Al-Habib, M.Si.

UNIVERSITAS NEGERI SURABAYA 2025

DAFTAR ISI

DAFTAR GAMBAR

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Di era digital saat ini, tuntutan terhadap pengembangan dan operasional sistem (DevOps) semakin meningkat. Sistem tidak hanya dituntut untuk berfungsi secara baik, tetapi juga mudah dikembangkan, dipelihara, serta mampu beradaptasi dengan kebutuhan skalabilitas. Pendekatan arsitektur monolitik telah menjumpai berbagai tantangan, seperti terdampaknya keseluruhan sistem akibat perubahan kecil yang dilakukan pada salah satu bagian. Kondisi ini tentunya memperlambat proses pengembangan dan operasional sistem.

Sebagai respon dari permasalahan tersebut, *microsevice* hadir guna memecah sistem yang kompleks menjadi beberapa bagian kecil. Setiap bagian memiliki fungsi yang lebih spesifik dan mudah dikembangkan secara independen. Dengan adanya *microservice*, proses pengembangan dan operasional aplikasi menjadi lebih modular, kolaboratif, dan adaptif terhadap perubahan.

Untuk mempermudah penerapan *microservice*, teknologi *container* hadir sebagai solusi yang mampu menjamin konsistensi lingkungan sistem dari tahap pengembangan hingga produksi. Docker merupakan salah satu teknologi *container* yang banyak digunakan di seluruh dunia. Docker memungkinkan setiap bagian sistem dikemasi dalam suatu container yang ringan, portabel, dan terisolasi. Dengan adanya docker, proses pengembangan, pengujian, dan deployment dapat dilakukan secara efisien dan handal.

Dalam konteks perkuliahan, tugas diberikan kepada mahasiswa sebagai bentuk uji pemahaman dari materi arsitektur komputer dan sistem operasi yang telah dipaparkan sebelumnya, khususnya terkait penerapan arsitektur microservice dan virtualisasi *container*.. Tugas ini mengimplementasikan konsep *microservice* yang terbagi menjadi dua sub-layanan yaitu pengelolaan akun sederdana (*Auth Service*) dan perhitungan IPS mahasiswa (*Academic Service*). Kemudian, mahasiswa dituntut untuk melengkapi konfigurasi sub-layanan tersebut, mengatur komunikasi antar container, serta memahami hubungan antara sistem operasi, virtualisasi container, dan arsitektur perangkat lunak modern. Oleh karena itu, laporan akhir ini disusun sebagai bentuk pertanggungjawaban akademik atas penyelesaian tugas yang diberikan.

1.2 Tujuan

1. Melakukan konfigurasi microservice menggunakan docker agar seluruh layanan dapat berjalan secara terintegrasi.
2. Menerapkan query database dan pemrosesan data dalam aplikasi backend.

3. Mengimplementasikan logika perhitungan IPS sesuai aturan akademik menggunakan data KRS dan SKS.
4. Memahami konsep response API berbentuk JSON yang siap dikonsumsi oleh aplikasi lain.
5. Melatih penerapan praktik backend yang baik, seperti validasi data dan penanganan error.

BAB 2
ISI DAN PEMBAHASAN

2.1 Penambahan Konfigurasi pada Docker Compose

Projek ini berfokus pada implementasi arsitektur *micrservice* menggunakan docker dan docker compose. Secara fungsional, sistem *microservice* ini terbagi menjadi dua, yaitu auth-service dan acad-service. Pada bagian ini, akan dijelaskan bagaimana keseluruhan sistem bekerja setelah beberapa konfigurasi. Setiap sub-layanan sistem dijalankan menggunakan container yang berbeda yang kemudian diorkestrasi dalam satu kesatuan melalui Docker compose. Berikut beberapa penjelasan mengenai kompoen utama setelah ditambahkan beberapa konfigurasi dalam arsitektur *microservice* ini.

2.1.1 API Gateaway (Nginx)

```yaml
services:
  api-gateway:
    image: nginx:alpine
    container_name: api-gateway
    depends_on:
      - auth-service
      - acad-service
    ports:
      - "8081:80"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    networks:
      - microservices-network
```

Gambar 2.1.1

Penjelasan:

- Service digunakan untuk mendefinisikan sekumpulan layanan yang akan dijalankan oleh Docker compose.
- Api-gateway berfungsi sebagai titik masuk utama (gateway) untuk seluruh permintaan (request) ke sistem microservices, memfasilitasi routing, autentikasi, dan pengelolaan komunikasi antar layanan secara terpusat.
- Image digunakan sebagai cetakan atau template untuk membuat container. Dengan image, Docker bisa menjalankan aplikasi lengkap dengan sistem operasi, dependensi, dan konfigurasi tanpa harus menginstal manual di setiap komputer.
- Image nginx:alpine digunakan karena ringan dan efesien dari Nginx yang ideal untuk container.Image ini memiliki footprint kecil namun performa tinggi, sesuai standar praktik terbaik DevOps.
- Container adalah instansi atau wadah yang berjalan dari sebuah Docker image, yang menjalankan aplikasi secara terisolasi. Container memiliki filesystem

sendiri,konfigurasi,dan dependensi yang dibutuhkan aplikasi, namun tetap berbagi kernel dengan host.

- Container_name dijalankan dengan nama api-gateway untuk memudahkan identifikasi dan pengelolaan melalui Docker, terutama saat monitoring dan troubleshooting.
- Depends_on adalah parameter di Docker Compose untuk menentukan urutan startup service agar tidak error saat startup, menyederhanakan manajemen microservices, menentukan ketergantungan antar service.
- Dependensi layanan Layanan bergantung pada `auth-service` dan `acad-service`, sehingga container `api-gateway` hanya dijalankan setelah kedua layanan siap. Hal ini menjamin routing ke backend berjalan lancar tanpa error karena layanan belum tersedia.
- Ports adalah saluran komunikasi yang memungkinkan container berinteraksi dengan host atau dunia luar, sehingga aplikasi di dalam container bisa diakses melalui port tertentu di host.
- Port `80` di container dipetakan ke port `8081` di host. Artinya, akses melalui `http://localhost:8081` akan diteruskan ke Nginx di container. (Sebelumnya port host menggunakan `8080`, kini diubah menjadi `8081` untuk menghindari konflik atau kebutuhan port lain.)
- volume adalah ruang penyimpanan yang bisa diakses oleh container dan host. Volume digunakan untuk menyimpan data atau file konfigurasi agar tidak hilang ketika container dihentikan atau dihapus.
- File konfigurasi Nginx (`nginx.conf`) di-host di `./nginx/` dan di-mount ke container dengan mode read-only (`ro`) di `/etc/nginx/nginx.conf`. Konfigurasi ini memungkinkan pengelolaan terpusat dari host tanpa memodifikasi container.
- Networks adalah jaringan virtual yang menghubungkan container agar bisa saling berkomunikasi secara aman dan terisolasi. Juga menyederhanakan pengaturan alamat,cukup menggunakan nama service sebagai hostname(microservices-network).

2.1.2 Konfigurasi Auth Service

```yaml
auth-service:
  build: ./auth-service
  container_name: auth-service
  cpus: 0.5
  mem_limit: 512m
  depends_on:
    - auth-db
  environment:
    - NODE_ENV=development
    - PORT=3001
    - MONGO_URI=mongodb://auth-db:27017/auth
    - JWT_SECRET=6cedd4e3de39ace0d401e12e9661bac17ec15e485d7a8d219ee72233c75208e2
  ports:
    - "3001:3001"
  networks:
    - microservices-network
  volumes:
    - ./auth-service:/app
    - /app/node_modules
```

Gambar 2.1.2

Penjelasan:

- Mendefinisikan sebuah layanan bernama auth-service.
- Instruksi build digunakan agar Docker dapat membangun image berdasarkan Dockerfile yang terdapat pada direktori auth-service.
- Memberikan nama auth-service pada container menggunakan instruksi container_name.
- Parameter cpus dan mem_limit berfungsi secara berturut-turut untuk membatasi penggunaan CPU hingga setengah inti prosesor serta membatasi penggunaan memori sebesar 512 MB.
- Bagian depends_on digunakan agar Docker Compose menjalankan layanan auth data terlebih dahulu sebelum menjalankan layanan auth-service.
- Variabel lingkungan (environment) digunakan untuk memisahkan konfigurasi aplikasi dari kode program. Variabel NODE_ENV=development menandakan bahwa aplikasi dijalankan dalam mode pengembangan dengan port internal 3001. Variabel MONGO_URI=mongodb://auth-db:27017/auth berfungsi untuk menghubungkan aplikasi ke basis data MongoDB dengan auth-db sebagai hostname. Selanjutnya, JWT_SECRET digunakan sebagai kunci JSON Web Token yang berperan sebagai mekanisme utama dalam proses autentikasi.
- Konfigurasi ports digunakan untuk menghubungkan port yang berada di dalam container (3001) dengan port yang tersedia pada komputer (3001).
- Agar antar layanan dapat saling berkomunikasi, digunakan konfigurasi jaringan (network) bernama microservices-network.
- Volume pertama digunakan untuk menyinkronkan source code antara host dan container sehingga setiap perubahan kode dapat langsung diterapkan tanpa perlu membangun ulang image. Volume kedua digunakan untuk menyimpan

dependensi aplikasi secara terpisah guna menghindari konflik antara lingkungan host dan container.

2.1.3 Auth Database

```
auth-db:
  image: mongo:6.0
  container_name: auth-db
  restart: always
  networks:
    - microservices-network
  volumes:
    - ./auth-data:/data/db
  ports:
    - "27017:27017"
```

Gambar 2.1 3

Penjelasan:

- `auth-db` adalah database service yang menggunakan MongoDB untuk menyimpan data autentikasi pengguna. Database ini menjadi backend dari `auth-service`.
- Image yang digunakan mongo:6.0, MongoDB yang stabil dan mendukung performa tinggi.
- Container dijalankan dengan nama `auth-db` untuk memudahkan identifikasi dan manajemen melalui Docker.
- `restart: always` berarti container akan otomatis dijalankan ulang jika berhenti atau terjadi kegagalan, sehingga layanan database tetap tersedia.
- Networks adalah jaringan virtual di Docker yang menghubungkan container satu sama lain, sehingga container dapat saling berkomunikasi dengan aman dan terisolasi dari jaringan luar.Juga menyederhanakan pengaturan alamat,cukup menggunakan nama service sebagai hostname(microservices- network).
- `volumes: /auth-data:/data/db` artinya data MongoDB disimpan di host pada folder `/auth-data` dan di-mount ke container di `/data/db`. Dengan ini, data tetap aman meskipun container dihentikan atau dihapus.
- Port `27017` di container dipetakan ke port `27017` di host. Ini memungkinkan MongoDB diakses dari host melalui port yang sama, memudahkan koneksi dari aplikasi lain.

2.1.4 Acad Service

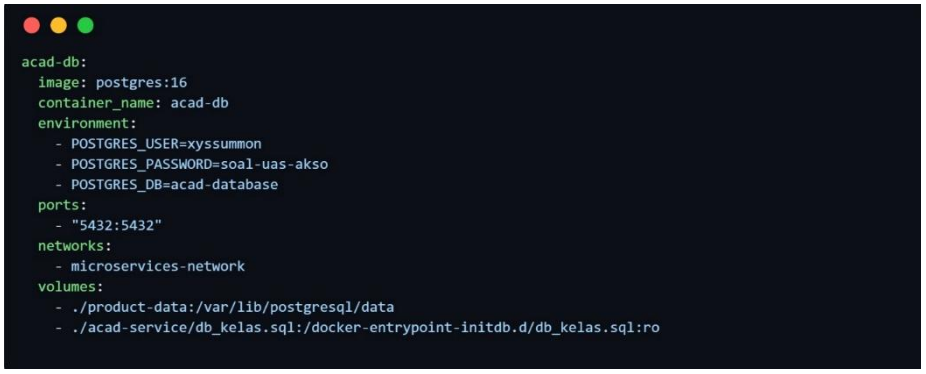
```yaml
acad-service:
  build: ./acad-service
  container_name: acad-service
  depends_on:
    - acad-db
  environment:
    - NODE_ENV=development
    - PORT=3002
    - DB_HOST=acad-db
    - DB_PORT=5432
    - DB_NAME=acad-database
    - DB_USER=xyssummon
    - DB_PASSWORD=soal-uas-akso
  ports:
    - "3002:3002"
  networks:
    - microservices-network
  volumes:
    - ./acad-service:/app
```

Gambar 2.1 4

Penjelasan:

- Acad-db dibangun menggunakan dockerfile yang berada di direktori ./acad-service/
- Bagian dependensi diatur agar docker compose dapat menjalankan layanan setelah auth-db diproses.
- Terdapat variabel lingkungan (*Environment*) yang digunakan untuk memisahkan konfigurasi aplikasi dari kode program. Menentukan nama database (DB_NAME), username database (DB_USER), password database (DB_PASSWORD), port internal aplikasi NODE.js, dan port *default* (DB_PORT).
- Port 3002 di container dipetakan ke port 3002 di host. Ini memungkinkan MongoDB diakses dari host melalui port yang sama, memudahkan koneksi dari aplikasi lain.
- Volume ./acad-service:/app artinya data dari host disimpan dalam folder /acad-service kemudian di mount ke container di /app.

2.1.5 Acad Database



Gambar 2.1 5

Penjelasan:

- `acad-db` adalah database service yang menggunakan PostgreSQL untuk menyimpan data akademik. Database ini menjadi backend dari `acad-service`.
- Menggunakan image `postgres:16`, versi terbaru PostgreSQL yang stabil dan mendukung performa tinggi.
- Container dijalankan dengan nama `acad-db` agar mudah diidentifikasi dan dikelola melalui Docker.
- Environment Variables untuk mengatur konfigurasi database melalui variabel lingkungan. POSTGRES_USER=xyssurmon untuk username database, POSTGRES_PASSWORD=soal-uas-akso untuk passaword database, POSTGRES_DB=acad-database untuk nama database yang dibuat otomatis saat container dijalankan.
- Port `5432` di container dipetakan ke port `5432` di host, sehingga PostgreSQL dapat diakses dari host atau aplikasi lain melalui port standar PostgreSQL.
- `Volume:/product-data:/var/lib/postgresql/data` untuk menyimpan data database agar tetap aman saat container dihentikan atau dihapus. `/acad-service/db_kelas.sql:/docker-entrypoint-initdb.d/db_kelas.sql:ro` untuk file SQL inisialisasi database di-mount dari host, otomatis saat container pertama kali start
- Network agar terhubung ke microservices-network agar container acad-db bisa berkomunikasi dengan layanan lain di arsitektur microservices secara aman dan terisolasi.

2.1.6 Konfigurasi Network

```
networks:
  microservices-network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.16.0.0/28
```

Gambar 2.1 6

Penjelasan:

- Network adalah jaringan khusus yang digunakan agar layanan satu dan yang lain dapat berinteraksi dalam docker compose.
- Microservices-network adalah nama yang digunakan untuk jaringan khsusu di service ini.
- Jenis network yang digunakan adalah driver: bridge yang memiliki fungsi membuat jaringan virtual internal dalam satu host, setiap container mendapatkan IP privat, dan menghubungkan container satu dengan lainnya.
- *IP Address Management* (IPAM) adalah mekanisme atau sistem dalam mengatur, mengalokasikan, dan mengelola alamat IP. IPAM digunakan untuk mengatur alamat IP secara manual dalam konteks ini.
- Konfigurasi IPAM (Config) adalah pengaturan pengelolaan alamat IP (IP Address Management) pada jaringan Docker. Digunakan untuk menentukan rentang IP (subnet misal 16, 24,dan 28), gateway,dan aturan pembagian IP.

2.1.7 Konfigurasi Volume

```
volumes:
  auth-data:
  product-data:
  order-data:
```

Gambar 2.1 7

Penjelasan:

Docker volume dibuat untuk memastikan data agar tidak hilang meskipun docker dihentikan atau dihapus. Auth-data dan product-data didefinisikan sebagai volume yang terpisah. Karena tidak ada konfigurasi tambahan dalam docker volume, maka docker akan otomatis menyimpan data di /var/lib/docker/volumes/. Auth-data volume digunakan untuk auth-service dan product-data volume digunakan untuk bagian acad-service serta order-data digunakan untuk fitur lainnya.

2.2 Penambahan Konfigurasi pada Main.py

Kode ini ditambahkan untuk menyediakan endpoint API yang berfungsi menghitung dan menampilkan IPS (Indeks Prestasi Semester) mahasiswa berdasarkan data KRS, nilai, dan

SKS yang tersimpan di database.

2.2.1 Endpoint API

Gambar 2.2. 1

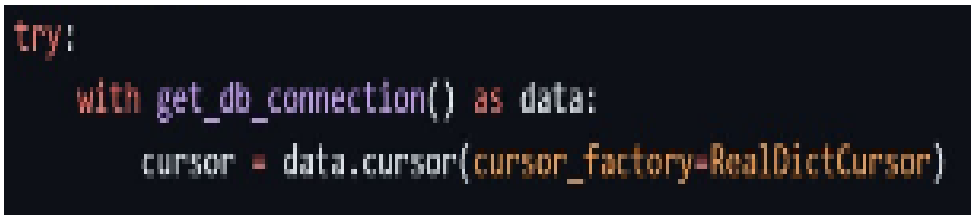
Penjelasan:
Konfigurasi ini digunakan untuk menambahkan endpoint HTTP GET yang dapat diakses melalui URL `/api/acad/ips` untuk mengambil data IPS mahasiswa.

2.2.2 Fungsi Asinkron

Gambar 2.2. 2

Mendefinisikan fungsi API yang bersifat **asinkron**, sehingga lebih efisien dalam menangani request secara bersamaan.

2.2.3 Koneksi Database



Gambar 2.2. 3

Membuka koneksi ke database dan menggunakan RealDictCursor agar hasil query berbentuk dictionary, sehingga mudah diakses berdasarkan kolom.

2.2.4 Query SQL

```
        query =
"select m.nim, m.nama, m.jurusan, krs.nilai, mk.sks  from mahasiswa m join krs on krs.nim = m.nim join mata
_kuliah mk ON mk.kode_mk = krs.kode_mk where m.nim = '22002'"
```

Gambar 2.2. 4

Query ini digunakan untuk mengambil data mahasiswa, mengambil nilai mata kuliah, mengambil jumlah SKS, dan menggabungkan data dari beberapa tabel untuk perhitungan IPS.

2.2.5 Eksekusi Query dan Validasi Data

```
cursor.execute(query)
rows = cursor.fetchall()

if not rows:
    raise HTTPException(status_code=404, detail="Data mahasiswa tidak ditemukan")
```

Gambar 2.2. 5

Berfungsi untuk mengambil semua data hasil query dan Kode `fetchall()` digunakan untuk mengambil seluruh hasil query dari database, sedangkan pengecekan `if not rows` berfungsi sebagai validasi data untuk memastikan bahwa data mahasiswa tersedia sebelum proses perhitungan IPS dilakukan. Jika data tidak ditemukan, API akan mengembalikan respons 404 untuk menjaga kestabilan dan keamanan sistem.

2.2.6 Konversi Nilai Huruf ke Angka

```
    nilai_map = {
        "A": 4,
        "B+": 3.5,
        "B": 3,
        "B-": 2.75,
        "C+": 2.5,
        "C": 2,
        "D": 1,
        "E": 0,
    }
```

Gambar 2.2. 6

Bagian ini adalah konfigurasi yang berfungsi untuk mengubah nilai huruf menjadi nilai angka dan digunakan sebagai dasar perhitungan IPS sesuai standar akademik.

2.2.7 Proses Perhitungan IPS

```
value_total = 0
sks_total = 0

for i in rows:
    value_in_angka = nilai_map.get(i["nilai"], 0)
    sks = i["sks"]

    value_total += value_in_angka * sks
    sks_total += sks

IPS = round(value_total / sks_total, 2) if sks_total > 0 else 0
```

Gambar 2.2. 7

Konfigurasi ini berfungsi untuk mengalikan nilai angka dengan SKS, menjumlahkan seluruh bobot nilai dan total SKS, menghitung IPS, dibulatkan hingga 2 angka desimal.

2.2.8 Response API

```
return {
    "NIM": rows[0]["nim"],
    "Nama Mahasiswa": rows[0]["nama"],
    "Jurusan": rows[0]["jurusan"],
    "IPS": IPS
}
```

Gambar 2.2. 8

API mengembalikan data terstruktur (JSON) yang berisi NIM mahasiswa, nama mahasiswa, jurusan, dan nilai IPS hasil perhitungan.

2.2.9 Penanganan Error

```
except Exception as e:
    raise HTTPException(status_code=500, detail=str(e))
```

Gambar 2.2. 9

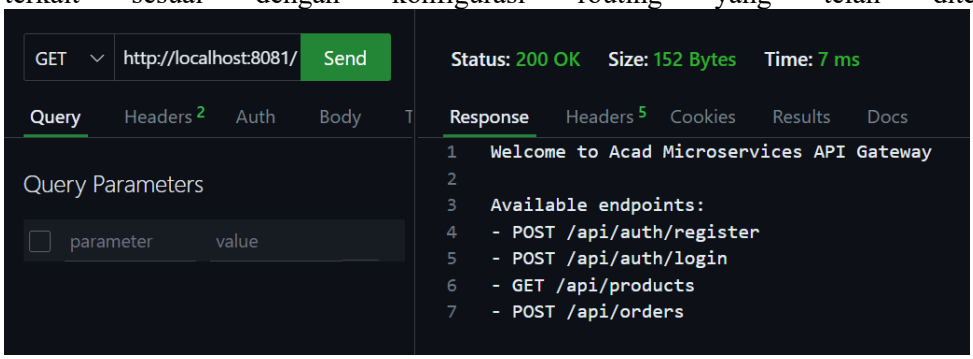
Digunakan untuk menangani error tak terduga dan mengembalikan status 500(internal Server Error) agar API tetap aman dan terkontrol.

2.3 API Testing

Setelah semua services berhasil dijalankan melalui docker compose, langkah selanjautnya adalah melakukan pengujian API menggunakan API Client. Kegiatan ini dilakukan untuk memastikan semua endpoint berjalan sesuai fungsinya. Alat yang akan digunakan dalam testing ini adalah Thunder Client.

1. Pengujian Endpoint API-Getaway

Pengujian endpoint pada API Gateway dapat dilakukan melalui alamat http://localhost:8081, karena API Gateway dijalankan dan diekspos melalui port 8081. Melalui alamat tersebut, seluruh permintaan klien akan diteruskan ke layanan backend terkait sesuai dengan konfigurasi routing yang telah ditentukan.

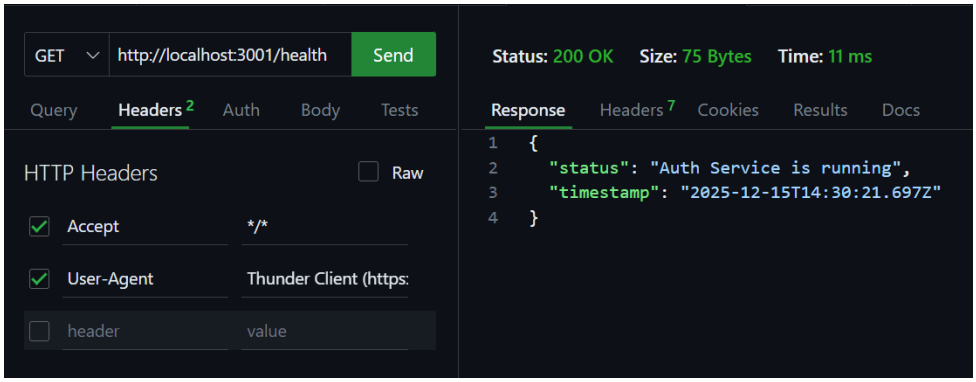


Gambar 2.3. 1

Hasil pengujian API menunjukkan bahwa permintaan berhasil diproses dengan status kode HTTP 200, yang menandakan keberhasilan pengujian pada API Gateway. Indikator keberhasilan lainnya ditunjukkan oleh waktu respons sekitar 7 ms serta struktur respons yang telah sesuai dengan spesifikasi yang diharapkan.

2. Pengujian Endpoint Auth-Service

Pengujian endpoint pada Auth-Service dapat dilakukan melalui endpoint http://localhost:3001/health, yang berfungsi sebagai *health check* untuk memastikan bahwa layanan autentikasi berjalan dengan baik dan siap menerima permintaan dari sistem.



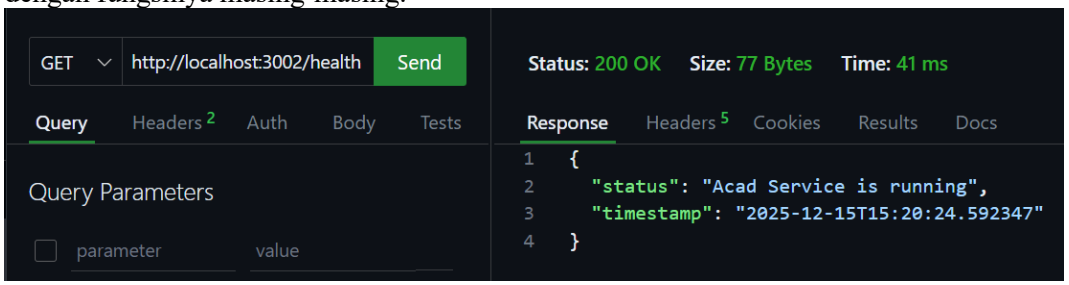
Gambar 2.3. 2

Hasil dari Test API untuk bagian Auth-service menunjukkan bahwa permintaan berhasil diproses dengan status kode HTTP 200, yang menandakan keberhasilan pengujian pada Auth-Serice. Kemudian, terdapat indikator lain seperti waktu respon

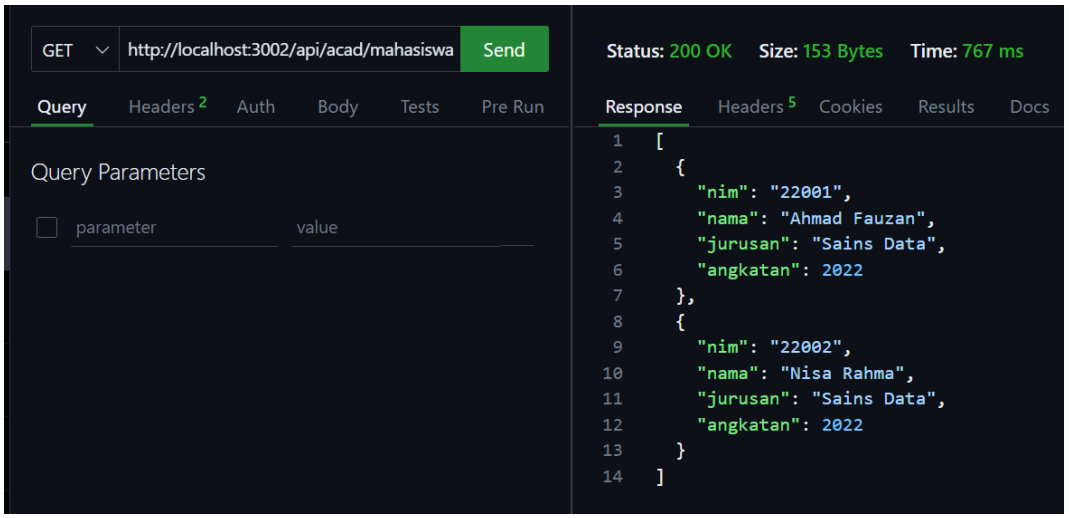
sekitar 11 ms dan struktur respon yang telah sesuai dengan yang diharapkan.

3. Pengujian Endpoint Acad-Service

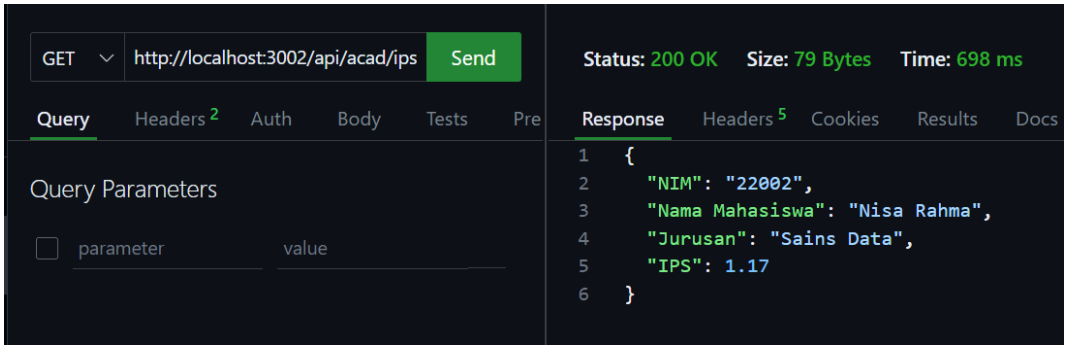
Pengujian endpoint pada Acad-Service dapat dilakukan dengan mengakses beberapa endpoint utama, antara lain endpoint *health* untuk memastikan layanan berjalan dengan baik, endpoint daftar mahasiswa (/api/acad/mahasiswa) untuk memverifikasi pengambilan data mahasiswa, serta endpoint nilai IPS (/api/acad/ips) untuk menguji pengambilan dan pemrosesan data indeks prestasi semester. Pengujian ini bertujuan untuk memastikan bahwa setiap endpoint dapat merespons permintaan dengan benar sesuai dengan fungsinya masing-masing.



Gambar 2.3. 3



Gambar 2.3. 4



Gambar 2.3. 5

Berdasarkan hasil keseluruhan pengujian API pada Acad-Service, setiap endpoint berhasil diakses dan mengembalikan status kode HTTP 200. Selain itu, struktur respons yang dihasilkan telah sesuai dengan spesifikasi yang ditetapkan di dalam kode aplikasi.

BAB 3
KESIMPULAN

3.1 Kesimpulan

Berdasarkan pelaksanaan tugas ini, dapat disimpulkan bahwa penerapan arsitektur microservices dengan memanfaatkan Docker dan Docker Compose memungkinkan pengembangan sistem backend yang terstruktur, terisolasi, dan terintegrasi dengan baik. Setiap layanan dapat dijalankan secara mandiri dalam container yang berbeda, namun tetap dapat saling berkomunikasi melalui jaringan internal. Selain itu, penggunaan API Gateway, service backend, serta database yang terpisah mendukung pengelolaan sistem yang lebih efisien, stabil, dan mudah dikembangkan. Implementasi API untuk pengolahan data akademik, seperti perhitungan IPS, menunjukkan penerapan logika bisnis yang sesuai dengan kebutuhan sistem serta mendukung praktik pengembangan perangkat lunak yang baik.