

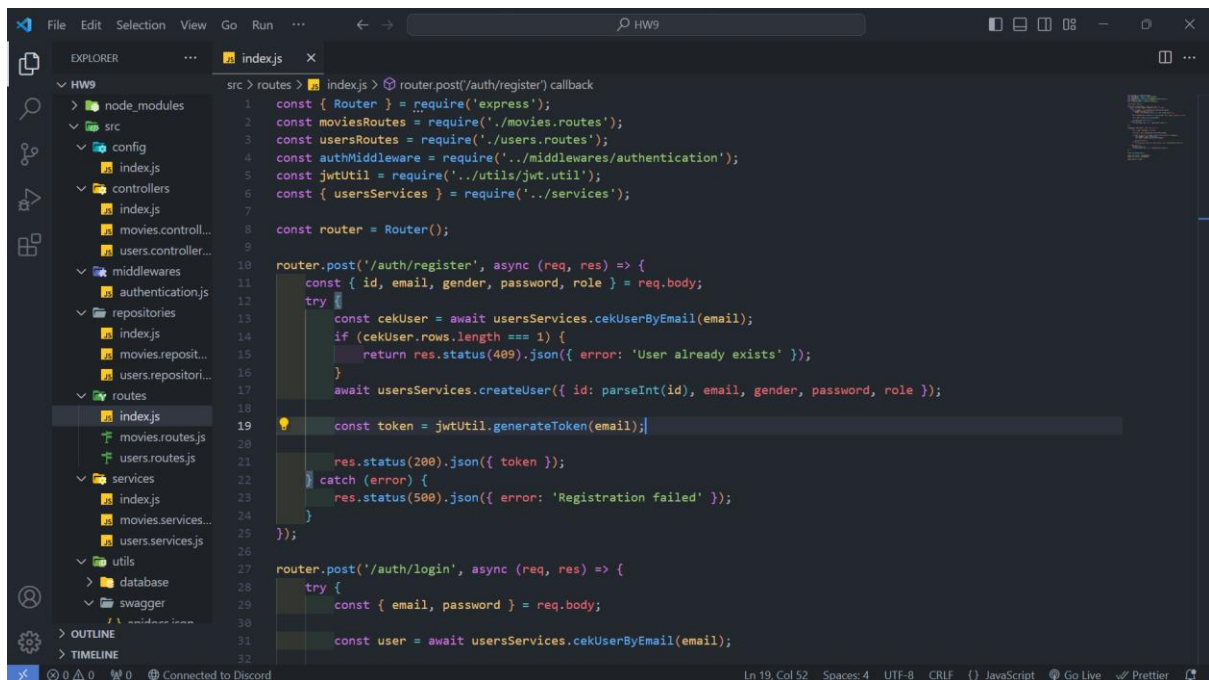
Nama : Aswangga Bhanu Rizqullah

Kelas : 5 A / Kelompok 3

Homework week 9

Kerjakan soal homework ini berdasarkan data SQL berikut.

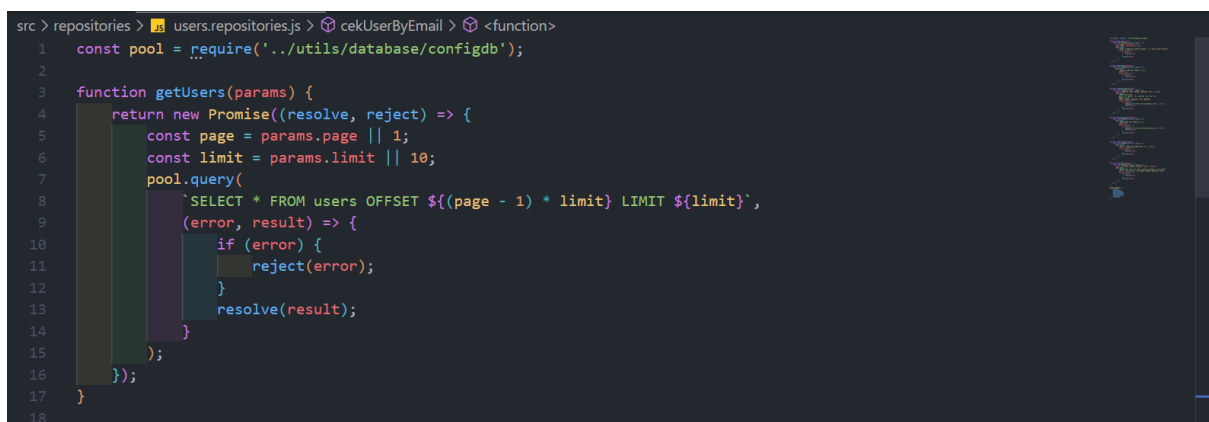
Soal 1: Buatlah RESTful API yang terdiri dari GET, POST, DELETE, dan PUT. Setelah itu buatlah endpoint untuk register user dan login user untuk implementasi authorization dan authentication. Pastikan yang hanya bisa mengakses API hanyalah user yang terdaftar.



```
File Edit Selection View Go Run ... HW9
EXPLORER
  HW9
    node_modules
    src
      config
        index.js
      controllers
        index.js
        movies.controller.js
        users.controller.js
      middlewares
        authentication.js
      repositories
        index.js
        movies.repository.js
        users.repository.js
      routes
        index.js
        movies.routes.js
        users.routes.js
      services
        index.js
        movies.services.js
        users.services.js
      utils
        database
        swagger
  OUTLINE
  TIMELINE
  Connected to Discord

src > routes > index.js > router.post('/auth/register') callback
1  const { Router } = require('express');
2  const moviesRoutes = require('../movies.routes');
3  const usersRoutes = require('../users.routes');
4  const authMiddleware = require('../middlewares/authentication');
5  const jwtUtil = require('../utils/jwt.util');
6  const { usersServices } = require('../services');
7
8  const router = Router();
9
10 router.post('/auth/register', async (req, res) => {
11   const { id, email, gender, password, role } = req.body;
12   try {
13     const cekUser = await usersServices.cekUserByEmail(email);
14     if (cekUser.rows.length === 1) {
15       return res.status(409).json({ error: 'User already exists' });
16     }
17     await usersServices.createUser({ id: parseInt(id), email, gender, password, role });
18   }
19   const token = jwtUtil.generateToken(email);
20   res.status(200).json({ token });
21   catch (error) {
22     res.status(500).json({ error: 'Registration failed' });
23   }
24 });
25
26 router.post('/auth/login', async (req, res) => {
27   try {
28     const { email, password } = req.body;
29     const user = await usersServices.cekUserByEmail(email);
30   }
31 });
32
```

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



```
src > repositories > users.repositories.js > cekUserByEmail > <function>
1  const pool = require('../utils/database/configdb');
2
3  function getUsers(params) {
4    return new Promise((resolve, reject) => {
5      const page = params.page || 1;
6      const limit = params.limit || 10;
7      pool.query(
8        `SELECT * FROM users OFFSET ${(page - 1) * limit} LIMIT ${limit}`,
9        (error, result) => {
10         if (error) {
11           reject(error);
12         }
13         resolve(result);
14       }
15     });
16   });
17 }
18
```

```

1  const pool = require('../utils/database/configdb');
2
3  function getMovies(params) {
4      return new Promise((resolve, reject) => {
5          const page = params.page || 1;
6          const limit = params.limit || 10;
7          pool.query(
8              `SELECT * FROM movies OFFSET ${page - 1} * limit} LIMIT ${limit}`,
9              (error, result) => {
10                 if (error) {
11                     reject(error);
12                 }
13                 resolve(result);
14             }
15         );
16     });
17 }

```

Soal 3: Buatlah dokumentasi API menggunakan swagger

The screenshot displays the Swagger UI interface for an API. At the top, the 'Servers' section shows the base URL 'http://localhost:3000/ - localhost' and an 'Authorize' button. The main content is divided into two sections: 'Users' and 'Movies'.

Users (The User managing API)

- POST** `/api/v1/auth/register`: Create user data
- POST** `/api/v1/auth/login`: Login users
- GET** `/api/v1/users`: Get all user (locked)
- GET** `/api/v1/users/{id}`: Get User by Id (locked)
- PUT** `/api/v1/users/{id}`: Update User by Id (locked)
- DELETE** `/api/v1/users/{id}`: Delete User by Id (locked)

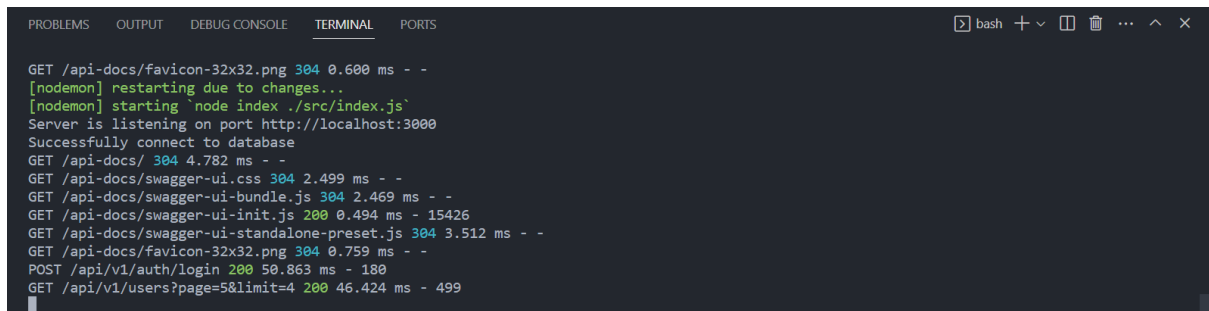
Movies (The Country managing API)

- GET** `/api/v1/movies`: Get all movie (locked)
- POST** `/api/v1/movies`: Add movie data (locked)
- GET** `/api/v1/movies/{id}`: Get Movie by Id (locked)
- PUT** `/api/v1/movies/{id}`: Update Movie by Id (locked)
- DELETE** `/api/v1/movies/{id}`: Delete Movie by Id (locked)

Schemas

- Users** >
- Movie** >

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

A screenshot of a terminal window with a dark background. The terminal shows the output of a Node.js application running with nodemon. The logs include messages about restarting the server, starting the index.js file, and listening on port 3000. It also shows successful database connection and several HTTP requests with their status codes and response times. The requests include GET requests for API docs, Swagger UI, and a POST request for login.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
GET /api-docs/favicon-32x32.png 304 0.600 ms - -
[nodemon] restarting due to changes...
[nodemon] starting `node index ./src/index.js`
Server is listening on port http://localhost:3000
Successfully connect to database
GET /api-docs/ 304 4.782 ms - -
GET /api-docs/swagger-ui.css 304 2.499 ms - -
GET /api-docs/swagger-ui-bundle.js 304 2.469 ms - -
GET /api-docs/swagger-ui-init.js 200 0.494 ms - 15426
GET /api-docs/swagger-ui-standalone-preset.js 304 3.512 ms - -
GET /api-docs/favicon-32x32.png 304 0.759 ms - -
POST /api/v1/auth/login 200 50.863 ms - 180
GET /api/v1/users?page=5&limit=4 200 46.424 ms - 499
```

Repository github:

<https://github.com/Bhanurzh/RakaminFSWD/tree/main/HW9>