

Laporan Tugas 2 Cloud Computing

Nathaniel Ryo Kurniadi – 5025221019

Link Repository: <https://github.com/rororyo/komputasi-awan-2025/tree/main/Tugas%202>

Pendahuluan

Tugas 2 bertujuan untuk memperkenalkan mahasiswa ke Docker Compose dengan fokus pada konfigurasi dan pengelolaan berbagai skenario layanan. Docker Compose sendiri merupakan alat yang digunakan untuk mendefinisikan dan menjalankan aplikasi multi-container secara efisien melalui satu berkas konfigurasi YAML, sehingga memudahkan proses orkestrasi, otomatisasi, dan replikasi lingkungan pengembangan maupun produksi. Dalam tugas ini, diminta untuk menjalankan empat kasus berbeda menggunakan Docker Compose, mendokumentasikan langkah-langkah implementasi, dan memberikan penjelasan detail terkait proses dari repositori <https://github.com/rm77/cloud2023/tree/master/containers/compose/compose>. Selain itu, juga ditugaskan untuk mengembangkan satu kasus tambahan (Case 5) berdasarkan kreativitas masing-masing, yang mencakup desain arsitektur, skrip pendukung, serta dokumentasi hasil implementasi.

Deskripsi Tugas

Tugas ini terdiri dari empat kasus yang perlu dijalankan menggunakan Docker Compose. Untuk penjelasan tiap kasusnya adalah sebagai berikut:

- **Case 1:** Menjalankan web server Nginx dengan protokol HTTP
- **Case 2:** Menjalankan web server Nginx dengan protokol HTTPS menggunakan sertifikat TLS yang sudah disediakan
- **Case 3:** Menjalankan WordPress menggunakan web server Nginx dengan protokol HTTPS
- **Case 4:** Menjalankan aplikasi PHP dan phpMyAdmin dalam container terpisah dan menghubungkannya untuk pengelolaan database.
- **Case 5 (variasi pribadi):** Menjalankan aplikasi backend sederhana menggunakan node.js dan terkoneksi ke MongoDB sebagai database.

Arsitektur dan Script yang Digunakan

Case 1

1. Deskripsi

Pada Case ini, kita diminta untuk menjalankan web server menggunakan Nginx dengan konfigurasi yang telah disediakan pada *docker-compose.yml*. Ketika berhasil, web server akan menampilkan template Bootstrap yang dapat diakses dengan HTTP

2. Langkah-langkah

Karena *docker-compose.yml* telah disediakan, maka cukup masuk ke dalam direktori dimana file tersebut berada dan menjalankan perintah **docker-compose up -d**

3. Penjelasan

Dengan menggunakan *docker-compose.yml*, kita dapat membuat sebuah container yang di dalamnya terdapat image nginx untuk menjalankan webserver, lalu di port ke port 9999 di host. Sehingga, webserver template Bootstrap, dapat di akses pada <http://localhost:9999>

Case 2

1. Deskripsi

Hampir sama dengan Case 1, pada Case ini kita diminta untuk menjalankan web server, namun dengan HTTPS untuk menyajikan template bootstrap.'

2. Langkah-langkah

Untuk menjalankan konfigurasi docker yang telah diberikan, cukup masuk ke direktori *case2*, dan jalankan perintah **docker-compose up -d**, maka docker compose akan menjalankan semua konfigurasi yang ada di *docker-compose.yml*. Setelah itu, bisa gunakan **docker ps** untuk melihat apakah container berhasil dijalankan.

3. Penjelasan

Bila container berhasil dijalankan, maka layanan web nginx dapat di akses di <https://localhost> untuk menampilkan beberapa template bootstrap.

Case 3

1. Deskripsi

Pada Case ini, kita diminta untuk menyediakan layanan WordPress dan phpMyAdmin di 2 container berbeda. Layanan dapat diakses dengan protokol HTTPS sehingga koneksi lebih aman.

2. Langkah-langkah

Untuk menjalankan konfigurasi, cukup ke direktori *case3* dan jalankan perintah **docker-compose up -d**, karena konfigurasi dan file-file relevan juga sudah disediakan di dalam direktori tersebut.

3. Penjelasan

Ketika kedua container tersebut berhasil dijalankan, maka phpMyAdmin akan berjalan di port 30081 (<http://localhost:30081>) dan WordPress akan berjalan di port https pada umumnya (443 atau 80), namun untuk mengakses cukup dengan menggunakan <https://localhost>.

Case 4

1. Deskripsi

Pada Case 4, kita diminta untuk menyediakan layanan aplikasi PHP dan phpMyAdmin di 2 container berbeda. Lalu, semua dependensi aplikasi PHP tersebut dikelola dengan Docker.

2. Langkah-langkah

Karena file-file relevan dan *docker-compose.yml* sudah disediakan, maka cukup jalankan perintah **docker-compose up -d** untuk menjalankan konfigurasi yang ada di yml tersebut.

3. Penjelasan

Ketika kedua container tersebut berhasil dijalankan, maka aplikasi PHP dapat di akses dengan <http://localhost:34001>, dan phpMyAdmin dapat di akses dengan <http://localhost:10000>.

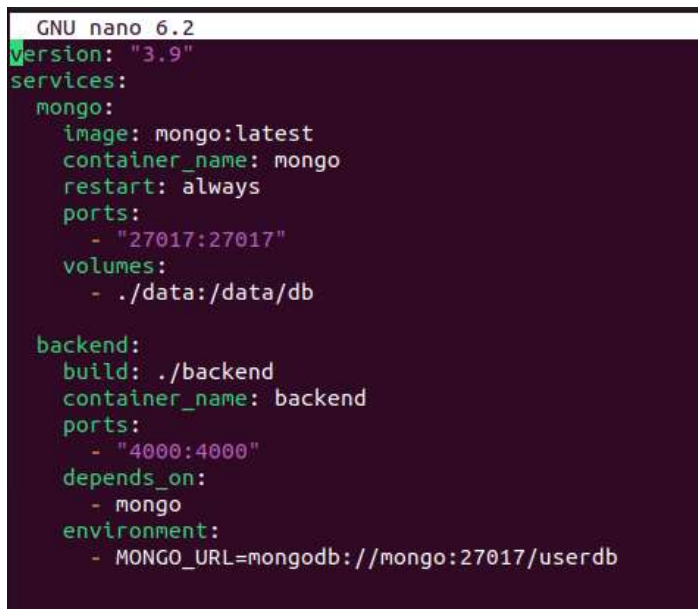
Case 5

1. Deskripsi

Pada case ini, kita akan menyediakan sebuah aplikasi backend sederhana yang dapat melayani request sederhana dengan data-data yang di store di database agar persistent. Terdapat 3 buah endpoint yang dibuat, yaitu:

- GET / : Menampilkan pesan “API is running” untuk mengetahui bahwa aplikasi backend sudah berjalan.
- GET /users : Mengambil data users dari MongoDB.
- POST /users : Menambahkan data user baru ke database.

2. Langkah-langkah



```
GNU nano 6.2
Version: "3.9"
services:
  mongo:
    image: mongo:latest
    container_name: mongo
    restart: always
    ports:
      - "27017:27017"
    volumes:
      - ./data:/data/db

  backend:
    build: ./backend
    container_name: backend
    ports:
      - "4000:4000"
    depends_on:
      - mongo
    environment:
      - MONGO_URL=mongodb://mongo:27017/userdb
```

Buat *docker-compose.yml* seperti gambar di atas, setelah itu jalankan dengan perintah **docker-compose up -d**.

3. Penjelasan

Bila kedua container berhasil di jalankan, maka kita dapat mengakses beberapa endpoint berikut:

- GET / : Menampilkan pesan “API is running” untuk mengetahui bahwa aplikasi backend sudah berjalan.
- GET /users : Mengambil data users dari MongoDB.
- POST /users : Menambahkan data user baru ke database.

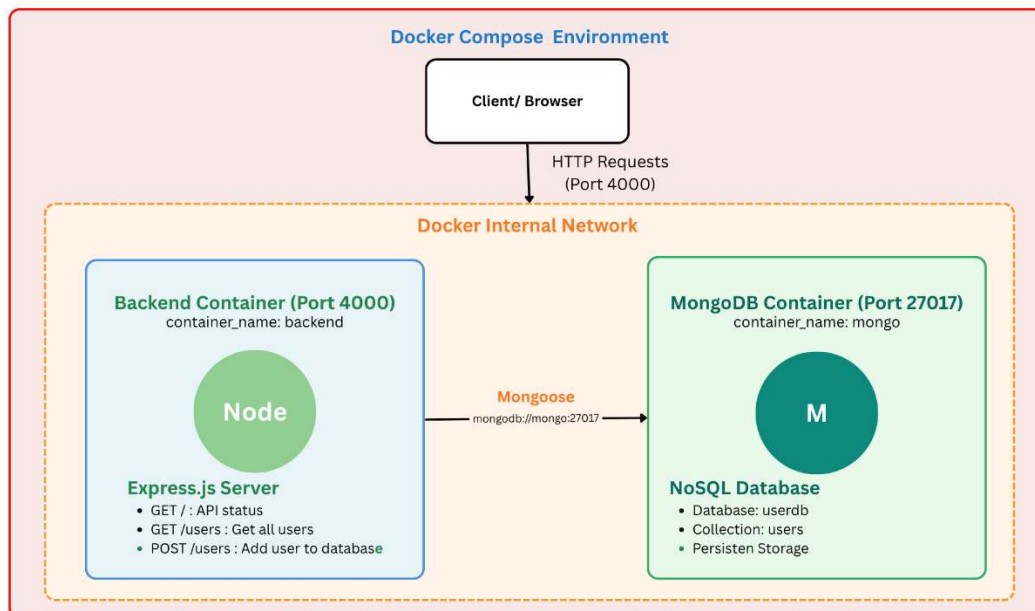
Untuk menambah data user baru, dapat menggunakan curl. Contoh perintahnya adalah sebagai berikut:

```
curl -X POST http://localhost:4000/users \
-H "Content-Type: application/json" \
-d '{"name": "Nathaniel"}'
```

Dengan perintah ini, user dengan nama Nathaniel akan terbuat, dan dapat kita lihat perubahannya di <http://localhost:4000/users>.

4. Arsitektur

Setelah membuat direktori baru bernama *case5*, kita masuk ke direktori tersebut , dan membuat direktori *backend* untuk menyimpan source code aplikasi backend, dan direktori *data* untuk menyimpan data aplikasi. Berikut adalah gambar arsitektur yang digunakan:



Kita membuat 2 container, yang pertama adalah mongodb untuk databasenya, dengan spesifikasi berikut:

- Nama: mongo

- Image: mongo:latest
- Ports: 27017, ini port agar aplikasi backend kita dapat mengakses database, kita harus mengekspos port tersebut karena aplikasi backend berjalan pada container terpisah
- Volumes: kita melakukan mapping direktori *data* pada host ke direktori *data/db* di dalam container

Lalu, untuk container backend sendiri dibuat dengan spesifikasi berikut:

- Nama: backend
- Ports: 4000, port 4000 perlu di ekspos agar dapat diakses oleh host sehingga kita dapat memanggil endpoint yang ada.
- Depends on: mongo, jadi container backend punya dependensi ke mongo, jadi mongo harus sudah up terlebih dahulu
- Environment: variabel environment hanya ada MONGO_URL, yaitu URL yang digunakan backend untuk terkoneksi ke database.

Setelah pembuatan *docker-compose.yml* selesai, maka selanjutnya adalah membuat aplikasi backend.

```
const express = require("express");
const mongoose = require("mongoose");
const app = express();
app.use(express.json());

mongoose.connect("mongodb://mongo:27017/userdb")
  .then(() => console.log("MongoDB connected"))
  .catch(err => console.log(err));

const UserSchema = new mongoose.Schema({ name: String });
const User = mongoose.model("User", UserSchema);

app.get("/", (req, res) => res.send("API is running"));

app.get("/users", async (req, res) => {
  const users = await User.find();
  res.json(users);
});

app.post("/users", async (req, res) => {
  const user = await User.create({ name: req.body.name });
  res.json(user);
});

app.listen(4000, () => console.log("Server running on port 4000"));
```

Pertama, kita melakukan koneksi ke database, lalu membuat schema user dengan key bernama name dan value yang bertipe string. Setelah itu, kita mendaftarkan schema tersebut ke mongoose. Terakhir, kita membuat 3 rute untuk memastikan API berjalan, melihat list users, dan membuat user baru. Aplikasi backend ini berjalan di port 4000.

5. Kapan Skenario Ini Cocok Digunakan?

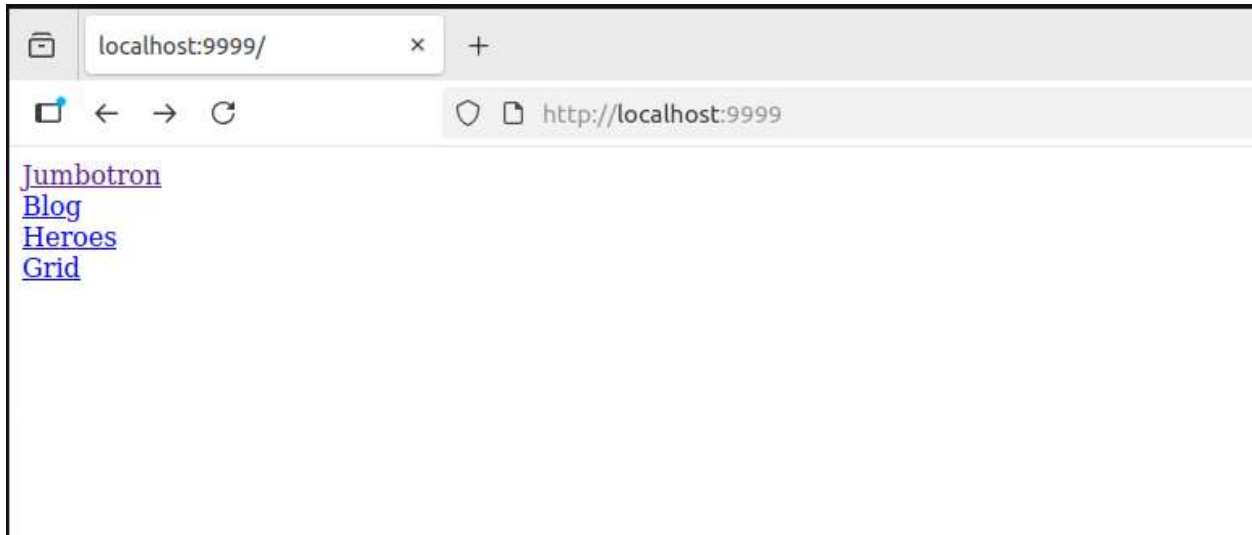
Arsitektur ini cocok digunakan pada beberapa skenario berikut:

- Mengembangkan aplikasi fullstack dengan API dan koneksi database.

- Pengujian otomatis terintegrasi dengan pipeline CI/CD.
- Belajar dan Eksperimen Arsitektur Layanan.

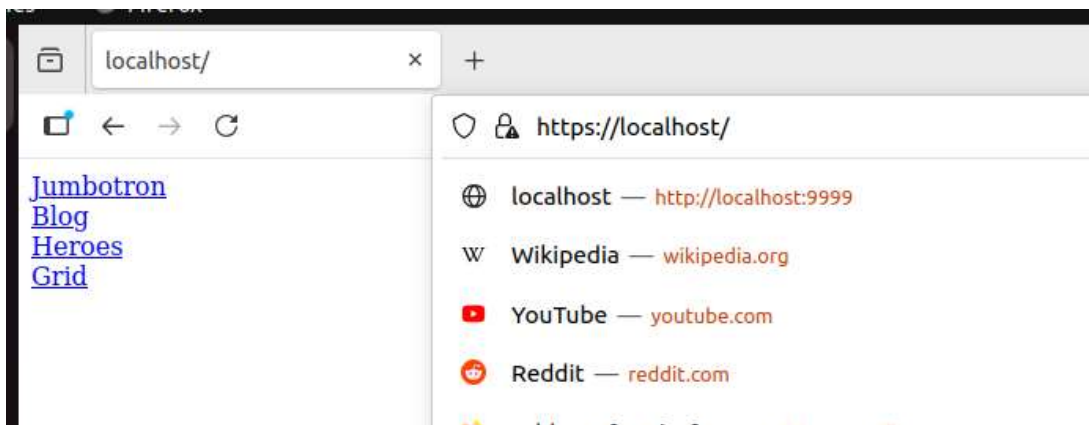
Screenshot dan Penjelasan

Case 1



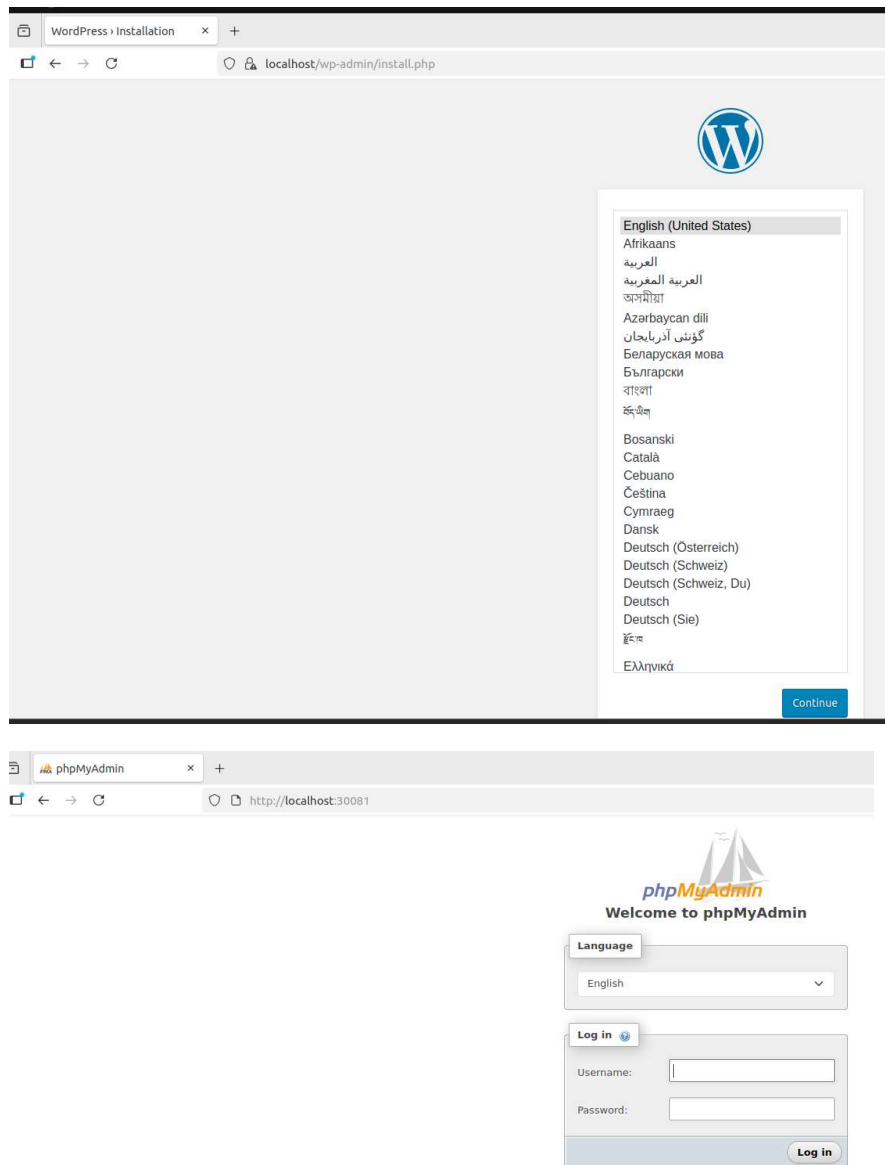
Penjelasan: Gambar di atas merupakan laman web ketika docker container yang dikonfigurasi melalui docker compose berhasil dijalankan. Port 9999 akan menampilkan web server yang berisi template bootstrap, yang telah disediakan dari Case 1, dan sudah di mapping ke dalam container. Untuk mengaksesnya, dapat menggunakan <http://localhost:9999>

Case 2



Penjelasan: Gambar di atas merupakan laman web ketika container berhasil dijalankan. Hampir sama seperti Case 1, namun pada Case ini protokol yang digunakan adalah https dengan memanfaatkan sertifikat di direktori *certs* dan memetakannya ke dalam container. Untuk mengaksesnya, dapat menggunakan <https://localhost>

Case 3



Penjelasan: Kedua gambar di atas adalah tampilan bila kedua container tersebut berhasil dijalankan. Pada container pertama terdapat WordPress yang dapat di akses di <https://localhost>. Bila berhasil, maka akan langsung di arahkan ke <https://localhost/wp-admin/install.php>. Ini adalah tampilan awal bila wordpress pertama kali dijalankan setelah imagenya di install, kita dapat melakukan konfigurasi pada wordpress seperti pada komputer host pada umumnya. Lalu, pada container kedua terdapat phpMyAdmin yang dapat di akses melalui <http://localhost:30081>.

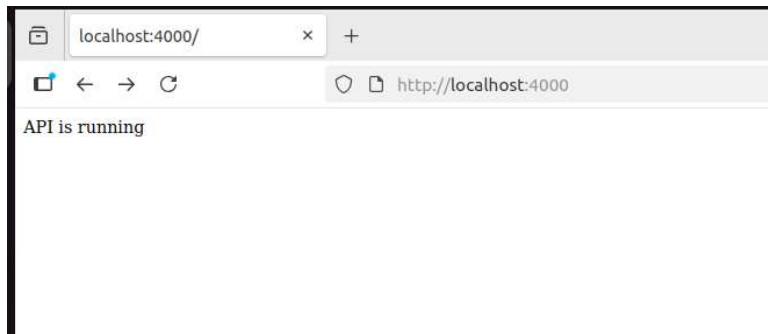
Case 4

The top screenshot shows a web browser window with the address bar displaying `http://localhost:34001`. The page content is the PHP version information page for PHP 7.2.33, showing system details, build date, and configuration command.

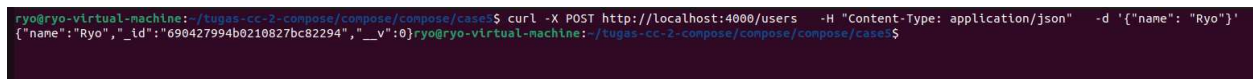
The bottom screenshot shows a web browser window with the address bar displaying `http://localhost:10000/index.php?route=/&route=%2F`. The page content is the phpMyAdmin interface, showing the 'General settings' and 'Appearance settings' tabs. The 'General settings' tab is active, showing the 'Server connection collation' set to 'utf8mb4_unicode_ci'.

Penjelasan: Kedua gambar di atas adalah tampilan bila kedua container berhasil dijalankan. Container pertama terdapat aplikasi PHP yang dapat diakses dengan <http://localhost:34001>. Pada halaman tersebut, kita dapat melihat tampilan layanan PHP beserta dependensinya. Lalu, pada container kedua, terdapat laman phpMyAdmin yang dapat diakses pada <http://localhost:10000>, menunjukkan bahwa database berhasil terkoneksi dan kita dapat masuk dengan password dan user yang telah ditentukan (user= root, password= mydb6789tyui).

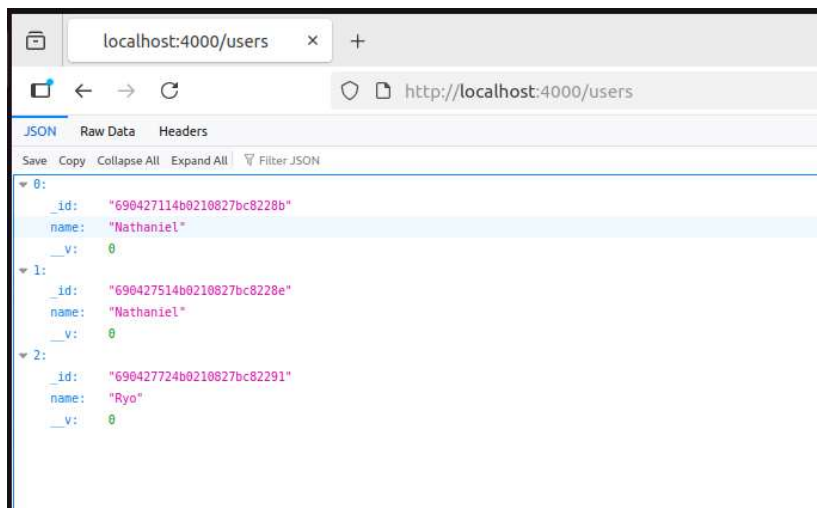
Case 5



Gambar 1. GET /



Gambar 2. POST /users



Gambar 3. GET /users

Penjelasan: Ketiga gambar di atas adalah tampilan dari masing-masing endpoint bila kedua container tersebut berhasil dijalankan. Pada Gambar 1, terdapat tampilan “API is running” untuk memastikan bahwa API yang dibuat telah berjalan. Gambar 2 adalah cara menambahkan data menggunakan curl. Bila berhasil, maka penambahan data dapat dilihat pada Gambar 3, yaitu menunjukkan semua user yang ada.

Kesimpulan

Dari tugas yang diberikan kali ini, mahasiswa dapat mengetahui dasar-dasar orkestrasi menggunakan Docker Compose melalui 4 Case yang diberi + 1 variasi Case. 4 Case pertama memperkenalkan konfigurasi umum layanan berbasis web, mulai dari penyajian konten statis hingga melakukan koneksi ke database. Sementara, pada Case 5 implementasi lebih mendalam dari Case sebelumnya, yaitu mengintegrasikan database ke dalam aplikasi backend, dan

membuat layanan sederhana yang dapat berinteraksi dengan database melalui koneksi string. Secara keseluruhan, mahasiswa belajar bahwa Docker Compose sangat membantu dalam menyederhanakan proses pembuatan, pengujian, dan pengelolaan aplikasi multi-container karena hanya perlu membuat satu berkas konfigurasi YAML.