

**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:**

<b>PIRNI</b>	<b>:</b> <b>20240040179</b>
<b>Neng Sahla Nurul Fauziah</b>	<b>:</b> <b>20240040130</b>
<b>Muhammad Denindra Pratama</b>	<b>:</b> <b>20240040290</b>

**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:

```
1 CREATE DATABASE smart_disaster_volunteer;
2 USE smart_disaster_volunteer;
```

### 2.1.2 Perancangan Tabel Users

Tabel Users digunakan untuk menyimpan data akun pengguna yang dapat mengakses sistem

Jalankan perintah SQL pada basis data **smart\_disaster\_volunteer**

```
1 CREATE TABLE users(
2     id_user INT AUTO_INCREMENT PRIMARY KEY,
3     nama VARCHAR(50) NOT NULL,
4     email VARCHAR(30) NOT NULL,
5     password VARCHAR(50) NOT NULL,
6     role ENUM('admin', 'koordinator',
7               'relawan') NOT NULL,
8     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
9 );
```

### 2.1.3 Tabel Disaster

Tabel disasters digunakan untuk menyimpan informasi bencana

```
1 CREATE TABLE disasters (
2     id_disaster INT AUTO_INCREMENT PRIMARY
3     KEY,
4     nama_bencana VARCHAR(100),
5     lokasi VARCHAR(100),
6     tanggal DATE,
7     status ENUM('aktif','selesai'),
8     created_at TIMESTAMP DEFAULT
9     CURRENT_TIMESTAMP
10    );
11 |
```

### 2.1.4 Tabel Volunteers

Tabel Volunteers menyimpan data relawan yang terhubung dengan akun pengguna

```
1 CREATE TABLE volunteers (
2     id_volunteer INT AUTO_INCREMENT PRIMARY
3     KEY,
4     id_user INT,
5     skill VARCHAR(100),
6     FOREIGN KEY (id_user) REFERENCES
7     users(id_user)
8 );
9 |
```

### 2.1.5 Tabel Assignments

Tabel assignments berfungsi sebagai penghubung antara relawan dan bencana

```
1 CREATE TABLE assignments (
2     id_assignment INT AUTO_INCREMENT PRIMARY
3     KEY,
4     id_volunteer INT,
5     id_disaster INT,
6     tugas VARCHAR(100),
7     FOREIGN KEY (id_volunteer) REFERENCES
8     volunteers(id_volunteer),
9     FOREIGN KEY (id_disaster) REFERENCES
9     disasters(id_disaster)
8 );
9 |
```

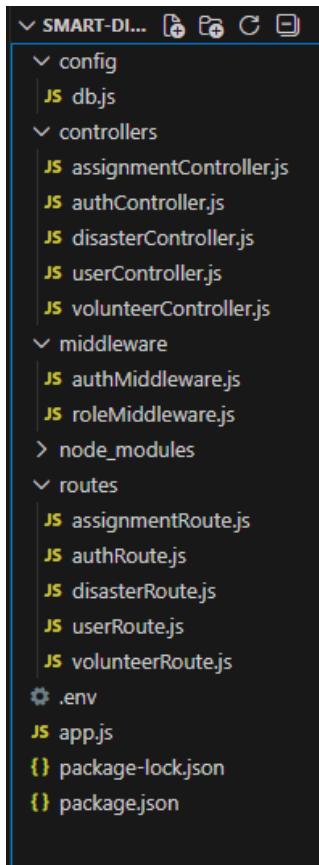
## 2.3 Struktur Folder Backend

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
- config: berisi konfigurasi database

### 2.2.1 Struktur Folder Project di Visual Code

Struktur folder backend dirancang modulr untuk memudahkan emelihraan dan pengembangan 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.



```
1 const jwt = require("jsonwebtoken");
2
3 module.exports = (req, res, next) => {
4   const token = req.headers.authorization?.split(" ")[1];
5   if (!token) return res.status(403).json({ message: "Token tidak ada" });
6
7   jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
8     if (err) return res.status(401).json({ message: "Token invalid" });
9     req.user = decoded;
10    next();
11  });
12};
13
```

## 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	Meampilkkan 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 Penugasan 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)**

A screenshot of the Postman application interface. At the top, it shows a POST method and the URL <http://localhost:3000/api/auth/register>. Below the URL, there are tabs for Docs, Params, Auth, Headers (10), Body (selected), and Scripts. Under the Body tab, there are two options: raw and JSON. The JSON option is selected, showing a JSON object with fields: nama: "Sahla", email: "sahla@gmail.com", password: "password123", and role: "relawan". Below the body, the response section shows the status 201 Created. The response body is a JSON object with a single field: message: "Register berhasil".

```
POST http://localhost:3000/api/auth/register

{
  "nama": "Sahla",
  "email": "sahla@gmail.com",
  "password": "password123",
  "role": "relawan"
}

Body 201 Created
[{"message": "Register berhasil"}]
```

### Respon Gagal (400 Bad Request)

A screenshot of the Postman application interface. At the top, it shows a POST method and the URL <http://localhost:3000/api/auth/register>. Below the URL, there are tabs for Docs, Params, Auth, Headers (10), Body (selected), and Scripts. Under the Body tab, there are two options: raw and JSON. The JSON option is selected, showing a JSON object with fields: nama: "pirni", email: "pirni@gmail.com", and password: "" (empty string). Below the body, the response section shows the status 400 Bad Request. The response body is a JSON object with a single field: message: "Data tidak lengkap".

```
POST http://localhost:3000/api/auth/register

{
  "nama": "pirni",
  "email": "pirni@gmail.com",
  "password": ""
}

Body 400 Bad Request
[{"message": "Data tidak lengkap"}]
```

### 2.8.2 Login User

Respon Berhasil (200 OK)

Body JSON

200 OK • 270 ms • 393 B

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpZF91c2VyIjoxMiwiaWF0IjoxNzY5NDQ3OTk0LCJleHAIoje3Njk1MzQ  
z0TR9.CgpvJl-Kw1JDewkTI7o3ThuIG26Y7edpa25EBHDILs8"  
3 }
```

### Respon Gagal (40 Unauthorized)

Body JSON

401 Unauthorized

Pass the correct auth cred

```
1 {  
2   "message": "Password salah"  
3 }
```

#### 2.8.3 Users API

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

	Key	Value
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZF91c2VyIjoxMiwiaWF0IjoxNzY5NDQ3OTk0LCJleHAIoje3Njk1MzQz0TR9.CgpvJl-Kw1JDewkTI7o3ThuIG26Y7edpa25EBHDILs8
<input type="checkbox"/>	Content-Type	
	Key	

#### 2.8.4 Get All Users, Users By ID dan Update Users

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

HTTP <http://localhost:3000/api/users> Save

GET <http://localhost:3000/api/users>

Headers (10) Docs Params Auth Headers (10) Body Scripts Settings

Headers 8 hidden

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1...
<input type="checkbox"/> Content-Type	application/json
Key	Value

Body 200 OK 171 ms 69

{ } JSON Preview Visualize

```
1 [  
2 {  
3   "id_user": 4,  
4   "nama": "",  
5   "email": "andi_update@gmail.com"  
6 },  
7 {  
8   "id_user": 5,  
9   "nama": "pirni",  
10  "email": "pirni@gmail.com"  
11 },  
12 ]
```

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

GET <http://localhost:3000/api/users/8>

Headers (10) Docs Params Auth Headers (10) Body Scripts

Headers 8 hidden

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIU...
<input type="checkbox"/> Content-Type	application/json
Key	Value

Body 200 OK 171 ms 69

{ } JSON Preview Visualize

```
1 {  
2   "id_user": 8,  
3   "nama": "Sahla",  
4   "email": "sahla@gmail.com"  
5 }
```

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

A screenshot of a POST request to update a user. The URL is `http://localhost:3000/api/users/8`. The request body is:

```
1 {  
2   "name": "Pirni Updated",  
3   "role": "user"  
4 }  
5
```

The response status is `200 OK`. The response body is:

```
{ } JSON ▾ ▷ Preview ⚡ Visualize | ▾  
1 {  
2   "message": "User berhasil diperbarui"  
3 }
```

### 2.8.5 Delete Users

`DELETE /api/users/:id`

**Response Berhasil (200 OK)**

A screenshot of a DELETE request to delete a user. The URL is `http://localhost:3000/api/users/9`. The request body is:

```
1 {  
2   "name": "Pirni Updated",  
3   "role": "user"  
4 }  
5
```

The response status is `200 OK`. The response body is:

```
{ } JSON ▾ ▷ Preview ⚡ Visualize | ▾  
1 {  
2   "message": "User berhasil dihapus"  
3 }
```

## 2.8.6 Disaster API

- **Create Disaster**

POST /api/disaster jika REspone berhasil aka **201 created**

The screenshot shows a Postman interface with the following details:

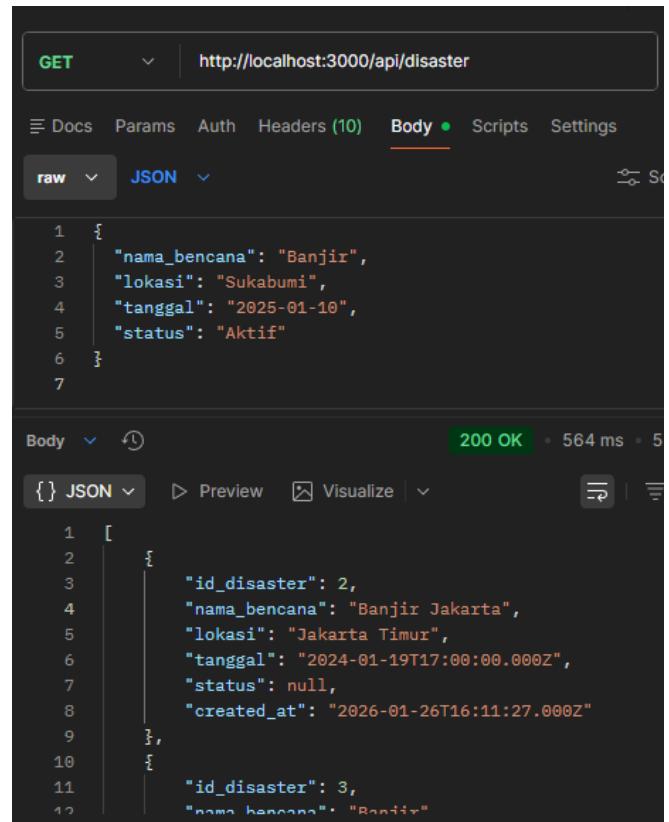
- Method:** POST
- URL:** http://localhost:3000/api/disaster
- Body:** JSON (shown in raw and JSON tabs)
- Raw Body Content:**

```
1  {
2   "nama_bencana": "Banjir",
3   "lokasi": "Sukabumi",
4   "tanggal": "2025-01-10",
5   "status": "Aktif"
6 }
7
```
- Response Status:** 201 Created
- Response Time:** 93 ms
- Response Body:** JSON (shown in raw and JSON tabs)
- Response Content:**

```
1  {
2   "message": "Data bencana berhasil ditambahkan",
3   "id_disaster": 3
4 }
```

- **Get All Disaster**

GET /api/disaster jika Respone Berhasil **200 OK**



GET <http://localhost:3000/api/disaster>

Docs Params Auth Headers (10) Body **Body** Scripts Settings

raw JSON

```
1 {  
2   "nama_bencana": "Banjir",  
3   "lokasi": "Sukabumi",  
4   "tanggal": "2025-01-10",  
5   "status": "Aktif"  
6 }  
7
```

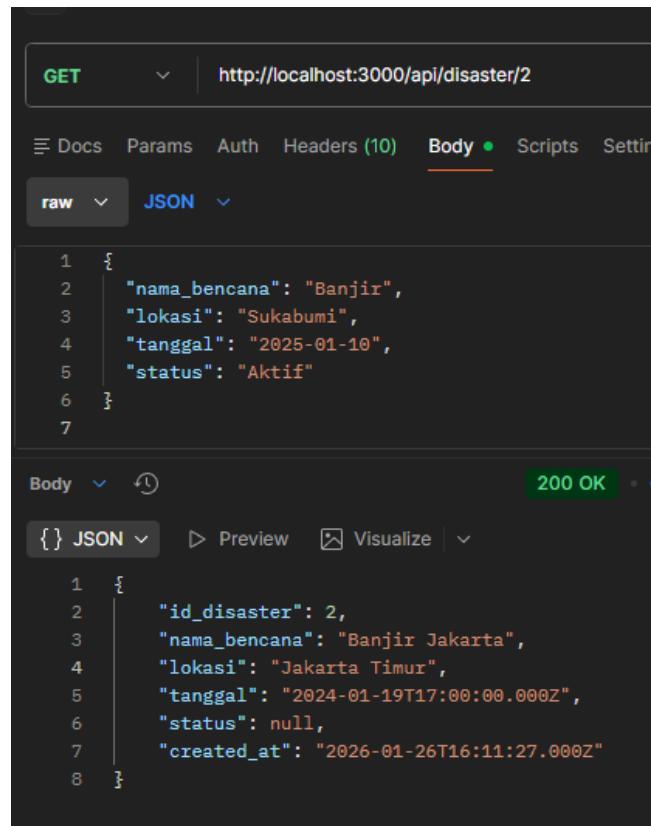
Body 200 OK 564 ms 5

{ } JSON Preview Visualize

```
[  
  {  
    "id_disaster": 2,  
    "nama_bencana": "Banjir Jakarta",  
    "lokasi": "Jakarta Timur",  
    "tanggal": "2024-01-19T17:00:00.000Z",  
    "status": null,  
    "created_at": "2026-01-26T16:11:27.000Z"  
  },  
  {  
    "id_disaster": 3,  
    "nama_bencana": "Banjir"  
  }]
```

- **GET Disaster By ID**

GET /api/disaster/:id



GET <http://localhost:3000/api/disaster/2>

Docs Params Auth Headers (10) Body **Body** Scripts Settings

raw JSON

```
1 {  
2   "nama_bencana": "Banjir",  
3   "lokasi": "Sukabumi",  
4   "tanggal": "2025-01-10",  
5   "status": "Aktif"  
6 }  
7
```

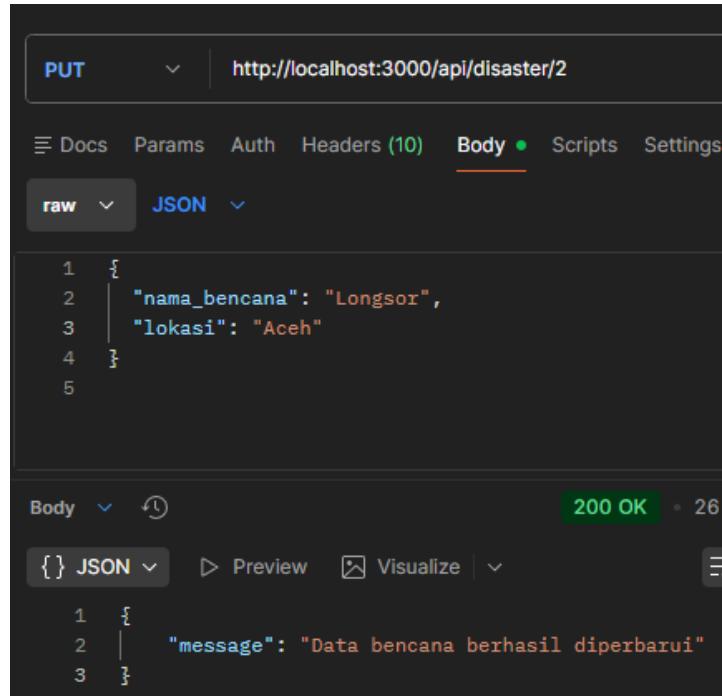
Body 200 OK

{ } JSON Preview Visualize

```
{  
  "id_disaster": 2,  
  "nama_bencana": "Banjir Jakarta",  
  "lokasi": "Jakarta Timur",  
  "tanggal": "2024-01-19T17:00:00.000Z",  
  "status": null,  
  "created_at": "2026-01-26T16:11:27.000Z"}  
}
```

- **Update Disaster**

PUT /api/disaster/:id



PUT http://localhost:3000/api/disaster/2

Body `{ "nama_bencana": "Longsor", "lokasi": "Aceh" }`

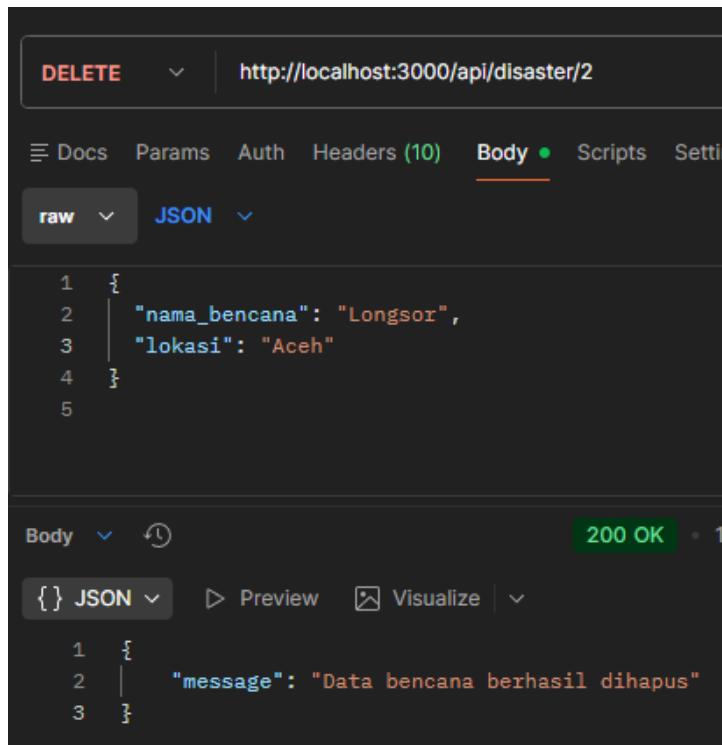
200 OK

{ } JSON

message: "Data bencana berhasil diperbarui"

- **DELETE Disaster**

DELETE /api/disaster/:id



DELETE http://localhost:3000/api/disaster/2

Body `{ "nama_bencana": "Longsor", "lokasi": "Aceh" }`

200 OK

{ } JSON

message: "Data bencana berhasil dihapus"

## 2.8.7 Volunteer API

- **Create Volunteer**

POST /api/volunteer jika berhasil Respone **201 Created**

POST http://localhost:3000/api/volunteer

Body

```
1 {  
2   "id_user": 4,  
3   "skill": "Logistik",  
4   "created_at": "2026-01-26 22:52:38"  
5 }  
6
```

201 Created

```
{ } JSON ▾ ▷ Preview Visualize | ▾  
1 {  
2   "message": "Volunteer berhasil ditambahkan",  
3   "id_volunteer": 3  
4 }
```

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

GET http://localhost:3000/api/volunteer

Body

```
1 {  
2   "id_user": 4,  
3   "skill": "Logistik",  
4   "created_at": "2026-01-26 22:52:38"  
5 }  
6
```

200 OK

```
{ } JSON ▾ ▷ Preview Visualize | ▾  
1 [  
2   {  
3     "id_volunteer": 2,  
4     "nama": "pirni",  
5     "email": "pirni@gmail.com",  
6     "skill": "Medis",  
7     "created_at": "2026-01-26T15:52:38.000Z"  
8   },  
9   {  
10    "id_volunteer": 3,  
11    "nama": ""  
12  }  
13 ]
```

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

GET http://localhost:3000/api/volunteer/2

Body (JSON)

```
1 {  
2   "id_user": 4,  
3   "skill": "Logistik",  
4   "created_at": "2026-01-26 22:52:38"  
5 }  
6
```

200 OK

{ } JSON ▾ Preview Visualize

```
1 {  
2   "id_volunteer": 2,  
3   "nama": "pirni",  
4   "email": "pirni@gmail.com",  
5   "skill": "Medis",  
6   "created_at": "2026-01-26T15:52:38.000Z"  
7 }
```

- **Delete Volunteer**

DELETE /api/volunteer/:id

DELETE http://localhost:3000/api/volunteer/3

Body (JSON)

```
1 {  
2   "id_user": 4,  
3   "skill": "Logistik",  
4   "created_at": "2026-01-26 22:52:38"  
5 }  
6
```

200 OK

{ } JSON ▾ Preview Visualize

```
1 {  
2   "message": "Volunteer berhasil dihapus"  
3 }
```

## 2.8.8 Assignment Api

- **Create Assignment**

POST /api/assignment jika Berhasil **201 Created**

POST http://localhost:3000/api/assignments

Body (10) Body **JSON**

```
1 {  
2   "id_volunteer": 2,  
3   "id_disaster": 3,  
4   "tugas": "Distribusi bantuan logistik"  
5 }  
6
```

Body (1) 201 Created

{ } JSON Preview Visualize

```
1 {  
2   "message": "Assignment berhasil dibuat",  
3   "id_assignment": 5  
4 }
```

- **Get All Assignment**

GET /api/assignments

GET http://localhost:3000/api/assignments

Headers (10) Body **JSON**

```
1 [  
2   {  
3     "id_assignment": 5,  
4     "nama_volunteer": "pirni",  
5     "nama_bencana": "Banjir",  
6     "tugas": "Distribusi bantuan logistik",  
7     "created_at": "2026-01-28T07:12:12.000Z"  
8   }  
9 ]
```

Body (1) 200 OK 59

{ } JSON Preview Visualize

- **Get Assignment By ID**

GET /api/assignment/:id

GET http://localhost:3000/api/assignments/5

Body

{} JSON

1 {  
2 "id\_assignment": 5,  
3 "nama\_volunteer": "pirni",  
4 "nama\_bencana": "Banjir",  
5 "tugas": "Distribusi bantuan logistik",  
6 "created\_at": "2026-01-28T07:12:12.000Z"  
7 }

200 OK

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

DELETE http://localhost:3000/api/assignments/5

Body

{} JSON

1 {  
2 "message": "Assignment berhasil dihapus"  
3 }

200 OK

## **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



```
● ● ●

1 require("dotenv").config();
2 const express = require("express");
3 const app = express();
4
5 app.use(express.json());
6
7 const authRoutes = require("./routes/authRoute");
8 const volunteerRoutes = require("./routes/volunteerRoute");
9 const disasterRoute = require("./routes/disasterRoute");
10 const assignmentRoute = require("./routes/assignmentRoute");
11 const userRoute = require("./routes/userRoute");
12
13 app.use("/api/auth", authRoutes);
14 app.use("/api/volunteer", volunteerRoutes);
15 app.use("/api/disaster", disasterRoute);
16 app.use("/api/assignments", assignmentRoute);
17 app.use("/api/users", userRoute);
18
19 const PORT = process.env.PORT || 3000;
20 app.listen(PORT, () => {
21   console.log(`Server berjalan di port ${PORT}`);
22 });
23
```

## A.2 Contollers

- AssignmentController.js



```
1 const db = require('../config/db');
2
3 exports.createAssignment = (req, res) => {
4   const { id_volunteer, id_disaster, tugas } = req.body;
5
6   if (!id_volunteer || !id_disaster || !tugas) {
7     return res.status(400).json({ message: 'Data assignment tidak lengkap' });
8   }
9
10  db.query(
11    'INSERT INTO assignments (id_volunteer, id_disaster, tugas) VALUES (?, ?, ?)',
12    [id_volunteer, id_disaster, tugas],
13    (err, result) => {
14      if (err) return res.status(500).json(err);
15
16      res.status(201).json({
17        message: 'Assignment berhasil dibuat',
18        id_assignment: result.insertId
19      });
20    }
21  );
22};
23
24
25 exports.getAssignments = (req, res) => {
26  db.query(
27    'SELECT
28      a.id_assignment,
29      u.nama AS nama_volunteer,
30      d.nama_bencana,
31      a.tugas,
32      a.created_at
33    FROM assignments a
34    JOIN volunteers v ON a.id_volunteer = v.id_volunteer
35    JOIN users u ON v.id_user = u.id_user
36    JOIN disasters d ON a.id_disaster = d.id_disaster',
37    (err, results) => {
38      if (err) return res.status(500).json(err);
39      res.json(results);
40    }
41  );
42};
43
44 exports.getAssignmentById = (req, res) => {
45   const { id } = req.params;
46
47   db.query(
48     'SELECT
49       a.id_assignment,
50       u.nama AS nama_volunteer,
51       d.nama_bencana,
52       a.tugas,
53       a.created_at
54     FROM assignments a
55     JOIN volunteers v ON a.id_volunteer = v.id_volunteer
56     JOIN users u ON v.id_user = u.id_user
57     JOIN disasters d ON a.id_disaster = d.id_disaster
58     WHERE a.id_assignment = ?',
59     [id],
60     (err, results) => {
61       if (err) return res.status(500).json(err);
62       if (results.length === 0) {
63         return res.status(404).json({ message: 'Assignment tidak ditemukan' });
64       }
65       res.json(results[0]);
66     }
67   );
68 };
69
70
71 exports.deleteAssignment = (req, res) => {
72   const { id } = req.params;
73
74   db.query(
75     'DELETE FROM assignments WHERE id_assignment = ?',
76     [id],
77     (err) => {
78       if (err) return res.status(500).json(err);
79       res.json({ message: 'Assignment berhasil dihapus' });
80     }
81   );
82 };
83 }
```

- **AuthController.js**



```
1 const db = require('../config/db');
2 const bcrypt = require('bcryptjs');
3 const jwt = require('jsonwebtoken');
4
5 exports.register = (req, res) => {
6   const { nama, email, password } = req.body;
7
8   if (!nama || !email || !password) {
9     return res.status(400).json({ message: 'Data tidak lengkap' });
10  }
11
12  const hashedPassword = bcrypt.hashSync(password, 10);
13
14  db.query(
15    'INSERT INTO users (nama, email, password) VALUES (?, ?, ?)',
16    [nama, email, hashedPassword],
17    (err) => {
18      if (err) return res.status(500).json(err);
19      res.status(201).json({ message: 'Register berhasil' });
20    }
21  );
22};
23
24 exports.login = (req, res) => {
25   const { email, password } = req.body;
26
27   db.query(
28     'SELECT * FROM users WHERE email = ?',
29     [email],
30     (err, results) => {
31       if (err) return res.status(500).json(err);
32       if (results.length === 0) {
33         return res.status(401).json({ message: 'Email tidak ditemukan' });
34       }
35
36       const user = results[0];
37       const isMatch = bcrypt.compareSync(password, user.password);
38
39       if (!isMatch) {
40         return res.status(401).json({ message: 'Password salah' });
41       }
42
43       const token = jwt.sign(
44         { id_user: user.id_user },
45         process.env.JWT_SECRET,
46         { expiresIn: '1d' }
47       );
48
49       res.json({ token });
50     }
51   );
52};
53
54 exports.me = (req, res) => {
55   const id_user = req.user.id_user;
56
57   db.query(
58     'SELECT id_user, nama, email FROM users WHERE id_user = ?',
59     [id_user],
60     (err, results) => {
61       if (err) return res.status(500).json(err);
62       res.json(results[0]);
63     }
64   );
65 };
66
```

- **DisasterController.js**



```
1 const db = require('../config/db');
2
3 exports.createDisaster = (req, res) => {
4   const { nama_bencana, lokasi, tanggal, status } = req.body;
5
6   db.query(
7     'INSERT INTO disasters (nama_bencana, lokasi, tanggal, status) VALUES (?, ?, ?, ?)',
8     [nama_bencana, lokasi, tanggal, status],
9     (err, result) => {
10       if (err) return res.status(500).json(err);
11       res.status(201).json({
12         message: 'Data bencana berhasil ditambahkan',
13         id_disaster: result.insertId
14       });
15     }
16   );
17 };
18
19 exports.getDisasters = (req, res) => {
20   db.query(
21     'SELECT * FROM disasters',
22     (err, results) => {
23       if (err) return res.status(500).json(err);
24       res.json(results);
25     }
26   );
27 };
28
29 exports.getDisasterById = (req, res) => {
30   const { id } = req.params;
31
32   db.query(
33     'SELECT * FROM disasters WHERE id_disaster = ?',
34     [id],
35     (err, results) => {
36       if (err) return res.status(500).json(err);
37       if (results.length === 0) {
38         return res.status(404).json({ message: 'Bencana tidak ditemukan' });
39       }
40       res.json(results[0]);
41     }
42   );
43 };
44
45 exports.updateDisaster = (req, res) => {
46   const { id } = req.params;
47   const { nama_bencana, lokasi, tanggal, status } = req.body;
48
49   db.query(
50     'UPDATE disasters SET nama_bencana=?, lokasi=?, tanggal=?, status=? WHERE id_disaster=?',
51     [nama_bencana, lokasi, tanggal, status, id],
52     (err) => {
53       if (err) return res.status(500).json(err);
54       res.json({ message: 'Data bencana berhasil diperbarui' });
55     }
56   );
57 };
58
59
60 exports.deleteDisaster = (req, res) => {
61   const { id } = req.params;
62
63   db.query(
64     'DELETE FROM disasters WHERE id_disaster = ?',
65     [id],
66     (err) => {
67       if (err) return res.status(500).json(err);
68       res.json({ message: 'Data bencana berhasil dihapus' });
69     }
70   );
71 };
72 }
```

- **VolunteerController.js**

```
1 const db = require('../config/db');
2
3 exports.createVolunteer = (req, res) => {
4     const { id_user, skill } = req.body;
5
6     if (!id_user || !skill) {
7         return res.status(400).json({ message: 'id_user dan skill wajib diisi' });
8     }
9
10    db.query(
11        'INSERT INTO volunteers (id_user, skill) VALUES (?, ?)',
12        [id_user, skill],
13        (err, result) => {
14            if (err) return res.status(500).json(err);
15            res.status(201).json({
16                message: 'Volunteer berhasil ditambahkan',
17                id_volunteer: result.insertId
18            });
19        }
20    );
21 };
22
23 exports.getVolunteers = (req, res) => {
24     db.query(
25         'SELECT v.id_volunteer, u.nama, u.email, v.skill, v.created_at
26         FROM volunteers v
27         JOIN users u ON v.id_user = u.id_user',
28         (err, results) => {
29             if (err) return res.status(500).json(err);
30             res.json(results);
31         }
32     );
33 };
34
35 exports.getVolunteerById = (req, res) => {
36     const { id } = req.params;
37
38     db.query(
39         'SELECT v.id_volunteer, u.nama, u.email, v.skill, v.created_at
40         FROM volunteers v
41         JOIN users u ON v.id_user = u.id_user
42         WHERE v.id_volunteer = ?',
43         [id],
44         (err, results) => {
45             if (err) return res.status(500).json(err);
46             if (results.length === 0) {
47                 return res.status(404).json({ message: 'Volunteer tidak ditemukan' });
48             }
49             res.json(results[0]);
50         }
51     );
52 };
53
54 exports.deleteVolunteer = (req, res) => {
55     const { id } = req.params;
56
57     db.query(
58         'DELETE FROM volunteers WHERE id_volunteer = ?',
59         [id],
60         (err) => {
61             if (err) return res.status(500).json(err);
62             res.json({ message: 'Volunteer berhasil dihapus' });
63         }
64     );
65 };
66
```

- **UsersController.js**



```
1 const db = require('../config/db');
2
3 exports.getUsers = (req, res) => {
4     db.query(
5         'SELECT id_user, nama, email FROM users',
6         (err, results) => {
7             if (err) return res.status(500).json(err);
8             res.json(results);
9         }
10    );
11 };
12
13 exports.getUserById = (req, res) => {
14     const { id } = req.params;
15
16     db.query(
17         'SELECT id_user, nama, email FROM users WHERE id_user = ?',
18         [id],
19         (err, results) => {
20             if (err) return res.status(500).json(err);
21             if (results.length === 0) {
22                 return res.status(404).json({ message: 'User tidak ditemukan' });
23             }
24             res.json(results[0]);
25         }
26    );
27 };
28
29 exports.updateUser = (req, res) => {
30     const { id } = req.params;
31     const { name, email } = req.body;
32
33     db.query(
34         'UPDATE users SET nama = ?, email = ? WHERE id_user = ?',
35         [name, email, id],
36         (err) => {
37             if (err) return res.status(500).json(err);
38             res.json({ message: 'User berhasil diperbarui' });
39         }
40    );
41 };
42
43 exports.deleteUser = (req, res) => {
44     const { id } = req.params;
45
46     db.query(
47         'DELETE FROM users WHERE id_user = ?',
48         [id],
49         (err) => {
50             if (err) return res.status(500).json(err);
51             res.json({ message: 'User berhasil dihapus' });
52         }
53    );
54};
```

### A.3 Middleware

- **AuthMiddleware.js**

```
● ● ●
```

```
1 const jwt = require("jsonwebtoken");
2
3 module.exports = (req, res, next) => {
4     const token = req.headers.authorization?.split(" ")[1];
5     if (!token) return res.status(403).json({ message: "Token tidak ada" });
6
7     jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
8         if (err) return res.status(401).json({ message: "Token invalid" });
9         req.user = decoded;
10        next();
11    });
12};
13
```

- **roleMiddleware.js**

```
● ● ●
```

```
1 module.exports = (role) => {
2     return (req, res, next) => {
3         if (req.user.role !== role) {
4             return res.status(403).json({
5                 message: "Akses ditolak"
6             });
7         }
8         next();
9     };
10};
11
```

## A.4 Routes

- AssignmentRoute.js

```
● ● ●
1 const express = require('express');
2 const router = express.Router();
3 const assignmentController = require('../controllers/assignmentController');
4 const authMiddleware = require('../middleware/authMiddleware');
5
6 router.post('/', authMiddleware, assignmentController.createAssignment);
7 router.get('/', authMiddleware, assignmentController.getAssignments);
8 router.get('/:id', authMiddleware, assignmentController.getAssignmentById);
9 router.delete('/:id', authMiddleware, assignmentController.deleteAssignment);
10
11 module.exports = router;
12
```

- AuthRoutes.js

```
● ● ●
1 const express = require('express');
2 const router = express.Router();
3 const authController = require('../controllers/authController');
4 const authMiddleware = require('../middleware/authMiddleware');
5
6 router.post('/register', authController.register);
7 router.post('/login', authController.login);
8 router.get('/me', authMiddleware, authController.me);
9
10 module.exports = router;
```

- Disasterroute.js

```
● ● ●
1 const express = require('express');
2 const router = express.Router();
3 const disasterController = require('../controllers/disasterController');
4 const authMiddleware = require('../middleware/authMiddleware');
5
6 router.post('/', authMiddleware, disasterController.createDisaster);
7 router.get('/', authMiddleware, disasterController.getDisasters);
8 router.get('/:id', authMiddleware, disasterController.getDisasterById);
9 router.put('/:id', authMiddleware, disasterController.updateDisaster);
10 router.delete('/:id', authMiddleware, disasterController.deleteDisaster);
11
12 module.exports = router;
13
```

- **Volunteerroute.js**

```
● ● ●

1 const express = require('express');
2 const router = express.Router();
3 const volunteerController = require('../controllers/volunteerController');
4 const authMiddleware = require('../middleware/authMiddleware');
5
6 router.post('/', authMiddleware, volunteerController.createVolunteer);
7 router.get('/', authMiddleware, volunteerController.getVolunteers);
8 router.get('/:id', authMiddleware, volunteerController.getVolunteerById);
9 router.delete('/:id', authMiddleware, volunteerController.deleteVolunteer);
10
11 module.exports = router;
```

- **Usersroute.js**

```
● ● ●

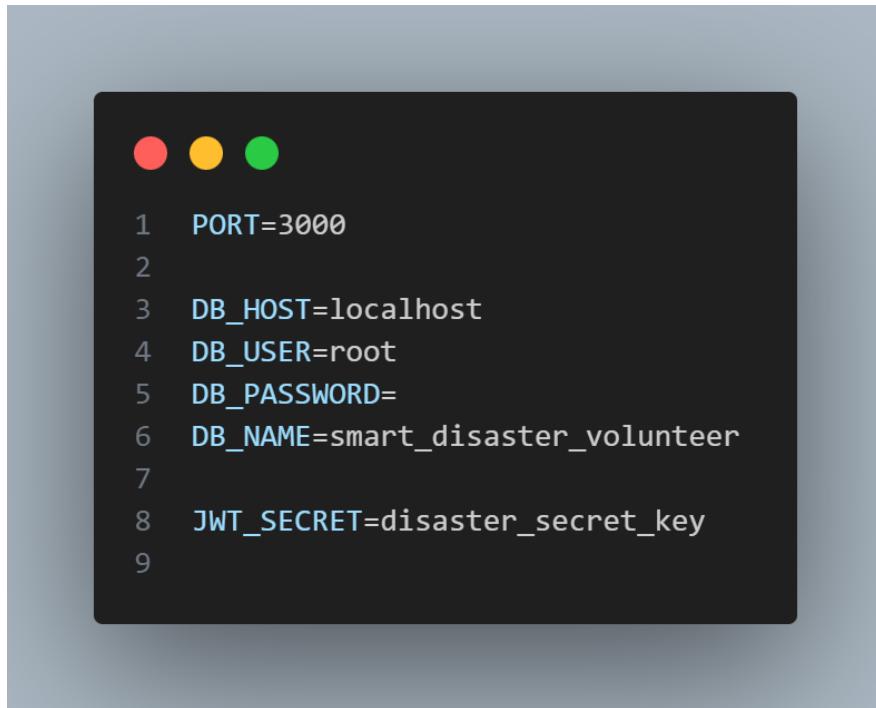
1 const express = require('express');
2 const router = express.Router();
3 const userController = require('../controllers/userController');
4 const authMiddleware = require('../middleware/authMiddleware');
5
6 router.get('/', authMiddleware, userController getUsers);
7 router.get('/:id', authMiddleware, userController.getUserById);
8 router.put('/:id', authMiddleware, userController.updateUser);
9 router.delete('/:id', authMiddleware, userController.deleteUser);
10
11 module.exports = router;
```

## A.5 db.js

```
● ● ●

1 const mysql = require("mysql");
2
3 const db = mysql.createConnection({
4   host: "localhost",
5   user: "root",
6   password: "",
7   database: "smart_disaster_volunteer"
8 });
9
10 db.connect(err => {
11   if (err) {
12     console.log("Database connection failed:", err);
13   } else {
14     console.log("Database connected");
15   }
16 });
17
18 module.exports = db;
```

## A.6 .env

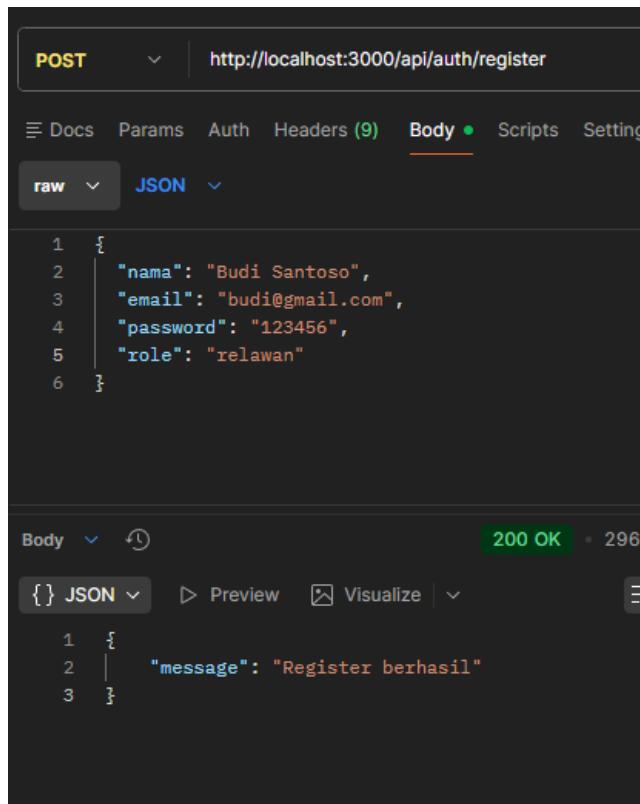


```
1 PORT=3000
2
3 DB_HOST=localhost
4 DB_USER=root
5 DB_PASSWORD=
6 DB_NAME=smart_disaster_volunteer
7
8 JWT_SECRET=disaster_secret_key
9
```

## Lampiran B

### B.1 AUTH

- POST /api/auth/register



POST http://localhost:3000/api/auth/register

Body (JSON)

```
1 {
2   "nama": "Budi Santoso",
3   "email": "budi@gmail.com",
4   "password": "123456",
5   "role": "relawan"
6 }
```

200 OK

```
1 {
2   "message": "Register berhasil"
3 }
```

- POST /api/auth/login

The screenshot shows a Postman interface with a POST request to `http://localhost:3000/api/auth/login`. The request body is a JSON object with `"email": "budi@gmail.com"` and `"password": "123456"`. The response status is `200 OK` with a response time of 188 ms and a response size of 435 B. The response body contains a message "Login berhasil" and a token.

```
POST http://localhost:3000/api/auth/login
{
  "email": "budi@gmail.com",
  "password": "123456"
}

200 OK
{
  "message": "Login berhasil",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Niwicm9sZSI6InJlbGF3YW4iLCJpYXQiOjE3NjkzOTQ5NDUsImV4cCI6MTc2OTQ4MTM0NX0.0gXw8ZdJvxptqU2zRgSLDR-NH2zdw4ty_3-fFarKeSA"
}
```

- GET /api/auth/me

The screenshot shows a Postman interface with a GET request to `http://localhost:3000/api/auth/me`. The request includes an `Authorization` header with the value `Bearer eyJhbGciOiJIUzI1...` and a `Content-Type` header with the value `application/json`. The response status is `200 OK` with a response time of 99 ms. The response body is empty.

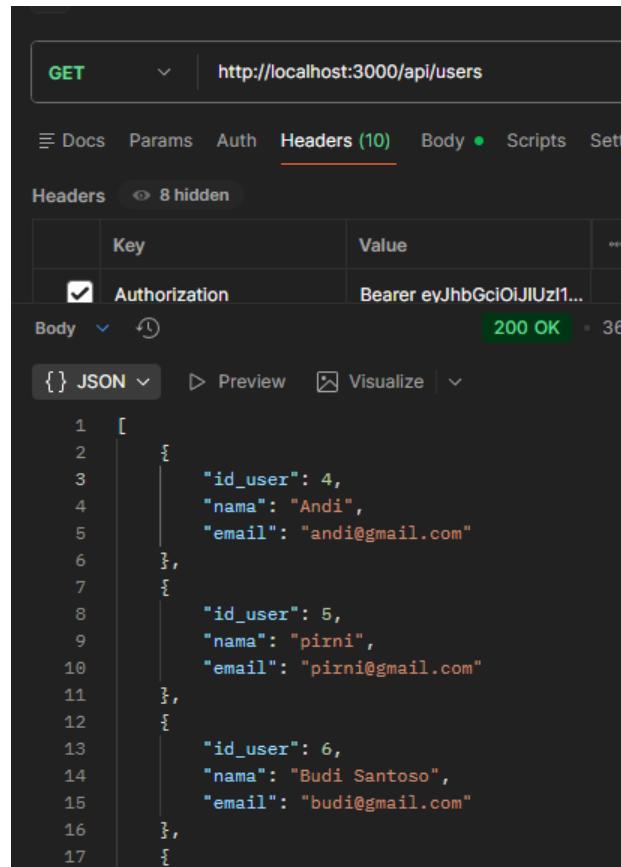
Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1...
Content-Type	application/json

```
GET http://localhost:3000/api/auth/me
Authorization: Bearer eyJhbGciOiJIUzI1...
Content-Type: application/json

200 OK
{ } JSON
```

## B.2 USERS

- GET /api/users



GET http://localhost:3000/api/users

Headers (10)

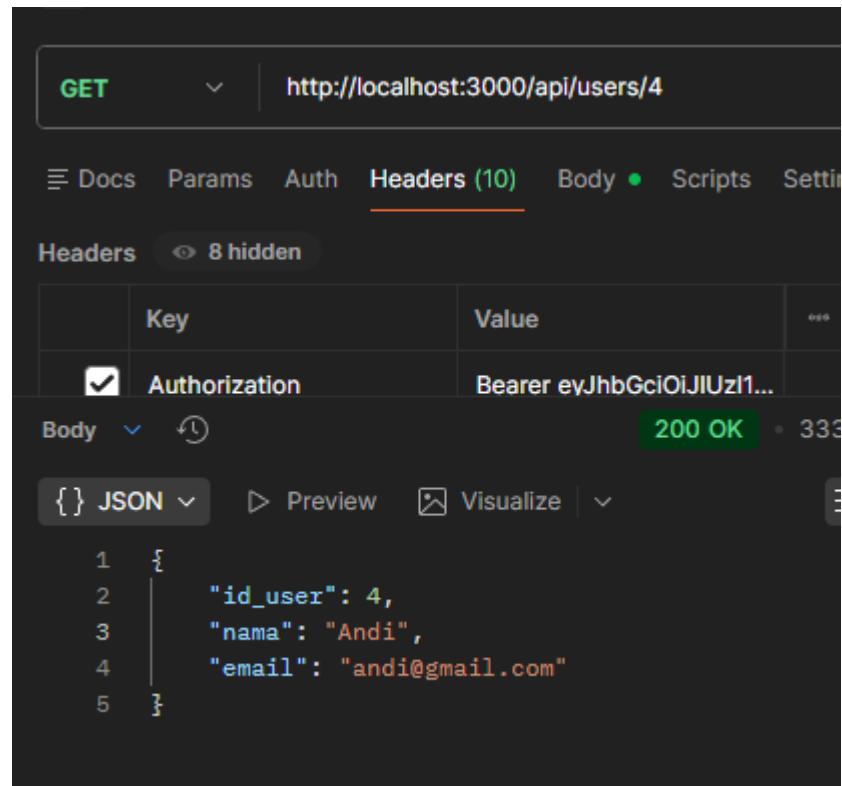
Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1...

200 OK • 36

{ } JSON ▾ Preview Visualize

```
1 [  
2 {  
3   "id_user": 4,  
4   "nama": "Andi",  
5   "email": "andi@gmail.com"  
6 },  
7 {  
8   "id_user": 5,  
9   "nama": "pirni",  
10  "email": "pirni@gmail.com"  
11 },  
12 {  
13   "id_user": 6,  
14   "nama": "Budi Santoso",  
15   "email": "budi@gmail.com"  
16 },  
17 {
```

- GET /api/users/:id



GET http://localhost:3000/api/users/4

Headers (10)

Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1...

200 OK • 333

{ } JSON ▾ Preview Visualize

```
1 {  
2   "id_user": 4,  
3   "nama": "Andi",  
4   "email": "andi@gmail.com"  
5 }
```

- PUT /api/users/:id

The screenshot shows a POSTMAN interface. The top bar indicates a **PUT** method and the URL **http://localhost:3000/api/users/4**. The **Body** tab is selected, showing a JSON payload:

```
1 {  
2   "nama": "Andi Update",  
3   "email": "andi_update@gmail.com"  
4 }
```

Below the request, the response status is **200 OK** with a response time of **154 ms**. The response body is:

```
1 {  
2   "message": "User berhasil diperbarui"  
3 }
```

- **DELETE /api/users/:id**

The screenshot shows a POSTMAN interface. The top bar indicates a **DELETE** method and the URL **http://localhost:3000/api/users/7**. The **Body** tab is selected, showing a JSON payload identical to the PUT request:

```
1 {  
2   "nama": "Andi Update",  
3   "email": "andi_update@gmail.com"  
4 }
```

Below the request, the response status is **200 OK** with a response time of **86 ms**. The response body is:

```
1 {  
2   "message": "User berhasil dihapus"  
3 }
```

### B.3 DISASTERS

- **POST /api/disaster**

POST http://localhost:3000/api/disaster

Body (11) Body (JSON)

```
1 {  
2   "nama_bencana": "Banjir",  
3   "lokasi": "Jakarta Selatan",  
4   "tanggal": "2026-01-20"  
5 }
```

200 OK

{ } JSON ▾ Preview Visualize

```
1 {  
2   "message": "Create disaster - OK"  
3 }
```

- GET /api/disaster

GET http://localhost:3000/api/disaster

Headers (8) Headers (6 hidden)

Key	Value	Bulk Ed
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1...	
<input type="checkbox"/> Content-Type	application/json	
Key	Value	Description

200 OK

{ } JSON ▾ Preview Visualize

```
1 [  
2   {  
3     "id_disaster": 1,  
4     "nama_bencana": "Gempa Bumi",  
5     "lokasi": "Cianjur",  
6     "tanggal": "2025-01-11T17:00:00.000Z",  
7     "status": null,  
8     "created_at": "2026-01-25T16:36:28.000Z"  
9   }  
10 ]
```

- GET /api/disaster/:id

GET http://localhost:3000/api/disaster/1

Headers (8)

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1...
<input type="checkbox"/> Content-Type	application/json
Key	Value

Body 200 OK

```
{
  "id_disaster": 1,
  "nama_bencana": "Gempa Bumi",
  "lokasi": "Cianjur",
  "tanggal": "2025-01-11T17:00:00.000Z",
  "status": null,
  "created_at": "2026-01-25T16:36:28.000Z"
}
```

- PUT /api/disaster/:id

PUT http://localhost:3000/api/disaster/1

Headers (10)

raw JSON

```
{
  "nama_bencana": "Banjir Besar",
  "lokasi": "Jakarta Timur",
  "tanggal": "2024-01-21",
  "status": "Terkendali"
}
```

Body 200 OK

```
{
  "message": "Data bencana berhasil diperbarui"
}
```

- DELETE /api/disaster/:id

DELETE http://localhost:3000/api/disaster/1

Body

```

1 {
2   "nama_bencana": "Banjir Besar",
3   "lokasi": "Jakarta Timur",
4   "tanggal": "2024-01-21",
5   "status": "Terkendali"
6 }

```

200 OK

```

1 {}
2   "message": "Data bencana berhasil dihapus"
3

```

## B.4 VOLUNTEER

- POST /api/volunteers

POST http://localhost:3000/api/volunteer

Body

```

1 {
2   "id_user": 5,
3   "skill": "Medis"
4 }

```

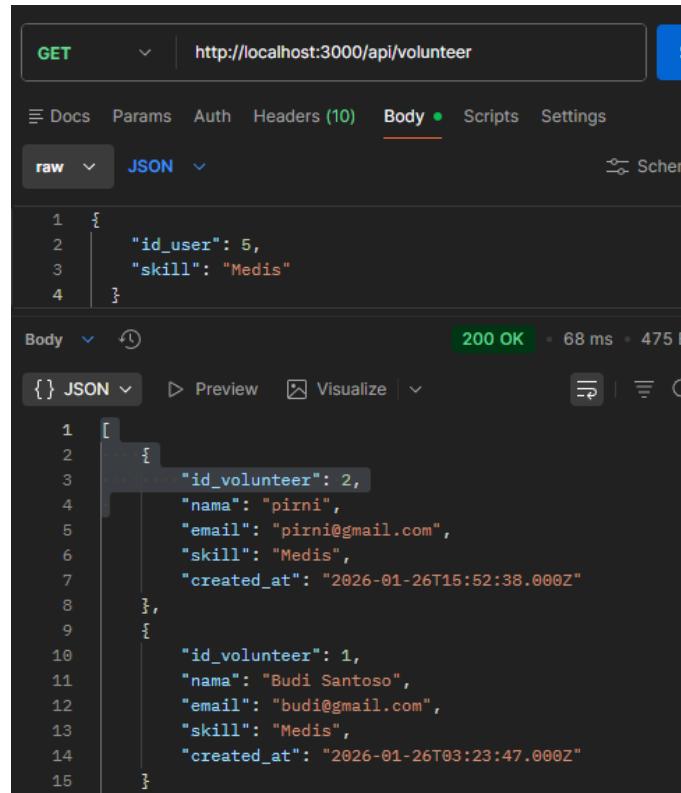
201 Created

```

1 {}
2   "message": "Volunteer berhasil ditambahkan",
3   "id_volunteer": 2
4

```

- GET /api/volunteers



POST http://localhost:3000/api/volunteer

Body

```

1 {
2   "id_user": 5,
3   "skill": "Medis"
4 }

```

200 OK

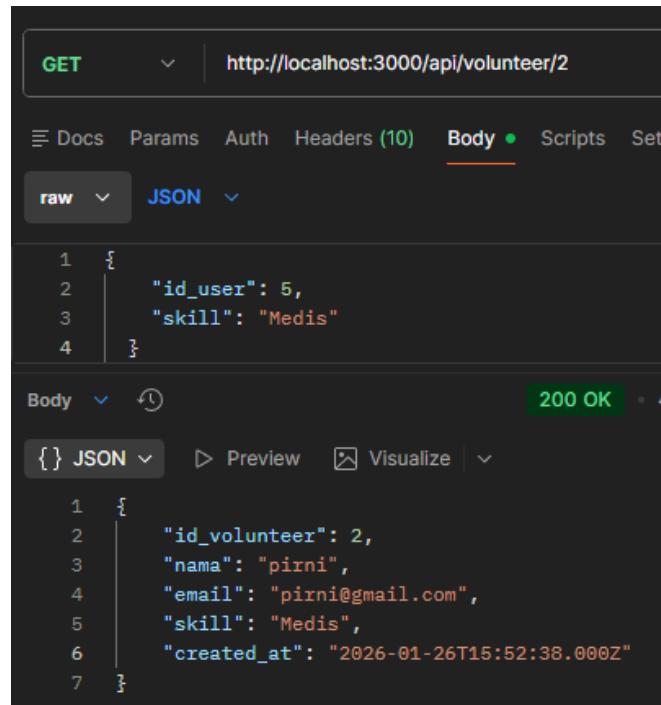
Body

```

1 [
2   {
3     "id_volunteer": 2,
4     "nama": "pirni",
5     "email": "pirni@gmail.com",
6     "skill": "Medis",
7     "created_at": "2026-01-26T15:52:38.000Z"
8   },
9   {
10    "id_volunteer": 1,
11    "nama": "Budi Santoso",
12    "email": "budi@gmail.com",
13    "skill": "Medis",
14    "created_at": "2026-01-26T03:23:47.000Z"
15  }
]

```

- GET /api/volunteers/:id



GET http://localhost:3000/api/volunteer/2

Body

```

1 {
2   "id_user": 5,
3   "skill": "Medis"
4 }

```

200 OK

Body

```

1 {
2   "id_volunteer": 2,
3   "nama": "pirni",
4   "email": "pirni@gmail.com",
5   "skill": "Medis",
6   "created_at": "2026-01-26T15:52:38.000Z"
7 }

```

- DELETE /api/volunteers/:id

DELETE http://localhost:3000/api/volunteer/1

Body

```
1 {  
2   "id_user": 5,  
3   "skill": "Medis"  
4 }
```

200 OK

```
{ } JSON ▾ ▷ Preview Visualize
```

```
1 {  
2   "message": "Volunteer berhasil dihapus"  
3 }
```

## B.5 ASSIGNMENT

- POST /api/assignments

POST http://localhost:3000/api/assignments

Body

```
1 {  
2   "id_volunteer": 2,  
3   "id_disaster": 2,  
4   "tugas": "Distribusi logistik"  
5 }
```

201 Created

```
{ } JSON ▾ ▷ Preview Visualize
```

```
1 {  
2   "message": "Assignment berhasil dibuat",  
3   "id_assignment": 4  
4 }
```

- GET /api/assignments

GET http://localhost:3000/api/assignments

Body raw JSON

```
1 "id_volunteer": 2,
2 "id_disaster": 2,
3 "tugas": "Distribusi logistik"
```

Body raw JSON

200 OK

```
[{"id_assignment": 4, "nama_volunteer": "pirni", "nama_bencana": "Banjir Jakarta", "tugas": "Distribusi logistik", "created_at": "2026-01-26T16:12:38.000Z"}]
```

- GET /api/assignments/:id

GET http://localhost:3000/api/assignments/4

Body raw JSON

```
{ "id_volunteer": 2, "id_disaster": 2, "tugas": "Distribusi logistik"}
```

Body raw JSON

200 OK

```
{"id_assignment": 4, "nama_volunteer": "pirni", "nama_bencana": "Banjir Jakarta", "tugas": "Distribusi logistik", "created_at": "2026-01-26T16:12:38.000Z"}
```

- DELETE /api/assignments/:id

**DELETE** | <http://localhost:3000/api/assignments/4>

Docs Params Auth Headers (10) **Body** • Scripts Setting

raw JSON

```
1  {
2    "id_volunteer": 2,
3    "id_disaster": 2,
4    "tugas": "Distribusi logistik"
5 }
```

Body 200 OK 7

{ } JSON Preview Visualize

```
1  {
2    "message": "Assignment berhasil dihapus"
3 }
```