

Develop Service Media dengan Node.js, Express, dan MySQL

Pada praktikum ini mahasiswa akan belajar membuat sebuah service microservice sederhana bernama **service-media** menggunakan framework Express.js. Service ini nantinya dapat dikembangkan menjadi layanan terpisah dalam arsitektur microservices

LANGKAH 1 — Membuka CMD

Buka Command Prompt terlebih dahulu, Masuk ke folder **micro** pada drive C.

```
C:\>dir
Volume in drive C has no label.
Volume Serial Number is 7237-F6B3

Directory of C:\

28/02/2025  14.51    <DIR>          composer
09/04/2025  09.38    <DIR>          inetpub
12/03/2026  19.50    <DIR>          micro
01/04/2024  14.26    <DIR>          PerfLogs
02/03/2026  14.09    <DIR>          Program Files
26/05/2025  12.15    <DIR>          Program Files (x86)
11/08/2025  09.55    <DIR>          src
07/08/2025  11.25    <DIR>          Users
12/03/2026  10.53    <DIR>          Windows
27/02/2026  10.12    <DIR>          xampp
               0 File(s)                0 bytes
             10 Dir(s)  383.813.431.296 bytes free

C:\>cd m*

C:\micro>dir
Volume in drive C has no label.
Volume Serial Number is 7237-F6B3

Directory of C:\micro

12/03/2026  19.50    <DIR>          .
06/03/2026  09.10    <DIR>          api-gateway
12/03/2026  20.17    <DIR>          service-media
               0 File(s)                0 bytes
               3 Dir(s)  383.813.431.296 bytes free
```

LANGKAH 2 — Membuat Project Express Baru

Ketik perintah berikut: **express --no-view service-media**, lalu Masuk ke Folder Project **cd service-media**, lalu instal dependency awal **npm install** , berikutnya di vs code

Buka Visual Studio Code, lalu buka folder service media. Install dotenv dengan perintah : **npm install dotenv --save**, berikutnya Mengubah File Routing Masuk ke folder routes klik dan rename users.js menjadi media.js

Berikutnya kita Membuat File .env dengan cara klik kosong diluar new file buat .env dan setelah itu isi file .env dengan kodingan berikut :

PORT=8080

DB_NAME=media-database

DB_USERNAME=root

DB_PASSWORD=

DB_HOSTNAME=localhost

lalu pada app.js ganti var dengan const (cara cepat dg ctrl + f2)

caranya : shift + ctrl + alt + tanda bawah

tambahkan di baris 1 require ('dotenv').config();

```
require('dotenv').config();
```

berikutnya Mengubah Routing User Menjadi Media, pada app.js

cari kode

```
const usersRouter = require('./routes/users');
```

ganti dengan

```
const mediaRouter = require('./routes/media');
```

lalu bikin database pada

DB_NAME=media-database

DB_USERNAME=root

DB_PASSWORD=

DB_HOSTNAME=localhost

aktifkan XAMPP, mysql lalu buat database di php my admin media-database

SETUP SEQUELIZE

buka cmd prompt

npm install sequelize sequelize-cli --save

```
C:\micro\service-media>npm install sequelize sequelize-cli --save
npm warn deprecated glob@10.5.0: Old versions of glob are not supported
h have been fixed in the current version. Please update. Support for old
g i@izs.me

added 104 packages, and audited 159 packages in 21s

19 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (5 low, 4 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues, run:
  npm audit fix --force

Run `npm audit` for details.
npm notice
npm notice New major version of npm available! 10.9.2 -> 11.11.1
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.11.1
npm notice To update run: npm install -g npm@11.11.1
npm notice

C:\micro\service-media>
```

npx sequelize (untuk cek command)

npx sequelize init (untuk menginit) nanti akan membuat 3 folder, migration, model dan seeders

```
C:\micro\service-media>npx sequelize init

Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

Created "config\config.json"
Successfully created models folder at "C:\micro\service-media\models".
Successfully created migrations folder at "C:\micro\service-media\migrations".
Successfully created seeders folder at "C:\micro\service-media\seeders".

C:\micro\service-media>
```

lalu pada vs code rename config.json menjadi config.js dan tambahkan koding berikut pada config.js

```
require('dotenv').config();

const {
} = process.env;

module.exports =
```

lalu pada models buka index.js di baris 9 ganti config.json menjadi config, lalu pada config.js isikan

```
const {  
  DB_USERNAME,  
  DB_PASSWORD,  
  DB_NAME,  
  DB_HOSTNAME  
}
```

Lalu untuk setingan username, password serta database nya menjadi seperti berikut :

```
{  
  
  "development": {  
  
    "username": "DB_USERNAME",  
  
    "password": DB_PASSWORD,  
  
    "database": DB_NAME,  
  
    "host": DB_HOSTNAME,  
  
    "dialect": "mysql"  
  
  },  
  
  "test": {  
  
    "username": "DB_USERNAME",  
  
    "password": DB_PASSWORD,  
  
    "database": DB_NAME,  
  
    "host": DB_HOSTNAME,  
  
    "dialect": "mysql"  
  
  },  
  
  "production": {  
  
    "username": "DB_USERNAME",  
  
    "password": DB_PASSWORD,  
  
    "database": DB_NAME,  
  
    "host": DB_HOSTNAME,  
  
    "dialect": "mysql"  
  
  }  
  
}
```

MEMBUAT MIGRATION dan MODEL

pada cmd ketik perintah

npx sequelize migration:create --name=create-table-media

pada vs code buka folder migrations

klik js 2026 lalu baris ke 2 : `/** @type {import('sequelize-cli').Migration} */`

dihapus

lalu hapus tulisan `async`

setelah buat codingan di vs code

```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable('media', {
      id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false
      },
      image: {
        type: Sequelize.STRING,
        allowNull: false
      },
      created_at: {
        type: Sequelize.DATE,
        allowNull: false
      },
      updated_at: {
        type: Sequelize.DATE,
        allowNull: false
      }
    });
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('media');
  }
};
```

buka cmd **`npm install mysql2--save`**

`npx sequelize db:migrate`

```
C:\micro\service-media>npx sequelize db:migrate

Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

[dotenv@17.3.1] injecting env (5) from .env -- tip: ⚙️ l
Loaded configuration file "config\config.js".
Using environment "development".
== 20260312133416-create-table-media: migrating =====
== 20260312133416-create-table-media: migrated (0.036s)
```

Lalu kita undo, nanti di php my admin nya akan hilang tabel media lalu kita migrate lagi

npx sequelize db:migrate:undo

npx sequelize db:migrate, untuk migrate lagi

nah langkah berikutnya pada vs code buat pada models

kita new file media.js ketik kodingan

```
module.exports = (Sequelize, DataTypes) => {
  const Media = sequelize.define('Media', {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
      allowNull: false
    },
    image: {
      type: DataTypes.STRING,
      allowNull: false
    },
    createdAt: {
      field: 'created_at',
      type: DataTypes.DATE,
      allowNull: false
    },
    updatedAt: {
      field: 'update_at',
      type: DataTypes.DATE,
      allowNull: false
    }
  }, {
    tableName: 'media'
  });
  return Media;
}
```

Berikutnya undo dan migrate lagi Hasilnya pada php my admin

localhost/phpmyadmin/index.php?route=/table/structure&server=1&db=media-database&table=media

Server: 127.0.0.1 » Database: media-database » Tabel: media

Jelajahi Struktur SQL Cari Tambahkan Ekspor Impor Hak Akses Operasi Lainnya

Struktur tabel Tampilan hubungan

#	Nama	Jenis	Penyortiran	Atribut	Tak Ternilai	Bawaan	Komentar	Ekstra	Tindakan
☐	1 id	int(11)			Tidak	Tidak ada		AUTO_INCREMENT	Ubah Hapus Lainnya
☐	2 image	varchar(255) utf8mb4_general_ci			Tidak	Tidak ada			Ubah Hapus Lainnya
☐	3 created_at	datetime			Tidak	Tidak ada			Ubah Hapus Lainnya
☐	4 updated_at	datetime			Tidak	Tidak ada			Ubah Hapus Lainnya

☐ Pilih Semua Dengan pilihan: Jelajahi Ubah Hapus Utama Unik Indeks Spasial

Teks penuh Add to central columns Remove from central columns

Cetak Usulkan struktur tabel Lacak tabel Move columns Normalisasi

Tambahkan 1 kolom setelah updated_at Kirim

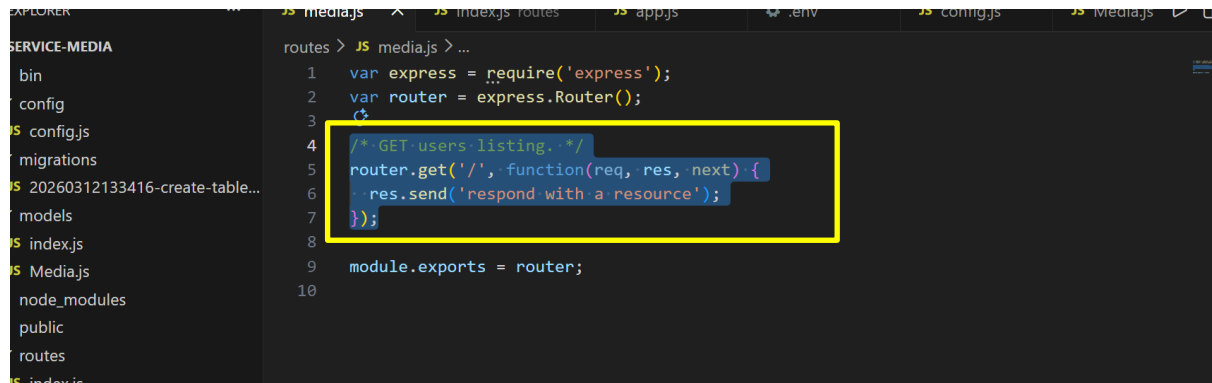
Indeks

API POST dan Upload IMAGE (Membuat API untuk mengupload gambar)

Didalam folder routes ada file media.js

Ganti koding var pada baris 1 dan 2 menjadi const dan

Hapus koding pada baris 4 sd 7 lalu diganti dengan buat router baru



```
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET users listing. */
5 router.get('/', function(req, res, next) {
6   res.send('respond with a resource');
7 });
8
9 module.exports = router;
```

```
router.post('/', (req, res) => {
  res.send('ok');
});
```

Lalu varnya menjadi const

```
const express = require('express');
const router = express.Router();
```

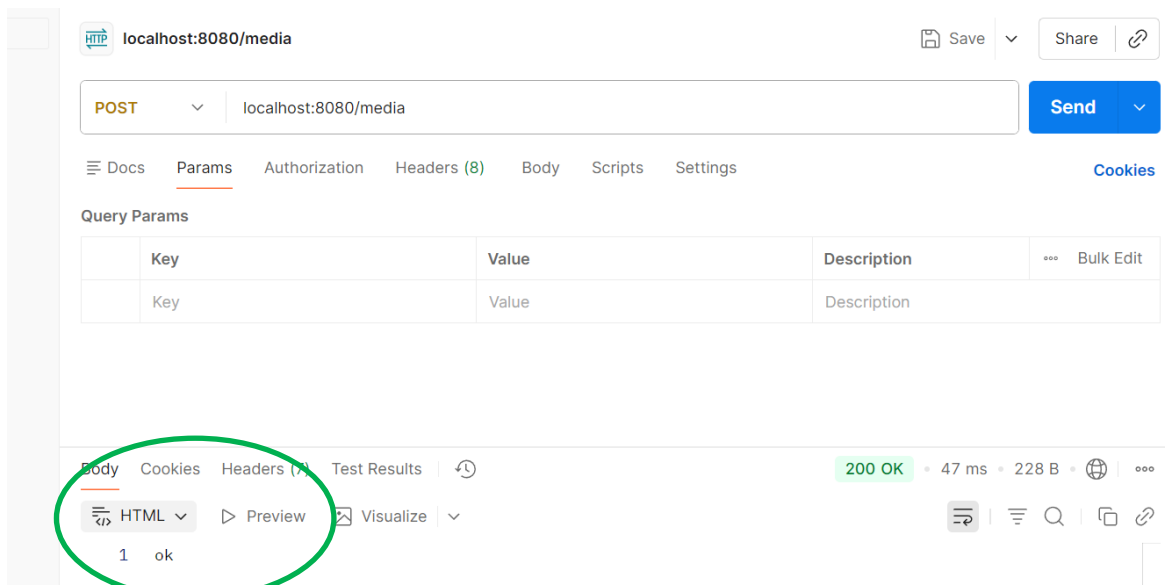
setelah itu buka cmd. Didalam service media jalankan npm start

```
C:\micro\service-media>npm start

> service-media@0.0.0 start
> node ./bin/www

[dotenv@17.3.1] injecting env (5) from .env -- tip: 🗝️ prevent building .env in docker:
```

di post man : post localhost:8080/media lalu klik send hasilnya html ok



Lalu kita instal sebuah package bernama nodemon (librari untuk mereload otomatis) dengan cara di cmd : npm install -g nodemon

```
C:\micro\service-media>npm install -g nodemon  
  
removed 1 package, and changed 28 packages in 8s  
  
5 packages are looking for funding  
  run `npm fund` for details  
  
C:\micro\service-media>
```

Lalu kita jalankan dengan perintah di cmd : service-media>nodemon bin/www

```
C:\micro\service-media>nodemon bin/www  
[nodemon] 3.1.14  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node bin/www`  
[dotenv@17.3.1] injecting env (5) from .env -- tip: ⚡ secrets for
```

Sekarang server kita sudah berjalan

Sekarang instal sebuah library isbase 64

tekan di cmd ctrl c lalu : npm install is-base64 base64-img --save

```
C:\micro\service-media>npm install is-base64 base64-img --save
added 6 packages, and audited 175 packages in 6s

22 packages are looking for funding
  run `npm fund` for details

14 vulnerabilities (5 low, 4 high, 5 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible, run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.

C:\micro\service-media>
```

pada media.js di vs code tambahkan kode baris 3 dan 4 sehingga menjadi

```
const express = require('express');
const router = express.Router();
const isBase64 = require('is-base64');
const base64img = require('base64-img');

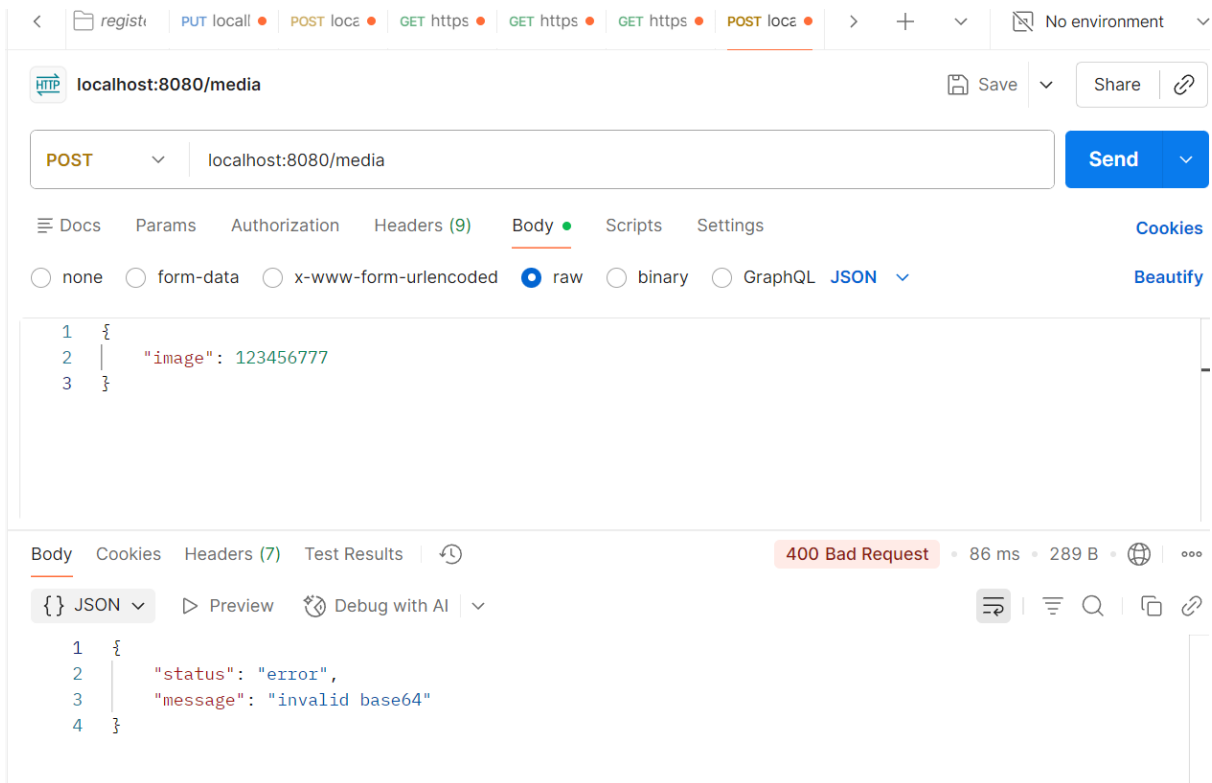
router.post('/', (req, res) => {
  const image = req.body.image;

  if (!isBase64(image, { mimeRequired: true })) {
    return res.status(400).json({
      status: 'error',
      message: 'invalid base64'
    });
  }
});

module.exports = router;
```

lalu jalankan server kita lagi di cmd dg perintah nodemon bin/www

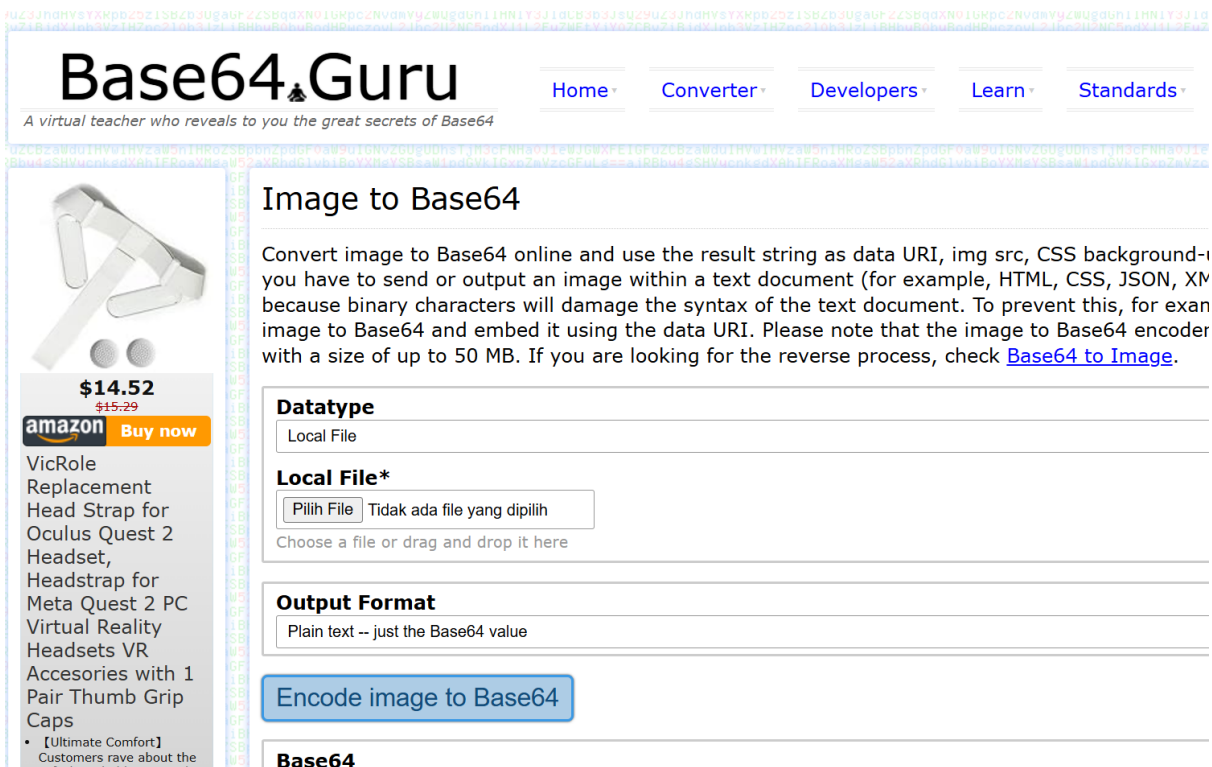
kita test dulu di postman, pada postman pilih body - raw "image" "kode diambil dari google"



Dia akan memberikan respon invalid base

Langkah selanjutnya buka google, ketikan image to base64

Pilih yang base64.guru



Lalu upload lah sebuah image dan pada output format pilih data Uri

Setelah dikonver

Copi dan tarok di postman setelah “image” :

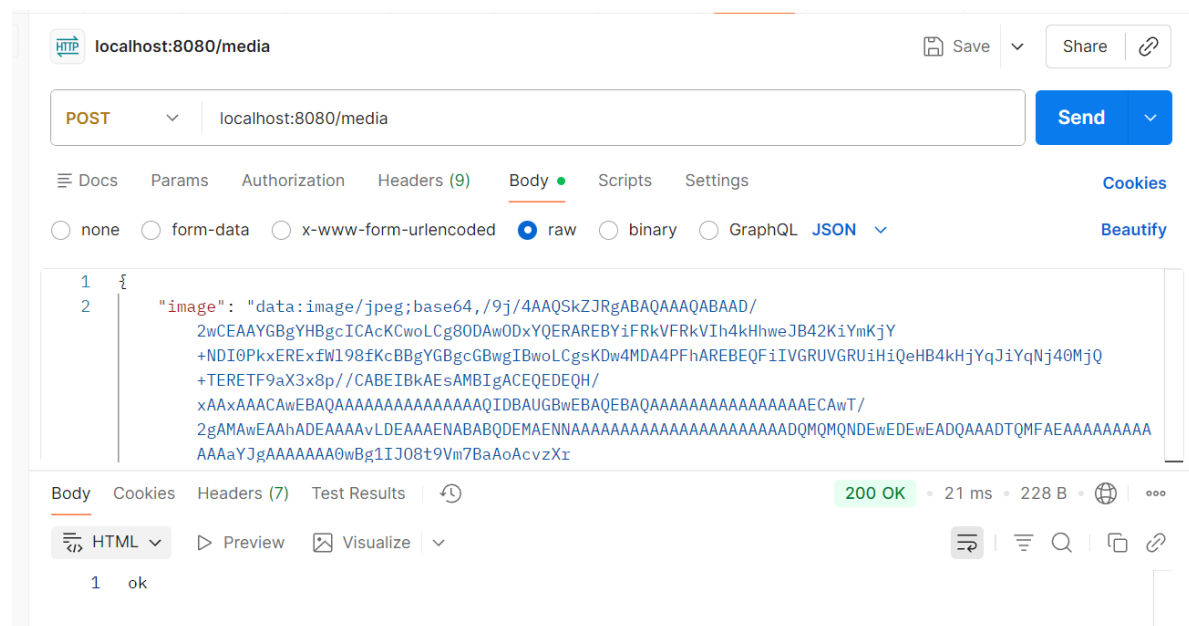
Sebelum dijalankan di postman buka dulu vs code tambhkn di baris ke 15 media.js

```
return res.send('ok');
```

sebelum dijalankan pada app.js baris ke 13 dan 14 tambhkan limit

```
app.use(express.json({ limit: '50mb' }));  
app.use(express.urlencoded({ extended: false, limit: '50mb' }));
```

sehingga nampak di postman nya memberikan respon ok



Nah sekarang untuk mengupload image, buka media.js

Tambahkan pada baris ke 15 ganti

```
return res.send('ok');
```

menjadi

```
const express = require('express');
const router = express.Router();
const isBase64 = require('is-base64');
const base64Img = require('base64-img');

const { Media } = require('../models');

router.post('/', async (req, res) => {
  const image = req.body.image;

  // Validasi base64
  if (!isBase64(image, { mimeRequired: true })) {
    return res.status(400).json({
      status: 'error',
      message: 'invalid base64'
    });
  }

  // Convert base64 ke file
  base64Img.img(image, './public/images', Date.now(), async (err, filepath) => {
    if (err) {
      return res.status(400).json({
        status: 'error',
        message: err.message
      });
    }

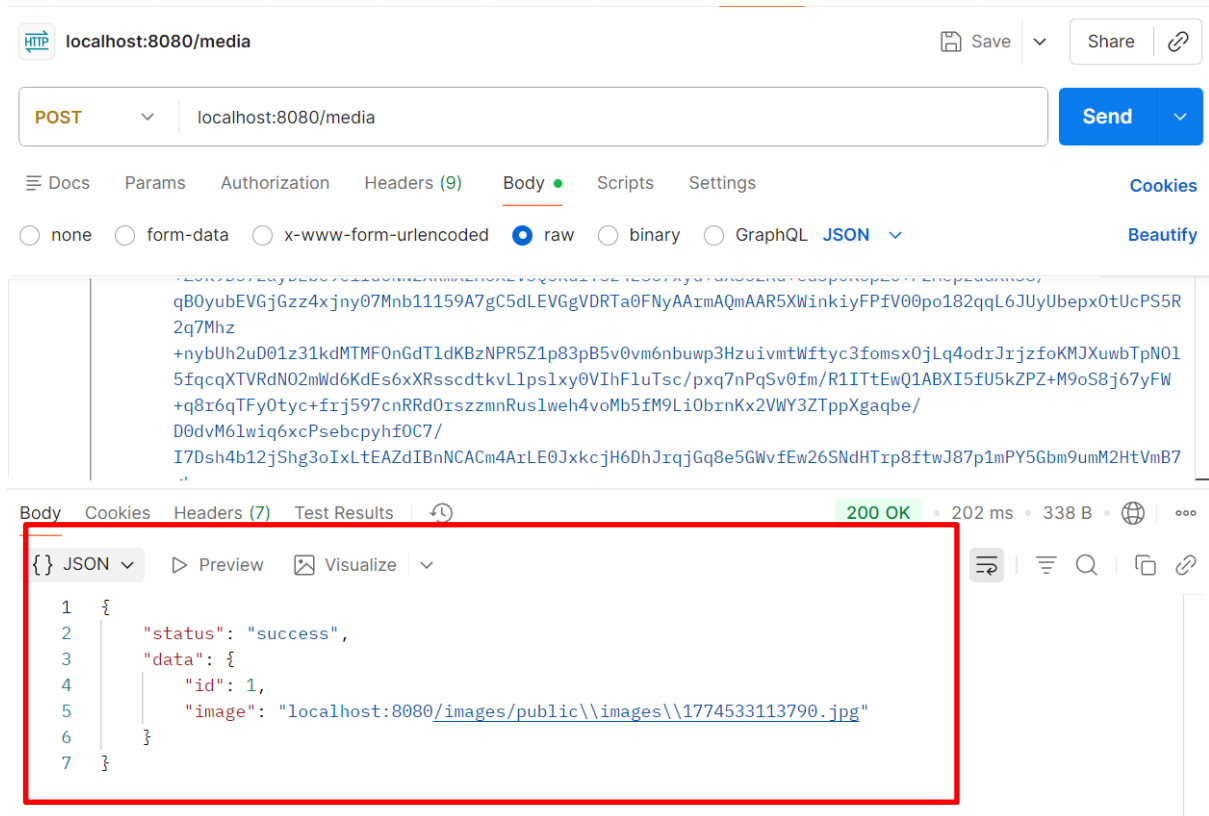
    try {
      // Ambil nama file
      const filename = filepath.split('/').pop();

      // Simpan ke database
      const media = await Media.create({
        image: `images/${filename}` // <-- diperbaiki (tanpa typo)
      });

      // Response
      return res.json({
        status: 'success',
        data: {
          id: media.id,
          image: `${req.protocol}://${req.get('host')}/images/${filename}`
        }
      });
    } catch (error) {
      return res.status(500).json({
```

```
status: 'error',  
message: error.message  
});  
}  
});  
});  
  
module.exports = router;
```

jika berhasil pada postman



Dan jika kita cek pada direktori akan muncul (disimpan dalam folder public/images)

Dosen : Riyan Ikbal Salam M.Kom dan Putra Manda M.Kom – PNP PSDKU Solok Selatan



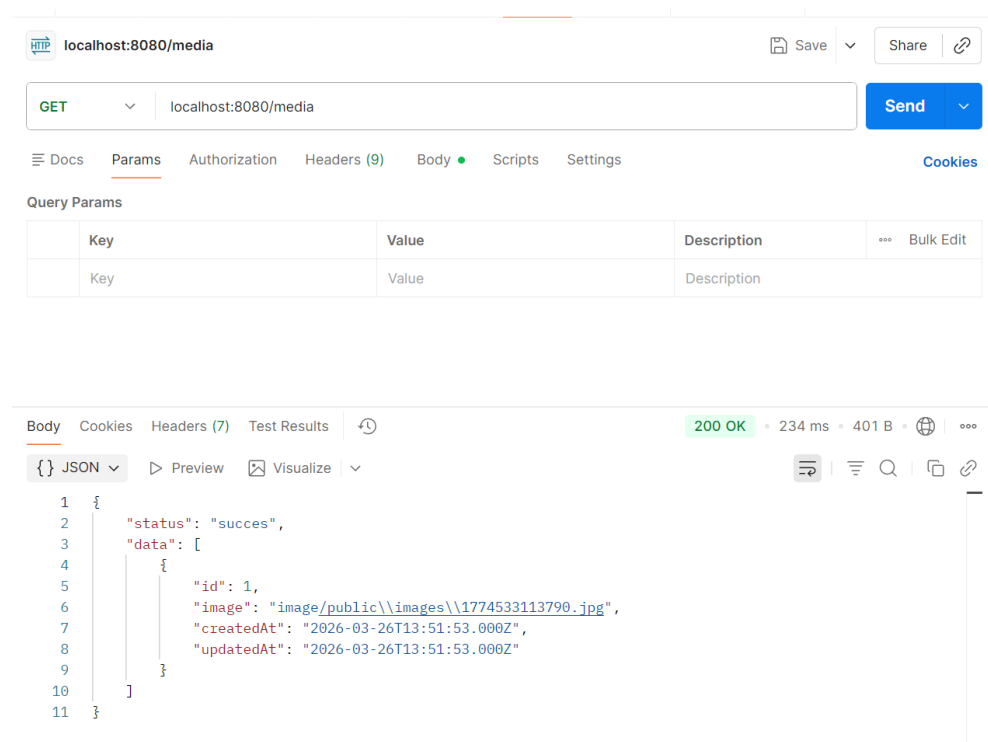
Menggunakan Methode Get

Pada media.js

Di Baris ke 8 tambahkan kodingan

```
router.get('/', async (req, res) => {  
  const media = await Media.findAll();  
  return res.json({  
    status: 'succes',  
    data: media  
  });  
});
```

Nanti pada postman kita panggil menggunakan methode get seperti gambar



Berikutnya pada baris ke 11 di media.js

```
const mappedMedia = media.map((m) => {  
  m.image = `${req.get('host')}/${m.image}`;  
  return m;  
});
```

Balik ke post lagi

HTTP localhost:8080/media Save Share

POST localhost:8080/media Send

Docs Params Authorization Headers (9) Body • Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results 200 OK • 62 ms • 338 B

{ } JSON Preview Visualize

```
1 {
2   "status": "success",
3   "data": {
4     "id": 2,
5     "image": "localhost:8080/images/public\\images\\1775102166220.jpg"
6   }
7 }
```

Lalu get lagi di postman nya :

HTTP localhost:8080/media Save Share

GET localhost:8080/media Send

Docs Params Authorization Headers (9) Body • Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results 200 OK • 24 ms • 1.14 KB

{ } JSON Preview Visualize

```
29   "id": 5,
30   "image": "localhost:8080/images/public\\images\\1775103748618.jpg",
31   "createdAt": "2026-04-02T04:22:28.000Z",
32   "updatedAt": "2026-04-02T04:22:28.000Z"
33 },
34 {
35   "id": 6,
36   "image": "localhost:8080/images/1775104024284.jpg",
37   "createdAt": "2026-04-02T04:27:04.000Z",
38   "updatedAt": "2026-04-02T04:27:04.000Z"
39 }
40 ]
41 }
```

Kodingan media.js

```
const express = require('express');
const router = express.Router();
const isBase64 = require('is-base64');
const base64Img = require('base64-img');

const { Media } = require('../models');

router.get('/', async (req, res) => {
  const media = await Media.findAll();
  const mappedMedia = media.map((m) => {
    m.image = `${req.get('host')}/${m.image}`;
    return m;
  })
  return res.json({
    status: 'succes',
    data: mappedMedia
  });
});

router.post('/', (req, res) => {
  const image = req.body.image;

  if (!isBase64(image, { mimeRequired: true })) {
    return res.status(400).json({ status: 'error', message: 'invalid base64' });
  }

  base64Img.img(image, './public/images', Date.now(), async (err, filepath) => {
    if (err) {
      return res.status(400).json({ status: 'error', message: err.message });
    }

    const filename = filepath.split('\\').pop().split("/").pop();

    const media = await Media.create({ image: `images/${filename}` });

    return res.json({
      status: 'success',
      data: {
        id: media.id,
        image: `${req.get('host')}/images/${filename}`
      }
    });
  });
});

module.exports = router;
```

lalu tambahkan kodingan

```
const media = await Media.findAll({
  attributes: ['id', 'image']
```

```
});
```

Agar yang tampil atributnya id sama image saja tanpa create at:

Post lagi di postman nya sehingga nanti hasilnya menjadi berikut:

The screenshot shows the Postman interface. At the top, a GET request is configured to `localhost:8080/media`. The 'Send' button is visible. Below the request bar, the 'Params' tab is selected, showing an empty table with columns 'Key', 'Value', and 'Description'. The 'Body' tab is also visible. The response section shows a status of '200 OK' with a response time of 131 ms and a body size of 695 B. The response body is displayed in JSON format:

```
{
  "status": "succes",
  "data": [
    {
      "id": 1,
      "image": "localhost:8080/image/public\\images\\1774533113790.jpg"
    },
    {
      "id": 2,
      "image": "localhost:8080/image/public\\images\\1775102166220.jpg"
    }
  ]
}
```

Ok sampai disini dan dilanjutkan ke methode delete

Method Delete

Sekarang kita akan membuat API untuk menghapus Image

Pada baris paling bawah di media.js sebelum module.export

Ketikan kodingan:

```
router.delete('/:id', async (req, res) => {
  const id = req.params.id;

  const media = await Media.findByPk(id);

  if (!media) {
    return res.status(404).json({ status: 'error', message: 'media not found' });
  }

  fs.unlink(`./public/${media.image}`, async (err) => {
    if (err) {
      return res.status(400).json({ status: 'error', message: err.message });
    }

    await media.destroy();

    return res.json({
      status: 'success',
      message: 'image deleted'
    });
  });
});
```

Dan tambahkan pada baris ke 5

```
const fs = require('fs'); // ! ini (wajib)
```

Pada postman ganti dg methode delete. Misal disini ide 6 yang kita hapus

The screenshot shows the Postman interface for a DELETE request. The URL bar at the top shows 'localhost:8080/media/6'. The method dropdown is set to 'DELETE'. The 'Send' button is visible. Below the URL bar, the 'Params' tab is selected, showing a table with columns 'Key', 'Value', and 'Description'. The 'Body' tab is also visible. The response section at the bottom shows a '200 OK' status, a response time of '33 ms', and a response size of '281 B'. The response body is displayed in JSON format, showing a successful deletion message.

localhost:8080/media/6

DELETE localhost:8080/media/6

Send

Docs Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (7) Test Results

200 OK • 33 ms • 281 B •

{ } JSON Preview Visualize

```
1 {
2   "status": "success",
3   "message": "image deleted"
4 }
```

INTEGRASI DENGAN API GATEWAY – Untuk Praktik Jumat 10 April 2026

Sekarang kita akan mengintegrasikan service media dengan api gateway

Langkah awal pada vs codenya buka folder API Gateway:

Pada media.js

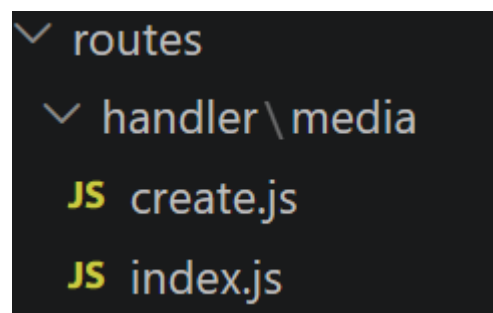
Hapus baris ke 3

```
const { APP_NAME } = process.env;
```

lalu rubah var pada baris ke 1 dan 2 menjadi cons

```
const express = require('express');  
const router = express.Router();
```

lalu buatlah sebuah handler di dalam routes folder handler dan dalam nya buat folder media lalu buat 2 file create.js dan indeks.js:



Lalu lakukan konfigurasi pada .env tambahkan di baris ke 4

```
URL_SERVICE_MEDIA=http://localhost:8080
```

Langkah berikutnya buat create.js isi dengan kode berikut:

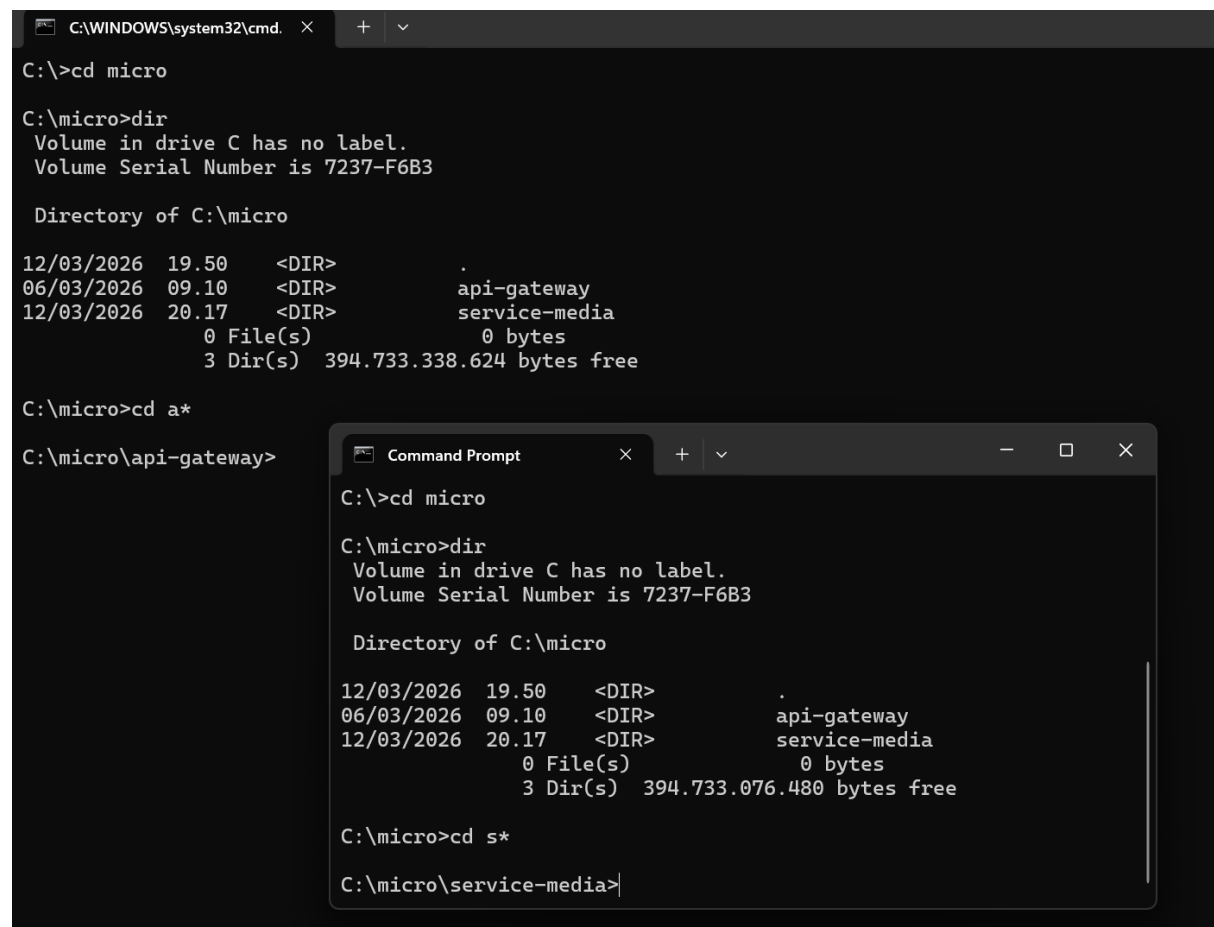
```
const apiAdapter = require('../apiAdapter');  
const {  
  URL_SERVICE_MEDIA  
} = process.env;  
  
const api = apiAdapter(URL_SERVICE_MEDIA);  
  
module.exports = async (req, res) => {  
  try {  
    const media = await api.post('/media', req.body);  
    return res.json(media.data);  
  } catch (error) {  
    const { status, data } = error.response;  
    return res.status(status).json(data);  
  }  
}
```

```
}
```

Dan buat indeks.js didalam handler ketikan kodingan berikut :

```
const create = require('./create');  
  
module.exports = {  
  create  
};
```

Lalu pada cmd buka 2 terminal: api-gateway dan service media



```
C:\>cd micro  
  
C:\micro>dir  
Volume in drive C has no label.  
Volume Serial Number is 7237-F6B3  
  
Directory of C:\micro  
  
12/03/2026  19.50    <DIR>        .  
06/03/2026  09.10    <DIR>        api-gateway  
12/03/2026  20.17    <DIR>        service-media  
               0 File(s)              0 bytes  
               3 Dir(s) 394.733.338.624 bytes free  
  
C:\micro>cd a*  
  
C:\micro\api-gateway>
```

```
Command Prompt  
C:\>cd micro  
  
C:\micro>dir  
Volume in drive C has no label.  
Volume Serial Number is 7237-F6B3  
  
Directory of C:\micro  
  
12/03/2026  19.50    <DIR>        .  
06/03/2026  09.10    <DIR>        api-gateway  
12/03/2026  20.17    <DIR>        service-media  
               0 File(s)              0 bytes  
               3 Dir(s) 394.733.076.480 bytes free  
  
C:\micro>cd s*  
  
C:\micro\service-media>
```

Di baris ke 4 tambahkan kodingan

```
const mediaHandler = require('./handler/media');
```

dan routernya menjadi post pada baris ke 6

```
router.post('/', mediaHandler.create);
```

sehingga kodingan pada media.js menjadi

```
const express = require('express');  
const router = express.Router();
```

```
const mediaHandler = require('./handler/media');  
  
router.post('/', mediaHandler.create);  
  
module.exports = router;
```

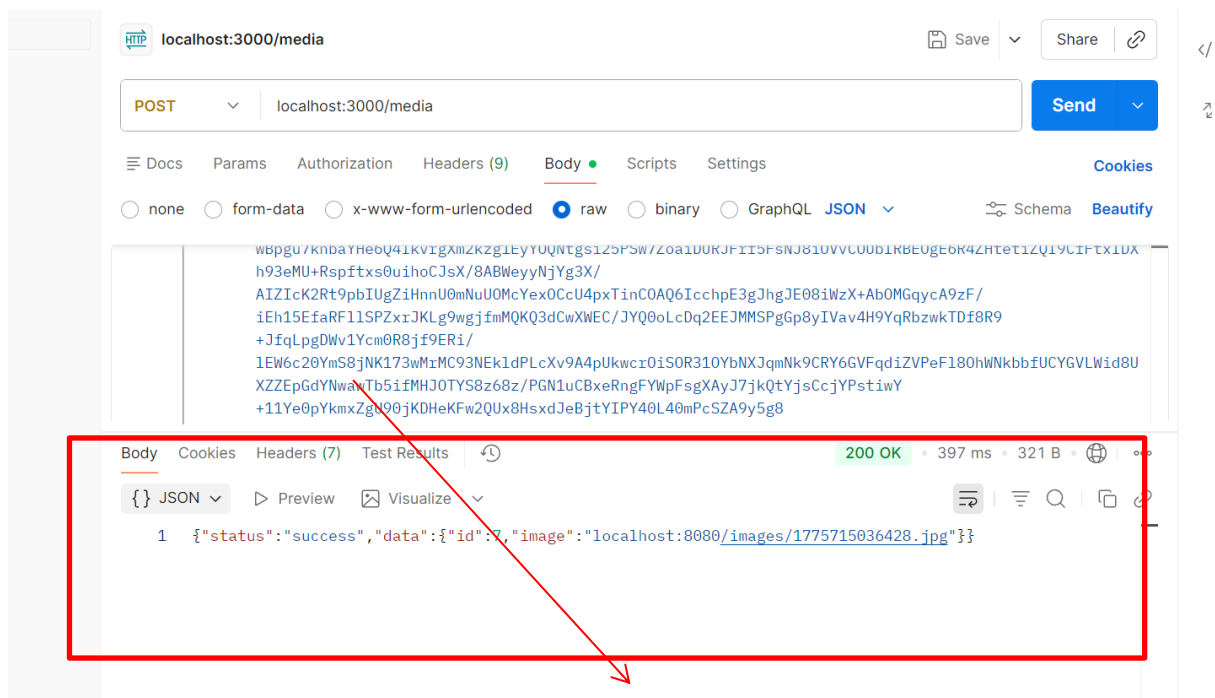
Jangan lupa tambahkan limit reqs pada app.js tambahkan limit 50mb

```
app.use(express.json({limit: '50mb'}));  
app.use(express.urlencoded({ limit: '50mb', extended: true }));
```

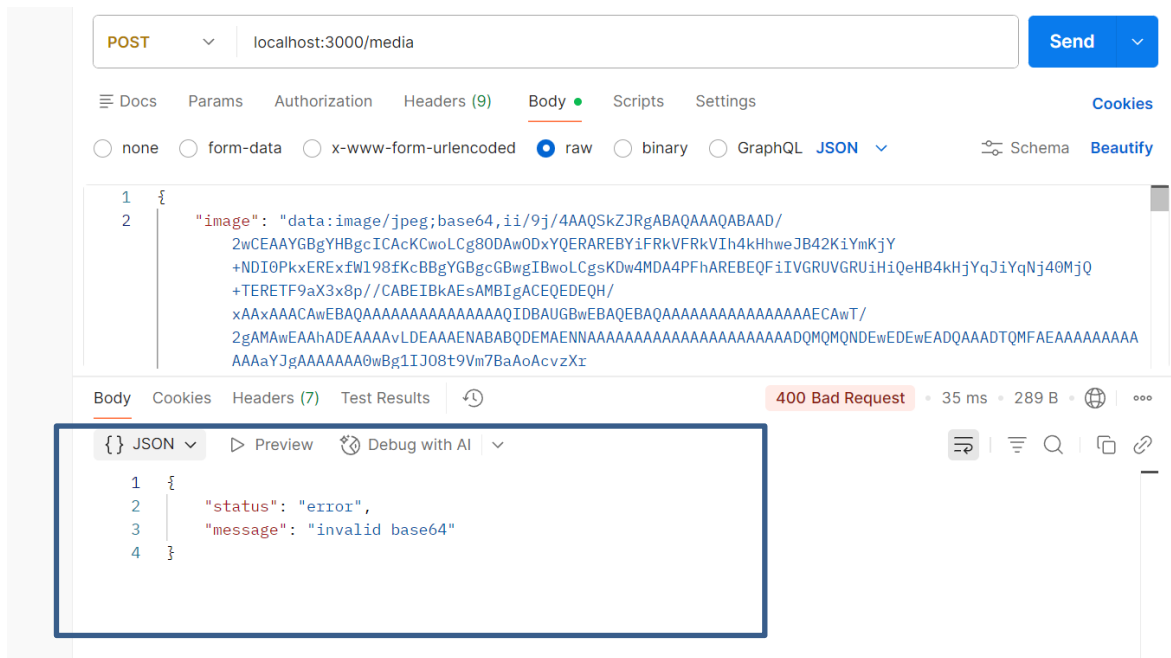
Lalu jalankan kedua server tersebut dengan nodemon bin/www

Dan lakukan pengujian dengan postman Dengan method post localhost:3000/media lalu pada body / raw samakan dg sebelumnya:

Setelah itu baru jalankan di postman: dengan method post hasilnya sebagai berikut



Lalu coba salahkan pada postman dengan mengetikkan sembarangan huruf tambhan di raw json:



Akan muncul keterangan invalid base 64.

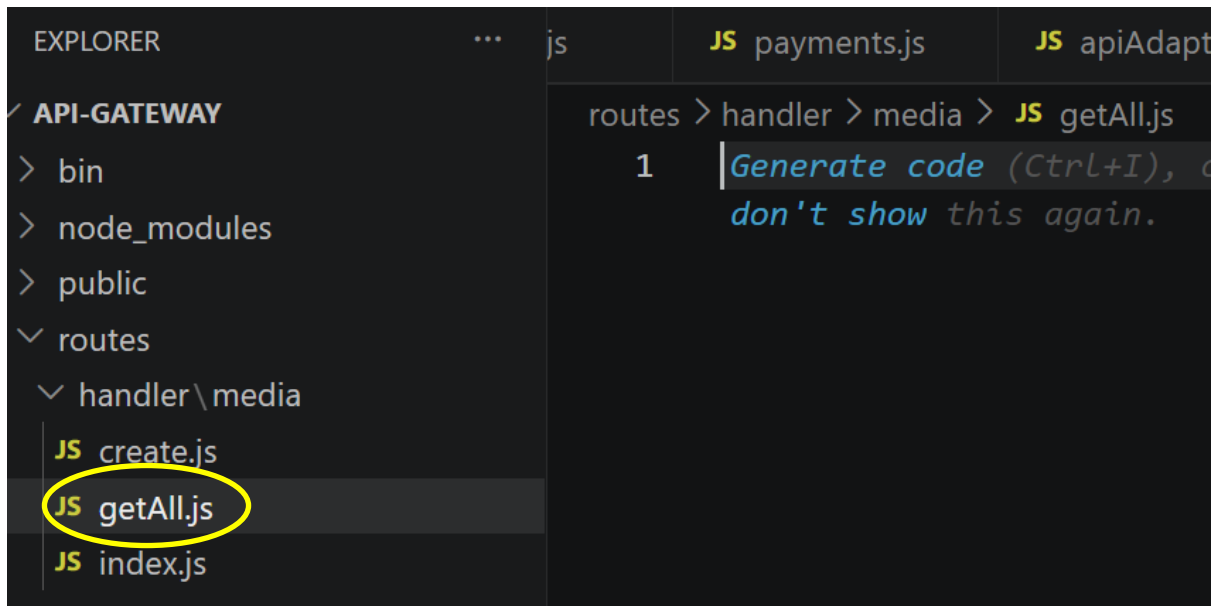
Dan ada keterangan 400 bad request

Yang diharapkan pada tahap ini ada 2 Jika berhasil:

1. **Data akan diteruskan dari API Gateway ke Service Media**
2. **Response berhasil dikembalikan ke Postman**

INTEGRASI DENGAN API GET dengan Service Media – Untuk Praktik Jumat 17 April 2026

Langkah awal pada vs codenya buat getAll.js dalam folder handler/media:



Lalu isikan kodingan berikut:

```
const apiAdapter = require('../apiAdapter');

const {
  URL_SERVICE_MEDIA
} = process.env;

const api = apiAdapter(URL_SERVICE_MEDIA);

module.exports = async (req, res) => {
  try {
    const media = await api.get('/media');
    return res.json(media.data);
  } catch (error) {

    if (error.code === 'ECONNREFUSED') {
      return res.status(500).json({
        status: 'error',
        message: 'service unavailable'
      });
    }

    const { status, data } = error.response;
    return res.status(status).json(data);
  }
};
```

Dan tambahkan pada indeks.js baris ke 2

```
const getAll = require('./getAll');
```

```
module.exports = {  
  create,  
  getAll
```

lalu pada media.js

baris ke 7 tambahkan router.get

```
router.get('/', mediaHandler.getAll);
```

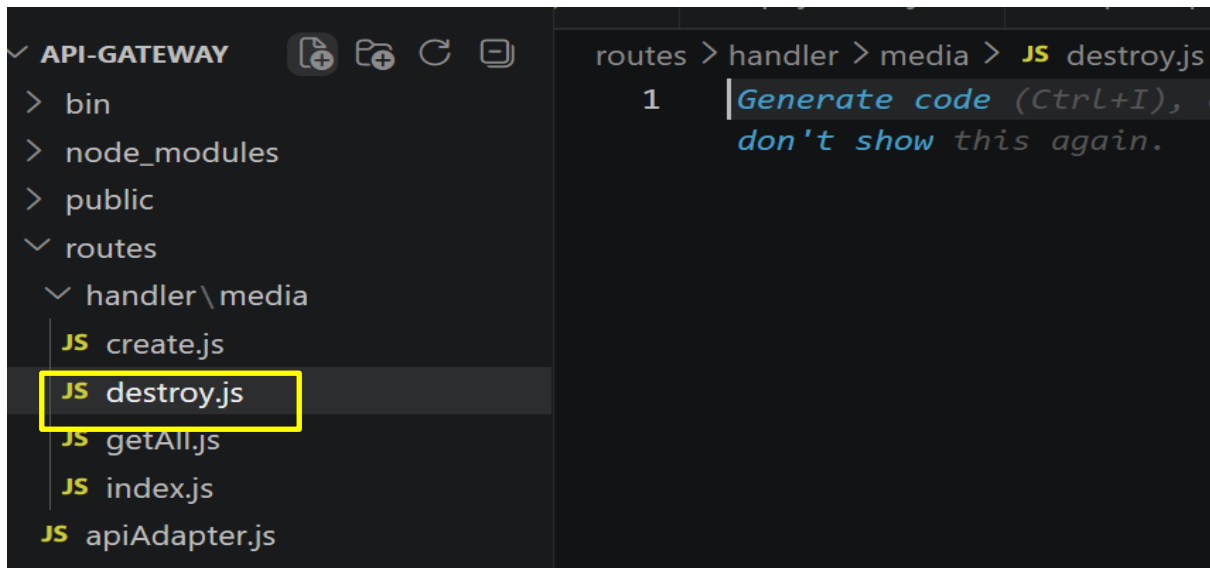
lalu kita cobakan di postman:

The screenshot shows a Postman interface with a GET request to `localhost:3000/media`. The response is a 200 OK status with a response time of 25 ms and a body size of 814 B. The response body is a JSON array of three objects, each containing an `id` and an `image` URL.

```
{  
  "id": 7,  
  "image": "localhost:8080/images/1775715036428.jpg"  
},  
{  
  "id": 8,  
  "image": "localhost:8080/images/1776316916706.jpg"  
},  
{  
  "id": 9,  
  "image": "localhost:8080/images/1776317065489.jpg"  
}]
```

INTEGRASI DENGAN API DELETE dengan Service Media

Langkah awal pada vs codenya buat destroy.js dalam folder handler/media:



Ketikan koding berikut:

```
const apiAdapter = require('../apiAdapter');

const {
  URL_SERVICE_MEDIA
} = process.env;

const api = apiAdapter(URL_SERVICE_MEDIA);

module.exports = async (req, res) => {
  try {
    const id = req.params.id;
    const media = await api.delete(`/media/${id}`);
    return res.json(media.data);
  } catch (error) {

    if (error.code === 'ECONNREFUSED') {
      return res.status(500).json({
        status: 'error',
        message: 'service unavailable'
      });
    }

    const { status, data } = error.response;
    return res.status(status).json(data);
  }
};
```

Lalu pada indeks.js tambahkan kodingan berikut pada baris ke 3

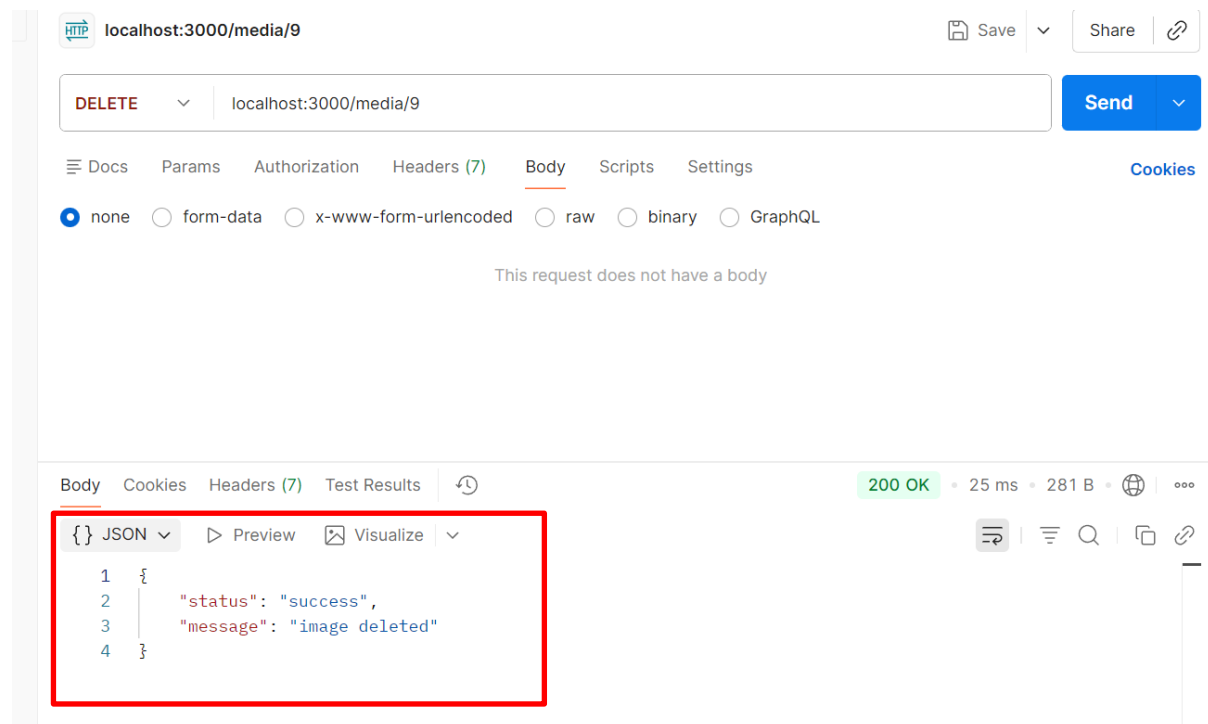
```
const destroy = require('./destroy');
```

```
module.exports = {  
  create,  
  getAll,  
  destroy
```

dan pada media.js juga dilakukan pemanggilan pada routernya pada baris ke 8

```
router.delete('/:id', mediaHandler.destroy);
```

berikutnya kita hapus di postman:



**Dan besok akan dilanjutkan dengan materi
develop Service User**

DEVELOP SERVICE USER

```
C:\micro>dir
Volume in drive C has no label.
Volume Serial Number is 7237-F6B3

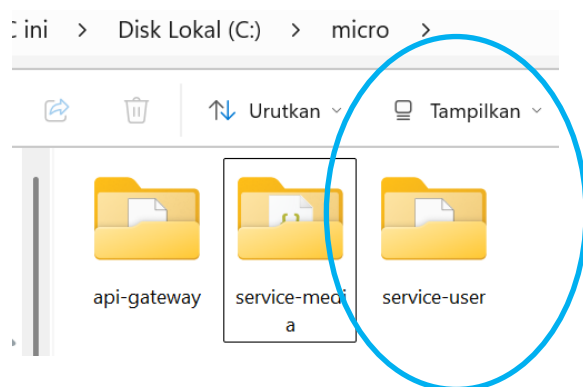
Directory of C:\micro

12/03/2026  19.50    <DIR>          .
06/03/2026  09.10    <DIR>          api-gateway
12/03/2026  20.17    <DIR>          service-media
                0 File(s)                0 bytes
                3 Dir(s)  392.257.470.464 bytes free

C:\micro>
```

Pastikan posisi terminal berada di folder utama micro. Buat project baru menggunakan express-generator dimana sebelumnya sudah ada folder api gateway dan service media

```
express --no-view service-user
```



Masuk ke folder tersebut dan instal dependensi bawaan serta dotenv untuk mengelola variabel lingkungan (environment variables). `npm install` Dan juga `npm install dotenv`

```
C:\micro\service-user>npm install

added 53 packages, and audited 54 packages in 3s

9 vulnerabilities (5 low, 4 high)

To address issues that do not require attention, run:
  npm audit fix

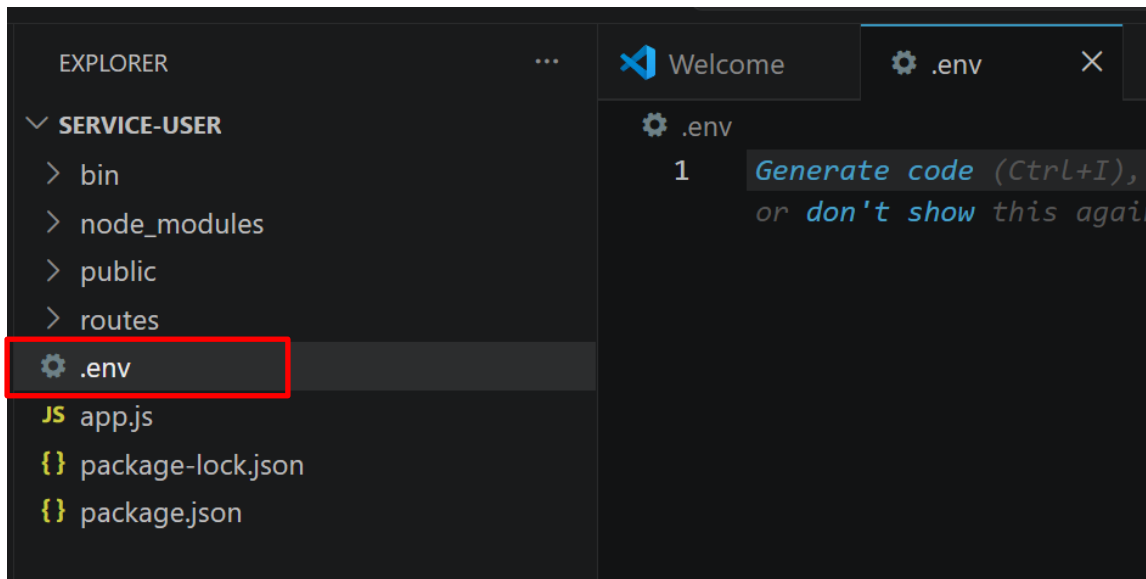
To address all issues, run:
  npm audit fix --force

Run `npm audit` for details.

C:\micro\service-user>
```

Lalu buka vs code direktori service user

Langkah awalnya adalah buat file .env



Buat File .env: Di root folder service-user, buat file bernama .env. Konfigurasi Database: Masukkan detail koneksi berikut:

```
PORT=5000  
  
DB_HOST=127.0.0.1  
DB_USERNAME=root  
DB_PASSWORD=  
DB_NAME=service_user
```

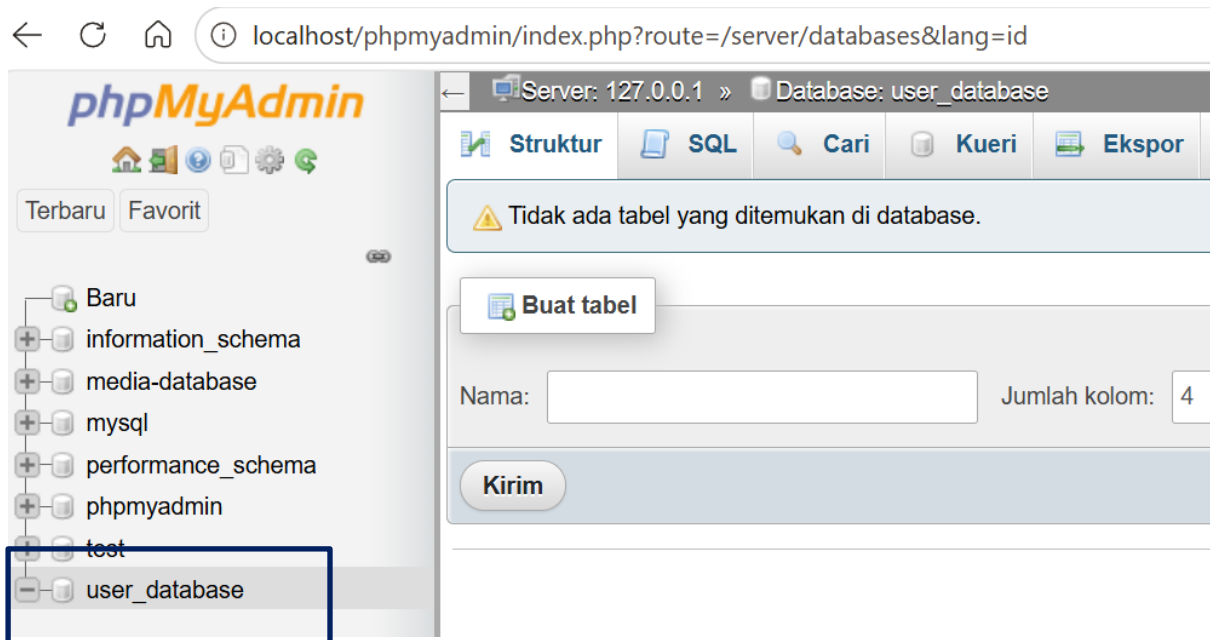
Buka app.js, letakkan konfigurasi dotenv di baris paling atas, dan ubah deklarasi variabel dari var menjadi const.

```
require('dotenv').config();  
const express = require('express');  
const path = require('path');  
const cookieParser = require('cookie-parser');  
const logger = require('morgan');  
  
const indexRouter = require('./routes/index');  
const usersRouter = require('./routes/users');  
  
const app = express();
```

Lakukan hal yang sama pada folder routes /users.js

```
const express = require('express');  
const router = express.Router();
```

sekarang buka php my admin dan buat databasenya user database



Setup sequelize

Kita perlu menginstal library Sequelize dan mysql2 sebagai penghubung (driver) ke database MySQL. Perintahnya `npm install sequelize sequelize-cli --save`

```
C:\micro\service-user>npm install sequelize sequelize-cli --save
npm warn deprecated dottie@2.0.7: Package no longer supported. Contact
s.com/support for more info.
npm warn deprecated glob@10.5.0: Old versions of glob are not supported
ized security vulnerabilities, which have been fixed in the current ver
t for old versions may be purchased (at exorbitant rates) by contacting
added 105 packages, and audited 160 packages in 6s

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

9 vulnerabilities (5 low, 4 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues, run:
  npm audit fix --force

Run 'npm audit' for details.

C:\micro\service-user>
```

Dan langkah berikutnya adalah Inisialisasi Folder: Jalankan `npx sequelize-cli init.`

```
C:\micro\service-user>npx sequelize-cli init

Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

Created "config\config.json"
Successfully created models folder at "C:\micro\service-user\models".
Successfully created migrations folder at "C:\micro\service-user\migrations".
Successfully created seeders folder at "C:\micro\service-user\seeders".

C:\micro\service-user>
```

Buka vs code rubah config.json menjadi **config.js** dan Kodingannya pada config.js

```
require('dotenv').config();

const {
  DB_HOST,
  DB_PASSWORD,
  DB_USERNAME,
  DB_NAME
} = process.env;

module.exports = {
  "development": {
    "username": DB_USERNAME,
    "password": DB_PASSWORD,
    "database": DB_NAME,
    "host": DB_HOST,
    "dialect": "mysql"
  },
  "test": {
    "username": DB_USERNAME,
    "password": DB_PASSWORD,
    "database": DB_NAME,
    "host": DB_HOST,
    "dialect": "mysql"
  },
  "production": {
    "username": DB_USERNAME,
    "password": DB_PASSWORD,
    "database": DB_NAME,
    "host": DB_HOST,
    "dialect": "mysql"
  }
}
```

Buka indeks.js didalam models ubah di baris ke 9 config.json menjadi config.js

```
const config = require(__dirname + '/../config/config.js')(env);
```

1. config: Tempat menyimpan rahasia dapur (koneksi DB).
2. models: Representasi tabel dalam bentuk objek JavaScript.

3. migrations: "Version Control" untuk database.

Video 5: Migration Table Users

Migration adalah cara kita "coding" database tanpa perlu menyentuh SQL di PHPMyAdmin.

kodingan nya `npx sequelize migration:create --name=create-table-users`

```
C:\micro\service-user>npx sequelize migration:create --name=create-table-users
Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

migrations folder at "C:\micro\service-user\migrations" already exists.
New migration was created at C:\micro\service-user\migrations\20260417024942-c
.

C:\micro\service-user>
```

Berikutnya kita buat table refresh token

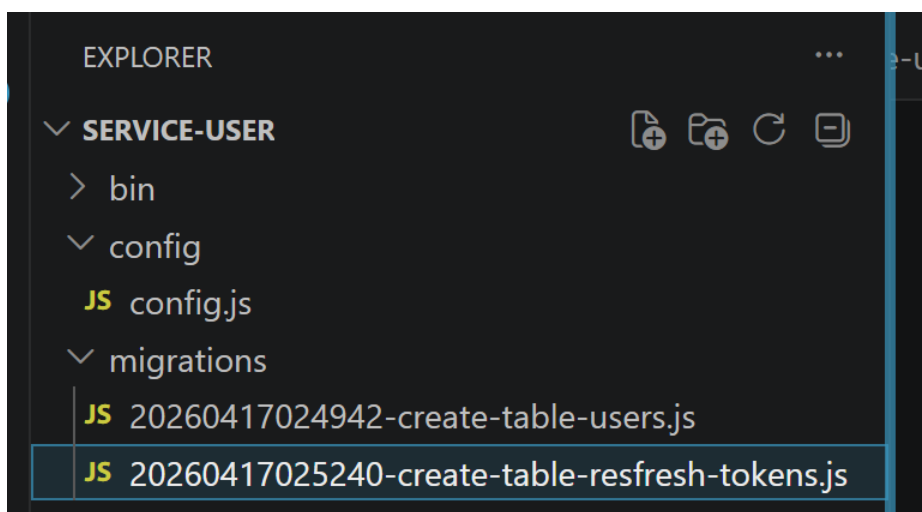
`npx sequelize migration:create --name=create-table-resfresh-tokens`

```
C:\micro\service-user>npx sequelize migration:create --name=create-table-resfresh-tokens
Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

migrations folder at "C:\micro\service-user\migrations" already exists.
New migration was created at C:\micro\service-user\migrations\20260417025240-create-table-resfresh-tokens.js
.

C:\micro\service-user>
```

Sekarang buka vs code lihat di folder migration



Kita isikan koding pada table/users.js

```
'use strict';
```

```
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('users', {
      id: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        primaryKey: true,
        allowNull: false
      },
      name: {
        type: Sequelize.STRING,
        allowNull: false
      },
      profession: {
        type: Sequelize.STRING,
        allowNull: true
      },
      avatar: {
        type: Sequelize.STRING,
        allowNull: true
      },
      role: {
        type: Sequelize.ENUM,
        values: ['admin', 'student'],
        allowNull: false
      },
      email: {
        type: Sequelize.STRING,
        allowNull: false
      },
      password: {
        type: Sequelize.STRING,
        allowNull: false
      },
      created_at: {
        type: Sequelize.DATE,
        allowNull: false
      },
      updated_at: {
        type: Sequelize.DATE,
        allowNull: false
      }
    });

    await queryInterface.addConstraint('users', ['email'], {
      type: 'unique',
      name: 'UNIQUE_USERS_EMAIL'
    });
  },
  down: async (queryInterface, Sequelize) => {
    await queryInterface.dropTable('users');
  }
};
```

Berikutnya di terminal ketikkan **npx sequelize db:migrate**

```
C:\micro\service-user>npx sequelize db:migrate

Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

◇ injected env (5) from .env // tip: ⚠ multiple files { path:
Loaded configuration file "config/config.js".
Using environment "development".

ERROR: Please install mysql2 package manually

C:\micro\service-user>
```

Setelah itu **npm install mysql2 --save**

```
C:\micro\service-user>npm install mysql2 --save

added 10 packages, and audited 170 packages in 2s

22 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (5 low, 4 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues, run:
  npm audit fix --force

Run `npm audit` for details.

C:\micro\service-user>
```

Jika berhasil migrate

```
C:\micro\service-user>npx sequelize db:migrate

Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

◇ injected env (5) from .env // tip: ◇ secrets for agents [www.dotenvx.com]
Loaded configuration file "config/config.js".
Using environment "development".
== 20260417024942-create-table-users: migrating =====
== 20260417024942-create-table-users: migrated (0.072s)

== 20260417025240-create-table-resfresh-tokens: migrating =====
== 20260417025240-create-table-resfresh-tokens: migrated (0.008s)

C:\micro\service-user>
```

Cek di php my admin

← ↻ 🏠 ⓘ localhost/phpmyadmin/index.php?route=/database/structure&server=1&db=user_database

phpMyAdmin

Terbaru Favorit

- Baru
- information_schema
- media-database
- mysql
- performance_schema
- phpmyadmin
- test
- user_database
 - Baru
 - sequelizemeta
 - users

Server: 127.0.0.1 » Database: user_database

Struktur SQL Cari Kueri Ekspor Impor Operasi

Filters

Mengandung kata:

Tabel	Tindakan
☐ sequelizemeta	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus
☐ users	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus

2 tabel Jumlah

☐ Pilih Semua Dengan pilihan: ▼

Cetak Kamus data

Jumat 24 April Video 6: Migration Table Refresh Tokens

Tabel ini berfungsi menyimpan token agar user tidak perlu login berulang kali. Sekarang buka dulu file migration refresh token dan ketikkan koding berikut

```
'use strict';

module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('refresh_tokens', {
      id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false
      },
      token: {
        type: Sequelize.TEXT,
        allowNull: false
      },
      user_id: {
        type: Sequelize.INTEGER,
        allowNull: false
      },
      created_at: {
        type: Sequelize.DATE,
        allowNull: false
      },
      updated_at: {
        type: Sequelize.DATE,
        allowNull: false
      }
    });

    await queryInterface.addConstraint('refresh_tokens', {
      type: 'foreign key',
      name: 'REFRESH_TOKENS__USER_ID',
      fields: ['user_id'],
      references: {
        table: 'users',
        field: 'id'
      },
      onDelete: 'cascade',
      onUpdate: 'cascade'
    });
  },

  down: async (queryInterface, Sequelize) => {
    // Fungsi ini untuk menghapus tabel jika migrasi di-rollback
    await queryInterface.dropTable('refresh_tokens');
  }
}
```

```
}  
npx sequelize db:migrate undo:all
```

```
C:\micro\service-user>npx sequelize db:migrate:undo:all  
  
Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]  
  
◇ injected env (5) from .env // tip: % enable debugging { debug: true }  
Loaded configuration file "config\config.js".  
Using environment "development".  
== 20260417025240-create-table-resfresh-tokens: reverting =====  
== 20260417025240-create-table-resfresh-tokens: reverted (0.025s)  
  
== 20260417024942-create-table-users: reverting =====  
== 20260417024942-create-table-users: reverted (0.045s)
```

npx sequelize db:migrate

```
C:\micro\service-user>npx sequelize db:migrate  
  
Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]  
  
◇ injected env (5) from .env // tip: ◇ secrets for agents [www.dotenvx.com]  
Loaded configuration file "config\config.js".  
Using environment "development".  
== 20260417024942-create-table-users: migrating =====  
== 20260417024942-create-table-users: migrated (0.113s)  
  
== 20260417025240-create-table-resfresh-tokens: migrating =====  
== 20260417025240-create-table-resfresh-tokens: migrated (0.107s)  
  
C:\micro\service-user>
```

Nanti akan muncul table refresh token

← ↻ 🏠 ⓘ localhost/phpmyadmin/index.php?route=/table/structure&dl

phpMyAdmin

🏠 📄 ⚙️ 🔄

Terbaru Favorit

- Baru
- + information_schema
- + media-database
- + mysql
- + performance_schema
- + phpmyadmin
- + test
- user_database
 - Baru
 - + refresh_tokens
 - + sequelizemeta
 - + users

← Server: 127.0.0.1 » Database: user

Jelajahi Struktur SQL

Struktur tabel Tampilan hul

	#	Nama	Jenis
☐	1	id 🔑	int(11)
☐	2	name	varchar(255)
☐	3	profession	varchar(255)
☐	4	avatar	varchar(255)
☐	5	role	enum('admin', 'student')
☐	6	email 🔑	varchar(255)
☐	7	password	varchar(255)
☐	8	created_at	datetime
☐	9	updated_at	datetime

Membuat Seeders

Seeder sangat membantu agar kita tidak perlu melakukan input manual melalui form yang belum jadi. (Ini kegunaannya) sekarang kita ketik pada cmd perintah berikut

`npx sequelize seed:create --name user-seeders`

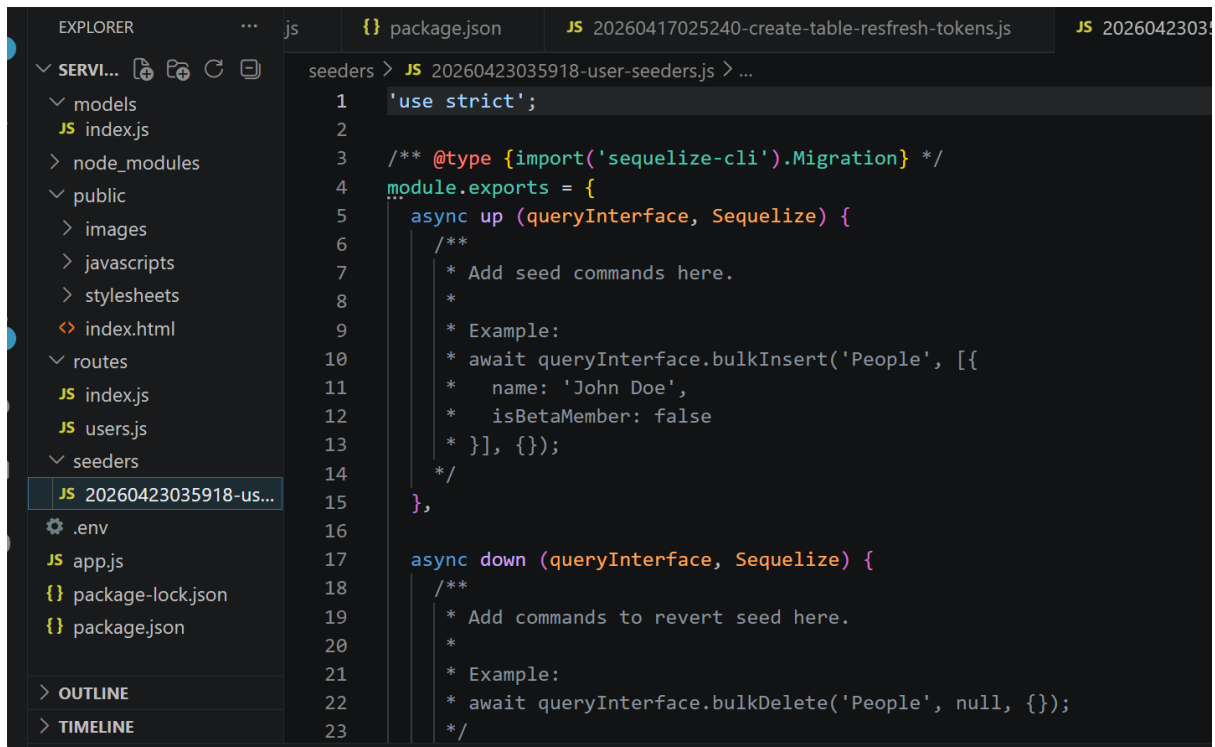
```
C:\micro\service-user>npx sequelize seed:create --name user-seeders

Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

seeders folder at "C:\micro\service-user\seeders" already exists.
New seed was created at C:\micro\service-user\seeders\20260423035918-user-seeders.js .

C:\micro\service-user>
```

nanti dia akan membuat sebuah file di dalam seeder yang bisa di cek di visual studio code. Dan jika berhasil akan tampil seperti ini



```
EXPLORER  ...  js  {} package.json  JS 20260417025240-create-table-refresh-tokens.js  JS 20260423035918-user-seeders.js

SEVER...  models
  JS index.js
node_modules
public
  images
  javascripts
  stylesheets
  index.html
routes
  JS index.js
  JS users.js
seeders
  JS 20260423035918-user-seeders.js
.env
JS app.js
{} package-lock.json
{} package.json

OUTLINE
TIMELINE

seeders > JS 20260423035918-user-seeders.js > ...
1  'use strict';
2
3  /** @type {import('sequelize-cli').Migration} */
4  module.exports = {
5    async up (queryInterface, Sequelize) {
6      /**
7       * Add seed commands here.
8       *
9       * Example:
10      * await queryInterface.bulkInsert('People', [{
11      *   name: 'John Doe',
12      *   isBetaMember: false
13      * }], {});
14    },
15
16    async down (queryInterface, Sequelize) {
17      /**
18       * Add commands to revert seed here.
19       *
20       * Example:
21       * await queryInterface.bulkDelete('People', null, {});
22     }
23   }
```

Ganti kodingan pada seeders tersebut:

```
'use strict';
const bcrypt = require('bcrypt');

module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.bulkInsert('users', [
      {
        name: "widada",
        profession: "Admin Micro",
        role: "admin",
        email: "juarakoding@gmail.com",
        password: ""
```

```
    created_at: new Date(),
    updated_at: new Date()
  },
  {
    name: "Yein",
    profession: "Front End Developer",
    role: "student", // Perbaikan typo dari "stundet"
    email: "yein@mail.com",
    password: "",
    created_at: new Date(),
    updated_at: new Date()
  }
]);
},

down: async (queryInterface, Sequelize) => {
  await queryInterface.bulkDelete('users', null, {});
}
};
```

Lalu ketikkan pada terminal **npx sequelize db:seed:all**

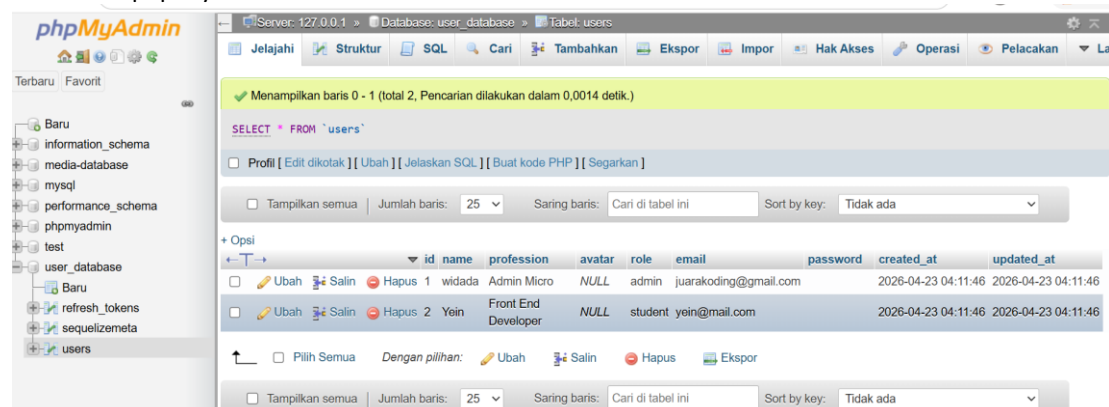
```
C:\micro\service-user>npx sequelize db:seed:all

Sequelize CLI [Node: 22.14.0, CLI: 6.6.5, ORM: 6.37.8]

◇ injected env (5) from .env // tip: ◇ encrypted .env [www.dotenvx.com]
Loaded configuration file "config\config.js".
Using environment "development".
== 20260423035918-user-seeders: migrating =====
== 20260423035918-user-seeders: migrated (0.016s)

C:\micro\service-user>
```

Lalu cek di php my admin



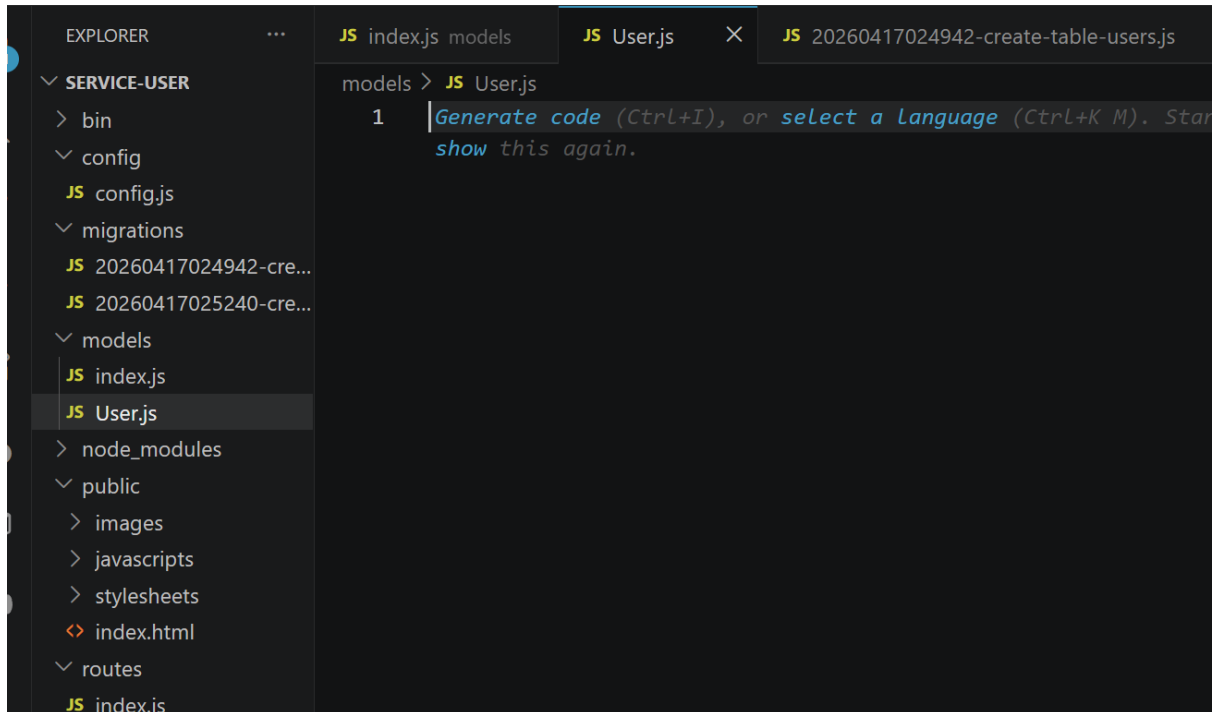
	id	name	profession	avatar	role	email	password	created_at	updated_at
☐	1	widada	Admin Micro	NULL	admin	juarakoding@gmail.com		2026-04-23 04:11:46	2026-04-23 04:11:46
☐	2	Yein	Front End Developer	NULL	student	yein@mail.com		2026-04-23 04:11:46	2026-04-23 04:11:46

Silahkan hapus dengan undo. Dan ulangi create kembali..

Membuat Model User dan Model Refresh Token

Model adalah tempat kita mengatur logika data, seperti enkripsi password atau validasi input.

Langkahnya adalah pada vs code di dalam folder Models buat User.js



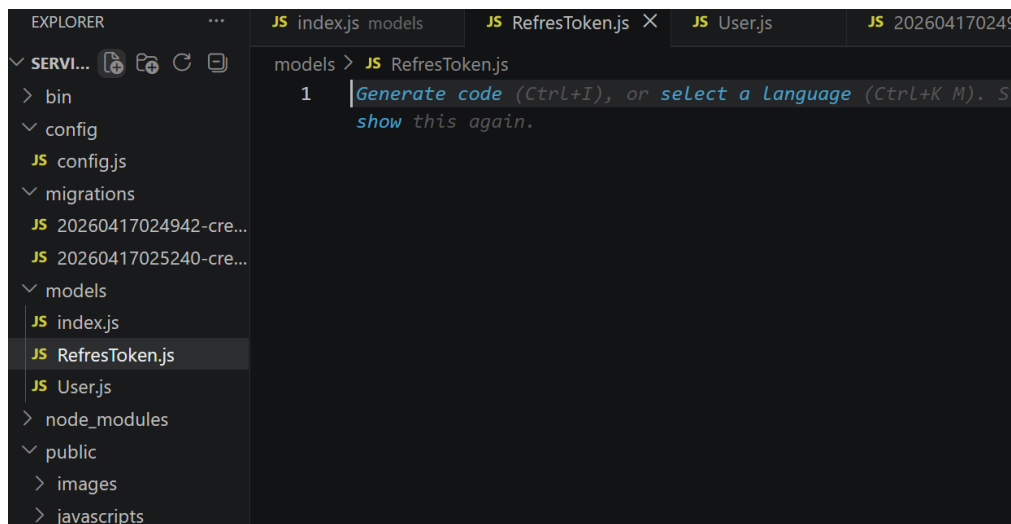
Lalu isikan kodingan berikut

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
      allowNull: false
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false
    },
    role: {
      type: DataTypes.ENUM,
      values: ['admin', 'student'],
    }
  });
```

```
    allowNull: false,
    defaultValue: 'student'
  },
  avatar: {
    type: DataTypes.STRING,
    allowNull: true
  },
  profession: {
    type: DataTypes.STRING,
    allowNull: true
  },
  createdAt: {
    field: 'created_at',
    type: DataTypes.DATE,
    allowNull: false
  },
  updatedAt: {
    field: 'updated_at',
    type: DataTypes.DATE,
    allowNull: false
  }
}, {
  tableName: 'users',
  timestamps: true
});

return User;
};
```

Langkahnya Refres Token berikutnya adalah buat RefresToken.js



Dan berikut kodingannya :

```
module.exports = (sequelize, DataTypes) => {
  const RefreshToken = sequelize.define('RefreshToken', {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
```

```
        primaryKey: true,
        allowNull: false
    },
    token: {
        type: DataTypes.TEXT,
        allowNull: false,
    },
    user_id: {
        type: DataTypes.INTEGER,
        allowNull: false,
    },
    createdAt: {
        type: DataTypes.DATE,
        field: 'created_at',
        allowNull: false,
    },
    updatedAt: {
        type: DataTypes.DATE,
        field: 'updated_at',
        allowNull: false,
    }
}, {
    tableName: 'refresh_tokens',
    timestamps: true
});

return RefreshToken;
}
```

Materi Bulan Mei 2026 adalah membuat API Register, API Login, API Update, API Get User by id, API Get Users, API Save Refresh Token, API Get Refresh Token, dan API Logout. Baru setelah itu UTS.

Struktur API yang Akan Dibangun

Endpoint	Method	Fungsi
/register	POST	Mendaftarkan user baru (Hashing password).
/login	POST	Validasi user & generate Access + Refresh Token.
/users	GET	Mengambil semua data user (Admin only).
/users/:id	GET	Mengambil detail user berdasarkan ID.
/users/:id	PUT	Memperbarui profil user.
/refresh-token	POST	Menukar Refresh Token lama menjadi Access Token baru.
/logout	DELETE	Menghapus Refresh Token dari database.

MEMBUAT API REGISTER DENGAN EXPRESS.JS DAN MYSQL

Pada praktikum ini mahasiswa akan belajar membuat API Register pada service user menggunakan:

1. Express.js
2. Sequelize
3. MySQL
4. Fastest Validator
5. Bcrypt

Konsep Dasar API Register

Saat user melakukan register:

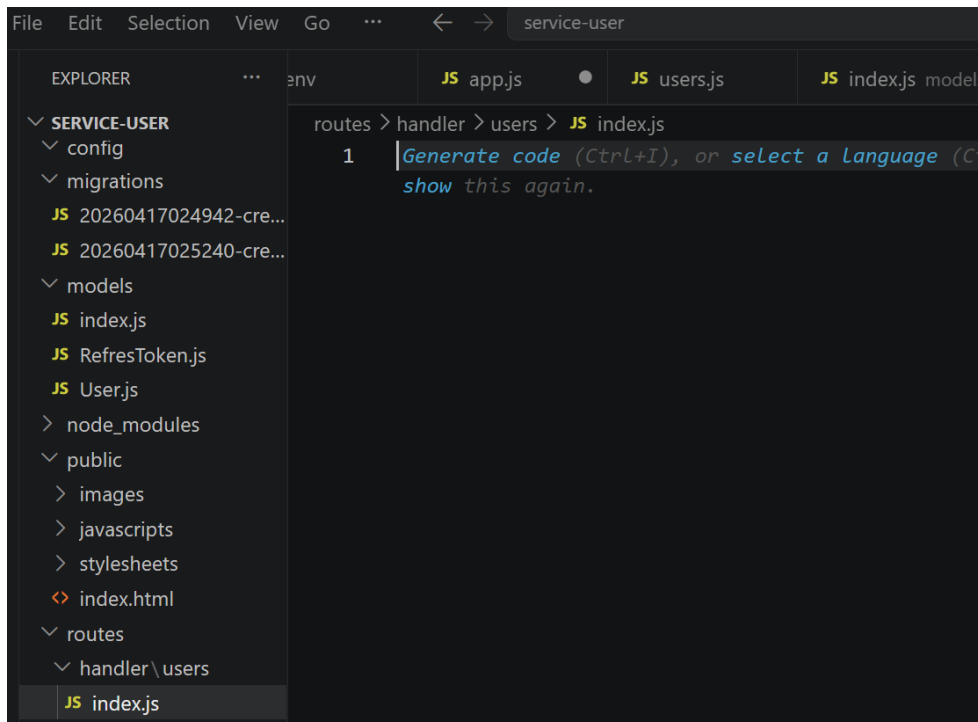
1. User mengirim data:
 - a. nama
 - b. email
 - c. password
 - d. profesi
2. Server melakukan validasi
3. Server mengecek apakah email sudah ada
4. Password dienkripsi menggunakan bcrypt
5. Data disimpan ke database
6. Server mengirim response sukses

Langkah pertamanya adalah **Membuat Folder Handler**

Masuk ke folder: routes Buat folder baru: handler Di dalam folder handler, buat folder lagi: users

Di dalam folder users, buat 2 file:

1. index.js
2. register.js



Berikutnya Install Fastest Validator

Dengan cara mengetikkan ini pada terminal **npm install fastest-validator--save**

```
C:\micro\service-user>npm install fastest-validator
added 1 package, and audited 171 packages in 2s

22 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (5 low, 4 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues, run:
  npm audit fix --force

Run `npm audit` for details.

C:\micro\service-user>
```

Membuat Koding Awal register.js

```
const bcrypt = require('bcrypt');
const { User } = require('../models');
const Validator = require('fastest-validator');
const v = new Validator();

module.exports = async (req, res) => {
  const schema = {
    name: 'string|empty:false',
```

```
email: 'email|empty:false',
password: 'string|min:6',
profession: 'string|optional'
}

const validate = v.validate(req.body, schema);

if (validate.length) {
  return res.status(400).json({
    status: 'error',
    message: validate
  });
}
}
```

Menjalankan Server bisa dengan npm start atau nodemon

```
C:\micro\service-user>nodemon bin/www
[nodemon] 3.1.14
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node bin/www`
◇ injected env (5) from .env // tip: ⌘ enable debugg
|
```

Tambahkan dulu pada baris ke 4 users.js

```
const usersHandler = require('./handler/users');
```

sehingga menjadi seperti ini

```
const express = require('express');
const router = express.Router();

const usersHandler = require('./handler/users');

router.post('/register', usersHandler.register);

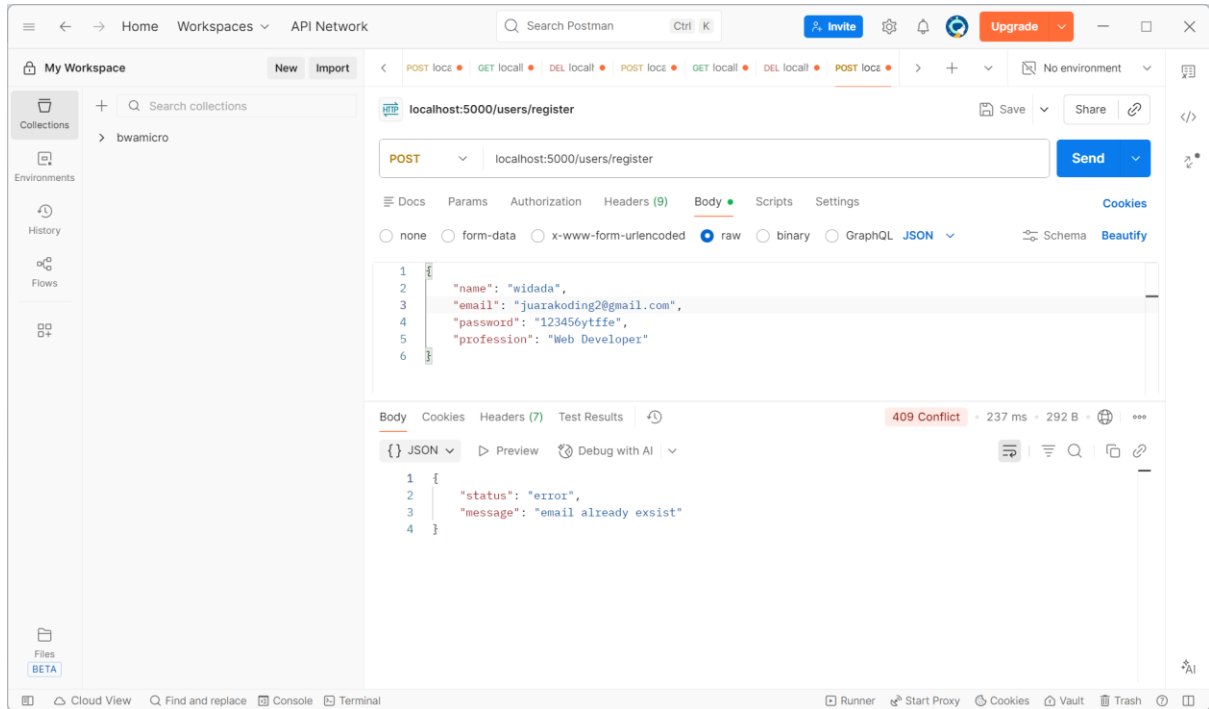
module.exports = router;
```

Mengisi File index.js dengan kodingan berikut

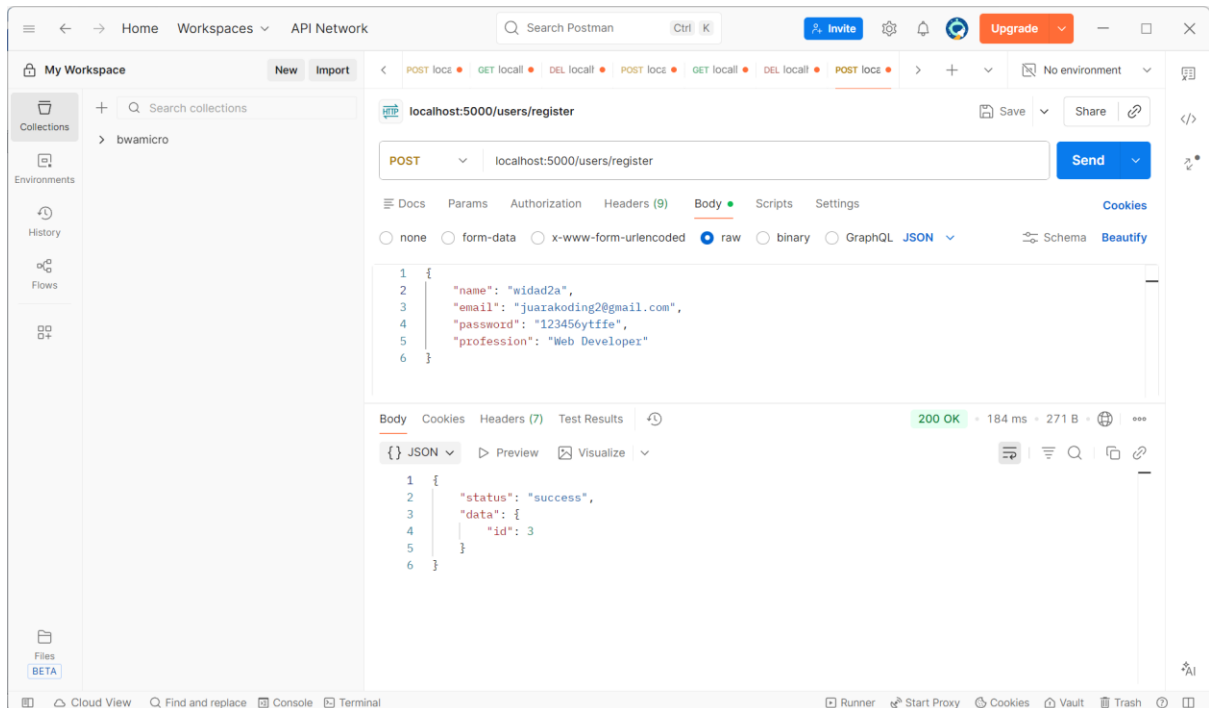
```
const register = require('./register');

module.exports = {
  register
}
```

Lakukan langkah berikut Menguji API di Postman



Sampai berhasil sukses



Berikut kodingan lengkap register.js

```
const bcrypt = require('bcrypt');
const { User } = require('../models');
const Validator = require('fastest-validator');
const v = new Validator();

module.exports = async (req, res) => {
  const schema = {
    name: 'string|empty:false',
    email: 'email|empty:false',
    password: 'string|min:6',
    profession: 'string|optional'
  }

  const validate = v.validate(req.body, schema);

  if (validate.length) {
    return res.status(400).json({
      status: 'error',
      message: validate
    });
  }

  const user = await User.findOne({
    where: { email: req.body.email }
  });

  if (user) {
    return res.status(409).json({
      status: 'error',
      message: 'email already exist'
    });
  }

  const password = await bcrypt.hash(req.body.password, 10);

  const data = {
    password,
    name: req.body.name,
    email: req.body.email,
    profession: req.body.profession,
    role: 'student'
  };

  const createdUser = await User.create(data);

  return res.json({
    status: 'success',
    data: {
      id: createdUser.id
    }
  });
}
```

Terakhir Cek Data di phpMyAdmin

The screenshot shows the phpMyAdmin web interface in a browser. The address bar indicates the URL: `localhost/phpmyadmin/index.php?route=/sql&server=1&db=user_database&table=users&pos=0`. The interface is in Indonesian. On the left, a sidebar shows the database structure with 'user_database' selected and the 'users' table highlighted. The main panel displays the 'users' table with a message: 'Menampilkan baris 0 - 2 (total 3, Pencarian dilakukan dalam 0,0016 detik.)'. Below this, a SQL query is shown: `SELECT * FROM `users``. The table data is displayed with columns: id, name, profession, avatar, role, email, and password. There are three rows of data. At the bottom, there are options for 'Operasi hasil kueri' (Print, Copy to clipboard, Export, Show diagram, Generate view) and a 'Markahi kueri SQL ini' (Mark this SQL query) button.

	id	name	profession	avatar	role	email	password
☐	1	widada	Admin Micro	NULL	admin	juarakoding@gmail.com	
☐	2	Yein	Front End Developer	NULL	student	yein@mail.com	
☐	3	widad2a	Web Developer	NULL	student	juarakoding2@gmail.com	\$2b\$10\$cQ3zCEQAPY6HqKvu/Rzar.DMnKV/IG.NCsuiQvU2k

Membuat API LOGIN

Langkah pertamanya adalah buat login.js didalam handler

```
const bcrypt = require('bcrypt');
const { User } = require('../models');
const Validator = require('fastest-validator');
const v = new Validator();

module.exports = async (req, res) => {
  const schema = {
    email: 'email|empty:false',
    password: 'string|min:6'
  }

  const validate = v.validate(req.body, schema);
  if (validate.length) {
    return res.status(400).json({
      status: 'error',
      message: validate
    });
  }

  const user = await User.findOne({
    where: { email: req.body.email }
  });

  if (!user) {
    return res.status(404).json({
      status: 'error',
      message: 'user not found'
    });
  }

  const isValidPassword = await bcrypt.compare(req.body.password, user.password);
  if (!isValidPassword) {
    return res.status(404).json({
      status: 'error',
      message: 'user not found' // Sengaja disamakan dengan pesan di atas demi keamanan (agar hacker
    tidak tahu emailnya terdaftar atau tidak)
    });
  }

  return res.status(200).json({
    status: 'success',
    data: {
      id: user.id,
      name: user.name,
      email: user.email,
      role: user.role,
      avatar: user.avatar,
      profession: user.profession
    }
  });
}
```

```
}
```

Lalu isi kodingan pada users.jsnya

```
const express = require('express');
const router = express.Router();

const usersHandler = require('./handler/users');

router.post('/register', usersHandler.register);
router.post('/login', usersHandler.login);

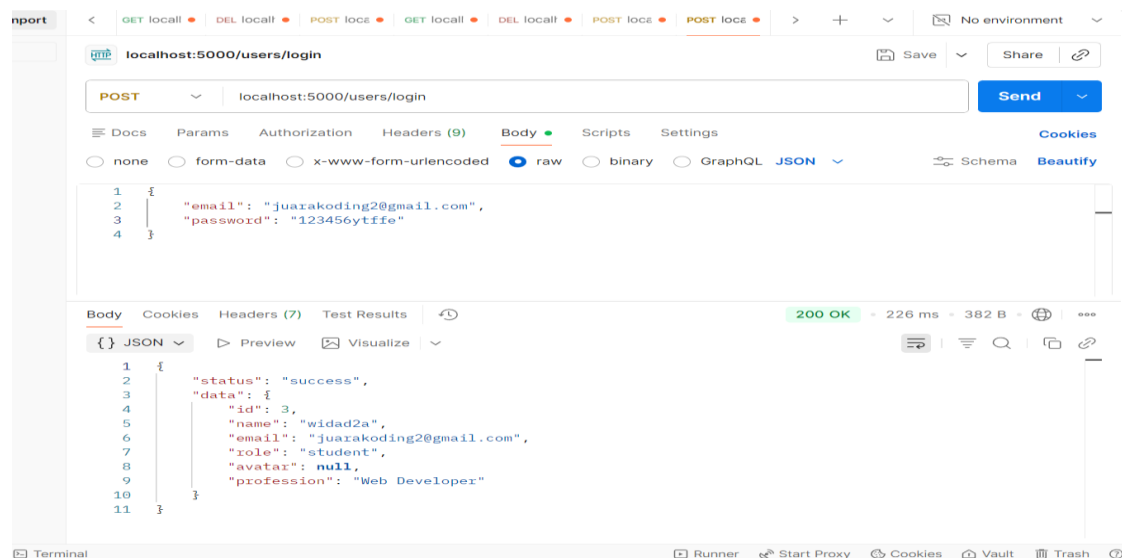
module.exports = router;
```

dan indeks.js

```
const register = require('./register')
const login = require('./login')

module.exports = {
  register,
  login
}
```

Ketika berhasil



Membuat API UPDATE

Langkah pertamanya adalah buat update.js didalam handler

```
const bcrypt = require('bcrypt');
const { User } = require('../models');
const Validator = require('fastest-validator');
const v = new Validator();

module.exports = async (req, res) => {
  // 1. Validasi Input Data
  const schema = {
    name: 'string|empty:false',
    email: 'email|empty:false',
    password: 'string|min:6',
    profession: 'string|optional',
    avatar: 'string|optional'
  };

  const validate = v.validate(req.body, schema);
  if (validate.length) {
    return res.status(400).json({
      status: 'error',
      message: validate
    });
  }

  // 2. Cek Apakah User Eksis di Database
  const id = req.params.id;
  const user = await User.findById(id);
  if (!user) {
    return res.status(404).json({
      status: 'error',
      message: 'user not found'
    });
  }

  // 3. Validasi Email (Jika Email Diubah, Pastikan Belum Dipakai Orang Lain)
  const email = req.body.email;
  if (email) {
    const checkEmail = await User.findOne({
      where: { email }
    });

    if (checkEmail && email !== user.email) {
      return res.status(409).json({ // Perbaikan typo dari res.json(409).json
        status: 'error',
        message: 'email already exist'
      });
    }
  }

  // 4. Hashing Password & Destructuring Request Body
  const password = await bcrypt.hash(req.body.password, 10);
```

```
const {
  name,
  profession,
  avatar
} = req.body;

// 5. Proses Update ke Database
await user.update({
  email,
  password,
  name,
  profession,
  avatar
});

// 6. Mengirimkan Response Sukses
return res.json({
  status: 'success',
  data: {
    id: user.id,
    name,
    email,
    profession,
    avatar
  }
});
}
```

Tambahkan pada indeks.js

```
const update = require('./update')
```

dan di baris 8 pada users.js

```
router.put('/:id', usersHandler.update);
```

lalu testing di postman menggunakan method put

Dan jika berhasil akan terupdate user nomor 3

The screenshot displays a REST client interface with a sequence of requests at the top: DEL localhost, POST localhost, GET localhost, DEL localhost, POST localhost, POST localhost, and PUT localhost (highlighted). The main area shows a PUT request to `localhost:5000/users/3`. The request body is a JSON object:

```
1 {
2   "name": "widada3",
3   "email": "padangxipunggasan@gmail.com",
4   "password": "121212121"
5 }
```

The response section shows a `200 OK` status with a response time of 189 ms and a body size of 326 B. The response body is a JSON object:

```
1 {
2   "status": "success",
3   "data": {
4     "id": 3,
5     "name": "widada3",
6     "email": "padangxipunggasan@gmail.com"
7   }
8 }
```

Membuat API Get User by id

Langkah pertamanya adalah buat getUser.js didalam handler

Jika berhasil

The screenshot shows a REST client interface with a list of requests at the top. The selected request is a GET request to `localhost:5000/users/3`. The response is a 200 OK status with a response time of 12 ms and a body size of 387 B. The response body is displayed in JSON format, showing a successful status and user data for ID 3.

Query Params

Key	Value	Description
Key	Value	Description

Body

```
1 {
2   "status": "success",
3   "data": {
4     "id": 3,
5     "name": "widada3",
6     "email": "padangxipunggasan@gmail.com",
7     "role": "student",
8     "profession": "Web Developer",
9     "avatar": null
10  }
11 }
```

Membuat API Get Users by id

Langkah pertamanya adalah buat getUsers.js didalam handle

Berikut kodingan nya

```
const { User } = require("../models");

module.exports = async (req, res) => {

  const userIds = req.query.user_ids || [];

  const sqlOptions = {
    attributes: ['id', 'name', 'email', 'role', 'profession', 'avatar']
  }

  if (userIds.length) {
    sqlOptions.where = {
      id: userIds
    }
  }

  const users = await User.findAll(sqlOptions);

  return res.json({
    status: 'success',
    data: users
  });
}
```

Tambahkan pada indeks.js

```
const getUsers = require('./getUsers')
```

baris ke 5 dan baris ke 12

```
getUsers
```

lalu tambahkan pada baris ke 10 users.js

```
router.get('/:id', usersHandler.getUsers);
```

berikut langkah pada postman kita hanya menampilkan 2 buah ide yang di filter saja

The screenshot shows a REST client interface with the following details:

- URL:** localhost:5000/users/
- Method:** GET
- Query Params:** user_ids[]=1, user_ids[]=2
- Status:** 200 OK
- Response Body (JSON):**

```
{
  "status": "success",
  "data": [
    {
      "id": 1,
      "name": "widada",
      "email": "juarakoding@gmail.com",
      "role": "admin",
      "profession": "Admin Micro",
      "avatar": null
    },
    {
      "id": 2,
      "name": "Yein",
      "email": "yein@mail.com"
    }
  ]
}
```

Membuat API Save Refresh Token

Langkah pertamanya adalah buat RefreshToken.js didalam handler

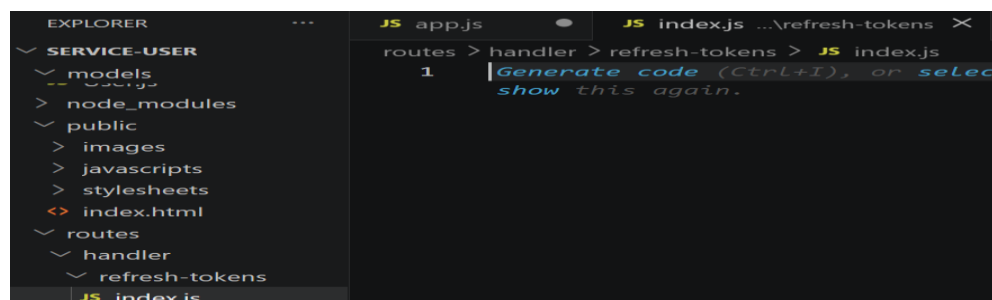
Dan tambahkan dalam app.js baris ke 9

```
const refreshTokensRouter = require('./routes/refreshTokens');
```

juga pada baris ke 21

```
app.use('/refresh_tokens', refreshTokensRouter);
```

buat sebuah folder baru didalam handler, yaitu folder refresh-tokens dan didalamnya ada file indeks.js dan create.js



Berikut kodingan create.js

```
const {
  User,
  RefreshToken
} = require('../models');
const Validator = require('fastest-validator');
const v = new Validator();

module.exports = async (req, res) => {
  const userId = req.body.user_id;
  const refreshToken = req.body.refresh_token;

  const schema = {
    refresh_token: 'string',
    user_id: 'number'
  }

  const validate = v.validate(req.body, schema);
  if (validate.length) {
    return res.status(400).json({
      status: 'error',
      message: validate
    });
  }

  const user = await User.findByPk(userId);
  if (!user) {
    return res.status(404).json({
      status: 'error',
      message: 'user not found'
    });
  }

  // Bagian perbaikan dan lanjutan baru
  const createdRefreshToken = await RefreshToken.create({
    token: refreshToken,
    user_id: userId
  });

  return res.json({
    status: 'success',
    data: {
      id: createdRefreshToken.id
    }
  });
}
```

Lalu kodingan pada indeks.js

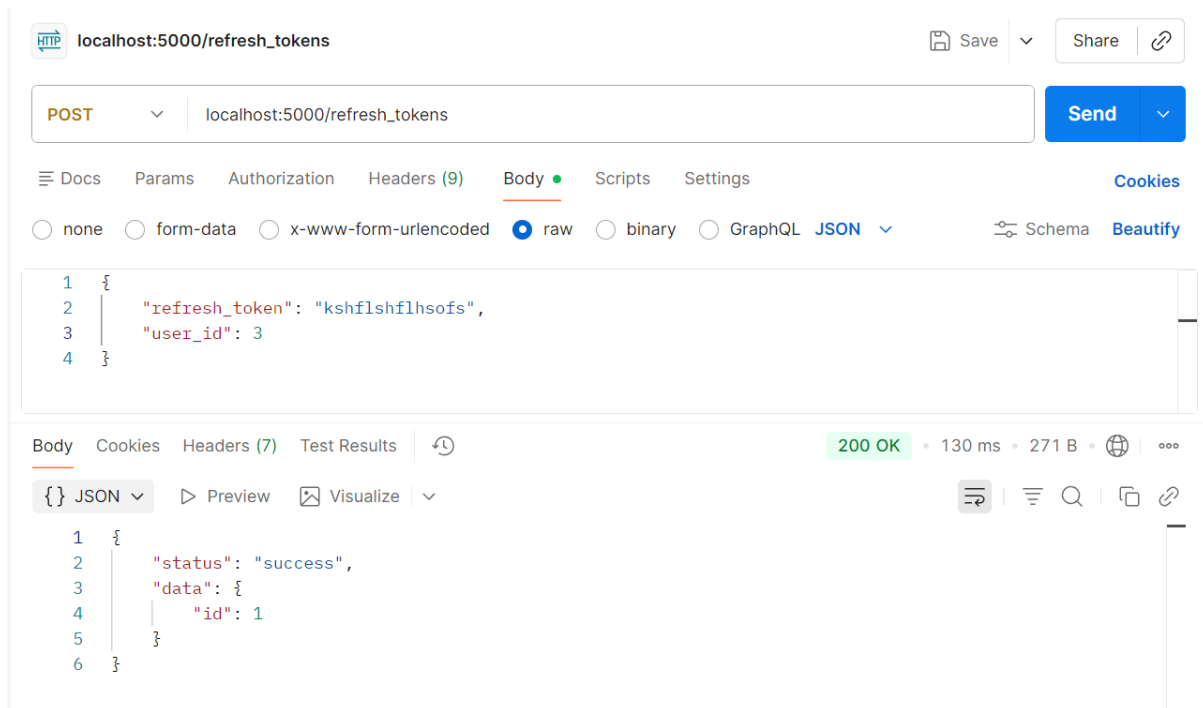
```
const create = require('./create');
```

```
module.exports = {  
  create  
}
```

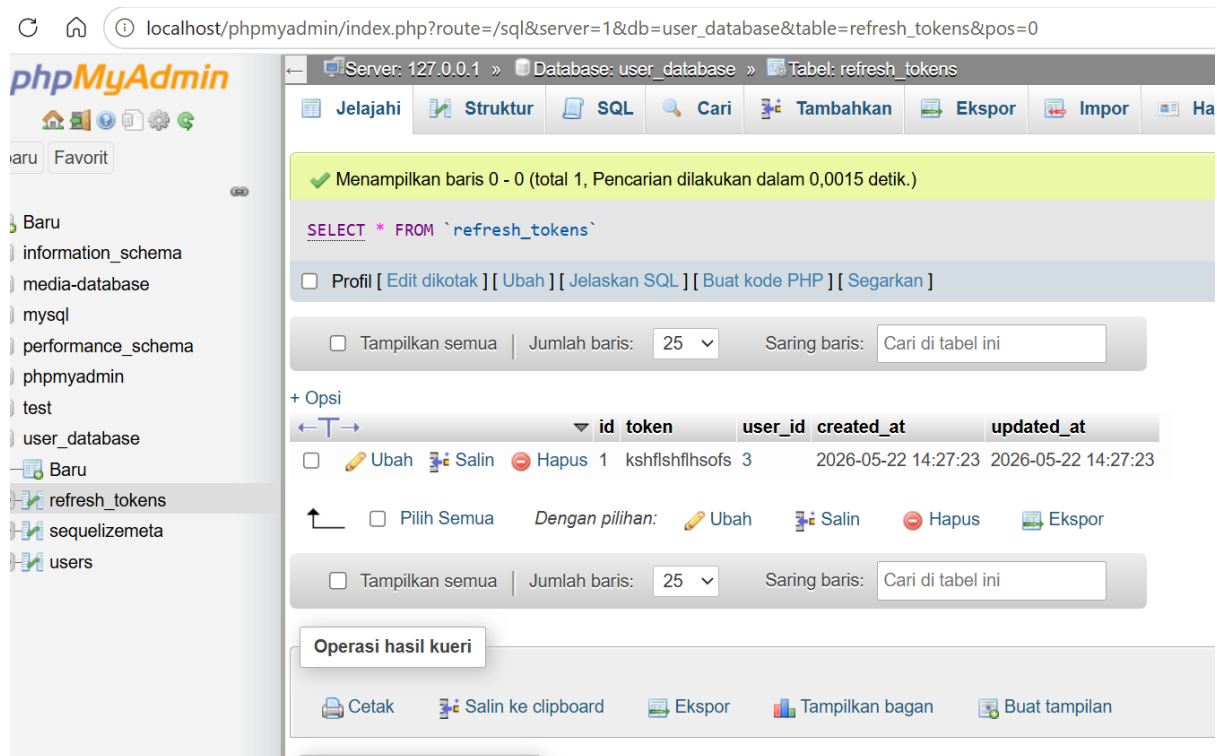
Dan berikut kodingan pada refreshTokens.js

```
const express = require('express');  
const router = express.Router();  
  
const refreshTokensHandler = require('./handler/refresh-tokens');  
  
router.post('/', refreshTokensHandler.create);  
  
module.exports = router;
```

lalu kita cek di postman dg method post

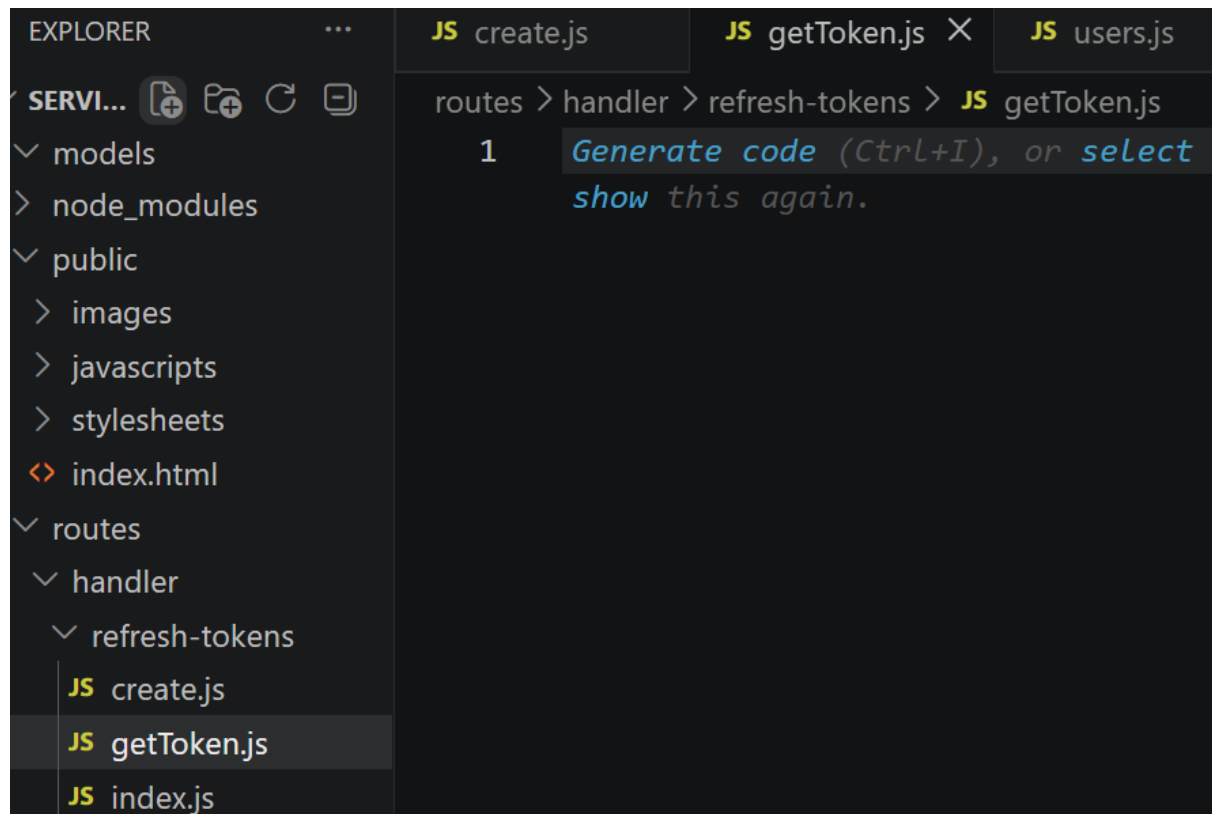


Lalu di cek di php my admin apakah berhasil



Membuat API Get Refresh Token

Langkah pertamanya adalah buat getToken.js dalam handler/refreshToken



Dan berikut adalah isi kodingannya

```
const { RefreshToken } = require('../models');

module.exports = async (req, res) => {
  const refreshToken = req.query.refresh_token;
  const token = await RefreshToken.findOne({
    where: { token: refreshToken }
  });

  if (!token) {
    return res.status(400).json({
      status: 'error',
      message: 'invalid token'
    });
  }

  return res.json({
    status: 'success',
    token
  });
};
```

Lalu tambahkan pada baris ke 2 indeks.js

```
const getToken = require('./getToken');
```

lalu panggil di root dengan menambahkan kodingan pada refreshTokens.js pada baris ke 7

```
router.get('/', refreshTokensHandler.getToken);
```

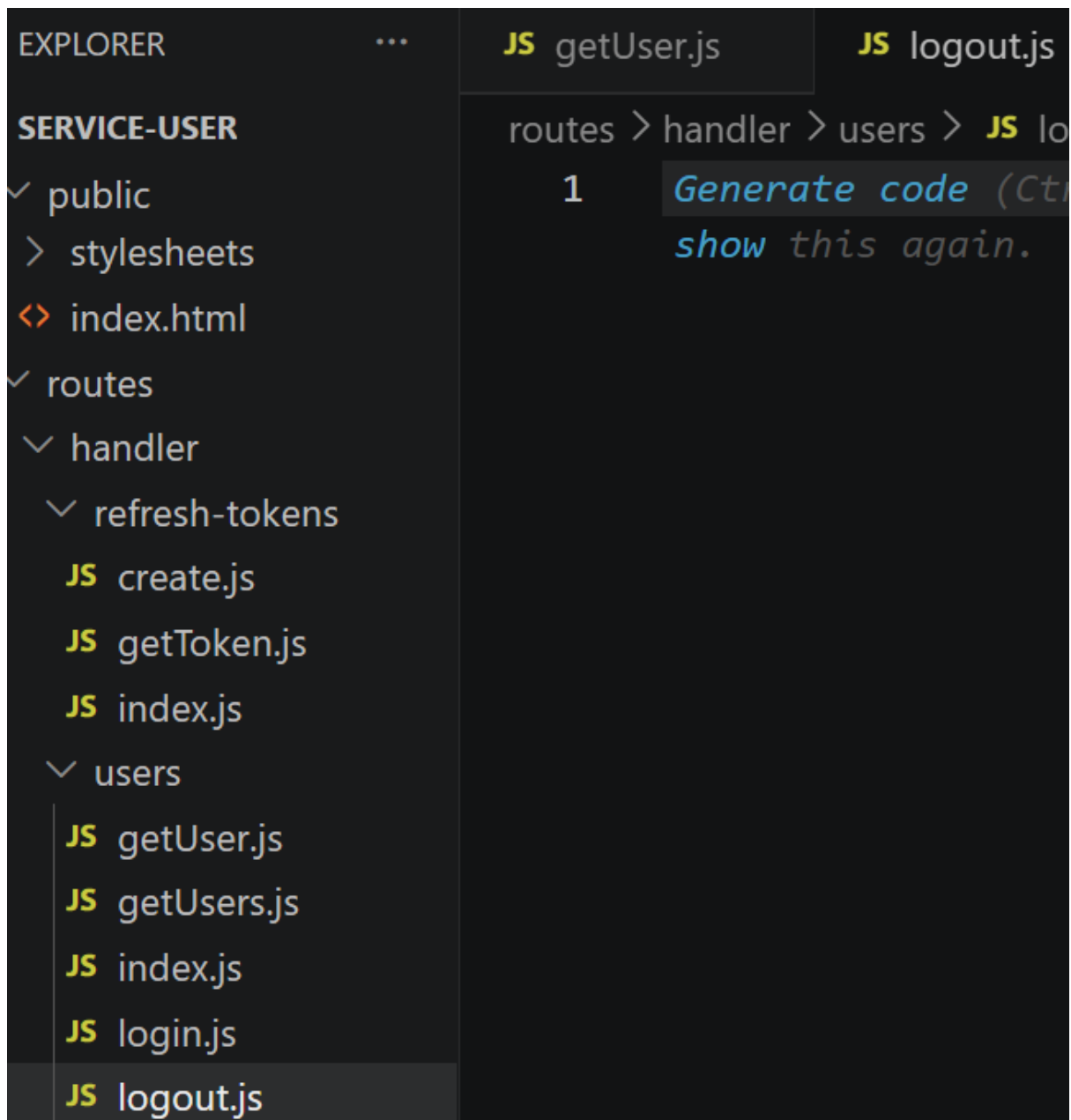
lalu dicobakan di postman dengan method get

The screenshot shows a Postman interface for a GET request to `localhost:5000/refresh_tokens?refresh_token=kshflshflhsofs`. The request is successful, returning a `200 OK` status with a response time of `103 ms` and a body size of `388 B`. The response body is a JSON object:

```
{
  "status": "success",
  "token": {
    "id": 1,
    "token": "kshflshflhsofs",
    "user_id": 3,
    "createdAt": "2026-05-22T14:27:23.000Z",
    "updatedAt": "2026-05-22T14:27:23.000Z"
  }
}
```

Membuat API Logout

Langkah pertamanya adalah buat logout.js dalam handler/users



Berikut kodingannya

```
const {
  User,
  RefreshToken
} = require('../models');

module.exports = async (req, res) => {
  const userId = req.body.user_id;
  const user = await User.findByPk(userId);

  if (!user) {
```

```
return res.status(404).json({
  status: 'error',
  message: 'user not found'
});
}

await RefreshToken.destroy({
  where: { user_id: userId }
});

return res.json({
  status: 'success',
  message: 'refresh token deleted'
});
};
```

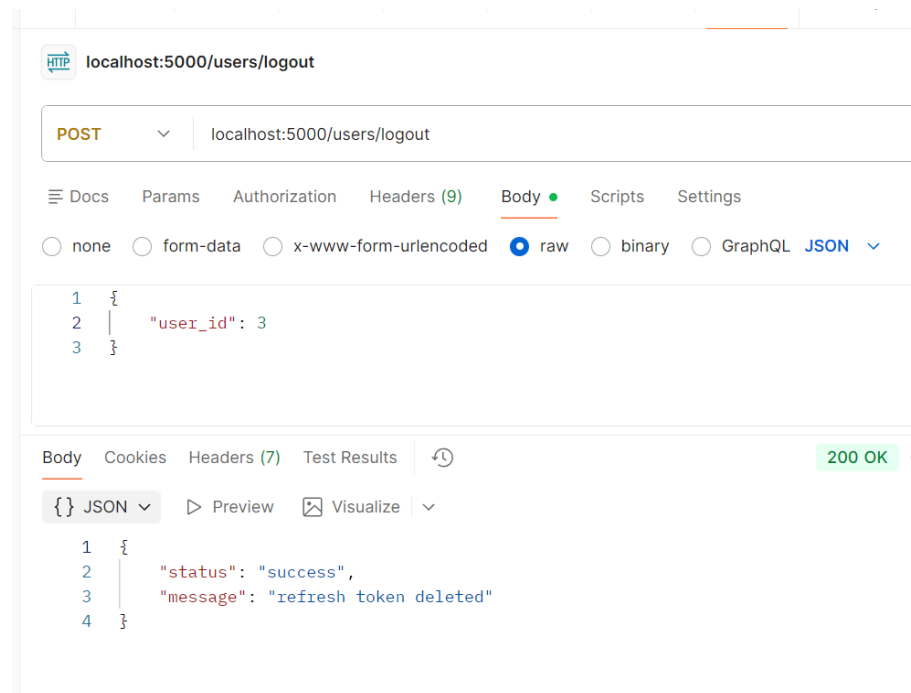
Lalu daftarkan pada indeks.js baris ke 6

```
const logout = require('./logout')
```

lalu di dalam users.js tambahkan kodingan berikut baris ke 8

```
router.post('/logout', usersHandler.logout);
```

cek di postman



Dan cek juga di php my admin

Integrasi Servis User (Pengenalan JSON WEB TOKEN)

Pada praktikum ini mahasiswa akan mempelajari:

1. Mengetahui konsep JSON Web Token (JWT).
2. Menginstal library JWT pada Node.js.
3. Membuat token JWT.
4. Memverifikasi token JWT.
5. Memahami proses autentikasi menggunakan JWT.
6. Menjalankan program JWT menggunakan Node.js.

A. Pengenalan JSON Web Token (JWT)

JSON Web Token (JWT) adalah sebuah standar terbuka **RFC 7519** yang digunakan untuk mengirimkan informasi secara aman antara client dan server dalam bentuk objek JSON. JWT biasanya digunakan pada:

- Login Authentication
- Authorization
- API Security
- Single Sign-On (SSO)

Ketika pengguna berhasil login, server akan membuat sebuah token JWT. Token tersebut kemudian dikirim ke client dan digunakan pada setiap permintaan ke server sebagai bukti bahwa pengguna telah terautentikasi. Library JWT yang digunakan pada Node.js adalah: <https://www.npmjs.com/package/jsonwebtoken>

B. Membuat Project Baru

Langkah 1 – Membuat Direktori Project

Buka Command Prompt atau Terminal kemudian ketik: `mkdir test-jwt`

```
C:\micro>mkdir test-jwt
```

```
C:\micro>dir
```

```
Volume in drive C has no label.
```

```
Volume Serial Number is 7237-F6B3
```

```
Directory of C:\micro
```

```
04/06/2026  19.22    <DIR>          .
06/03/2026  09.10    <DIR>          api-gateway
12/03/2026  20.17    <DIR>          service-media
17/04/2026  09.33    <DIR>          service-user
04/06/2026  19.22    <DIR>          test-jwt
               0 File(s)                0 bytes
               5 Dir(s)  401.294.151.680 bytes free
```

Langkah 2 – Masuk ke Folder Project

Masuk ke folder yang telah dibuat: `cd test-jwt`

C. Inisialisasi Project Node.js

Langkah 3 – Membuat Package Node.js

Jalankan perintah berikut: `npm init -y`

Perintah tersebut akan membuat file: `package.json` secara otomatis.

Contoh hasil:

```
{  
  "name": "test-jwt",  
  "version": "1.0.0"  
}
```

File `package.json` berfungsi untuk menyimpan informasi project dan daftar library yang digunakan.

```
C:\micro\test-jwt>npm init -y  
Wrote to C:\micro\test-jwt\package.json:  
  
{  
  "name": "test-jwt",  
  "version": "1.0.0",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "description": ""  
}
```

D. Menginstal Library JWT

Langkah 4 – Instalasi Jsonwebtoken

Ketik perintah berikut: `npm install jsonwebtoken --save`

Setelah berhasil, akan muncul folder: `node_modules`

dan file: `package-lock.json` Library JWT sekarang sudah siap digunakan.

```
C:\micro\test-jwt>npm install jsonwebtoken --save  
  
added 15 packages, and audited 16 packages in 3s  
  
1 package is looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
  
C:\micro\test-jwt>
```

E. Membuka Project di Visual Studio Code

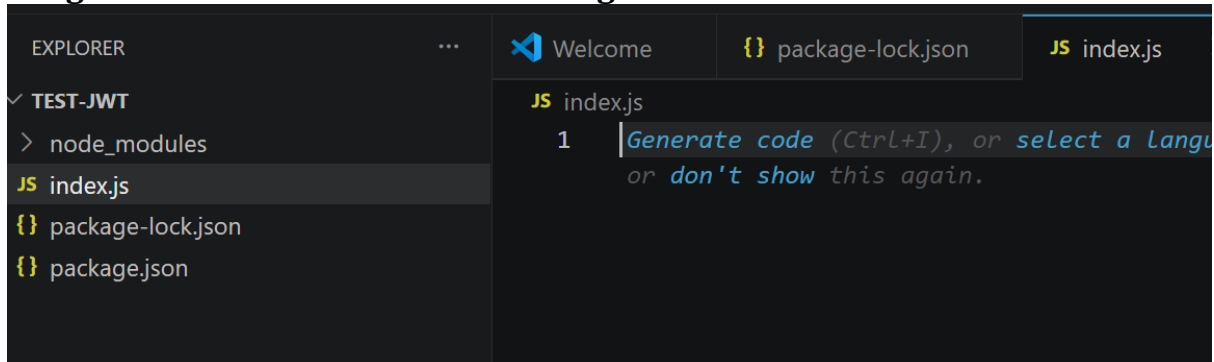
Langkah 5 – Membuka Folder Project

Buka Visual Studio Code. Pilih: File → Open Folder

Kemudian pilih folder: test-jwt

G. Menulis Program JWT

Langkah 7 – Menambahkan Kode Program



Masukkan kode berikut ke dalam file index.js (Cara synchronous)

```
const jwt = require('jsonwebtoken');
const JWT_SECRET = 'bwamicro!23';
// create basic token dengan proses synchronous
const token = jwt.sign(
  { data: { kelas: 'bwamicro' } },
  JWT_SECRET,
  { expiresIn: '5m' }
);
console.log(token);
const token1 = 'ISI_TOKEN_DISINI';
// verifikasi token
try {
  const decoded = jwt.verify(token1, JWT_SECRET);
  console.log(decoded);
} catch (error) {
  console.log(error.message);
}
```

Isi dengan kodingan berikut (Cara asynchronous di koment)

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

const JWT_SECRET = 'bwamicro!23';

// create basic token dengan proses synchronous
const token = jwt.sign(
  { data: { kelas: 'bwamicro' } },
  JWT_SECRET,
  { expiresIn: '5m' }
);

console.log(token);

const token1 =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7ImtleGFzIjoibWamicro!23ifSwiaWF0IjoxNzgwNTc2OTExLCJleHAiOjE3MDA1NzcyMTF9.BRrfb3Qgo1imwi40s5fYZPRT0RUE7uWSjmgBT0Ej4mc';
```

```
// cara 2
try {
  const decoded = jwt.verify(token1, JWT_SECRET);
  console.log(decoded);
} catch (error) {
  console.log(error.message);
}

// asynchronous - create token
// jwt.sign({ data: { kelas: 'bwamicro' } }, JWT_SECRET, { expiresIn: '1m' },
(err, token) => {
  //   console.log(token);
  // });

// cara 1
// jwt.verify(token1, JWT_SECRET, (err, decoded) => {
//   if (err) {
//     console.log(err.message);
//     return;
//   }
//   console.log(decoded);
// });
```

I. Menjalankan Program

Langkah 8 – Menjalankan Program Pertama

Buka Terminal kemudian jalankan:

```
node index.js
```

Contoh hasil:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Salin token tersebut.

Langkah 9 – Tempelkan Token

Ganti bagian:

```
const token1 = 'ISI_TOKEN_DISINI';
```

menjadi:

```
const token1 = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...';
```

Langkah 10 – Jalankan Kembali Program

```
node index.js
```

Jika berhasil maka akan tampil:

```
{
  data: { kelas: 'bwamicro' },
  iat: 1780576911,
  exp: 1780577211
}
```

Jika langkah nya berhasil akan terlihat seperti pada gambar dibawah :

```
C:\micro\test-jwt>node index.js
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7ImtleGFzIjoiYndhbWljcm8ifSwiaWF0IjoxNzgwNTc2OTExLCJleHAiOjE3ODU1NzcyMTF9.BRr-fb3Qgo1i
mwi40s5fYZPRT0RUE7uWSjmgBT0Ej4mc
jwt malformed
C:\micro\test-jwt>
```

```
C:\micro\test-jwt>node index.js
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7ImtlbGFzIjoiaWdhbWljcm8ifSwiaWF0IjoxNzgwNTc3MDA4LCJleHAiOjE3ODAxNzczMDh9.gk7ou_6zqJQ
ST_vp2Syqy9wi5se2hx_dqsX9yZC0M0c
{ data: { kelas: 'bwamicro' }, iat: 1788576911, exp: 1788577211 }
C:\micro\test-jwt>
```

Kesimpulan

Pada praktikum ini mahasiswa telah mempelajari:

1. Konsep dasar JSON Web Token (JWT).
2. Menginstal library `jsonwebtoken`.
3. Membuat token menggunakan `jwt.sign()`.
4. Memverifikasi token menggunakan `jwt.verify()`.
5. Menjalankan program JWT dengan Node.js.
6. Menggunakan metode synchronous dan asynchronous pada JWT.

JWT merupakan komponen penting dalam pengembangan aplikasi Mikroservis karena digunakan sebagai mekanisme autentikasi dan otorisasi antar layanan (service) sehingga komunikasi antar API menjadi lebih aman.

Integrasi Service User dengan API Gateway Bagian API Register

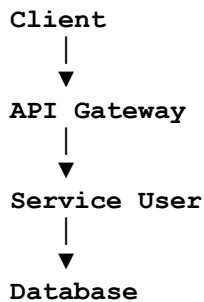
Tujuan Praktikum

Pada praktikum ini mahasiswa akan mempelajari:

1. Menghubungkan Service User dengan API Gateway.
2. Membuat handler Register pada API Gateway.
3. Mengirim request dari API Gateway ke Service User.
4. Menggunakan Axios Adapter sebagai penghubung antar service.
5. Menguji API Register melalui API Gateway.
6. Memahami konsep komunikasi antar Microservice.

A. Konsep Integrasi API Gateway

Pada arsitektur Mikroservis, client tidak langsung mengakses Service User. Alur komunikasi yang digunakan adalah:



Keuntungan menggunakan API Gateway:

- Semua request masuk melalui satu pintu.
- Meningkatkan keamanan sistem.
- Mempermudah manajemen API.
- Mempermudah autentikasi menggunakan JWT.
- Mempermudah monitoring dan logging.

Pada praktikum ini API Gateway akan meneruskan request Register ke Service User.

B. Membuka Project API Gateway

Langkah 1 – Membuka Folder API Gateway

Buka Visual Studio Code. Pilih folder: api-gateway

Pastikan project API Gateway yang telah dibuat sebelumnya sudah dapat dijalankan.

C. Membuat Handler User

Langkah 2 – Membuat Folder users didalam Handler

Masuk ke folder: `routes/handler`

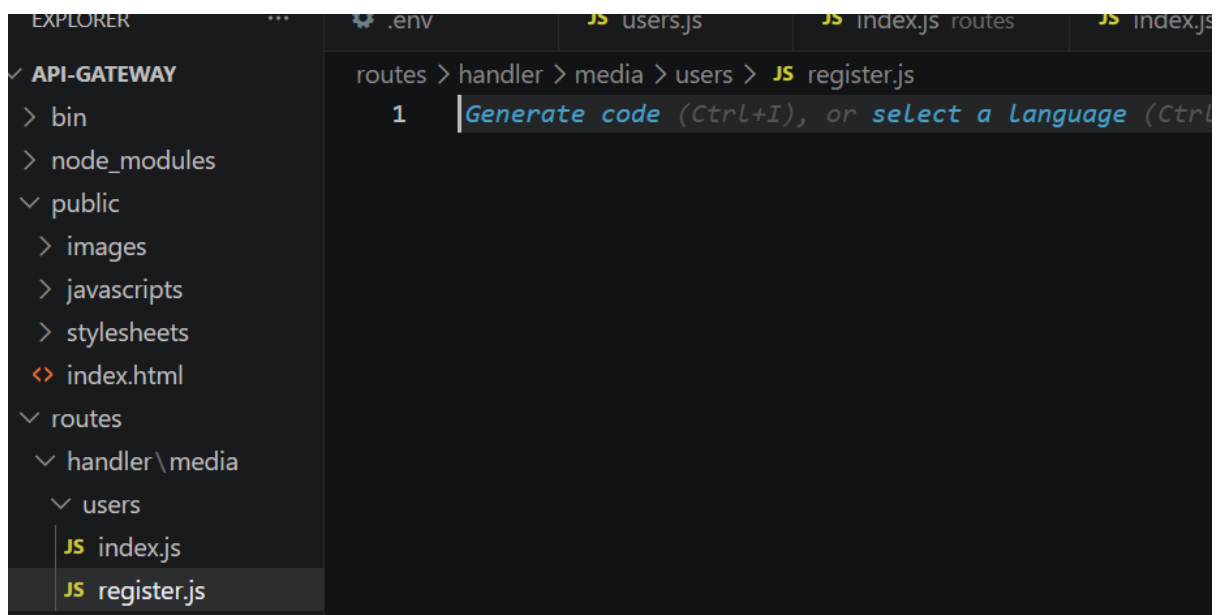
Kemudian buat struktur folder berikut:

```
routes
├── handler
│   └── users
│       ├── index.js
│       └── register.js
```

Struktur lengkap:

```
api-gateway
├── routes
│   ├── handler
│   │   ├── users
│   │   │   ├── index.js
│   │   │   └── register.js
│   └── users.js
└── .env
```

Buat sebuah handler baru yang bernama users didalam nya buat file indeks.js serta register.js



D. Membuat Handler Register

Langkah 3 – Membuat File register.js

Buka file: `routes/handler/users/register.js`

Kemudian isi dengan kode berikut:

```
const apiAdapter = require('../../apiAdapter');
const {
  URL_SERVICE_USER
} = process.env;

const api = apiAdapter(URL_SERVICE_USER);

module.exports = async (req, res) => {
  try {
    const user = await api.post('/users/register', req.body);
    return res.json(user.data);
  } catch (error) {

    if (error.code === 'ECONNREFUSED') {
      return res.status(500).json({ status: 'error', message: 'service
unavailable' });
    }

    const { status, data } = error.response;
    return res.status(status).json(data);
  }
}
```

F. Membuat File index.js

Langkah 4 – Membuat File index.js

Buka file: `routes/handler/users/index.js`

Isi dengan kode berikut:

```
const register = require('./register');

module.exports = {
  register
};
```

Penjelasan: File ini berfungsi sebagai pusat pengumpulan seluruh handler User.

G. Mengatur Environment Variable

Langkah 5 – Membuka File .env

Tambahkan kode berikut pada baris terakhir:

```
URL_SERVICE_USER=http://localhost:5000
```

H. Menghubungkan Route User

Langkah 6 – Membuka File users.js

Buka file: `routes/users.js`

Lalu ganti kodingan pada users.js

```
const express = require('express');
const router = express.Router();

const usersHandler = require('../handler/users');

router.post('/register', usersHandler.register);

module.exports = router;
```

J. Menjalankan Service

Langkah 7 – Membuka Dua Terminal

Buka dua terminal secara bersamaan.

Terminal 1

Masuk ke folder: `cd service-user`

Jalankan: `npm start` Pastikan muncul: `Server running on port 5000`

Terminal 2

Masuk ke folder: `cd api-gateway`

Jalankan: `npm start` Pastikan muncul: `Server running on port 3000`

K. Pengujian Menggunakan Postman

Langkah 8 – Membuat Request Baru

Method: POST URL: <http://localhost:3000/register>

Langkah 9 – Pilih Body, Raw, JSON

Masukkan data berikut: (boleh diisi nama sendiri)

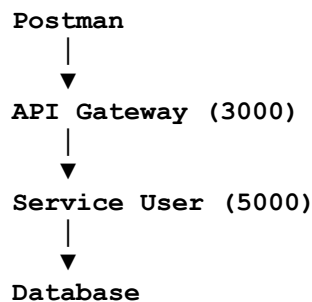
```
{
  "name": "Putra Manda",
  "email": "putramanda@pnp.ac.id",
  "password": "123456"
}
```

Langkah 10 – Klik Send

Jika berhasil maka API Gateway akan:

1. Menerima request dari Postman.
2. Mengirim request ke Service User.
3. Service User menyimpan data ke database.
4. Service User mengirim response.
5. API Gateway meneruskan response ke Postman.

Alurnya:



Kesimpulan

Pada praktikum ini mahasiswa telah berhasil:

1. Membuat handler Register pada API Gateway.
2. Menghubungkan API Gateway dengan Service User.
3. Menggunakan Api Adapter sebagai penghubung antar service.
4. Menambahkan konfigurasi URL Service User pada file `.env`.
5. Menguji API Register melalui Postman.
6. Memahami alur komunikasi antara Client → API Gateway → Service User pada arsitektur Mikroservis.

Pada praktikum berikutnya, API Gateway akan diintegrasikan dengan **API Login** sehingga proses autentikasi pengguna dapat dilakukan menggunakan **JSON Web Token (JWT)**.

Integrasi Service User dengan API Gateway Bagian API Login

Tujuan Praktikum

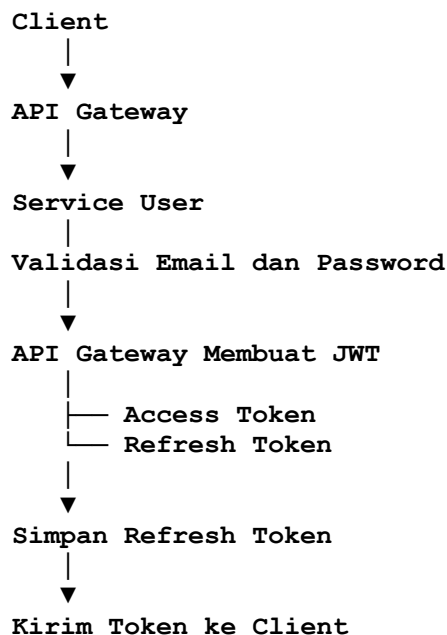
Pada praktikum ini mahasiswa akan mempelajari:

1. Mengintegrasikan API Login pada Service User dengan API Gateway.
2. Menggunakan library JSON Web Token (JWT) untuk autentikasi.
3. Membuat Access Token dan Refresh Token.
4. Menyimpan Refresh Token ke Service User.
5. Mengatur konfigurasi JWT menggunakan Environment Variable.
6. Menguji API Login menggunakan Postman.
7. Memahami alur autentikasi pada arsitektur Mikroservis

A. Konsep Login Menggunakan JWT

Pada aplikasi Mikroservis, proses login biasanya tidak langsung mengembalikan data pengguna saja, tetapi juga menghasilkan token autentikasi.

Alur proses login:



B. Menambahkan Konfigurasi JWT

Langkah 1 – Membuka File .env

Buka file: `.env`

Tambahkan kode berikut pada bagian paling bawah:

```
JWT_SECRET=juarakodingbwa123
JWT_SECRET_REFRESH_TOKEN=refreshtoken!23
JWT_ACCESS_TOKEN_EXPIRED=5m
JWT_REFRESH_TOKEN_EXPIRED=1d
```

Penjelasan:

Variabel	Fungsi
JWT_SECRET	Secret Key untuk Access Token
JWT_SECRET_REFRESH_TOKEN	Secret Key untuk Refresh Token
JWT_ACCESS_TOKEN_EXPIRED	Masa berlaku Access Token
JWT_REFRESH_TOKEN_EXPIRED	Masa berlaku Refresh Token

C. Membuat Handler Login

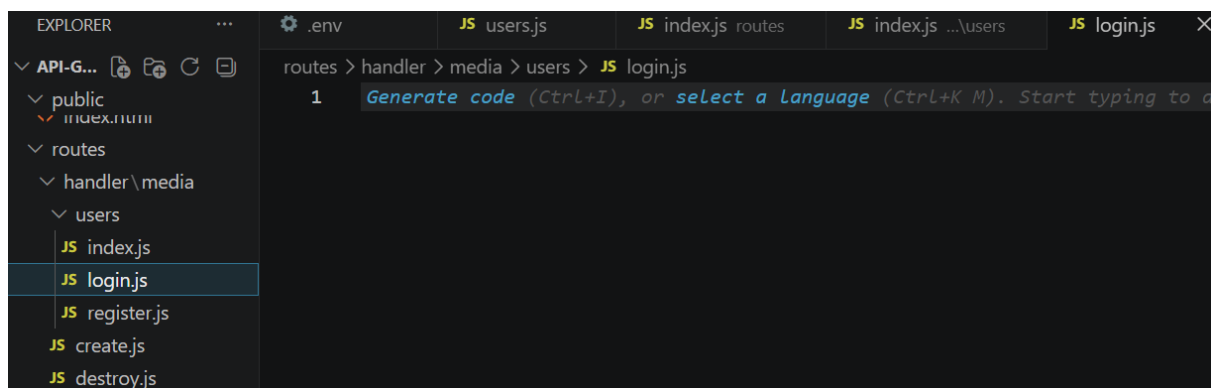
Langkah 2 – Membuat File login.js

Buat file baru:

routes/handler/users/login.js

Struktur folder menjadi:

```
routes
├── handler
│   └── users
│       ├── index.js
│       ├── register.js
│       └── login.js
```



F. Mendaftarkan Handler Login

Langkah 4 – Membuka File index.js

Buka file: routes/handler/users/index.js

Kemudian ubah menjadi:

```
module.exports = {  
  register: require('./register'),  
  login: require('./login')  
}
```

Lalu isikan kodingan berikut pada login.js

```
const apiAdapter = require('../../apiAdapter');  
const jwt = require('jsonwebtoken');  
  
const {  
  URL_SERVICE_USER,  
  JWT_SECRET,  
  JWT_SECRET_REFRESH_TOKEN,  
  JWT_ACCESS_TOKEN_EXPIRED,  
  JWT_REFRESH_TOKEN_EXPIRED  
} = process.env;  
  
const api = apiAdapter(URL_SERVICE_USER);  
  
module.exports = async (req, res) => {  
  try {  
  
    const user = await api.post('/users/login', req.body);  
    const data = user.data.data;  
  
    const token = jwt.sign(  
      { data },  
      JWT_SECRET,  
      { expiresIn: JWT_ACCESS_TOKEN_EXPIRED }  
    );  
  
    const refreshToken = jwt.sign(  
      { data },  
      JWT_SECRET_REFRESH_TOKEN,  
      { expiresIn: JWT_REFRESH_TOKEN_EXPIRED }  
    );  
  
    await api.post('/refresh_tokens', {  
      refresh_token: refreshToken, user_id: data.id  
    });  
  }  
}
```

```
        return res.json({
          status: 'success',
          data: {
            token,
            refreshToken
          }
        });
      } catch (error) {

        if (error.code === 'ECONNREFUSED') {
          return res.status(500).json({
            status: 'error',
            message: 'service unavailable'
          });
        }

        const { status, data } = error.response;

        return res.status(status).json(data);
      }
    });
  });
};
```

G. Menambahkan Route Login

Langkah 5 – Membuka File users.js

Buka file: routes/users.js

Ubah menjadi:

```
const express = require('express');
const router = express.Router();

const usersHandler = require('./handler/users');

router.post('/register', usersHandler.register);
router.post('/login', usersHandler.login);

module.exports = router;
```

H. Menjalankan Service

Langkah 6 – Menjalankan Service User

Buka Terminal 1:

```
cd service-user  
nodemon bin/www
```

Pastikan muncul: `Server running on port 5000`

Langkah 7 – Menjalankan API Gateway

Buka Terminal 2:

```
cd api-gateway  
nodemon bin/www
```

Pastikan muncul: `Server running on port 3000`

I. Pengujian Menggunakan Postman

Langkah 8 – Membuat Request Login

Method: `POST` URL: `http://localhost:3000/login`

Langkah 9 – Memasukkan Data Login

Pilih:

Body
↓
raw
↓
JSON

Masukkan data:

```
{  
  "email": "putra@gmail.com",  
  "password": "123456"  
}
```

Langkah 10 – Klik Send

Jika login berhasil maka akan muncul:

```
{  
  "status": "success",  
  "data": {  
    "token": "eyJhbGciOiJIUzI1NiIs...",  
    "refreshToken": "eyJhbGciOiJIUzI1NiIs..."  
  }  
}
```

Kesimpulan

Pada praktikum ini mahasiswa telah berhasil:

1. Mengintegrasikan API Login antara Service User dan API Gateway.
2. Membuat Access Token menggunakan JWT.
3. Membuat Refresh Token menggunakan JWT.
4. Menyimpan Refresh Token ke database melalui Service User.
5. Menggunakan Environment Variable untuk konfigurasi JWT.
6. Menguji proses login menggunakan Postman.
7. Memahami konsep autentikasi pada arsitektur Mikroservis menggunakan JSON Web Token (JWT).

Pada praktikum berikutnya akan dipelajari **API Refresh Token**, yaitu proses menghasilkan Access Token baru menggunakan Refresh Token yang telah disimpan sebelumnya.