

LAPORAN
RANCANGAN REST API



TEMA:
SISTEM ASURANSI

Dirancang oleh:

Regal Nugraha (20230040175)

Muhammad Ibdaul M. (20230040291)

Eneng Safitri (20230040010)

UNIVERSITAS NUSA PUTRA
JURUSAN TEKNIK INFORMATIKA 2024/2025

PENDAHULUAN

Segala puji hanya milik Allah SWT yang senantiasa meridhoi apa yang hamba-Nya kerjakan dengan baik, sehingga kami dapat menyelesaikan tugas Akhir yakni laporan "Perancangan REST API pada sisitem Asuransi" yang diberikan oleh dosen pengampu mata kuliah Pemograman Berbasis Platform

Shalawat serta salam kami panjatkan kepada Nabi besar kita Muhammad SAW beserta para sahabat, keluarga, dan umatnya hingga akhir zaman.

laporan ini kami buat dengan harapan dapat bermanfaat bagi para sahabat semua dan juga bagi diri kami sendiri.

Selanjutnya kami mohon maaf atas laporan yang kami buat, apabila terdapat kesalahan dalam pembahasan maupun penulisan. Untuk itu kami mengharapkan kritik dan saran yang membangun dari para pembaca demi kesempurnaan laporan ini.

A. Latar Belakang

Sistem asuransi modern membutuhkan teknologi untuk meningkatkan efisiensi dalam pengelolaan data pengguna, polis, klaim, dan pembayaran premi. REST API menjadi solusi untuk memfasilitasi integrasi dan aksesibilitas data tersebut secara cepat dan aman

B. Tujuan

Merancang REST API untuk sistem asuransi dengan struktur database yang terdiri dari empat tabel utama: users, policies, claims, dan payments. API ini dirancang agar dapat menangani pengelolaan data asuransi secara efisien.

LAPORAN HASIL

A. Analisis Kebutuhan

1. Kebutuhan Fungsional

- Autentikasi dan otorisasi pengguna (login/logout).
- CRUD data pengguna (users).
- CRUD data polis (policies).
- CRUD data klaim (claims).
- CRUD data pembayaran (payments).
- Relasi data antar tabel sesuai dengan kebutuhan bisnis.

2. Kebutuhan Non-Fungsional

- Keamanan: JWT untuk autentikasi.
- Performa: Implementasi caching menggunakan Node-Cache.
- Format Data: JSON.
- Pengujian: Menggunakan Postman.

B. Desain Database

tabel 1: data pengguna/pemegang polis				
Users				
fields	Type data	Nullable	Keterangan	
id	INT	Not Null	Primay Key, Auto Increment	
username	VARCHAR(255)	Not Null	nama pengguna	
password	VARCHAR(255)	Not Null	kata sandi	
email	VARCHAR(50)	Not Null	email pengguna	
phone	VARCHAR(15)	Not Null	nomor telepon pengguna	
address	TEXT	Null	alamat lengkap pengguna	
tabel 2: data polis asuransi				
policies				
fields	Type data	Nullable	Keterangan	
id	INT	Not Null	Primary key, Auto Increment	
user_id	INT	Not Null	foreign key ke users	
policy_number	VARCHAR(255)	Not Null	nomor jenis asuransi	
policy_type	VARCHAR(255)	Not Null	jenis asuransi	
premium	DECIMAL(10,2)	Not Null	jumlah premi per bulan	
start_date	DATE	Not Null	tanggal mulai polis	
end_date	DATE	Not Null	tanggal akhir polis	
created_at	DATETIME	Not Null Default Now()	Tanggal dan waktu saat entri dibuat	
relasi: user_id di policies merujuk ke id di users				
tabel 3: data klaim asuransi				
claims				
fields	Type data	Nullable	Keterangan	
id	INT	Not Null	Primary key, Auto Increment	
policy_id	INT	Not Null	foreign key ke policies	
claim_amount	DECIMAL(10,2)	Not Null	jumlah klaim	
claim_date	DATE	Not Null	Tanggal klaim diajukan	
status	ENUM('pending','approved','rejected')	Not Null	status klaim (pending, approved, rejected)	
created_at	DATETIME	Not Null Default Now()	Tanggal dan waktu saat entri dibuat	
relasi: policy_id di claims merujuk ke policy_id di policies				
tabel 4: data pembayaran premi				
payments				
fields	Type data	Nullable	Keterangan	
id	INT	Not Null	Primary key, Auto Increment	
policy_id	INT	Not Null	foreign key ke policies	
amount	DECIMAL(10,2)	Not Null	jumlah yang dibayarkan	
payment_date	DATE	Not Null	tanggal pembayaran	
payment_method	ENUM('credit_card','bank_transfer')	Not Null	Metode pembayaran(Transfer, credit card, dll)	
created_at	DATETIME	Not Null Default Now()	Tanggal dan waktu saat entri dibuat	
relasi: policy_id di payments merujuk ke policy_id di policies				
1. users → policies: One-to-Many (Satu pengguna bisa memiliki banyak polis).				
2. policies → claims: One-to-Many (Satu polis bisa memiliki banyak klaim).				
3. policies → payments: One-to-Many (Satu polis bisa memiliki banyak pembayaran).				

C. Endpoints

Endpoint	HTTP Method	Description	Parameter	Request Body	Request Header
/api/users/register	POST	Mendaftarkan pengguna baru	-	{ "username": "regal.nugraha", "email": "regal@gmail.com", "phone": "085794929291", "password": "12345" }	-
/api/users/login	GET	Login pengguna	-	{ "username": "regal.nugraha", "password": "12345" }	-
/api/users	GET	Mendapatkan daftar pengguna	-	-	Authorization: Bearer <token_jwt>
/api/users/:id	GET	Mendapatkan detail pengguna	id(int)	-	Authorization: Bearer <token_jwt>
/api/users/:id	PUT	Memperbarui data pengguna	id(int)	{ "username": "regal.nugraha", "email": "r@gmail.com", "phone": "updatephone" }	Authorization: Bearer <token_jwt>
/api/users/:id	DELETE	Menghapus pengguna	id(int)	-	Authorization: Bearer <token_jwt>
/api/policies	POST	Membuat polis baru	-	{ "policy_number": "PN1234567", "policy_type": "Health", "premium": 40000, "start_date": "2025-01-11", "end_date": "2028-01-11", "user_id": "3" }	Authorization: Bearer <token_jwt>
/api/policies	GET	Mendapatkan Daftar Polis	-	-	Authorization: Bearer <token_jwt>
/api/policies/:id	GET	Mendapatkan Detail Polis	id(int)	-	Authorization: Bearer <token_jwt>
/api/policies/:id	PUT	Memperbarui Polis	id(int)	{ "policy_number": "POL654321", "policy_type": "Life", "premium": 600.00, "start_date": "2023-02-01", "end_date": "2024-02-01" }	Authorization: Bearer <token_jwt>
/api/policies/:id	DELETE	Menghapus Polis	id(int)	-	Authorization: Bearer <token_jwt>
/api/claims	POST	Mengajukan Klaim	-	{ "policy_id": 1, "claim_amount": 1000.00, "claim_date": "2023-01-01", "status": "approved" }	Authorization: Bearer <token_jwt>
/api/claims	GET	Mendapatkan Daftar Klaim	-	-	Authorization: Bearer <token_jwt>
/api/claims/:id	GET	Mendapatkan Detail Klaim	id(int)	-	Authorization: Bearer <token_jwt>
/api/claims/:id	PUT	Memperbarui Status Klaim	id(int)	{ "status": "approved" }	Authorization: Bearer <token_jwt>
/api/claims/:id	DELETE	Menghapus Klaim	id(int)	-	Authorization: Bearer <token_jwt>
/api/claims/statistics	GET	Mendapatkan Statistik Klaim	-	-	Authorization: Bearer <token_jwt>
/api/payments	POST	Membuat Pembayaran	-	{ "policy_id": 2, "amount": 2000.00, "payment_date": "2025-01-13", "payment_method": "credit_card" }	Authorization: Bearer <token_jwt>
/api/payments/:id	GET	Mendapatkan Detail Pembayaran	id(int)	-	Authorization: Bearer <token_jwt>
/api/payments	GET	Mendapatkan Riwayat Pembayaran	-	-	Authorization: Bearer <token_jwt>

D. Langkah-langkah Pemograman

1. Buka command prompt
2. Ubah drive yang aktif sesuai dengan hard disk Anda sendiri, misalnya ubah ke drive d:\ seperti gambar di bawah ini:

```
Microsoft Windows [Version 10.0.17134.1602]  
(c) Microsoft Corporation. All rights reserved.  
C:\Users\ASUS>cd  
D:\>
```

3. Buat folder “insurance” menggunakan sintaks nama-direktori mkdir seperti yang ditunjukkan di bawah ini:

```
D:\>mkdir insurance
```

4. Kemudian ubah direktori saat ini ke “insurance”, menggunakan sintaks cd nama-direktori (ubah direktori), sebagai berikut:

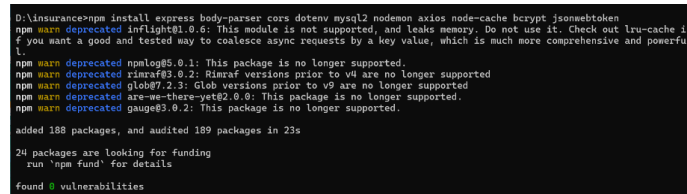
```
D:\>cd insurance
```

5. Untuk menginisialisasi proyek **Node.js** dengan cepat dan membuat file package.json secara otomatis di direktori proyek ini, gunakan perintah berikut:

```
D:\insurance>npm init -y
```

6. Instal library berikut:

- 1) Express
- 2) Body-parser
- 3) Cors
- 4) Dotenv
- 5) Mysql2
- 6) Nodemon
- 7) Axios
- 8) Node-cache
- 9) Bcrypt
- 10) Jsonwebtoken



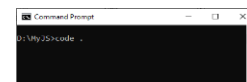
```
D:\insurance>npm install express body-parser cors dotenv mysql2 nodemon axios node-cache bcrypt jsonwebtoken
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache i
f you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerfu
l.
npm warn deprecated npmlog@5.0.1: This package is no longer supported.
npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated are-we-there-yet@2.0.0: This package is no longer supported.
npm warn deprecated gauge@3.0.2: This package is no longer supported.

added 188 packages, and audited 189 packages in 23s

24 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

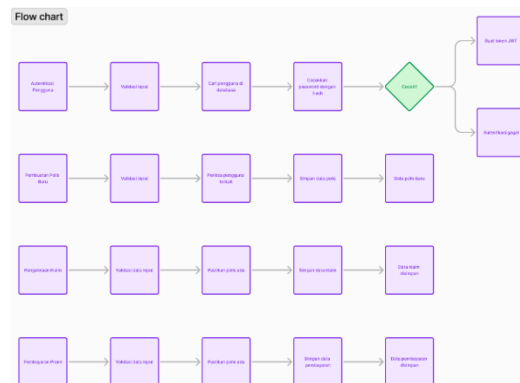
7. Untuk membuka folder proyek ke dalam kode visual studio gunakan perintah berikut:



8. Membuat file dan folder dengan susunan sebagai berikut:

- 1) Config
 - db.s// Konfigurasi koneksi database
 - node-cache .js
 - weather.js
- 2) Controllers
 - userController.js
 - policyController.js
 - claimController.js
 - paymentController.js
- 3) Middleware
 - logger.js
 - auth.js // Middleware autentikasi
- 4) Models
 - userModel.js
 - policyModel.js
 - claimModel.js
 - paymentModel.js
- 5) Routes //Mengarahkan request HTTP ke fungsi controller yang sesuai.
 - userRoutes.js
 - policyRoutes.js

- claimRoutes.js
 - paymentRoutes.js
- 6) env // File konfigurasi environment
- 7) Index.js // File utama
9. Sebelum memulai pemograman ada baiknya memperhatikan flowchart berikut untuk aloritmanya



10. Membuat file ‘.env’. File .env digunakan untuk menyimpan konfigurasi aplikasi, seperti pengaturan database, kunci rahasia JWT (JSON Web Token), dan port aplikasi. File ini bertujuan untuk menjaga informasi sensitif agar tidak tertanam langsung di kode sumber.

```

DB_HOST=localhost
DB_USER=root
DB_PASSWORD=
DB_NAME=insurance
JWT_SECRET='jsdgjhgbdvbjdngjdthur75894utifhjkdvnjfkht834yerhjsdfbvkgfhigfdvj'
PORT=3000
  
```

DB_HOST=localhost → Host untuk database

DB_USER=root → Username untuk koneksi ke database

DB_PASSWORD → Password untuk koneksi ke database

DB_NAME=insurance → Nama database yang digunakan

JWT_SECRET='jsdgjhgbdvbjdngjdthur75894utifhjkdvnjfkht834yerhjsdfbvkgfhigfdvj'
→ Kunci rahasia JW

PORT=3000 → Port untuk menjalankan server

11. Pada folder config buatlah 2 file yaitu db.js dan node-cache.js. Folder ini bertugas untuk mengelola konfigurasi

1) db.js

File ini berfungsi untuk membuat dan mengelola koneksi ke database MySQL menggunakan mysql2. Dan juga terdapat variable Pooling untuk meningkatkan efisiensi saat menangani banyak permintaan.

```
// Mengimpor modul mysql2 untuk koneksi database MySQL
const mysql = require('mysql2');

// Memuat variabel lingkungan dari file .env
require('dotenv').config();

// Membuat pool koneksi dengan konfigurasi dari .env
const pool = mysql.createPool({
  host: process.env.DB_HOST, // Host database
  user: process.env.DB_USER, // Username database
  password: process.env.DB_PASSWORD, // Password database
  database: process.env.DB_NAME, // Nama database
  waitForConnections: true, // Mengizinkan antrian koneksi jika limit tercapai
  connectionLimit: 10, // Maksimum koneksi simultan
  queueLimit: 0 // Tidak ada batas antrian koneksi
});

// Membuat koneksi pool dengan promise untuk memudahkan penggunaan async/await
const poolPromise = pool.promise()

// Mengekspor poolPromise agar dapat digunakan di bagian lain aplikasi
module.exports = poolPromise
```

2) node-cache.js

File ini bertugas mengatur caching data di sisi server untuk meningkatkan performa. Data yang sering digunakan dapat disimpan di memori dengan TTL (Time-to-Live) untuk menghindari pengambilan data berulang dari sumber utama (misalnya, database). Dan Modul ini berguna untuk caching hasil query atau data API sementara.

```
// Mengimpor modul node-cache untuk caching data
const nodeCache = require('node-cache');

// Membuat instance cache dengan TTL standar 3600 detik (1 jam)
const cache = new nodeCache({ stdTTL: 3600 });

// Mengekspor instance cache agar dapat digunakan di bagian lain aplikasi
module.exports = cache
```

12. Setelah itu buat beberapa modul pada folder controller. Fungsi utama Folder ini adalah Menangani Permintaan dari Klien. Fungsi-fungsi di dalam controller dipanggil oleh router saat menerima request (misalnya, GET, POST, PUT, atau DELETE). Controller bertanggung jawab untuk memvalidasi permintaan, memanggil model untuk

melakukan operasi database, dan mengembalikan respons ke klien. Berikut modul-modulnya:

a) UserController.js

File user-controller.js adalah bagian dari implementasi RestAPI yang menangani operasi pengguna, seperti pendaftaran, login, pembaruan data, penghapusan, dan pengambilan data pengguna. File ini berkomunikasi dengan model data (user-model) untuk menjalankan logika database.

```
const userModel = require('../models/user-model'); // Mengimport model user untuk operasi database

// Fungsi untuk mendaftarkan pengguna baru
const register = async (req, res) => {
  const data = req.body;

  try {
    const addUser = await userModel.register(data); // Memanggil fungsi register dari userModel
    if (addUser) {
      return res.status(200).json({ id: addUser.id, hash: addUser.pass }); // Mengembalikan ID dan hash password pengguna
    }
    return res.status(400).send({ msg: "Error Registration" }); // Respon jika pendaftaran gagal
  } catch (error) {
    console.log(error); // Log error ke konsol
  }
};

// Fungsi untuk login pengguna
const login = async (req, res) => {
  const data = req.body;

  try {
    const user = await userModel.login(data); // Memanggil fungsi login dari userModel
    if (user) {
      return res.status(200).json({ id: user.id, token: user.token }); // Mengembalikan ID dan token pengguna
    }
    return res.status(400).json({ msg: "Error Login" }); // Respon jika login gagal
  } catch (error) {
    console.log(error); // Log error ke konsol
  }
};
```

b) Policy-controller.js

Mengelola data polis asuransi, termasuk:

1. Membuat polis baru.
2. Mengambil data polis berdasarkan ID pengguna atau filter tertentu.
3. Memperbarui informasi polis.
4. Menghapus atau menonaktifkan polis.
5. Validasi status polis (aktif atau tidak).

```
const policyModel = require('../models/policy-model');

// Fungsi untuk membuat polis baru/registrasi polis
const createPolicy = async (req, res) => {
  const data = req.body;

  try {
    const result = await policyModel.createPolicy(data); // Memanggil fungsi createPolicy dari policyModel
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }
    res.status(201).json({ message: 'Policy created successfully', policyId: result.insertId }); // Mengembalikan pesan-
    // sukses dan ID polis yang baru dibuat
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' }); // Respon jika terjadi kesalahan server
  }
};

// Fungsi untuk mendapatkan semua data polis
const getAllPolicies = async (req, res) => {
  try {
    const policies = await policyModel.getAllPolicies(); // Memanggil fungsi getAllPolicies dari policyModel
    res.status(200).json(policies);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' }); // Respon jika terjadi kesalahan server
  }
};
```



```

//fungsi untuk mendapatkan data polis berdasarkan ID
const getPolicyById = async (req, res) => { //Mengambil parameter ID dari URL
  const { id } = req.params; //Mengambil parameter ID dari URL

  try {
    const policy = await policyModel.getPolicyById(id); //Memanggil fungsi getPolicyById dari policyModel
    if (!policy) {
      return res.status(404).json({ message: 'Policy not found' });
    }

    res.status(200).json(policy);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//fungsi untuk memperbarui data polis
const updatePolicy = async (req, res) => {
  const { id } = req.params; // Mengambil parameter ID dari URL
  const data = req.body; // Mengambil data polis yang dikirimkan melalui body request

  try {
    const result = await policyModel.updatePolicy(id, data);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }

    res.status(200).json({ message: 'Policy updated successfully' });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//fungsi untuk menghapus data polis
const deletePolicy = async (req, res) => {
  const { id } = req.params;

  try {
    const result = await policyModel.deletePolicy(id);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }

    res.status(200).json({ message: 'Policy deleted successfully' });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

module.exports = {
  createPolicy,
  getAllPolicies,
  getPolicyById,
  updatePolicy,
  deletePolicy
};

```

c) Payment-controller

File ini bertugas sebagai controller untuk menangani logika pembayaran dalam aplikasi, seperti membuat, membaca, memperbarui, dan menghapus data pembayaran dengan menggunakan paymentModel.js untuk berinteraksi dengan database.

```

const paymentModel = require('../models/payment-model'); // mengimpor payment model
//fungsi untuk membuat pembayaran baru
const createPayment = async (req, res) => {
  const data = req.body;

  try {
    const result = await paymentModel.createPayment(data);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }

    res.status(201)
      .json({ message: 'Payment created successfully', paymentId: result.insertId });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//fungsi untuk mendapatkan semua data pembayaran
const getAllPayments = async (req, res) => {
  try {
    const payments = await paymentModel.getAllPayments();
    res.status(200).json(payments);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//fungsi untuk menghapus pembayaran
const deletePayment = async (req, res) => {
  const { id } = req.params;

  try {
    const result = await paymentModel.deletePayment(id);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }

    res.status(200).json({ message: 'Payment deleted successfully' });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//export semua fungsi
module.exports = {
  createPayment,
  getAllPayments,
  getPaymentById,
  updatePayment,
  deletePayment,
};

```

```

//fungsi untuk mendapatkan data pembayaran berdasarkan ID
const getPaymentById = async (req, res) => {
  const { id } = req.params;

  try {
    const payment = await paymentModel.getPaymentById(id);
    if (!payment) {
      return res.status(404).json({ message: 'Payment not found' });
    }
    res.status(200).json(payment);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//fungsi untuk memperbarui data pembayaran
const updatePayment = async (req, res) => {
  const { id } = req.params;
  const data = req.body;

  try {
    const result = await paymentModel.updatePayment(id, data);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }
    res.status(200).json({ message: 'Payment updated successfully' });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

```

d) Claims-controller.js

Fungsi File claims-controller.js adalah Mengelola klaim asuransi, meliputi:

1. Pengajuan klaim baru.
2. Validasi klaim (kelengkapan data dan status polis).
3. Mengambil data klaim (per ID pengguna atau semua klaim).
4. Memperbarui status klaim (proses, disetujui, atau ditolak).
5. Menghapus klaim jika diperlukan.

```

// fungsi-fungsi CRUD untuk tabel klaim
const claimModel = require('../models/claim-model'); // mengimpor claim model
//fungsi untuk membuat klaim baru
const createClaim = async (req, res) => {
  const data = req.body;

  try {
    const result = await claimModel.createClaim(data);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }
    res.status(201).json({ message: 'Claim created successfully', claimId: result.insertId });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//fungsi untuk mendapatkan semua data klaim
const getAllClaims = async (req, res) => {
  try {
    const claims = await claimModel.getAllClaims();
    res.status(200).json(claims);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

```

```

//fungsi untuk mendapatkan data klaim berdasarkan ID
const getClaimById = async (req, res) => {
  const { id } = req.params;

  try {
    const claim = await claimModel.getClaimById(id);
    if (!claim) {
      return res.status(404).json({ message: 'Claim not found' });
    }
    res.status(200).json(claim);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//fungsi untuk memperbarui data klaim
const updateClaim = async (req, res) => {
  const { id } = req.params;
  const data = req.body;

  try {
    const result = await claimModel.updateClaim(id, data);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }
    res.status(200).json({ message: 'Claim updated successfully' });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

```

```

//fungsi untuk menghapus klaim
const deleteClaim = async (req, res) => {
  const { id } = req.params;

  try {
    const result = await claimModel.deleteClaim(id);
    if (result.error) {
      return res.status(400).json({ error: result.error });
    }
    res.status(200).json({ message: 'Claim deleted successfully' });
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

```

```

//fungsi untuk mendapatkan statistik klaim
const getClaimStatistics = async (req, res) => {
  try {
    const statistics = await claimModel.getClaimStatistics();
    res.status(200).json(statistics);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

//export semua fungsi CRUD
module.exports = {
  createClaim,
  getAllClaims,
  getClaimById,
  updateClaim,
  deleteClaim,
  getClaimStatistics
};

```

13. Setelah membuat folder controller.js ,selanjutnya kita masuk ke folder middleware.

Fungsi folder middleware dalam pembuatan REST API adalah untuk menampung kode yang berfungsi sebagai perantara antara permintaan (request) dan respons (response). Middleware digunakan untuk memproses request sebelum diteruskan ke handler atau controller, seperti autentikasi, validasi, logging, dan penanganan error. Dalam folder ini terdapat 2 modul yang memiliki fungsi yang berbeda yakni:

a) auth.js

Modul ini berfungsi untuk memeriksa dan mengautentikasi pengguna, biasanya dengan memverifikasi token atau kredensial yang diberikan pada request untuk memastikan bahwa pengguna memiliki hak akses yang sesuai.

```
const jwt = require('jsonwebtoken');//untuk mengimport library jsonwebtoken

const auth = (req, res, next) => { //membuat middleware auth
  const token = req.header('Authorization');//mengambil token dari header
  if(!token) {
    return res.status(401).json({message: "Unauthorized"});//respon jika tidak ada token
  }
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => { //verifikasi token
    if(err) {
      return res.json({message: "Invalid Token"});
    }
    next();//melanjutkan ke fungsi berikutnya
  });
}

module.exports = auth;
```

b) logger.js

Modul ini digunakan untuk mencatat log aktivitas request dan response, seperti informasi tentang permintaan yang masuk, waktu eksekusi, dan status respons, yang berguna untuk pemantauan dan debugging.

```
const logger = (req, res, next) => { //membuat middleware logger
  console.log(`${req.method} ${req.url}`);//menampilkan method dan URL request
  next();
}

module.exports = logger;
```

14. Setelah selesai ,lanjut membuat folder **model**, Folder model dalam pembuatan REST API digunakan untuk mendefinisikan struktur data atau entitas yang akan digunakan dalam aplikasi, seperti objek pengguna, produk, atau transaksi. Model ini biasanya berfungsi untuk berinteraksi dengan database, seperti mendefinisikan skema tabel (dalam database relasional) atau koleksi (dalam database NoSQL). Di dalamnya, terdapat logika terkait data seperti validasi, relasi antar entitas, dan operasi CRUD (Create, Read, Update, Delete). Pada folder ini terdapat 5 modul seperti sebelumnya yakni:

a) User-model.js

Modul ini mendefinisikan struktur data pengguna (user) yang logikanya telah diatur oleh user-controller.js dan berfungsi untuk berinteraksi dengan database mengenai informasi pengguna, seperti autentikasi, registrasi, dan profil pengguna.

```
const db = require('../config/db');//mengambil koneksi database
const bcrypt = require('bcrypt');//untuk mengimport library bcrypt
const jwt = require('jsonwebtoken');//untuk mengimport library jsonwebtoken
//fungsi untuk register user
const register = async (data) => {
  const {username, email, phone, password} = data//mengambil data username,
  // email, phone, password dari parameter data

  if(!username || !email || !phone || !password) {
    return {error: "username, email, phone, password is required"};
  }
  try {
    const salt = 10//salt untuk enkripsi password
    const hash = await bcrypt.hash(password, salt)
    const {result} =
      await db.query ("insert into users (username, email, phone, password) values (?, ?, ?, ?)",
      [username, email, phone, hash])//query untuk insert data ke tabel users
    return ({pass:hash});
  }
  catch (error) {
    console.log(error);
    return {error: "Database Error"};
  }
}

//fungsi untuk login user
const login = async (data) => {
  const {username, password} = data

  if(!username || !password) {
    return {error: "username and password is required"};
  }
  try {
    const [result] = await db.query("select * from users where username = ?",
    [username])//query untuk mengambil data user berdasarkan username
    if(result) {
      const isLogin = await bcrypt.compare(password, result[0].password)

      if(isLogin) {
        const payload = {payload untuk menyimpan data user
          id : result[0].id,
          username : result[0].username,
          email: result[0].email,
        }
        const token = jwt.sign(payload, process.env.JWT_SECRET,
        {expiresIn: '1h'})//membuat token
        return ({
          id: result[0].id,
          token: token
        });
      }
      return {message: "Invalid username or password"};
    }
    return {message: "Invalid username or password"};
  }catch (error) {
    console.log(error);
  }
}

//fungsi untuk mendapatkan semua data user
const getAllUsers = async () => {
  try{
    const [users] =
      await db.query('select * from users');//query untuk mengambil semua data user
    return users;
  }catch(error){
    console.log(error);
  }
}

//fungsi untuk mendapatkan data user berdasarkan id
const getUserById = async (id) => {
  // try{
  //   const [users] =
  //     await db.query('select * from users where id = ?', [id])//query untuk mengambil data user berdasarkan id
  //     return users;
  //   }catch (error){
  //     console.log(error);
  //   }
  // }
```

Dan disini terdapat fungsi untuk Public API untuk mengakses current dari weather.com. Berikut adalah source codenya:

```
const getUserById = async (id) => {
  try {
    // Ambil data user berdasarkan ID
    const [user] = await db.query('SELECT * FROM users WHERE id = ?', [id]);

    if (!user.length) {
      return null;
    }

    const userData = user[0];

    // Pastikan alamat tersedia sebelum memanggil API cuaca
    if (!userData.address) {
      return { ...userData, weather: null };
    }

    // Panggil API cuaca berdasarkan alamat
    const weatherResponse = await axios.get(BASE_URL, {
      params: {
        key: API_KEY,
        q: userData.address,
      },
    });

    // Ambil data cuaca
    const weatherData = weatherResponse.data;

    // Gabungkan data user dengan data cuaca
    return { ...userData, weather: {
      location: weatherData.location.name,
      region: weatherData.location.region,
    }
  };
}
```

```
    region: weatherData.location.region,
    country: weatherData.location.country,
    temperature: weatherData.current.temp_c,
    condition: weatherData.current.condition.text,
    humidity: weatherData.current.humidity,
    wind_speed: weatherData.current.wind_kph,
    wind_direction: weatherData.current.wind_dir,
    pressure: weatherData.current.pressure_mb,
    feels_like: weatherData.current.feelslike_c,
    last_updated: weatherData.current.last_updated,
  },
};
} catch (error) {
  console.error('Error fetching user or weather data:', error.message);
}
```

Dan beberapa fungsi lainnya yang sudah ada penjelasan pada comment line.

```
//fungsi untuk memperbarui data user
const updateUser = async (id, data) => {
  const {username, email, phone, role} = data

  if(!username || !email || !phone || !role) {
    return {error: "username, email, phone and role is required"};
  }

  try {
    await db.query('update users set username = ?, email = ?, phone = ?, role = ? where id = ?',
      [username, email, phone, role, id]); //query untuk update data user
    return {id, username, email, phone, role};
  } catch (error) {
    console.log(error);
  }
}

//fungsi untuk menghapus data user
const deleteUser = async (id) => {
  try {
    await db.query('delete from users where id = ?', [id]); //query untuk menghapus data user
    return {message: "User deleted"};
  } catch (error) {
    console.log(error);
  }
}

module.exports = {register, login, getAllUsers, getUserById, updateUser, deleteUser}
```

b) Policy-model.js

Modul ini mendefinisikan data terkait polis asuransi dari policy-controller.js, termasuk informasi seperti nomor polis, jenis asuransi, tanggal mulai, dan detail lainnya. Modul ini juga berfungsi untuk operasi terkait polis.

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

//fungsi untuk register user
const createPolicy = async (data) => {
  const { policy_number, policy_type, premium, start_date, end_date, user_id } = data;

  if (!policy_number || !policy_type || !premium || !start_date || !end_date || !user_id) {
    return { error: "policy_number, policy_type, premium, start_date, end_date, and user_id are required" };
  }

  try {
    // Periksa apakah user_id ada di tabel users
    const [users] = await db.query('SELECT id FROM users WHERE id = ?', [user_id]);
    if (users.length === 0) {
      return { error: "User ID not found" };
    }

    // Masukkan data ke tabel policies
    const [result] =
      await db.query('INSERT INTO policies (policy_number, policy_type, premium, start_date, end_date, user_id) VALUES (?, ?, ?, ?, ?, ?)',
        [policy_number, policy_type, premium, start_date, end_date, user_id]);
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};
```

```

//fungsi untuk mendapatkan semua data user
const getAllPolicies = async () => {
  try {
    const [policies] = await db.query('SELECT * FROM policies');//query untuk mengambil semua data polis
    return policies;
  } catch (error) {
    console.log(error);
    return [];
  }
};

//fungsi untuk mendapatkan data user berdasarkan ID
const getPolicyById = async (id) => {
  try {
    const [policies] =
      await db.query('SELECT * FROM policies WHERE id = ?', [id]);//query untuk mengambil data polis berdasarkan ID
    return policies[0];
  } catch (error) {
    console.log(error);
    return null;
  }
};

//fungsi untuk memperbarui data polis
const updatePolicy = async (id, data) => {
  const { policy_number, policy_type, premium, start_date, end_date, user_id } = data;/mengambil data polis yang dikirimkan melalui body request

  if (!policy_number || !policy_type || !premium || !start_date || !end_date || !user_id) {
    return { error: "policy_number, policy_type, premium, start_date, end_date, and user_id are required" };
  }

  try {
    const [result] = await db.query(`UPDATE policies SET policy_number = ?, policy_type = ?, premium = ?, start_date = ?, end_date = ?, user_id = ? WHERE id = ?`,
      [policy_number, policy_type, premium, start_date, end_date, user_id, id]
    );
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

//fungsi untuk menghapus data polis
const deletePolicy = async (id) => {
  try {
    const [result] = await db.query('DELETE FROM policies WHERE id = ?', [id]);//query untuk menghapus data polis
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

module.exports = {
  createPolicy,
  getAllPolicies,
  getPolicyById,
  updatePolicy,
  deletePolicy
};

```

c) Payment-model.js

Modul ini menangani data dan operasi terkait pembayaran, seperti pembayaran premi asuransi, riwayat transaksi, dan status pembayaran dari payment-controller.js

```

const db = require('.../config/db');//untuk mengimport koneksi database
//fungsi untuk membuat pembayaran
const createPayment = async (data) => {
  const { policy_id, amount, payment_date, payment_method } = data;
  //mengambil data policy_id, amount, payment_date, payment_method dari parameter data
  if (!policy_id || !amount || !payment_date || !payment_method) {
    return { error: "policy_id, amount, payment_date, and payment_method are required" };
  }

  try {
    // Periksa apakah policy_id ada di tabel policies
    const [policies] = await db.query('SELECT id FROM policies WHERE id = ?', [policy_id]);
    if (policies.length === 0) {
      return { error: "Policy ID not found" };
    }

    // Masukkan data ke tabel payments
    const [result] = await db.query(
      'INSERT INTO payments (policy_id, amount, payment_date, payment_method) VALUES (?, ?, ?, ?)',
      [policy_id, amount, payment_date, payment_method]
    );
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

//fungsi untuk mendapatkan semua data pembayaran
const getAllPayments = async () => {
  try {
    const [payments] = await db.query('SELECT * FROM payments');//query untuk mengambil semua data pembayaran
    return payments;
  } catch (error) {
    console.log(error);
    return [];
  }
};

//fungsi untuk mendapatkan data pembayaran berdasarkan ID
const getPaymentById = async (id) => {
  try {
    const [payments] = await db.query(
      'SELECT * FROM payments WHERE id = ?', [id]);//query untuk mengambil data pembayaran berdasarkan ID
    return payments[0];
  } catch (error) {
    console.log(error);
    return null;
  }
};

//fungsi untuk memperbarui data pembayaran
const updatePayment = async (id, data) => {
  const { policy_id, amount, payment_date, payment_method } = data;

  if (!policy_id || !amount || !payment_date || !payment_method) {
    return { error: "policy_id, amount, payment_date, and payment_method are required" };
  }

  try {
    const [result] = await db.query(

```

```

        "UPDATE payments SET policy_id = ?, amount = ?, payment_date = ?, payment_method = ? WHERE id = ?",
        [policy_id, amount, payment_date, payment_method, id]//query untuk update data pembayaran
    );
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
});

//fungsi untuk menghapus pembayaran
const deletePayment = async (id) => {
  try {
    const [result] = await db.query("DELETE FROM payments WHERE id = ?", [id]); //query untuk menghapus data pembayaran
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

module.exports = {
  createPayment,
  getAllPayments,
  getPaymentById,
  updatePayment,
  deletePayment,
};

```

d) Claims-model.js

Modul ini merupakan definisi database dari claims-controller.js yang berfokus pada data klaim asuransi, termasuk informasi klaim, CRUD klaim, status klaim, jumlah yang diklaim, dan proses verifikasi klaim.

```

const db = require("../config/db");//mengambil koneksi database
//fungsi untuk membuat klaim
const createClaim = async (data) => {
  const { policy_id, claim_amount, claim_date, status } = data;//mengambil data policy_id, claim_amount, claim_date,
  // status dari parameter data

  if (!policy_id || !claim_amount || !claim_date || !status) {
    return { error: "policy_id, claim_amount, claim_date, and status are required" };
  }

  try {
    // Periksa apakah policy_id ada di tabel policies
    const [policies] = await db.query("SELECT id FROM policies WHERE id = ?", [policy_id]);
    if (policies.length === 0) {
      return { error: "Policy ID not found" };
    }

    // Masukkan data ke tabel claims
    const [result] = await db.query(
      "INSERT INTO claims (policy_id, claim_amount, claim_date, status) VALUES (?, ?, ?, ?)",
      [policy_id, claim_amount, claim_date, status]//query untuk insert data ke tabel claims
    );
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

//fungsi untuk mendapatkan semua data klaim
const getAllClaims = async () => {
  try {
    const [claims] = await db.query("SELECT * FROM claims");
    return claims;
  } catch (error) {
    console.log(error);
    return [];
  }
};

//fungsi untuk mendapatkan data klaim berdasarkan ID
const getClaimById = async (id) => {
  try {
    const [claims] = await db.query("SELECT * FROM claims WHERE id = ?", [id]);
    return claims[0]; //query untuk mengambil data klaim berdasarkan ID
  } catch (error) {
    console.log(error);
    return null;
  }
};

//fungsi untuk memperbarui data klaim
const updateClaim = async (id, data) => {
  const { policy_id, claim_amount, claim_date, status } = data;

  if (!policy_id || !claim_amount || !claim_date || !status) {
    return { error: "policy_id, claim_amount, claim_date, and status are required" };
  }

  try {
    const [result] = await db.query(
      "UPDATE claims SET policy_id = ?, claim amount = ?, claim date = ?, status = ? WHERE id = ?",

```

```

        [policy_id, claim_amount, claim_date, status, id]//query untuk update data klaim
    );
    return result;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

//fungsi untuk menghapus klaim
const deleteClaim = async (id) => {
  try {
    const [result] = await db.query("DELETE FROM claims WHERE id = ?", [id]);
    return result; //query untuk menghapus data klaim
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

//fungsi untuk mendapatkan statistik klaim
const getClaimStatistics = async () => {
  try {
    const [statistics] = await db.query(
      "SELECT
        status,
        COUNT(*) as count,
        SUM(claim_amount) as total_amount
      FROM claims
      GROUP BY status;"
    ); //query untuk mendapatkan statistik klaim

    return statistics;
  } catch (error) {
    console.log(error);
    return { error: "Database Error" };
  }
};

module.exports = {
  createClaim,
  getAllClaims,
  getClaimById,
  updateClaim,
  deleteClaim,
  getClaimStatistics
};

```


15. Selanjutnya membuat folder untuk Routes. Folder routes dalam rancangan REST API digunakan untuk mengelola dan mendefinisikan rute API. Rute API menentukan bagaimana permintaan HTTP (GET, POST, PUT, DELETE) diarahkan ke controller yang sesuai. Folder ini membantu memisahkan logika routing dari bagian lain, sehingga proyek lebih modular dan terstruktur. Modul sama seperti sebelumnya yakni terdapat:

a) user-route.js

Fungsi utama modul ini adalah mengelola semua rute terkait pengguna (users), seperti registrasi, login, dan pengelolaan data pengguna.

```
const express = require('express');//untuk mengimport modul express
const router = express.Router();//router untuk mengatur rute
const userController = require('../controllers/user-controller');//mengimport user-controller
const auth = require('../middlewares/auth');//mengimport auth

router.post('/register', userController.register);//rute untuk register
router.get('/login', userController.login);//rute untuk login
router.get('/', auth, userController.getAllUsers); //rute untuk mendapatkan semua data user
router.get('/:id', auth, userController.getUserById);//rute untuk mendapatkan data user berdasarkan ID
router.put('/:id', auth, userController.updateUser);//rute untuk memperbarui data user
router.delete('/:id', auth, userController.deleteUser);//rute untuk menghapus data user

module.exports = router;
```

b) policy-route.js

Fungsi utama modul ini adalah mengelola semua rute terkait polis asuransi (policies), seperti membuat, membaca, memperbarui, dan menghapus data polis.

```
const express = require('express');//untuk mengimport modul express
const router = express.Router();//router untuk mengatur rute
const policyController = require('../controllers/policy-controller');//mengimport policy-controller
const authJWT = require('../middlewares/auth');//mengimport authJWT

router.post('/', policyController.createPolicy);//rute untuk membuat policy
router.get('/', policyController.getAllPolicies);//rute untuk mendapatkan semua data policy
router.get('/:id', authJWT, policyController.getPolicyById);//rute untuk mendapatkan data policy berdasarkan ID
router.put('/:id', authJWT, policyController.updatePolicy);//rute untuk memperbarui data policy
router.delete('/:id', authJWT, policyController.deletePolicy);//rute untuk menghapus data policy

module.exports = router;
```

c) payment-route.js

Fungsi utama modul ini adalah Mengelola semua rute terkait pembayaran premi asuransi (payments), seperti mencatat pembayaran, melihat daftar pembayaran, dan menghapus catatan pembayaran.

```
const express = require('express');//untuk mengimport modul express
const router = express.Router();//menggunakan router untuk mengatur rute
const paymentController = require('../controllers/payment-controller');//mengimport payment-controller
const authJWT = require('../middlewares/auth');//mengimport authJWT

router.post('/', authJWT, paymentController.createPayment);//rute untuk membuat pembayaran
router.get('/', authJWT, paymentController.getAllPayments);//rute untuk mendapatkan semua data pembayaran
router.get('/:id', authJWT, paymentController.getPaymentById);//rute untuk mendapatkan data pembayaran berdasarkan ID
router.put('/:id', authJWT, paymentController.updatePayment);//rute untuk memperbarui data pembayaran
router.delete('/:id', authJWT, paymentController.deletePayment);//rute untuk menghapus data pembayaran

module.exports = router;
```


d) claims-route.js

Fungsi utama modul ini adalah mengelola semua rute terkait klaim asuransi (claims), seperti membuat klaim baru, melihat klaim yang ada, dan memperbarui status klaim

```
const express = require('express')//untuk mengimport modul express
const router = express.Router()//menggunakan router untuk mengatur rute
const claimController = require('../controllers/claim-controller')//mengimport claim-controller
const authJWT = require('../middlewares/auth')//mengimport authJWT

router.post('/', authJWT, claimController.createClaim)//rute untuk membuat claim
router.get('/', authJWT, claimController.getAllClaims)//rute untuk mendapatkan semua data claim
router.get('/:id', authJWT, claimController.getClaimById)//rute untuk mendapatkan data claim berdasarkan ID
router.put('/:id', authJWT, claimController.updateClaim)//rute untuk memperbarui data claim
router.delete('/:id', authJWT, claimController.deleteClaim)//rute untuk menghapus data claim
router.get('/statistics', authJWT, claimController.getClaimStatistics)//rute untuk mendapatkan statistik claim

module.exports = router;
```

16. Sebelum ke file/modul index.js(file utama untuk menjalankan program),disini kami menambahkan API Public dari [weather.com](https://openweathermap.org/) saat sebelumnya terdapat fungsi yang sudah menerapkan Public API.Jadi setiap kita mengakses user by id,currentweather dari [weather.com](https://openweathermap.org/) akan menampilkan cuaca berdasarkan address user.Dan untuk implementasi API key nya diterapkan di getUserById.Berikut modul yang ditambahkan di folder config:

a) Weather.js

Modul ini ditambahkan pada folder config, Modul ini digunakan untuk menyimpan konfigurasi terkait API Weather.com, seperti:

- Base URL: URL dasar untuk mengakses API Weather.com.
- API Key: Kunci autentikasi untuk mengakses API.

Dengan menyimpan pengaturan ini di file config/weather.js, kita dapat dengan mudah mengubah URL atau API Key tanpa perlu mengubah kode utama.

```
require('dotenv').config()//mengimport dotenv
const axios = require('axios')//mengimport axios
//fungsi untuk membuat instance axios
const weatherAPI = axios.create({
  baseURL: "https://www.weatherapi.com/v1",//base url dari weather api
  headers:{
    Accept: 'application/json',//header untuk menerima data dalam bentuk json
    Authorization : `Bearer${process.env.WEATHER_BEARER}`//header untuk mengirim token
  }
})

module.exports = weatherAPI
```

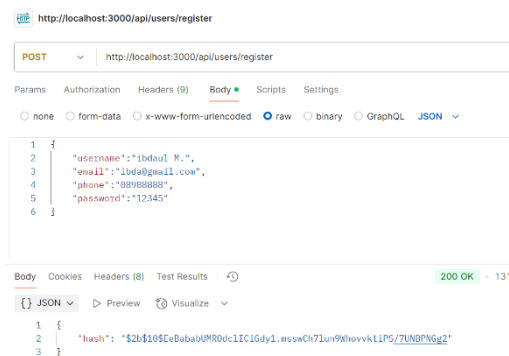
17. Jalankan program dengan perintah `npx nodemon index.js/node index.js` pada terminal, untuk implementasi program kami menggunakan localhost dan Postman

```
PS D:\insurance> npx nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server is running on port 3000
```

Jika sudah seperti ini maka endpoint sudah bisa digunakan.

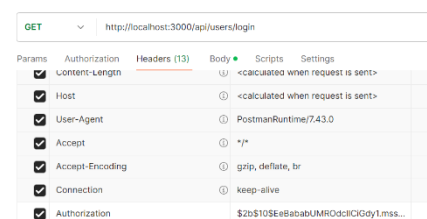
E. Implementasi Endpoint

No	Endpoint	HTTP Method	Description	Parameter	Request Header
1	/api/users/register	POST	Mendaftarkan pengguna baru	-	-



No	Endpoint	HTTP Method	Description	Parameter	Request Header
2	/api/users/register	GET	Login Pengguna	-	-

- Langkah pertama aktifkan Authorizatin di bagian header setelah itu masukan hash yang ada pada respon body saat login user sebelumnya.

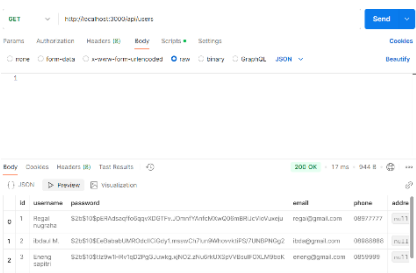


- Setelah itu Kembali pada body dan lakukan seperti yang terlihat di gambar berikut.

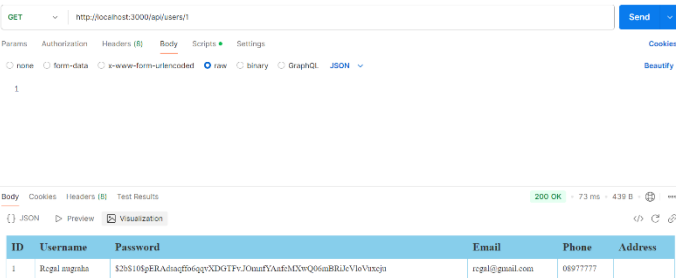


- Copy token diatas untuk Endpoint berikutnya
- **Note:** Setiap request header yang memiliki keterangan Authorization maka lakukan hal yang serupa.

No	Endpoint	HTTP Method	Description	Parameter	Request Header
3	/api/users/register	GET	Mendaftarkan pengguna baru	-	Authorization:Bearer<token_jwt>

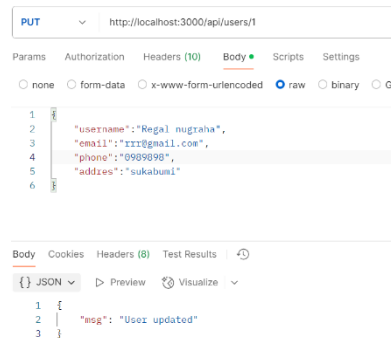


No	Endpoint	HTTP Method	Description	Parameter	Request Header
4	/api/users/register	GET	Mendafatkan detail pengguna berdasarkan id	Id(int)	Authorization:Bearer<token_jwt>



No	Endpoint	HTTP Method	Description	Parameter	Request Header
5	/api/users/register	PUT	Memperbarui data pengguna	Id(int)	Authorization:Bearer<token_jwt>

- Perhatikan tabel sebelumnya pada user-id1

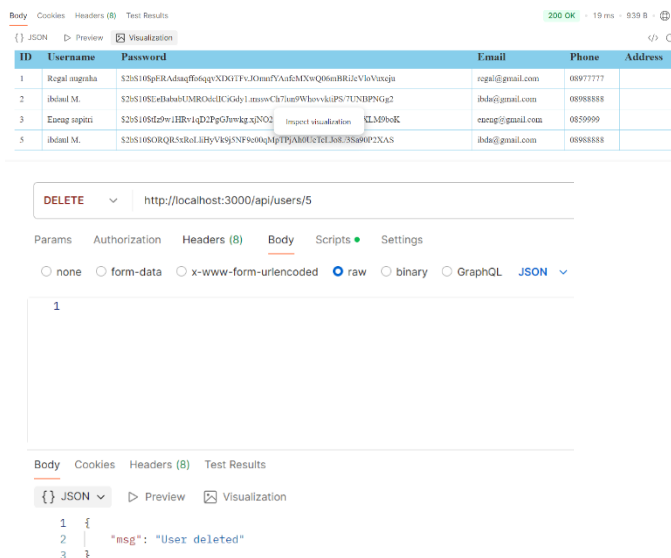


- Setelah Update:

			id	username	password		email	phone	address
<input type="checkbox"/>	Ubah	Salin	Hapus	1	Regal nugraha	\$2b\$10\$ERAdsagffo6qqvXDGTfV.JOmnyAnfcMXwQ06mBRJ...	rr@gmail.com	0989898	sukabumi
<input type="checkbox"/>	Ubah	Salin	Hapus	2	ibdaul M.	\$2b\$10\$EeBababUMROdclCIGdy1.msswCh7un9WhovktiPS...	ibda@gmail.com	08988888	NULL
<input type="checkbox"/>	Ubah	Salin	Hapus	3	Eneng sapitri	\$2b\$10\$tlz9w1HRv1qD2PgGJuwkg.xjNO2.zNu6rkUXSpVVBSu...	eneng@gmail.com	0859999	NULL

No	Endpoint	HTTP Method	Description	Parameter	Request Header
6	/api/users/register	DELETE	Menghapus Pengguna	Id(int)	Authorization:Bearer<token_jwt>

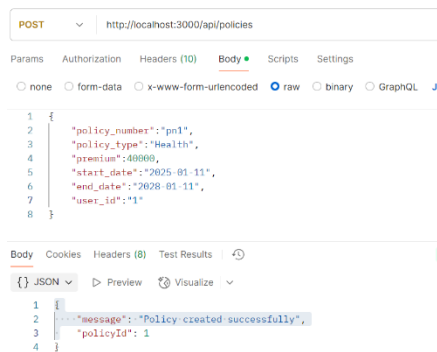
- Data sebelum dihapus (user id no 5 sengaja ditambahkan)



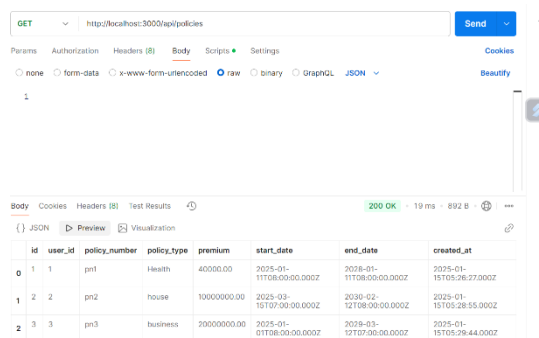
- Dan setelah dihapus

<div><div><div></div><div></div></div><div></div></div>												id	username	password	email	phone	address
<input type="checkbox"/>	<div><div><div>Ubah</div><div></div><div></div></div><div><div></div><div></div><div></div></div></div>	<div><div></div><div></div><div></div></div>	Salin	<div><div></div><div></div><div></div></div>	Hapus	1	Regal nugraha	\$2b\$10\$peERAdsagf6oqqvXDGTfVJ0mnlYAnfoMXwQ06mBRJ...	regal@gmail.com	08977777	NULL						
<input type="checkbox"/>	<div><div><div>Ubah</div><div></div><div></div></div><div><div></div><div></div><div></div></div></div>	<div><div></div><div></div><div></div></div>	Salin	<div><div></div><div></div><div></div></div>	Hapus	2	ibdaul M.	\$2b\$10\$EeBababUMROdcilCIGdy1.msswCh7lun9WhovktiPS...	ibda@gmail.com	08988888	NULL						
<input type="checkbox"/>	<div><div><div>Ubah</div><div></div><div></div></div><div><div></div><div></div><div></div></div></div>	<div><div></div><div></div><div></div></div>	Salin	<div><div></div><div></div><div></div></div>	Hapus	3	Eneng sapitri	\$2b\$10\$Ilz9w1HRv1qD2PgGJuwkg.xjN02.zNu6rKUXSpVVBsu...	eneng@gmail.com	08599999	NULL						

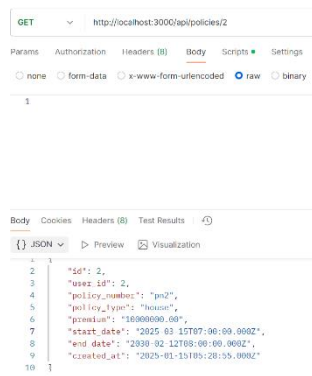
No	Endpoint	HTTP Method	Description	Parameter	Request Header
7	/api/policies	POST	Membuat polis baru	-	Authorization:Bearer <token_jwt>



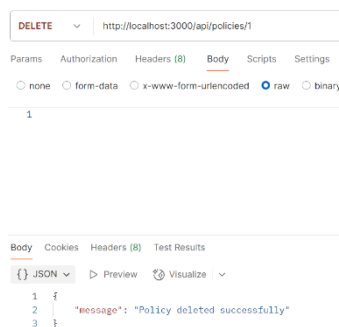
No	Endpoint	HTTP Method	Description	Parameter	Request Header
8	/api/policies	GET	Mendapatkan daftar polis	-	Authorization:Bearer <token_jwt>












No	Endpoint	HTTP Method	Description	Parameter	Request Header
9	/api/policies/:id	GET	Mendapatkan detail polis berdasarkan id	Id(int)	Authorization:Bearer <token_jwt>



No	Endpoint	HTTP Method	Description	Parameter	Request Header
10	/api/policies/:id	DELETE	Menghapus polis berdasarkan id	Id(int)	Authorization:Bearer <token_jwt>

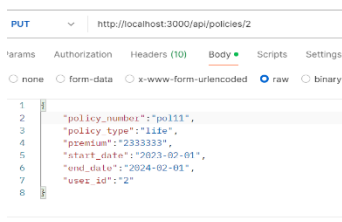


- Setelah di delete maka polis ber-id 1 hilang

+T-											id	user_id	policy_number	policy_type	premium	start_date	end_date	created_at
<input type="checkbox"/>		Ubah		Salin		Hapus	2	2	pn2	house	10000000.00	2025-03-15	2030-02-12	2025-01-14 21:28:55				
<input type="checkbox"/>		Ubah		Salin		Hapus	3	3	pn3	business	20000000.00	2025-01-01	2029-03-12	2025-01-14 21:29:44				
↑ <input type="checkbox"/> Pilih Semua											Dianan pilhan:  1 liah					 Salin	 Hapus	 Fksnr

No	Endpoint	HTTP Method	Description	Parameter	Request Header
11	/api/policies/:id	PUT	Memperbarui polis berdasarkan id	Id(int)	Authorization:Bearer <token_jwt>

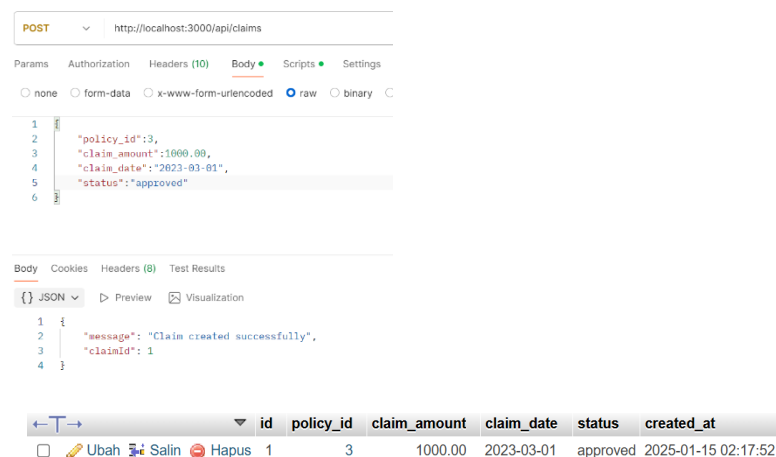
- Perhatikan table pada bagian endpoint DELETE



- Setelah di update :

<div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div></div><div></div></div></div>												id	user_id	policy_number	policy_type	premium	start_date	end_date	created_at
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	<div><div></div><div></div></div> Ubah	<div><div></div><div></div></div> Salin	<div><div></div><div></div></div> Hapus	2	2	pn2	house	10000000.00	2025-03-15	2030-02-12	2025-01-14 21:28:55								
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	<div><div></div><div></div></div> Ubah	<div><div></div><div></div></div> Salin	<div><div></div><div></div></div> Hapus	3	3	pn3	business	20000000.00	2025-01-01	2029-03-12	2025-01-14 21:29:44								
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	<div><div><div></div><div></div></div> Pilih Semua</div> <div><div></div><div></div></div> Dandan nilaihan: <div><div><div></div><div></div></div> Ubah</div> <div><div><div></div><div></div></div> Salin</div> <div><div><div></div><div></div></div> Hapus</div> <div><div><div></div><div></div></div> Fksnor</div>																		

No	Endpoint	HTTP Method	Description	Parameter	Request Header
12	/api/claims	POST	Mengajukan klaim	-	Authorization:Bearer <token_jwt>



No	Endpoint	HTTP Method	Description	Parameter	Request Header
13	/api/claims/:id	GET	Mendapatkan detail klaim berdasarkan id	Id(int)	Authorization:Bearer <token_jwt>

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

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary Graph

1

body Cookies Headers (8) Test Results

JSON Preview Visualization

```
1 {
2   "id": 2,
3   "policy_id": 2,
4   "claim_amount": "5000.00",
5   "claim_date": "2023-10-11T07:00:00.000Z",
6   "status": "",
7   "created_at": "2025-01-15T02:22:10.000Z"
8 }
```

No	Endpoint	HTTP Method	Description	Parameter	Request Header
14	/api/claims	GET	Mendapatkan daftar klaim	-	Authorization:Bearer <token_jwt>

GET http://localhost:3000/api/claims

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

body Cookies Headers (8) Test Results

JSON Preview Visualization

	id	policy_id	claim_amount	claim_date	status	created_at
0	1	2	1000.00	2023-09-01T08:00:00.000Z	approved	2025-01-15T03:52:00.000Z
1	2	2	5000.00	2023-10-11T07:00:00.000Z		2025-01-15T02:22:10.000Z

No	Endpoint	HTTP Method	Description	Parameter	Request Header
15	/api/claims/:id	PUT	Memperbarui status klaim	Id(int)	Authorization:Bearer <token_jwt>

http://localhost:3000/api/policies/1

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

Params Authorization Headers (10) Body Scripts

none form-data x-www-form-urlencoded raw

```
1 {
2   "policy_id":2,
3   "claim_amount":5000.00,
4   "claim_date":"2023-10-11",
5   "status":"approved"
6 }
```

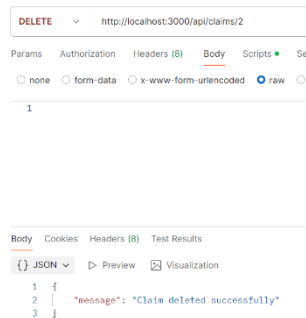
body Cookies Headers (8) Test Results

JSON Preview Visualization

```
1 {
2   "message": "Claim updated successfully"
3 }
```

				id	policy_id	claim_amount	claim_date	status	created_at
<input type="checkbox"/>	Ubah	Salin	Hapus	1	3	1000.00	2023-03-01	approved	2025-01-15 02:17:52
<input type="checkbox"/>	Ubah	Salin	Hapus	2	2	5000.00	2023-10-11	approved	2025-01-15 02:22:10

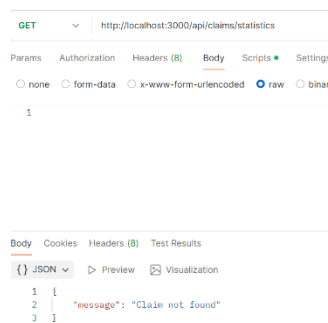
No	Endpoint	HTTP Method	Description	Parameter	Request Header
16	/api/claims/:id	DELETE	Menghapus klaim berdasarkan id	Id(int)	Authorization:Bearer <token_jwt>



- Setelah klaim ber-id 2 terhapus:

	id	policy_id	claim_amount	claim_date	status	created_at
	1	3	1000.00	2023-03-01	approved	2025-01-15 02:17:52

No	Endpoint	HTTP Method	Description	Parameter	Request Header
17	/api/claims/statistic	GET	Mendapatkan statistic klaim	-	Authorization:Bearer <token_jwt>



- Disini tidak bisa mendapatkan statistik karena data hanya 1

No	Endpoint	HTTP Method	Description	Parameter	Request Header
18	/api/claims/payments	POST	Membuat Pembayaran	-	Authorization:Bearer <token_jwt>

POST http://localhost:3000/api/payments

Params Authorization Headers (10) Body • Scripts • Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```

1 {
2   "policy_id": 2,
3   "amount": 5000.00,
4   "payment_date": "2023-10-11",
5   "payment_method": "credit_card"
6 }

```

Body Cookies Headers (8) Test Results

☐ JSON ☒ Preview ☐ Visualization

```

1 {
2   "message": "Payment created successfully",
3   "paymentId": 2
4 }

```

No	Endpoint	HTTP Method	Description	Parameter	Request Header
19	/api/claims/payments	GET	Mendapatkan Detail Pembayaran	Id(int)	Authorization:Bearer <token_jwt>

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

Params Authorization Headers (8) Body • Scripts • Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```

1

```

Body Cookies Headers (8) Test Results

☐ JSON ☒ Preview ☐ Visualization

```

1 {
2   "id": 1,
3   "policy_id": 2,
4   "amount": "2000.00",
5   "payment_date": "2023-01-13T08:00:00.000Z",
6   "payment_method": "credit_card",
7   "created_at": "2025-01-15T11:08:47.000Z"
8 }

```

No	Endpoint	HTTP Method	Description	Parameter	Request Header
20	/api/payments	GET	Mendapatkan Riwayat Pembayaran	-	Authorization:Bearer <token_jwt>

http://localhost:3000/api/payments

GET http://localhost:3000/api/payments Send

Params Authorization Headers (8) Body • Scripts • Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☒ JSON Beautify

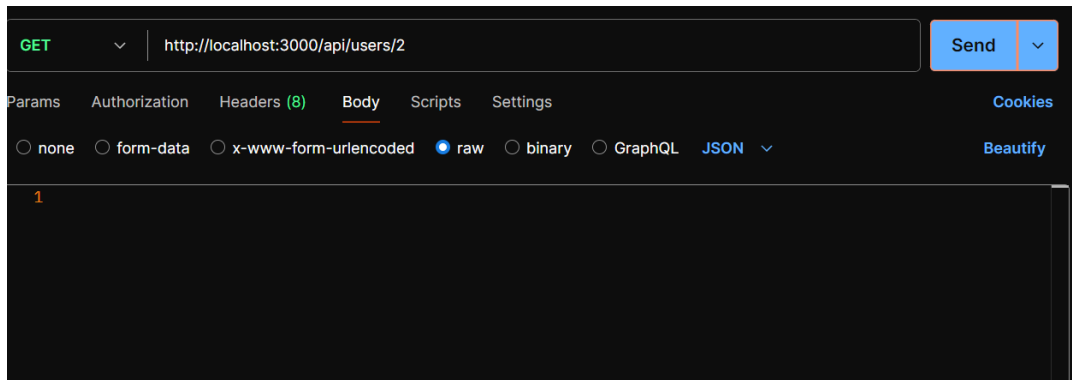
1

body Cookies Headers (8) Test Results 200 OK · 8 ms · 580 B ·

☐ JSON ☒ Preview ☐ Visualization

	id	policy_id	amount	payment_date	payment_method	created_at
0	1	2	2000.00	2023-01-13T08:00:00.000Z	credit_card	2025-01-15T11:08:47.000Z
1	2	3	5000.00	2023-10-11T07:00:00.000Z	credit_card	2025-01-15T11:10:18.000Z

- Selanjutnya kita gunakan http method pada Endpoint terakhir yakni untuk mengakses current dari weather.com dengan menggunakan getUserById



```
{
  "id": 2,
  "username": "ibda.mutafakkir",
  "password": "$2b$10$Z93I8rlMLwX0uQ/nx.IFb.rsZ6aIhL3fG/ihPuKBplxp2rS6/ZpP2",
  "email": "ibda@gmail.com",
  "phone": 857575757,
  "address": "sukabumi",
  "weather": {
    "location": "Sukabumi",
    "region": "West Java",
    "country": "Indonesia",
    "temperature": 22.1,
    "condition": "Patchy rain nearby",
    "humidity": 92,
    "wind_speed": 5.8,
    "wind_direction": "SW",
    "pressure": 1011,
    "feels_like": 24.6,
    "last_updated": "2025-01-15 20:00"
  }
}
```

- **SELESAI!**