LAPORAN PRAKTIKUM DATABASE MOVIE & DIRECTOR

INTEROPERABILITAS

POLITEKNIK NEGERI BANYUWANGI

TAHUN PELAJARAN 2025

DI SUSUN OLEH :

FIRMAN ARDIANSYAH
NIM: 362458302101

DOSEN PENGAMPU :

SEPYAN PURNAMA KRISTANTO, S.KOM., M.KOM.

PROGRAM STUDI TEKNOLOGI REKAYASA PERANGKAT LUNAK

JURUSAN BISNIS DAN INFORMATIKA

POLITEKNIK NEGERI BANYUWANGI

2025

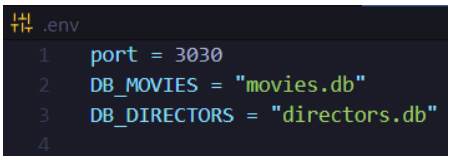
I. PENDAHULUAN

Laporan praktikum ini membahas mengenai implementasi persistensi data sutradara menggunakan Node.js, Express, dan SQLite. Tujuan dari praktikum ini yaitu untuk mengaplikasikan keterampilan menghubungkan express dengan sqlite pada sumber daya baru dan memperkuat pemahaman tentang perintah SQL serta penanganan logika asinkron dengan callback.

II. AKTIVITAS PRAKTIKUM

Pada aktivitas praktikum ini, kita akan melakukan modifikasi skema databse dan refactoring endpoint /directors, karena percobaan sebelumnya pada endpoint /movies sudah dilakukan. Tujuannya untuk menyimpan dan mengelola data director/sutradara secara persistensi di dalam database SQLite.

- Tambahkan/ubah file ”.env”
  Untuk menentukan port dimana server akan berjalan dan untuk memisahkan nilai-nilai konfigurasi dari logika inti program, sehingga memudahkan untuk merubah pengaturan tanpa perlu memodifiksai kode utama.



- Modifikasi skema database
  Modifikasi file ”database.js”, tambahkan kode program untuk membuat tabel directors dengan 3 kolom yaitu id, name, dan birthYear. Kemudian isikan data direktor sesuai dengan filmnya.

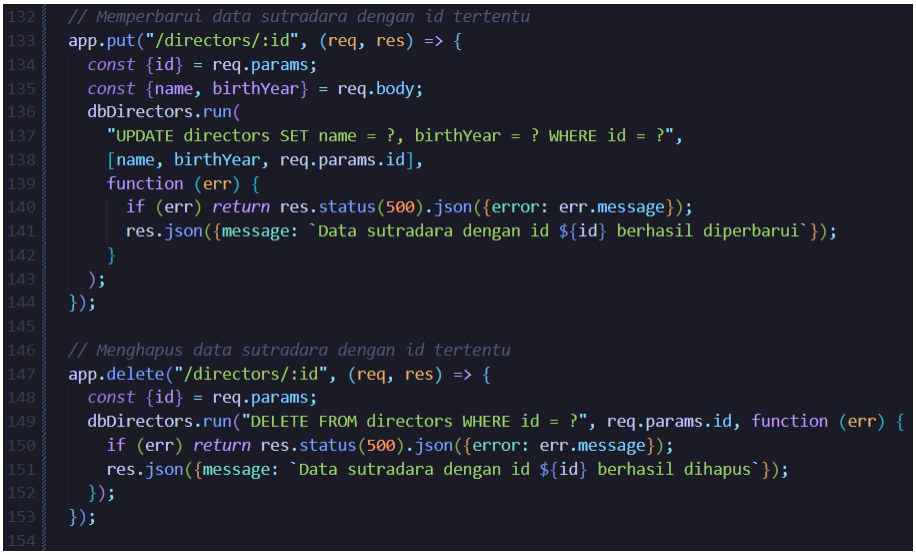
```
// Koneksi ke database directors || directors.db
const dbDirectors = new sqlite3.Database(DB_DIRECTORS, (err) => {
  if (err) {
    console.error("Error connect ke directors.db:", err.message);
    throw err;
  } else {
    console.log("Connected to directors.db");

    dbDirectors.run(`CREATE TABLE IF NOT EXISTS directors (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      birthYear INTEGER
    )`, (err) => {
      if (!err) {
        console.log("Table 'directors' created. Seeding initial data...");
        const insert = 'INSERT INTO directors (name, birthYear) VALUES (?, ?)';
        dbDirectors.run(insert, ["Peter Jackson", 1961]);
        dbDirectors.run(insert, ["Sam Raimi", 1959]);
        dbDirectors.run(insert, ["Firman Ardiansyah", 2006]);
      }
    });
  }
});

// Export dua koneksi database
module.exports = { dbMovies, dbDirectors };
```

Pada bagian atas tambahkan kode berikut untuk membaca nilai dari variabel (DB_MOVIES dan DB_DIRECTORS) yang ditetapkan di luar program.

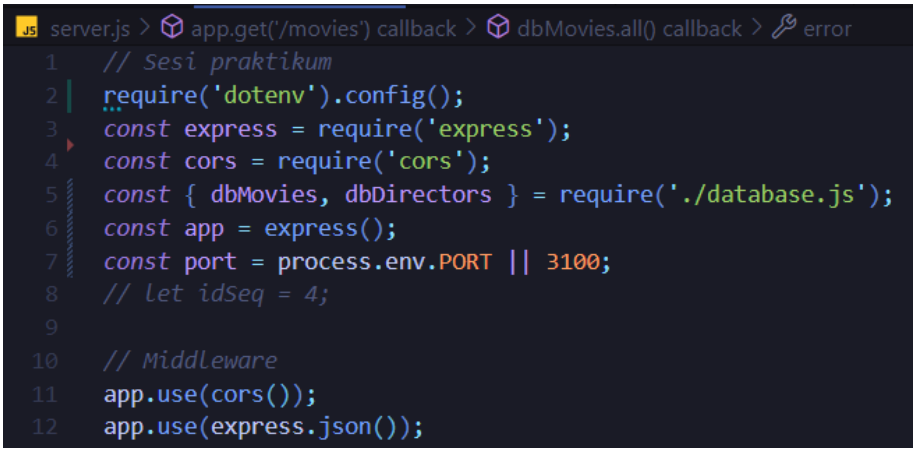
```
require('dotenv').config();
const sqlite3 = require('sqlite3').verbose();

// Mengambil nama file DB dari .env (dengan fallback)
const DB_MOVIES = process.env.DB_MOVIES || "movies.db";
const DB_DIRECTORS = process.env.DB_DIRECTORS || "directors.db";
```

- Refactoring endpoint /director
  Modifikasi file ”server.js”, semua endpoint directors menggunakan query SQL ke tabel directors (select untuk GET, insert into untuk POST, update untuk PUT, dan delete untuk DELETE).

```
// Melihat daftar semua sutradara
app.get("/directors", (req, res) => {
  dbDirectors.all("SELECT * FROM directors", [], (err, rows) => {
    if (err) return res.status(500).json({error: err.message});
    res.json(rows);
  });
});

// Melihat data sutradara berdasarkan id
app.get("/directors/:id", (req, res) => {
  dbDirectors.get("SELECT * FROM directors WHERE id = ?", [req.params.id], (err, row) =>
    if (err) return res.status(500).json({error: err.message});
    if (!row) return res.status(404).json({error: "Director not found"});
    res.json(row);
  });
});

// Menambah data sutradara
app.post("/directors", (req, res) => {
  const {name, birthYear} = req.body;
  dbDirectors.run(
    "INSERT INTO directors (name, birthYear) VALUES (?, ?)",
    [name, birthYear],
    function (err) {
      if (err) return res.status(500).json({error: err.message});
      res.json({id: this.lastID, name, birthYear});
    }
  );
});
```



```
// Memperbarui data sutradara dengan id tertentu
app.put("/directors/:id", (req, res) => {
  const {id} = req.params;
  const {name, birthYear} = req.body;
  dbDirectors.run(
    "UPDATE directors SET name = ?, birthYear = ? WHERE id = ?",
    [name, birthYear, req.params.id],
    function (err) {
      if (err) return res.status(500).json({error: err.message});
      res.json({message: `Data sutradara dengan id ${id} berhasil diperbarui`});
    }
  );
});

// Menghapus data sutradara dengan id tertentu
app.delete("/directors/:id", (req, res) => {
  const {id} = req.params;
  dbDirectors.run("DELETE FROM directors WHERE id = ?", req.params.id, function (err) {
    if (err) return res.status(500).json({error: err.message});
    res.json({message: `Data sutradara dengan id ${id} berhasil dihapus`});
  });
});
```



```
server.js > app.get('/movies') callback > dbMovies.all() callback > error
1   // Sesi praktikum
2   require('dotenv').config();
3   const express = require('express');
4   const cors = require('cors');
5   const { dbMovies, dbDirectors } = require('./database.js');
6   const app = express();
7   const port = process.env.PORT || 3100;
8   // let idSeq = 4;
9
10  // Middleware
11  app.use(cors());
12  app.use(express.json());
```

Pada bagian atas tambahkan kode “baris ke 1 dan ke 5”. Kode baris ke 1 digunakan untuk mengimpor dan mengkonfigurasi pustaka dotenv. Pustaka

tersebut berfungsi untuk memuat environment variables dari file ”.env” kedalam ”process.env”. kode baris ke 5 digunakan untuk mengimpor 2 objek (dbMovies dan dbDirectors) dari file ”database.js”. objek tersebut adalah koneksi ke database SQLite yang berbeda (dbMovies untuk data film, dbDirectors untuk data sutradara).

III. ANALISIS

Dari aktivitas praktikum ini menunjukkan penerapan prinsip separation of concerns yang mana memisahkan kode menjadi bagian-bagian yang berbeda, setiap bagian menangani satu urusan/tanggung jawab tunggal yang spesifik. Tujuan dari prinsip tersebut yaitu untuk membuat kode lebih terstruktur, mudah dikelola, dan dapat digunakan kembali/reuseable. Jadi, dengan penerapan prinsip tersebut logika database nantinya dipisahkan dari logika server dalam file yang berbeda (database.js dan server.js). penggunaan SQLite memungkinkan kita untuk memiliki database lokal yang ringan, ideal untuk pengembangan. Selain itu, penanganan logika asinkron dengan menggunakan callback menjadi kunci dalam berinteraksi dengan database. Setiap operasi databsae (seperti db.run, db.all, dll) memerlukan callback untuk menangani respons atau kesalahan setelah operasi selesai.

IV. KESIMPULAN

Dari hasil praktikum ini menunjukkan bahwa kita berhasil mengimplementasikan persistensi data untuk table movies/film dan directors/sutradara. Dengan memodifikasi skema database untuk menambahkan/membuat tabel baru dan merefaktor/merubah endpoint /movies dan /directors dari menggunakan data dalam memori menjadi menggunakan query sql untuk berinteraksi dengan baris data. Untuk menghindari data yang duplikasi selama aktivitas praktikum disarankan saat running server.js untuk tidak menggunakan perintah ”nodemon server.js” dikarenakan jika terdapat perubahan/update kode program maka sistem/server akan auto restart, jadi sistem akan menjalankan perintah create/insert setiap kali server restart, atau kita bisa mengatasinya dengan menambahkan fungsi untuk mengecek data terlebih dahulu agar tidak terjadi redundansi data.