IMPLEMENTASI SISTEM SMART DISASTER VOLUNTEER BERBASIS REST API MENGGUNAKAN NODE.JS DAN MYSQL

Laporan Perancangan dan Implementasi REST API

Dosen Pengampu:

Shinta Ayuningtias, S.Kom., M.Kom.

Dipersiapkan Oleh:

PIRNI	**: 20240040179**
Neng Sahla Nurul Fauziah	**: 20240040130**
Muhammad Denindra Pratama	**: 20240040290**

PROGRAM STUDI TEKNIK INFORMATIKA

UNIVERSITAS NUSA PUTRA

2025

KATA PENGANTAR

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, kami dapat menyelesaikan laporan yang berjudul **"Implementasi Sistem Smart Disaster Volunteer Berbasis API Menggunakan Node.js dan MySQL"** dengan baik dan tepat waktu.

Penyusunan laporan ini bertujuan untuk memenuhi salah satu tugas pada mata kuliah yang berkaitan dengan pengembangan perangkat lunak dan sistem backend. Selain itu, laporan ini juga disusun sebagai bentuk dokumentasi proses perancangan, pembuatan, serta pengujian sistem **Smart Disaster Volunteer**, yang mencakup pengelolaan data relawan, autentikasi pengguna, serta penerapan keamanan sistem menggunakan **JSON Web Token (JWT)**.

Dalam proses penyusunan laporan dan pengembangan sistem ini, kami memperoleh banyak pengalaman serta pemahaman baru, khususnya dalam penerapan teknologi **Node.js**, **Express.js**, dan **MySQL** dalam membangun layanan Application Programming Interface (API). Kami juga mempelajari pentingnya perancangan struktur folder, manajemen basis data, serta pengujian endpoint sebagai bagian dari pengembangan sistem yang terstruktur dan sistematis.

Kami menyadari bahwa laporan ini masih memiliki keterbatasan dan kemungkinan terdapat kekurangan, baik dari segi penulisan maupun isi pembahasan. Oleh karena itu, kami dengan terbuka menerima kritik dan saran yang bersifat membangun demi penyempurnaan laporan dan peningkatan kualitas pembelajaran di masa yang akan datang.

Akhir kata, kami berharap laporan ini dapat memberikan manfaat bagi pembaca, khususnya mahasiswa, dosen, maupun pihak lain yang tertarik dalam pengembangan sistem informasi dan aplikasi backend. Semoga laporan ini dapat menjadi referensi dan tambahan wawasan dalam memahami implementasi sistem berbasis API secara praktis.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang pesat telah mendorong pemanfaatan sistem berbasis Application Programming Interface (API) dalam berbagai bidang, termasuk dalam pengelolaan informasi dan layanan kemanusiaan. REST API menjadi salah satu arsitektur yang banyak digunakan karena kemudahan integrasi, skalabilitas, serta kemampuannya dalam mendukung pertukaran data secara efisien antar sistem.

Dalam konteks penanggulangan bencana, pengelolaan data relawan, informasi bencana, serta penugasan relawan secara manual sering kali menimbulkan permasalahan seperti keterlambatan informasi, data yang tidak terstruktur, dan sulitnya proses koordinasi. Oleh karena itu, diperlukan sebuah sistem backend yang mampu mengelola data tersebut secara terintegrasi, aman, dan mudah diakses.

Berdasarkan permasalahan tersebut, dikembangkan **Smart Disaster Volunteer System**, yaitu sebuah sistem backend berbasis REST API yang dibangun menggunakan Node.js, Express, dan MySQL. Sistem ini dirancang untuk mendukung pengelolaan data relawan, data bencana, serta penugasan relawan dengan menerapkan autentikasi dan keamanan menggunakan JSON Web Token (JWT) serta integrasi API publik sebagai sumber data pendukung.

1.2 Tujuan

Tujuan dari penyusunan laporan dan pengembangan sistem ini adalah sebagai berikut:

1. Merancang dan mengimplementasikan REST API berbasis Node.js dan Express untuk pengelolaan data relawan dan bencana.
2. Membangun perancangan database relasional menggunakan MySQL dengan minimal empat tabel yang saling berelasi.
3. Mengimplementasikan mekanisme autentikasi dan otorisasi menggunakan JSON Web Token (JWT) untuk meningkatkan keamanan sistem.
4. Mengintegrasikan API publik sebagai sumber data eksternal pendukung sistem.
5. Melakukan pengujian terhadap seluruh endpoint REST API untuk memastikan sistem berjalan sesuai dengan kebutuhan.

6. Menyusun dokumentasi backend API sebagai bentuk laporan dan referensi pengembangan sistem.

1.3 Ruang Lingkup

Agar pembahasan dalam laporan ini terarah dan tidak meluas, maka ruang lingkup pembahasan dibatasi pada hal-hal berikut:

1. Perancangan arsitektur sistem backend berbasis REST API.
2. Perancangan dan implementasi database MySQL yang terdiri dari tabel users, disasters, volunteers, dan assignments.
3. Pengembangan REST API menggunakan Node.js dan Express untuk pengelolaan data pengguna, bencana, relawan, dan penugasan relawan.
4. Penerapan middleware dan autentikasi menggunakan JSON Web Token (JWT).
5. Integrasi API publik untuk memperoleh informasi pendukung terkait bencana.
6. Pengujian endpoint REST API menggunakan Postman.
7. Penyusunan dokumentasi backend API dan laporan hasil implementasi sistem.

BAB II

PEMBAHASAN

2.1 Arsitektur Sistem

Smart Disaster Volunteer System menggunakan arsitektur **client–server berbasis REST API**. Sistem ini terdiri dari beberapa komponen utama, yaitu client, backend API, database, dan API publik.

Client (seperti Postman atau aplikasi frontend) mengirimkan request ke backend melalui HTTP dengan metode GET, POST, PUT, dan DELETE. Backend API dibangun menggunakan Node.js dan Express yang berfungsi untuk memproses request, menjalankan logika bisnis, serta melakukan autentikasi menggunakan JSON Web Token (JWT).

Setiap request yang mengakses endpoint tertentu harus menyertakan token JWT. Middleware JWT akan memvalidasi token tersebut sebelum request diteruskan ke controller. Jika token valid, sistem akan memproses permintaan, sedangkan jika tidak valid maka akses akan ditolak.

Backend API berkomunikasi dengan database MySQL untuk menyimpan dan mengambil data seperti data pengguna, data bencana, data relawan, dan data penugasan relawan. Selain itu, sistem juga terintegrasi dengan API publik untuk memperoleh informasi pendukung terkait bencana alam.

Arsitektur ini memungkinkan sistem berjalan secara terstruktur, aman, dan mudah dikembangkan karena pemisahan antara client, backend, dan database.

2.2 Perancangan Database

Database Smart Disaster Volunteer System dirancang menggunakan MySQL untuk menyimpan dan mengelola data yang berkaitan dengan pengguna, bencana, relawan, dan penugasan relawan. Perancangan database dilakukan agar data tersimpan secara terstruktur dan saling terhubung.

Database terdiri dari empat tabel utama yaitu **users**, **disasters**, **volunteers**, dan **assignments**. Tabel **users** menyimpan data akun pengguna sistem, tabel **disasters** menyimpan

informasi bencana alam, tabel **volunteers** menyimpan data relawan, dan tabel **assignments** digunakan untuk menghubungkan relawan dengan bencana tertentu.

Relasi antar tabel dibentuk menggunakan primary key dan foreign key, khususnya pada tabel **assignments** yang mereferensikan tabel **volunteers** dan **disasters**. Dengan perancangan ini, sistem dapat mengelola data relawan dan penugasannya secara konsisten dan efisien.

2.1.1 Pembuatan Database

Database **Smart Disaster Volunteer** dirancang menggunakan MySQL untuk menyimpan dan mengelola data pengguna, relawan, bencana, serta penugasan relawan. Perancangan database dilakukan secara relasional untuk menjamin konsistensi data antar tabel.

Nama Database yang di gunakan Adalah sebagai berikut:

```sql
CREATE DATABASE smart_disaster_volunteer;
USE smart_disaster_volunteer;
```

2.1.2 Perancangan Tabel Users

Tabel Users digunkan untuk menyimpn data akun pengguna yang dapat mengakses sistem

Jalankan perintah SQL pada basis data smart_disaster_voluntee

```sql
CREATE TABLE users(
    id_user INT AUTO_INCREMENT PRIMARY KEY,
    nama VARCHAR(50) NOT NULL,
    email VARCHAR(30) NOT NULL,
    password VARCHAR(50) NOT NULL,
    role ENUM('admin', 'koordinator', 'relawan') NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2.1.3 Tabel Disaster

Tbel disasters digunkan untuk menyimpan informasi bencana

```
CREATE TABLE disasters (
  id_disaster INT AUTO_INCREMENT PRIMARY
KEY,
  nama_bencana VARCHAR(100),
  lokasi VARCHAR(100),
  tanggal DATE,
  status ENUM('aktif','selesai'),
  created_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP
);
```

2.1.4 Tabel Volunteers

Tabel Volunteers menyimpan data relawan yang terhbung dengan akun pengguna

```
CREATE TABLE volunteers (
  id_volunteer INT AUTO_INCREMENT PRIMARY
KEY,
  id_user INT,
  skill VARCHAR(100),
  FOREIGN KEY (id_user) REFERENCES
users(id_user)
);
```

2.1.5 Tabel Assignments

Tabel assignment berfungsi sebagai penghubung antara relawan dan bencanaa

```sql
CREATE TABLE assignments (
  id_assignment INT AUTO_INCREMENT PRIMARY KEY,
  id_volunteer INT,
  id_disaster INT,
  tugas VARCHAR(100),
  FOREIGN KEY (id_volunteer) REFERENCES volunteers(id_volunteer),
  FOREIGN KEY (id_disaster) REFERENCES disasters(id_disaster)
);
```

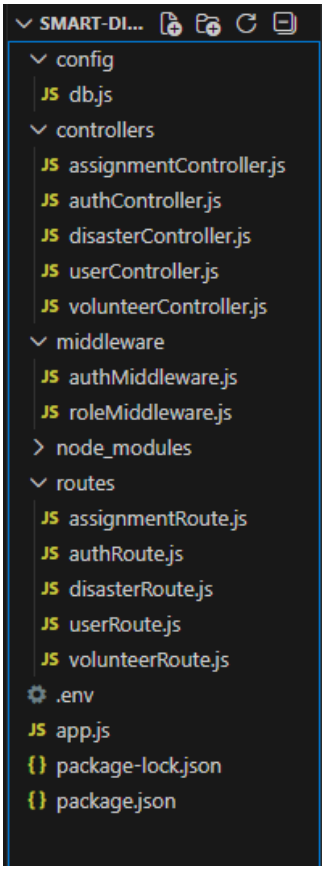
2.3 Struktur Folder Backend

Struktur folder backend dirancang agar mudah dipahami dan dikembangkan. Folder utama yang digunakan antara lain:

- controllers: berisi logika bisnis aplikasi
- routes: berisi definisi endpoint API
- middleware: berisi middleware autentikasi JWT
- cnfig: berisi konfigurasi database

2.2.1 Struktur Folder Project di Visual Code

Struktur folder backend dirancang modulr untuk memudahkan emelihraan dan pengembaangan sistem



2.4 Implementasi REST API

Implementasi REST API pada modul bencana dilakukan menggunakan Express Router. File disasterRoute.js digunakan untuk mendefinisikan endpoint yang berfungsi menangani request dari client. Endpoint GET digunakan untuk mengambil seluruh data bencana, sedangkan endpoint POST digunakan untuk menambahkan data bencana baru. Setiap endpoint dihubungkan dengan controller yang bertugas memproses logika aplikasi.

```
1  const express = require('express');
2  const router = express.Router();
3
4  const disasterController = require('../controllers/disasterController');
5
6  router.get('/', disasterController.getAllDisasters);
7  router.post('/', disasterController.createDisaster);
8
9  module.exports = router;
10
```

Potongan kode di bawah menunjukkan implementasi endpoint REST API menggunakan Express Router. Endpoint /register digunakan untuk menangani proses pendaftaran pengguna

baru, sedangkan endpoint GET pada modul bencana digunakan untuk mengambil data bencana dari sistem. Setiap endpoint dihubungkan dengan controller yang bertugas memproses logika aplikasi dan berinteraksi dengan database.

Potongan kode dibawah ini merupakan bagian dari middleware autentikasi yang digunakan untuk memverifikasi JSON Web Token (JWT). Token diperiksa pada setiap request yang mengakses endpoint tertentu. Apabila token tidak valid atau tidak tersedia, sistem akan menolak akses. Jika token valid, request akan diteruskan ke proses selanjutnya.

2.5 Implementasi Keamanan JWT

Keamanan REST API diterapkan menggunakan JSON Web Token (JWT) untuk memastikan bahwa hanya pengguna yang telah terautentikasi yang dapat mengakses endpoint tertentu. JWT diberikan kepada pengguna setelah berhasil melakukan proses login dan selanjutnya digunakan sebagai token akses.

Validasi JWT dilakukan melalui middleware authMiddleware.js. Middleware ini akan memeriksa keberadaan dan keabsahan token pada setiap request yang mengakses endpoint yang dilindungi. Jika token valid, request akan diteruskan ke controller, sedangkan jika token tidak valid atau tidak tersedia maka sistem akan menolak akses dengan status error.

```
const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(403).json({ message: "Token tidak ada" });

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) return res.status(401).json({ message: "Token invalid" });
    req.user = decoded;
    next();
  });
};
```

2.6 Integrasi API Publik

Sistem Smart Disaster Volunteer System juga mengintegrasikan API publik untuk memperoleh informasi pendukung terkait bencana alam. API publik digunakan untuk mengambil data eksternal yang relevan dan menampilkannya sebagai referensi tambahan dalam sistem.

Integrasi API publik dilakukan pada bagian controller dengan mengirimkan request ke layanan API eksternal dan mengolah response yang diterima. Data yang diperoleh kemudian dikembalikan ke client dalam format JSON sehingga dapat digunakan bersama data internal sistem.

2.7 Perancangan Endpoint REST API

Berikut adalah perancangan endpoint REST API yang digunakan pada sistem Smart Disaster Volunteer System. Endpoint dirancang untuk mendukung proses autentikasi, pengelolaan data bencana, relawan, dan penugasan relawan.

No	Method	Endpoint	Deskripsi
1	POST	/api/auth/register	Registrasi Pengguna
2	POST	/api/auth/login	Login pengguna
3	GET	/api/users	Menampilkan Semua User
4	GET	/api/users/:id	Detail User
5	PUT	/api/users/:id	Update data User

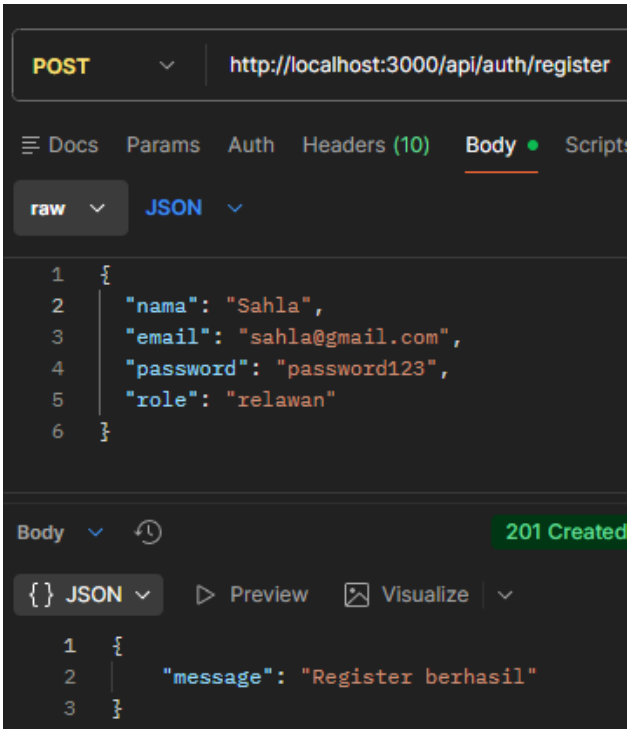
6	DELETE	/api/users/:id	Hapus User
7	POST	/api/disaster	Tambah Data Bencana
8	GET	/api/disaster	Meampilkan Semua Bencana
9	GET	/api/disaster/:id	Detail Bencana
10	PUT	/api/disaster/:id	Update data bencana
11	DELETE	/api/disaster/:id	Hapus Data Bencana
12	POST	/api/volunteer	Tambah Relawan
13	GET	/ api/volunteer	Menampilkan Relawan
14	GET	/ api/volunteer/:id	Detail Relawan
15	PUT	/ api/volunteer/:id	Update Relawan
16	DELETE	/ api/volunteer/:id	Hapus Relawan
17	POST	/api/assignments	Tambah Penugsan relawan
18	GET	/ api/assignments	Menampilkan Penugasan
19	GET	/ api/assignments/:id	Detail Penugasan
20	DELETE	api/assignments/:id	Hapus Penugasan

2.8 Authentication API

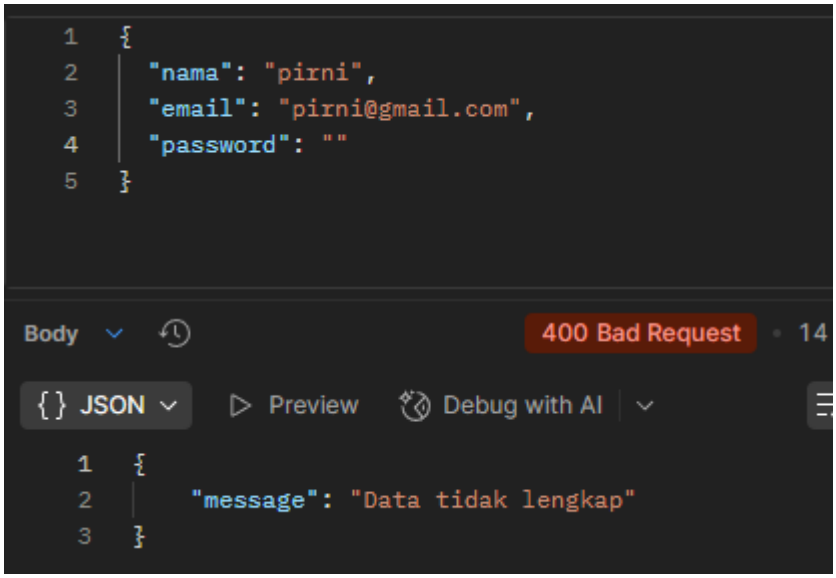
2.8.1 Register User

Endpoint POST /api/auth/register Digunakan untuk mendaftarkan pengguna baru ke dalam sistem.

Respone Berhasil (20 Created)

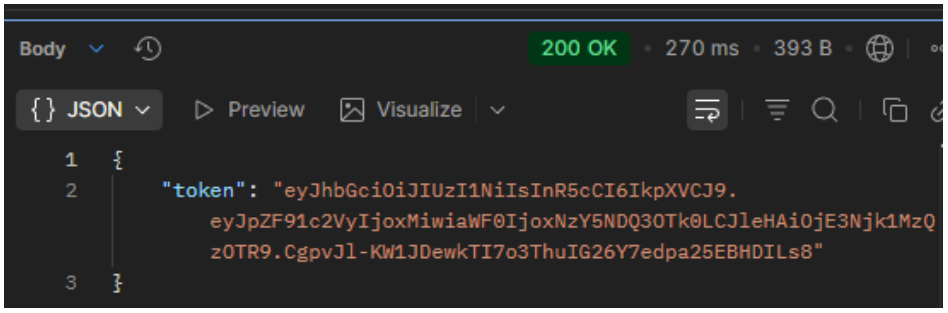


Respone Gagal (400 Bad Request)

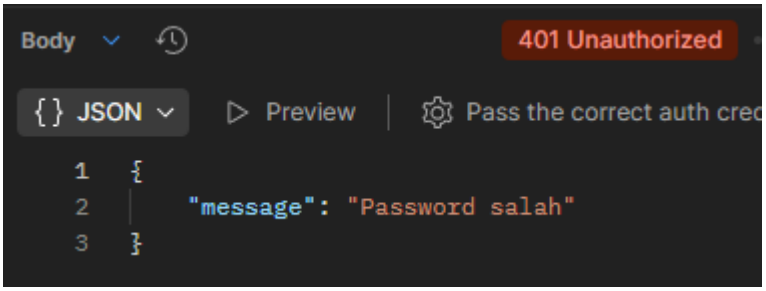


2.8.2 Login User

Respone Berhasil (200 OK)

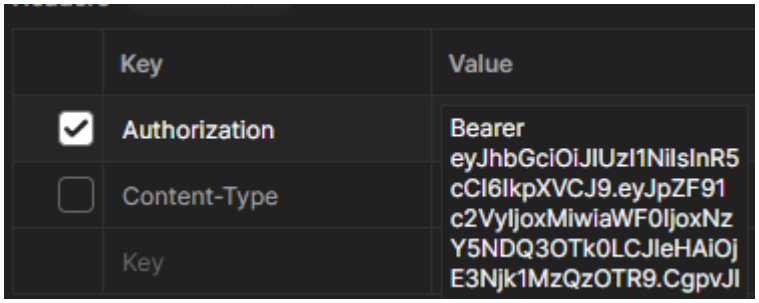


Respone Gagal (40 Unauthorized



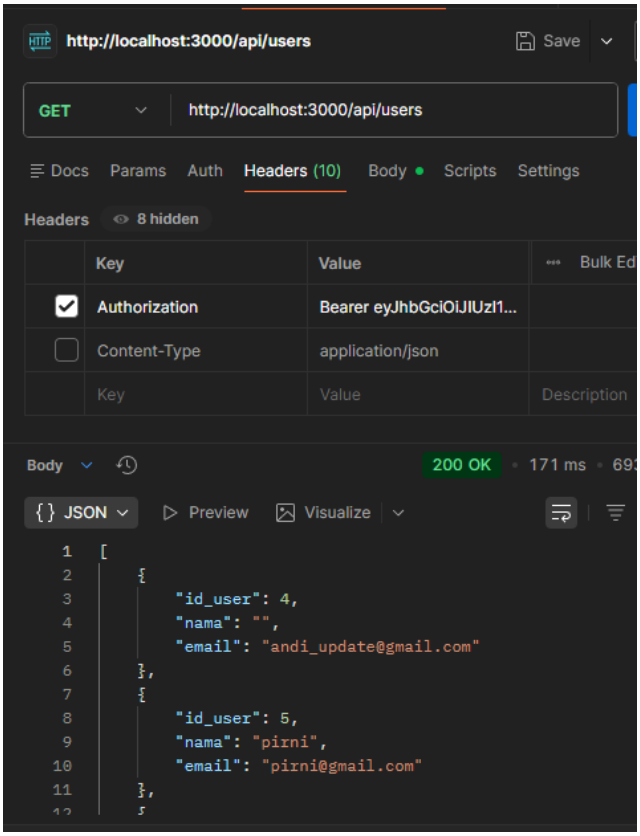
2.8.3 Users APi

Semua endpoint USERS **wajib menggunakan JWT, Header Authorization: Bearer <token>**

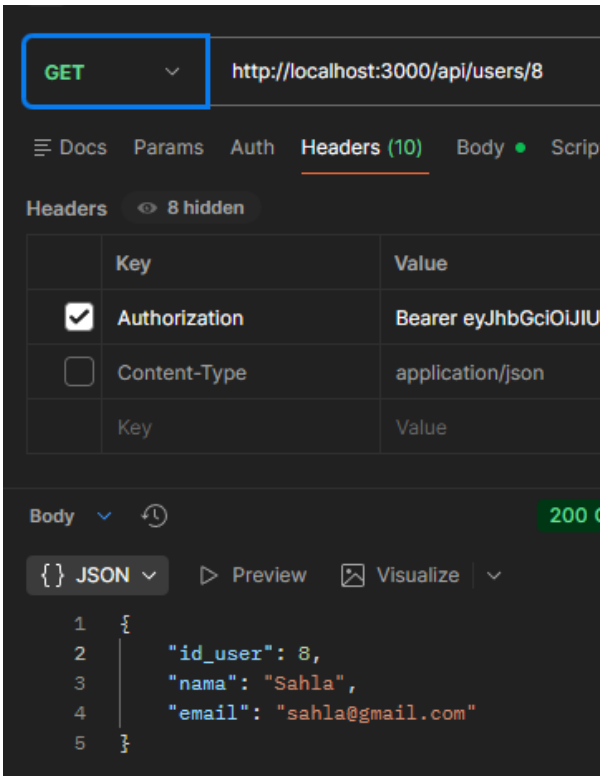


2.8.4 Get All Users, Users By ID dan Update Users

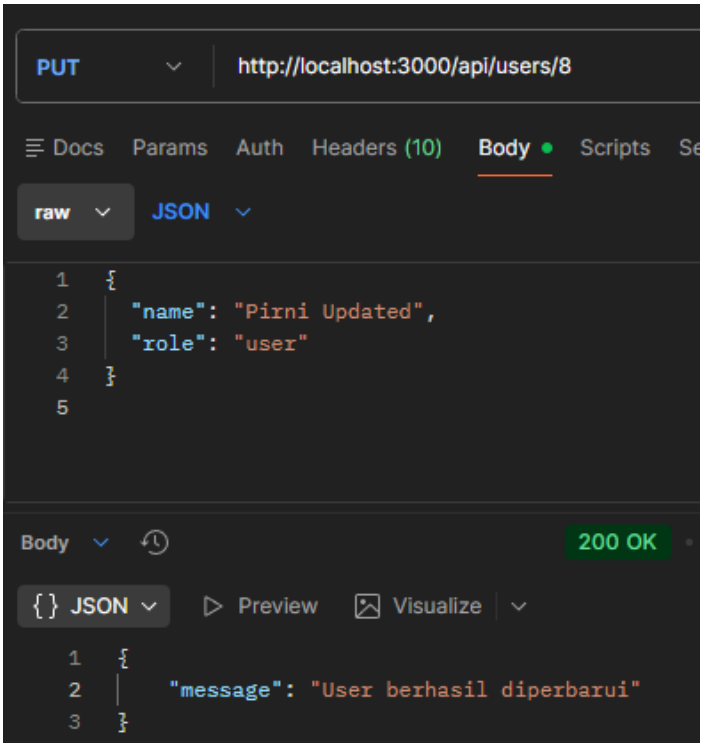
GET /api/user Jika Respone berhasil Maka (200 OK)



GET /api/user/:id Jika Respone berhasil Maka (200 OK)



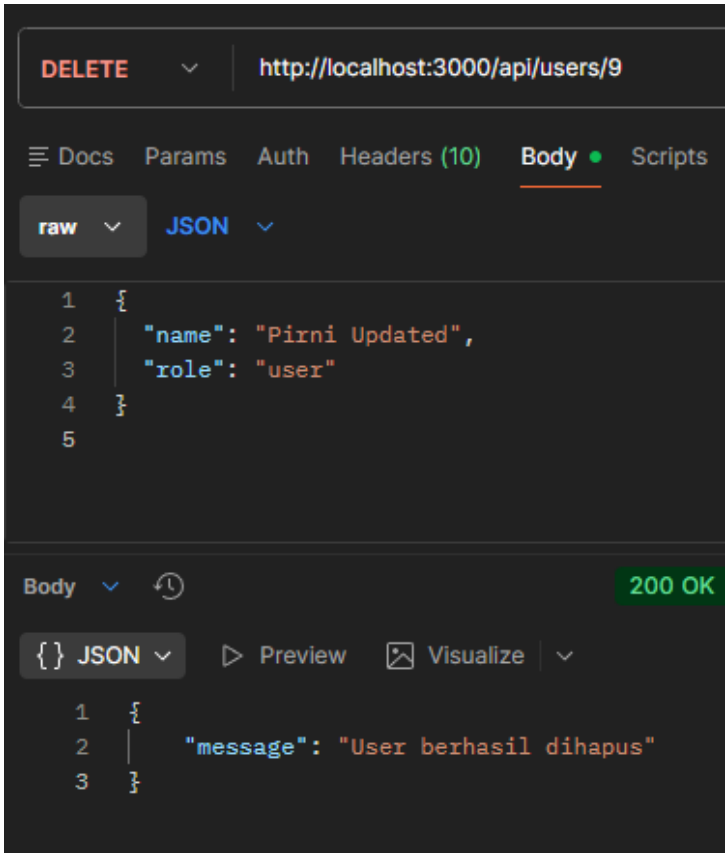
PUT /api/user/:id Jika Respone berhasil Maka (200 OK)



2.8.5 Delete Users

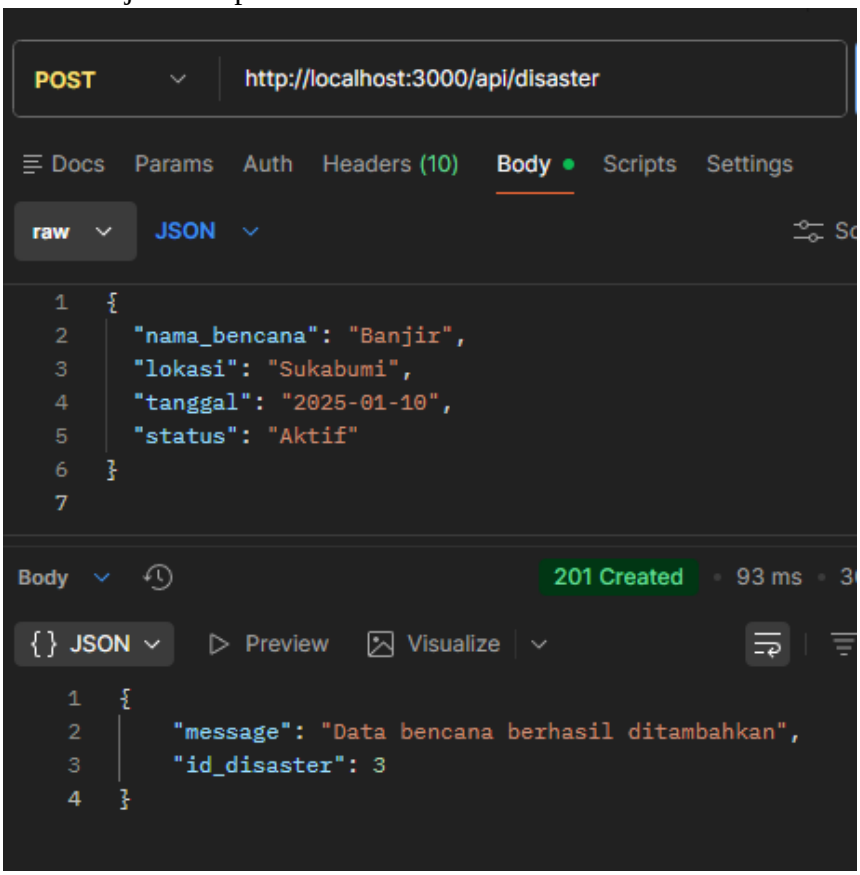
DELETE /api/users/:id

Respone Berhasil (200 OK)

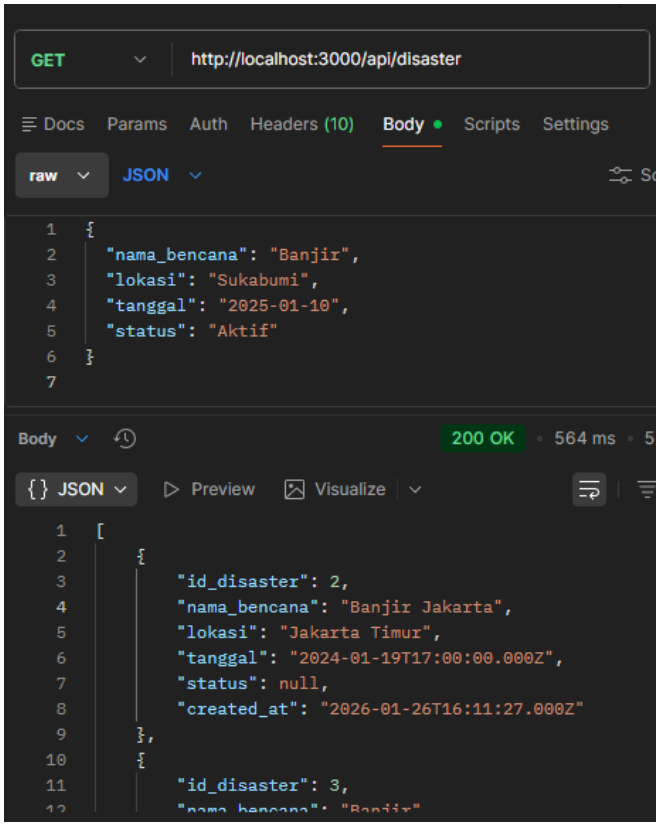


2.8.6 Disaster API

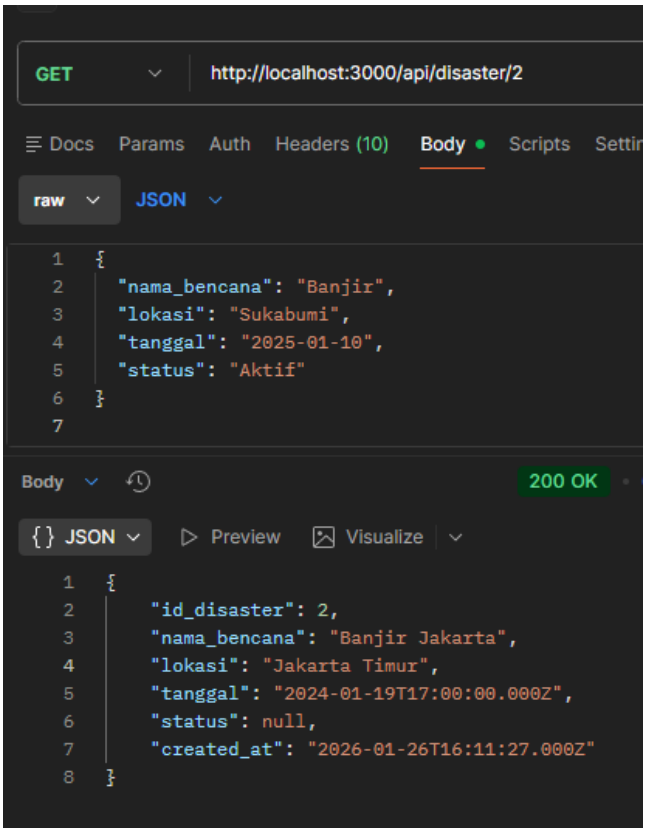
- **Create Disaster**
  POST /api/disaster jika REspone berhasil aka **201 created**



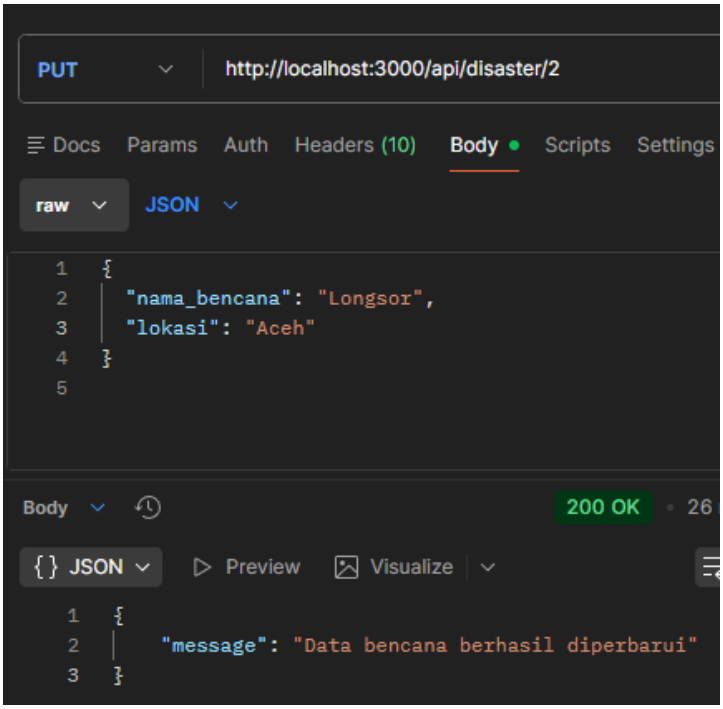
- **Get All Disaster**
  GET /api/disaster jika Respone Berhasil **200 OK**



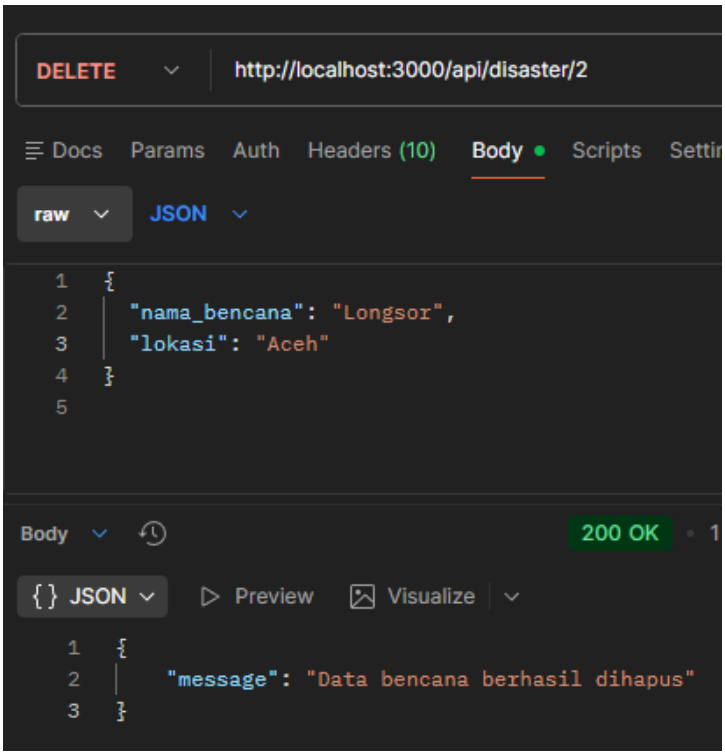
- **GET Disaster By ID**
  GET /api/disaster/:id



- **Update Disaster**
  PUT /api/disaster/:id

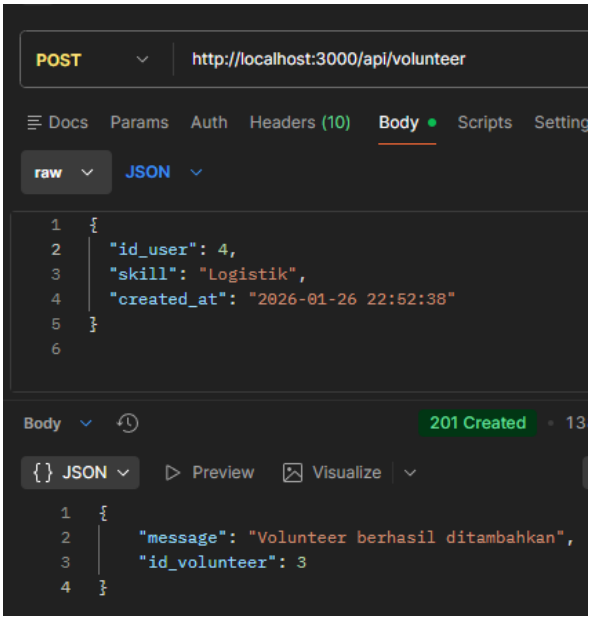


- **DELETE Disaster**
  DELETE /api/disaster/:id

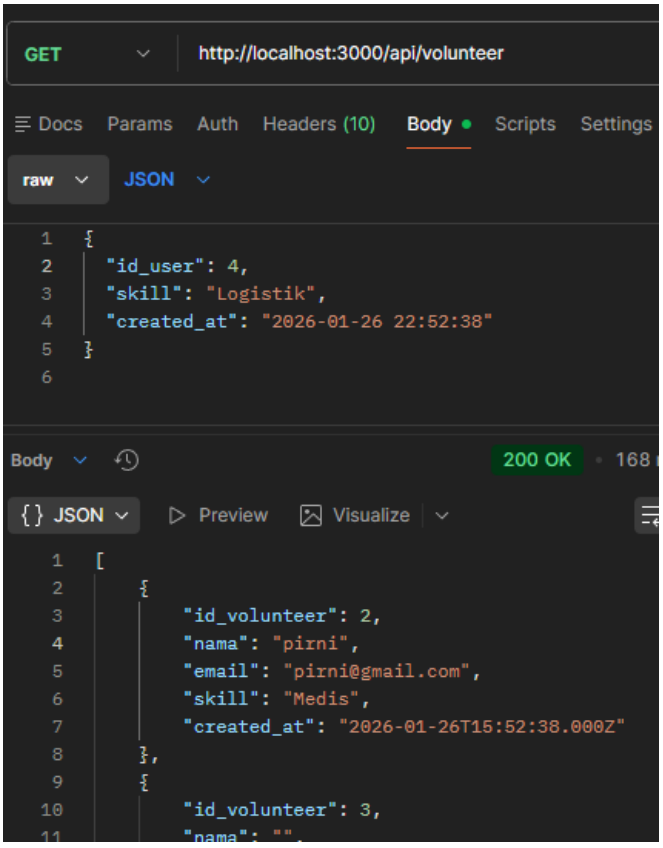


2.8.7 Volunteer API

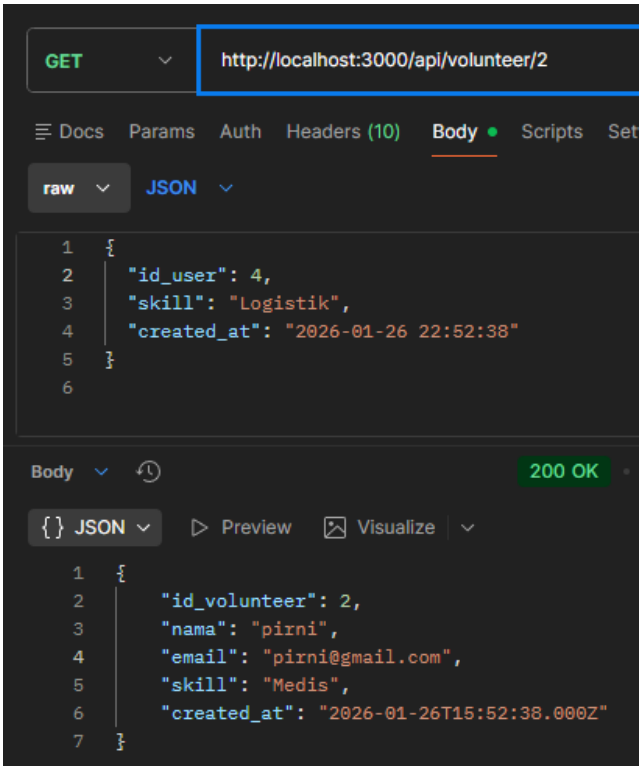
- **Create Volunteer**
  POST /api/volunteer jika berhasil Respone **201 Created**



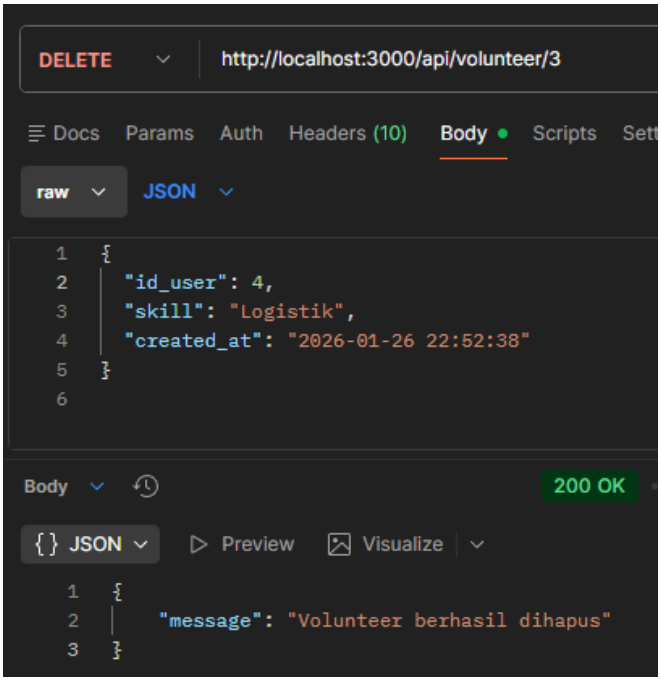
- **Get All Volunteer**
  GET /api/volunteer



- **Get Volunteer By ID**
  GET /api/volunteer/:id

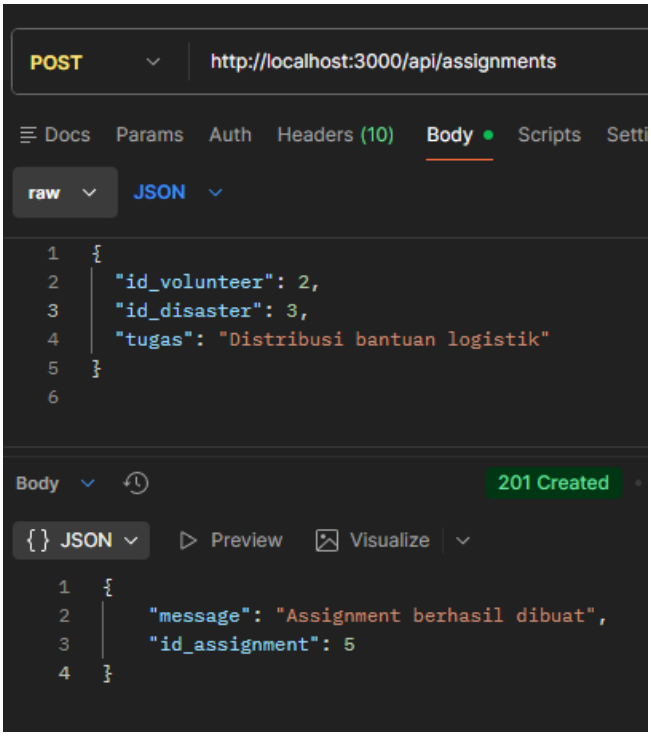


- **Delete Volunteer**
  DELETE /api/volunteer/;id

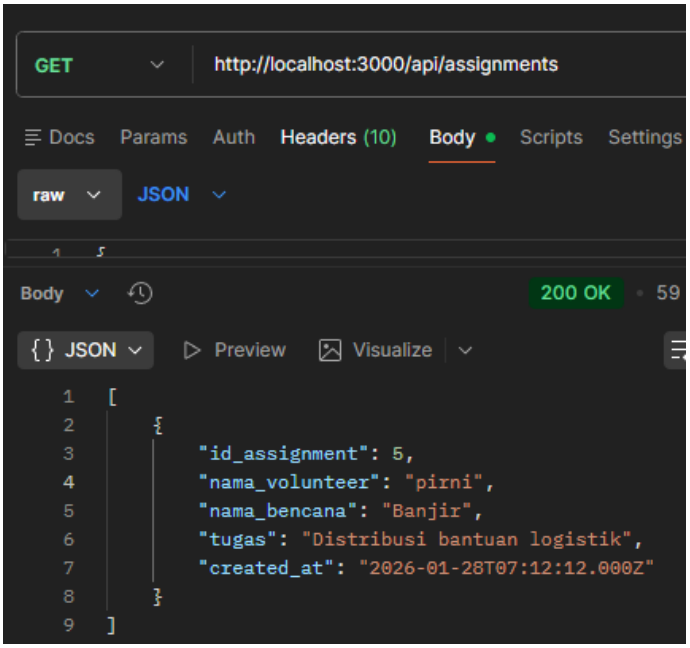


2.8.8 Assignment Api

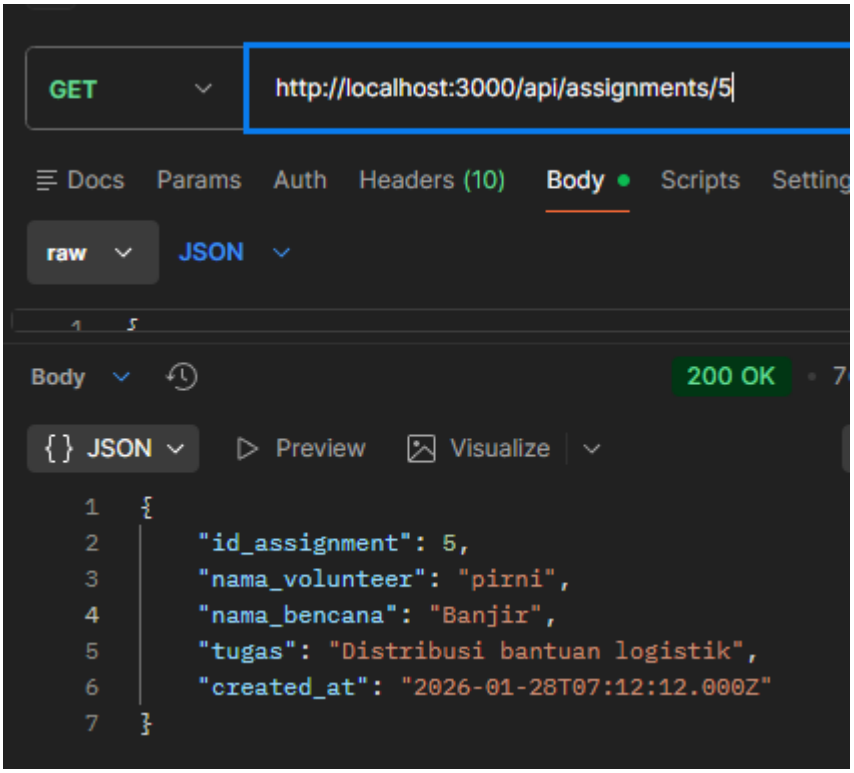
- **Create Assignment**
  POST /api/assignment jika Berhasi **201 Created**



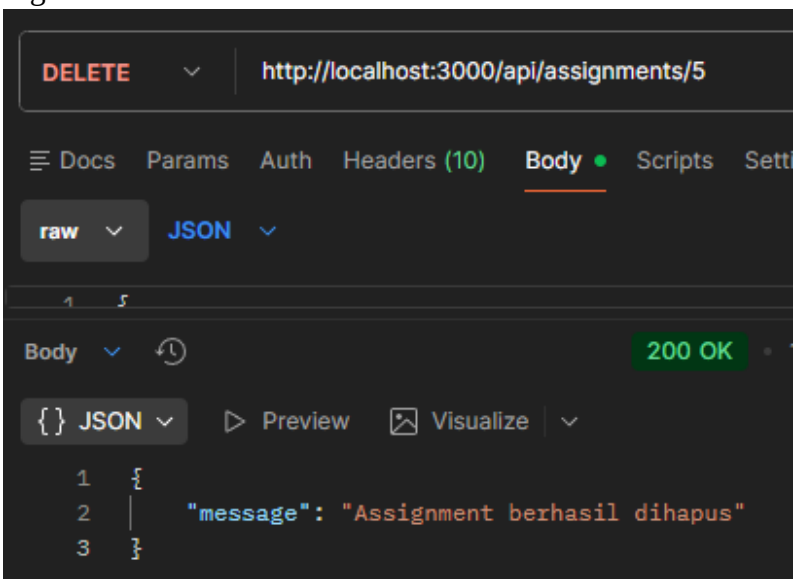
- **Get All Assignment**
  GET /api/assignments



- **Get Assignment By ID**
  GET /api/assignment/:id



- **Delete Assignment**
  DELETE /api/assignments/:id



KESIMPULAN

3. 1 Kesimpulan

Berdasarkan pembahasan pada bab ini, dapat disimpulkan bahwa sistem Smart Disaster Volunteer System telah berhasil dirancang dan diimplementasikan menggunakan arsitektur REST API berbasis Node.js dan Express. Perancangan database yang saling berelasi mampu mendukung pengelolaan data pengguna, bencana, relawan, dan penugasan relawan secara terstruktur. Selain itu, penerapan autentikasi dan otorisasi menggunakan JSON Web Token (JWT) memberikan lapisan keamanan dalam pengaksesan endpoint sistem.

3. 2 Saran

Sistem yang telah dibangun masih memiliki peluang untuk dikembangkan lebih lanjut. Beberapa pengembangan yang dapat dilakukan antara lain penambahan fitur manajemen lokasi relawan secara real-time, peningkatan validasi data pada setiap endpoint, serta optimalisasi integrasi API publik agar data bencana yang diperoleh lebih akurat dan terkini.

LAMPIRAN

Lampiran A

A.1 App.js

konfigurasi express dan pemanggilan routes

```
require("dotenv").config();
const express = require("express");
const app = express();

app.use(express.json());

const authRoutes = require("./routes/authRoute");
const volunteerRoutes = require("./routes/volunteerRoute");
const disasterRoute = require("./routes/disasterRoute");
const assignmentRoute = require("./routes/assignmentRoute");
const userRoute = require("./routes/userRoute");

app.use("/api/auth", authRoutes);
app.use("/api/volunteer", volunteerRoutes);
app.use("/api/disaster", disasterRoute);
app.use("/api/assignments", assignmentRoute);
app.use("/api/users", userRoute);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server berjalan di port ${PORT}`);
});

```

A.2 Contollers

- **AssignmentController.js**

```
const db = require('../config/db');

exports.createAssignment = (req, res) => {
  const { id_volunteer, id_disaster, tugas } = req.body;

  if (!id_volunteer || !id_disaster || !tugas) {
    return res.status(400).json({ message: 'Data assignment tidak lengkap' });
  }

  db.query(
    'INSERT INTO assignments (id_volunteer, id_disaster, tugas) VALUES (?, ?, ?)',
    [id_volunteer, id_disaster, tugas],
    (err, result) => {
      if (err) return res.status(500).json(err);

      res.status(201).json({
        message: 'Assignment berhasil dibuat',
        id_assignment: result.insertId
      });
    }
  );
};


exports.getAssignments = (req, res) => {
  db.query(
    `SELECT
        a.id_assignment,
        u.nama AS nama_volunteer,
        d.nama_bencana,
        a.tugas,
        a.created_at
     FROM assignments a
     JOIN volunteers v ON a.id_volunteer = v.id_volunteer
     JOIN users u ON v.id_user = u.id_user
     JOIN disasters d ON a.id_disaster = d.id_disaster`,
    (err, results) => {
      if (err) return res.status(500).json(err);
      res.json(results);
    }
  );
};

exports.getAssignmentById = (req, res) => {
  const { id } = req.params;

  db.query(
    `SELECT
        a.id_assignment,
        u.nama AS nama_volunteer,
        d.nama_bencana,
        a.tugas,
        a.created_at
     FROM assignments a
     JOIN volunteers v ON a.id_volunteer = v.id_volunteer
     JOIN users u ON v.id_user = u.id_user
     JOIN disasters d ON a.id_disaster = d.id_disaster
     WHERE a.id_assignment = ?`,
    [id],
    (err, results) => {
      if (err) return res.status(500).json(err);
      if (results.length === 0) {
        return res.status(404).json({ message: 'Assignment tidak ditemukan' });
      }
      res.json(results[0]);
    }
  );
};


exports.deleteAssignment = (req, res) => {
  const { id } = req.params;

  db.query(
    'DELETE FROM assignments WHERE id_assignment = ?',
    [id],
    (err) => {
      if (err) return res.status(500).json(err);
      res.json({ message: 'Assignment berhasil dihapus' });
    }
  );
};

```

- **AuthController.js**

```
const db = require('../config/db');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

exports.register = (req, res) => {
  const { nama, email, password } = req.body;

  if (!nama || !email || !password) {
    return res.status(400).json({ message: 'Data tidak lengkap' });
  }

  const hashedPassword = bcrypt.hashSync(password, 10);

  db.query(
    'INSERT INTO users (nama, email, password) VALUES (?, ?, ?)',
    [nama, email, hashedPassword],
    (err) => {
      if (err) return res.status(500).json(err);
      res.status(201).json({ message: 'Register berhasil' });
    }
  );
};

exports.login = (req, res) => {
  const { email, password } = req.body;

  db.query(
    'SELECT * FROM users WHERE email = ?',
    [email],
    (err, results) => {
      if (err) return res.status(500).json(err);
      if (results.length === 0) {
        return res.status(401).json({ message: 'Email tidak ditemukan' });
      }

      const user = results[0];
      const isMatch = bcrypt.compareSync(password, user.password);

      if (!isMatch) {
        return res.status(401).json({ message: 'Password salah' });
      }

      const token = jwt.sign(
        { id_user: user.id_user },
        process.env.JWT_SECRET,
        { expiresIn: '1d' }
      );

      res.json({ token });
    }
  );
};

exports.me = (req, res) => {
  const id_user = req.user.id_user;

  db.query(
    'SELECT id_user, nama, email FROM users WHERE id_user = ?',
    [id_user],
    (err, results) => {
      if (err) return res.status(500).json(err);
      res.json(results[0]);
    }
  );
};

```

- **DisasterController.js**

```
const db = require('../config/db');

exports.createDisaster = (req, res) => {
  const { nama_bencana, lokasi, tanggal, status } = req.body;

  db.query(
    'INSERT INTO disasters (nama_bencana, lokasi, tanggal, status) VALUES (?, ?, ?, ?)',
    [nama_bencana, lokasi, tanggal, status],
    (err, result) => {
      if (err) return res.status(500).json(err);
      res.status(201).json({
        message: 'Data bencana berhasil ditambahkan',
        id_disaster: result.insertId
      });
    }
  );
};

exports.getDisasters = (req, res) => {
  db.query(
    'SELECT * FROM disasters',
    (err, results) => {
      if (err) return res.status(500).json(err);
      res.json(results);
    }
  );
};

exports.getDisasterById = (req, res) => {
  const { id } = req.params;

  db.query(
    'SELECT * FROM disasters WHERE id_disaster = ?',
    [id],
    (err, results) => {
      if (err) return res.status(500).json(err);
      if (results.length === 0) {
        return res.status(404).json({ message: 'Bencana tidak ditemukan' });
      }
      res.json(results[0]);
    }
  );
};

exports.updateDisaster = (req, res) => {
  const { id } = req.params;
  const { nama_bencana, lokasi, tanggal, status } = req.body;

  db.query(
    'UPDATE disasters SET nama_bencana=?, lokasi=?, tanggal=?, status=? WHERE id_disaster=?',
    [nama_bencana, lokasi, tanggal, status, id],
    (err) => {
      if (err) return res.status(500).json(err);
      res.json({ message: 'Data bencana berhasil diperbarui' });
    }
  );
};


exports.deleteDisaster = (req, res) => {
  const { id } = req.params;

  db.query(
    'DELETE FROM disasters WHERE id_disaster = ?',
    [id],
    (err) => {
      if (err) return res.status(500).json(err);
      res.json({ message: 'Data bencana berhasil dihapus' });
    }
  );
};

```

- **VolunteerController.js**

```
const db = require('../config/db');

exports.createVolunteer = (req, res) => {
  const { id_user, skill } = req.body;

  if (!id_user || !skill) {
    return res.status(400).json({ message: 'id_user dan skill wajib diisi' });
  }

  db.query(
    'INSERT INTO volunteers (id_user, skill) VALUES (?, ?)',
    [id_user, skill],
    (err, result) => {
      if (err) return res.status(500).json(err);
      res.status(201).json({
        message: 'Volunteer berhasil ditambahkan',
        id_volunteer: result.insertId
      });
    }
  );
};

exports.getVolunteers = (req, res) => {
  db.query(
    `SELECT v.id_volunteer, u.nama, u.email, v.skill, v.created_at
     FROM volunteers v
     JOIN users u ON v.id_user = u.id_user`,
    (err, results) => {
      if (err) return res.status(500).json(err);
      res.json(results);
    }
  );
};

exports.getVolunteerById = (req, res) => {
  const { id } = req.params;

  db.query(
    `SELECT v.id_volunteer, u.nama, u.email, v.skill, v.created_at
     FROM volunteers v
     JOIN users u ON v.id_user = u.id_user
     WHERE v.id_volunteer = ?`,
    [id],
    (err, results) => {
      if (err) return res.status(500).json(err);
      if (results.length === 0) {
        return res.status(404).json({ message: 'Volunteer tidak ditemukan' });
      }
      res.json(results[0]);
    }
  );
};

exports.deleteVolunteer = (req, res) => {
  const { id } = req.params;

  db.query(
    'DELETE FROM volunteers WHERE id_volunteer = ?',
    [id],
    (err) => {
      if (err) return res.status(500).json(err);
      res.json({ message: 'Volunteer berhasil dihapus' });
    }
  );
};

```

- **UsersController.js**

```
const db = require('../config/db');

exports.getUsers = (req, res) => {
  db.query(
    'SELECT id_user, nama, email FROM users',
    (err, results) => {
      if (err) return res.status(500).json(err);
      res.json(results);
    }
  );
};

exports.getUserById = (req, res) => {
  const { id } = req.params;

  db.query(
    'SELECT id_user, nama, email FROM users WHERE id_user = ?',
    [id],
    (err, results) => {
      if (err) return res.status(500).json(err);
      if (results.length === 0) {
        return res.status(404).json({ message: 'User tidak ditemukan' });
      }
      res.json(results[0]);
    }
  );
};

exports.updateUser = (req, res) => {
  const { id } = req.params;
  const { name, email } = req.body;

  db.query(
    'UPDATE users SET nama = ?, email = ? WHERE id_user = ?',
    [name, email, id],
    (err) => {
      if (err) return res.status(500).json(err);
      res.json({ message: 'User berhasil diperbarui' });
    }
  );
};

exports.deleteUser = (req, res) => {
  const { id } = req.params;

  db.query(
    'DELETE FROM users WHERE id_user = ?',
    [id],
    (err) => {
      if (err) return res.status(500).json(err);
      res.json({ message: 'User berhasil dihapus' });
    }
  );
};
```

A.3 Middleware

- **AuthMiddleware.js**

```
const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(403).json({ message: "Token tidak ada" });

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) return res.status(401).json({ message: "Token invalid" });
    req.user = decoded;
    next();
  });
};

```

- **roleMiddleware.js**

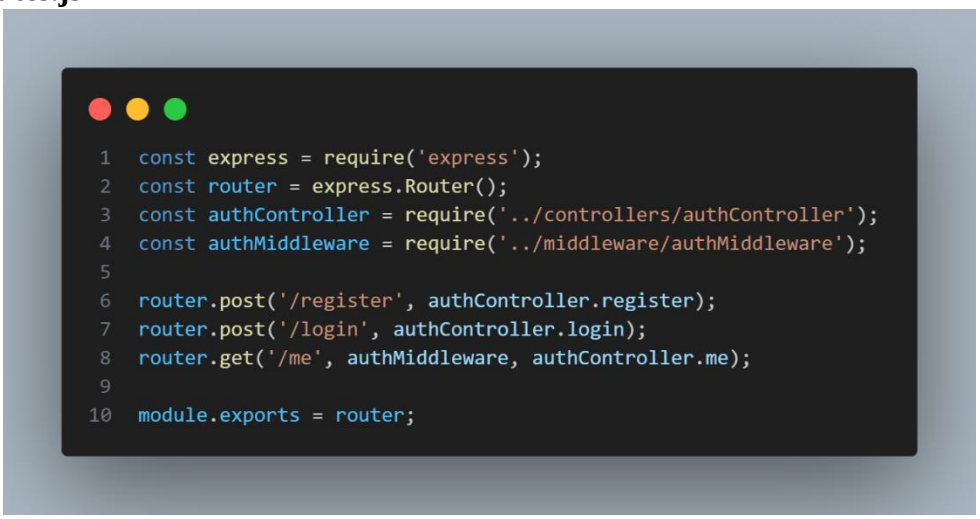
```
module.exports = (role) => {
  return (req, res, next) => {
    if (req.user.role !== role) {
      return res.status(403).json({
        message: "Akses ditolak"
      });
    }
    next();
  };
};

```

A.4 Routes

- **AssignmentRoute.js**

```
const express = require('express');
const router = express.Router();
const assignmentController = require('../controllers/assignmentController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/', authMiddleware, assignmentController.createAssignment);
router.get('/', authMiddleware, assignmentController.getAssignments);
router.get('/:id', authMiddleware, assignmentController.getAssignmentById);
router.delete('/:id', authMiddleware, assignmentController.deleteAssignment);

module.exports = router;
```

- **AuthRoutes.js**



```
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/register', authController.register);
router.post('/login', authController.login);
router.get('/me', authMiddleware, authController.me);

module.exports = router;
```

- **Disasterroute.js**

```
const express = require('express');
const router = express.Router();
const disasterController = require('../controllers/disasterController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/', authMiddleware, disasterController.createDisaster);
router.get('/', authMiddleware, disasterController.getDisasters);
router.get('/:id', authMiddleware, disasterController.getDisasterById);
router.put('/:id', authMiddleware, disasterController.updateDisaster);
router.delete('/:id', authMiddleware, disasterController.deleteDisaster);

module.exports = router;
```

- **Volunteerroute.js**

```js
const express = require('express');
const router = express.Router();
const volunteerController = require('../controllers/volunteerController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/', authMiddleware, volunteerController.createVolunteer);
router.get('/', authMiddleware, volunteerController.getVolunteers);
router.get('/:id', authMiddleware, volunteerController.getVolunteerById);
router.delete('/:id', authMiddleware, volunteerController.deleteVolunteer);

module.exports = router;
```

- **Usersroute.js**

```js
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');
const authMiddleware = require('../middleware/authMiddleware');

router.get('/', authMiddleware, userController.getUsers);
router.get('/:id', authMiddleware, userController.getUserById);
router.put('/:id', authMiddleware, userController.updateUser);
router.delete('/:id', authMiddleware, userController.deleteUser);

module.exports = router;
```

A.5 db.js

```js
const mysql = require("mysql");

const db = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "",
    database: "smart_disaster_volunteer"
});

db.connect(err => {
  if (err) {
    console.log("Database connection failed:", err);
  } else {
    console.log("Database connected");
  }
});

module.exports = db;
```

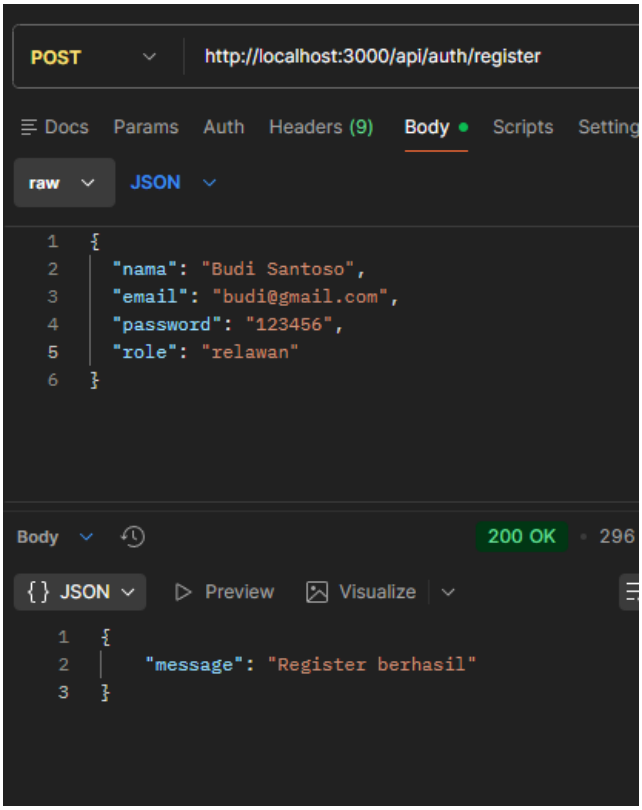
A.6 .env

```
PORT=3000

DB_HOST=localhost
DB_USER=root
DB_PASSWORD=
DB_NAME=smart_disaster_volunteer

JWT_SECRET=disaster_secret_key

```

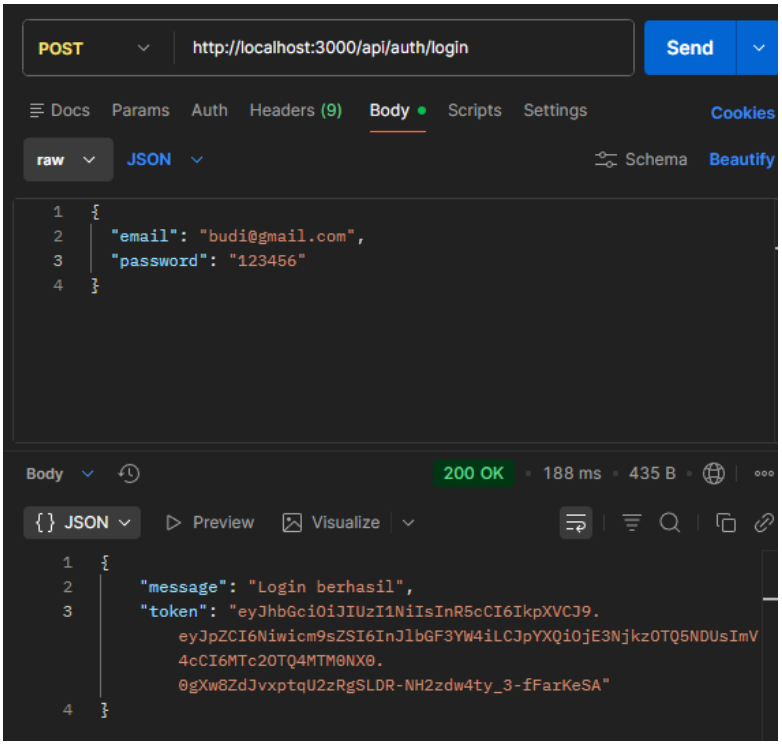
Lampiran B

B.1 AUTH

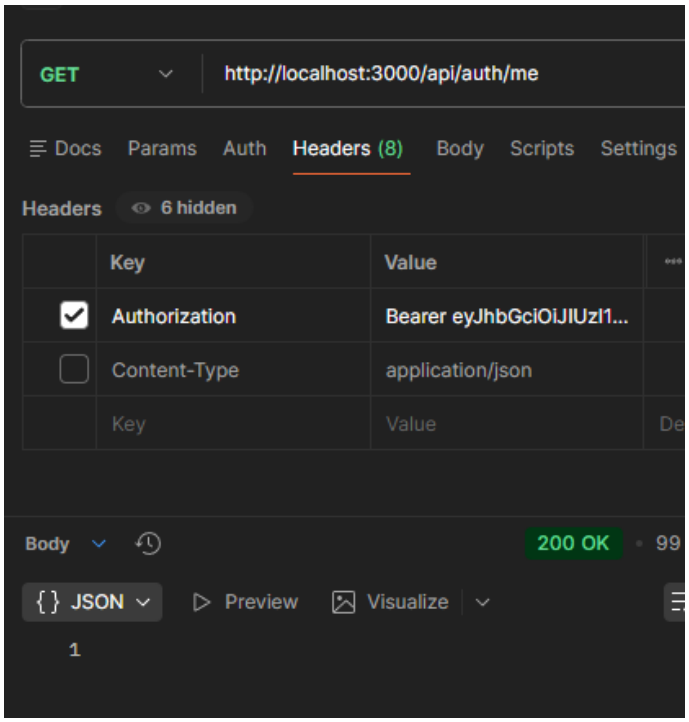
- POST /api/auth/register



- POST /api/auth/login

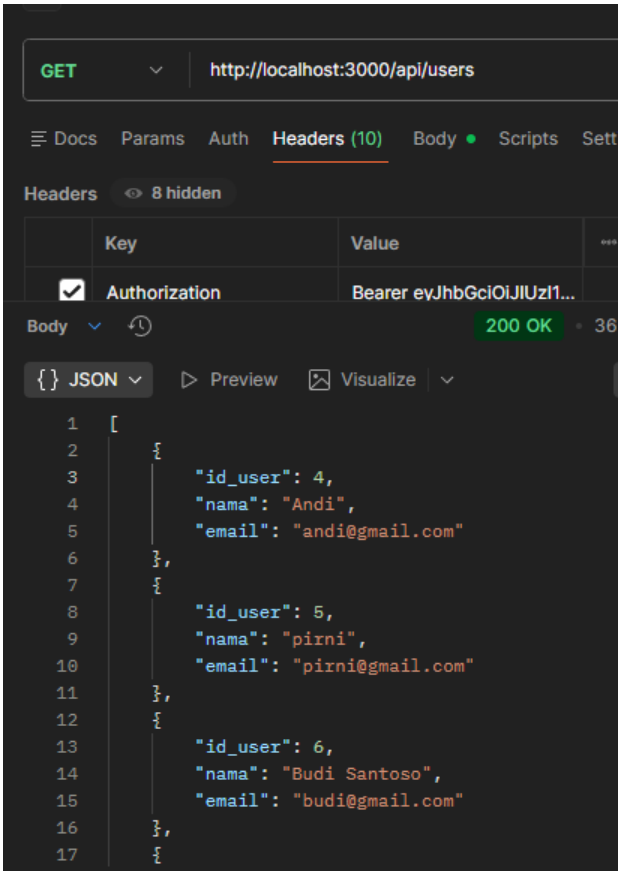


- GET /api/auth/me

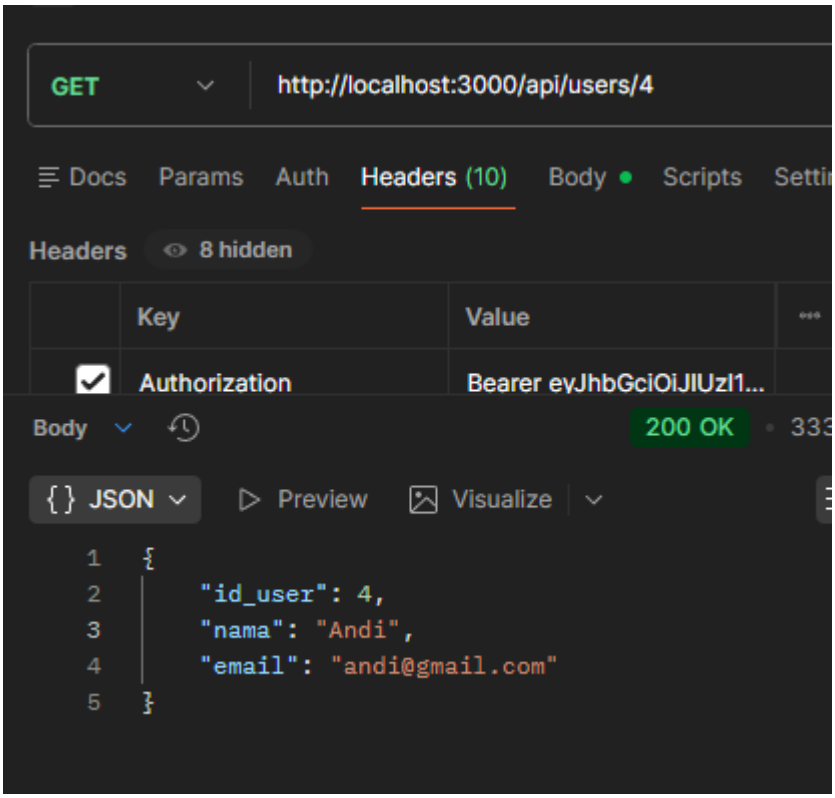


B.2 USERS

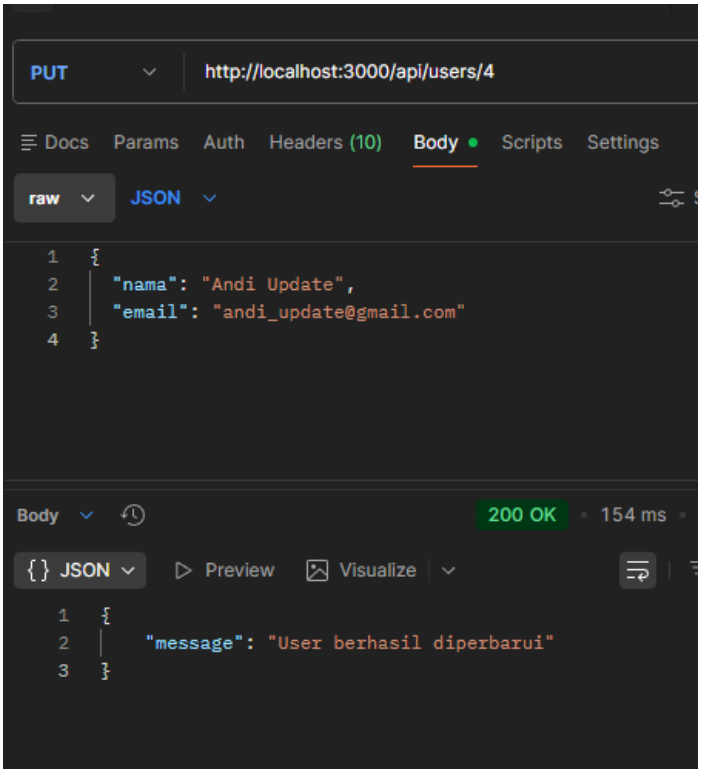
- GET /api/users



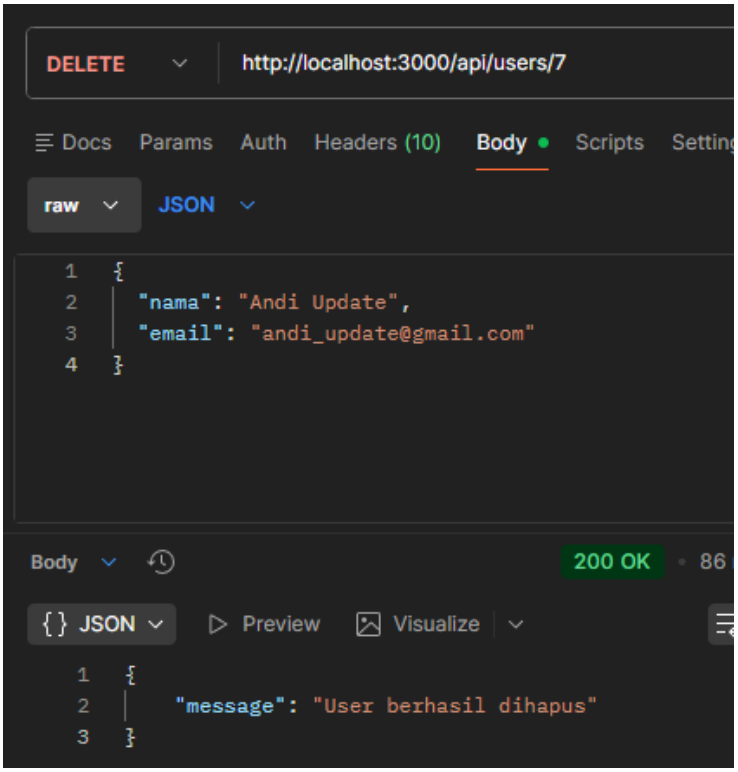
- GET /api/users/:id



- PUT /api/users/:id

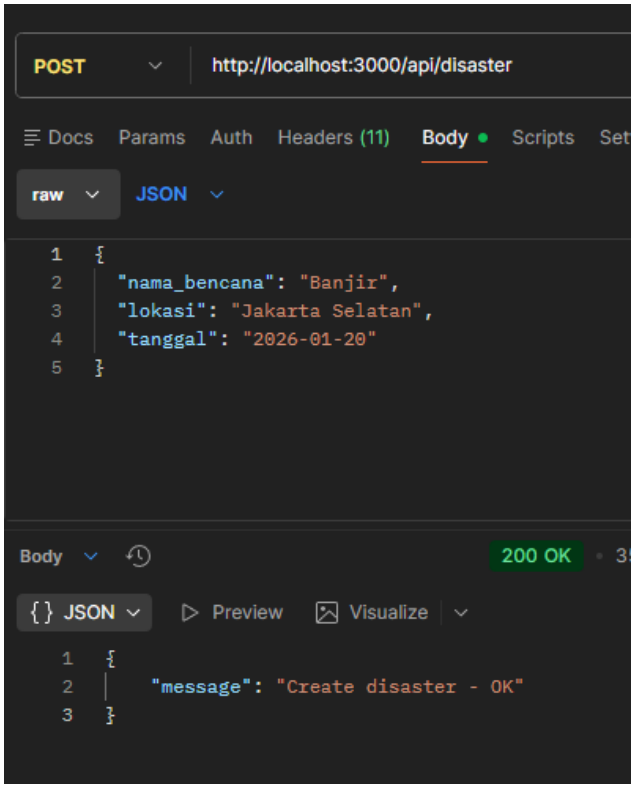


- DELETE /api/users/:id

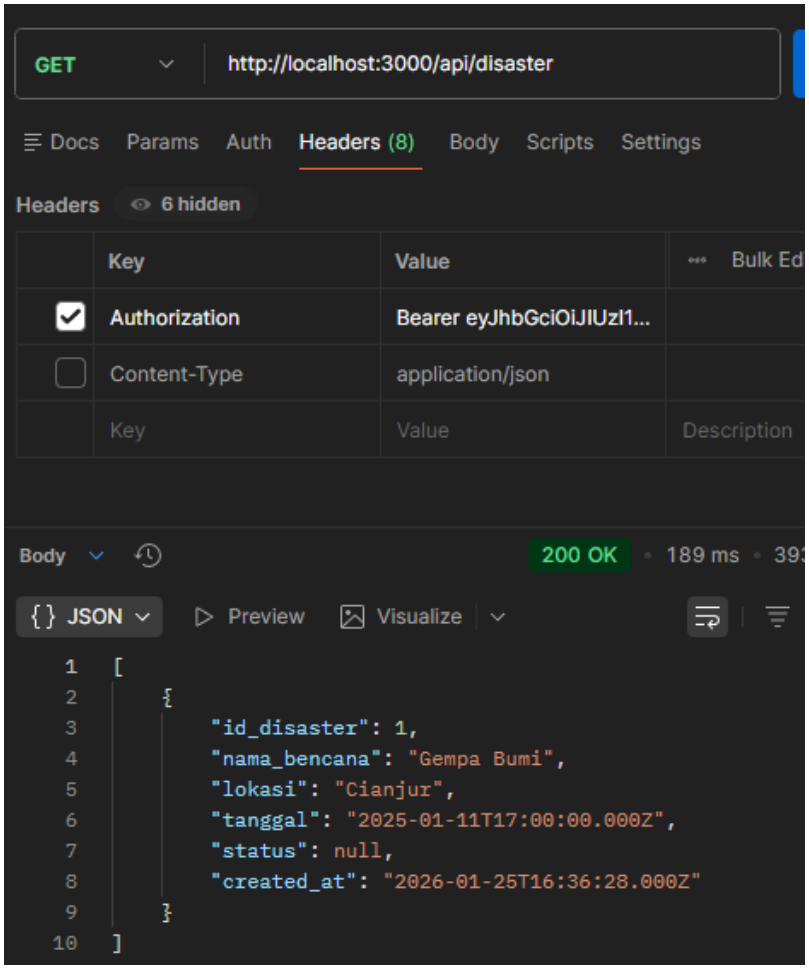


B.3 DISASTERS

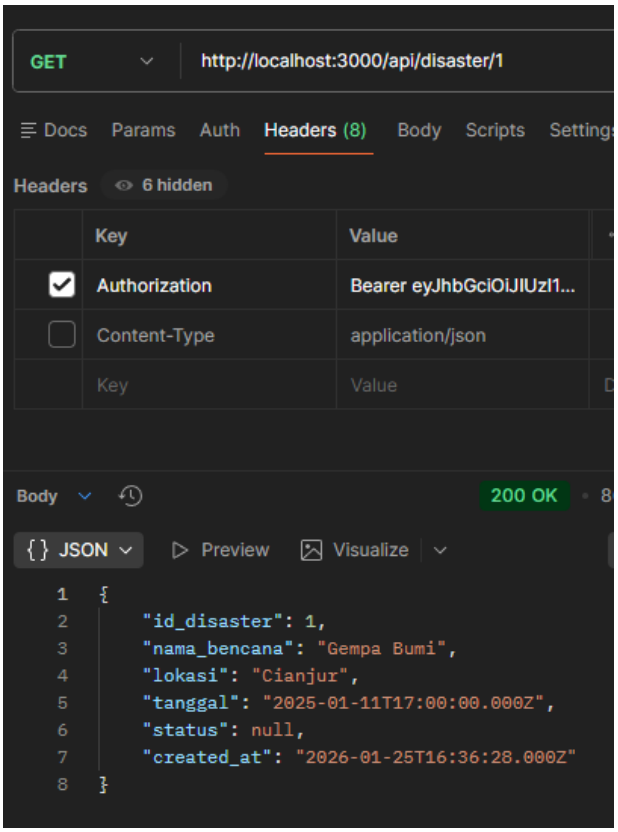
- POST /api/disaster



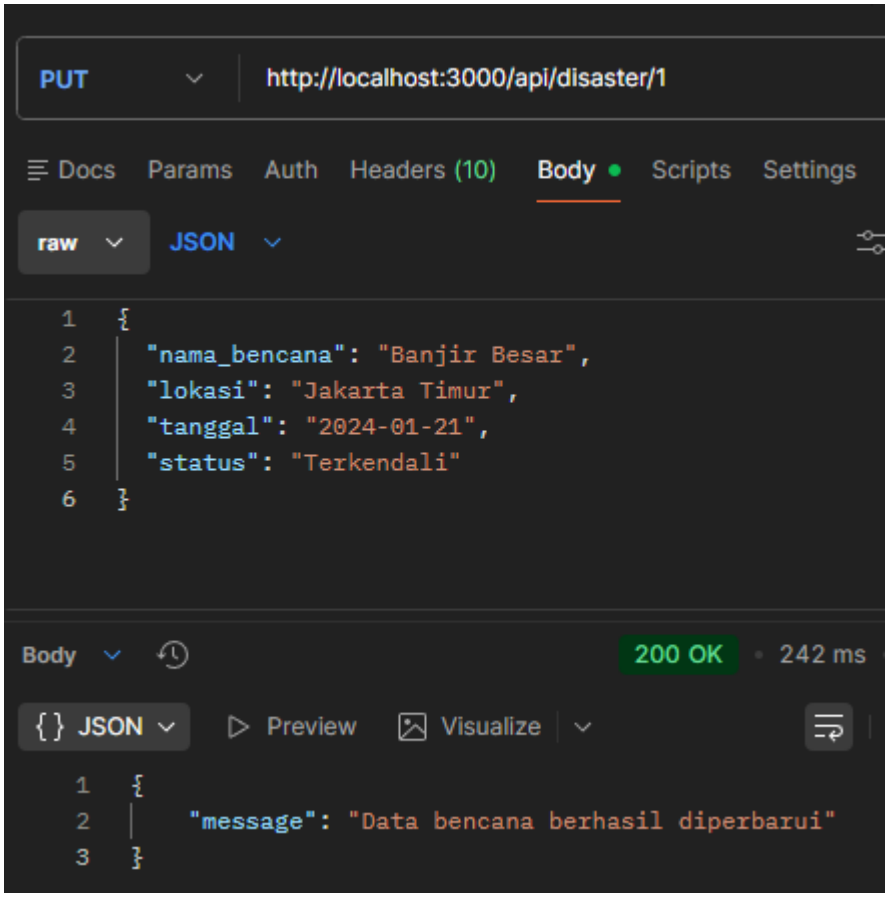
- GET /api/disaster



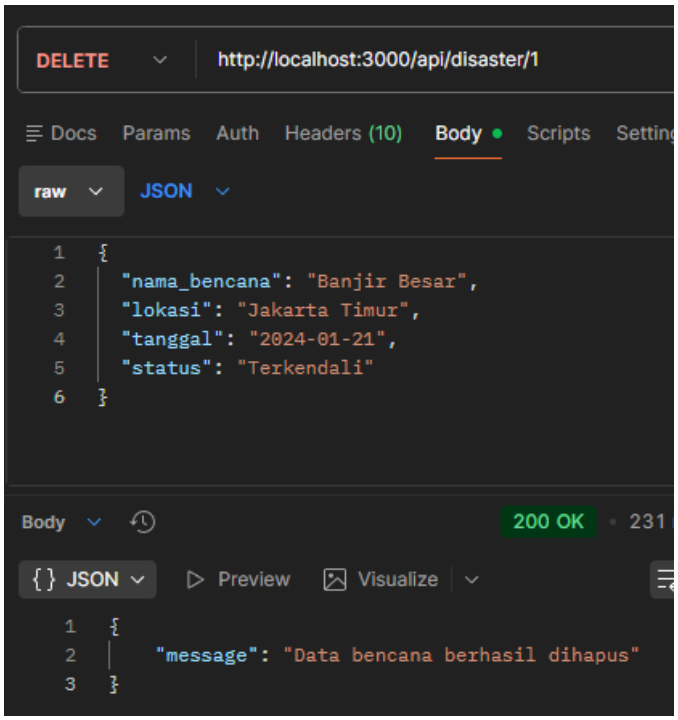
- GET /api/disaster/:id



- PUT /api/disaster/:id

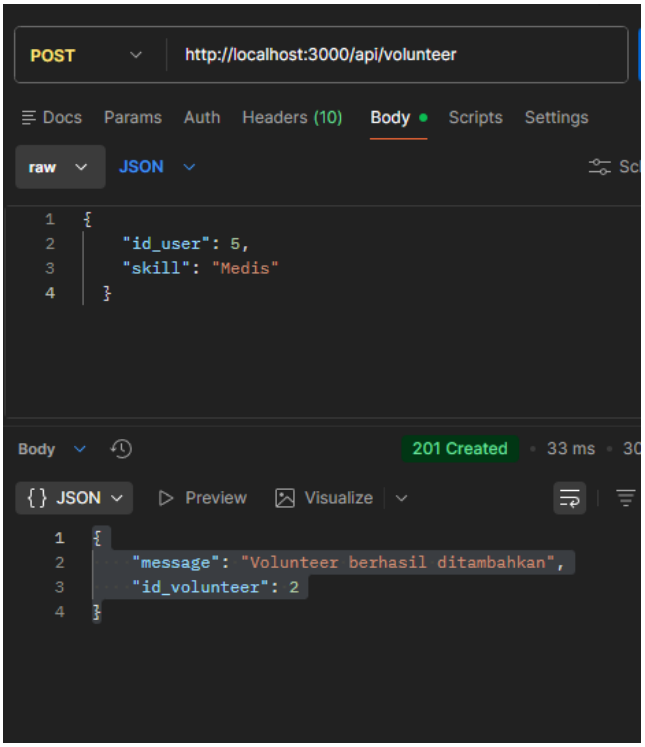


- DELETE /api/disaster/:id

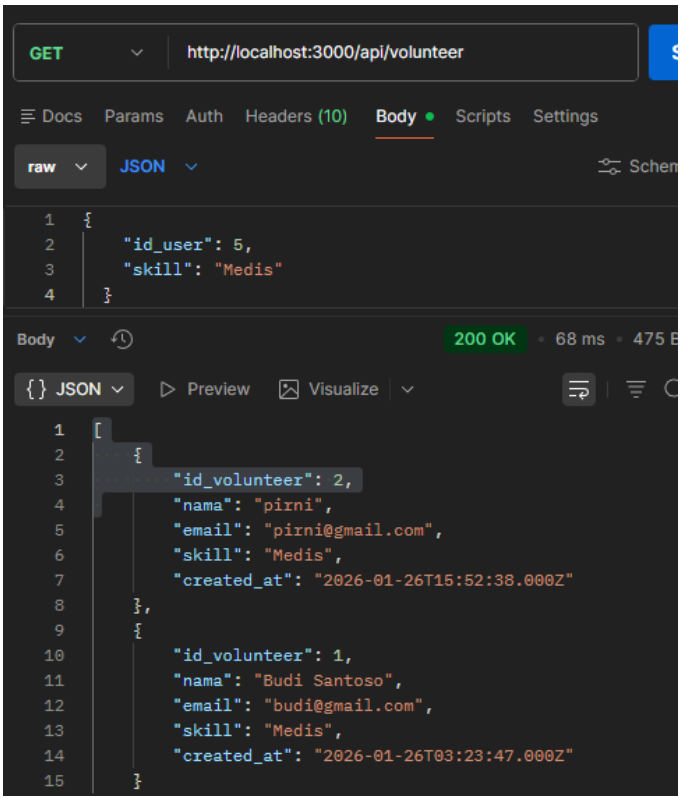


B.4 VOLUNTEER

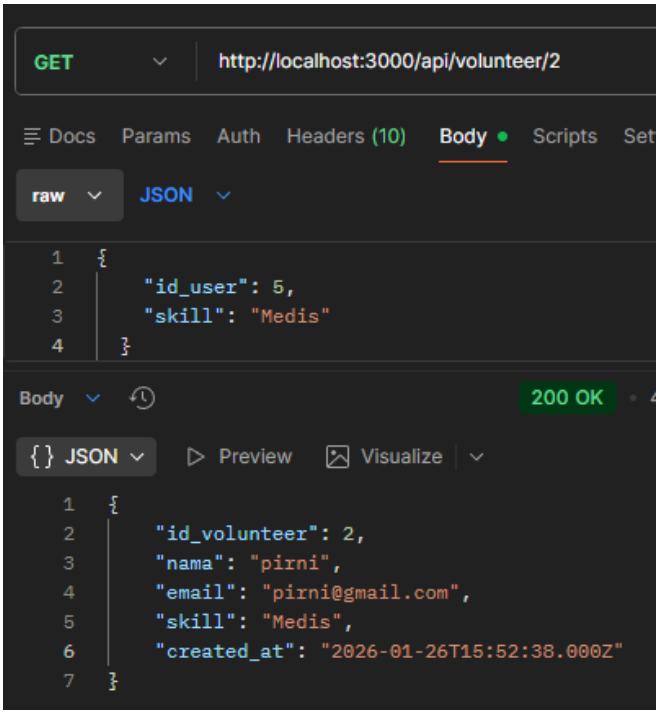
- POST /api/volunteers



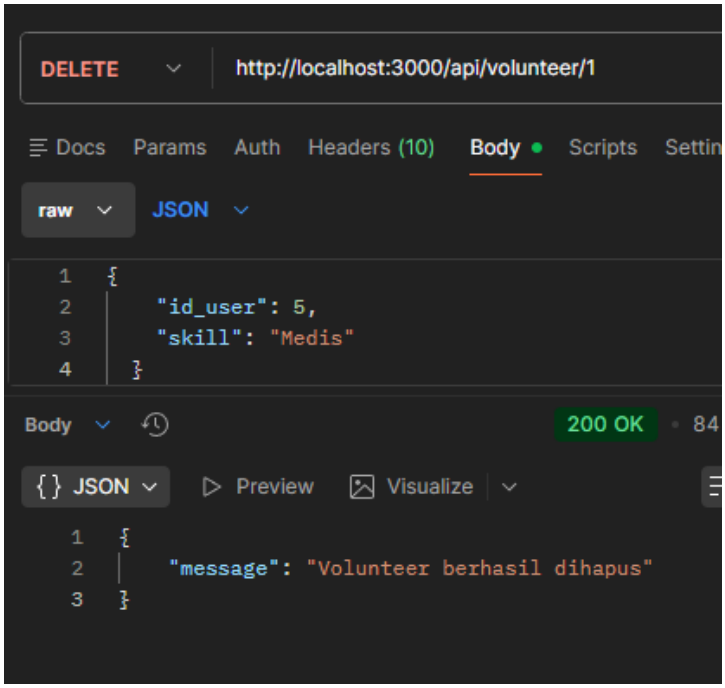
- GET /api/volunteers



- GET /api/volunteers/:id

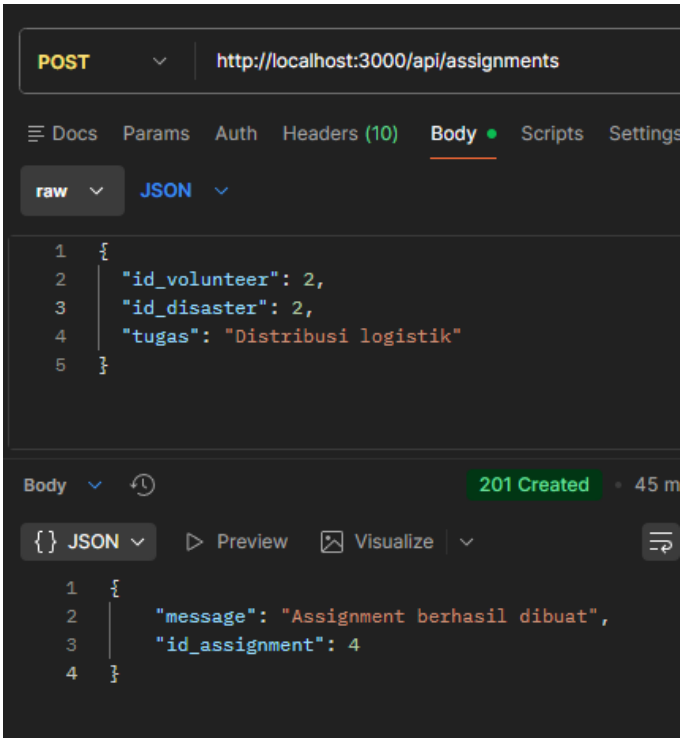


- DELETE /api/volunteers/:id

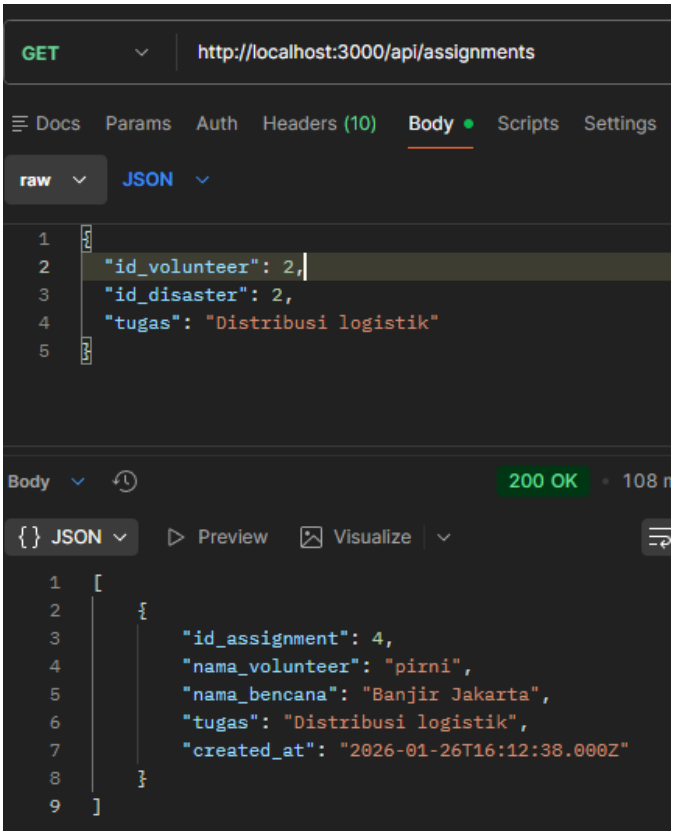


B.5 ASSIGNMENT

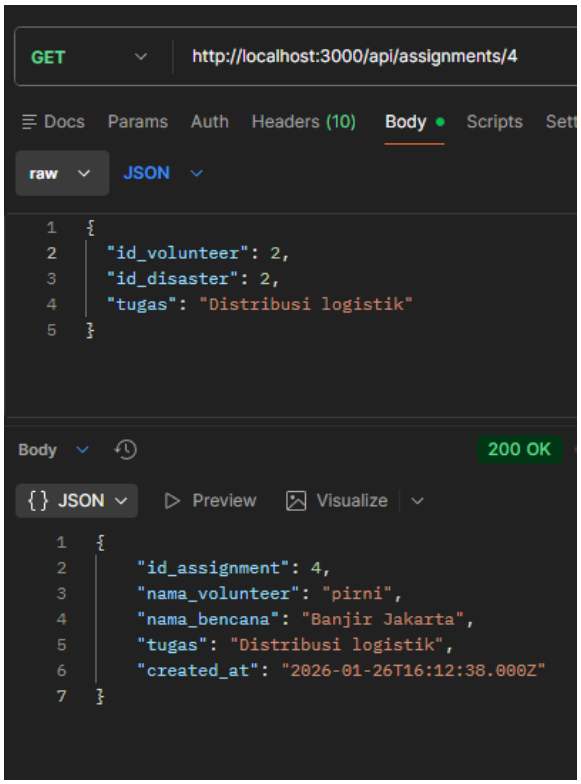
- POST /api/assignments



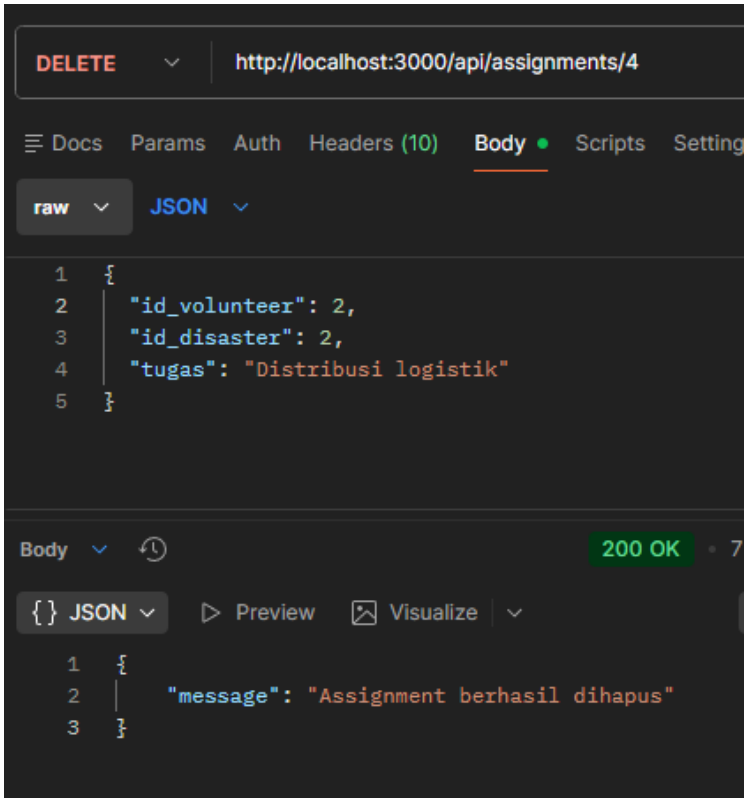
- GET /api/assignments



- GET /api/assignments/:id



- DELETE /api/assignments/:id


DELETE
http://localhost:3000/api/assignments/4
Docs
Params
Auth
Headers (10)
Body
Scripts
Setting
raw
JSON
{
"id_volunteer": 2,
"id_disaster": 2,
"tugas": "Distribusi logistik"
}
Body
200 OK
JSON
Preview
Visualize
{
"message": "Assignment berhasil dihapus"
}