LAPORAN AKHIR PROYEK (DOCKER) ARSITEKTUR KOMPUTER DAN SISTEM OPERASI

NAMA KELOMPOK:

1. **Bulan Safitri (25031554048)**
2. **Salum Nur Anisa Septriani (250315534046)**
3. **Syabina Suwardhana Poetri (25031554-)**
4. **Talitha Najwa Karima (25031554246)**

UNIVERSITAS NEGERI SURABAYA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM / SAINS DATA

TAHUN 2025

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan Laporan Akhir Proyek (Docker) pada Mata Kuliah Arsitektur Komputer dan Sistem Operasi dengan baik dan tepat waktu. Laporan ini disusun sebagai salah satu bentuk pemenuhan tugas akhir proyek serta sebagai sarana untuk mengimplementasikan pemahaman teori ke dalam praktik nyata, khususnya dalam penerapan teknologi kontainerisasi menggunakan Docker.

Penyusunan laporan ini bertujuan untuk menjelaskan proses perancangan, implementasi, dan pengujian sistem berbasis Docker yang berkaitan dengan konsep arsitektur komputer dan sistem operasi. Melalui proyek ini, penulis diharapkan mampu memahami bagaimana sistem operasi bekerja dalam mengelola sumber daya, serta bagaimana Docker dapat digunakan untuk melakukan deployment aplikasi secara efisien, terisolasi, dan terstruktur.

Penulis menyadari bahwa dalam penyusunan laporan ini tidak terlepas dari bantuan dan bimbingan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada dosen pengampu Mata Kuliah Arsitektur Komputer dan Sistem Operasi yang telah memberikan arahan dan ilmu pengetahuan, serta kepada semua pihak yang telah membantu secara langsung maupun tidak langsung dalam penyelesaian laporan ini.

Penulis menyadari bahwa laporan ini masih memiliki keterbatasan dan kekurangan. Oleh sebab itu, penulis mengharapkan kritik dan saran yang bersifat membangun demi penyempurnaan laporan ini di masa mendatang. Semoga laporan ini dapat memberikan manfaat dan menambah wawasan bagi pembaca, khususnya dalam bidang arsitektur komputer, sistem operasi, dan teknologi Docker.

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi mendorong penerapan sistem berbasis *microservice* dan *Application Programming Interface* (API) pada berbagai bidang, termasuk layanan kesehatan dan akademik. Pendekatan ini memungkinkan sistem dibangun secara modular, terintegrasi, serta mudah dikembangkan. Dalam praktiknya, banyak sistem yang digunakan masih berupa *legacy software* yang memerlukan penyesuaian fitur dan konfigurasi agar tetap relevan dengan kebutuhan saat ini. Oleh karena itu, pemahaman mengenai pemrograman dasar, arsitektur *microservice*, serta proses *deployment* menjadi keterampilan penting bagi mahasiswa di bidang sains data dan teknologi informasi.

Pada proyek ini, sistem disimulasikan melalui sebuah *Hospital System* yang memiliki dua sub-layanan utama, yaitu layanan Otentikasi (*auth*) untuk pengelolaan akun dan layanan Penilaian (*score*) untuk perhitungan Indeks Prestasi Semester (IPS) mahasiswa. Kedua layanan diimplementasikan menggunakan konsep *Application Programming Interface* (API) dan dijalankan dalam lingkungan terpisah menggunakan *Docker* serta dikelola dengan *Docker Compose*. Pendekatan ini memungkinkan setiap layanan berjalan secara independen namun tetap saling terhubung, sehingga sistem menjadi lebih fleksibel dan mudah dikembangkan.

Selain aspek pemrograman, proyek ini juga menekankan pentingnya peran *DevOps* dalam mengelola siklus pengembangan dan *deployment* perangkat lunak. Melalui pemanfaatan *skeleton assets* berupa *Dockerfile*, *docker-compose.yml*, dan *source code* yang telah tersedia, mahasiswa dilatih untuk memahami struktur sistem yang sudah ada, melakukan konfigurasi ulang, serta menambahkan fitur baru tanpa mengganggu fungsionalitas utama sistem. Dengan demikian, proyek ini tidak hanya meningkatkan kemampuan teknis pemrograman, tetapi juga membangun pemahaman praktis mengenai pengelolaan sistem terdistribusi yang umum digunakan di dunia industri.

1.2 Rumusan Masalah

1. Bagaimana cara mengonfigurasi *docker-compose.yml* agar seluruh sub-layanan dalam sistem dapat berjalan secara terintegrasi?
2. Bagaimana implementasi pemrograman dasar pada *Application Programming Interface* (API) *service* untuk menghitung Indeks Prestasi Semester (IPS) mahasiswa?

3. Bagaimana penerapan konsep *microservice* dan *containerization* pada sistem *legacy* berbasis *Application Programming Interface* (API)?

1.3 Tujuan

1. Mengonfigurasi dan menjalankan keseluruhan layanan sistem menggunakan *Docker* dan *Docker Compose*.
2. Mengimplementasikan pemrograman dasar pada API *service* untuk melakukan perhitungan IPS mahasiswa.
3. Menerapkan konsep API dan *microservice* pada sistem *legacy* yang telah tersedia.

BAB II
ISI

2.1 Konfigurasi Docker Compose

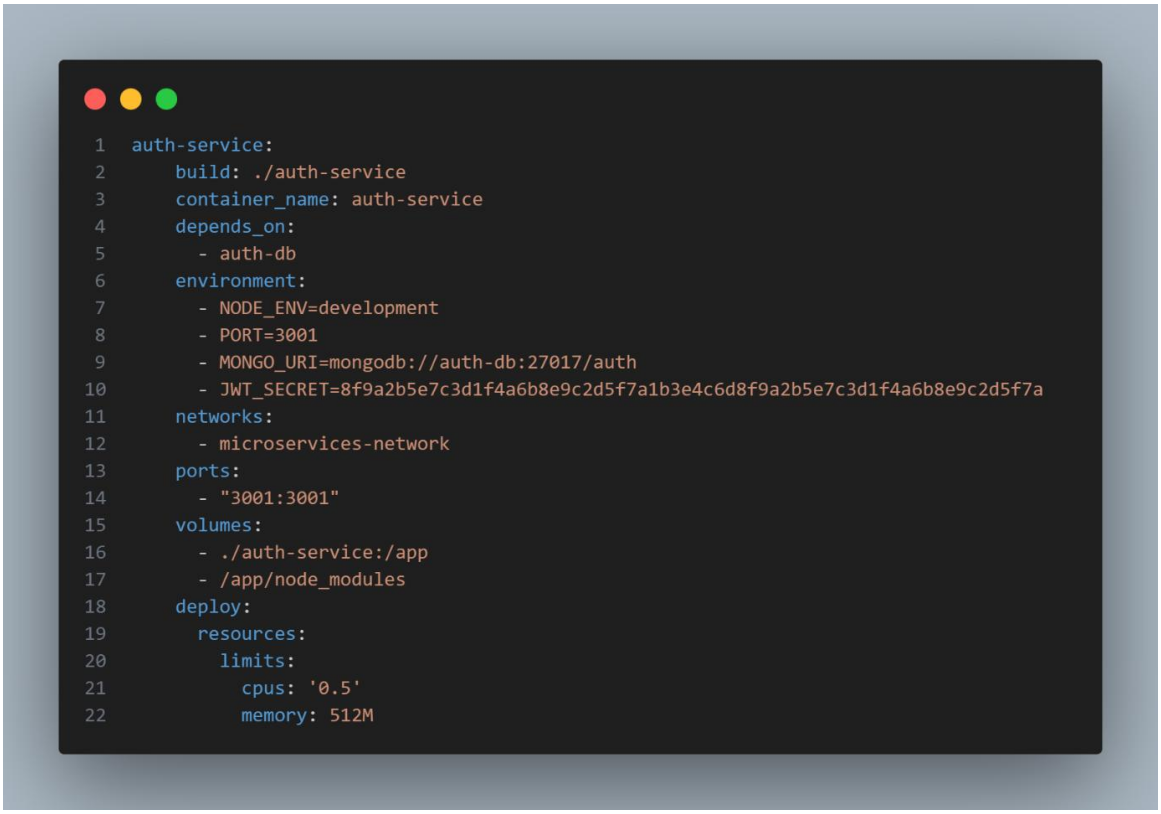
Pada bagian ini, sistem dikonfigurasi menggunakan Docker Compose agar semua layanan bisa berjalan bersama-sama secara terintegrasi. Dengan pendekatan ini, setiap komponen mulai dari API Gateway, layanan autentikasi, hingga layanan akademik dijalankan dalam container masing-masing, tetapi tetap saling terhubung melalui satu jaringan. Hal ini membuat sistem lebih rapi, mudah diatur, dan fleksibel untuk dikembangkan.

2.1.1 API Gateway (Nginx)

Gambar 2. 1 Konfigurasi layanan API Gateway (Nginx) pada Docker Compose.

Konfigurasi API Gateway menggunakan Nginx ditunjukkan pada Gambar 2.1, berfungsi sebagai pintu masuk utama ke dalam sistem microservices. Semua permintaan dari pengguna akan diarahkan terlebih dahulu ke gateway ini sebelum diteruskan ke layanan yang sesuai. Layanan ini menggunakan image nginx:alpine dan memanfaatkan file konfigurasi nginx.conf yang dipasang ke dalam container secara read-only. Port eksternal diubah menjadi 8081 agar tidak bentrok dengan port default Nginx. Selain itu, layanan ini baru dapat berjalan setelah layanan autentikasi aktif, karena terdapat dependensi terhadap auth-service.

2.1.2 Auth Service (Node.js)

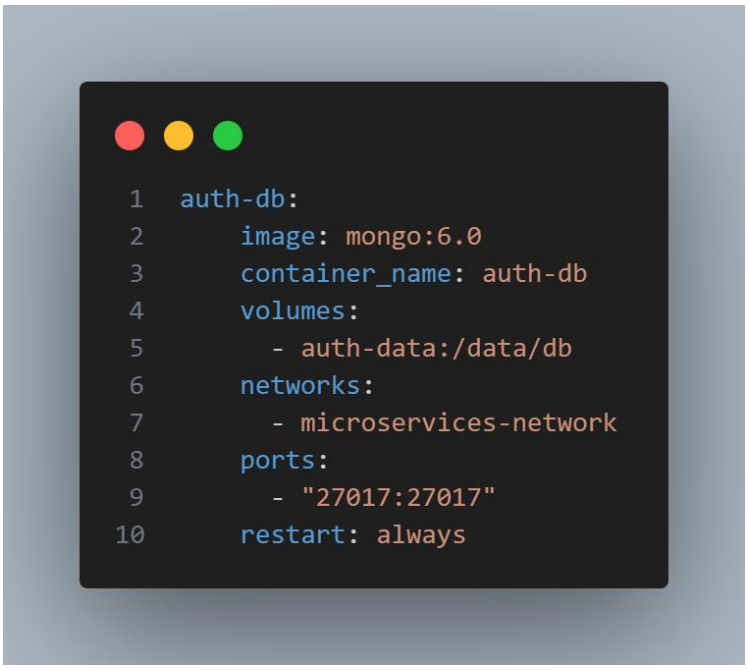


```yaml
auth-service:
    build: ./auth-service
    container_name: auth-service
    depends_on:
      - auth-db
    environment:
      - NODE_ENV=development
      - PORT=3001
      - MONGO_URI=mongodb://auth-db:27017/auth
      - JWT_SECRET=8f9a2b5e7c3d1f4a6b8e9c2d5f7a1b3e4c6d8f9a2b5e7c3d1f4a6b8e9c2d5f7a
    networks:
      - microservices-network
    ports:
      - "3001:3001"
    volumes:
      - ./auth-service:/app
      - /app/node_modules
    deploy:
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
```

Gambar 2. 2 Konfigurasi Auth Service menggunakan Node.js pada Docker Compose

Konfigurasi Auth Service menggunakan Node.js ditunjukkan pada Gambar 2.2. Layanan autentikasi dibangun menggunakan Node.js dan bertugas menangani proses login serta pembuatan token identitas digital (JWT). Service ini berjalan di port 3001 dan memiliki beberapa variabel lingkungan seperti NODE_ENV, PORT, MONGO_URI, dan JWT_SECRET untuk mengatur mode pengembangan, koneksi ke database MongoDB, serta keamanan token. Container ini bergantung pada layanan auth-db agar dapat menyimpan dan mengambil data akun pengguna. Selain itu, konfigurasi volume digunakan untuk menghubungkan folder lokal ke dalam container, layanan ini dibatasi menggunakan CPU sebesar setengah core dan RAM maksimal 512 MB.

2.1.3 Auth Database (MongoDB)



Gambar 2. 3 Konfigurasi Auth Database menggunakan MongoDB pada Docker Compose

Konfigurasi Auth Database menggunakan MongoDB ditunjukkan pada Gambar 2.3. Layanan database untuk autentikasi menggunakan image mongo:6.0 dan berfungsi sebagai tempat penyimpanan data akun pengguna. Data disimpan di volume bernama auth-data agar tetap persisten meskipun container dihentikan. Port yang digunakan adalah 27017, yaitu port default MongoDB. Konfigurasi restart: always memastikan bahwa layanan ini akan otomatis aktif kembali jika terjadi gangguan atau container berhenti secara tidak sengaja.

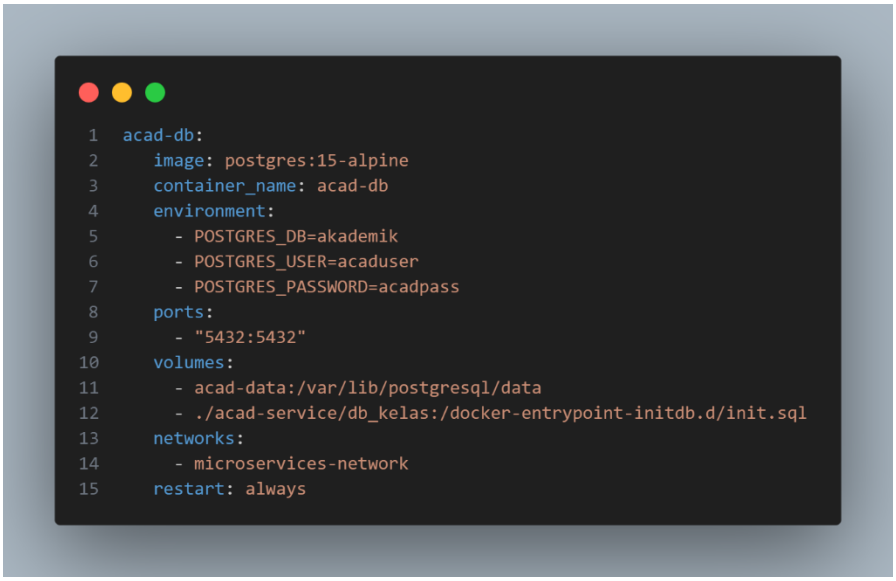
2.1.4 Acad Service (API Akademik)

```yaml
acad-service:
    build: ./acad-service
    container_name: acad-service
    depends_on:
     - acad-db
    environment:
     - DB_HOST=acad-db
     - DB_PORT=5432
     - DB_NAME=acad_db
     - DB_USER=acaduser
     - DB_PASSWORD=acadpass
    networks:
     - microservices-network
    ports:
     - "3002:3002"
    volumes:
     - ./acad-service:/app
```

Gambar 2. 4 Konfigurasi Acad Service (API Akademik) pada Docker Compose

Konfigurasi Acad Service (API Akademik) ditunjukkan pada Gambar 2.4. Layanan ini bertugas untuk menghitung Indeks Prestasi Semester (IPS) mahasiswa. Acad Service ini dibangun dari folder ./acad-service dan berjalan di port 3002. Ia bergantung pada layanan acad-db sebagai sumber data. Variabel lingkungan seperti DB_HOST, DB_PORT, DB_NAME, DB_USER, dan DB_PASSWORD digunakan untuk mengatur koneksi ke database PostgreSQL. Volume lokal juga dipasang ke dalam container agar kode aplikasi dapat diakses dan dijalankan dengan baik.

2.1.5 Acad Database (PostgreSQL)

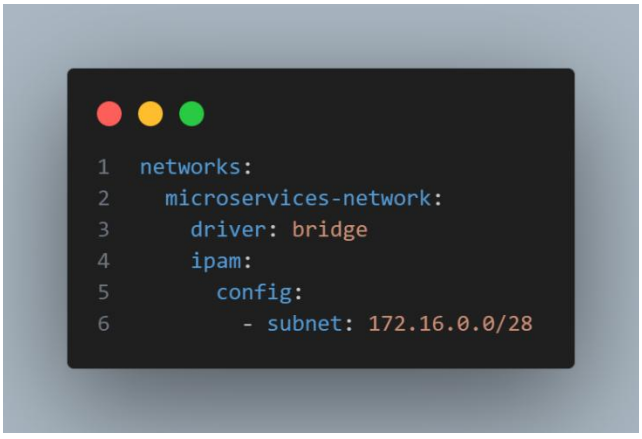


```yaml
acad-db:
  image: postgres:15-alpine
  container_name: acad-db
  environment:
    - POSTGRES_DB=akademik
    - POSTGRES_USER=acaduser
    - POSTGRES_PASSWORD=acadpass
  ports:
    - "5432:5432"
  volumes:
    - acad-data:/var/lib/postgresql/data
    - ./acad-service/db_kelas:/docker-entrypoint-initdb.d/init.sql
  networks:
    - microservices-network
  restart: always
```

Gambar 2. 5 Konfigurasi Acad Database menggunakan PostgreSQL pada Docker Compose

Konfigurasi Acad Database menggunakan PostgreSQL ditunjukkan pada Gambar 2.5. Database ini berfungsi untuk menyimpan data akademik mahasiswa. Layanan database akademik menggunakan image postgres:15-alpine dan berfungsi menyimpan seluruh data akademik. Database ini diinisialisasi dengan file db_kelas.sql yang otomatis dijalankan saat container pertama kali dibuat. Data disimpan di volume acad-data agar tetap aman dan tidak hilang saat container dimatikan. Port yang digunakan adalah 5432, dan konfigurasi restart: always memastikan layanan ini tetap aktif secara otomatis jika terjadi gangguan.

2.1.6 Networks



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

Gambar 2. 6 Konfigurasi Jaringan Microservices-network

Konfigurasi Networks ditunjukkan pada Gambar 2.6. Layanan ini berfungsiAgar semua layanan bisa saling berkomunikasi, sistem ini menggunakan sebuah jaringan khusus bernama microservices-network. Jaringan tersebut dibuat

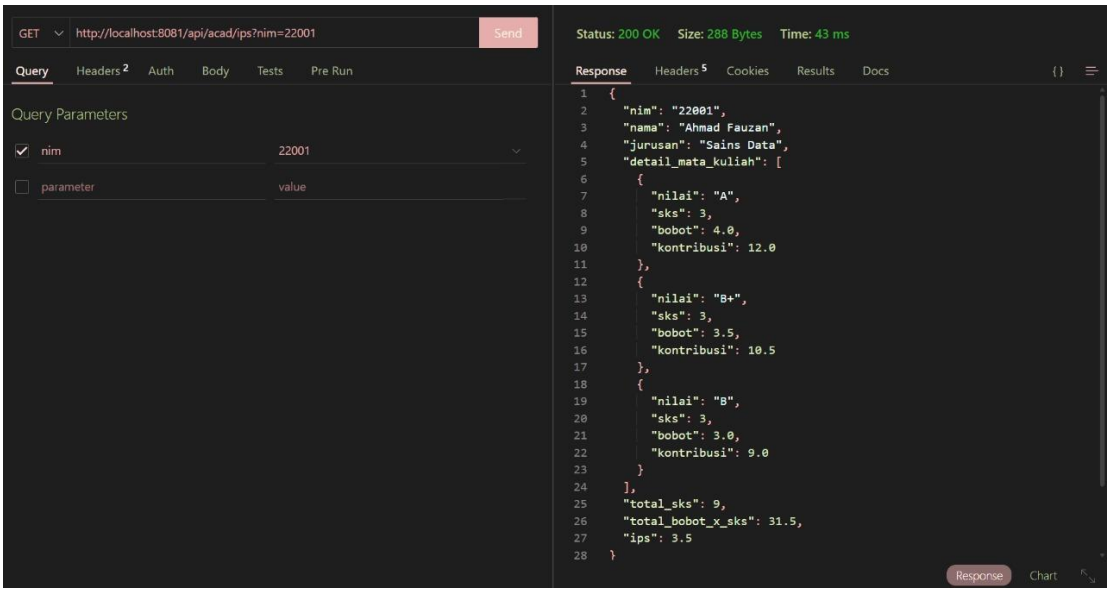
dengan driver *bridge*, sehingga setiap container memiliki alamat IP sendiri namun tetap berada dalam satu ruang komunikasi yang aman. Subnet 172.16.0.0/28 dipakai untuk memastikan setiap layanan mendapat alamat unik dan tidak saling bertabrakan. Dengan cara ini, API Gateway, Auth Service, dan Acad Service bisa terhubung tanpa masalah.

2.1.7 Volume

Gambar 2. 7 Konfigurasi Volume Data untuk Setiap Layanan

Untuk menjaga agar data penting dari setiap layanan mikro tetap aman dan tidak hilang meskipun *container* mengalami perubahan atau penghapusan, sistem ini menggunakan named volumes. Seperti yang ditunjukkan pada Gambar 2.7, setiap layanan memiliki tempat penyimpanan datanya sendiri, seperti *volume* auth-data untuk semua yang berkaitan dengan autentikasi pengguna, product-data dan order-data untuk data produk dan transaksi, serta acad-data untuk informasi akademik. Dengan memisahkan data menggunakan *named volumes*, kita memastikan bahwa data tersebut persisten—artinya data tidak akan hilang—dan kinerjanya lebih optimal saat diakses oleh *container*. Cara ini adalah praktik terbaik untuk menjamin bahwa semua data kritis sistem tetap tersedia setiap saat.

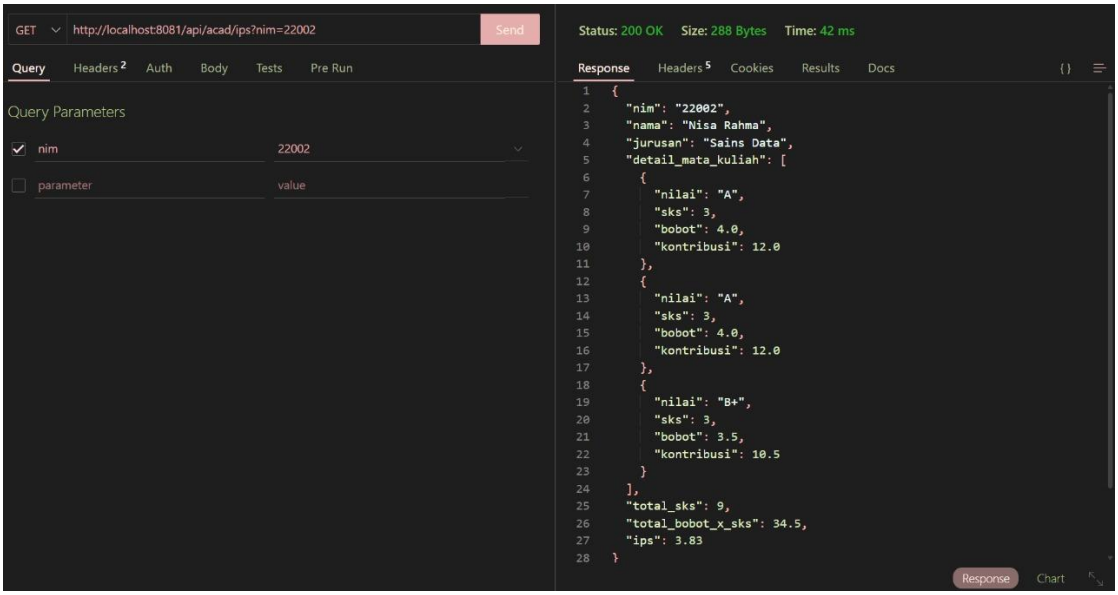
2.2 Implementasi API IPS Mahasiswa



Gambar 2. 8 Respon NIM 22001

Berdasarkan gambar yang ditampilkan, dilakukan pengujian terhadap endpoint API GET /api/acad/ips menggunakan parameter (http://localhost:8081/api/acad/ips?nim=22001) melalui aplikasi pengujian API (seperti Postman). Endpoint ini berfungsi untuk menampilkan informasi akademik mahasiswa sekaligus menghitung Indeks Prestasi Semester (IPS) berdasarkan data nilai mata kuliah yang diambil.

Hasil respons menunjukkan data mahasiswa dengan NIM 22001 atas nama Ahmad Fauzan dari jurusan Sains Data. Data yang ditampilkan meliputi daftar mata kuliah beserta nilai huruf, jumlah SKS, bobot nilai, dan kontribusi bobot terhadap perhitungan IPS. Dari hasil perhitungan sistem, diperoleh total SKS sebesar 9, total bobot SKS sebesar 31,5, dan nilai IPS sebesar 3,5. Status respons HTTP yang diterima adalah 200 OK, yang menandakan bahwa permintaan berhasil diproses oleh server.



Gambar 2. 9 Respon NIM 22002

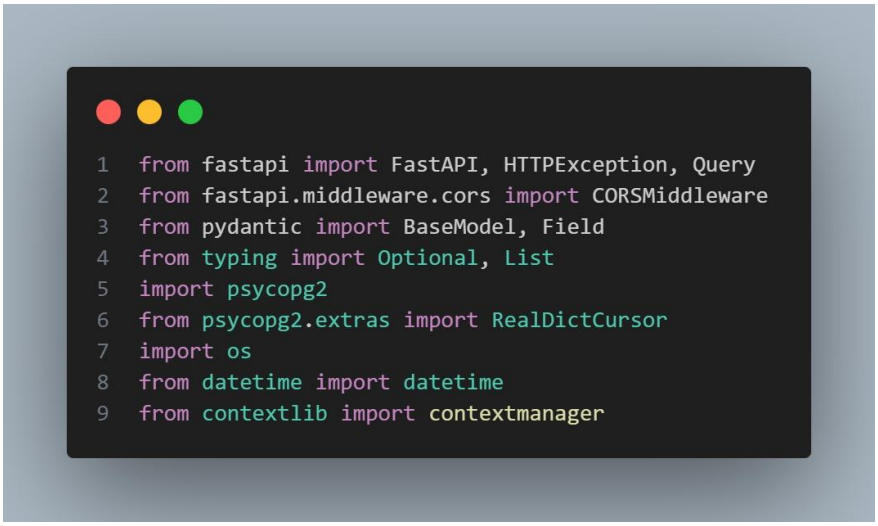
Pada pengujian ini, sistem menampilkan data mahasiswa dengan NIM 22002 (http://localhost:8081/api/acad/ips?nim=22002) atas nama Nisa Rahma, juga dari jurusan Sains Data. Respons API menampilkan detail nilai mata kuliah yang terdiri dari beberapa nilai huruf seperti A dan B+, beserta jumlah SKS dan bobot masing-masing. Berdasarkan hasil perhitungan, diperoleh total SKS sebesar 9, total bobot SKS sebesar 34,5, dan nilai IPS sebesar 3,83. Sama seperti pengujian sebelumnya, respons HTTP menunjukkan status 200 OK, yang menandakan bahwa endpoint berfungsi dengan baik.

Secara keseluruhan, hasil pengujian menunjukkan bahwa endpoint API GET /api/acad/ips telah berhasil menerima parameter nim, memproses data akademik mahasiswa, serta menghitung nilai IPS secara otomatis dan akurat. Hal ini membuktikan bahwa layanan API yang dikembangkan telah berjalan sesuai dengan kebutuhan sistem akademik dan siap untuk digunakan atau dideploy menggunakan teknologi Docker.

2.3 Penjelasan File main.py

File main.py merupakan file utama yang menjalankan layanan akademik. Di dalamnya terdapat konfigurasi server dan endpoint untuk menghitung IPS mahasiswa. File ini menerima data nilai dalam format JSON, mengonversi nilai huruf menjadi angka mutu, menghitung bobot SKS, lalu menghasilkan IPS. File main.py dijalankan dalam container acad-service melalui port 3002, sehingga dapat diakses melalui API Gateway.

2.3.1 Library



```
from fastapi import FastAPI, HTTPException, Query
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel, Field
from typing import Optional, List
import psycopg2
from psycopg2.extras import RealDictCursor
import os
from datetime import datetime
from contextlib import contextmanager
```

Gambar 2. 10 Konfigurasi Library

Bagian awal kode ini berfungsi sebagai pondasi utama aplikasi. Pertama, digunakan FastAPI sebagai kerangka kerja untuk membangun layanan API, dengan tambahan HTTPException untuk menangani kesalahan dan Query untuk mendefinisikan parameter pada endpoint. Agar API dapat diakses dari berbagai domain, ditambahkan CORS Middleware yang membuka izin akses lintas origin.

Selanjutnya, Pydantic dimanfaatkan melalui BaseModel dan Field untuk mendefinisikan serta memvalidasi struktur data, misalnya data mahasiswa. Modul typing (Optional, List) digunakan untuk mendukung tipe data yang lebih kompleks. Untuk koneksi ke database PostgreSQL, dipakai psycopg2 sebagai driver, dengan RealDictCursor agar hasil query lebih mudah diolah dalam bentuk dictionary.

Selain itu, modul os digunakan untuk membaca environment variable sehingga konfigurasi database bisa diatur dari luar kode. Modul datetime dipakai untuk menangani waktu, misalnya menampilkan timestamp pada health check. Terakhir, contextmanager dari contextlib digunakan untuk mengatur koneksi database agar aman dibuka dan ditutup secara otomatis, bahkan jika terjadi error.

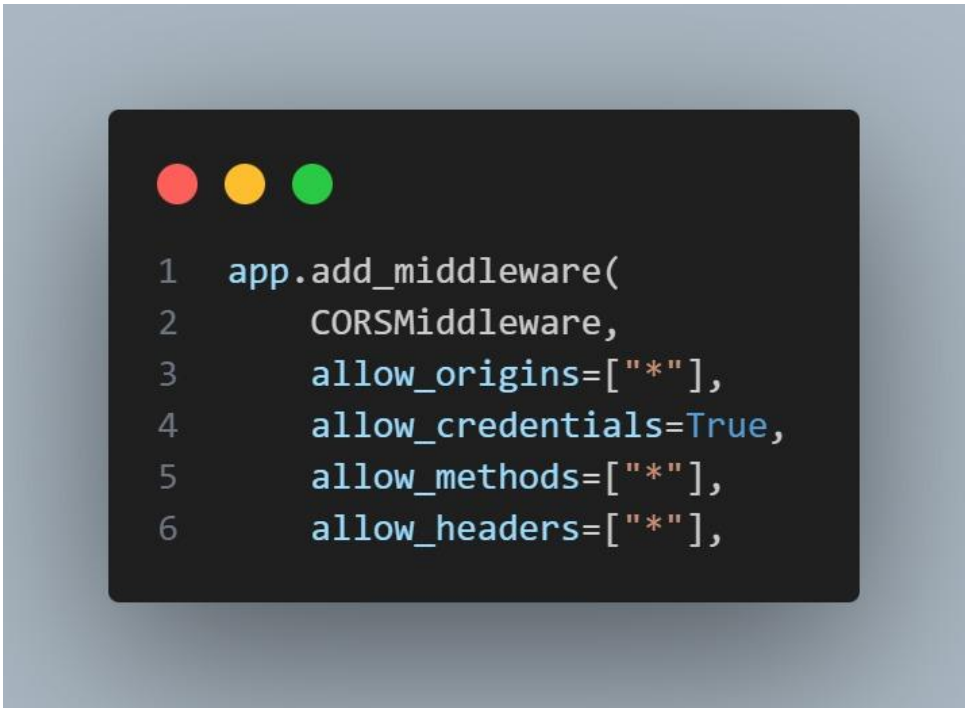
Secara keseluruhan, bagian ini menyiapkan semua kebutuhan dasar: framework API, validasi data, akses database, konfigurasi sistem, serta manajemen koneksi. Dengan fondasi ini, aplikasi siap dikembangkan menjadi layanan akademik digital yang rapi dan terstruktur.

2.3.2 Inisialisasi Aplikasi FastAPI

Gambar 2. 11 Konfigurasi Aplikasi FastAPI

Bagian ini mendefinisikan aplikasi utama dengan menggunakan framework FastAPI. Aplikasi diberi nama Product Service dengan versi 1.0.0. Inisialisasi ini berfungsi sebagai titik awal agar seluruh layanan API dapat dijalankan dan dikenali identitasnya.

2.3.3 Middleware CORS



Gambar 2. 12 Konfigurasi Middleware CORS

Konfigurasi CORS Middleware ditambahkan untuk mengatur akses dari berbagai domain. Dengan pengaturan ini, API dapat menerima permintaan dari semua asal (origin), metode, dan header. Hal ini penting agar layanan bisa digunakan oleh aplikasi web atau mobile tanpa terhalang oleh kebijakan keamanan browser.

2.3.4 Konfigurasi Database

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

def row_to_dict(row):
    if row is None:
        return None
    return dict(row)
```

Gambar 2. 13 Konfigurasi Database

Detail koneksi ke database PostgreSQL disimpan dalam variabel DB_CONFIG. Informasi seperti host, port, nama database, user, dan password diambil dari environment variable. Jika tidak tersedia, sistem menggunakan nilai default. Bagian ini memastikan aplikasi dapat terhubung dengan database untuk menyimpan dan mengambil data.

2.3.5 Model Mahasiswa

```
class Mahasiswa(BaseModel):
    nim: str
    nama: str
    jurusan: str
    angkatan: int = Field(ge=0)
```

Gambar 2. 14 Konfigruasi Model Mahasiswa

Model Mahasiswa dibuat menggunakan Pydantic. Model ini mendefinisikan struktur data mahasiswa dengan atribut nim, nama, jurusan, dan angkatan. Validasi otomatis dilakukan, misalnya nilai angkatan harus berupa angka positif. Model ini membantu menjaga konsistensi data yang masuk ke sistem.

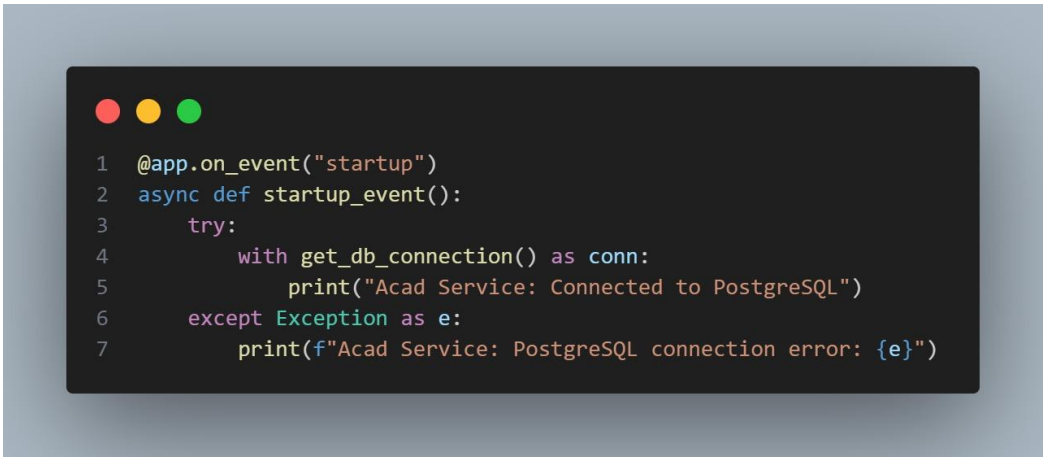
2.3.6 Context Manager Koneksi Database

```python
@contextmanager
def get_db_connection():
    conn = psycopg2.connect(**DB_CONFIG)
    try:
        yield conn
        conn.commit()
    except Exception as e:
        conn.rollback()
        raise e
    finally:
        conn.close()
```

Gambar 2. 15 Konfigurasi Manager Koneksi Database

Fungsi get_db_connection() menggunakan context manager untuk membuka dan menutup koneksi ke database. Jika transaksi berhasil, perubahan akan disimpan (commit). Jika terjadi error, transaksi dibatalkan (rollback). Dengan cara ini, integritas data tetap terjaga.

2.3.7 Event Startup



```python
@app.on_event("startup")
async def startup_event():
    try:
        with get_db_connection() as conn:
            print("Acad Service: Connected to PostgreSQL")
    except Exception as e:
        print(f"Acad Service: PostgreSQL connection error: {e}")
```

Gambar 2. 16 Konfigurasi Event Startup

Event startup_event dijalankan saat aplikasi pertama kali aktif. Bagian ini mencoba membuka koneksi ke PostgreSQL. Jika berhasil, sistem menampilkan pesan "Acad Service: Connected to PostgreSQL". Jika gagal, pesan error ditampilkan. Event ini berfungsi sebagai pengecekan awal kesehatan sistem.

2.3.8 Endpoint Health Check

```python
@app.get("/health")
async def health_check():
    return {
        "status": "Acad Service is running",
        "timestamp": datetime.now().isoformat()
    }
```

Gambar 2. 17 Konfigurasi Health Check

Endpoint /health digunakan untuk memeriksa status layanan. Ketika dipanggil, sistem mengembalikan status "Acad Service is running" beserta timestamp. Endpoint ini berguna sebagai indikator bahwa layanan berjalan normal.

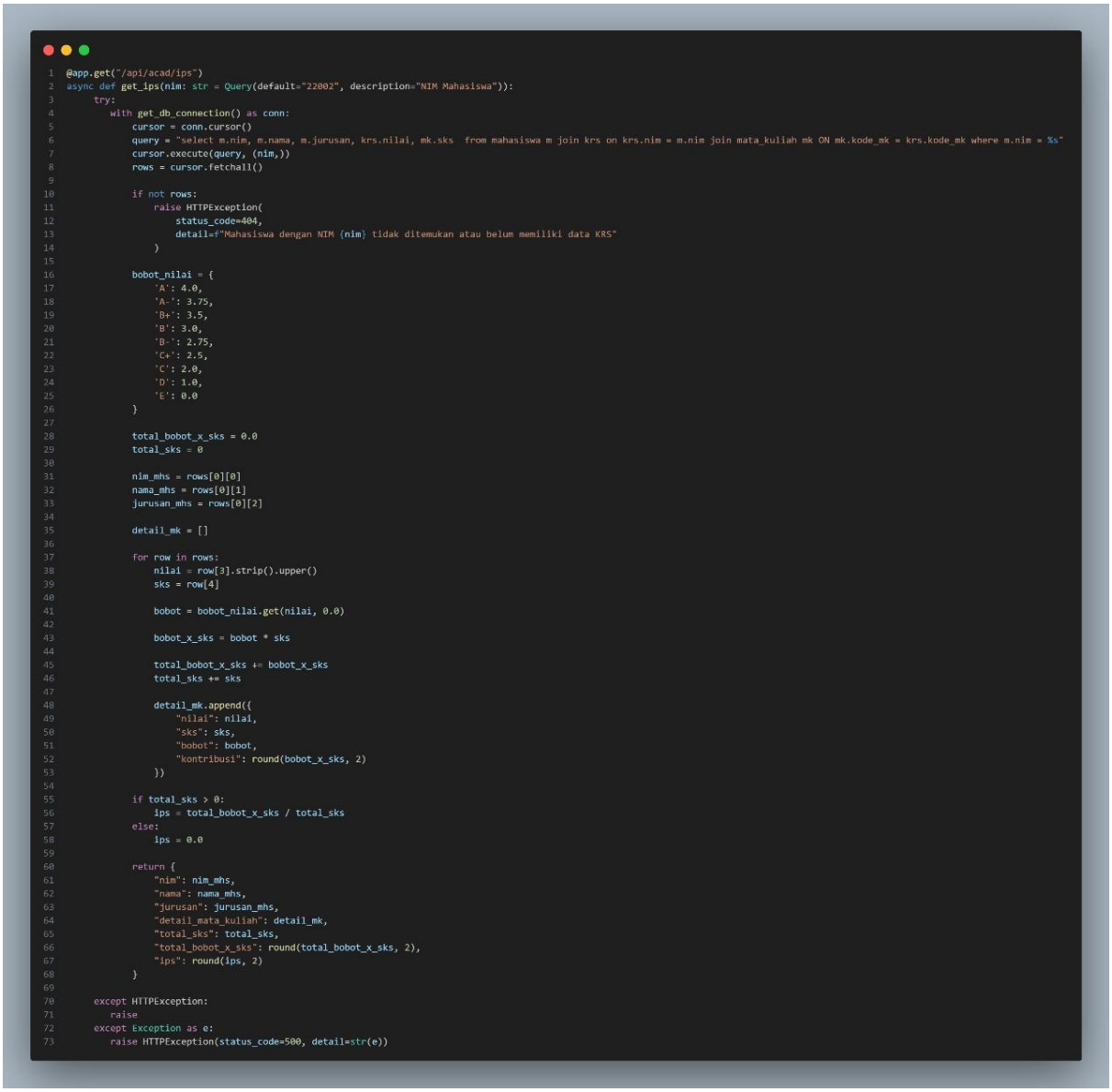
2.3.9 Endpoint Data Mahasiswa

```python
@app.get("/api/acad/mahasiswa")
async def get_mahasiswas():
    try:
        with get_db_connection() as conn:
            cursor = conn.cursor()
            query = "SELECT * FROM mahasiswa"
            cursor.execute(query)
            rows = cursor.fetchall()
            return [{"nim": row[0], "nama": row[1], "jurusan": row[2], "angkatan": row[3]} for row in rows]
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Gambar 2. 18 Konfigurasi Endpoint Data Mahasiswa

Endpoint /api/acad/mahasiswa berfungsi mengambil daftar mahasiswa dari tabel mahasiswa. Query SELECT * FROM mahasiswa dijalankan, lalu hasilnya dikembalikan dalam format JSON berisi NIM, nama, jurusan, dan angkatan. Jika terjadi error, sistem mengembalikan pesan kesalahan dengan status 500.

2.3.10 Endpoint Perhitungan IPS



```python
@app.get("/api/acad/ips")
async def get_ips(nim: str = Query(default="22002", description="NIM Mahasiswa")):
    try:
        with get_db_connection() as conn:
            cursor = conn.cursor()
            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 = %s"
            cursor.execute(query, (nim,))
            rows = cursor.fetchall()

            if not rows:
                raise HTTPException(
                    status_code=404,
                    detail=f"Mahasiswa dengan NIM {nim} tidak ditemukan atau belum memiliki data KRS"
                )

            bobot_nilai = {
                'A': 4.0,
                'A-': 3.75,
                'B+': 3.5,
                'B': 3.0,
                'B-': 2.75,
                'C+': 2.5,
                'C': 2.0,
                'D': 1.0,
                'E': 0.0
            }

            total_bobot_x_sks = 0.0
            total_sks = 0

            nim_mhs = rows[0][0]
            nama_mhs = rows[0][1]
            jurusan_mhs = rows[0][2]

            detail_mk = []

            for row in rows:
                nilai = row[3].strip().upper()
                sks = row[4]

                bobot = bobot_nilai.get(nilai, 0.0)

                bobot_x_sks = bobot * sks

                total_bobot_x_sks += bobot_x_sks
                total_sks += sks

                detail_mk.append({
                    "nilai": nilai,
                    "sks": sks,
                    "bobot": bobot,
                    "kontribusi": round(bobot_x_sks, 2)
                })

            if total_sks > 0:
                ips = total_bobot_x_sks / total_sks
            else:
                ips = 0.0

            return {
                "nim": nim_mhs,
                "nama": nama_mhs,
                "jurusan": jurusan_mhs,
                "detail_mata_kuliah": detail_mk,
                "total_sks": total_sks,
                "total_bobot_x_sks": round(total_bobot_x_sks, 2),
                "ips": round(ips, 2)
            }

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

Gambar 2. 19 Konfigurasi Endpoint Perhitungan IPS

Endpoint /api/acad/ips dibuat untuk menghitung Indeks Prestasi Semester (IPS) seorang mahasiswa berdasarkan NIM yang diberikan. Ketika endpoint ini dipanggil, sistem akan membuka koneksi ke database dan menjalankan query untuk mengambil data mahasiswa beserta nilai KRS dan jumlah SKS dari tabel mata kuliah.

Jika data mahasiswa tidak ditemukan, sistem akan mengembalikan pesan kesalahan dengan status 404. Namun, jika data tersedia, setiap nilai huruf (misalnya A, B+, C) akan dikonversi menjadi bobot angka sesuai aturan yang berlaku. Bobot tersebut kemudian dikalikan dengan jumlah SKS masing-masing mata kuliah untuk mendapatkan kontribusi

nilai. Semua kontribusi dijumlahkan, lalu dibagi dengan total SKS yang diambil mahasiswa.

Hasil perhitungan ini menghasilkan IPS, yang kemudian dikembalikan dalam format JSON. Data yang ditampilkan mencakup identitas mahasiswa (NIM, nama, jurusan), detail mata kuliah beserta nilai dan bobotnya, total SKS, total bobot, serta IPS akhir. Dengan cara ini, endpoint tidak hanya memberikan angka IPS, tetapi juga transparansi mengenai bagaimana nilai tersebut dihitung.

BAB III
KESIMPULAN

Berdasarkan hasil perancangan, implementasi, dan pengujian sistem yang telah dilakukan pada proyek ini, dapat ditarik beberapa kesimpulan yang menjawab rumusan masalah pada Bab I sebagai berikut.

Pertama, konfigurasi docker-compose.yml berhasil dilakukan sehingga seluruh sub-layanan dalam sistem dapat berjalan secara terintegrasi. Melalui Docker Compose, layanan API Gateway (Nginx), Auth Service (Node.js), Acad Service (FastAPI), serta database MongoDB dan PostgreSQL dapat dijalankan secara bersamaan dalam container yang terisolasi namun tetap saling terhubung melalui jaringan khusus (microservices-network). Penggunaan depends_on, pengaturan port, environment variable, volume, dan network menunjukkan bahwa Docker Compose efektif dalam mengelola sistem berbasis multi-container secara terstruktur dan efisien.

Kedua, implementasi pemrograman dasar pada Application Programming Interface (API) service untuk perhitungan Indeks Prestasi Semester (IPS) mahasiswa telah berhasil dilakukan. Melalui endpoint GET /api/acad/ips, sistem mampu menerima parameter NIM, mengambil data akademik mahasiswa dari database PostgreSQL, mengonversi nilai huruf menjadi bobot angka, menghitung total bobot dan total SKS, serta menghasilkan nilai IPS secara otomatis dan akurat. Hasil pengujian menunjukkan bahwa API dapat memberikan respons yang benar dengan status HTTP 200 OK, sehingga fungsionalitas perhitungan IPS berjalan sesuai dengan tujuan yang ditetapkan.

Ketiga, penerapan konsep microservice dan containerization pada sistem legacy berbasis API telah berhasil direalisasikan. Setiap layanan dijalankan dalam container terpisah dengan tanggung jawab masing-masing, sehingga sistem menjadi lebih modular, fleksibel, dan mudah dikembangkan. Dengan pendekatan ini, perubahan atau pengembangan pada satu layanan tidak memengaruhi layanan lain secara langsung. Hal ini membuktikan bahwa teknologi Docker dan arsitektur microservice dapat diterapkan secara efektif untuk memodernisasi sistem legacy agar lebih sesuai dengan kebutuhan sistem terdistribusi di dunia industri saat ini.

Secara keseluruhan, proyek ini berhasil mencapai tujuan pembelajaran dalam mata kuliah Arsitektur Komputer dan Sistem Operasi, khususnya dalam memahami konsep containerization, microservice, serta implementasi API dalam lingkungan terdistribusi. Selain

meningkatkan pemahaman teknis, proyek ini juga memberikan gambaran nyata mengenai peran DevOps dalam pengelolaan dan deployment sistem perangkat lunak modern.