

**Implementasi Arsitektur Microservice Akademik Menggunakan
Docker Compose pada Layanan Pengelolaan Akun Sederhana
dan Perhitungan IPS Mahasiswa**



Disusun Oleh:

Nasywa Callula Azzah Darmawan : 25031554235

Kane Matthew Prastersen : 25031554141

Dicky Alviano : 25031554227

**PROGRAM STUDI SAINS DATA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN
ALAM
UNIVERSITAS NEGERI SURABAYA
NOVEMBER, 2025**

DAFTAR ISI

BAB I	4
1.1 Latar Belakang	4
1.2 Rumusan Masalah	4
1.3 Tujuan	5
1.4 Manfaat	5
Bab II.....	6
2.1 Landasan Teori.....	6
2.1.1 Microsevice Architecture	6
2.1.2 Containerization dan Docker.....	6
2.1.3 Docker composer	6
2.1.4 API Gateway	7
2.1.5 Layanan Otentikasi (Authentication Service)	7
2.1.6 Layanan akademik dan perhitungan IPS.....	7
2.2 Tahapan pekerjaan	7
2.2.1 Perancangan Docker – Compose.Yml	7
2.2.2 Siapkan Konfigurasi API Gateway (Nginx)	8
2.2.3 Definisi Layanan auth-db (MongoDB)	8
2.2.4 Definisi layanan auth-service (Node.js).....	8
2.2.5 Definisi layanan acad-service (Node.js)	8
2.2.6 Jalankan Orkestrasi	9
2.2.7 Implementasi Main Menu (FastAPI - main.py)	9
2.2.8 Pengujian (testing)	9
2.3 Tahapan Pengujian Di Insomnia	9
2.3.1 Cek Konfigurasi API Gateway.....	9
2.3.2 Implementasi Main Menu (FastAPI - main.py)	10
2.3.3 Cek Register (Auth Service) - Pilih metode POST.....	10
2.3.4 Cek Data Mahasiswa (Acad Service) - pilih metode GET	10
2.3.5 Cek Perhitungan IPS (Acad Service) - Pilih metode GET.....	10
2.4 Penerapan Pengaturan Lanjutan.....	11
BAB III	11
3.1 Implementasi.....	12

3.1.1 Hasil Implementasi Docker-compose Up --Build -D	12
3.1.2 Pastikan implementasi container Docker(API Gateway, auth-service, auth-db, acad-service, acad-db).....	12
3.1.3 Hasil konfigurasi API Gateway	13
3.1.4 Hasil endpoint /health yang menunjukkan service aktif.	13
3.1.5 Hasil endpoint /register	13
3.1.6 hasil endpoint /register (User already exists)	13
3.1.7 Hasil Endpoint /Login	14
3.1.8 Hasil query mahasiswa (/api/acad/mahasiswa) yang menampilkan data dari.....	14
database.....	14
3.1.9 Hasil perhitungan IPS (/api/acad/ips?nim=22001) sesuai dengan data di db_kelas.sql.....	14
3.1.10 Hasil perhitungan IPS (/api/acad/ips?nim=22002) sesuai dengan data di db_kelas.sql.....	14
BAB IV	15
4.1 Kesimpulan	15
4.2 Saran.....	16
DAFTAR PUSTAKA	16

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang sangat pesat menuntut adanya sistem yang efisien, fleksibel, dan mudah dikelola dalam proses pengembangan serta penyebaran aplikasi. Metode tradisional dalam menjalankan aplikasi sering kali menghadapi permasalahan perbedaan lingkungan antara pengembang dan server produksi, yang dikenal dengan istilah “*it works on my machine*”. Masalah ini dapat menyebabkan kegagalan aplikasi ketika dipindahkan ke lingkungan lain. Docker merupakan platform open-source yang menjalankan aplikasi, menyederhanakan proses pengembangan dan distribusinya. Aplikasi yang dibangun diatas Docker dikemas bersama semua dependensi penting ke dalam suatu format standar yang dikenal sebagai container. Container ini beroperasi secara terpisah di atas kernel dari sistem operasi yang ada. Tambahan lapisan abstraksi ini mungkin mempengaruhi kinerja. Meskipun teknologi container telah ada lebih dari satu dekade, Docker, yang pada dasarnya adalah pendatang baru dengan potensi besar, saat ini telah menjadi salah satu inovasi paling menarik, karena menawarkan kemampuan yang tidak dimiliki teknologi sebelumnya. Pada awalnya, Docker memungkinkan penciptaan dan pengelolaan container dengan mudah. Selain itu, para pengembang dapat dengan cepat mengemas aplikasi ke dalam container Docker yang efisien. Aplikasi virtualisasi tersebut dapat dijalankan di berbagai tempat tanpa perlu modifikasi apapun. Selain itu, Docker mampu memberikan lebih banyak lingkungan virtual dibandingkan teknologi lainnya pada perangkat keras yang sama.

1.2 Rumusan Masalah

1. Apa langkah-langkah untuk mengembangkan sistem autentikasi mahasiswa serta layanan perhitungan IPS dengan pendekatan microservice?
2. Bagaimana cara mengatur konfigurasi Docker Compose agar seluruh layanan (otentikasi, skor, basis data, API Gateway) dapat beroperasi secara harmonis?

1.3 Tujuan

1. Mengembangkan sistem autentikasi untuk mahasiswa serta layanan kalkulasi IPS dengan pendekatan microservice.
2. Mengatur konfigurasi Docker Compose agar seluruh layanan (otentikasi, skor, basis data, API Gateway) dapat beroperasi secara terkoordinasi.

1.4 Manfaat

1. Memberikan pengalaman langsung dalam menciptakan sistem akademik sederhana yang berbasis microservice menggunakan Docker Compose.
2. Membantu mahasiswa memahami cara mengintegrasikan sejumlah layanan kecil agar dapat berkolaborasi dalam satu sistem yang bersifat modular dan adaptif.

BAB II

PEMBAHASAN

2.1 Landasan Teori

2.1.1 Microsevice Architecture

Microsevice Architecture kini telah menjadi hal yang umum. Evolusi dari sistem lama ini, maka sistem tersebut rentan terhadap penurunan kinerja. Salah satu cara yang paling penting untuk mendeteksi penurunan kinerja adalah dengan mampu membangun kembali arsitektur perangkat lunak suatu sistem berdasarkan pada sistem yang saat ini beroperasi di lingkungan produksi. Berbagai pendekatan telah diajukan sebelumnya pendekatan yang menggunakan analisis statis dari kode sumber mikrolayanan serta pendekatan yang mengandalkan analisis log yang diambil saat sistem sedang berjalan.

2.1.2 Containerization dan Docker

Kontainerisasi telah membawa komputasi awan ke level yang lebih tinggi di mana para pengembang aplikasi bisa menciptakan metode inovatif untuk mendistribusikan aplikasi web yang dapat dipindahkan. Dokumen ini mengulas perbandingan antara Docker, salah satu teknologi kontainerisasi yang paling terkenal, dan Kubernetes berkaitan dengan kecakapan mereka dalam mengelola sumber daya cloud, integrasi canggih, serta penerapan layanan mikro. Docker, yang memproduksi kontainer yang ringan, meminimalkan kompleksitas dalam proses pengiriman aplikasi dengan menggabungkan semua elemen perangkat lunak beserta konfigurasi yang diperlukan. Sementara itu, Kubernetes berfungsi sebagai alat untuk otomatisasi dalam hal penyebaran, penskalaan, dan pengelolaan aplikasi yang terkontainerisasi di dalam kluster.

2.1.3 Docker composer

Merupakan komponen yang terdapat di dalam ekosistem docker yang berfungsi sebagai alat orkestrasi ringan guna mengelola aplikasi berbasis multi-

container. Melalui berkas konfigurasi berformat YAML. Docker compose ini memungkinkan terdapat di dalam ekosistem docker yang berfungsi sebagai alat orkestrasi ringan guna mengelola aplikasi berbasis multi-container .

2.1.4 API Gateaway

API Gateway merupakan komponen perantara (middleware) dalam arsitektur sistem terdistribusi yang berfungsi sebagai titik masuk tunggal (single entry point) bagi klien untuk mengakses berbagai layanan (microservices). API Gateway bertanggung jawab dalam melakukan pengelolaan permintaan, seperti routing, authentication, authorization, rate limiting, serta agregasi respons, sehingga kompleksitas komunikasi antar layanan dapat disembunyikan dari klien dan kinerja serta keamanan sistem dapat ditingkatkan.

2.1.5 Layanan Otentikasi (Authentication Service)

Layanan Otentikasi (Authentication Service) merupakan komponen sistem yang berfungsi untuk memverifikasi identitas pengguna atau entitas sebelum diberikan akses ke sumber daya atau layanan tertentu. Layanan ini melakukan proses validasi kredensial, seperti nama pengguna dan kata sandi atau token keamanan, serta mengelola mekanisme autentikasi guna memastikan bahwa hanya pengguna yang sah yang dapat mengakses sistem, sehingga mendukung keamanan dan integritas pada arsitektur aplikasi terdistribusi.

2.1.6 Layanan akademik dan perhitungan IPS

Layanan Akademik dan Perhitungan IPS merupakan komponen sistem yang berfungsi untuk mengelola data akademik mahasiswa, seperti mata kuliah, nilai, dan bobot kredit, serta melakukan perhitungan Indeks Prestasi Semester (IPS) berdasarkan ketentuan akademik yang berlaku. Layanan ini mengolah data nilai dan SKS secara terstruktur untuk menghasilkan informasi IPS yang akurat, sehingga mendukung proses evaluasi akademik dan pengambilan keputusan dalam sistem informasi akademik.

2.2 Tahapan pekerjaan

2.2.1 Perancangan Docker – Compose.Yml

Perancangan berkas *docker-compose.yml* dilakukan untuk mengorkestrasikan layanan aplikasi berbasis *microservices*, seperti API Gateway, auth-service (Node.js), auth-db (MongoDB), acad-service (Node.js), acad-db (PostgreSQL) dalam satu lingkup yang terintegrasi. Berkas ini mendefinisikan konfigurasi setiap

layanan secara deklaratif, mencakup *image*, pemetaan port, jaringan, dan dependensi antar-container, sehingga mendukung konsistensi lingkungan, kemudahan pengelolaan layanan, serta efisiensi proses pengembangan dan pengujian sesuai prinsip *Infrastructure as Code*.

2.2.2 Siapkan Konfigurasi API Gateway (Nginx)

Konfigurasi **API Gateway menggunakan Nginx** disiapkan untuk berperan sebagai titik masuk tunggal (*single entry point*) yang mengatur distribusi permintaan dari klien ke berbagai layanan backend. Nginx dikonfigurasikan untuk melakukan *request routing* berdasarkan endpoint tertentu menuju Layanan Otentikasi maupun Layanan Akademik dan Perhitungan IPS, serta mendukung pengelolaan koneksi, peningkatan kinerja, dan pemisahan tanggung jawab antar layanan.

2.2.3 Definisi Layanan auth-db (MongoDB)

Selanjutnya, layanan database otentikasi dikonfigurasikan menggunakan *image* MongoDB versi 6.0. Container diberi nama **auth-db**, dihubungkan ke jaringan **oska_net**, serta dilakukan pemetaan direktori lokal **auth-data** ke direktori **/data/db** pada container. Konfigurasi ini bertujuan untuk menjaga keberlanjutan penyimpanan data akun sehingga tetap tersedia meskipun container dihentikan, dihapus, atau dijalankan kembali.

2.2.4 Definisi layanan auth-service (Node.js)

Setelah database terinisialisasi, layanan otentikasi didefinisikan dalam container bernama auth-service. Layanan ini dilengkapi dengan *environment variables* seperti:

- NODE_ENV=development
- PORT=3001
- MONGO_URI=mongodb://auth-db:27017/auth
- JWT_SECRET (digenerate dari situs jwtsecrets.com)

Dengan memanfaatkan auth-db sebagai database, layanan ini mampu menyajikan endpoint registrasi dan login secara aman.

2.2.5 Definisi layanan acad-service (Node.js)

Berikutnya, dikonfigurasikan layanan perhitungan IPS mahasiswa dengan nama score-service. Container ini terhubung ke database akademik acad-db (PostgreSQL) dan menggunakan *environment variables* seperti:

- DB_HOST=acad-db
- DB_USER=postgres
- DB_PASSWORD
- DB_NAME=akademik

Melalui konfigurasi tersebut, sistem mampu mengelola nilai mahasiswa dan menghasilkan perhitungan IPS.

2.2.6 Jalankan Orkestrasi

Usai seluruh konfigurasi diterapkan, eksekusi container dilakukan melalui perintah docker-compose up --build -d untuk memastikan integrasi layanan berjalan dengan baik tanpa kesalahan.

2.2.7 Implementasi Main Menu (FastAPI - main.py)

Pada tahap ini, berkas *main.py* diimplementasikan sebagai *entry point* layanan akademik berbasis FastAPI, yang meliputi pengaturan CORS untuk akses eksternal, koneksi ke PostgreSQL melalui *psycopg2*, serta pemodelan entitas Mahasiswa menggunakan Pydantic. Endpoint yang disediakan antara lain:

- /health untuk pengecekan status,
- /api/acad/mahasiswa untuk daftar mahasiswa,
- /api/acad/ips?nim=... untuk perhitungan IPS.

2.2.8 Pengujian (testing)

Pada tahap ini, berkas *main.py* digunakan sebagai titik utama layanan akademik berbasis FastAPI yang mencakup konfigurasi CORS untuk akses dari luar sistem, koneksi ke PostgreSQL menggunakan *psycopg2*, serta pemodelan data Mahasiswa berbasis Pydantic.

2.3 Tahapan Pengujian Di Insomnia

2.3.1 Cek Konfigurasi API Gateway

Pengujian diawali dengan memverifikasi bahwa API Gateway beroperasi sesuai dengan konfigurasi yang telah ditetapkan. Melalui aplikasi Insomnia, pengguna memasukkan URL <http://localhost:8080> dan mengirimkan permintaan sederhana,

seperti ke endpoint *health*. Apabila API Gateway berhasil meneruskan permintaan ke layanan tujuan tanpa menimbulkan kesalahan, maka dapat disimpulkan bahwa konfigurasi gateway telah berjalan dengan baik dan siap digunakan sebagai titik akses utama sistem.

2.3.2 Implementasi Main Menu (FastAPI - main.py)

Tahap selanjutnya dilakukan dengan menguji layanan akademik yang diimplementasikan menggunakan FastAPI melalui berkas *main.py*. Pengujian dilakukan dengan mengakses endpoint /*health* pada aplikasi Insomnia untuk memastikan layanan berada dalam kondisi aktif. Selanjutnya, dilakukan pengujian pada endpoint /api/acad/mahasiswa dan /api/acad/ips?nim=... guna memverifikasi apakah data mahasiswa serta hasil perhitungan IPS ditampilkan sesuai dengan data yang tersimpan pada basis data. Apabila hasil pengujian sesuai dengan yang diharapkan, maka dapat disimpulkan bahwa implementasi menu utama sistem telah berjalan dengan baik.

2.3.3 Cek Register (Auth Service) - Pilih metode POST.

- Masukkan URL: <http://localhost:8080/api/auth/register>.
- Klik tab Body, ubah format menjadi raw → JSON, lalu isi dengan:

```
{  
    "username": "user1",  
    "email": "user1@email.com",  
    "password": "password123"  
}
```
- Klik Send.
- Jika hasil preview menampilkan message, token, dan user (berisi id, username, email), maka proses register berhasil. Jika muncul error, berarti gagal.

2.3.4 Cek Data Mahasiswa (Acad Service) - pilih metode GET

- Masukkan URL: <http://localhost:8080/api/acad/mahasiswa>.
- Klik Send.
- Jika hasil preview menampilkan data mahasiswa sesuai dengan isi database db_kelas.sql, maka koneksi berhasil. Jika error, berarti gagal.

2.3.5 Cek Perhitungan IPS (Acad Service) - Pilih metode GET.

- Masukkan URL:

<http://localhost:8080/api/acad/ips?nim=2201> atau
<http://localhost:8080/api/acad/ips?nim=2202> - Klik Send.

- Jika hasil preview menampilkan data mahasiswa beserta total SKS, bobot SKS, dan nilai IPS sesuai program di main.py, maka pengujian berhasil. Jika error, berarti gagal.

2.4 Penerapan Pengaturan Lanjutan

Sebagai tambahan, konfigurasi opsional dapat diterapkan untuk meningkatkan kualitas proses deployment. antara lain:

- Mendefinisikan sub-jaringan khusus 172.16.0.0/28
- Mengubah *exposed port* API Gateway menjadi 8081
- Membatasi resource auth-service dengan CPU 0.5 core dan RAM 512 MB
- Mengaktifkan *auto-restart* untuk auth-db

Langkah tersebut memberikan nilai tambah berupa optimalisasi penggunaan resource dan peningkatan reliabilitas sistem.

BAB III

IMPLEMENTASI

3.1 Implementasi

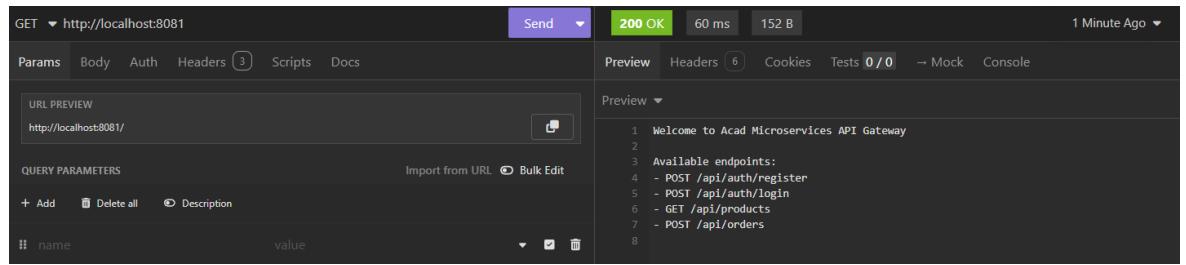
3.1.1 Hasil Implementasi Docker-compose Up --Build -D

```
PS C:\Users\HP ELITEBOOK 840\Documents\Kelas Docker\UAS-AKSO-Docker> docker-compose up --build -d
[+] Building 6.7s (25/25) FINISHED
  => [internal] load local bake definitions
  => => reading from stdin 1.23kB
  => [auth-service internal] load build definition from Dockerfile
  => => transferring dockerfile: 155B
  => [acad-service internal] load build definition from Dockerfile
  => => transferring dockerfile: 242B
  => [acad-service internal] load metadata for docker.io/library/python:3.11-slim
  => [auth-service internal] load metadata for docker.io/library/node:18-alpine
  => [auth] library/node:pull token for registry-1.docker.io
  => [auth] library/python:pull token for registry-1.docker.io
  => [acad-service internal] load .dockerignore
  => => transferring context: 2B
  => [auth-service internal] load .dockerignore
  => => transferring context: 2B
  => [acad-service 1/5] FROM docker.io/library/python:3.11-slim@sha256:158caf0e080e2cd74ef2879ed3c4e697792ee65251c8208b7afb56683c32ea6c
  => => resolve docker.io/library/python:3.11-slim@sha256:158caf0e080e2cd74ef2879ed3c4e697792ee65251c8208b7afb56683c32ea6c
  => [acad-service internal] load build context
  => => transferring context: 233B
  => [auth-service 1/5] FROM docker.io/library/node:18-alpine@sha256:8d421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
  => => resolve docker.io/library/node:18-alpine@sha256:8d421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
  => [auth-service internal] load build context
  => => transferring context: 124B
  => CACHED [acad-service 2/5] WORKDIR /app
  => CACHED [acad-service 3/5] COPY requirements.txt .
  => CACHED [acad-service 4/5] RUN pip install --no-cache-dir -r requirements.txt
  => CACHED [acad-service 5/5] COPY . .
  => [acad-service] exporting to image
  => => exporting layers
  => => exporting manifest sha256:ee9f028b61f4a47967f20b2cf5362b01a9098787124df0656fdb840a64411b02
  => => exporting config sha256:ec2b163fbdb0be65a53a7b9326ea9e3660994a6b9e18b51b3f17728966f230
  => => exporting attestation manifest sha256:8eca7eff24c87a672213ae606922dd05af0a6e2f3714e44eed30e4e81a6be40
  => => exporting manifest list sha256:9853448109e142380fb0731lddb54ce80120cce6e98b7d1ec03f54d6e5206a
  => => naming to docker.io/library/uas-akso-docker-acad-service:latest
  => => unpacking to docker.io/library/uas-akso-docker-acad-service:latest
  => CACHED [auth-service 2/5] WORKDIR /app
  => CACHED [auth-service 3/5] COPY package*.json .
  => CACHED [auth-service 4/5] RUN npm install
  => CACHED [auth-service 5/5] COPY . .
  => [auth-service] exporting to image
  => => exporting layers
  => => exporting manifest sha256:d94037b47bc6c67b9489cb452ef7ceacc46ec59e3d481066b69e52eb1ed5ad45
  => => exporting config sha256:696be1fb5007d03a5bbc86bf2654d2977665fb6b5397afde28013c9b
  => => exporting attestation manifest sha256:56d4d41d5fb2db9b530ae6ff0e3d26d3db71fa8e6c1f006c71ab520c8dcfc
  => => exporting manifest list sha256:b51d280bae2b2ce18271251f2c73fb53455eac52ca228a3dae7d3d0418710e1
  => => naming to docker.io/library/uas-akso-docker-auth-service:latest
  => => unpacking to docker.io/library/uas-akso-docker-auth-service:latest
  => [auth-service] resolving provenance for metadata file
  => [auth-service] resolving provenance for metadata file
[+] Running 8/8
  ✓ uas-akso-docker-acad-service      Built
  ✓ uas-akso-docker-auth-service     Built
  ✓ Network uas-akso-docker_uas-network Created
  ✓ Container auth-db                Started
  ✓ Container acad-db               Started
  ✓ Container auth-service          Started
  ✓ Container acad-service         Started
  ✓ Container api-gateway          Started
PS C:\Users\HP ELITEBOOK 840\Documents\Kelas Docker\UAS-AKSO-Docker> []
```

3.1.2 Pastikan implementasi container Docker(API Gateway, auth-service, auth-db, acad-service, acad-db).

Container CPU usage ⓘ		Container memory usage ⓘ		Show charts	
1.04% / 800% (8 CPUs available)		283.27MB / 7.33GB			
Search		Only show running containers			
Actions	Name	Container ID	Image	Port(s)	CPU (%)
	uas-akso-docker	-	-	-	1.04% 3 hours ago
	acad-db	b3e0ce183592	postgres:15-alpine	-	0.01% 3 hours ago
	auth-db	288a7a2d88ed	mongo:6.0	27017:27017 ⚡	0.76% 3 hours ago
	acad-service	4005aedbba2f	uas-akso-docker-acad-s	-	0.27% 3 hours ago
	auth-service	0dacc47096d0	uas-akso-docker-auth-se	3001:3001 ⚡	0% 3 hours ago
	api-gateway	5c7f1e867a44	nginx:alpine	8081:80 ⚡	0% 3 hours ago

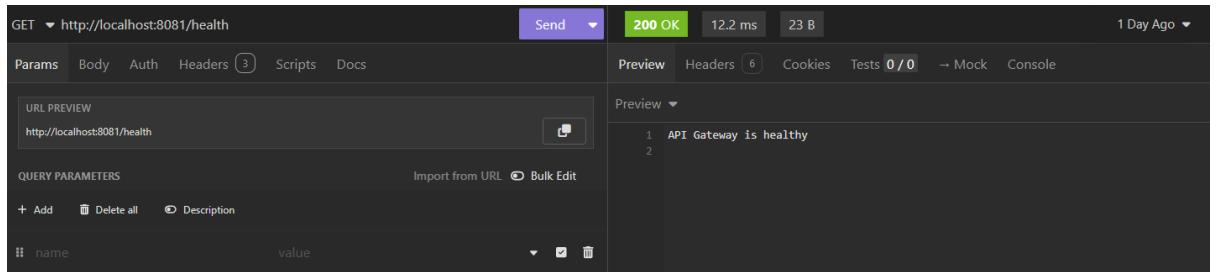
3.1.3 Hasil konfigurasi API Gateway



The screenshot shows the configuration interface for an API Gateway. At the top, it displays a successful GET request to `http://localhost:8081` with a status of 200 OK, 60 ms response time, and 152 B size, timestamped 1 Minute Ago. Below the header, there are tabs for Params, Body, Auth, Headers (3), Scripts, and Docs. The Params tab is selected, showing a URL PREVIEW of `http://localhost:8081/`. The QUERY PARAMETERS section is empty. On the right, the Preview pane shows the following JSON output:

```
1 Welcome to Acad Microservices API Gateway
2
3 Available endpoints:
4 - POST /api/auth/register
5 - POST /api/auth/login
6 - GET /api/products
7 - POST /api/orders
8
```

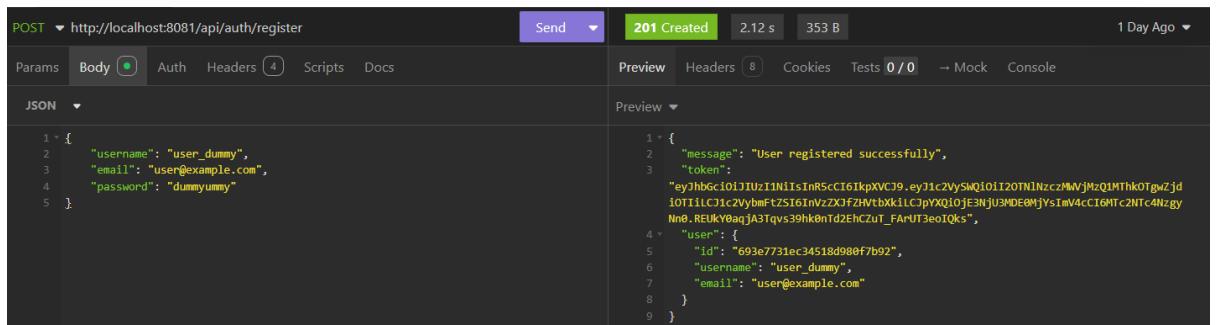
3.1.4 Hasil endpoint /health yang menunjukkan service aktif.



The screenshot shows the configuration interface for an API Gateway. At the top, it displays a successful GET request to `http://localhost:8081/health` with a status of 200 OK, 12.2 ms response time, and 23 B size, timestamped 1 Day Ago. Below the header, there are tabs for Params, Body, Auth, Headers (3), Scripts, and Docs. The Params tab is selected, showing a URL PREVIEW of `http://localhost:8081/health`. The QUERY PARAMETERS section is empty. On the right, the Preview pane shows the following JSON output:

```
1 API Gateway is healthy
2
```

3.1.5 Hasil endpoint /register



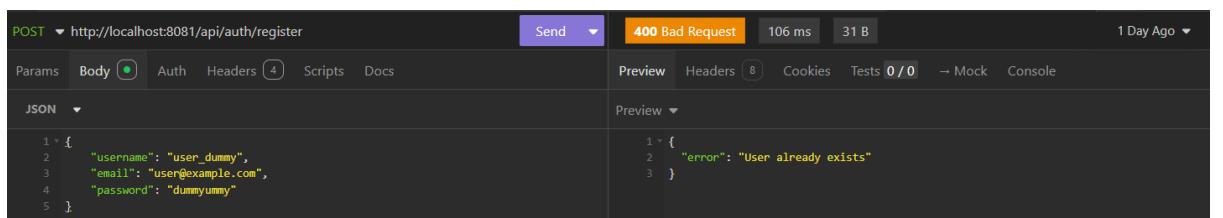
The screenshot shows the configuration interface for an API Gateway. At the top, it displays a successful POST request to `http://localhost:8081/api/auth/register` with a status of 201 Created, 2.12 s response time, and 353 B size, timestamped 1 Day Ago. Below the header, there are tabs for Params, Body (green dot indicating data sent), Auth, Headers (4), Scripts, and Docs. The Body tab is selected, showing a JSON payload:

```
1 + {
2   "username": "user_dummy",
3   "email": "user@example.com",
4   "password": "dummyummy"
5 }
```

On the right, the Preview pane shows the following JSON response:

```
1 + {
2   "message": "User registered successfully",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2OTNIczcMVAJmZQ1MTk0TgwzjdI0T1lC3ic2VybmtZS16InrzXF2zHvbxbk1LCj3YXQ1oje3NjU3MDE0MjYsImV4cCI6MTc2NTc4NzgyNn0.REIUkYbaqJA3Tqvs39hk8nTd2EhcZuT_FArUT3eoIQks",
4 +   "user": {
5     "id": "693e7731ec34518d980f7b92",
6     "username": "user_dummy",
7     "email": "user@example.com"
8   }
9 }
```

3.1.6 hasil endpoint /register (User already exists)



The screenshot shows the configuration interface for an API Gateway. At the top, it displays a failed POST request to `http://localhost:8081/api/auth/register` with a status of 400 Bad Request, 106 ms response time, and 31 B size, timestamped 1 Day Ago. Below the header, there are tabs for Params, Body (green dot), Auth, Headers (4), Scripts, and Docs. The Body tab is selected, showing a JSON payload:

```
1 + {
2   "username": "user_dummy",
3   "email": "user@example.com",
4   "password": "dummyummy"
5 }
```

On the right, the Preview pane shows the following JSON response:

```
1 + {
2   "error": "User already exists"
3 }
```

3.1.7 Hasil Endpoint /Login

POST ▾ http://localhost:8081/api/auth/login Send ▾ 201 Created 2.12 s 353 B 1 Day Ago

Params	Body	Auth	Headers	Scripts	Docs
	JSON				

```

1 * {
2     "username": "user_dummy",
3     "password": "dummyummy"
4 }

```

Preview ▾

```

1 * {
2     "message": "User registered successfully",
3     "token": "eyJhbGkOIjIuIjUzIjNisInR5cCI6IkpXVCJ9.eyJic2VysMqj0Ii20TNlczcMwVjMzQ1MTlkOTgwZjdIOTI1lC1cVybmfZSf6InvzXF2FzHvbK1LClpXY0i0je3NJU3WDE0tjYsImV4cCI6MTc2NTc4NggyNn0.REIukYbaqjA3Tqvs39hk8nTd2EhcCzU_FArUT3eoIQks",
4     "user": {
5         "id": "693e7731ec34518d980f7b92",
6         "username": "user_dummy",
7         "email": "user@example.com"
8     }
9 }

```

3.1.8 Hasil query mahasiswa (/api/acad/mahasiswa) yang menampilkan data dari database.

GET ▾ http://localhost:8081/api/acad/mahasiswa Send ▾ 200 OK 11.7 ms 153 B 3 Days Ago

Params	Body	Auth	Headers	Scripts	Docs

URL PREVIEW
http://localhost:8081/api/acad/mahasiswa

QUERY PARAMETERS Import from URL Bulk Edit

+ Add Delete all Description

name	value

Preview ▾

```

1 * [
2     {
3         "nim": "22001",
4         "nama": "Ahmad Fauzan",
5         "jurusan": "Sains Data",
6         "angkatan": 2022
7     },
8     {
9         "nim": "22002",
10        "nama": "Nisa Rahma",
11        "jurusan": "Sains Data",
12        "angkatan": 2022
13    }
14 ]

```

3.1.9 Hasil perhitungan IPS (/api/acad/ips?nim=22001) sesuai dengan data di db_kelas.sql.

GET ▾ http://localhost:8081/api/acad/ips?nim=22001 Send ▾ 200 OK 17.2 ms 130 B 3 Days Ago

Params	Body	Auth	Headers	Scripts	Docs

URL PREVIEW
http://localhost:8081/api/acad/ips?nim=22001

QUERY PARAMETERS Import from URL Bulk Edit

+ Add Delete all Description

name	value

Preview ▾

```

1 * {
2     "mahasiswa": {
3         "nim": "22001",
4         "nama": "Ahmad Fauzan",
5         "jurusan": "Sains Data"
6     },
7     "total_sks_diamambil": 9,
8     "total_bobot_sks": 10.5,
9     "ips": 1.17
10 }

```

3.1.10 Hasil perhitungan IPS (/api/acad/ips?nim=22002) sesuai dengan data di db_kelas.sql.

GET ▾ http://localhost:8081/api/acad/ips?nim=22002 Send ▾ 200 OK 46.3 ms 128 B 1 Day Ago

Params	Body	Auth	Headers	Scripts	Docs

URL PREVIEW
http://localhost:8081/api/acad/ips?nim=22002

QUERY PARAMETERS Import from URL Bulk Edit

+ Add Delete all Description

name	value

Preview ▾

```

1 * {
2     "mahasiswa": {
3         "nim": "22002",
4         "nama": "Nisa Rahma",
5         "jurusan": "Sains Data"
6     },
7     "total_sks_diamambil": 9,
8     "total_bobot_sks": 10.5,
9     "ips": 1.17
10 }

```

BAB IV

PENUTUP

4.1 Kesimpulan

Implementasi arsitektur microservice dalam konteks akademik melalui Docker Compose untuk layanan autentikasi dan perhitungan Indeks Prestasi Semester (IPS) menunjukkan bahwa pendekatan microservice dapat memberikan

peningkatan dalam fleksibilitas, skalabilitas, serta kemudahan dalam pengelolaan sistem akademik. Dengan memisahkan fungsi autentikasi dan penghitungan IPS ke dalam layanan terpisah, sistem menjadi lebih modular, memfasilitasi proses pengembangan dan pemeliharaan yang lebih efisien. Pemakaian Docker Compose terbukti manjur dalam mengatur beberapa kontainer secara serentak, sehingga tahapan deployment, konfigurasi, dan pengujian sistem dapat dilakukan dengan lebih mudah dan konsisten di berbagai lingkungan. Setiap layanan dapat dioperasikan secara terpisah tanpa saling mengganggu, yang meningkatkan keandalan sistem ketika ada perubahan atau gangguan pada salah satu layanan. Penerapan layanan autentikasi sebagai gerbang akses sistem dapat memperkuat keamanan data akademik, sementara layanan perhitungan IPS dapat mengolah data secara mandiri dengan hasil yang tepat dan cepat. Dengan kata lain, penggunaan arsitektur microservice berbasis Docker Compose bisa menjadi pilihan yang tepat untuk mengembangkan sistem akademik yang modern, dinamis, dan berkelanjutan.

4.2 Saran

Untuk pengembangan berikutnya, sistem ini bisa ditingkatkan melalui penambahan fitur keamanan yang lebih canggih, seperti penerapan JSON Web Token (JWT), OAuth 2.0, ataupun enkripsi komunikasi antar layanan yang menggunakan HTTPS. Ini bertujuan untuk memperkuat perlindungan terhadap data pengguna dan mencegah akses yang tidak sah. Eksplorasi untuk mengelola skala sistem yang lebih luas dan memenuhi kebutuhan ketersediaan tinggi. Terakhir, penting untuk melakukan pengujian kinerja dan pengujian beban secara mendalam untuk mengevaluasi kemampuan sistem dalam mengelola jumlah pengguna yang besar. Dokumentasi teknis yang lebih komprehensif dan pengawasan layanan secara real-time juga sangat dianjurkan agar sistem dapat dioptimalkan dan dikembangkan dengan lebih baik di masa depan.

DAFTAR PUSTAKA

Cerny Tomas, Abdelfattah Amr S., Bushong Vincent, Al Maruf Abdullah, Taibi Davide. (2022). Microservice Architecture Reconstruction and Visualization Techniques: A Review. Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/SOSE55356.2022.00011>

Cherukuri Bangar Raju. (2024). Containerization in cloud computing: comparing Docker and Kubernetes for scalable web applications. International Journal of Science and Research Archive, 2024, 13(01), 3302–3315.
<https://doi.org/10.30574/ijsra.2024.13.1.2035>

Ligios Andrea. (2025). Introduction to Docker Compose. [Introduction to Docker Compose | Baeldung on Ops](#)

[Writing a Dockerfile | Docker Docs](#)