LAPORAN PENGERJAAN UJIAN AKHIR SEMESTER (UAS) ARSITEKTUKTUR KOMPUTER DAN SISTEM OPERASI TOPIK: MICROSERVICE SEDERHANA DENGAN DOCKER

Semester Gasal Tahun Akademik 2025/2026

By :

1. **Nabil Aryo Suharnoto (25031554092)**
2. **Rafli Akbar Maulana (25031554256)**
3. **Farid Aufa Rafiqi (25031554088)**
4. **Ibnu A. Zahran Latua (25031554053**)

Lecturer:

Hasanuddin Al-Habib, S.Si., M.Si Sabrina Amelialevi, S.Kom., M.Kom.

STUDY PROGRAM DATA SCIENCE

FACULTY OF MATHEMATICS AND NATURAL SCIENCES STATE UNIVERSITY OF SURABAYA

2025

LAPORAN PROYEK

Deployment Sistem Layanan Akademik Berbasis Microservice sederhana dengan Docker

Latar Belakang

Perkembangan teknologi informasi yang semakin pesat menuntut sistem perangkat lunak untuk bersifat fleksibel, mudah dikembangkan, serta mampu menangani peningkatan jumlah pengguna secara efisien. Arsitektur monolitik yang menggabungkan seluruh fungsi aplikasi dalam satu sistem sering kali menimbulkan kendala, seperti sulitnya proses pemeliharaan, pengembangan fitur baru, serta tingginya risiko kegagalan sistem secara keseluruhan. Oleh karena itu, diperlukan pendekatan arsitektur yang lebih modular dan scalable.

Microservice merupakan salah satu pendekatan arsitektur perangkat lunak yang memecah sebuah sistem besar menjadi beberapa sub-layanan yang berdiri sendiri dan saling berkomunikasi melalui Application Programming Interface (API). Setiap sub-layanan dikembangkan, dijalankan, dan dikelola secara independen sehingga memudahkan proses pengembangan, pengujian, serta deployment. Pendekatan ini banyak diterapkan pada sistem berskala besar atau *super apps* karena mampu meningkatkan kecepatan pengembangan dan isolasi permasalahan antar layanan.

Salah satu teknologi pendukung utama dalam penerapan arsitektur microservice adalah virtualisasi berbasis container menggunakan Docker. Dengan Docker, setiap sub-layanan dapat dikemas dalam sebuah container yang ringan, konsisten, dan mudah dijalankan di berbagai lingkungan. Selain itu, penggunaan Docker Compose memungkinkan pengelolaan beberapa container secara terpusat, termasuk pengaturan jaringan internal, dependensi layanan, serta volume penyimpanan data.

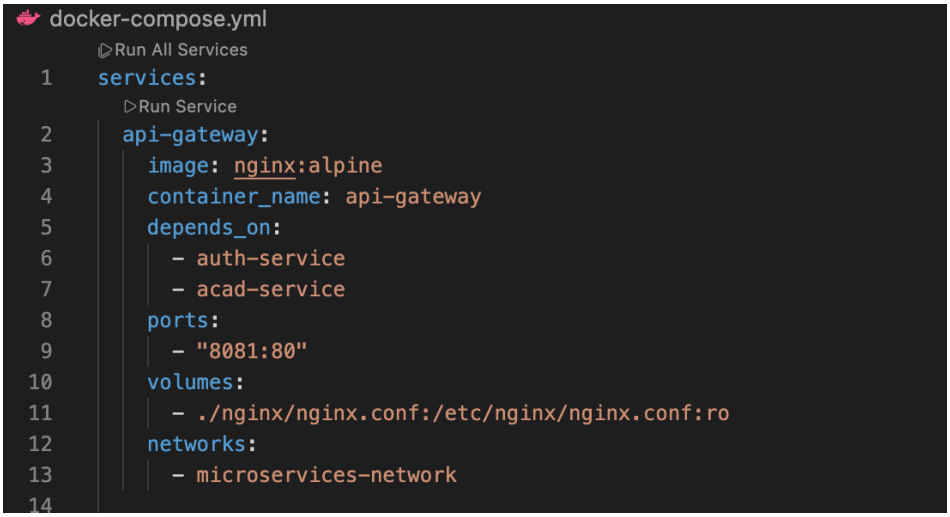
Dalam proyek ini, dilakukan penerapan konsep microservice pada layanan akademik yang terdiri dari beberapa sub-layanan, yaitu layanan otentikasi (auth-service) dan layanan akademik/penilaian (acad-service), yang diakses melalui sebuah API Gateway berbasis Nginx. Sistem ini merupakan *legacy software* sehingga diperlukan konfigurasi ulang serta penambahan fitur tertentu agar layanan dapat berjalan secara terintegrasi dan optimal. Melalui proyek ini diharapkan dapat memberikan pemahaman praktis mengenai penerapan microservice, containerization, serta peran DevOps dalam melakukan deployment dan konfigurasi layanan berbasis Docker.

A. Konfigurasi Docker Compose

Docker Compose digunakan sebagai alat orkestrasi container untuk menjalankan seluruh sub-layanan dalam arsitektur microservice secara terintegrasi. Melalui berkas docker-compose.yml, setiap layanan didefinisikan beserta dependensi, jaringan internal, dan volume penyimpanan yang digunakan. Dengan konfigurasi ini, seluruh sistem dapat dijalankan menggunakan satu perintah, yaitu docker-compose up --build -d.

1. API Gateway (`api-gateway`)

- **Image**: `nginx:alpine`
  Digunakan sebagai *reverse proxy* untuk meneruskan permintaan dari klien ke sub-layanan terkait.
- **Container Name**: `api-gateway`
  Memberikan identitas container yang permanen agar mudah dikelola.
- **Depends On**:
  - `auth-service`
  - `acad-service`
    API Gateway hanya berjalan setelah layanan autentikasi dan akademik aktif.
- **Port Mapping**: `8081:80`
  Port 8081 pada host digunakan sebagai pintu masuk sistem dan diteruskan ke port 80 di dalam container.
- **Volume**:
  - `./nginx/nginx.conf:/etc/nginx/nginx.conf:ro`
    File konfigurasi Nginx dimapping secara *read-only* untuk menjaga konsistensi dan keamanan konfigurasi.
- **Network**: `microservices-network`
  Menghubungkan API Gateway dengan seluruh sub-layanan secara internal.



Gambar A.1

Sub-layanan API Gateway berperan sebagai gerbang utama yang menerima seluruh permintaan (*request*) dari klien dan meneruskannya ke sub-layanan terkait. API Gateway pada proyek ini menggunakan **Nginx versi Alpine** karena memiliki ukuran image yang ringan dan performa yang baik untuk keperluan *reverse proxy*.

Container API Gateway diberi nama permanen `api-gateway` untuk memudahkan pengelolaan. Layanan ini memiliki ketergantungan pada layanan `auth-service` dan `acad-service`, sehingga hanya dapat dijalankan setelah kedua layanan tersebut aktif. Hal ini bertujuan untuk memastikan bahwa setiap permintaan yang diterima dapat langsung diteruskan tanpa kendala dependensi.

API Gateway mengekspose port **8081** pada sisi host yang dipetakan ke port **80** di dalam container. Konfigurasi Nginx dimapping dari direktori proyek ke dalam container dengan mode *read-only* agar konfigurasi dapat dikelola secara terpusat dan aman. Seluruh komunikasi API Gateway dengan layanan lainnya dilakukan melalui jaringan internal yang telah didefinisikan.

2. Auth Service (`auth-service`)

- **Build Context**: `./auth-service`
  Container dibangun dari Dockerfile yang terdapat pada direktori Auth Service.
- **Container Name**: `auth-service`
- **Resource Limitation**:
  - CPU: 0.5 core
  - Memory: 512 MB
    Bertujuan untuk membatasi penggunaan sumber daya sesuai spesifikasi tugas.
- **Depends On**:
  - `auth-db`
    Auth Service hanya berjalan jika database MongoDB aktif.
- **Environment Variables**:
  - `NODE_ENV=development` → Mode pengembangan
  - `PORT=3001` → Port layanan Auth
  - `MONGO_URI=mongodb://auth-db:27017/auth` → Koneksi database
  - `JWT_SECRET` → Kunci keamanan untuk autentikasi berbasis JWT
- **Port Mapping**: `3001:3001`
  Digunakan untuk komunikasi API dan pengujian.
- **Volume**:
  - `./auth-service:/app` → Sinkronisasi source code
  - `/app/node_modules` → Isolasi dependensi Node.js
- **Network**: `microservices-network`

```
▷Run Service
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=Vzl2rkSw4GNdn+3gH6Q5BleIkTs9FOa5wI6qiPFp3lHCSm8iX0nc9uEkZ0+yExnC
  ports:
    - "3001:3001"
  networks:
    - microservices-network
  volumes:
    - ./auth-service:/app
    - /app/node_modules

```

Gambar A.2

Sub-layanan otentikasi bertugas menangani proses autentikasi pengguna, seperti login dan pengelolaan token keamanan. Auth Service dibangun dari Dockerfile yang terdapat pada direktori `auth-service` dan diberi nama container `auth-service`.

Layanan ini memiliki ketergantungan langsung terhadap database MongoDB (`auth-db`), sehingga tidak akan dijalankan sebelum database aktif. Untuk meningkatkan efisiensi penggunaan sumber daya, Auth Service dibatasi penggunaan CPU sebesar **0.5 virtual core** dan memori sebesar **512 MB**, sesuai dengan ketentuan tugas.

Konfigurasi lingkungan (*environment variables*) digunakan untuk menentukan mode aplikasi, port layanan, koneksi ke database MongoDB, serta *JWT Secret* yang berfungsi sebagai kunci keamanan dalam proses autentikasi berbasis token. Auth Service berjalan pada port **3001** dan terhubung ke jaringan internal agar dapat diakses oleh API Gateway.

Volume digunakan untuk melakukan sinkronisasi kode sumber antara host dan container, serta menjaga dependensi aplikasi tetap terisolasi di dalam container.

3. Auth Database (`auth-db`)

- **Image**: `mongo:6.0`
  Digunakan sebagai basis data untuk layanan autentikasi.
- **Container Name**: `auth-db`
- **Restart Policy**: `always`
  Container otomatis berjalan kembali jika terjadi kegagalan.
- **Volume**:
  - `./auth-data:/data/db`
    Menyimpan data MongoDB secara persisten.
- **Port Mapping**: `27017:27017`
  Memudahkan akses database untuk keperluan debugging.
- **Network**: `microservices-network`

```
▷Run Service
37  auth-db:
38    image: mongo:6.0
39    container_name: auth-db
40    restart: always
41    networks:
42      - microservices-network
43    volumes:
44      - ./auth-data:/data/db
45    ports:
46      - "27017:27017"
47
```

Gambar A.3

Basis data otentikasi menggunakan **MongoDB versi 6.0** sebagai media penyimpanan data akun pengguna. Container database diberi nama permanen `auth-db` dan dikonfigurasikan untuk selalu melakukan *restart* secara otomatis apabila terjadi kegagalan.

Data MongoDB disimpan secara persisten dengan melakukan pemetaan direktori lokal `auth-data` ke direktori `/data/db` di dalam container. Dengan konfigurasi ini, data tidak akan hilang meskipun container dihentikan atau dijalankan ulang. Auth Database terhubung pada jaringan internal yang sama dengan Auth Service.

4. Academic Service (`acad-service`)

- **Build Context**: `./acad-service`
  Container dibangun dari source code layanan akademik.
- **Container Name**: `acad-service`
- **Depends On**:
  - `acad-db`
    Menjamin database PostgreSQL tersedia sebelum layanan dijalankan.
- **Environment Variables**:
  - `PORT=3002` → Port layanan akademik
  - `DB_HOST=acad-db` → Host database
  - `DB_PORT=5432` → Port PostgreSQL
  - `DB_NAME`, `DB_USER`, `DB_PASSWORD` → Kredensial database
- **Port Mapping**: `3002:3002`
  Digunakan untuk komunikasi API akademik.
- **Volume**:
  - `./acad-service:/app`
    Memungkinkan pengembangan dan perubahan kode tanpa rebuild container.
- **Network**: `microservices-network`

```
▷Run Service
48  acad-service:
49    build: ./acad-service
50    container_name: acad-service
51    depends_on:
52      - acad-db
53    environment:
54      - NODE_ENV=development
55      - PORT=3002
56      - DB_HOST=acad-db
57      - DB_PORT=5432
58      - DB_NAME=acad-database
59      - DB_USER=zhrnltuaaxhh
60      - DB_PASSWORD=duniagamesss
61    ports:
62      - "3002:3002"
63    networks:
64      - microservices-network
65    volumes:
66      - ./acad-service:/app
67
```

Gambar A. 4

Sub-layanan akademik bertanggung jawab dalam pengelolaan data akademik mahasiswa, termasuk perhitungan Indeks Prestasi Semester (IPS). Layanan ini dibangun dari source code pada direktori `acad-service` dan dijalankan dalam container bernama `acad-service`.

Acad Service memiliki ketergantungan terhadap database PostgreSQL (`acad-db`). Konfigurasi koneksi database diatur melalui *environment variables* yang mencakup alamat host database, port, nama database, serta kredensial akses. Layanan ini berjalan pada port **3002** dan terhubung ke jaringan internal untuk berkomunikasi dengan API Gateway dan database.

5. Academic Database (`acad-db`)

- **Image**: `postgres:16`
  Digunakan sebagai basis data akademik.

- **Container Name**: `acad-db`
- **Environment Variables**:
  - `POSTGRES_USER`
  - `POSTGRES_PASSWORD`
  - `POSTGRES_DB`
    Digunakan untuk inisialisasi database PostgreSQL.
- **Volume**:
  - `./product-data:/var/lib/postgresql/data` → Penyimpanan data persisten
  - `db_kelas.sql:/docker-entrypoint-initdb.d/` → Inisialisasi tabel otomatis
- **Port Mapping**: `5432:5432`
- **Network**: `microservices-network`

```
▷Run Service
68  acad-db:
69    image: postgres:16
70    container_name: acad-db
71    environment:
72      - POSTGRES_USER=zhrnltuaaxhh
73      - POSTGRES_PASSWORD=duniagamesss
74      - POSTGRES_DB=acad-database
75    ports:
76      - "5432:5432"
77    networks:
78      - microservices-network
79    volumes:
80      - ./product-data:/var/lib/postgresql/data
81      - ./acad-service/db_kelas.sql:/docker-entrypoint-initdb.d/db_kelas.sql:ro
82
```

Gambar A.5

Basis data akademik menggunakan **PostgreSQL versi 16**. Container database diberi nama `acad-db` dan dikonfigurasikan dengan nama pengguna, kata sandi, serta nama database yang telah ditentukan.

Untuk mendukung persistensi data, direktori penyimpanan PostgreSQL di dalam container dipetakan ke direktori lokal `product-data`. Selain itu, sebuah berkas SQL (`db_kelas.sql`) dimapping ke direktori inisialisasi PostgreSQL agar struktur tabel dapat dibuat secara otomatis saat container pertama kali dijalankan.

6. Jaringan Internal (`microservices-network`) dan volume
Jaringan Internal:

- **Driver**: `bridge`
- **Subnet**: `172.16.0.0/28`
  Digunakan untuk mengisolasi komunikasi antar container serta meningkatkan keamanan sistem.

Volume:

- **auth-data** → Penyimpanan data MongoDB
- **product-data** → Penyimpanan data PostgreSQL

- **order-data** → Disiapkan untuk pengembangan lanjutan

Volume digunakan untuk menjaga data tetap tersedia meskipun container dihentikan atau di-*rebuild*.

```
82
83  networks:
84    microservices-network:
85      driver: bridge
86      ipam:
87        config:
88          - subnet: 172.16.0.0/28
89
90  volumes:
91    auth-data:
92    product-data:
93    order-data:
```

Gambar A.6

Seluruh sub-layanan terhubung pada satu jaringan internal bernama `microservices-network` dengan driver **bridge**. Jaringan ini dikonfigurasikan menggunakan subnet **172.16.0.0/28** untuk memastikan isolasi komunikasi antar container serta meningkatkan keamanan sistem. Dengan jaringan internal ini, setiap layanan dapat saling berkomunikasi menggunakan nama container sebagai hostname.

Volume didefinisikan untuk mendukung penyimpanan data persisten dan keberlangsungan layanan. Volume ini digunakan oleh database MongoDB dan PostgreSQL agar data tetap tersedia meskipun container dihentikan atau di-*rebuild*. Pendekatan ini mendukung prinsip skalabilitas dan keandalan sistem microservice.

B. Implementasi API Perhitungan Indeks Prestasi Semester (IPS)

Potongan kode berikut merupakan bantuk Challenge yang harus kita tambahkan dan implementasi pada endpoint API pada sub-layanan akademik (`acad-service`) yang berfungsi untuk menghitung Indeks Prestasi Semester (IPS) mahasiswa berdasarkan nilai mata kuliah dan jumlah SKS.

1. Definisi Endpoint API

- **Method**: `GET`
- **Path**: `/api/acad/ips`

Endpoint ini digunakan untuk mengambil data akademik mahasiswa sekaligus menghitung nilai IPS secara dinamis dari basis data.

```
@app.get("/api/acad/ips")
async def get_IPS():
```

Penggunaan metode **GET** dipilih karena endpoint ini hanya melakukan pengambilan data tanpa mengubah isi database.

2. Koneksi ke Database

- Koneksi ke database PostgreSQL dilakukan menggunakan fungsi `get_db_connection()`.
- Cursor menggunakan `RealDictCursor` agar hasil query berbentuk dictionary, sehingga lebih mudah diakses berdasarkan nama kolom.

```
with get_db_connection() as data:
    cursor = data.cursor(cursor_factory=RealDictCursor)
```

Pendekatan ini memastikan koneksi database ditutup secara otomatis setelah proses selesai.

3. Query Pengambilan Data Akademik

- Query SQL digunakan untuk mengambil data mahasiswa, nilai mata kuliah, dan jumlah SKS.
- Tabel yang digunakan:
    - `mahasiswa`
    - `krs`
    - `mata_kuliah`

```
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'
```

Query ini menggabungkan beberapa tabel untuk mendapatkan data lengkap yang dibutuhkan dalam perhitungan IPS.

4. Validasi Data Mahasiswa

- Jika hasil query kosong, sistem akan mengembalikan error **404 (Not Found)**.
- Hal ini menandakan bahwa data mahasiswa tidak ditemukan dalam database.

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

Validasi ini penting untuk mencegah perhitungan IPS pada data yang tidak valid.

5. Konversi Nilai Huruf ke Angka

- Nilai huruf dikonversi ke bobot angka menggunakan dictionary `nilai_map`.

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

```
}
```

Konversi ini mengikuti standar penilaian akademik yang umum digunakan di perguruan tinggi.

6. Proses Perhitungan IPS

- Variabel `value_total` digunakan untuk menyimpan total nilai berbobot.
- Variabel `sks_total` digunakan untuk menyimpan total SKS.

```
value_total += value_in_angka * sks
sks_total += sks
```

Setiap nilai mata kuliah dikalikan dengan jumlah SKS, kemudian dijumlahkan.

7. Perhitungan Akhir IPS.

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

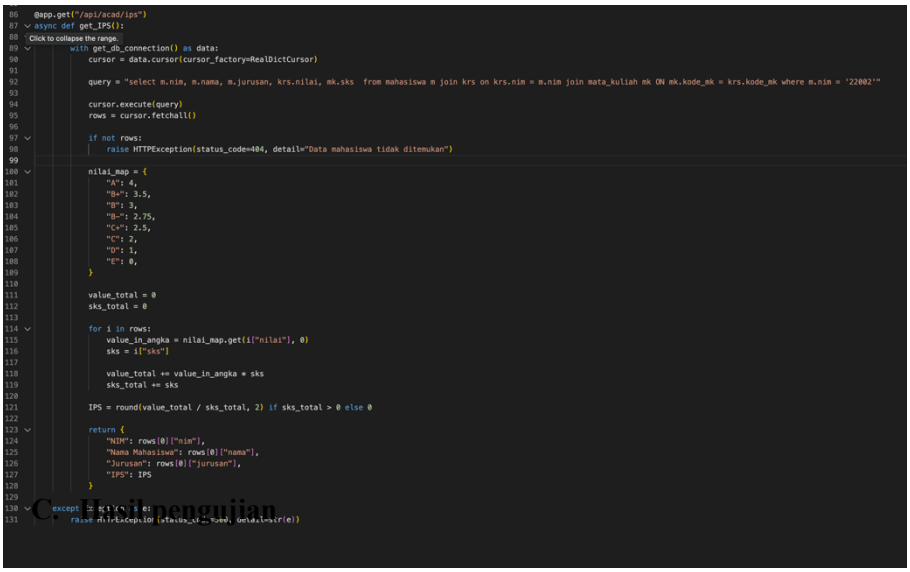
Validasi dilakukan untuk menghindari pembagian dengan nol.

8. Response API

- API mengembalikan data dalam format JSON yang berisi:
  - NIM mahasiswa
  - Nama mahasiswa
  - Jurusan
  - Nilai IPS

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

Response ini memudahkan integrasi dengan API Gateway maupun aplikasi frontend.

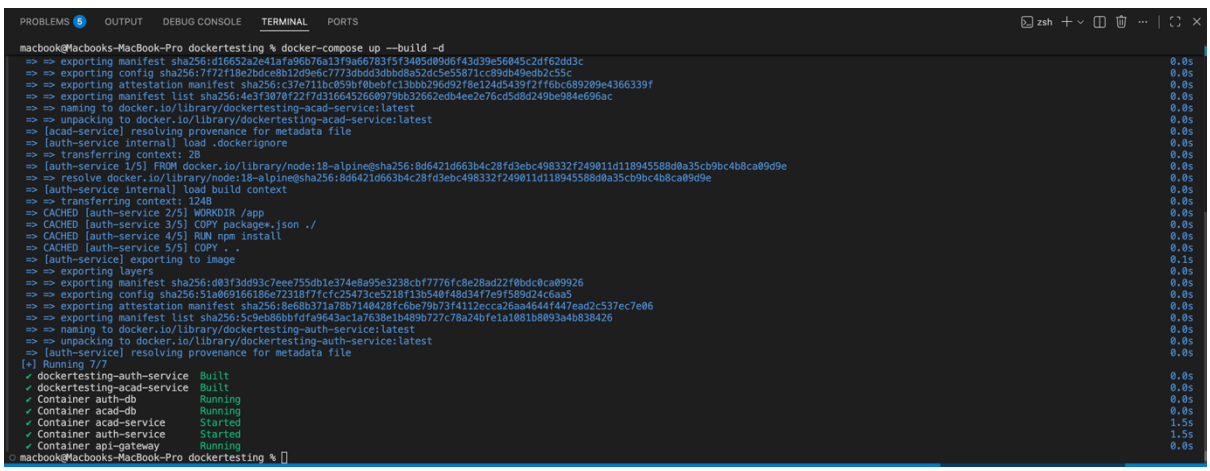


C. Hasil pengujian

Berdasarkan implementasi dan pengujian yang telah dilakukan pada proyek layanan akademik berbasis microservice, diperoleh beberapa hasil sebagai berikut:

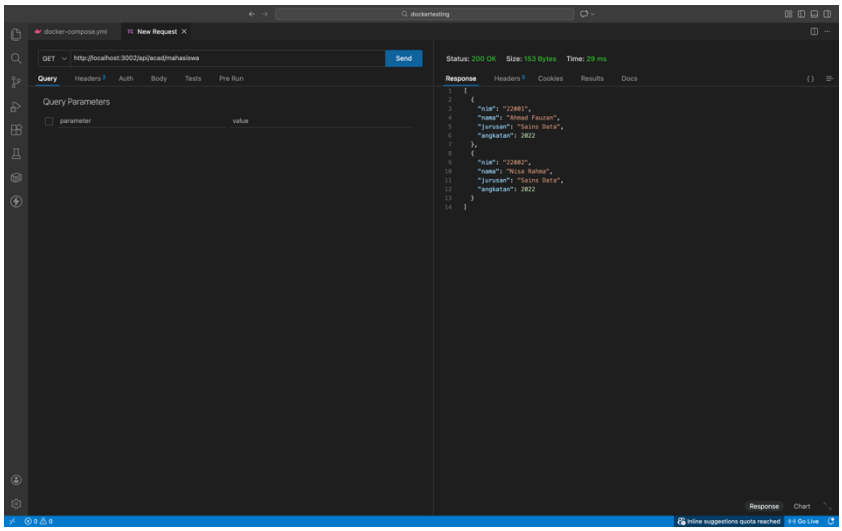
1. **Implementasi Arsitektur Microservice**
   - Sistem berhasil dibangun menggunakan pendekatan arsitektur microservice yang terdiri dari beberapa sub-layanan, yaitu API Gateway, Auth Service, dan Academic Service.
   - Setiap sub-layanan berjalan pada container terpisah sehingga memiliki isolasi proses dan dependensi yang baik.
2. **Keberhasilan Deployment Menggunakan Docker Compose**
   - Seluruh layanan berhasil dijalankan secara terintegrasi menggunakan perintah `docker-compose up --build -d`.
   - Docker Compose mempermudah pengelolaan dependensi antar layanan, jaringan internal, serta volume penyimpanan data.
3. **API Gateway Berfungsi dengan Baik**
   - API Gateway berbasis Nginx berhasil berperan sebagai gerbang utama sistem.
   - Permintaan dari klien dapat diteruskan ke Auth Service dan Academic Service melalui satu titik akses dengan port 8081.
4. **Layanan Otentikasi Berjalan Stabil**
   - Auth Service berhasil terhubung dengan database MongoDB.
   - Proses autentikasi menggunakan JWT dapat dikonfigurasi dengan baik melalui environment variable.
   - Pembatasan sumber daya CPU dan memori berjalan sesuai konfigurasi.
5. **Perhitungan IPS Mahasiswa Berhasil Diimplementasikan**
   - Endpoint API `/api/acad/ips` berhasil menghitung Indeks Prestasi Semester (IPS) mahasiswa berdasarkan nilai dan jumlah SKS.
   - Data akademik diambil langsung dari database PostgreSQL dan diproses secara dinamis.
   - Hasil perhitungan ditampilkan dalam format JSON yang mudah diintegrasikan dengan layanan lain.
6. **Penggunaan Jaringan Internal dan Volume**
   - Seluruh container berhasil terhubung melalui jaringan internal dengan subnet yang telah ditentukan.
   - Data database tersimpan secara persisten menggunakan volume, sehingga tidak hilang saat container dihentikan atau dijalankan ulang.

Output:

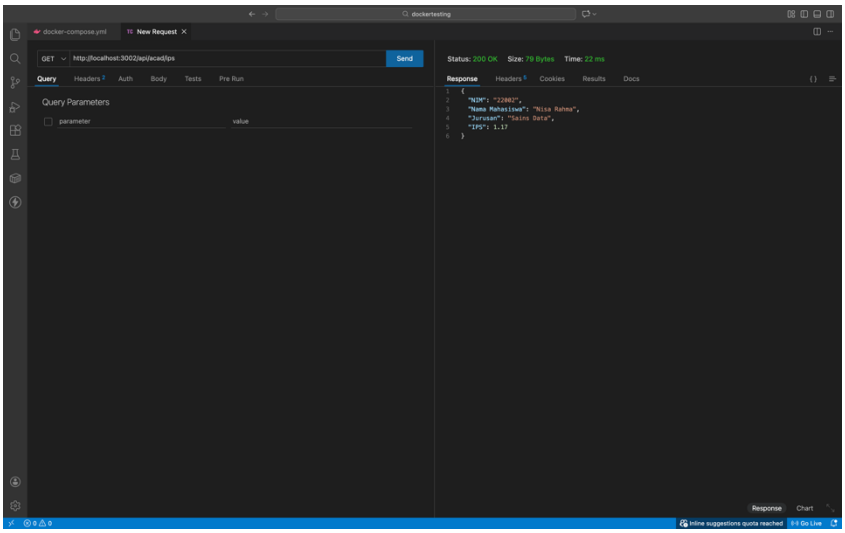


semua container berjalan

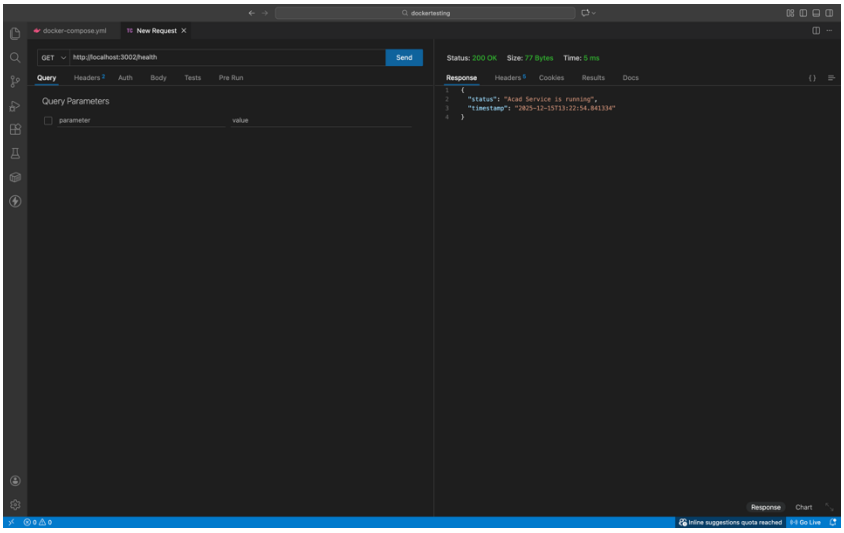
http://localhost:3002/api/acad/mahasiswa



http://localhost:3002/api/acad/ips



http://localhost:3002/health



KESIMPULAN

Berdasarkan hasil yang diperoleh, dapat disimpulkan bahwa proyek implementasi layanan akademik berbasis microservice menggunakan Docker dan Docker Compose telah berhasil dilakukan sesuai dengan spesifikasi yang ditetapkan. Arsitektur microservice memungkinkan sistem menjadi lebih modular, mudah dikembangkan, serta memiliki tingkat keandalan yang lebih baik dibandingkan arsitektur monolitik.

Penggunaan containerisasi dengan Docker memberikan kemudahan dalam proses deployment, konfigurasi, dan pengelolaan layanan. Selain itu, pemisahan tanggung jawab antar sub-layanan, seperti otentikasi dan akademik, membuat sistem lebih terstruktur dan scalable. Implementasi API perhitungan IPS menunjukkan bahwa logika perhitungan dapat diisolasi dengan baik dalam satu layanan khusus tanpa memengaruhi layanan lainnya.

Dengan demikian, kami harap projek ini dapat memberikan kita pemahaman praktis mengenai penerapan konsep DevOps, microservice, serta pengelolaan layanan berbasis container. Sistem yang dibangun masih dapat dikembangkan lebih lanjut, seperti penambahan autentikasi berbasis role, parameter NIM dinamis, serta integrasi dengan antarmuka pengguna (frontend).

.