

## README.md

# Modul 5 dan Modul 6 - ExpressJS REST API CRUD Mahasiswa & Networking Frontend Backend

Nama	Ahmad Mu'min Faisal
NIM	1203210011
Kelas	IF-01-02

Repositori Frontend (React): <https://github.com/fzl-22/react-crud-mahasiswa>

Repositori Backend (Express): <https://github.com/fzl-22/express-rest-api-mahasiswa>

## 1 Pendahuluan

Pada praktikum kali ini kita akan belajar membuat CRUD RESTful API Dengan Express.js. Dalam programming, CRUD merupakan singkatan dari Create, Read, Update, dan Delete, yakni aplikasi yang berisi pengolahan data. Restful API ini merupakan backend yang nantinya akan digunakan frontend untuk berkomunikasi dengan database.

Selain itu, RESTful API yang telah dibuat akan dihubungkan dengan web yang telah dibuat pada modul 4 dengan penyesuaian tambahan.

## 2 Persiapan Project

### 2.1 Membuat Project

Untuk membuat project, buat direktori bernama `express-api`.

```
mkdir express-api
```

Kemudian, masuk ke direktori tersebut.

```
cd express-api
```

Di dalam direktori ini, inisiasikan project Node.js.

```
npm init
```

Isikan permintaan sesuai dengan identitas project. Setelah itu, maka project Node.js telah dibuat dengan ditandai oleh adanya 1 file baru, yaitu `package.json`.

Kemudian, buat file bernama `.gitignore` di direktori root project (sejajar dengan `package.json`). Masukkan sebaris string bernama `node_modules/` di file tersebut agar diabaikan oleh Git.

File `.gitignore` :

```
node_modules/
```

## 2.2 Install Dependency

### 2.2.1 express

Untuk menginstall ExpressJS, jalankan perintah berikut:

```
npm install express
```

### 2.2.2 nodemon

Nodemon merupakan package untuk membantu developer untuk lebih produktif. Nodemon dapat melakukan restart otomatis ketika terjadi perubahan kode sehingga developer tidak perlu berulang-ulang menjalankan script. Untuk menginstall Nodemon untuk lingkungan pengembangan, jalankan perintah berikut:

```
npm install --save-dev nodemon
```

Kemudian, buat script baru di `package.json` di objek `scripts`, yaitu `dev` yang menjalankan `nodemon index.js`. Sehingga, menjalankan API dengan Nodemon dapat dilakukan dengan perintah `npm run dev`.

### 2.2.3 mysql2

MySQL2 merupakan client untuk database MySQL di NodeJS. Untuk menginstall package ini, jalankan perintah berikut:

```
npm install mysql2
```

### 2.2.4 body-parser

Package ini berguna untuk melakukan parsing dari request ke dalam `req.body`. Untuk menginstall package ini, jalankan perintah berikut:

```
npm install body-parser
```

### 2.2.5 express-validator

Package ini berguna untuk melakukan validasi request terhadap project Express. Untuk menginstall package ini, jalankan perintah berikut:

```
npm install express-validator
```

### 2.2.6 cors

CORS (Cross Origin Resource Sharing) merupakan teknik yang mengizinkan browser pada suatu domain mendapatkan akses ke server pada sumber yang berbeda. Dengan menggunakan CORS, RESTful API yang saat ini dibuat bisa diakses oleh aplikasi client side, seperti web atau aplikasi mobile.

Untuk menginstall package ini, jalankan perintah berikut:

```
npm install cors
```

### 2.2.7 dotenv

Package dotenv merupakan package yang digunakan untuk menyembunyikan kredensial atau environment variable lainnya, misalnya username dan password database di dalam source code yang akan di git public repository, misalnya Github untuk meningkatkan keamanan repository. Untuk menginstall package ini, jalankan perintah berikut:

```
npm install dotenv
```

Kemudian, buat file bernama `.env`. Setelah itu, isikan beberapa baris berikut ke dalam file `.env`.

```
MYSQL_HOST="localhost"
MYSQL_USER="{username}" # isi dengan username MySQL
MYSQL_PASSWORD="{password}" # isi dengan password user MySQL
MYSQL_DATABASE="db_express_api"
PORT=3001
```

Setelah itu, tambahkan baris baru berisi `.env` di file `.gitignore` agar file ini diabaikan oleh Git.

## 3 Source Code

---

### 3.1 Konfigurasi Koneksi Database

Untuk melakukan konfigurasi database, buat direktori `config`. Kemudian, buat file bernama `database.config.js`.

Di dalam file ini. Import file `.env` dan package `mysql2`. Kemudian, buat object `connection` berdasarkan kredensial yang telah ditulis di `.env`. Kemudian, hubungkan project Express dengan MySQL agar dapat di-export ke module lain. File `config/database.conf.js`:

```
require('dotenv').config();

let mysql = require("mysql2");

let connection = mysql.createConnection({
  host: process.env.MYSQL_HOST,
  user: process.env.MYSQL_USER,
  password: process.env.MYSQL_PASSWORD,
  database: process.env.MYSQL_DATABASE,
});

connection.connect(function (error) {
  if (error) {
    console.log(error);
    return;
  }

  console.log("[CONNECTED] Connection established...");
});

module.exports = connection;
```

### 3.2 Konfigurasi Database MySQL

Pastikan service MySQL telah menyala. Jika belum, jalankan perintah `sudo systemctl start mysql.service`. Login ke user `root`. Buat sebuah database bernama `db_express_api`

```
CREATE DATABASE db_express_api;
```

Kemudin, berikan akses penuh database ini ke user reguler:

```
GRANT ALL PRIVILEGES ON db_express_api.* TO {username}@localhost;
```

Keluar dari akun `root`. Kemudian, masuk ke user regular dan akses databasenya.

```
USE db_express_api
```

Setelah itu, buat tabel seperti yang dispesifikan pada modul dengan cara menjalankan query berikut:

```
CREATE TABLE mahasiswa (  
  nim VARCHAR(255) PRIMARY KEY,  
  nama VARCHAR(255),  
  jurusan VARCHAR(255),  
  asal_provinsi VARCHAR(255)  
);
```

Dengan ini, database di MySQL telah dikonfigurasi.

### 3.3 Membuat Router

Buat direktori bernama `routes`, kemudian buat file bernama `mahasiswa.route.js` di dalamnya untuk melakukan konfigurasi route dari RESTful API.

Buat objek `router` dari class `express.Router()`. Kemudian, buat masing-masing route di tabel di bawah ini. Callback function yang dipanggil di setiap route telah di-refactor ke `controllers/mahasiswa.controller.js`. Terakhir, object `router` di-export ke module lain.

Method	Path	Kegunaan
GET	/	<code>getAllMahasiswa</code> , mendapatkan data semua mahasiswa

Method	Path	Kegunaan
POST	/store	postSingleMahasiswa , menyimpan data satu mahasiswa
GET	/:id	getSingleMahasiswa , mendapatkan data satu mahasiswa berdasarkan parameter id (nim)
PATCH	/update/:id	updateSingleMahasiswa , mengupdate data satu mahasiswa berdasarkan id (nim)
DELETE	/delete/:id	deleteSingleMahasiswa , menghapus data satu mahasiswa berdasarkan id (nim)

File routes/mahasiswa.route.js :

```
const express = require("express");

const mahasiswaControllers = require("../controllers/mahasiswa.controller")

const router = express.Router();

const { body } = require("express-validator");

router.get("/", mahasiswaControllers.getAllMahasiswa);

router.post(
  "/store",
  [
    body("nim").notEmpty(),
    body("nama").notEmpty(),
    body("jurusan").notEmpty(),
    body("asal_provinsi").notEmpty(),
  ],
  mahasiswaControllers.postSingleMahasiswa
);

router.get("/:id", mahasiswaControllers.getSingleMahasiswa);

router.patch(
  "/update/:id",
  [
    body("nama").notEmpty(),
    body("jurusan").notEmpty(),
    body("asal_provinsi").notEmpty(),
  ],
  mahasiswaControllers.updateSingleMahasiswa
);

router.delete("/delete/:id", mahasiswaControllers.deleteSingleMahasiswa);
```

```
module.exports = router;
```

### 3.4 Membuat Controller

Buat direktori bernama `controllers`, kemudian buat file bernama `mahasiswa.controller.js` di dalamnya, File ini berisi fungsi-fungsi yang dipanggil oleh setiap route di `routes/mahasiswa.route.js`.

Pertama-tama, panggil objek `connection` dari `config/database.conf.js` dan `validationResult` dari `express-validator`:

```
const connection = require("../config/database.conf");
const { validationResult } = require("express-validator");
```

Kemudian, buat masing-masing fungsi untuk menangani setiap route.

#### 3.4.1 getAllMahasiswa

Fungsi `getAllMahasiswa` digunakan untuk mendapatkan data semua mahasiswa melalui path `/`. Fungsi ini melakukan query untuk mengambil semua data mahasiswa yang diurutkan berdasarkan `nim`. Jika gagal, route akan mengirimkan status code 500 dan pesan error. Jika berhasil, route akan mengirimkan status code 200 beserta data semua mahasiswa dalam bentuk JSON.

```
function getAllMahasiswa(req, res) {
  connection.query(
    "SELECT * FROM mahasiswa ORDER BY nim DESC",
    function (err, rows) {
      if (err) {
        return res.status(500).json({
          status: false,
          message: "Internal Server Error",
        });
      }

      return res.status(200).json({
        status: true,
        message: "List Data Mahasiswa",
        data: rows,
      });
    }
  );
}
```

### 3.4.2 postSingleMahasiswa

Fungsi `postSingleMahasiswa` digunakan untuk menambahkan data satu mahasiswa melalui path `/store`. Fungsi ini menerima data `nim`, `nama`, `jurusan`, dan `asal_provinsi` dari query parameter serta memvalidasinya. Kemudian, apabila validasi berhasil, maka akan menambahkan data satu mahasiswa ke database dan status code 201. Apabila gagal menambahkan ke database, maka akan mengirimkan status code 500.

```
function postSingleMahasiswa(req, res) {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(422).json({
      errors: errors.array(),
    });
  }

  // define formData
  let formData = {
    nim: req.body.nim,
    nama: req.body.nama,
    jurusan: req.body.jurusan,
    asal_provinsi: req.body.asal_provinsi,
  };

  // insert query
  connection.query(
    "INSERT INTO mahasiswa SET ?",
    formData,
    function (err, rows) {
      if (err) {
        return res.status(500).json({
          status: false,
          message: "Internal Server Error",
        });
      }

      return res.status(201).json({
        status: true,
        message: "Insert Data Successfully",
        data: rows[0],
      });
    }
  );
}
```

### 3.4.3 getSingleMahasiswa



Fungsi `getSingleMahasiswa` digunakan untuk mendapatkan data satu mahasiswa berdasarkan `nim` melalui path `/:id`. Fungsi ini mengambil data satu mahasiswa dari database, kemudian mengirimkannya ke client dalam bentuk JSON.

```
function getSingleMahasiswa(req, res) {
  let id = req.params.id;

  connection.query(`SELECT * FROM mahasiswa WHERE nim = ${id}`, (err, rows)
  // if error
  if (err) {
    return res.status(500).json({
      status: false,
      message: "Internal Server Error",
    });
  }

  if (rows.length <= 0) {
    return res.status(404).json({
      status: false,
      message: "Data Mahasiswa Not Found!",
    });
  }

  return res.status(200).json({
    status: true,
    message: "Detail Data Mahasiswa",
    data: rows[0],
  });
}
```

### 3.4.4 updateSingleMahasiswa

Fungsi `updateSingleMahasiswa` digunakan untuk memperbarui data mahasiswa berdasarkan `nim` melalui path `/update/:id`. Fungsi ini menerima request body data mahasiswa (kecuali `nim`) dari path, kemudian melakukan validasi. Apabila validasi berhasil, maka data mahasiswa akan diupdate berdasarkan `nim`-nya dan status code 200. Apabila gagal, maka akan mengirimkan status code 500.

```
function updateSingleMahasiswa(req, res) {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(422).json({
      errors: errors.array(),
    });
  }
}
```

```
let id = req.params.id;

let formData = {
  nama: req.body.nama,
  jurusan: req.body.jurusan,
  asal_provinsi: req.body.asal_provinsi,
};

connection.query(
  `UPDATE mahasiswa SET ? WHERE nim = ${id}`,
  formData,
  (err, rows) => {
    if (err) {
      return res.status(500).json({
        status: false,
        message: "Internal Server Error",
      });
    }

    return res.status(200).json({
      status: true,
      message: "Update Data Successfully!",
    });
  }
);
}
```

### 3.4.5 deleteSingleMahasiswa

Fungsi `deleteSingleMahasiswa` digunakan untuk menghapus data mahasiswa berdasarkan `nim` melalui path `/delete/:id`. Fungsinya ini akan menghapus data dari database berdasarkan `id` (`nim`). Apabila gagal, maka akan mengirimkan status code 500.

```
function deleteSingleMahasiswa(req, res) {
  let id = req.params.id;

  connection.query(`DELETE FROM mahasiswa WHERE nim = ${id}`, (err, rows) :
    if (err) {
      return res.status(500).json({
        status: false,
        message: "Internal Server Error",
      });
    }

    return res.status(200).json({
      status: true,
      message: "Delete Data Successfully!",
    });
  });
}
```

```
});  
}
```

### 3.4.6 Export Modul

Setelah semua fungsi dibuat, maka semua fungsi tersebut perlu di-export agar bisa digunakan oleh modul lainnya. Tambahkan kode berikut di akhir file:

```
module.exports = {  
  getAllMahasiswa: getAllMahasiswa,  
  postSingleMahasiswa: postSingleMahasiswa,  
  getSingleMahasiswa: getSingleMahasiswa,  
  updateSingleMahasiswa: updateSingleMahasiswa,  
  deleteSingleMahasiswa: deleteSingleMahasiswa,  
};
```

## 3.5 Membuat Web Server

Buat file `index.js` di root direktori project, kemudian isikan kode program di bawah ini. Kode ini adalah entry point dari project ExpressJS. Pertama-tama import `dotenv`, `express`, `body-parser`, `cors`, dan `mahasiswaRouter`. Kemudian, buat server Express di port 3001. Setelah itu, gunakan middleware `cors` dan `bodyParser`. Kemudian, gunakan middleware router `mahasiswaRouter` dengan prefix `/api/mahasiswa`. Hal ini membuat path yang didefinisikan di file `routes/mahasiswa.route.js` memiliki prefix tersebut. Misalnya, path `/update/:id` akan menjadi `/api/mahasiswa/update/:id`.

File `index.js`:

```
require('dotenv').config();  
  
const express = require("express");  
const bodyParser = require("body-parser");  
const cors = require("cors");  
const mahasiswaRouter = require("./routes/mahasiswa.route");  
  
// server configuration  
const app = express();  
const port = process.env.PORT;  
  
// use CORS  
app.use(cors());  
  
// parse application/x-www-form-urlencoded  
app.use(bodyParser.urlencoded({ extended: false }));
```

```
// parse application/json
app.use(bodyParser.json());

// use Router middleware
app.use('/api/mahasiswa', mahasiswaRouter);

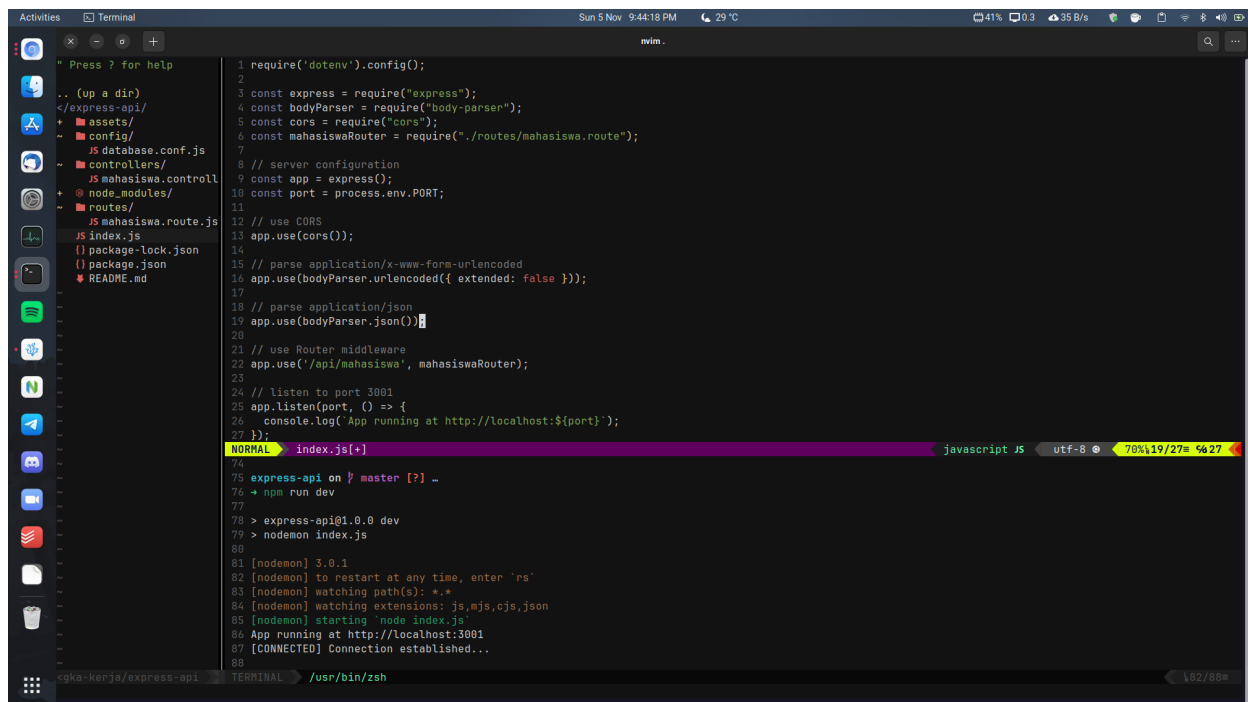
// listen to port 3001
app.listen(port, () => {
  console.log(`App running at http://localhost:${port}`);
});
```

## 3.6 Jalankan Web Server ExpressJS

Jalankan web server ExpressJS dengan perintah berikut:

```
npm run dev
```

Maka API akan berjalan di port 3001, output dari terminal dapat dilihat pada gambar di bawah ini.



```
1 require('dotenv').config();
2
3 const express = require("express");
4 const bodyParser = require("body-parser");
5 const cors = require("cors");
6 const mahasiswaRouter = require("./routes/mahasiswa.route");
7
8 // server configuration
9 const app = express();
10 const port = process.env.PORT;
11
12 // use CORS
13 app.use(cors());
14
15 // parse application/x-www-form-urlencoded
16 app.use(bodyParser.urlencoded({ extended: false }));
17
18 // parse application/json
19 app.use(bodyParser.json());
20
21 // use Router middleware
22 app.use('/api/mahasiswa', mahasiswaRouter);
23
24 // listen to port 3001
25 app.listen(port, () => {
26   console.log(`App running at http://localhost:${port}`);
27 });
28
29 NORMAL index.js[+]
30
31 express-api on master [?] -
32 + npm run dev
33
34 > express-api@1.0.0 dev
35 > nodemon index.js
36
37 [nodemon] 3.0.1
38 [nodemon] to restart at any time, enter `rs`
39 [nodemon] watching path(s): *.*
40 [nodemon] watching extensions: js,mjs,cjs,json
41 [nodemon] starting `node index.js`
42 App running at http://localhost:3001
43 [CONNECTED] Connection established...
```

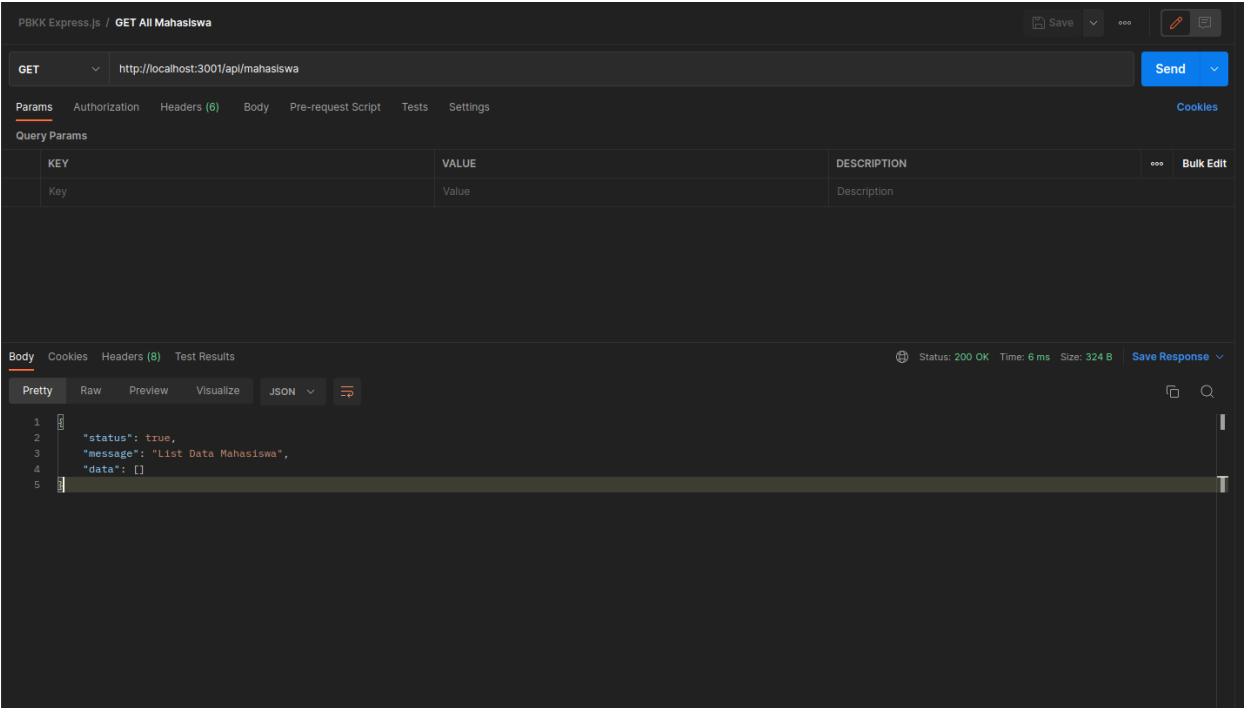
## 4 Pengujian REST API

Pertama-tama buka aplikasi Postman. Kemudian, buat collection baru bernama PBKK Express.js . Kemudian, buat request-request berikut:

### 4.1 GET All Mahasiswa

Key	Value
Method	GET
Endpoint	<a href="http://localhost:3001/api/mahasiswa">http://localhost:3001/api/mahasiswa</a>

Output:



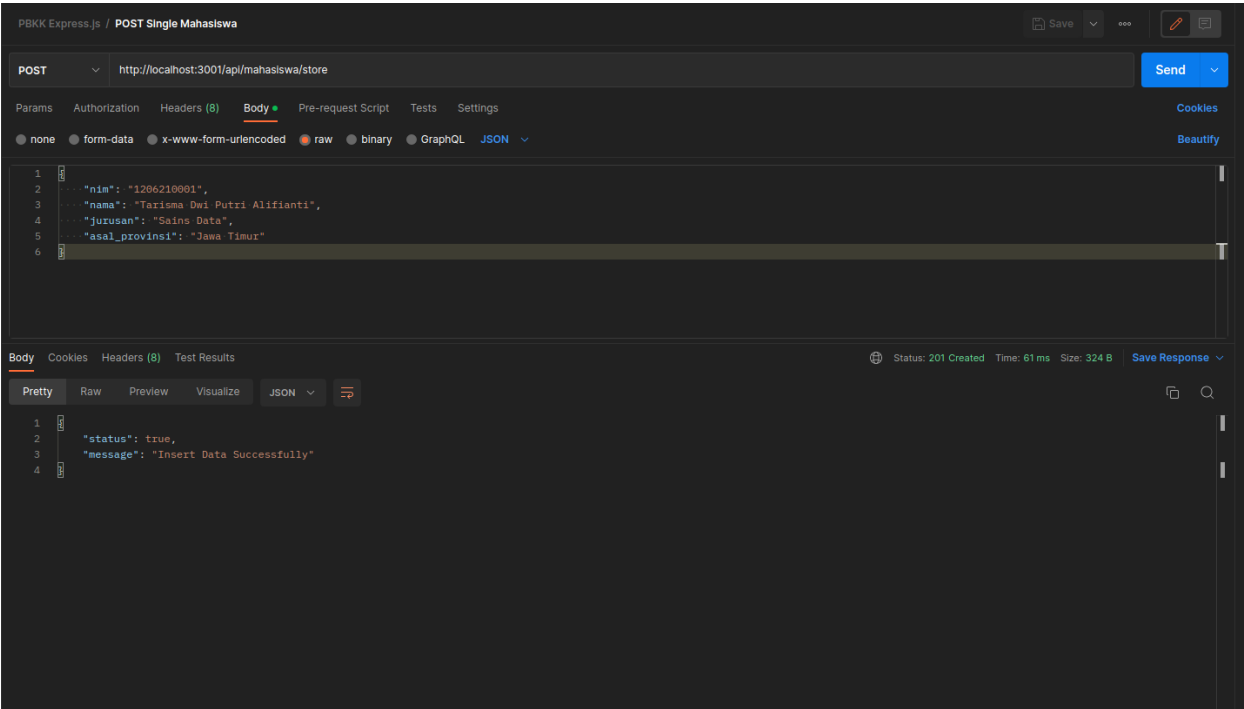
## 4.2 POST Single Mahasiswa

Key	Value
Method	POST
Endpoint	<a href="http://localhost:3001/api/mahasiswa/store">http://localhost:3001/api/mahasiswa/store</a>

Request Body:

```
{
  "nim": "1206210001",
  "nama": "Tarisma Dwi Putri Alifianti",
  "jurusan": "Sains Data",
  "asal_provinsi": "Jawa Timur"
}
```

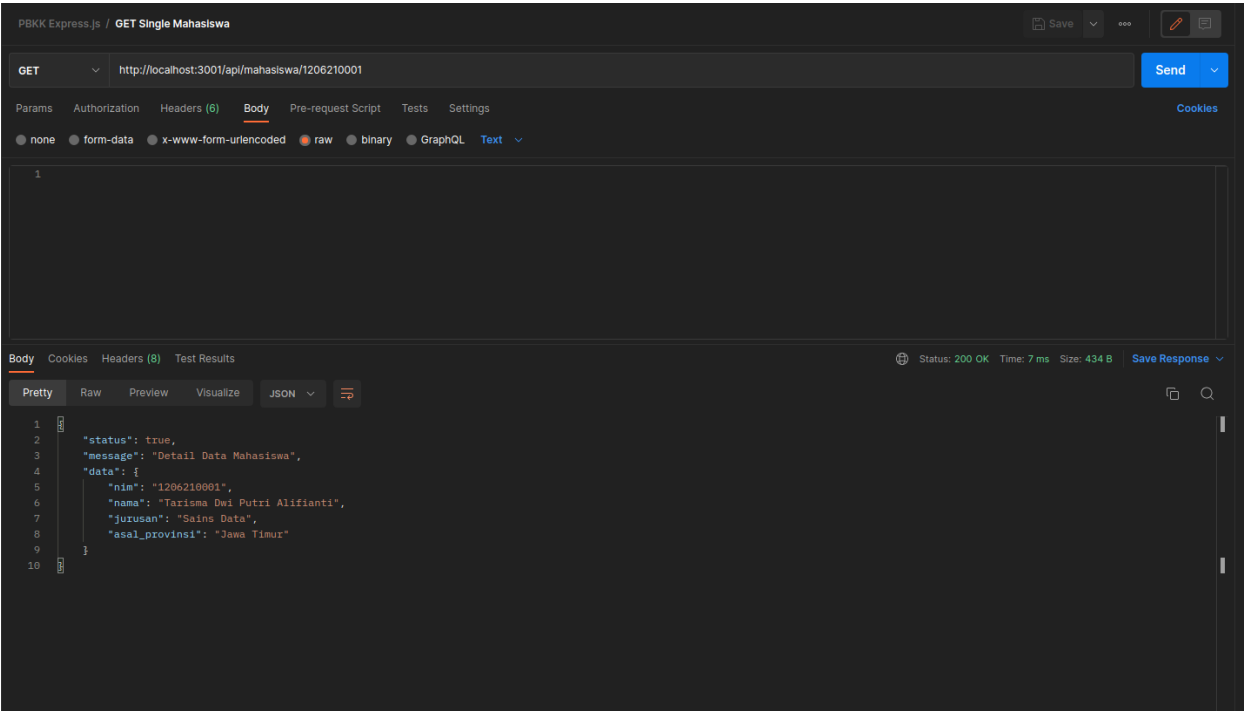
Output:



### 4.3 GET Single Mahasiswa

Key	Value
Method	GET
Endpoint	<a href="http://localhost:3001/api/mahasiswa/1206210001">http://localhost:3001/api/mahasiswa/1206210001</a>

Output:



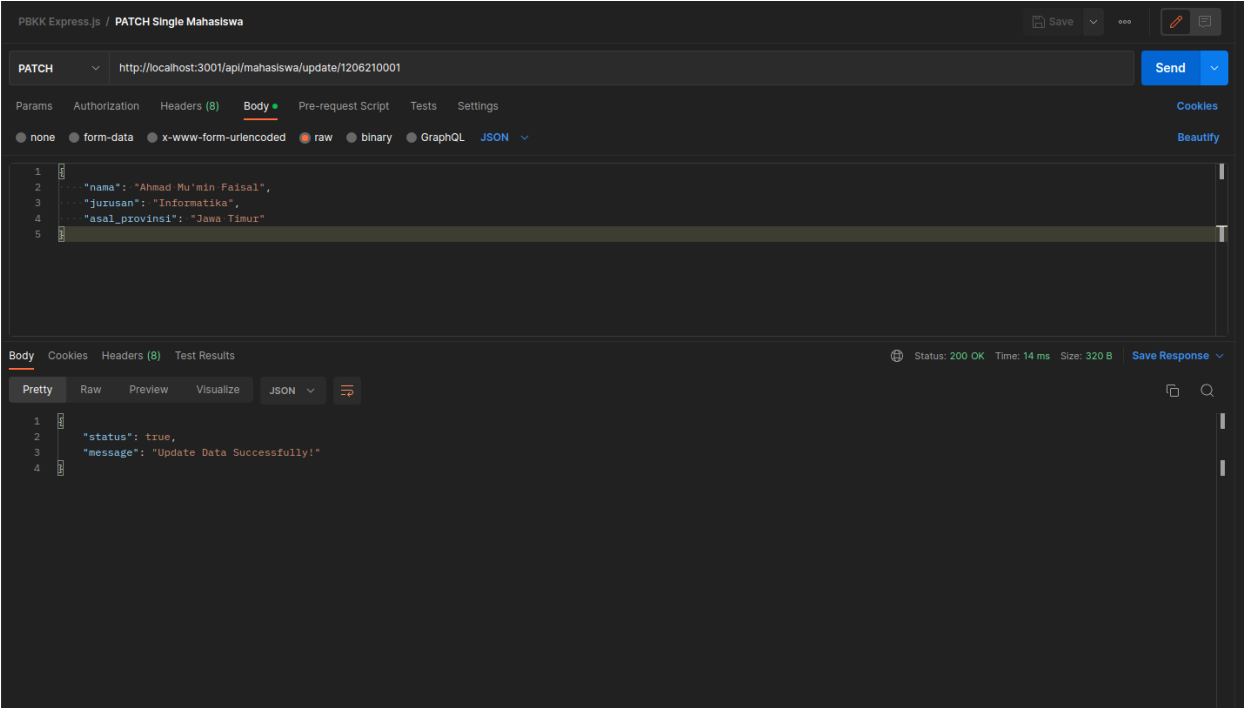
### 4.4 PATCH Single Mahasiswa

Key	Value
Method	PATCH
Endpoint	<a href="http://localhost:3001/api/mahasiswa/update/1206210001">http://localhost:3001/api/mahasiswa/update/1206210001</a>

Request Body:

```
{
  "nama": "Ahmad Mu'min Faisal",
  "jurusan": "Informatika",
  "asal_provinsi": "Jawa Timur"
}
```

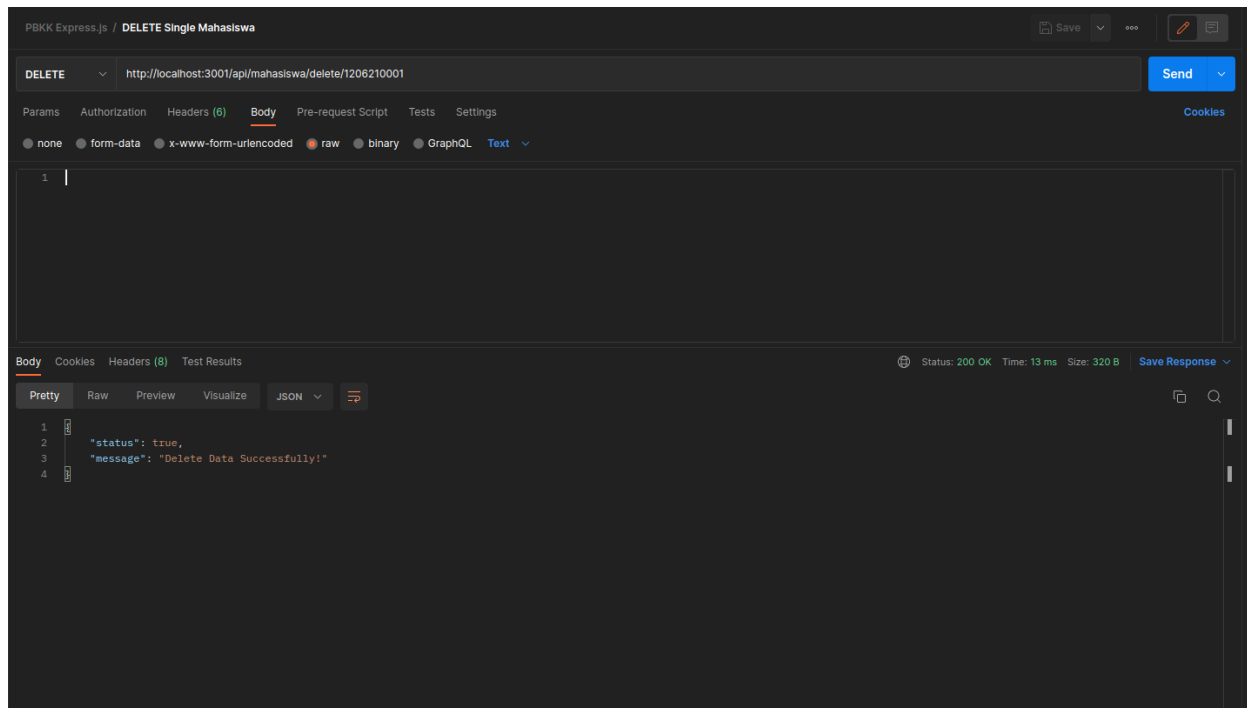
Output:



### 4.5 DELETE Single Mahasiswa

Key	Value
Method	DELETE
Endpoint	<a href="http://localhost:3001/api/mahasiswa/delete/1206210001">http://localhost:3001/api/mahasiswa/delete/1206210001</a>

Output:



## 5 Networking Front-end & Back-end

Repositori yang akan digunakan sebagai frontend adalah repositori CRUD mahasiswa pada modul 4, yaitu di <https://github.com/fzl-22/react-crud-mahasiswa>. Namun, kode pada repositori tersebut perlu disesuaikan lagi.

### 5.1 Install Axios

Axios merupakan package HTTP client untuk Node.js. Untuk menginstall package ini, jalankan perintah berikut:

```
npm install axios
```

Kemudian, import di baris pertama `src/App.js`

```
import axios from "axios";
```

### 5.2 Refactor Kode

Karena nama key dari salah request body di backend adalah `asal_provinsi`, ganti semua nama variabel / properti di frontend dari `asalProvinsi` menjadi `asal_provinsi` untuk mempertahankan konsistensi.

### 5.3 Memodifikasi Handler Function



Fungsi handler dari setiap operasi CRUD yang ada di `src/App.js` perlu dimodifikasi untuk menyesuaikan dengan HTTP request, sehingga semua fungsi akan menjadi asynchronous function.

### 5.3.1 getList

Fungsi ini digunakan untuk mendapatkan semua data mahasiswa dan melakukan rendering ulang.

```
const getList = async () => {
  try {
    const response = await axios.get("http://localhost:3001/api/mahasiswa")
    setMahasiswas(response.data.data);
  } catch (error) {
    console.log(error);
  }
};
```

Kemudian, fungsi ini juga perlu diakses pertama kali ketika komponen di-render, tambahkan kode berikut di bawahnya:

```
useEffect(() => {
  getList();
}, []);
```

### 5.3.2 handleTambahMahasiswa

Fungsi ini digunakan untuk menangani penambahan data mahasiswa:

```
const handleTambahMahasiswa = async (data) => {
  try {
    const response = await axios.post('http://localhost:3001/api/mahasiswa', data)

    if(!response.data.status){
      return;
    }

    getList();
  } catch (error) {
    console.log(error);
  }
};
```

### 5.3.3 handleEditMahasiswa

Fungsi ini digunakan untuk menangani perbaruan data mahasiswa:

```
const handleEditMahasiswa = async (data) => {  
  try {  
    const response = await axios.patch('http://localhost:3001/api/mahasiswa/  
  
    if(!response.data.status){  
      return;  
    }  
  
    getList();  
  } catch (error) {  
    console.log(error);  
  }  
};
```

### 5.3.4 handleDeleteMahasiswa

Fungsi ini digunakan untuk menangani penghapusan data mahasiswa:

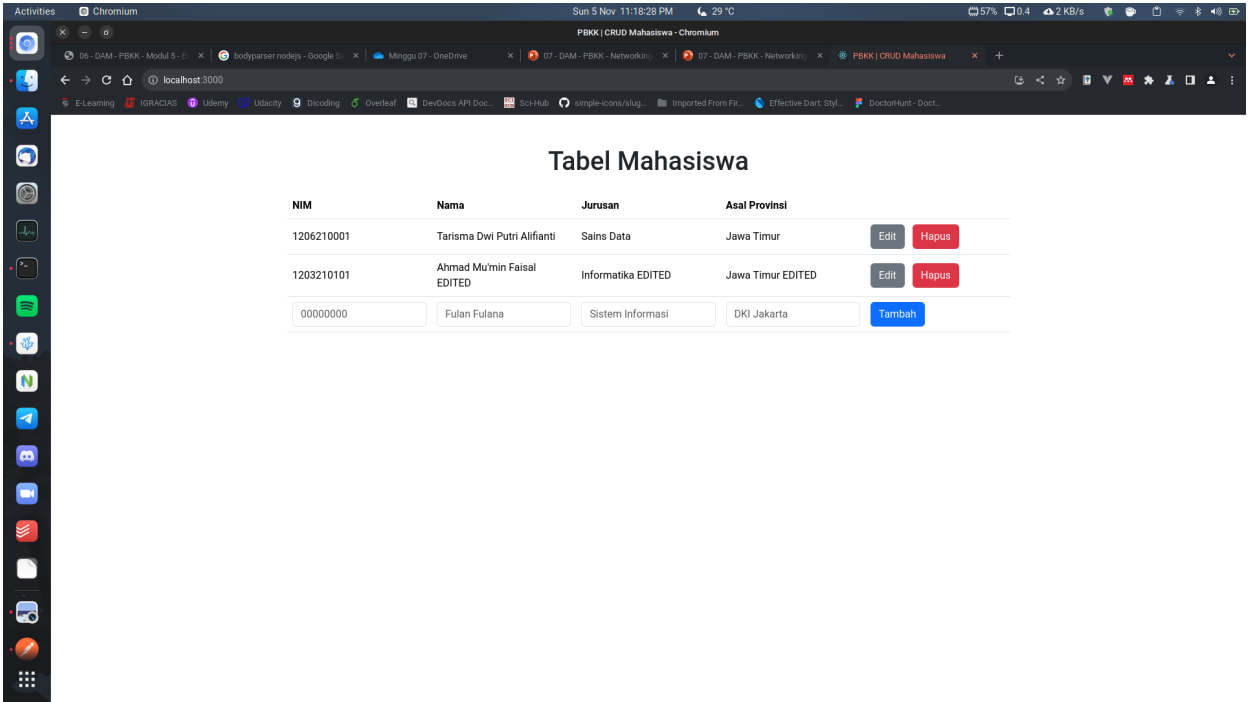
```
const handleHapusMahasiswa = async (e) => {  
  try {  
    const response = await axios.delete('http://localhost:3001/api/mahasiswa/  
  
    if(!response.data.status){  
      return;  
    }  
  
    getList();  
  } catch (error) {  
    console.log(error);  
  }  
};
```

## 5.4 Menjalankan Project React.js

Untuk menjalankan project React.js ini, jalankan perintah berikut:

```
npm run start
```

Kemudian, buka browser di `http://localhost:3000` , maka web akan ditampilkan. Ujicoba API pada web yang telah berjalan.



Dengan ini, frontend (react.js) dan backend (express.js) telah berhasil terhubung.