LAPORAN PROJEK AKSO

Disusun oleh :

1. Wiliyan Surya (25031554168)
2. Nayla Salsabila Az Zahra (25031554268)
3. Candra Rafiq Ikromullah (25031554125)

HIMPUNAN MAHASISWA SAINS DATA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS NEGERI SURABAYA

2025

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi saat ini menuntut aplikasi untuk dapat berjalan dengan cepat, stabil, dan mudah dikelola. Salah satu pendekatan yang banyak digunakan adalah containerization dengan teknologi Docker, yang memungkinkan aplikasi dijalankan dalam lingkungan yang konsisten meskipun berpindah perangkat. Selain itu, arsitektur microservice juga semakin populer karena mampu memecah aplikasi menjadi layanan-layanan kecil yang berdiri sendiri, sehingga proses pengembangan, pemeliharaan, dan pengembangan fitur dapat dilakukan dengan lebih fleksibel tanpa harus mengubah keseluruhan sistem.

Di lingkungan akademik, mahasiswa sering dihadapkan pada perhitungan IPS (Indeks Prestasi Semester) sebagai indikator pencapaian akademik. Proses perhitungan yang dilakukan secara manual berpotensi memakan waktu dan menimbulkan kesalahan. Oleh karena itu, diperlukan sebuah aplikasi sederhana yang dapat membantu melakukan perhitungan IPS secara otomatis. Aplikasi ini tidak hanya berfungsi sebagai alat bantu, tetapi juga sebagai media pembelajaran untuk menerapkan teknologi Docker dan konsep microservice agar sistem yang dibangun lebih efisien, terstruktur, dan mudah dijalankan.

Untuk menjalankan microservice secara efektif, dibutuhkan teknologi yang mampu menyediakan lingkungan terisolasi untuk setiap layanan. Docker menjadi salah satu teknologi containerization yang populer karena mampu membungkus aplikasi beserta seluruh dependensinya ke dalam sebuah container yang ringan dan mudah dijalankan pada berbagai platform. Selain itu, ketika sebuah sistem memiliki lebih dari satu layanan, diperlukan mekanisme untuk menjalankan banyak container sekaligus secara terkoordinasi. Dalam hal ini, Docker Compose berperan penting sebagai alat yang mengatur, menghubungkan, dan menjalankan seluruh container dalam satu berkas konfigurasi.

Dalam proyek ini, mahasiswa diminta mengimplementasikan konsep microservice melalui tiga layanan utama, yaitu API Gateway berbasis Nginx, layanan otentikasi (auth-service), dan database MongoDB (auth-db). Ketiga layanan tersebut harus dikonfigurasi agar dapat berkomunikasi melalui internal network yang sama, memiliki dependensi yang jelas, serta dapat berjalan tanpa error. Pengaturan environment variables pada auth-service juga diperlukan untuk memastikan layanan dapat terhubung ke database dan mendukung proses autentikasi dengan benar.

Melalui pengerjaan proyek ini, mahasiswa dituntun untuk tidak sekadar memahami microservice dan container, melainkan secara langsung mempraktikkan alur kerja DevOps fundamental. Ini mencakup strategi deployment, penyiapan internal networking, pengelolaan persistent volume, serta otomatisasi build dan testing melalui Docker

Compose. Pengalaman praktik ini menjadi modal utama untuk membangun dan mengelola sistem skala besar yang membutuhkan skalabilitas dan keandalan tinggi di kancah industri.

1.2 Rumusan Masalah

1. Bagaimana menyusun konfigurasi Docker Compose agar seluruh layanan dapat dijalankan secara bersamaan dalam satu lingkungan?
2. Bagaimana mekanisme pengaturan keterkaitan dan urutan eksekusi antar layanan (service dependency) dalam arsitektur microservice?
3. Bagaimana melakukan pengaturan variabel lingkungan (environment variables) pada layanan autentikasi (auth-service) agar dapat berfungsi dengan baik?
4. Bagaimana cara memastikan setiap container dapat berjalan secara optimal dan bebas dari kesalahan saat proses deployment?

1.3 Tujuan

Tujuan dari proyek "Docker dan Microservice: Aplikasi Perhitungan IPS Mahasiswa" adalah untuk memberikan pemahaman praktis mengenai penerapan teknologi container dan arsitektur microservice dalam pengembangan aplikasi sederhana. Adapun tujuan khusus dari proyek ini meliputi:

1. Menghasilkan konfigurasi *docker-compose.yml* yang dapat menjalankan ketiga layanan secara bersamaan.
2. Mengatur dependensi antar layanan agar urutan start-up berjalan dengan benar.
3. Menambahkan environment variables pada auth-service agar layanan otentikasi berfungsi dan terhubung ke database.
4. Memastikan seluruh container dapat berjalan tanpa error melalui proses build dan pengujian.

Secara lebih mendalam, proyek ini dirancang sebagai sarana pembelajaran untuk memahami bagaimana teknologi Docker dan arsitektur microservice dapat diterapkan secara nyata dalam pengembangan aplikasi. Dengan menggunakan Docker, setiap komponen aplikasi dijalankan dalam container yang terisolasi, sehingga perbedaan sistem operasi atau konfigurasi perangkat tidak memengaruhi kinerja aplikasi. Hal ini bertujuan agar aplikasi perhitungan IPS dapat dijalankan dengan cara yang konsisten, stabil, dan mudah direplikasi di berbagai lingkungan.

Penerapan arsitektur microservice pada proyek ini bertujuan untuk memisahkan fungsi-fungsi utama aplikasi, seperti antarmuka pengguna dan layanan perhitungan IPS, ke dalam layanan yang berdiri sendiri. Pendekatan ini diharapkan dapat memberikan pemahaman kepada pembaca bahwa pengembangan aplikasi tidak harus dilakukan dalam satu kesatuan sistem yang besar, melainkan dapat dibagi menjadi bagian-bagian kecil yang lebih terfokus. Dengan demikian, proses pengembangan, pengujian, dan pengelolaan aplikasi menjadi lebih terstruktur dan mudah dilakukan.

Selain itu, proyek ini bertujuan untuk menghasilkan aplikasi perhitungan IPS mahasiswa yang bersifat otomatis dan akurat, sehingga dapat mengurangi kesalahan yang sering terjadi pada perhitungan manual. Melalui proyek ini, pembaca diharapkan mampu memahami hubungan antara konsep teoritis Docker dan microservice dengan implementasinya dalam sebuah aplikasi sederhana yang memiliki manfaat nyata di lingkungan akademik.

BAB II

LANDASAN TEORI

2.1 Arsitektur Microservices

Arsitektur microservices merupakan pendekatan pengembangan perangkat lunak yang memecah sebuah sistem besar menjadi beberapa layanan kecil (services) yang berdiri secara independen. Setiap layanan memiliki tanggung jawab yang spesifik, dapat dikembangkan, diuji, dan dideploy secara terpisah, serta berkomunikasi menggunakan protokol ringan seperti HTTP berbasis REST API.

Pendekatan ini memberikan berbagai keuntungan, antara lain peningkatan skalabilitas, fleksibilitas dalam pemilihan teknologi, serta isolasi kesalahan (fault isolation) yang lebih baik. Apabila satu layanan mengalami gangguan, layanan lain tetap dapat berjalan tanpa memengaruhi keseluruhan sistem. Oleh karena itu, arsitektur microservices banyak digunakan pada sistem modern dan aplikasi berskala besar (Newman, 2015; Dragoni et al., 2017).

2.2 Docker Container

Docker merupakan teknologi containerization yang memungkinkan aplikasi dijalankan dalam lingkungan terisolasi yang disebut container. Setiap container berisi aplikasi beserta seluruh dependensinya, sehingga aplikasi dapat berjalan secara konsisten di berbagai lingkungan, baik pada komputer lokal maupun server produksi.

Berbeda dengan virtual machine, container tidak memerlukan sistem operasi tersendiri sehingga lebih ringan, cepat dijalankan, dan efisien dalam penggunaan sumber daya. Docker sangat mendukung penerapan arsitektur microservices karena setiap layanan dapat dikemas dalam container terpisah dan dijalankan secara independen (Merkel, 2014; Boettiger, 2015).

2.3 Docker Compose

Docker Compose adalah alat yang digunakan untuk mendefinisikan dan menjalankan beberapa container secara bersamaan melalui satu berkas konfigurasi bernama *docker-compose.yml*. Compose memungkinkan pengaturan network, volume, environment variables, serta dependensi antar layanan sehingga seluruh container dapat berjalan terkoordinasi dalam satu sistem microservice. Alat ini sangat bermanfaat dalam

proses automasi deployment dan pengujian layanan berbasis container *(Turnbull, 2014; Raj & Radhakrishnan, 2020)*.

2.4 Nginx

Nginx merupakan web server dan reverse proxy yang banyak digunakan sebagai API Gateway dalam arsitektur microservices. Sebagai API Gateway, Nginx berfungsi sebagai pintu masuk utama bagi seluruh request dari client sebelum diteruskan ke layanan internal.

Peran utama Nginx dalam sistem ini meliputi routing request ke layanan yang sesuai, penyederhanaan endpoint API, serta pemisahan antara client dan layanan internal. Dengan adanya API Gateway, client tidak berinteraksi langsung dengan sub-layanan, sehingga sistem menjadi lebih aman, terstruktur, dan mudah dikembangkan (Reese, 2018; Adamczyk & Rabiej, 2020).

2.5 MongoDB

MongoDB adalah sistem manajemen basis data NoSQL berbasis dokumen yang menyimpan data dalam format BSON (Binary JSON). MongoDB bersifat schema-less, sehingga struktur data dapat lebih fleksibel dibandingkan database relasional.

Karakteristik tersebut membuat MongoDB cocok digunakan dalam arsitektur microservices, terutama untuk layanan otentikasi yang memerlukan penyimpanan data pengguna secara dinamis. MongoDB juga mendukung skalabilitas horizontal dan performa yang baik untuk aplikasi modern (Chodorow, 2013; Pokorny, 2013).

BAB III

ANALISIS DAN PERANCANGAN

3.1 Struktur Layanan

Sistem microservice yang dibangun terdiri dari beberapa sub-layanan yang saling terhubung melalui satu internal network Docker. Setiap layanan memiliki peran dan tanggung jawab yang jelas, sebagai berikut:

1. API Gateway (Nginx)

1. API-Gateway (Nginx)
   API Gateway berfungsi sebagai gerbang utama sistem yang menerima seluruh request dari client. Layanan ini menggunakan Nginx sebagai reverse proxy untuk meneruskan request ke sub-layanan yang sesuai. Konfigurasi routing diatur melalui berkas nginx.conf yang di-mount ke dalam container dengan mode read-only. API Gateway hanya dapat berjalan apabila layanan yang menjadi dependensinya telah aktif.
2. Auth-Service (Node.js)
   Auth-Service bertanggung jawab menangani proses otentikasi pengguna, seperti registrasi dan login. Layanan ini dijalankan menggunakan Node.js dan terhubung ke database MongoDB melalui environment variable MONGO_URI. Auth-Service bergantung pada auth-db agar koneksi database dapat dilakukan dengan benar saat startup container.
3. Auth-DB (MongoDB)

Auth-DB berfungsi sebagai penyimpan data pengguna dalam bentuk dokumen. Untuk menjaga persistensi data, layanan ini menggunakan volume sehingga data tetap tersimpan meskipun container dihentikan atau direstart. Auth-DB menjadi dependensi utama bagi Auth-Service.

Alur komunikasi antar layanan dapat digambarkan sebagai berikut:

Client → API Gateway → Auth-Service → Auth-DB

Seluruh layanan berjalan dalam satu network internal Docker sehingga dapat saling berkomunikasi tanpa terekspos langsung ke jaringan luar.

3.2 Cara kerja sistem

Sistem microservice yang dibangun terdiri dari API Gateway, Auth Service, dan Auth Database yang saling terintegrasi. Alur kerja sistem secara umum adalah sebagai berikut:

1. **Client Mengirimkan Request**
   Client, seperti browser atau Thunder Client, mengirimkan request HTTP ke endpoint sistem, misalnya:

- POST /api/auth/register

- POST /api/auth/login

Seluruh request ini pertama kali diterima oleh API Gateway.

2. **API Gateway Melakukan Routing**
   API Gateway bertindak sebagai pintu masuk utama sistem. Nginx melakukan routing request berdasarkan konfigurasi pada `nginx.conf` dan meneruskannya ke Auth-Service yang berjalan pada port internal tertentu. Client tidak mengetahui secara langsung alamat maupun detail implementasi layanan internal.

3. **Auth-Service Memproses Logika Aplikasi**
   Auth-Service menerima request dari API Gateway dan menjalankan logika aplikasi. Pada proses registrasi, layanan ini melakukan hashing password dan menyimpan data pengguna ke Auth-DB. Pada proses login, layanan memverifikasi kredensial pengguna dan menghasilkan token JWT jika valid. Seluruh konfigurasi layanan bergantung pada environment variable yang telah ditentukan.

4. **Auth-Service Terhubung ke Auth-DB**
   Untuk operasi penyimpanan dan pengambilan data, Auth-Service terhubung ke MongoDB melalui internal network Docker menggunakan URI:
   mongodb://auth-db:27017/auth

5. **Response Dikembalikan ke Client**
   Setelah proses selesai, Auth-Service mengirimkan response ke API Gateway, kemudian API Gateway meneruskan response tersebut ke client. Response dapat berupa token JWT, data pengguna, atau pesan kesalahan sesuai hasil proses.

Dengan alur ini, sistem bekerja secara modular, aman, dan terisolasi sesuai prinsip arsitektur microservices, serta mudah dikembangkan dan diuji menggunakan alat seperti Thunder Client.

BAB IV
IMPLEMENTASI

4.1 Network dan Volume

Pada fase implementasi, sistem microservices dikonfigurasikan agar seluruh layanan berjalan dalam satu jaringan internal yang terisolasi. Tujuan utama dari penggunaan jaringan internal ini adalah untuk memastikan komunikasi antarlayanan dapat berlangsung secara aman tanpa membuka akses langsung ke jaringan eksternal. Dengan pendekatan ini, setiap service hanya dapat diakses oleh service lain yang berada dalam network yang sama.

Network yang digunakan memiliki spesifikasi sebagai berikut:

- Nama network: microservices-network
- Driver: bridge
- Subnet: 172.20.0.0/16

Konfigurasi tersebut memungkinkan setiap container saling terhubung secara otomatis menggunakan nama service sebagai host, sehingga mempermudah proses integrasi dan mengurangi kompleksitas pengaturan alamat IP.

Selain pengaturan jaringan, sistem ini juga menerapkan mekanisme volume untuk menjaga persistensi data pada layanan basis data. Volume digunakan agar data tetap tersimpan meskipun container dihentikan, dihapus, atau dibangun ulang. Dua volume utama yang digunakan dalam sistem ini adalah:

- **auth-data**, digunakan untuk menyimpan data MongoDB pada layanan autentikasi.
- **acad-data**, digunakan untuk menyimpan data PostgreSQL pada layanan akademik.

Penggunaan volume menjadi komponen penting dalam menjaga konsistensi dan keandalan data pada sistem berbasis container.

4.2 Konfigurasi docker-compose.yml

Seluruh layanan microservices didefinisikan dan dijalankan melalui satu berkas konfigurasi docker-compose.yml. Berkas ini berfungsi sebagai pusat pengaturan yang mencakup pendefinisian service, network, volume, environment variables, serta dependensi antar layanan.

a. API Gateway (Nginx)

API Gateway diimplementasikan menggunakan image nginx: alpine dan berperan sebagai gerbang utama yang menerima seluruh permintaan dari client. File konfigurasi

nginx.conf di-mount ke dalam container dengan mode read-only untuk mengatur mekanisme reverse proxy. Melalui API Gateway, request dari client diteruskan ke auth-service maupun acad-service sesuai dengan endpoint yang diakses.

b. Auth-Service (Node.js)

Auth-service bertugas menangani proses autentikasi pengguna, meliputi registrasi, login, dan verifikasi akun. Layanan ini menggunakan environment variables untuk mengatur konfigurasi penting seperti port aplikasi, mode eksekusi, alamat koneksi MongoDB, serta secret key untuk pembuatan JSON Web Token (JWT). Pendekatan ini memungkinkan pengelolaan konfigurasi yang lebih fleksibel dan aman.

c. Auth-DB (MongoDB)

Auth-db merupakan layanan basis data MongoDB yang berfungsi menyimpan data pengguna. Database ini terhubung dengan volume auth-data sehingga data bersifat persistent. Auth-db menjadi komponen utama yang mendukung keberlangsungan proses autentikasi dalam sistem.

d. Acad-Service (Node.js)

Acad-service adalah layanan akademik yang bertanggung jawab dalam pengolahan data mahasiswa, khususnya perhitungan Indeks Prestasi Semester (IPS). Layanan ini terhubung ke database PostgreSQL melalui jaringan internal dan memanfaatkan environment variables untuk konfigurasi koneksi database.

e. Acad-DB (PostgreSQL)

Acad-db merupakan database relasional berbasis PostgreSQL yang digunakan untuk menyimpan data akademik mahasiswa. Database ini dihubungkan dengan volume acad-data agar data tetap terjaga meskipun container mengalami penghentian atau restart.

4.3 Dependensi Antar Layanan

Dalam sistem microservices, pengaturan dependensi antar layanan menjadi hal yang krusial untuk mencegah kegagalan saat proses startup. Beberapa service memerlukan service lain untuk aktif terlebih dahulu, khususnya layanan yang bergantung pada database.

Dependensi antar layanan dalam sistem ini diatur sebagai berikut:

1. API Gateway bergantung pada auth-service, karena gateway hanya dapat meneruskan request jika layanan autentikasi telah aktif.
2. Auth-service bergantung pada auth-db, sebab seluruh proses autentikasi memerlukan akses ke database MongoDB.
3. Auth-db dijalankan lebih awal sebagai fondasi layanan autentikasi agar data pengguna tersedia saat service lain mulai berjalan.

Pengaturan dependensi ini memastikan seluruh container dapat berjalan secara berurutan dan meminimalkan potensi error akibat koneksi service yang belum siap.

BAB V

PENGUJIAN DAN PEMBAHASAN

5.1 Tujuan Pengujian

Tahap pengujian dilakukan untuk memastikan bahwa sistem informasi akademik berbasis microservices yang telah diimplementasikan dapat berjalan sesuai dengan perancangan. Pengujian ini difokuskan pada keterhubungan antarlayanan, kestabilan container, serta keberhasilan alur fungsional utama sistem, khususnya proses autentikasi pengguna dan penghitungan Indeks Prestasi Semester (IPS) mahasiswa.

5.2 Proses Menjalankan dan Memantau Container

Langkah awal pengujian dimulai dengan menjalankan seluruh layanan menggunakan Docker Compose melalui perintah ***docker-compose up -d.*** Perintah ini bertujuan untuk membangun image yang diperlukan sekaligus menjalankan seluruh container dalam mode background.

Setelah seluruh layanan dijalankan, dilakukan pengecekan status container menggunakan perintah ***docker-compose ps.*** Melalui perintah ini dapat dipastikan bahwa setiap service berada dalam kondisi aktif (running), tidak mengalami error, serta telah menggunakan port dan network yang sesuai dengan konfigurasi.

5.3 Skenario Pengujian Fungsional

Pengujian fungsional dilakukan untuk memastikan setiap fitur utama sistem berjalan dengan baik. Proses pengujian dilakukan menggunakan Postman sebagai client untuk mengirim request ke API Gateway.

5.3.1 Pengujian Registrasi Pengguna

Pengujian registrasi dilakukan dengan mengirim request POST ke endpoint registrasi melalui API Gateway. Data yang dikirimkan berupa username, email, dan password dalam format JSON. Sistem berhasil memproses request dan menyimpan data pengguna ke dalam database MongoDB, yang ditandai dengan respons keberhasilan dari server.

5.3.2 Pengujian Verifikasi Akun

Setelah proses registrasi, dilakukan pengujian verifikasi akun. Client mengirim request POST ke endpoint verifikasi dengan data pengguna yang telah terdaftar. Sistem

mampu memvalidasi keberadaan data pengguna dan memberikan respons sesuai dengan status verifikasi yang dihasilkan.

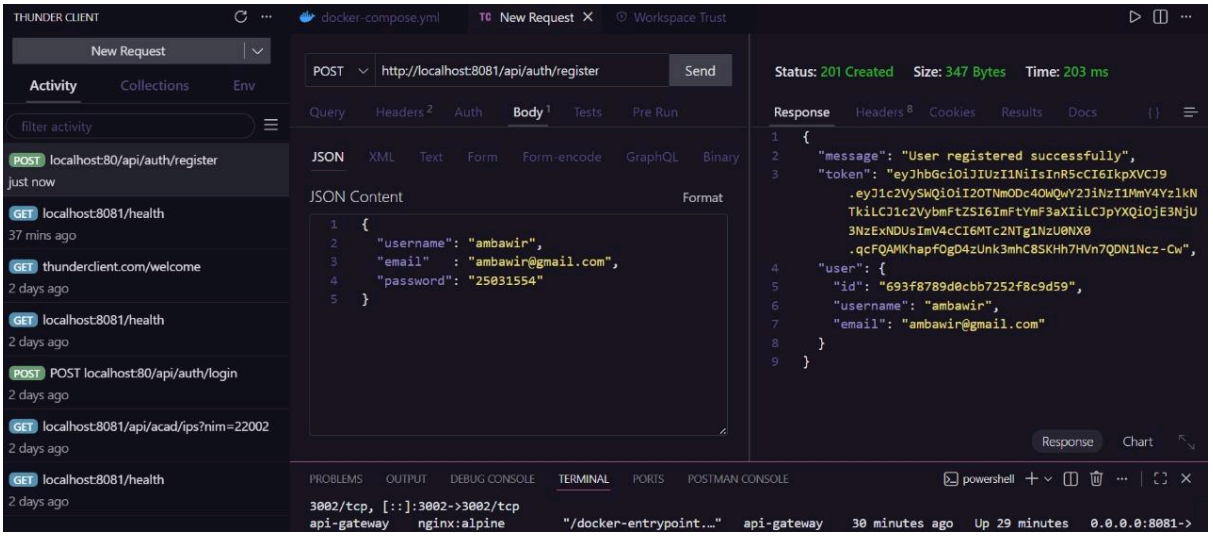
5.3.3 Pengujian Login Pengguna

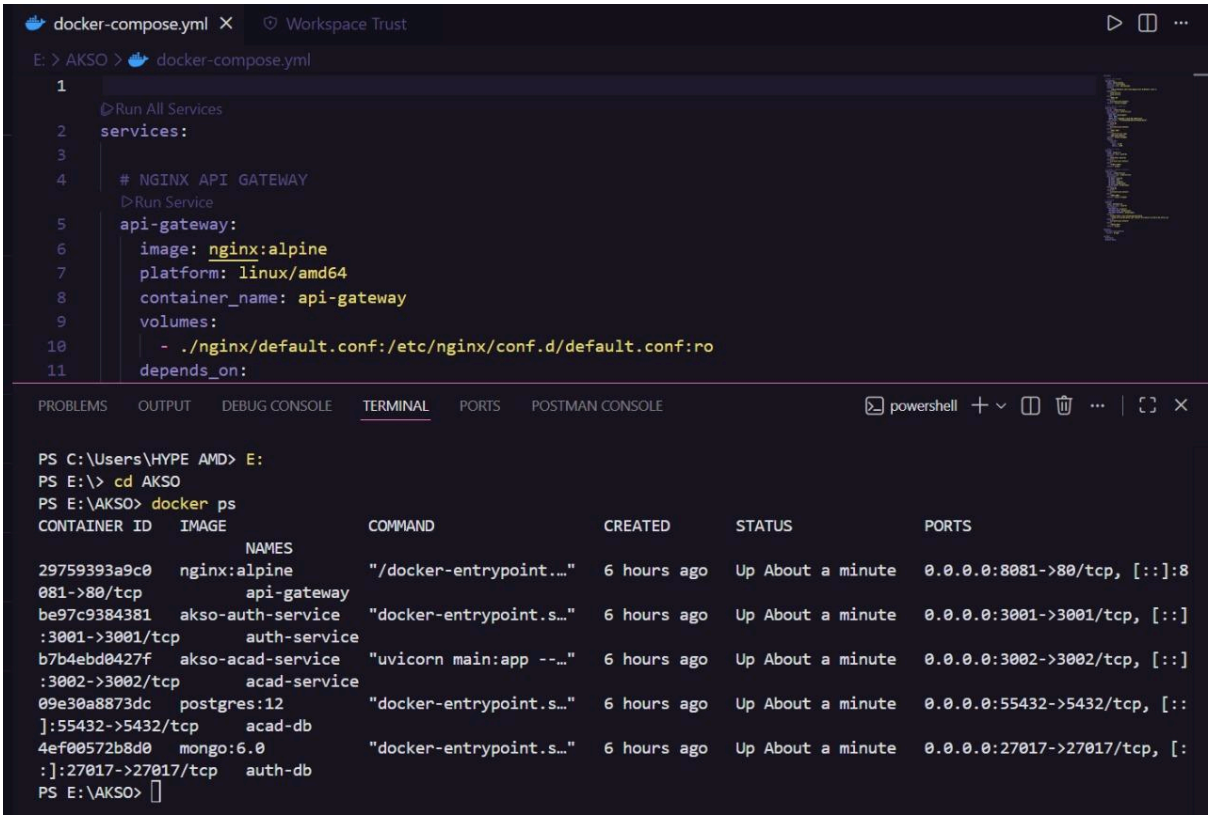
Pengujian login dilakukan dengan mengirimkan username, email, dan password yang valid. Jika data yang diberikan sesuai dengan data di database, sistem mengembalikan token JSON Web Token (JWT) sebagai bukti autentikasi. Keberhasilan proses ini menunjukkan bahwa mekanisme login dan manajemen autentikasi telah berjalan dengan benar.

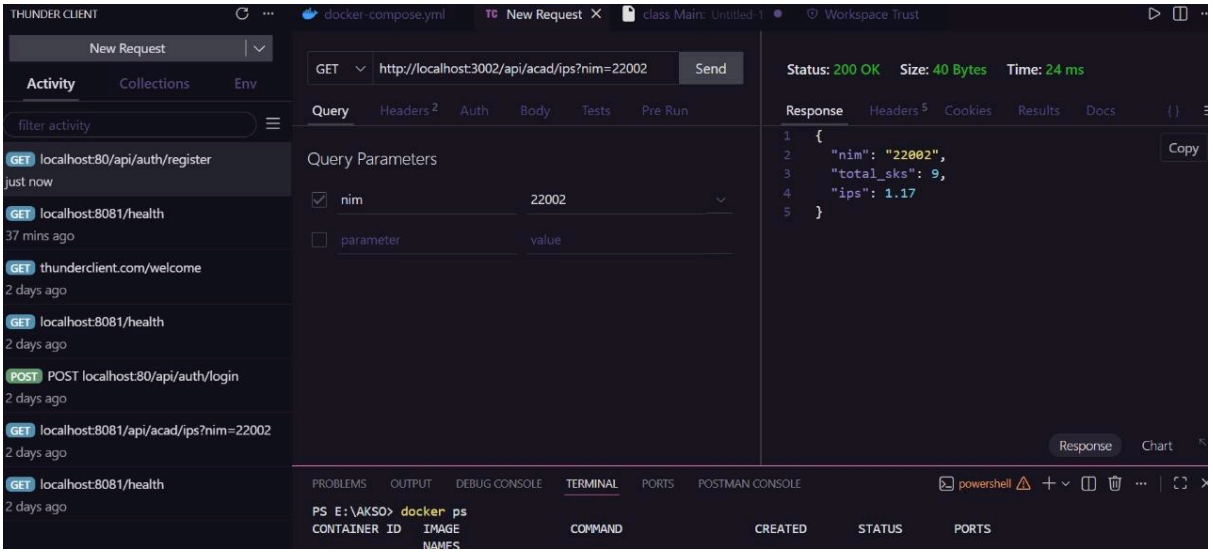
5.3.4 Pengujian Penghitungan IPS Mahasiswa

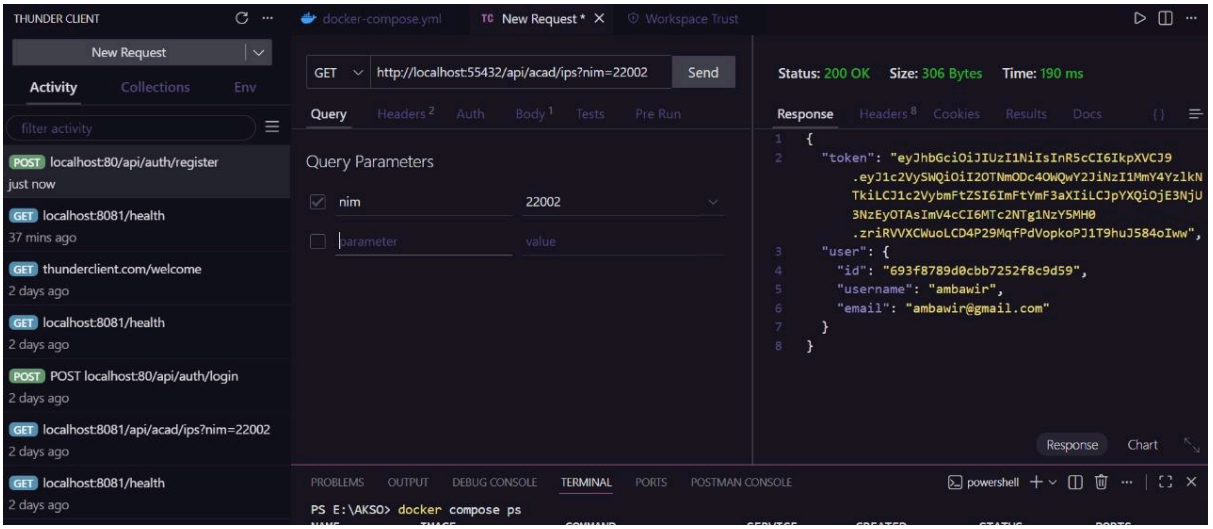
Tahap selanjutnya adalah pengujian layanan akademik. Client mengirim request GET ke endpoint penghitungan IPS dengan menyertakan parameter NIM mahasiswa. Sistem berhasil mengambil data akademik dari database PostgreSQL dan menampilkan informasi berupa NIM, nama mahasiswa, jurusan, total SKS, serta nilai IPS.

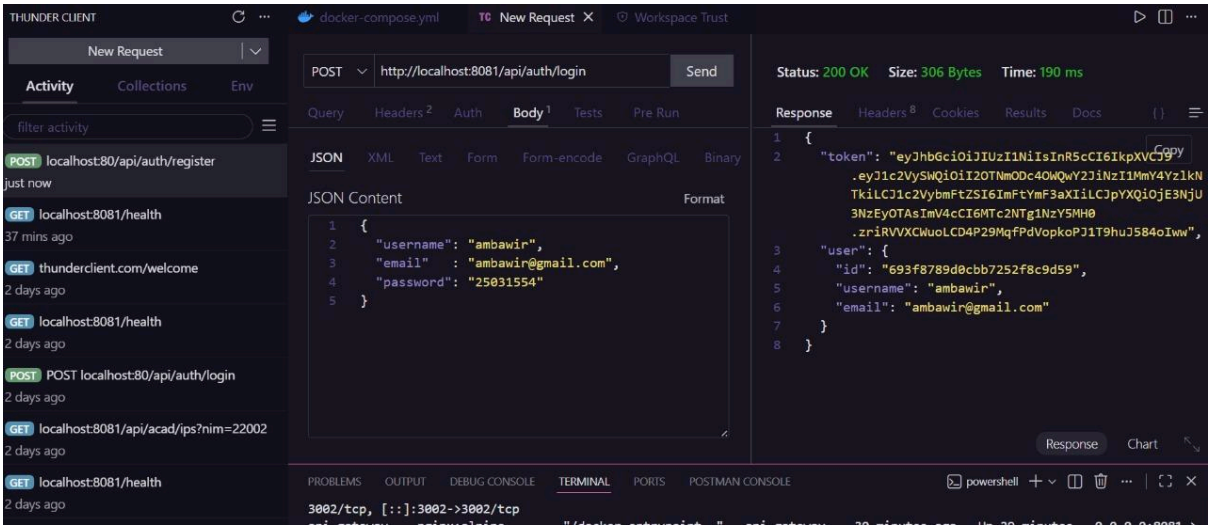
5.4 Hasil Pengujian




docker-compose.yml
Workspace Trust
E: > AKSO > docker-compose.yml
Run All Services
services:
NGINX API GATEWAY
Run Service
api-gateway:
image: nginx:alpine
platform: linux/amd64
container_name: api-gateway
volumes:
- ./nginx/default.conf:/etc/nginx/conf.d/default.conf:ro
depends_on:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
powershell
PS C:\Users\HYPE AMD> E:
PS E:\> cd AKSO
PS E:\AKSO> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
29759393a9c0 nginx:alpine "/docker-entrypoint.…" 6 hours ago Up About a minute 0.0.0.0:8081->80/tcp, [::]:8
081->80/tcp api-gateway
be97c9384381 akso-auth-service "docker-entrypoint.s…" 6 hours ago Up About a minute 0.0.0.0:3001->3001/tcp, [::]
:3001->3001/tcp auth-service
b7b4ebd0427f akso-acad-service "uvicorn main:app --…" 6 hours ago Up About a minute 0.0.0.0:3002->3002/tcp, [::]
:3002->3002/tcp acad-service
09e30a8873dc postgres:12 "docker-entrypoint.s…" 6 hours ago Up About a minute 0.0.0.0:55432->5432/tcp, [::
]:55432->5432/tcp acad-db
4ef00572b8d0 mongo:6.0 "docker-entrypoint.s…" 6 hours ago Up About a minute 0.0.0.0:27017->27017/tcp, [:
:]:27017->27017/tcp auth-db
PS E:\AKSO>


THUNDER CLIENT
New Request
Activity Collections Env
filter activity
GET localhost:80/api/auth/register
just now
GET localhost:8081/health
37 mins ago
GET thunderclient.com/welcome
2 days ago
GET localhost:8081/health
2 days ago
POST POST localhost:80/api/auth/login
2 days ago
GET localhost:8081/api/acad/ips?nim=22002
2 days ago
GET localhost:8081/health
2 days ago
docker-compose.yml
New Request
class Main: Untitled-1
Workspace Trust
GET http://localhost:3002/api/acad/ips?nim=22002
Send
Query Headers 2 Auth Body Tests Pre Run
Query Parameters
nim 22002
parameter value
Status: 200 OK Size: 40 Bytes Time: 24 ms
Response Headers 5 Cookies Results Docs
{
"nim": "22002",
"total_sks": 9,
"ips": 1.17
}
Copy
Response Chart
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
powershell
PS E:\AKSO> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES





Berdasarkan seluruh skenario pengujian yang telah dilakukan, sistem menunjukkan hasil yang sesuai dengan perancangan. Seluruh layanan dapat berkomunikasi melalui API Gateway, database dapat diakses dengan baik oleh service terkait, dan setiap fitur utama sistem dapat dijalankan tanpa kendala.

5.5 Pembahasan

Hasil pengujian membuktikan bahwa penerapan arsitektur microservices menggunakan Docker Compose mampu menghasilkan sistem yang modular dan terintegrasi. Pemisahan layanan autentikasi dan layanan akademik memberikan fleksibilitas dalam pengelolaan sistem serta mempermudah proses pengembangan dan pemeliharaan di masa mendatang.

Keberadaan API Gateway sebagai titik akses tunggal meningkatkan keamanan dan keteraturan komunikasi antarlayanan. Selain itu, penggunaan network internal dan volume persistent mendukung kestabilan sistem serta menjaga integritas data. Secara keseluruhan,

sistem yang dibangun telah memenuhi tujuan proyek dan layak digunakan sebagai dasar pengembangan sistem informasi akademik berbasis microservices.

BAB VI

PENUTUP

6.1 Kesimpulan

Berdasarkan laporan proyek akhir tersebut, berikut adalah kesimpulan yang dapat diambil:

- Implementasi Arsitektur Microservices

  Proyek ini berhasil mengimplementasikan arsitektur *microservices* yang membagi sistem besar menjadi layanan-layanan kecil yang independen, yaitu *API Gateway* (Nginx), *Auth-Service* (Node.js), *Auth-DB* (MongoDB), *Acad-Service*, dan *Acad-DB* (PostgreSQL).
- Efisiensi dengan Docker dan Docker Compose

  Penggunaan *Docker* memungkinkan aplikasi berjalan dalam lingkungan yang terisolasi dan konsisten. Sementara itu, *Docker Compose* berperan penting dalam mengatur, menghubungkan, dan menjalankan seluruh *container* secara terkoordinasi melalui satu berkas konfigurasi docker-compose.yml.
- Manajemen Komunikasi dan Keamanan

  Seluruh layanan dikonfigurasi untuk berkomunikasi melalui jaringan internal (*microservices-network*) sehingga aman dan tidak terekspos langsung ke luar sistem.

API Gateway berbasis Nginx berfungsi sebagai pintu masuk utama yang mengarahkan permintaan ke layanan yang sesuai.

- Keamanan Data (Persistence)

  Implementasi *volume* pada basis data (MongoDB dan PostgreSQL) memastikan bahwa data tetap tersimpan secara permanen meskipun *container* dihentikan atau dihapus.
- Pengaturan Dependensi

  Pengaturan dependensi antar layanan sangat krusial untuk memastikan urutan *startup* yang benar, seperti layanan *Auth-Service* yang harus menunggu basis data (*Auth-DB*) aktif sebelum dapat mulai beroperasi.
- Hasil Pengujian

  Melalui pengujian menggunakan Postman, sistem terbukti mampu menjalankan fungsi-fungsi utama seperti registrasi pengguna, verifikasi status, *login* (menghasilkan token JWT), dan pengecekan Indeks Prestasi Semester (IPS) mahasiswa.

6.2 Saran

Untuk pengembangan lebih lanjut, terdapat beberapa saran yang dapat dipertimbangkan:

1. Peningkatan Keamanan

   Sistem dapat dikembangkan dengan menambahkan validasi token JWT pada setiap endpoint layanan akademik agar hanya pengguna terautentikasi yang dapat mengakses data IPS mahasiswa.

2. Penerapan Load Balancing dan Scaling

   Pada tahap selanjutnya, sistem dapat ditingkatkan dengan menambahkan mekanisme load balancing dan auto-scaling untuk menangani jumlah request yang lebih besar.

3. Pemisahan Lingkungan Development dan Production

   Disarankan untuk membuat konfigurasi terpisah antara lingkungan development dan production guna meningkatkan keamanan serta kestabilan sistem.

4. Monitoring dan Logging

   Penambahan sistem monitoring dan logging akan membantu dalam mendeteksi error, memantau performa layanan, serta mempermudah proses debugging.

DAFTAR PUSTAKA

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). *Microservices: Yesterday, Today, and Tomorrow*. Present and Ulterior Software Engineering (PUSE).

Lewis, J., & Fowler, M. (2014). *Microservices: a definition of this new architectural term*. martinfowler.com.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

Richardson, C. (2018). *Microservices Patterns: With examples in Java*. Manning Publications.

Merkel, D. (2014). *Docker: Lightweight Linux Containers for Consistent Development and Deployment*. Linux Journal, 2014(239).

Boettiger, C. (2015). *An introduction to Docker for reproducible research*. ACM SIGOPS Operating Systems Review, 49(1), 71–79.

Nginx Inc. (2023). *NGINX Reverse Proxy Documentation*. nginx.org.

Node.js Foundation. (2023). *Node.js Documentation*. nodejs.org.

MongoDB Inc. (2023). *MongoDB Manual*. mongodb.com.

The PostgreSQL Global Development Group. (2023). *PostgreSQL Documentation*. postgresql.org.

Turnbull, J. (2014). *The Docker Book: Containerization is the New Virtualization*. James Turnbull Publishing.

Hoffman, B., & Wolf, R. (2020). *Cloud Native Applications with Docker and Kubernetes*. Addison-Wesley.

Ono, H. (2019). *Using Docker Compose for Multi-Container Applications*. Journal of Cloud Computing Systems, 7(2), 45–53.