

| | | |
|-------------|---|----------------------|
| NAMA | : | ALDI SETIAWAN |
| NIM | : | 20220040054 |

ANALISA CODE

1. RANCANGAN API ENDPOINT

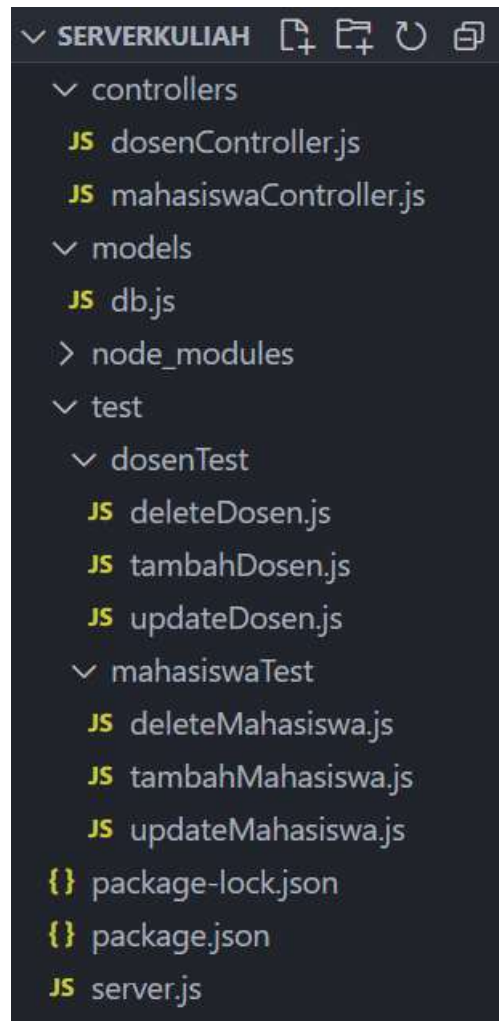
Berikut adalah contoh rancangan endpoint untuk API yang berhubungan dengan database kuliah, terutama untuk tabel mahasiswa dan dosen. API ini akan menyediakan operasi dasar seperti menampilkan semua mahasiswa atau dosen, menampilkan detail mahasiswa atau dosen, menambahkan mahasiswa baru atau dosen baru, mengupdate data mahasiswa atau dosen, dan menghapus mahasiswa atau dosen.

| Metode | Endpoint | Keterangan |
|-------------|-------------------------|--|
| GET | /mahasiswa | <ul style="list-style-type: none"> • Deskripsi: Mendapatkan daftar semua mahasiswa. • Response: Daftar semua mahasiswa dalam format JSON. |
| GET | /mahasiswa/{nim} | <ul style="list-style-type: none"> • Deskripsi: Mendapatkan detail mahasiswa berdasarkan nim. • Parameter: nim (nim mahasiswa). • Response: Detail mahasiswa dalam format JSON. |
| POST | /mahasiswa | <ul style="list-style-type: none"> • Deskripsi: Menambahkan mahasiswa baru. • Request Body: Data mahasiswa yang ingin ditambahkan dalam format JSON. • Response: Konfirmasi sukses atau gagal. |
| PUT | /mahasiswa/{nim} | <ul style="list-style-type: none"> • Deskripsi: Mengupdate data mahasiswa berdasarkan nim. • Parameter: nim (nim mahasiswa). • Request Body: Data mahasiswa yang ingin diupdate dalam format JSON. • Response: Konfirmasi sukses atau gagal. |

| | | |
|---------------|-------------------------|---|
| DELETE | /mahasiswa/{nim} | <ul style="list-style-type: none"> • Deskripsi: Menghapus mahasiswa berdasarkan nim. • Parameter: nim (nim mahasiswa). • Response: Pesan konfirmasi bahwa mahasiswa telah dihapus. |
| GET | /dosen | <ul style="list-style-type: none"> • Deskripsi: Mendapatkan daftar semua dosen. • Response: Daftar semua dosen dalam format JSON. |
| GET | /dosen/{nidn} | <ul style="list-style-type: none"> • Deskripsi: Mendapatkan detail dosen berdasarkan nidn. • Parameter: nidn (nidn dosen). • Response: Detail dosen dalam format JSON. |
| POST | /dosen | <ul style="list-style-type: none"> • Deskripsi: Menambahkan dosen baru. • Request Body: Data dosen yang ingin ditambahkan dalam format JSON. • Response: Konfirmasi sukses atau gagal. |
| PUT | /dosen/{dosen} | <ul style="list-style-type: none"> • Deskripsi: Mengupdate data dosen berdasarkan nidn. • Parameter: nidn (nidn dosen). • Request Body: Data dosen yang ingin diupdate dalam format JSON. • Response: Konfirmasi sukses atau gagal. |
| DELETE | /dosen/{nidn} | <ul style="list-style-type: none"> • Deskripsi: Menghapus dosen berdasarkan nidn. • Parameter: nidn (nidn dosen). • Response: Pesan konfirmasi bahwa dosen telah dihapus. |

2. ARSITEKTUR PROGRAM YANG TELAH DIBUAT

Berikut merupakan arsitektur program yang telah dibuat.



3. PENJELASAN SETIAP PROGRAM

1. Directory controllers

Merupakan sebuah directory untuk menampung setiap controller

a. dosenController.js

```
1 const express = require('express');
2 const router = express.Router();
3 const db = require('../models/db');
4
5 // GET /dosen
6 router.get('/', (req, res) => {
7   db.query('SELECT * FROM dosen', (error, results) => {
8     if (error) {
9       console.error('Error fetching dosen', error);
10      res.status(500).json({ message: 'Internal Server Error' });
11    } else {
12      res.json(results);
13    }
14  });
15 });
16
17 // GET /dosen/:id
18 router.get('/:id', (req, res) => {
19   const dosenId = req.params.id;
20   db.query('SELECT * FROM dosen WHERE id = ?', [dosenId], (error, results) => {
21     if (error) {
22       console.error('Error fetching dosen', error);
23       res.status(500).json({ message: 'Internal Server Error' });
24     } else if (results.length === 0) {
25       res.status(404).json({ message: 'Dosen not found' });
26     } else {
27       res.json(results[0]);
28     }
29   });
30 });
31
32 // PUT /dosen/:id
33 router.put('/:id', (req, res) => {
34   const dosenId = req.params.id;
35   const { nama, gender, prodi, matkul, alamat } = req.body;
36   const updateQuery = 'UPDATE dosen SET nama = ?, gender = ?, prodi = ?, matkul = ?, alamat = ? WHERE id = ?';
37
38   console.log('Received PUT request with data:', { nama, gender, prodi, matkul, alamat, dosenId });
39
40   db.query(updateQuery, [nama, gender, prodi, alamat, matkul, dosenId], (error) => {
41     if (error) {
42       console.error('Error updating dosen', error);
43       res.status(500).json({ message: 'Internal Server Error' });
44     } else {
45       res.json({ success: true, message: 'Data dosen berhasil diperbarui' });
46     }
47   });
48 });
49
50 // POST /dosen
51 router.post('/', (req, res) => {
52   const { id, nama, gender, prodi, matkul, alamat } = req.body;
53   const insertQuery = 'INSERT INTO dosen (id, nama, gender, prodi, matkul, alamat) VALUES (?, ?, ?, ?, ?, ?)';
54
55   db.query(insertQuery, [id, nama, gender, prodi, matkul, alamat], (error) => {
56     if (error) {
57       console.error('Error creating dosen', error);
58       res.status(500).json({ message: 'Internal Server Error' });
59     } else {
60       res.json({ success: true, message: 'Dosen created successfully' });
61     }
62   });
63 });
64
65 // DELETE /dosen/:id
66 router.delete('/:id', (req, res) => {
67   const dosenId = req.params.id;
68   const deleteQuery = 'DELETE FROM dosen WHERE id = ?';
69
70   db.query(deleteQuery, [dosenId], (error) => {
71     if (error) {
72       console.error('Error deleting dosen', error);
73       res.status(500).json({ message: 'Internal Server Error' });
74     } else {
75       res.json({ success: true, message: 'Dosen deleted successfully' });
76     }
77   });
78 });
79
80 module.exports = router;
```

Program di atas adalah bagian dari server aplikasi menggunakan framework Express.js untuk mengelola data dosen. Berikut adalah penjelasan singkat dari setiap bagian program:

- 1) Dependencies:
 - a) express: Framework Node.js untuk membangun aplikasi web.
 - b) router: Modul Express untuk membuat router.
 - c) db: Modul untuk berinteraksi dengan database (asumsi terdapat di direktori ../models/db).
- 2) GET /dosen:
 - a) Mengambil semua data dosen dari database.
 - b) Menggunakan method db.query untuk menjalankan query SQL.
 - c) Mengirimkan hasil sebagai respons JSON.
- 3) GET /dosen/:nidn:
 - a) Mengambil data dosen berdasarkan NIDN (Nomor Induk Dosen).
 - b) Menggunakan parameter :nidn dari URL.
 - c) Mengirimkan hasil sebagai respons JSON.
 - d) Menangani kasus jika dosen tidak ditemukan.
- 4) PUT /dosen/:nidn:
 - a) Mengupdate data dosen berdasarkan NIDN.
 - b) Menggunakan method db.query untuk menjalankan query SQL UPDATE.
 - c) Mengambil data dari body request.
 - d) Mengirimkan respons JSON setelah berhasil atau gagal.
- 5) POST /dosen:
 - a) Membuat data dosen baru.
 - b) Menggunakan method db.query untuk menjalankan query SQL INSERT.
 - c) Mengambil data dari body request.
 - d) Mengirimkan respons JSON setelah berhasil atau gagal.
- 6) DELETE /dosen/:nidn:
 - a) Menghapus data dosen berdasarkan NIDN.
 - b) Menggunakan method db.query untuk menjalankan query SQL DELETE.
 - c) Mengirimkan respons JSON setelah berhasil atau gagal.

Setiap operasi di atas juga menangani kasus kesalahan, memberikan respons status HTTP 500 (Internal Server Error) jika terjadi masalah dalam eksekusi query.

b. mahasiswaController

```
1 const express = require('express');
2 const router = express.Router();
3 const db = require('../models/db');
4
5 //GET /mahasiswa
6 router.get('/', (req, res) => {
7   db.query('SELECT * FROM mahasiswa', (error, results) => {
8     if (error) {
9       console.error('Error fetching mahasiswa', error);
10      res.status(500).json({ message: 'Internal Server Error' });
11    } else {
12      res.json(results);
13    }
14  });
15 });
16
17 //GET /mahasiswa/:nim
18 router.get('/:nim', (req, res) => {
19   const mahasiswaId = req.params.nim;
20   db.query('SELECT * FROM mahasiswa WHERE nim = ?', [mahasiswaId], (error, results) => {
21     if (error) {
22       console.error('Error fetching mahasiswa:', error);
23       res.status(500).json({ message: 'Internal not found' });
24     } else if (results.length === 0) {
25       res.status(404).json({ message: 'Mahasiswa not found' });
26     } else {
27       res.json(results[0]);
28     }
29   });
30 });
31
32 //PUT /mahasiswa/:nim
33 router.put('/:nim', (req, res) => {
34   const mahasiswaId = req.params.nim;
35   const { nama, gender, prodi, alamat } = req.body;
36   const updateQuery = 'UPDATE mahasiswa SET nama = ?, gender = ?, prodi = ?, alamat = ? WHERE nim = ?';
37
38   console.log('Received PUT request with data:', { nama, gender, prodi, alamat, mahasiswaId });
39
40   db.query(updateQuery, [nama, gender, prodi, alamat, mahasiswaId], (error) => {
41     if (error) {
42       console.error('Error updating mahasiswa:', error);
43       res.status(500).json({ message: 'Internal Server Error' });
44     } else {
45       res.json({ success: true, message: 'Data mahasiswa berhasil diperbarui' });
46     }
47   });
48 });
49
50 // POST /mahasiswa
51 router.post('/', (req, res) => {
52   const { nim, nama, gender, prodi, alamat } = req.body;
53   const insertQuery = 'INSERT INTO mahasiswa (nim, nama, gender, prodi, alamat) VALUES (?, ?, ?, ?, ?)';
54
55   db.query(insertQuery, [nim, nama, gender, prodi, alamat], (error) => {
56     if (error) {
57       console.error('Error creating mahasiswa:', error);
58       res.status(500).json({ message: 'Internal Server Error' });
59     } else {
60       res.json({ success: true, message: 'Mahasiswa created successfully' });
61     }
62   });
63 });
64
65 // DELETE /mahasiswa/:nim
66 router.delete('/:nim', (req, res) => {
67   const mahasiswaId = req.params.nim;
68   const deleteQuery = 'DELETE FROM mahasiswa WHERE nim = ?';
69
70   db.query(deleteQuery, [mahasiswaId], (error) => {
71     if (error) {
72       console.error('Error deleting mahasiswa:', error);
73       res.status(500).json({ message: 'Internal Server Error' });
74     } else {
75       res.json({ success: true, message: 'Mahasiswa deleted successfully' });
76     }
77   });
78 });
79
80 module.exports = router;
```

Program di atas adalah bagian dari server aplikasi menggunakan framework Express.js untuk mengelola data mahasiswa. Berikut adalah penjelasan singkat dari setiap bagian program:

- 1) Dependencies:
 - a) express: Framework Node.js untuk membangun aplikasi web.
 - b) router: Modul Express untuk membuat router.
 - c) db: Modul untuk berinteraksi dengan database (asumsi terdapat di direktori ../models/db).
- 2) GET /mahasiswa:
 - a) Mengambil semua data mahasiswa dari database.
 - b) Menggunakan method db.query untuk menjalankan query SQL.
 - c) Mengirimkan hasil sebagai respons JSON.
- 3) GET /mahasiswa/:nim:
 - a) Mengambil data mahasiswa berdasarkan NIM (Nomor Induk Mahasiswa).
 - b) Menggunakan parameter :nim dari URL.
 - c) Mengirimkan hasil sebagai respons JSON.
 - d) Menangani kasus jika mahasiswa tidak ditemukan.
- 4) PUT /mahasiswa/:nim:
 - a) Mengupdate data mahasiswa berdasarkan NIM.
 - b) Menggunakan method db.query untuk menjalankan query SQL UPDATE.
 - c) Mengambil data dari body request.
 - d) Mengirimkan respons JSON setelah berhasil atau gagal.
- 5) POST /mahasiswa:
 - a) Membuat data mahasiswa baru.
 - b) Menggunakan method db.query untuk menjalankan query SQL INSERT.
 - c) Mengambil data dari body request.
 - d) Mengirimkan respons JSON setelah berhasil atau gagal.
- 6) DELETE /mahasiswa/:nim:
 - a) Menghapus data mahasiswa berdasarkan NIM.
 - b) Menggunakan method db.query untuk menjalankan query SQL DELETE.
 - c) Mengirimkan respons JSON setelah berhasil atau gagal.

Setiap operasi di atas juga menangani kasus kesalahan, memberikan respons status HTTP 500 (Internal Server Error) jika terjadi masalah dalam eksekusi query.

2. Directory models

a. db.js



Program di atas adalah modul Node.js untuk membuat koneksi ke database MySQL menggunakan modul mysql. Berikut penjelasan singkatnya:

- 1) Dependencies:
 - a) mysql: Modul untuk berinteraksi dengan database MySQL.
- 2) Konfigurasi Koneksi:
 - a) Membuat objek koneksi dengan menggunakan `mysql.createConnection`.
 - b) Mengkonfigurasi host, username, password, dan nama database.
 - c) Connection details (seperti host, username, password, dan database) perlu disesuaikan dengan konfigurasi MySQL masing-masing.
- 3) Membuat Koneksi:
 - a) Menggunakan `connection.connect` untuk membuka koneksi ke database MySQL.
 - b) Menampilkan pesan ke konsol apakah koneksi berhasil atau tidak.
- 4) Ekspor Modul:
 - a) Modul ini diekspor agar bisa digunakan di file lain dalam proyek Node.js.

3. Directory dosenTest

a. deleteDosen.js



```
1 // DELETE Request dosen
2 const dosenNidn = '100003';
3 fetch(`http://localhost:3000/dosen/${dosenNidn}`, {
4   method: 'DELETE',
5   headers: {
6     'Content-Type': 'application/json'
7   }
8 })
9   .then(handleResponse)
10  .then(data => console.log('DELETE Success:', data))
11  .catch(handleError);
12
13 //handle response
14 function handleResponse(response) {
15   if (response.ok) {
16     return response.json();
17   } else {
18     throw new Error('Request failed with status: ${response.status}');
19   }
20 }
21
22 //handle error
23 function handleError(error) {
24   console.error('Request failed:', error.message);
25 }
```

File deleteDosen.js di atas adalah kode JavaScript yang menggunakan fetch API untuk melakukan permintaan DELETE ke endpoint API `http://localhost:3000/dosen/ ${dosenNidn}`, di mana `${dosenNidn}` adalah parameter yang digunakan dalam URL. Berikut penjelasan singkatnya:

1) DELETE Request menggunakan fetch:

- Menggunakan fetch untuk membuat permintaan DELETE ke endpoint tertentu.
- Alamat URL disusun dengan menggunakan backtick () dan interpolasi string untuk memasukkan nilai dari variabel `dosenNidn`.
- Menentukan metode request sebagai 'DELETE'.
- Menentukan header 'Content-Type' sebagai 'application/json'.

2) Handling Response:

- Setelah permintaan berhasil dilakukan, hasil respons dikirimkan ke fungsi `handleResponse`.
- Fungsi ini memeriksa apakah respons memiliki status OK (status kode 200).
- Jika OK, hasil respons diuraikan sebagai JSON.
- Jika tidak OK, dilemparkan error dengan pesan status yang tidak

berhasil.

3) Handling Error:

- a) Jika terjadi kesalahan selama permintaan, error ditangkap oleh fungsi `handleError`.
- b) Fungsi ini mencetak pesan kesalahan ke konsol.

4) Logging Success:

- a) Jika permintaan berhasil dan respons memiliki status OK, hasil respons dicetak ke konsol.

b. `tambahDosen.js`

A screenshot of a code editor with a dark background and light-colored text. The code is written in JavaScript and uses the `fetch` API to make a POST request. It includes a `postData` object with fields like `nidn`, `nama`, `gender`, `prodi`, `matkul`, and `alamat`. The code also features `handleResponse` and `handleError` functions to manage the response and any errors that might occur.

```
1 // POST Request dosen
2 const postData = {
3   nidn: '100003',
4   nama: 'Messi',
5   gender: 'L',
6   prodi: 'TI',
7   matkul: 'PHP',
8   alamat: 'Jl. ABC'
9 };
10
11 fetch('http://localhost:3000/dosen', {
12   method: 'POST',
13   headers: {
14     'Content-Type': 'application/json'
15   },
16   body: JSON.stringify(postData)
17 })
18   .then(handleResponse)
19   .then(data => console.log('POST Success:', data))
20   .catch(handleError);
21
22 //handle response
23 function handleResponse(response) {
24   if (response.ok) {
25     return response.json();
26   } else {
27     throw new Error('Request failed with status: ${response.status}');
28   }
29 }
30
31 //handle error
32 function handleError(error) {
33   console.error('Request failed:', error.message);
34 }
```

File `tambahDosen.js` adalah contoh kode JavaScript yang menggunakan `fetch` API untuk membuat permintaan POST ke endpoint API `http://localhost:3000/dosen`. Berikut penjelasan singkatnya:

- 1) POST Request menggunakan fetch:
 - a) Menggunakan fetch untuk membuat permintaan POST ke endpoint tertentu.
 - b) Alamat URL disusun dengan menggunakan backtick `()`.
 - c) Menentukan metode request sebagai 'POST'.
 - d) Menentukan header 'Content-Type' sebagai 'application/json'.
 - e) Menggunakan opsi body untuk mengirim data dalam format JSON. Data dipersiapkan dengan menggunakan `JSON.stringify` untuk mengubah objek JavaScript menjadi string JSON.
- 2) Handling Response:
 - a) Setelah permintaan berhasil dilakukan, hasil respons dikirimkan ke fungsi `handleResponse`.
 - b) Fungsi ini memeriksa apakah respons memiliki status OK (status kode 200).
 - c) Jika OK, hasil respons diuraikan sebagai JSON.
 - d) Jika tidak OK, dilemparkan error dengan pesan status yang tidak berhasil.
- 3) Handling Error:
 - a) Jika terjadi kesalahan selama permintaan, error ditangkap oleh fungsi `handleError`.
 - b) Fungsi ini mencetak pesan kesalahan ke konsol.
- 4) Logging Success:
 - a) Jika permintaan berhasil dan respons memiliki status OK, hasil respons dicetak ke konsol.

c. updateDosen.js

```
1 //PUT Request dosen
2 const dosenNidn = '100001';
3 const updatedData = {
4   nama : 'Siti',
5   gender : 'P',
6   prodi : 'TI',
7   matkul : 'Logika Happy',
8   alamat : 'Jl. Cibaraya Kidul'
9 };
10
11 fetch('http://localhost:3000/dosen/${dosenNidn}', {
12   method : 'PUT',
13   headers: {
14     'Content-Type': 'application/json'
15   },
16   body: JSON.stringify(updatedData)
17 })
18   .then(response => response.json())
19   .then(data => console.log(data))
20   .catch(error => console.log('Error', error));
21
22 //handle response
23 function handleResponse(response) {
24   if (response.ok) {
25     return response.json();
26   } else {
27     throw new Error('Request failed with status: ${response.status}');
28   }
29 }
30
31 //handle error
32 function handleError(error) {
33   console.error('Request failed:', error.message);
34 }
```

File updateDosen.js adalah kode JavaScript yang menggunakan fetch API untuk membuat permintaan PUT ke endpoint API `http://localhost:3000/dosen/${dosenNidn}`, di mana `${dosenNidn}` adalah parameter yang digunakan dalam URL. Berikut penjelasan singkatnya:

- 1) PUT Request menggunakan fetch:
 - a) Menggunakan fetch untuk membuat permintaan PUT ke endpoint tertentu.
 - b) Alamat URL disusun dengan menggunakan backtick () dan interpolasi string untuk memasukkan nilai dari variabel `dosenNidn`.
 - c) Menentukan metode request sebagai 'PUT'.
 - d) Menentukan header 'Content-Type' sebagai 'application/json'.
 - e) Menggunakan opsi body untuk mengirim data dalam format JSON. Data yang ingin diupdate dipersiapkan dengan menggunakan `JSON.stringify` untuk mengubah objek JavaScript menjadi string JSON.

2) Handling Response:

- a) Setelah permintaan berhasil dilakukan, hasil respons dikirimkan ke fungsi `handleResponse`.
- b) Fungsi ini memeriksa apakah respons memiliki status OK (status kode 200).
- c) Jika OK, hasil respons diuraikan sebagai JSON.
- d) Jika tidak OK, dilemparkan error dengan pesan status yang tidak berhasil.

3) Handling Error:

- a) Jika terjadi kesalahan selama permintaan, error ditangkap oleh fungsi `handleError`.
- b) Fungsi ini mencetak pesan kesalahan ke konsol.

4) Logging Success:

- 1) Jika permintaan berhasil dan respons memiliki status OK, hasil respons dicetak ke konsol.

4. Directory mahasiswaTest

a. `deleteMahasiswa.js`

```
1 // DELETE Request mahasiswa
2 const mahasiswaNim = '100003';
3 fetch('http://localhost:3000/mahasiswa/${mahasiswaNim}', {
4   method: 'DELETE',
5   headers: {
6     'Content-Type': 'application/json'
7   }
8 })
9   .then(handleResponse)
10  .then(data => console.log('DELETE Success:', data))
11  .catch(handleError);
12
13 //handle response
14 function handleResponse(response) {
15   if (response.ok) {
16     return response.json();
17   } else {
18     throw new Error(`Request failed with status: ${response.status}`);
19   }
20 }
21
22 //handle error
23 function handleError(error) {
24   console.error('Request failed:', error.message);
25 }
```

File deleteMahasiswa.js adalah kode JavaScript yang menggunakan fetch API untuk melakukan permintaan DELETE ke endpoint `http://localhost:3000/mahasiswa/${mahasiswaNim}`, di mana `${mahasiswaNim}` adalah parameter yang digunakan dalam URL. Berikut penjelasan singkatnya:

1) DELETE Request menggunakan fetch:

- a) Menggunakan fetch untuk membuat permintaan DELETE ke endpoint tertentu.
- b) Alamat URL disusun dengan menggunakan backtick () dan interpolasi string untuk memasukkan nilai dari variabel mahasiswaNim.
- c) Menentukan metode request sebagai 'DELETE'.
- d) Menentukan header 'Content-Type' sebagai 'application/json'.

2) Handling Response:

- a) Setelah permintaan berhasil dilakukan, hasil respons dikirimkan ke fungsi `handleResponse`.
- b) Fungsi ini memeriksa apakah respons memiliki status OK (status kode 200).
- c) Jika OK, hasil respons diuraikan sebagai JSON.
- d) Jika tidak OK, dilemparkan error dengan pesan status yang tidak berhasil.

3) Handling Error:

- a) Jika terjadi kesalahan selama permintaan, error ditangkap oleh fungsi `handleError`.
- b) Fungsi ini mencetak pesan kesalahan ke konsol.

4) Logging Success:

- a) Jika permintaan berhasil dan respons memiliki status OK, hasil respons dicetak ke konsol.

b. tambahMahasiswa.js



```
1 // POST Request mahasiswa
2 const postData = {
3   nim: '1908083',
4   nama: 'Messi',
5   gender: 'L',
6   prodi: 'IT',
7   alamat: 'Jl. ABC'
8 };
9
10 fetch('http://localhost:3000/mahasiswa', {
11   method: 'POST',
12   headers: {
13     'Content-Type': 'application/json'
14   },
15   body: JSON.stringify(postData)
16 })
17   .then(handleResponse)
18   .then(data => console.log('POST Success:', data))
19   .catch(handleError);
20
21 //handle response
22 function handleResponse(response) {
23   if (response.ok) {
24     return response.json();
25   } else {
26     throw new Error('Request failed with status: ${response.status}');
27   }
28 }
29
30 //handle error
31 function handleError(error) {
32   console.error('Request failed:', error.message);
33 }
```

File tambahMahasiswa.js adalah kode JavaScript yang menggunakan fetch API untuk membuat permintaan POST ke endpoint API `http://localhost:3000/mahasiswa`. Berikut penjelasan singkatnya:

1) POST Request menggunakan fetch:

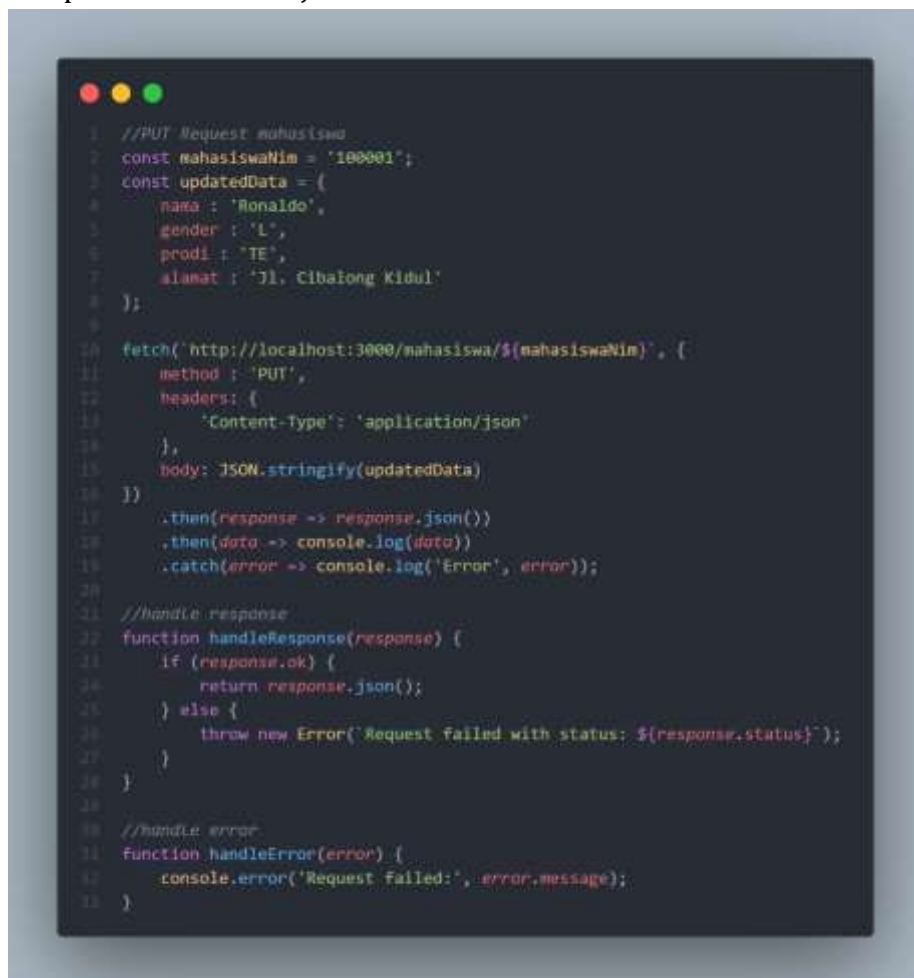
- a) Menggunakan fetch untuk membuat permintaan POST ke endpoint tertentu.
- b) Alamat URL disusun dengan menggunakan backtick ().
- c) Menentukan metode request sebagai 'POST'.
- d) Menentukan header 'Content-Type' sebagai 'application/json'.
- e) Menggunakan opsi body untuk mengirim data dalam format JSON. Data dipersiapkan dengan menggunakan `JSON.stringify` untuk mengubah objek JavaScript menjadi string JSON.

2) Handling Response:

- a) Setelah permintaan berhasil dilakukan, hasil respons dikirimkan ke fungsi `handleResponse`.
- b) Fungsi ini memeriksa apakah respons memiliki status OK (status kode 200).
- c) Jika OK, hasil respons diuraikan sebagai JSON.

- d) Jika tidak OK, dilemparkan error dengan pesan status yang tidak berhasil.
- 3) Handling Error:
 - a) Jika terjadi kesalahan selama permintaan, error ditangkap oleh fungsi `handleError`.
 - b) Fungsi ini mencetak pesan kesalahan ke konsol.
- 4) Logging Success:
 - a) Jika permintaan berhasil dan respons memiliki status OK, hasil respons dicetak ke konsol.

c. `updateMahasiswa.js`



```
1 //PUT Request mahasiswa
2 const mahasiswaNim = '100001';
3 const updatedData = {
4   nama : 'Ronaldo',
5   gender : 'L',
6   prodi : 'TE',
7   alamat : 'Jl. Cibalong Kidul'
8 };
9
10 fetch('http://localhost:3000/mahasiswa/${mahasiswaNim}', {
11   method : 'PUT',
12   headers: {
13     'Content-Type': 'application/json'
14   },
15   body: JSON.stringify(updatedData)
16 })
17 .then(response => response.json())
18 .then(data => console.log(data))
19 .catch(error => console.log('Error', error));
20
21 //handle response
22 function handleResponse(response) {
23   if (response.ok) {
24     return response.json();
25   } else {
26     throw new Error('Request failed with status: ${response.status}');
27   }
28 }
29
30 //handle error
31 function handleError(error) {
32   console.error('Request failed:', error.message);
33 }
```

File `updateMahasiswa.js` adalah kode JavaScript yang menggunakan `fetch` API untuk membuat permintaan PUT ke endpoint API `http://localhost:3000/mahasiswa/${mahasiswaNim}`, di mana `${mahasiswaNim}` adalah parameter yang digunakan dalam URL. Berikut penjelasan singkatnya:

- 1) PUT Request menggunakan fetch:
 - a) Menggunakan fetch untuk membuat permintaan PUT ke endpoint tertentu.
 - b) Alamat URL disusun dengan menggunakan backtick () dan interpolasi string untuk memasukkan nilai dari variabel mahasiswaNim.
 - c) Menentukan metode request sebagai 'PUT'.
 - d) Menentukan header 'Content-Type' sebagai 'application/json'.
 - e) Menggunakan opsi body untuk mengirim data dalam format JSON. Data yang ingin diupdate dipersiapkan dengan menggunakan JSON.stringify untuk mengubah objek JavaScript menjadi string JSON.
- 2) Handling Response:
 - a) Setelah permintaan berhasil dilakukan, hasil respons dikirimkan ke fungsi handleResponse.
 - b) Fungsi ini memeriksa apakah respons memiliki status OK (status kode 200).
 - c) Jika OK, hasil respons diuraikan sebagai JSON.
 - d) Jika tidak OK, dilemparkan error dengan pesan status yang tidak berhasil.
- 3) Handling Error:
 - a) Jika terjadi kesalahan selama permintaan, error ditangkap oleh fungsi handleError.
 - b) Fungsi ini mencetak pesan kesalahan ke konsol.
- 4) Logging Success:
 - a) Jika permintaan berhasil dan respons memiliki status OK, hasil respons dicetak ke konsol.

3. server.js



```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const mahasiswaController = require('./controllers/mahasiswaController');
4  const dosenController = require('./controllers/dosenController');
5
6  const app = express();
7  const PORT = 3000;
8
9  app.use(bodyParser.json());
10
11 app.use('/mahasiswa', mahasiswaController);
12 app.use('/dosen', dosenController);
13
14 app.listen(PORT, () => {
15   console.log(`Server is running on http://localhost:${PORT}`);
16 });
```

File server.js adalah file utama yang menginisialisasi dan menjalankan server menggunakan Express.js. Server ini menghubungkan endpoint-endpoint API dengan controller-controller yang bertanggung jawab untuk menangani operasi-operasi pada data mahasiswa dan dosen. Berikut penjelasan singkatnya:

a. Dependencies:

- 1) express: Framework Node.js untuk membangun aplikasi web.
- 2) body-parser: Middleware Express untuk mem-parsing data dari body request.

b. Inisialisasi Express:

- 1) Membuat instance dari Express dengan `const app = express();`.
- 2) Menetapkan nilai PORT ke 3000 dengan `const PORT = 3000;`.

c. Middleware body-parser:

- 1) Menggunakan body-parser untuk mem-parsing data JSON dari body request.
- 2) `app.use(bodyParser.json());` menunjukkan bahwa server ini dapat memahami dan mem-parsing data JSON.

d. Menggunakan Controller:

- 1) Menggunakan controller `mahasiswaController` dan `dosenController` untuk menangani rute-rute terkait mahasiswa dan dosen.
- 2) Endpoint untuk mahasiswa dimapping ke `/mahasiswa` dan endpoint untuk dosen ke `/dosen`.

e. Menjalankan Server:

- 1) Server mendengarkan pada port yang telah ditetapkan (PORT) dengan `app.listen`.
- 2) Ketika server berjalan, pesan log dicetak ke konsol.