RESTful API dengan NodeJs, MySQL, dan Docker

A. RESTful API dengan NodeJs, MySQL, dan Docker

Pada project kali ini kita akan membuat RESTful API sederhana untuk data user menggunakan NodeJs dan Mysql.

Apa itu REST?

REST (*Representational State Transfer*) adalah suatu arsitektur metode komunikasi yang menggunakan protokol HTTP untuk pertukaran data dan metode ini sering diterapkan dalam pengembangan aplikasi. Dimana tujuannya adalah untuk menjadikan sistem yang memiliki performa yang baik, cepat dan mudah untuk di kembangkan (*scale*) terutama dalam pertukaran dan komunikasi data.

Apa itu API?

API adalah singkatan dari *Application Programming Interface*, dan memungkinkan developer untuk mengintegrasikan dua bagian dari aplikasi atau dengan aplikasi yang berbeda secara bersamaan.

Apa itu Node.js?

Node.js adalah salah satu platform yang digunakan dalam pemrograman aplikasi web berbasis Javascript yang dikenalkan pada tahun 2009. Untuk menginstall Node.js anda bisa langsung unggah dari situs web nya https://nodejs.org/en/download/ atau dengan menggunakan NPM (Node Package Manager)- NPM adalah aplikasi untuk mengambangkan dan membagikan kode Javascript.

Apa itu Docker?

Docker adalah platform perangkat lunak yang memungkinkan Anda membuat, menguji, dan menerapkan aplikasi dengan cepat. Docker mengemas perangkat lunak ke dalam unit standar yang disebut kontainer yang memiliki semua yang diperlukan perangkat lunak agar dapat berfungsi termasuk pustaka, alat sistem, kode, dan waktu proses. Dengan menggunakan Docker, Anda dapat dengan cepat menerapkan dan menskalakan aplikasi ke lingkungan apa pun dan yakin bahwa kode Anda akan berjalan.

B. Dokumentasi API

1. User - Get all user data

Method : **GET**

URL : **/user**

Response OK :

```
{
    "status" : true,
    "message" : "Success",
    "data" : []
}
```

Response ERROR :

```
{
    "status" : false,
    "message" : "Error Message"
}
```

2. User - Create new user

Method : **POST**

URL : **/user**

Request :

```
{
    "name" : "Ade Putra Prima Suhendri",
    "email" : "admin@kasehat.co.id",
    "password" : "password",
    "confirm_password" : "password"
}
```

Response OK :

```
{
    "status" : true,
    "message" : "Created",
    "data" : 1
}
```

Response ERROR :

```
{
    "status" : false,
    "message" : "Error Message"
}
```

3. User - Update user

Method : **PUT**
URL : **/user**
Request :

```
{
    "id" : 1,
    "name" : "Ade Putra Prima Suhendri",
    "email" : "admin@kasehat.co.id"
}
```

Response OK :

```
{
    "status" : true,
    "message" : "Updated"
}
```

Response ERROR :

```
{
    "status" : false,
    "message" : "Error Message"
}
```

4. User - Delete user

Method : **DELETE**
URL : **/user**
Request :

```
{
    "id" : 1
}
```

Response OK :

```
{
    "status" : true,
    "message" : "Deleted"
}
```

Response ERROR :

```
{
    "status" : false,
    "message" : "Error Message"
}
```

C. Struktur File Project

Berikut adalah struktur file project yang akan kita buat :

app-root/

```
│
├── app/
│     ├── node_modules/
│     ├── package.json
│     ├── routes/
│     │     └── route.js
│     ├── .env
│     ├── db.js
│     └── server.js
├── docker-compose.yml
└── Dockerfile
```

package.json

File *package.json* adalah file yang berisi deskripsi project javascript kita, secara sederhana *package.json* adalah prosedur yang akan dijalankan oleh npm / yarn.

```
{
  "name": "node_mysql_docker",
  "version": "1.0.0",
  "description": "Restful API Node.js MySQL run on Docker",
  "author": "Ade Putra Prima Suhendri <admin@kasehat.co.id>",
  "main": "server.js",
  "scripts": {
     "start": "nodemon server.js"
  },
  "dependencies": {
     "bcrypt": "^5.0.0",
     "body-parser": "^1.19.0",
     "cors": "^2.8.5",
     "dotenv": "^8.1.0",
     "express": "^4.16.1",
     "mysql": "^2.18.1",
     "nodemon": "^1.19.2"
  }
}
```

bcrypt - Fungsi hashing kata sandi.

body-parser - Library yang dapat digunakan sebagai middleware untuk membantu mengurai JSON pada body request.

cors - Menyediakan middleware Connect / Express yang dapat digunakan untuk mengaktifkan CORS.

dotenv - Modul untuk memuat variabel dari .env.

express - Framework ini menawarkan beberapa fitur seperti routing, rendering view dan mendukung middleware.

nodemon - Alat yang secara otomatis memulai ulang aplikasi node ketika ada perubahan pada file.

Setelah package.json selesai kita buat, maka langkah selanjutnya adalah kita jalankan perintah berikut pada directory **root-app/app/** :

```
$ npm install
```

db.js

File *db.js* akan digunakan untuk mengatur koneksi ke database mysql, berikut kode nya :

```
"use strict"
var mysql = require('mysql');
var connection;
module.exports = {
      DB: function () {
      connection = mysql.createConnection({
            host: process.env.DB_HOST,
            user: process.env.DB_USER,
            password: process.env.DB_PASS,
            database: process.env.DB_NAME,
            port: process.env.DB_PORT,
      });
      return connection;
      }
};
```

server.js

File *server.js* adalah aplikasi utama yang akan dijalankan pertama kali :

```
'use strict';
require('dotenv').config();
const express = require('express');
const app = express();
const BodyParser = require('body-parser');
const cors = require('cors');
const PORT = process.env.PORT;
const HOST = process.env.HOST;
app.use(cors());
app.use(BodyParser.json());
const Route = require('./routes/Route');
app.get('/', (req, res) => {
     res.send('RESTful API With NodeJs & Mysql on Docker');
  });
app.use('/user', Route);
app.listen(PORT, HOST);
```

route.js

File *route.js* digunakan untuk mengatur routing aplikasi kita :

```
const express = require('express');
const bcrypt = require('bcrypt');
const router = express.Router();
var salt = bcrypt.genSaltSync(10);
var connection = require('../db');
var response;

// GET ALL DATA USER
router.get('/', async (req, res) => {
     try {
     connection.DB().connect();
     connection.DB().query('SELECT id, name, email FROM users
ORDER BY id ASC', function (error, results) {
          if (error) {
               response = {
               status: false,
               message: error.sqlMessage,
               };
          } else {
               response = {
               status: true,
               message:"Success",
               data: results,
               };
          }
          res.json(response)
     });

     } catch (error) {
     response = {
          status: false,
          message:error.message
     };
     res.json(response)
     }

});
```

```
// CREATE USER
router.post('/', async (req, res) => {
      try {
      const name = req.body.name
      const email = req.body.email
      const password = bcrypt.hashSync(req.body.password, salt)
      if(req.body.password == req.body.confirm_password){
            connection.DB().connect()
            connection.DB().query('INSERT INTO users (name, email,
password) VALUES (?, ?, ?)', [name, email, password], (error,
results) => {
                  if (error) {
                        response = {
                        status: false,
                        message: error.sqlMessage,
                        };
                  } else {
                        response = {
                        status: true,
                        message:"Created",
                        data: results.insertId,
                        };
                  }
                  res.json(response)
            })
      }else{
            response = {
                  status: false,
                  message:"Password not match"
            };
            res.json(response)
      }
      } catch (error) {
      response = {
            status: false,
            message:error.message
      };
      res.json(response)
      }
});
```

```
// DELETE USER
router.delete('/', async (req, res) => {
      try {
      const id = req.body.id
      connection.DB().connect()
      connection.DB().query('DELETE FROM users WHERE id = ?', [id],
(error, results) => {
            if (error) {
                  response = {
                  status: false,
                  message: error.sqlMessage,
                  };
            } else {
                  response = {
                  status: true,
                  message:"Deleted",
                  };
            }
            res.json(response)
      })
      } catch (error) {
      response = {
            status: false,
            message:error.message
      };
      res.json(response)
      }
});
```

```
// UPDATE USER
router.put('/', async (req, res) => {
      try {
      const id = req.body.id
      const name = req.body.name
      const email = req.body.email
      connection.DB().connect()
      connection.DB().query(
            'UPDATE users SET name = ?, email = ? WHERE id = ?',
            [name, email, id],
            (error, results) => {
                  if (error) {
                        response = {
                        status: false,
                        message: error.sqlMessage,
                  };
                  } else {
                        response = {
                        status: true,
                        message:"Updated"
                  };
                  }
                  res.json(response)
            }
      )
      } catch (error) {
      response = {
            status: false,
            message:error.message
      };
      res.json(response)
      }

});

module.exports = router;
```

.env

file env atau Environment Variable merupakan variabel dinamis pada komputer yang dapat diakses oleh sebuah program.

```
HOST=127.0.0.1
PORT=3000

DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASS=MYSQL_PASSWORD
DB_NAME=MYSQL_DATABASE_NAME
```

Pengaturan Docker

.env

```
HOST=0.0.0.0
PORT=3000

DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASS=MYSQL_PASSWORD
DB_NAME=MYSQL_DATABASE_NAME
```

Dockerfile

```
FROM node:12
WORKDIR /app
COPY ./app/package.json /app
COPY ./app /app
RUN npm install
CMD ["npm", "start"]
EXPOSE 3000
```

docker-compose.yml

```
version: "3"
services:
  node:
      container_name: NODE_SERVER
      restart: always
      build: .
      volumes:
      - ./app:/app
      ports:
      - "80:3000"
      links:
      - database
      environment:
      DB_PORT: 3306
      DB_HOST: database
  database:
      container_name: DB_MYSQL
      image: mariadb
      environment:
      MYSQL_ROOT_PASSWORD: MYSQL_PASSWORD
      MYSQL_DATABASE: MYSQL_DATABASE_NAME
      ports:
      - "3306"
```

Menjalankan Docker

Ketikkan perintah berikut untuk mulai menjalankan docker pada server:

```
$ docker-compose up --build
```