HOMEWORK - EXPRESS JS RESTFUL API & MIDDLEWARE

full code : https://github.com/saefulmuminin/express-api-with-swagger.git

Soal 1: Buatlah RESTful API yang terdiri dari GET, POST, DELETE, dan PUT. Setelah itu buatlah endpoint untuk register user dan login user untuk implementasi authorization dan authentication. Pastikan yang hanya bisa mengakses API hanyalah user yang terdaftar.

a. **Get**
   http://localhost:3000/users

   http://localhost:3000/movies

```
router.get("/users", (req, res) => {
  // Mengambil data pengguna dari basis data
  pool.query("SELECT * FROM users", (error, result) => {
    if (error) {
      return res
        .status(500)
        .json({ message: "Gagal mengambil data pengguna." });
    }
    return res.status(200).json(result.rows);
  });
});

// GET data movies
router.get("/movie", (req, res) => {
  // Mengambil data pengguna dari basis data
  pool.query("SELECT * FROM movie", (error, result) => {
    if (error) {
      return res
        .status(500)
        .json({ message: "Gagal mengambil data pengguna." });
    }
    return res.status(200).json(result.rows);
  });
});
```

b. **Post**
   http://localhost:3000/login
   http://localhost:3000/register

```javascript
router.post("/login", (req, res) => {
  const { email, password } = req.body;

  if (!email || !password) {
    return res.status(400).json({ message: "Harap isi semua field." });
  }

  // Lakukan otentikasi (perhatikan: ini hanya contoh)
  pool.query(
    "SELECT * FROM users WHERE email = $1 AND password = $2",
    [email, password],
    (error, result) => {
      if (error) {
        return res.status(500).json({ message: "Gagal melakukan otentikasi." });
      }
      if (result.rows.length === 0) {
        return res.status(401).json({
          message: "Gagal melakukan otentikasi. Periksa email dan password.",
        });
      }

      // Buat token JWT
      const token = jwt.sign({ email }, process.env.JWT_SECRET);
      return res.status(200).json({ token });
    }
  );
});
```

```javascript
router.post("/register", (req, res) => {
  const { email, password, gender, role } = req.body;

  if (!email || !password || !gender || !role) {
    return res.status(400).json({
      message: "Harap isi semua field ini email, password, gender, role.",
    });
  }

  pool.query(
    "INSERT INTO users (email, password, gender, role) VALUES ($1, $2, $3, $4) RETURNING id",
    [email, password, gender, role],
    (error, result) => {
      if (error) {
        return res
          .status(500)
          .json({ message: "Gagal mendaftar.", error: error.message });
      }

      // Periksa apakah ada hasil yang dikembalikan
      if (result.rows.length === 0) {
        return res.status(500).json({
          message: "Gagal mendaftar. Tidak ada ID yang dikembalikan.",
        });
      }

      // Ambil ID dari hasil query
      const insertedUserId = result.rows[0].id;

      return res.status(201).json({
        message: "Registrasi berhasil.",
        userId: insertedUserId,
      });
    }
  );
});
```

c. Delete
http://localhost:3000/**users/:id**
http://localhost:3000/**movie/:id**

```
router.delete("/users/:id", (req, res) => {
  const id = parseInt(req.params.id);

  // Periksa apakah pengguna dengan ID yang diberikan ada dalam database
  pool.query("SELECT * FROM users WHERE id = $1", [id], (error, result) => {
    if (error) {
      return res.status(500).json({ message: "Gagal menghapus pengguna." });
    }

    if (result.rows.length === 0) {
      return res.status(404).json({ message: "Pengguna tidak ditemukan." });
    }

    // Jika pengguna ditemukan, hapus pengguna dengan ID tersebut
    pool.query("DELETE FROM users WHERE id = $1", [id], (error) => {
      if (error) {
        return res.status(500).json({ message: "Gagal menghapus pengguna." });
      }

      return res.status(204).send(); // Respon tanpa konten (No Content) untuk mengindikasikan
penghapusan pengguna berhasil.
    });
  });
});

// DELETE data movies berdasarkan ID
// DELETE data movies berdasarkan ID
router.delete("/movies/:id", (req, res) => {
  const id = parseInt(req.params.id);

  // Periksa apakah film dengan ID yang diberikan ada dalam database
  pool.query("SELECT * FROM movies WHERE id = $1", [id], (error, result) => {
    if (error) {
      return res.status(500).json({ message: "Gagal menghapus film." });
    }

    if (result.rows.length === 0) {
      return res.status(404).json({ message: "Film tidak ditemukan." });
    }

    // Jika film ditemukan, hapus film dengan ID tersebut
    pool.query("DELETE FROM movies WHERE id = $1", [id], (error) => {
      if (error) {
        return res.status(500).json({ message: "Gagal menghapus film." });
      }

      return res.status(204).send(); // Respon tanpa konten (No Content) untuk mengindikasikan
penghapusan film berhasil.
    });
  });
});
```

d. Pus
http://localhost:3000/**users/:id**
http://localhost:3000/**movie/:id**

```
router.put("/users/:id", (req, res) => {
  const id = parseInt(req.params.id);
  const { email, password, gender, role } = req.body;

  if (!email || !password || !gender || !role) {
    return res.status(400).json({ message: "Harap isi semua field." });
  }

  // Lakukan pembaruan data users
  pool.query(
    "UPDATE users SET email = $1, password = $2, gender = $3, role = $4 WHERE id = $5",
    [email, password, gender, role, id],
    (error) => {
      if (error) {
        return res
          .status(500)
          .json({ message: "Gagal memperbarui data pengguna." });
      }
      return res
        .status(200)
        .json({ message: "Data pengguna (users) diperbarui." });
    }
  );
});

// PUT data movies
// PUT data movies berdasarkan ID
router.put("/movies/:id", (req, res) => {
  const id = parseInt(req.params.id);
  const { title, genres, year } = req.body;

  if (!title || !genres || !year) {
    return res.status(400).json({ message: "Harap isi semua field." });
  }

  // Lakukan pembaruan data film
  pool.query(
    "UPDATE movies SET title = $1, genres = $2, year = $3 WHERE id = $4",
    [title, genres, year, id],
    (error) => {
      if (error) {
        return res
          .status(500)
          .json({ message: "Gagal memperbarui data film." });
      }
      return res
        .status(200)
        .json({ message: "Data film (movies) diperbarui." });
    }
  );
});
```

Soal 2: Lakukan Pagination pada GET users dan GET movies dengan limit 10 user.

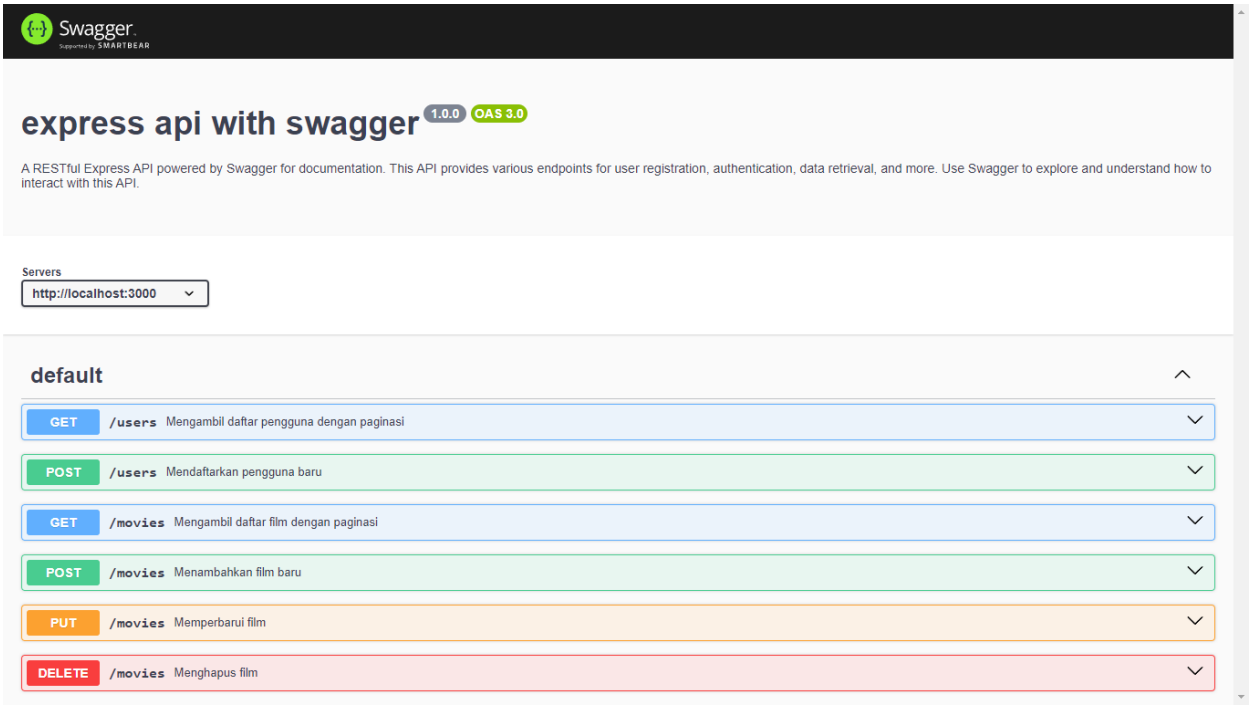
http://localhost:3000/users?page=1

http://localhost:3000/movie?page=2

```
router.get("/users", (req, res) => {
  const page = parseInt(req.query.page) || 1; // Halaman default adalah 1
  const limit = 10; // Batasan jumlah item per halaman

  // Menghitung offset (indeks data pertama untuk halaman yang diberikan)
  const offset = (page - 1) * limit;

  // Lakukan query database untuk mengambil data pengguna dengan paginasi
  pool.query(
    "SELECT * FROM users LIMIT $1 OFFSET $2",
    [limit, offset],
    (error, result) => {
      if (error) {
        return res
          .status(500)
          .json({ message: "Gagal mengambil data pengguna." });
      }
      return res.status(200).json(result.rows);
    }
  );
});

// Rute untuk mendapatkan film dengan paginasi
router.get("/movies", (req, res) => {
  const page = parseInt(req.query.page) || 1; // Halaman default adalah 1
  const limit = 10; // Batasan jumlah item per halaman

  // Menghitung offset (indeks data pertama untuk halaman yang diberikan)
  const offset = (page - 1) * limit;

  // Lakukan query database untuk mengambil data film dengan paginasi
  pool.query(
    "SELECT * FROM movies LIMIT $1 OFFSET $2",
    [limit, offset],
    (error, result) => {
      if (error) {
        return res.status(500).json({ message: "Gagal mengambil data film."
});   }
      return res.status(200).json(result.rows);
    }
  );
});
```

Soal 3: Buatlah dokumentasi API menggunakan swagger

http://localhost:3000/api-docs




```
const swaggerJSDoc = require("swagger-jsdoc");
const swaggerUI = require("swagger-ui-express");

const options = {
  definition: {
    openapi: "3.0.0",
    info: {
      title: "express api with swagger",
      version: "1.0.0",
      description:
        "A RESTful Express API powered by Swagger for documentation. This API provides various
endpoints for user registration, authentication, data retrieval, and more. Use Swagger to explore and
understand how to interact with this API.",
    },
    servers: [
      {
        url: "http://localhost:3000",
      },
    ],
  },
  // Definisikan file-file yang mengandung komentar Swagger
  apis: ["./src/routes/index.js"],
};

const swaggerSpec = swaggerJSDoc(options);
app.use("/api-docs", swaggerUI.serve, swaggerUI.setup(swaggerSpec));

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Soal 4: Implementasikan Logging server pada aplikasi yang teman-teman buat