**BAB IV**

**Middleware**

1. **Materi Pembelajaran**

Middleware pada Express.js adalah fungsi-fungsi yang berjalan di antara proses permintaan (*request*) dan proses tanggapan (*response*) pada aplikasi web . Middleware digunakan untuk melakukan berbagai tugas, seperti memodifikasi objek permintaan atau tanggapan, menangani kesalahan, melakukan otentikasi, atau menjalankan fungsi tertentu sebelum atau setelah pengiriman tanggapan ke klien.

Setiap middleware memiliki akses ke objek *request* (req), objek *response* (res), dan fungsi *next*. Fungsi *next* digunakan untuk melanjutkan proses middleware berikutnya dalam siklus permintaan-tanggapan. Middleware juga bisa mengakhiri siklus permintaan-tanggapan tanpa melanjutkan ke middleware berikutnya, misalnya dengan mengirim tanggapan langsung ke klien.

Berikut adalah contoh beberapa penggunaan middleware dalam Express.js:

1. *Logger Middleware*: Middleware ini digunakan untuk mencatat informasi permintaan seperti metode HTTP, URL, waktu permintaan, dan sebagainya.
2. *Body Parser Middleware*: Middleware ini digunakan untuk mem-parsing body dari permintaan HTTP, terutama dalam format JSON atau URL-encoded.
3. *Authentication Middleware*: Middleware ini digunakan untuk memverifikasi identitas pengguna, seperti memeriksa token atau kredensial pengguna.
4. *Error Handling Middleware*: Middleware ini digunakan untuk menangani kesalahan yang terjadi selama penanganan permintaan, seperti kesalahan server atau kesalahan validasi.
5. *Custom Middleware*: juga bisa membuat middleware kustom sesuai kebutuhan aplikasi , seperti middleware untuk menerapkan kebijakan keamanan tertentu atau middleware untuk melacak aktivitas pengguna.

**MERN STACK**

MERN adalah singkatan dari MongoDB, Express.js, React, dan Node.js. Ini adalah stack pengembangan perangkat lunak yang populer untuk membangun aplikasi web berbasis JavaScript. Di dalam MERN, Express.js berperan sebagai bagian dari backend untuk menangani permintaan HTTP, memproses logika bisnis, dan berkomunikasi dengan database MongoDB.

Hubungan antara Express.js dan MERN adalah sebagai berikut:

1. Express.js sebagai Bagian dari Backend MERN:

Express.js digunakan sebagai kerangka kerja backend dalam stack MERN. Ini berarti Express menangani rute permintaan HTTP, middleware, manajemen sesi, dan sebagainya dalam backend aplikasi web MERN.

1. API Endpoint dan Routing:

Dalam aplikasi MERN, Express.js digunakan untuk menentukan dan menangani endpoint API untuk berkomunikasi antara frontend (dibangun dengan React) dan backend. Express menyediakan kemampuan routing yang kuat untuk menangani permintaan ke berbagai rute dan mengirim tanggapan yang sesuai.

1. Middleware:

Seperti yang dijelaskan sebelumnya, Express menggunakan middleware untuk melakukan berbagai tugas dalam siklus permintaan-tanggapan. Middleware ini sangat penting dalam membangun aplikasi web MERN untuk melakukan tugas-tugas seperti otentikasi, parsing permintaan, logging, dan sebagainya.

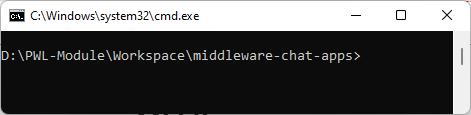
1. Integrasi dengan MongoDB:

Salah satu keunggulan MERN adalah penggunaan database NoSQL MongoDB. Express.js memungkinkan untuk berinteraksi dengan MongoDB melalui driver atau ORM seperti Mongoose. Dengan Express, dapat membuat endpoint API yang berkomunikasi dengan MongoDB untuk melakukan operasi CRUD (Create, Read, Update, Delete) pada data.

Dengan demikian, Express.js adalah bagian integral dari stack teknologi MERN dan berfungsi sebagai backend yang kuat untuk membangun aplikasi web modern dengan JavaScript.

1. **Installasi**
2. Directory project middleware express

Buatlah sebuah directory path project baru beranama middleware-chat-apps lalu bukalah terminal dan masuk kedalam path tersebut



1. Installasi Express JS

Untuk menginstall library Express JS pada node js, masukan perintah dibawah ini untuk melakukan initialisasi project:

|  |
| --- |
| npm init |

Selanjutnya masukan perintah dibawah ini untuk menginstall express kedalam project :

|  |
| --- |
| npm install express |

Jika telah melakukan installasi dari kedua perintah tersebut maka pada project akan membuat sebuah file bernama package.json dan package-lock.json

1. Runing Express JS

Jika installasi sudah berhasil buatlah file bernama index.js dan masukan code seperti dibawah ini untuk menjalankan server middleware:

|  |  |
| --- | --- |
| require("dotenv").config();  const express = require('express')  const app = express()  const port = process.env.API\_PORT;  app.get('/', (req, res) => {    res.send('Hello World!')  })  app.listen(port, () => {    console.log(`Server app listening on http://localhost:${port}`);  }); |  |

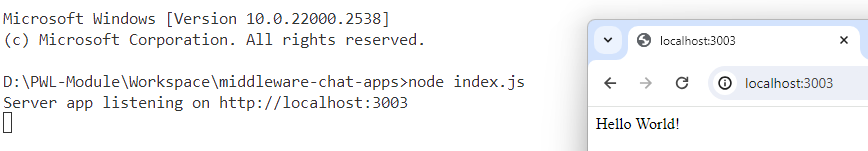
Buatlah file bernama *.env* digunakan untuk menyimpan konfigurasi lingkungan (*environment configuration*) yang sensitif dan spesifik untuk aplikasi , seperti kata sandi database, API keys, dan rahasia lainnya. File ini biasanya tidak dimasukkan ke dalam sistem kontrol versi (seperti Git) untuk menjaga agar informasi sensitif tidak secara tidak sengaja dibagikan atau disimpan di repositori kode sumber.

|  |
| --- |
| API\_PORT=3003  NODE\_ENV=development  EXPRESS\_DB\_HOST=localhost  EXPRESS\_DB\_USERNAME=root  EXPRESS\_DB\_PASSWORD=  EXPRESS\_DB\_NAME=  EXPRESS\_DB\_PORT=3306 |

Setelah membuat kedua file tersebut untuk menjalankan server middleware gunakan perintah dibawah ini:

|  |
| --- |
| node index.js |

Setelah muncul informasi seperti gambar dibawah ini, bukalah browser untuk mengecek apakah server tersebut sudah berjalan dengan baik:



1. **Penggunaan life cycle pada express js**
2. **Routing**

Routing pada Express.js adalah proses menentukan *endpoint* (*rute*) pada aplikasi web dan menanggapi permintaan HTTP yang diterima pada rute-rute tersebut. Express menyediakan metode seperti GET, POST, PUT, DELETE, dan lain-lain untuk menangani jenis-jenis permintaan HTTP yang berbeda.

Berikut adalah contoh sederhana dari basic routing pada Express.js:

|  |
| --- |
| // Routing untuk metode GET  app.get('/', (req, res) => {    res.send('Halo, dunia!');  });  // Routing untuk metode POST  app.post('/login', (req, res) => {    res.send(' telah berhasil masuk!');  });  // Routing untuk metode PUT  app.put('/users/:id', (req, res) => {    const userId = req.params.id;    res.send(` telah mengubah data user dengan ID ${userId}`);  });  // Routing untuk metode DELETE  app.delete('/users/:id', (req, res) => {    const userId = req.params.id;    res.send(` telah menghapus data user dengan ID ${userId}`);  }); |

* **Rute '/'** ditangani oleh metode GET. Ketika mengakses root URL aplikasi (misalnya http://localhost:3003/), server akan mengirimkan respons "Halo, dunia!".
* **Rute '/login'** ditangani oleh metode POST. Ketika ada permintaan POST ke URL http://localhost:3003/login, server akan mengirimkan respons " telah berhasil masuk!".
* **Rute '/users/:id'** ditangani oleh metode PUT. Ini adalah rute dinamis yang mengambil parameter id dari URL. Ketika ada permintaan PUT ke URL http://localhost:3003/users/123, server akan mengirimkan respons " telah mengubah data user dengan ID 123".
* **Rute '/users/:id'** juga ditangani oleh metode DELETE. Ketika ada permintaan DELETE ke URL http://localhost:3003/users/123, server akan mengirimkan respons " telah menghapus data user dengan ID 123".

1. **Static file**

Static file pada Express adalah file yang disajikan secara langsung oleh server web tanpa memprosesnya terlebih dahulu. File-file seperti HTML, CSS, JavaScript, gambar, dan file-file statis lainnya dapat disajikan sebagai static file oleh Express tanpa memerlukan pemrosesan tambahan.

Express menyediakan middleware built-in yang disebut *express.static* untuk menangani pengiriman file statis. Middleware ini berfungsi dengan cara mengonfigurasi direktori di mana file-file statis tersebut tersedia, dan kemudian mengirimkan file-file tersebut ke klien saat diminta.

Berikut adalah contoh cara menggunakan middleware express.static dalam aplikasi Express:

|  |
| --- |
| app.use(express.static('public')); |

Dalam contoh di atas, direktori 'public' diatur sebagai direktori statis menggunakan express.static. Ini berarti bahwa semua file dalam direktori public (misalnya public/index.html, public/style.css, public/script.js) dapat diakses secara langsung melalui server Express.

1. **Logger**

Middleware ini berfungsi untuk mencatat informasi permintaan HTTP yang masuk, seperti metode permintaan, URL, dan waktu permintaan.

|  |
| --- |
| const myLogger = (req, res, next) => {    console.log(`${new Date().toISOString()}: ${req.method} ${req.url}`);    next();  };  app.use(myLogger); |

Dengan menggunakan middleware seperti Logger, dapat dengan mudah menambahkan fungsionalitas tambahan ke aplikasi Express , seperti logging, otentikasi, manajemen sesi, dan banyak lagi, tanpa harus menyertakan kode yang sama di setiap rute. Middleware adalah komponen penting dalam arsitektur aplikasi Express karena membantu memisahkan logika bisnis dari penanganan permintaan HTTP.

1. Error Handling

Error handling di Express.js dilakukan menggunakan middleware khusus yang ditujukan untuk menangkap kesalahan yang terjadi selama penanganan permintaan (request) dan menanggapi kesalahan tersebut dengan cara yang sesuai. Middleware error handling dijalankan ketika terjadi kesalahan (error) dalam middleware atau handler rute lainnya, dan middleware ini akan menghentikan eksekusi dan menanggapi dengan pesan kesalahan yang sesuai.

Berikut adalah langkah-langkah umum dalam menangani error di Express.js:

1. Mendaftarkan Middleware Error Handling:

perlu mendaftarkan middleware error handling di aplikasi Express menggunakan metode app.use() atau app.all(). Middleware error handling ini harus memiliki empat parameter (err, req, res, next) dan diikuti oleh middleware atau handler rute lainnya. Middleware ini akan menangkap kesalahan yang dilemparkan oleh middleware atau handler rute lainnya dalam siklus permintaan-tanggapan.

1. Menangkap dan Menanggapi Kesalahan:

Middleware error handling kemudian menangkap kesalahan yang dilemparkan oleh middleware atau handler rute lainnya. bisa menanggapi kesalahan ini dengan cara mengirimkan respons yang sesuai, seperti pesan kesalahan atau kode status tertentu.

1. Memanggil Fungsi next(err):

Jika ingin menangani kesalahan dalam middleware atau handler rute, bisa memanggil fungsi next() dengan argumen kesalahan (biasanya disebut err). Ini akan mengarahkan eksekusi ke middleware error handling.

Berikut adalah contoh implementasi middleware error handling di Express.js:

|  |
| --- |
| app.use((err, req, res, next) => {    console.error(err.stack);    res.status(500).send('Terjadi kesalahan pada server');  });  app.get('/', (req, res) => {    const error = new Error('Kesalahan yang disimulasikan');    next(error);  }); |

1. **Object-Relational Mapping (ORM)**

Sequelize adalah sebuah ORM (Object-Relational Mapping) untuk Node.js yang menyediakan antarmuka berbasis JavaScript untuk berinteraksi dengan berbagai macam jenis database relasional, seperti MySQL, PostgreSQL, SQLite, dan lain-lain. Sequelize memudahkan pengembang untuk mengelola struktur database, melakukan operasi CRUD (Create, Read, Update, Delete), dan menjalankan kueri kompleks dengan menggunakan objek JavaScript, tanpa perlu menulis kueri SQL secara langsung.

Fungsi dan manfaat utama Sequelize dalam aplikasi Express:

1. Model-View-Controller (MVC): Sequelize memungkinkan pengembang untuk mendefinisikan model-model data dalam aplikasi mereka. Model-model ini merepresentasikan tabel-tabel dalam basis data dan berfungsi sebagai abstraksi yang memisahkan logika bisnis dari logika akses data.
2. Interaksi dengan Database:

* Membuat, membaca, memperbarui, dan menghapus data (CRUD).
* Mengeksekusi kueri kompleks menggunakan kriteria pencarian, pengelompokan, pengurutan, dan lain-lain.
* Menangani hubungan antara tabel, seperti one-to-one, one-to-many, dan many-to-many.

1. Validasi Data:

Sequelize memungkinkan Anda untuk menetapkan aturan validasi pada model Anda untuk memastikan bahwa data yang dimasukkan ke dalam database sesuai dengan kebutuhan aplikasi dan format yang diharapkan.

1. Migrasi Basis Data:

Sequelize menyediakan fasilitas untuk membuat dan menjalankan migrasi basis data. Ini memudahkan dalam mengelola struktur basis data selama siklus pengembangan aplikasi, dari pengembangan hingga produksi.

1. Keamanan:

Sequelize membantu dalam mencegah serangan SQL injection dengan menggunakan parameterized queries dan sanitasi data otomatis.

1. Portabilitas Basis Data:

Dengan menggunakan Sequelize, aplikasi Anda dapat lebih mudah dipindahkan dari satu jenis basis data ke basis data lainnya tanpa perlu melakukan perubahan besar pada kode aplikasi. Sequelize akan menangani perbedaan sintaksis SQL antara berbagai jenis basis data.

1. **Latihan: Membangun Aplikasi Messenger - Chapter 3**

Pada bagian ini kita akan membangun sebuah backend dari aplikasi messenger yang telah dibuat sebelumnya. Backend ini akan menggunakan middleware yang dikembangkan oleh Express JS.

1. Installasi library pendukung

Tambahkan perintah ini pada terminal node js projek anda:

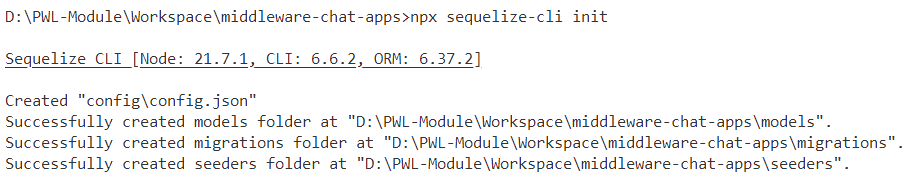
|  |
| --- |
| npm install cors sequelize mysql2 dotenv |

1. Membangun Migrations

Setelah anda menginstall library sequelize sebagai ORM pada middleware, masukan perintah dibawah ini untuk melakukan inisialisasi ORM:

|  |
| --- |
| npx sequelize-cli init |

Jika melihat gambar dibawah ini merupakan gambaran jika sudah berhasil melakukan inisialisasi ORM pada middleware:



1. Konfigurasi database pada migrations

Buatlah file *environment configuration* (.env) dengan menambahkan informasi variable konstanta sebagai berikut:

|  |
| --- |
| API\_PORT=3003  NODE\_ENV=development  EXPRESS\_DB\_HOST=localhost  EXPRESS\_DB\_USERNAME=root  EXPRESS\_DB\_PASSWORD=  EXPRESS\_DB\_NAME=pwl\_chat\_db  EXPRESS\_DB\_PORT=3306 |

Buatlah sebuah file pada config/config.js untuk menkoneksikan dengan database bernama pwl\_chat\_db, contoh disini akan menggunakan driver database MYSQL:

|  |
| --- |
| require("dotenv").config();  module.exports = {  development: {  username: process.env.EXPRESS\_DB\_USERNAME,  password: process.env.EXPRESS\_DB\_PASSWORD,  database: process.env.EXPRESS\_DB\_NAME,  host: process.env.EXPRESS\_DB\_HOST,  dialect: "mysql",  port: process.env.EXPRESS\_DB\_PORT,  pool: {  max: 10,  min: 0,  acquire: 30000,  idle: 10000  }  },  test: {...},  production: {...},  }; |

Modifikasi file index.js yang berada pada folder models, menjadi sebagai berikut:

|  |
| --- |
| Sebelum: |
| ...  const config = require(\_\_dirname + '/../config/**config.json'**)[env];  ... |
| Update menjadi: |
| ...  const config = require(\_\_dirname + '/../config/**config.js**')[env];  ... |

1. A screenshot of a computer

   Description automatically generatedModifikasi file server index.js untuk memanggil koneksi kedalam database

Pada file index.js tambahkan script dibawah ini untuk memanggil koneksi terhadap database apakah sudah berhasil terkoneksi atau tidak:

|  |
| --- |
| ...  const db\_mysql = require("./models");  db\_mysql.sequelize.sync();  ... |

Silakan anda restart server dan jalankan kembali untuk mengecek koneksi telah berhasil:



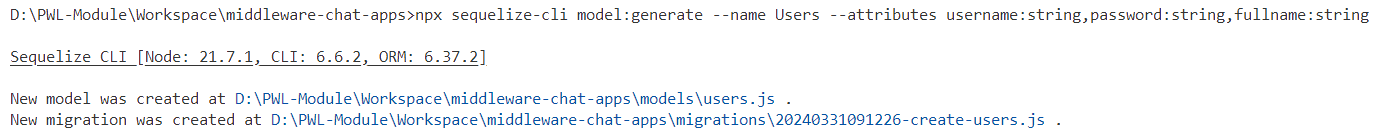
1. Membangun migration table users

Dalam contoh kali ini akan membuat sebuah table bernama users dimana hanya memiliki informasi umum seperti, user\_id, username, password dan fullname. Bukalah terminal anda dan jalankan perintah migration table dibawah ini dan update sesuai dengan attribute yang telah ditentukan:

|  |
| --- |
| npx sequelize-cli model:generate --name TableName --attributes fieldName:tipeData,fieldName:tipeData |

* **npx sequelize-cli:** npx adalah utilitas npm yang digunakan untuk menjalankan paket npm tanpa harus menginstalnya secara global. sequelize-cli adalah paket command-line interface (CLI) Sequelize yang digunakan untuk melakukan berbagai tugas, seperti menghasilkan model, migrasi, dll.
* **model:generate:** Ini adalah perintah yang digunakan untuk membuat model baru dengan Sequelize. Ini memberitahukan sequelize-cli bahwa Anda ingin membuat model.
* **--name TableName:** Argumen ini menentukan nama model yang ingin Anda buat. Dalam contoh ini, model akan dinamai "Users". Nama model harus dalam bentuk singular, meskipun tabel di database biasanya dalam bentuk plural.
* **--attributes fieldName:string:** Argumen ini menentukan atribut-atribut yang dimiliki oleh model yang akan Anda buat. Setiap atribut didefinisikan dalam format namaKolum:tipeData. Jika anda membuat lebih dari satu kolum table, jangan menggunakan tanda spasi ( ) sebagai pemisah antara tanda koma.

Contoh: fieldName:tipeData, fieldName:tipeData 



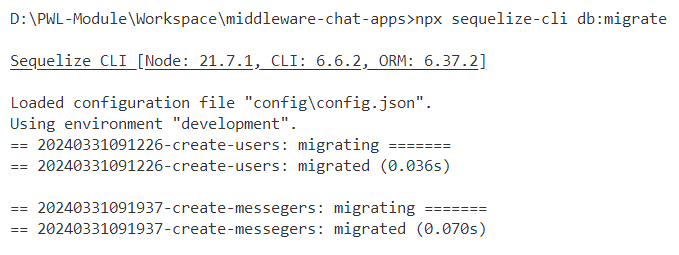
1. Membangun migration table messegers

Lakukan hal yang sama seperti tahap ke 5 untuk membuat sebuah table migration yang terdiri dari: from\_id, messages, submit\_date, to\_user\_id

1. Running Migrations

Masukan perintah dibawah ini jika ingin membuat table pada database yang telah ditentukan berdasarkan table migration yang telah dibuat sebelum:

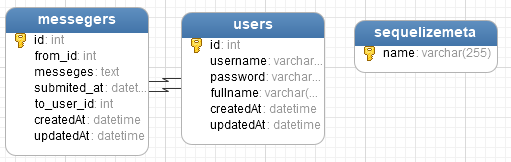
|  |
| --- |
| npx sequelize-cli db:migrate |



Jika anda ingin membatalkan perintah migration silakan masukan syntax dibawah ini:

|  |
| --- |
| npx sequelize-cli db:migrate:undo |

Jika sudah berhasil melakukan migration maka periksa database anda, table Users dan Messegers telah berhasil dibuat:



1. Membuat API data users

Buatlah folder controllers dan routes pada project middleware anda, folder controllers akan berisi beberapa script logika yang menghandle alur transaksional. Dalam contoh kasus disini kita akan membuat list API untuk data pengguna atau kontak messeger. Buatlah file **UsersController.js** seperti dibawah ini:

UsersController.js

|  |
| --- |
| const { Users } = require("../models");  exports.index = async (req, res) => {  res.json({  status: 200,  message: "prefix for end-poin users",  });  }; |

Lalu buatlah file pada routes bernama **UserRoute.js**, folder ini akan berisi layanan yang menerima permintaan HTTPS dan mengembalikan rute ideal antara dua lokasi:

UserRoute.js

|  |
| --- |
| const express = require("express");  const router = express.Router();  const userController = require("../controllers/UsersController");  router.get("/", userController.index);  module.exports = router; |

Setelah menambahkan file controller dan route, bukalah script server anda index.js, lalu tambahkan prefik end point untuk path API User:

|  |
| --- |
| ...  const userRoute = require("./routers/UserRoutes");  app.use("/api/user", userRoute);  ... |

Cek apakah API sudah berjalan susai dengan prefix dari masing-masing end-point:

A screenshot of a web browser

Description automatically generated

selanjutnya akan membuat sebuah API untuk melakukan CRUD, berikut tahapannya:

1. API Insert Data

Buatlah function createData pada controller user, untuk menambahkan data pengguna baru ke dalam database seperti dibawah ini:

|  |
| --- |
| exports.createData = async (req, res) => {  const payloadData = req.body;  try {  const results = await Users.create(payloadData);  res.json({  status: 200,  data: results,  });  } catch (error) {  res.json({  status: 502,  message:{  internal: error.message,  user:"Failed create a new data"  },  errors: error.errors  });  }  }; |

* Fungsi createData diekspor dari modul, sehingga dapat digunakan di tempat lain dalam aplikasi, misalnya dalam routing.
* req.body mengandung data yang dikirimkan oleh client melalui HTTP POST request. Data ini diharapkan berisi field-field yang dibutuhkan untuk membuat record user baru (seperti username, password, dll.). Data tersebut disimpan dalam variabel payloadData.
* Menggunakan Users.create(payloadData), kode ini mencoba memasukkan data baru ke dalam tabel Users database. Fungsi create adalah metode Sequelize yang membuat sebuah entri baru di database berdasarkan model Users.
* await digunakan untuk menunggu proses asynchronous dari Sequelize, yang mengembalikan instance dari model Users yang telah berhasil disimpan dalam database.
* Jika data berhasil disimpan, server mengembalikan respons JSON ke client dengan status 200 yang menandakan bahwa permintaan berhasil dilaksanakan. Objek results yang mengandung data user yang baru dibuat dikirimkan kembali ke client sebagai bagian dari respons.
* Jika terjadi kesalahan saat proses penyimpanan data (misalnya, masalah validasi, kesalahan koneksi database, dll.), blok catch akan menangkap error tersebut.
* Respon JSON dikirimkan dengan status 502 (Bad Gateway), yang sebenarnya kurang tepat untuk kesalahan operasi database; lebih umum menggunakan 500 (Internal Server Error) atau 400 (Bad Request) tergantung jenis kesalahannya.
* Objek message dalam respons mengandung dua bagian: pesan internal yang berisi error.message untuk debugging, dan pesan yang ramah pengguna ("Failed create a new data") yang dapat ditampilkan di UI client.
* error.errors mungkin berisi array detail kesalahan, seperti masalah validasi yang spesifik, jika ada.

Selanjutnya ialah mendaftarkan sebuah end-poin baru untuk melakukan insert data dengan menambahkan function createData pada UserRoutes dengan tipe request POST:

|  |
| --- |
| ...  router.post("/create", userController.createData);  ... |

Terakhir melakukan pengecekan terhadap end-poin yang telah didaftarkan:

A screenshot of a computer

Description automatically generated

1. API Retrive All Data

Buatlah function bernama getAll untuk menampilkan nilai seluruh data pada table Users di UsersController sebagai berikut:

|  |
| --- |
| exports.getAll = async (req, res) => {  try {  const results = await Users.findAll();  res.json({  status: 200,  data: results,  });  } catch (error) {  res.json({  status: 502,  message: {  internal: error.message,  user: "Failed retrive data",  },  errors: error.errors,  });  }  }; |

Users.findAll() adalah metode dari Sequelize yang digunakan untuk mengambil semua record dari tabel Users. Tidak ada parameter yang diberikan, sehingga mengembalikan semua entri dari tabel tersebut.

Tambahkanlah function getAll kedalam file UserRoutes agar dapat diakses menggunakan end-poin tertentu dengan tipe request HTTP ialah GET:

|  |
| --- |
| ...  router.get("/", userController.index);  router.post("/create", userController.createData);  **router.get("/fetch-all", userController.getAll);**  ... |

Selanjutnya melakukan pengetesan terhadap end-poin api tersebut:

A screenshot of a computer

Description automatically generated

1. API Retrive Single Data

Pada contoh kali ini akan menampilkan data single dengan menggunakan passing parameter dan condition pada Sequelize. Disini akan mencontohkan bagaimana cara mengambil sebuah data berdasarkan atribute id pada table Users, dimana atribute id merupakan Primary Key. function findByPK yang dimiliki oleh sequelize dapat dimanfaatkan untuk kondisi tersebut:

|  |
| --- |
| exports.getByID = async (req, res) => {  const user\_id = req.params.id;  try {  const results = await Users.findByPk(user\_id);  res.json({  status: 200,  data: results,  });  } catch (error) {  res.json({  status: 502,  message: {  internal: error.message,  user: "Failed retrive data",  },  errors: error.errors,  });  }  }; |

findByPk (find by primary key) adalah fungsi yang disediakan oleh Sequelize, sebuah library ORM (Object-Relational Mapping) untuk Node.js. Fungsi ini digunakan untuk mengambil satu entri dari database berdasarkan primary key (kunci utama) yang diberikan.

Selanjutnya memerlukan inisialisasi parameter url untuk mengirimkan nilai id pada UserRoutes:

|  |
| --- |
| ...  router.get("/", userController.index);  router.post("/create", userController.createData);  router.get("/fetch-all", userController.getAll);  **router.get("/:id", userController.getByID);**  ... |

"/:id": Ini adalah bagian dari URL endpoint yang menentukan pola URL yang akan ditangani oleh router ini. /:id adalah parameter dinamis yang akan menangkap nilai ID yang diberikan di URL.

A screenshot of a computer

Description automatically generated

1. API Update Data

Selanjutnya membuat sebuah API untuk melakukan update data berdasarkan sebuah kondisi tertentu. Tambahkan function bermanama updateData seperti dibawah ini:

|  |
| --- |
| exports.updatedData = async (req, res) => {  const payloadData = req.body;  try {  const results = await Users.update(payloadData, {  where: {  id: payloadData.id,  },  });  res.json({  status: 200,  data: results,  });  } catch (error) {  res.json({  status: 502,  message: {  internal: error.message,  user: "Failed retrive data",  },  errors: error.errors,  });  }  }; |

* Users.update(payloadData, { where: { id: payloadData.id } }): Ini adalah pemanggilan metode update pada model Users. Metode ini digunakan untuk mengupdate record yang sudah ada di dalam database.
* payloadData: Objek pertama yang diberikan kepada update adalah data yang akan digunakan untuk mengupdate record tersebut. payloadData biasanya berisi pasangan kunci-nilai yang menyatakan kolom dan nilai baru yang ingin Anda update di database.
* where: { id: payloadData.id }: Ini adalah objek yang menentukan kondisi where untuk operasi update. Ini memastikan bahwa hanya record dengan id yang sesuai dengan id yang diberikan dalam payloadData yang akan diupdate. Pada dasarnya, ini mengidentifikasi secara unik record yang perlu diubah berdasarkan primary keynya.

Daftarkan end-poin baru untuk function tersebut pada UserRoutes dengan tipe request HTTP ialah PUT:

|  |
| --- |
| ...  router.get("/", userController.index);  router.post("/create", userController.createData);  router.get("/fetch-all", userController.getAll);  router.get("/:id", userController.getByID);  **router.put("/update", userController.updatedData);**  ... |

router.put() adalah metode pada router Express yang digunakan untuk menangani HTTP PUT requests. PUT biasanya digunakan dalam API untuk mengupdate sumber daya yang sudah ada.

A screenshot of a computer

Description automatically generated

1. API Delete Data

Terakhir ialah API untuk melakukan penghapusan data, buatlah function bernama deleteData pada controller users:

|  |
| --- |
| exports.deleteData = async (req, res) => {  const { id } = req.body;  try {  var message = "";  const delete\_user = await Users.destroy({ where: { id } });  if (delete\_user) {  message = "Successfully removed";  }  res.json({  status: 200,  message: message,  });  } catch (error) {  res.json({  status: 502,  message: {  internal: error.message,  user: "Failed remove data",  },  errors: error.errors,  });  }  }; |

* Users.destroy(): Ini adalah pemanggilan metode destroy() pada model Users. Metode ini digunakan untuk menghapus satu atau lebih record dari database berdasarkan kriteria tertentu.
* { where: { id } }: Ini adalah objek yang menentukan kondisi untuk operasi penghapusan. Dalam hal ini, kita ingin menghapus record yang memiliki id yang sesuai dengan nilai yang diberikan. id adalah variabel yang digunakan di sini, dan diasumsikan bahwa nilainya telah didefinisikan sebelumnya.

Tambahkan route end-poin untuk memfasilitasi transaksi delete tersebut pada UserRoutes dengan tipe request HTTP ialah DELETE:

|  |
| --- |
| ...  router.get("/", userController.index);  router.post("/create", userController.createData);  router.get("/fetch-all", userController.getAll);  router.get("/:id", userController.getByID);  router.put("/update", userController.updatedData);  **router.delete("/delete", userController.deleteData);**  ... |

router.delete() adalah metode pada router Express yang digunakan untuk menangani HTTP DELETE requests. DELETE biasanya digunakan dalam API untuk menghapus sumber daya yang ada.

A screenshot of a web page

Description automatically generated

1. **Tugas Pelatihan**

Silakan anda lanjutkan pembuatan middleware untuk backend aplikasi messeger, dengan menambahkan skema database untuk CHAT:

A screenshot of a chat

Description automatically generated

1. Rancanglah bentuk table-table yang tepat untuk membangun sebuah aplikasi messeger seperti gambar diatas
2. Buatlah table tersebut pada middleware dengan memanfaatkan migration dan Sequelize
3. Buatlah target data apa saja yang perlu disiapkan jika melihat pada gambar aplikasi diatas. Tentukan bentuk request HTTP dari masing-masing API sesuai dengan kebutuhan.
4. Jika telah mengerjakan soal nomor 3, lakukan testing middleware pada aplikasi POSTMAN atau sejenisnya. Masing-masing target data berikan tampilan bagaimana middleware dijalankan atau melakukan request.

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PWL-TI-21-PA-NPM. File source code disimpan sesuai nama project-praktikum dan masukan kedalam repositori tersebut. Buatkanlah file dokumen dalam bentuk file pdf yang berisi Screen Capture dari hasil program yang telah dikerjakan. Simpan dalam file PDF tersebut kedalam project tersebut.

Tambahkan Collaborator management access pada repository ke @*IrvanRizkyAriansyah* dan *@FebryFairuz*