Praktikum 8
Membuat Web Service dengan Node.js

A. Tujuan

1. Mampu membangun Web Service menggunakan Hapi.js (Node.js HTTP framework).
2. Mampu membangun RESTful API sederhana (CRUD).

B. Dasar Teori

saat pengguna ingin menggunakan fungsi suatu aplikasi, perangkatnya akan mengirimkan permintaan melalui HTTP ke server. Server akan mencari resource dan mengomunikasikan representasi state sebagai respons kepada pengguna melalui protokol yang sama. Representasi ini bisa dibuat dalam berbagai format, seperti JSON atau XML

API dapat dikatakan “RESTful” jika memiliki fitur berikut :
Client – server : *Client* menangani *front end* dan server menangani *back end* dan keduanya dapat diganti secara independen satu sama lain.
Stateless : Tidak ada data klien yang disimpan di server ketika ada permintaan dan status sesi disimpan di klien.
Cacheable : Klien dapat men-*cache* respon (seperti *browser* yang men-*cache* elemen statis halaman web) untuk meningkatkan kinerja.

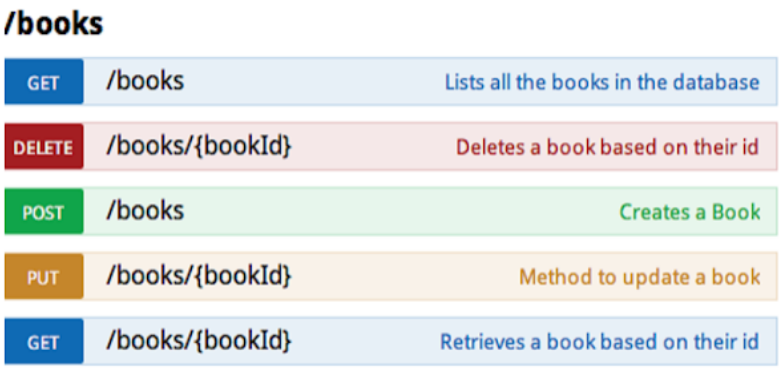
- **Method**

HTTP Method	Tujuan / Semantik
GET	Ambil Data
POST	Buat Data Baru
PUT	Update Data secara keseluruhan
PATCH	Update Data sebagian
DELETE	Hapus Data

Keberadaan Web Service memberikan dampak yang besar terhadap pembuatan maupun pengembangan sistem yang digunakan dalam lingkup organisasi. Dampak tersebut dapat dilihat dari Web Service yang berperan untuk menghubungkan masing-masing sumber data (database) dan client (aplikasi). Web Service dapat disefinisikan sebagai alat bantu dalam mediasi aplikasi dan database dalam hal menemukan, menambahkan, membuat dan menghapus (CRUD). RESTFul Web Service merupakan sebuah pendekatan baru dalam bidang Web Service. RESTFul Web Service adalah sebuah komponen sistem

yang berperan sebagai mediator atau penghubung dalam melakukan ekplorasi sumber daya. Komunikasi yang digunakan yaitu dengan memanfaatkan protokol HTTP.

- **URL**
  RESTful API diakses menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten akan menghasilkan API yang baik dan mudah untuk dimengerti developer. URL API biasa disebut *endpoint* dalam pemanggilannya.



- **HTTP Response Code**

  Status-Line merupakan salah satu bagian dari HTTP Response. Di dalam status line terdapat response code yang mengindikasikan bahwa permintaan yang client lakukan berhasil atau tidak. Karena itu, ketika membangun REST API perlu memperhatikan dan menetapkan response code secara benar.

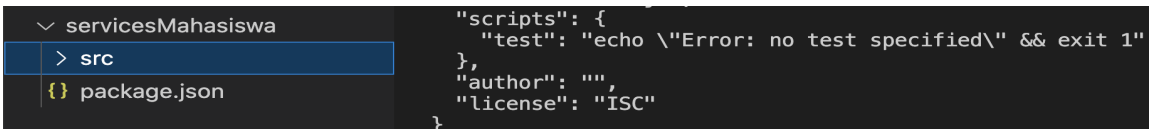
  Status code bernilai 3 digit angka. Pada REST API, berikut nilai-nilai status code yang sering digunakan:

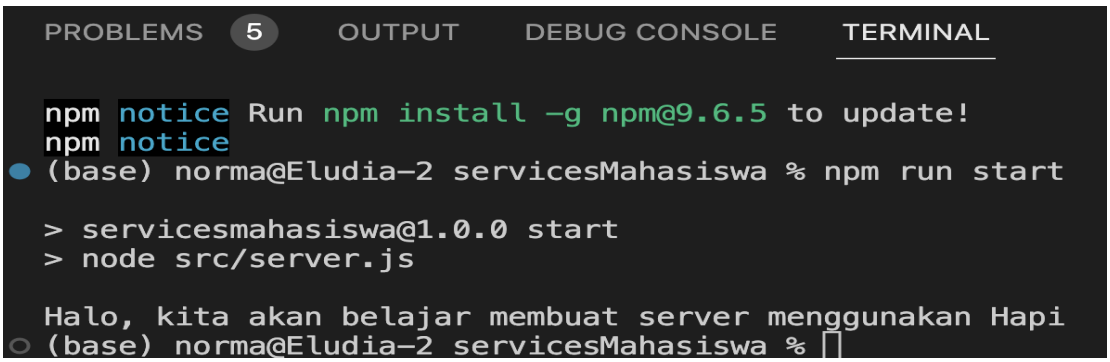
  - **200** (OK) - Permintaan client berhasil dijalankan oleh server.
  - **201** (Created) - Server berhasil membuat/menambahkan resource yang diminta client.
  - **400** (Bad Request) - Permintaan client gagal dijalankan karena proses validasi input dari client gagal.
  - **401** (Unauthorized) - Permintaan client gagal dijalankan. Biasanya ini disebabkan karena pengguna belum melakukan proses autentikasi.
  - **403** (Forbidden) - Permintaan client gagal dijalankan karena ia tidak memiliki hak akses ke resource yang diminta.
  - **404** (Not Found) - Permintaan client gagal dijalankan karena resource yang diminta tidak ditemukan.
  - **500** (Internal Server Error) - Permintaan client gagal dijalankan karena server mengalami eror (membangkitkan Exception).

C. Praktikum

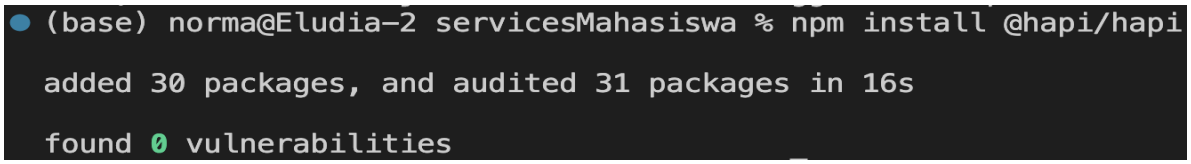
1. membuat proyek node js dengan buatlah folder baru terlebih dahulu misal pada *C -> servicesMahasiswa -> src*
2. buka folder *servicesMahasiswa* menggunakan VSCode. Caranya, pada Visual Studio Code pilih menu *File -> Open Folder -> [pilih foldernya]*.
3. Untuk membuat proyek JavaScript, silakan buka Terminal pada VSCode. Pilih menu *Terminal -> New Terminal*, kemudian tuliskan perintah: npm run start

   tampilan struktur folder di vscode



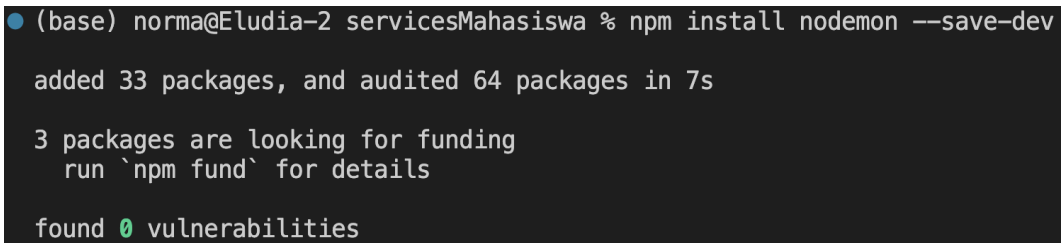


4. instal server hapi dengan perintah : npm install @hapi/hapi



   pastikan server hapi sudah terinstal dengan mengecek pada file package.json

5. instal nodemon dengan perintah : npm install nodemon –save-dev



```
"devDependencies": {
  "nodemon": "^2.0.22"
}
```

6. Instal nanoid untuk generate id mahasiswa secara otomatis dengan perintah : npm install nanoid@3

```
(base) norma@Eludia-2 servicesMahasiswa % npm install nanoid@3

changed 1 package, and audited 65 packages in 817ms

4 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ketentuan :

1. Aplikasi menggunakan port 9000 : ditentukan pada file server.js
2. Aplikasi dijalankan dengan perintah npm run start
   Tambahkan properti start ke dalam properti scripts pada **package.json** seperti berikut:

```
{
  "name": "submission",
  ...
  "scripts": {
    "start": "node src/server.js",
  }
}
```

3. API dapat menyimpan data mahasiswa dengan *route* berikut ini :
   - Method : POST
   - URL : /students
   - Body Request :

```
{
   "nama": string,
   "tahunMasuk": number,
   "alamat": string,
   "noHP": string,
   "prodi": string,
   "umur": number,
  “anakKe”: number,
   "jumlahSaudara": number,
   "menikah": boolean
}
```

Objek mahasiswa yang disimpan pada server memiliki struktur berikut :

```
{
   “id”: “Qbax50y7Lf4l”
   "nama": Reni,
```

```
    "tahunMasuk": 2021,
    "alamat": jl Gayungan,
    "noHP": 081111,
    "prodi": TRI,
    "umur": 20,
    “anakKe”: 3,
    "jumlahSaudara": 5,
    “anakTunggal”: false,
    "menikah": false,
    "tanggalLahir": "2021-03-04T09:11:44.598Z"
}
```

Properti yang ditebalkan diolah dan diperoleh disisi server sebagai berikut :

- id : nilai id haruslah unik. Untuk membuat nilai unik, bisa memanfaatkan nanoid.
- anakTunggal : merupakan properti *boolean* yang menjelaskan apakah mahaiswa merupakan anak tuggal atau tidak. Nilai anakTunggal didapatkan dari observasi anakKe === jumlahSaudara. jika anak tunggal, jumlah saudara diisi nilai 1. .
- tanggalLahir : merupakan properti yang menampung tanggal lahir mahasiswa. bisa gunakan new Date().toISOString() untuk menghasilkan nilainya.

Server harus merespons **gagal** bila:
Client tidak melampirkan properti nama pada *request body*. Bila hal ini terjadi, maka *server* akan merespons dengan:

- Status Code : **400**
- Response Body:

```
{
    "status": "fail",
    "message": "Gagal menambahkan mahasiswa. Mohon isi nama mahasiswa"
}
```

Client melampirkan nilai properti anakKe yang lebih besar dari nilai properti jumlahSaudara. Bila hal ini terjadi, maka *server* akan merespons dengan:

- Status Code : **400**
- Response Body:

```
{
    "status": "fail",
    "message": "Gagal menambahkan mahasiswa. anakKe tidak boleh lebih
besar dari jumlahSaudara"
}
```

Bila buku **berhasil** dimasukkan, *server* harus mengembalikan respons dengan:

- Status Code : **201**
- Response Body:

```
{
    "status": "success",
    "message": "mahasiswa berhasil ditambahkan",
    "data": {
        "mahasiswaId": "1L7ZtDUFeGs7VlEt"
    }
}
```

4. API dapat menampilkan seluruh mahasiswa

API yang dibuat harus dapat menampilkan seluruh mahasiswa yang disimpan melalui *route*:
- Method : **GET**
- URL: **/students**

Server harus mengembalikan respons dengan:
- Status Code : **200**
- Response Body:

```
{
    "status": "success",
    "data": {
        "students": [
            {
                "id": "Qbax5Oy7L8WKf74l",
                "nama": "mhs A",
                "prodi": "TRI"
            },
            {
                "id": "1L7ZtDUFeGs7VlEt",
                "nama": "mhs B",
                "prodi": "TRI"
            },
            {
                "id": "K8DZbfI-t3LrY7lD",
                "nama": "mhs C",
                "prodi": "TRI"
            }
        ]
    }
}
```

Jika **belum** terdapat mahasiswa yang dimasukkan, *server* bisa merespons dengan *array* students kosong.

```
{
    "status": "success",
    "data": {
        "students": []
    }
}
```

5. API dapat menampilkan detail mahasiswa

API yang dibuat harus dapat menampilkan seluruh mahasiswa yang disimpan melalui *route*:
- Method : **GET**
- URL: **/students/{mahasiswaId}**

Bila mahasiswa dengan id yang dilampirkan oleh *client* tidak ditemukan, maka *server* harus mengembalikan respons dengan:
- Status Code : **404**
- Response Body:

```
{
    "status": "fail",
    "message": "Mahasiswa tidak ditemukan"
}
```

Bila buku dengan id yang dilampirkan **ditemukan**, maka *server* harus mengembalikan respons dengan:

Status Code : **200**

Response Body:

```
{
    "status": "success",
    "data": {
        "mahasiswa": {
            “id”: “Qbax50y7Lf4l”
            "nama": Reni,
            "tahunMasuk": 2021,
            "alamat": jl Gayungan,
            "noHP": 081111,
            "prodi": TRI,
            "umur": 20,
            “anakKe”: 3,
            "jumlahSaudara": 5,
            “anakTunggal”: false,
            "menikah": false,
            "tanggalLahir": "2021-03-04T09:11:44.598Z"
        }
    }
}
```

6. API dapat mengubah data mahasiswa

API yang dibuat harus dapat mengubah data buku berdasarkan id melalui *route*:

- Method : **PUT**
- URL : **/students/{mahasiswaId}**
- Body Request:

```
{
    "nama": string,
    "tahunMasuk": number,
    "alamat": string,
    "noHP": string,
    "prodi": string,
    "umur": number,
   “anakKe”: number,
    "jumlahSaudara": number,
    "menikah": boolean
}
```

Server harus merespons **gagal** bila:

Client tidak melampirkan properti nama pada *request body*. Bila hal ini terjadi, maka *server* akan merespons dengan:

- Status Code : **400**
- Response Body:

```
{
    "status": "fail",
    "message": "Gagal memperbarui mahasiswa. Mohon isi nama mahasiswa"
}
```

Client melampirkan nilai properti anakKe yang lebih besar dari nilai properti jumlahSaudara. Bila hal ini terjadi, maka *server* akan merespons dengan:

- Status Code : **400**
- Response Body:

```
{
    "status": "fail",
    "message": "Gagal memperbarui mahasiswa. anakKe tidak boleh lebih besar dari jumlahSaudara"
}
```

Idyang dilampirkan oleh *client* tidak ditemukkan oleh *server*. Bila hal ini terjadi, maka *server* akan merespons dengan:

- Status Code : **404**
- Response Body:

```
{
    "status": "fail",
    "message": "Gagal memperbarui mahasiswa. Id tidak ditemukan"
}
```

7. API dapat menghapus data mahasiswa

   API yang dibuat harus dapat menghapus mahasiswa berdasarkan id melalui *route* berikut:

   - Method : **DELETE**
   - URL: /students/{mahasiswaId}

   Bila id yang dilampirkan tidak dimiliki oleh mahasiswa manapun, maka *server* harus mengembalikan respons berikut:

   - Status Code : **404**
   - Response Body:

```
{
    "status": "fail",
    "message": "Mahasiswa gagal dihapus. Id tidak ditemukan"
}
```

   Bila id dimiliki oleh salah satu buku, maka buku tersebut harus dihapus dan *server* mengembalikan respons berikut:

   - Status Code : **200**
   - Response Body:

```
{
    "status": "success",
    "message": "Mahasiswa berhasil dihapus"
}
```

Bila buku berhasil diperbarui, *server* harus mengembalikan respons dengan:

- Status Code : 200
- Response Body:

```
{
   "status": "success",
   "message": "Mahasiswa berhasil diperbarui"
}
```

Program :

1. Pada folder src buatlah 4 file dengan nama :
   mahasiswa.js
   route.js
   hadler.js
   server,js

mahasiswa.js

servicesMahasiswa > src > JS mahasiswa.js >

```
1   const students = [];
2
3   module.exports = students;
```

server.js

servicesMahasiswa > src > JS server.js > ...

```
1   const Hapi = require('@hapi/hapi');
2   const routes = require('./route');
3
4   const init = async () => {
5       const server = Hapi.server({
6           port: 9000,
7           host: 'localhost',
8       });
9
10      server.route(routes);
11
12      await server.start();
13      console.log(`Server berjalan pada ${server.info.uri}`);
14  };
15
16  init();
```

route.js

servicesMahasiswa > src > JS route.js > ...

```
1   const { addStudentsHandler,
2       getAllStudentsHandler,
3       getStudentByIdHandler,
4       editStudentByIdHandler,
5       deleteStudentByIdHandler
6   } = require("./handler")
7
8   const routes = [
9   {
10    method: 'POST',
11    path: '/students',
12    handler: addStudentsHandler
13  },
14  {
15      method: 'GET',
16      path: '/students',
17      handler: getAllStudentsHandler
18    },
19    {
20      method: 'GET',
21      path: '/students/{studentId}',
22      handler: getStudentByIdHandler
23    },
24    {
25      method: 'PUT',
26      path: '/students/{studentId}',
27      handler: editStudentByIdHandler,
28    },
29    {
30      method: 'DELETE',
31      path: '/students/{studentId}',
32      handler: deleteStudentByIdHandler,
33    },
34  ];
35
36  module.exports = routes;
```

handler.js

```
const { nanoid } = require("nanoid");
const students = require("./mahasiswa");

const addStudentsHandler = (request, h) => {
    const {
        name,
        tahunMasuk,
        alamat,
        noHP,
        prodi,
        umur,
        anakKe,
        jumlahSaudara,
        menikah
    } = request.payload;

    if (name === null || name === "" || name === undefined) {
        const response = h.response({
            status: "fail",
            message: "Gagal menambahkan mahasiswa. Mohon isi nama
mahasiswa"
        })
        response.code(400);
        return response;
    }

    if (anakKe > jumlahSaudara) {
        const response = h.response({
            status: "fail",
            message: "Gagal menambahkan mahasiswa. anakKe tidak
boleh lebih besar dari jumlahSaudara"
        })
        response.code(400);
        return response;
    }

    const id = nanoid();
    const anakTunggal = anakKe === jumlahSaudara;
    const insertedAt = new Date().toISOString();
    const tanggalLahir = insertedAt;

    const newstudent = {
        id,
        name,
        tahunMasuk,
        alamat,
        noHP,
        prodi,
        umur,
        anakKe,
        jumlahSaudara,
        anakTunggal,
        menikah,
```

```
        tanggalLahir
    }

    students.push(newstudent);
    console.log(students);


    const response = h.response({
        status: "success",
        message: "mahasiswa berhasil disimpan",
        data: {
            studentId: id
        }
    })
    response.code(201);
    return response;
};


const getAllStudentsHandler = (request, h) => {

    const response = h.response({
        status: "success",
        data: {
            students,
        }
    })
    response.code(200);
    return response;
};


const getStudentByIdHandler = (request, h) => {
    const { studentId } = request.params;
    const student = students.filter((n) => n.id === studentId)[0];

    console.log(studentId, student)

    if (student !== undefined) {
        return {
            status: 'success',
            data: {
                student,
            },
        };
    }
    const response = h.response({
        status: 'fail',
        message: 'mahasiswa tidak ditemukan',
    });
    response.code(404);
    return response;
};

const editStudentByIdHandler = (request, h) => {
```

```
    const { studentId } = request.params;

    const { name,
        tahunMasuk,
        alamat,
        noHP,
        prodi,
        umur,
        anakKe,
        jumlahSaudara,
        menikah
    } = request.payload;

    if (name === null || name === "" || name === undefined) {
        const response = h.response({
            status: "fail",
            message: "Gagal memperbarui mahasiswa. Mohon isi nama
mahasiswa"
        })
        response.code(400);
        return response;
    }

    if (anakKe > jumlahSaudara) {
        const response = h.response({
            status: "fail",
            message: "Gagal memperbarui mahasiswa. anakKe tidak
boleh lebih besar dari jumlahSaudara"
        })
        response.code(400);
        return response;
    }
    const updatedAt = new Date().toISOString();

    const index = students.findIndex((student) => student.id ===
studentId);

    if (index !== -1) {
        students[index] = {
            ...students[index],
            name,
            tahunMasuk,
            alamat,
            noHP,
            prodi,
            umur,
            anakKe,
            jumlahSaudara,
            menikah
        };

        const response = h.response({
            status: 'success',
            message: 'mahasiswa berhasil diperbarui',
        });
```

```
        response.code(200);
        return response;
    }

    const response = h.response({
        status: 'fail',
        message: 'Gagal memperbarui mahasiswa. Id tidak ditemukan',
    });
    response.code(404);
    return response;
};

const deleteStudentByIdHandler = (request, h) => {
    const { studentId } = request.params;

    const index = students.findIndex((student) => student.id ===
studentId);

    if (index !== -1) {
        students.splice(index, 1);
        return {
            status: 'success',
            message: 'Mahasiswa berhasil dihapus',
        };
    }

    const response = h.response({
        status: 'fail',
        message: 'Mahasiswa gagal dihapus. Id tidak ditemukan',
    });
    response.code(404);
    return response;
};


module.exports = {
    addStudentsHandler,
    getAllStudentsHandler,
    getStudentByIdHandler,
    editStudentByIdHandler,
    deleteStudentByIdHandler
};
```

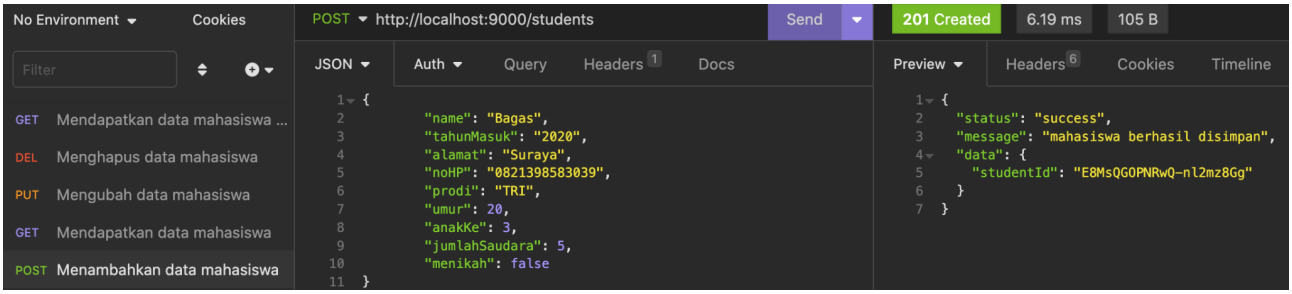
2. Pastikan pada file package.json sudah ter *depedency* dengan hapi, nanoid dan nodemon

servicesMahasiswa > {} package.json > ...

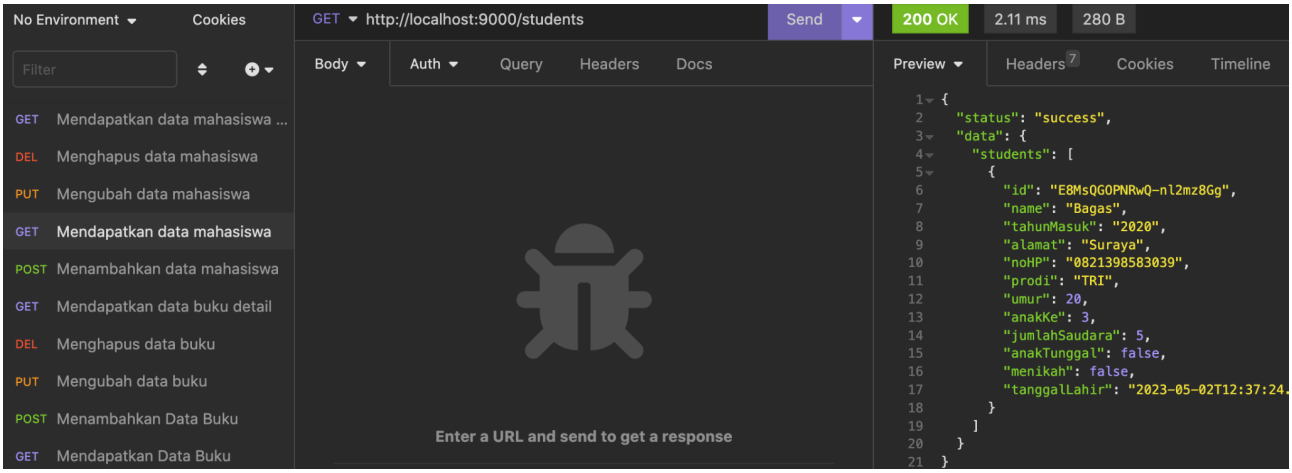
```json
{
  "name": "servicesmahasiswa",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon src/server.js",
    "dev": "nodemon src/server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@hapi/hapi": "^21.3.2",
    "nanoid": "^3.3.6"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

3. hasil tampilan pada insomnia

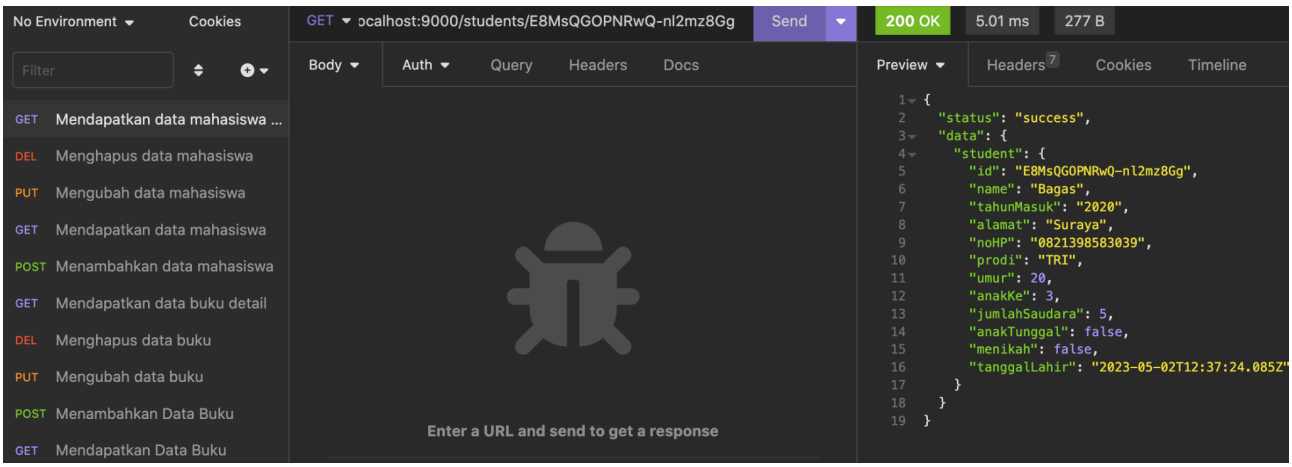
a. tambah data (POST)



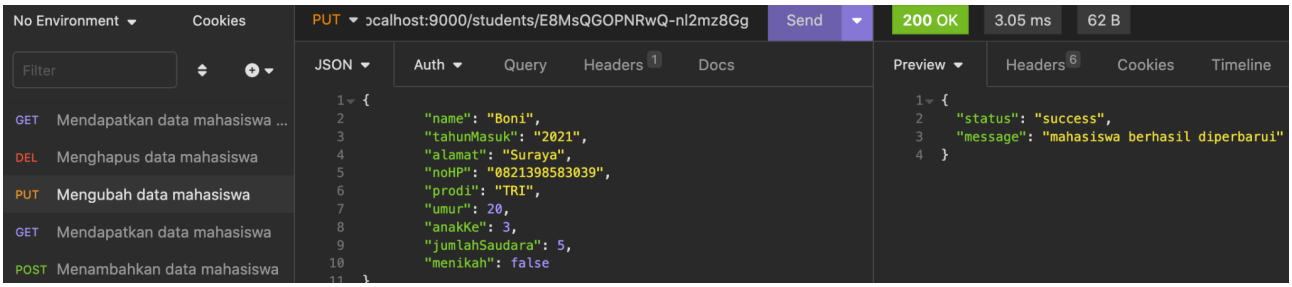
b. melihat data



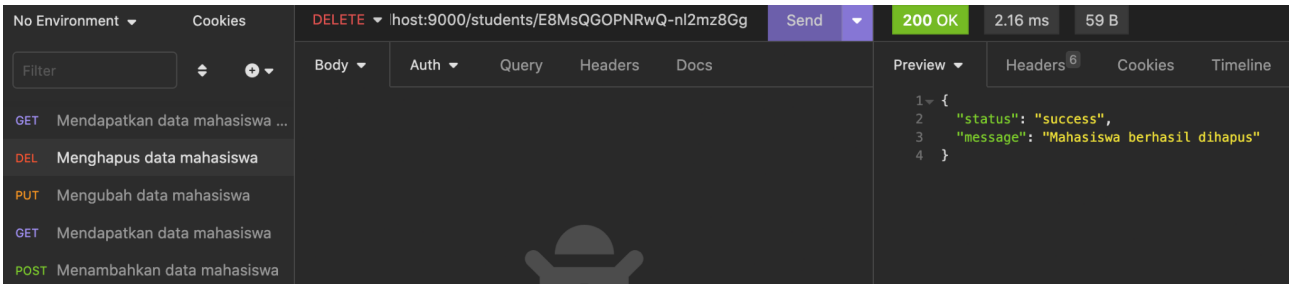
c. melihat data dengan id



d. mengubah data



e. menghapus data



D. Laporan Resmi

Analisalah semua program diatas dan buat kesimpulan