BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Posisi yang dijalankan adalah sebagai *Backend Developer Intern* dalam tim *developer* dan berada di bawah supervisi langsung Abdur Rozaq sebagai *Frontend Developer* serta Kaka Alfuna sebagai *Tech Lead*. Dalam pelaksanaan tugas, dilakukan koordinasi secara langsung dengan pihak terkait untuk menyampaikan perkembangan pekerjaan serta memperoleh bimbingan teknis. Selain itu, arahan dan dukungan juga diberikan oleh anggota tim lainnya selama proses pelaksanaan tugas.

Sebagai perusahaan yang beroperasi dalam ekosistem *startup*, pola komunikasi di lingkungan kerja bersifat langsung dan informal, didukung oleh kedekatan fisik antaranggota tim yang bekerja dalam satu ruang. Meskipun proyek yang dikerjakan tidak termasuk dalam proyek utama tim, partisipasi tetap dilakukan dalam rapat rutin tim *developer* guna memperluas pemahaman terhadap keseluruhan proses pengembangan, menjaga sinkronisasi dengan standar teknis yang diterapkan, serta mendukung integrasi kerja tim secara menyeluruh.

3.2 Tugas yang Dilakukan

Selama pelaksanaan kegiatan magang di Five Elements Marketing Agency, tanggung jawab utama difokuskan pada pengembangan *backend* serta dukungan terhadap integrasi sistem digital perusahaan. Tugas-tugas yang dilaksanakan meliputi:

- Pengembangan dan implementasi RESTful API untuk *website* Five Elements dan LGM Agency menggunakan framework Next.js. API ini dirancang untuk memungkinkan pengelolaan data secara dinamis melalui antarmuka *admin* internal.
- Perancangan dan pengelolaan *database* menggunakan MongoDB, termasuk pembuatan skema data untuk berbagai entitas seperti artis, *partner*, dan informasi perusahaan.
- Implementasi sistem autentikasi dan otorisasi berbasis JWT (JSON Web Token) untuk akun *admin*, guna memastikan keamanan akses data dalam sistem.

Seluruh tugas ini bertujuan untuk meningkatkan skalabilitas, efisiensi, dan fleksibilitas dalam pengelolaan konten digital pada platform milik perusahaan. Selain itu, kegiatan ini memberikan pengalaman langsung dalam penerapan praktik pengembangan perangkat lunak modern di lingkungan industri.

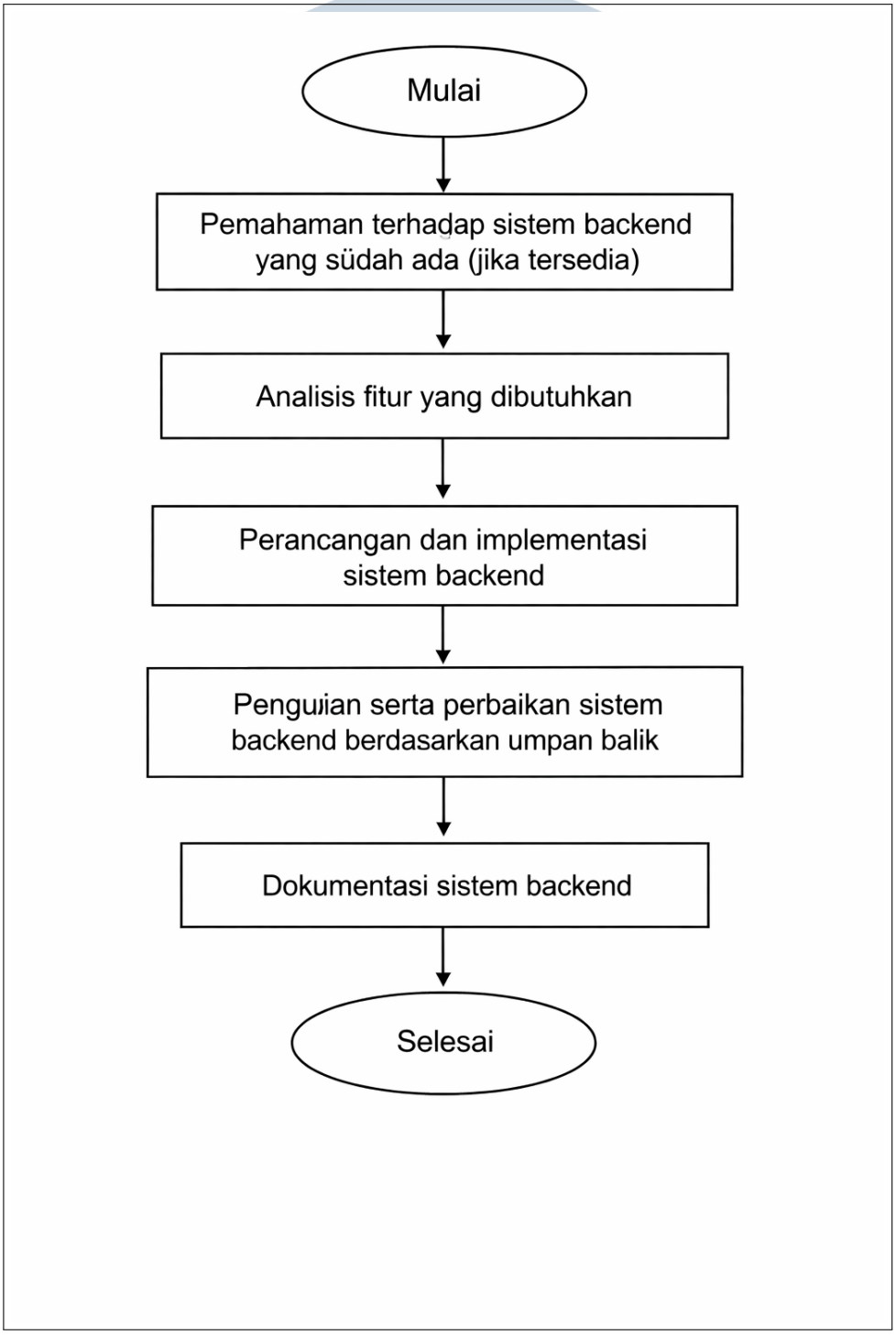
3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke-	**Pekerjaan yang dilakukan**
1	Orientasi dengan membaca dokumentasi dan kode pada *website* P-Club
2	Memperbarui dan menyusun ulang dokumentasi pada *website* P-Club
3	Mengembangkan *backend website* Five Elements: API *endpoint* autentikasi JWT
4	Mengembangkan *backend website* Five Elements: API *endpoint admin*
5	Mengembangkan *backend website* Five Elements: API *endpoint About Us*
6	Mengembangkan *backend website* Five Elements: API *endpoint layanan*
7	Mengembangkan *backend website* Five Elements: API *endpoint Partner dan Project*
8	Mengembangkan *backend website* Five Elements: API *endpoint Awards dan Blog*
9	Mengembangkan *backend website* Five Elements: API *endpoint Clients, Contact Us, dan Subscription*
10	Merancang *mockup dashboard* untuk mengakses CRUD *endpoint* API pada *website* Five Elements
11	Mengembangkan API *endpoint* autentikasi JWT dan *admin* pada *website* LGM Agency
12	Mengembangkan API *endpoint About Us* dan artis pada *website* LGM Agency
13	Mengembangkan API *endpoint* email staf, pengiriman email, dan permintaan tiket pada *website* LGM Agency
14	Mengembangkan API *endpoint* integrasi Spotify API dan iTunes API
15	Riset dan penyusunan laporan tentang API media sosial dan aplikasi musik
16	Review ulang kode *website* Five Elements dan LGM Agency serta penyusunan dokumentasi *backend API*

3.4 Proses dan Hasil



Gambar 3.1. *Flowchart* perancangan sistem *backend*.

Proses perancangan sistem *backend* diawali dengan pemahaman terhadap sistem *backend* yang telah digunakan pada *website* dan aplikasi lain milik perusahaan. Tahap ini bertujuan untuk menyesuaikan struktur dan pola pengembangan *backend* yang baru agar selaras dengan sistem yang sudah ada, sehingga memudahkan proses pemeliharaan dan pemahaman oleh *developer* internal. Selanjutnya, dilakukan analisis terhadap *website* yang akan dirancang sistem *backend*-nya untuk menentukan fitur-fitur yang dibutuhkan. Berdasarkan hasil analisis tersebut, sistem *backend* dirancang dan diimplementasikan melalui pembuatan *database*, *middleware*, serta *API*. Setelah implementasi selesai, *backend* diuji secara mandiri menggunakan *Postman* untuk memastikan setiap *endpoint API*, *middleware*, dan interaksi dengan *database* berjalan dengan baik. Hasil pengujian kemudian diajukan untuk direview oleh *supervisor* dan *mentor*, sehingga diperoleh umpan balik yang digunakan sebagai dasar perbaikan dan penyesuaian sistem *backend*. Tahap akhir dari proses ini adalah penyusunan dokumentasi sistem *backend* agar mudah dipahami dan digunakan oleh *developer* lain di masa mendatang.

3.4.1 Pengembangan RESTful API Pada *website* Five Elements dan LGM Agency

Website Five Elements dan LGM Agency dapat dikelola secara dinamis, memberikan akses data kepada *admin* melalui *dashboard* internal pada *frontend*, serta mendukung pengembangan *website* yang skalabel dengan menerapkan RESTful API. Melalui RESTful API, data pada *website* dapat diakses dan dimanipulasi secara *real time* menggunakan berbagai endpoint seperti:

- GET, untuk mengambil data;
- POST, untuk menambahkan data;
- PUT, untuk memperbarui data; dan
- DELETE, untuk menghapus data.

Dengan pendekatan ini, pengelolaan data menjadi lebih fleksibel dan dapat dilakukan langsung dari *dashboard frontend*. *Admin* juga dapat mengakses data *backend* secara aman melalui *frontend* internal yang dilindungi dengan sistem autentikasi berbasis JWT (JSON Web Token). Selain itu, RESTful API memungkinkan *backend* dan *frontend* dikembangkan secara terpisah (*decoupled*), sehingga sistem lebih mudah untuk ditingkatkan atau disesuaikan dengan kebutuhan saat ini maupun di masa mendatang. Pendekatan ini mendukung skalabilitas sistem digital perusahaan secara efisien dan berkelanjutan.

Dalam pengembangan RESTful API untuk *website* Five Elements dan LGM Agency, digunakan berbagai alat dan teknologi yang dikategorikan sebagai berikut:

- Framework
  - Next.js – *Framework* berbasis React yang digunakan untuk membangun aplikasi full-stack dengan kemampuan API routing dan server-side rendering.
- Language
  - TypeScript – Bahasa pemrograman yang merupakan superset dari JavaScript, menyediakan pengetikan statis untuk pengembangan yang lebih aman dan terstruktur.

- Database
  - MongoDB – Database NoSQL yang digunakan untuk menyimpan dan mengelola data dalam format dokumen JSON.
- Library
  - next – *Library* internal dari Next.js yang mencakup modul seperti `next/server` dan `next/headers` untuk pengembangan sisi server.
  - mongoose – *Library ODM (Object Data Modeling)* untuk integrasi antara aplikasi Node.js dan MongoDB.
  - @aws-sdk/client-s3 – Modul dari AWS SDK untuk mengelola penyimpanan dan pengunggahan file ke Amazon S3.
  - bcryptjs – *Library* untuk hashing dan verifikasi password dengan aman.
  - jose – *Library* untuk membuat dan memverifikasi JSON Web Token (JWT) untuk sistem autentikasi dan otorisasi.
- Tools
  - Postman – Digunakan untuk menguji dan memverifikasi *endpoint* RESTful API.
  - Visual Studio Code (VS Code) – Editor kode sumber utama yang digunakan dalam pengembangan proyek.

Langkah pertama dalam pengembangan RESTful API adalah melakukan analisis terhadap struktur dan kebutuhan data pada *website*. Setiap halaman ditinjau untuk mengidentifikasi bagian-bagian yang mengandung data dinamis atau informasi yang perlu diperbarui secara cepat oleh *admin*. Untuk memastikan keakuratan dan kelengkapan kebutuhan, dilakukan konsultasi dengan pembimbing guna memperoleh masukan terkait data yang perlu disediakan melalui API. Setelah kebutuhan data ditentukan, proyek *website* yang sudah ada diakses dan pengembangan API dilakukan dengan pendekatan *modular*. Struktur folder `api` dalam Next.js dimanfaatkan untuk membuat masing-masing *endpoint* secara terpisah sesuai dengan entitas data, seperti artis, *partner*, blog, dan informasi perusahaan. Setiap *endpoint* mendukung metode dasar seperti `GET`, `POST`, `PUT`, dan `DELETE`.

Setelah pengembangan API selesai, dilakukan pengujian menggunakan Postman untuk memastikan setiap *endpoint* merespons dengan benar sesuai dengan kebutuhan fungsionalitas dan keamanan sistem. Pengujian meliputi validasi data, pengujian autentikasi JWT, serta simulasi interaksi antara *frontend* dan *backend*. Selain itu, dibuat *mockup* sederhana untuk *dashboard frontend* yang berfungsi sebagai referensi awal dalam pengembangan antarmuka untuk mengakses API. *Endpoint* API kemudian dihubungkan dengan bagian *frontend* yang telah tersedia agar proses integrasi dapat berjalan dengan lancar dan dapat langsung digunakan oleh pihak terkait.

Setelah tahap implementasi, dilakukan evaluasi oleh pembimbing dan *supervisor* terkait struktur, keamanan, serta kelengkapan *endpoint* yang dikembangkan. Berdasarkan masukan tersebut, perbaikan dan penyesuaian dilakukan apabila diperlukan. Sebagai tahap akhir, disusun dokumentasi teknis yang mencakup daftar *endpoint*, metode HTTP yang digunakan, struktur *request* dan *response*, serta mekanisme autentikasi. Dokumentasi ini disiapkan untuk memudahkan tim *frontend* dan pengembang lainnya dalam memahami serta memanfaatkan API secara optimal.

A RESTful API Pada *Website* Five Elements

Tabel 3.2. Daftar API Endpoint pada Website Five Elements

No	Endpoint	Fungsi
1	/api/auth/login	Melakukan autentikasi admin menggunakan email dan kata sandi
2	/api/auth/logout	Mengakhiri sesi autentikasi admin
3	/api/admin	Mengelola data admin (menampilkan dan menambahkan admin)
4	/api/admin/{id}	Mengelola data admin berdasarkan ID (detail, ubah, hapus)
5	/api/aboutUs/protected	Mengelola data About Us oleh admin
6	/api/aboutUs/protected/{id}	Mengelola detail data About Us berdasarkan ID
7	/api/aboutUs/published	Menampilkan data About Us yang telah dipublikasikan
8	/api/aboutUs/published/{id}	Menampilkan detail About Us yang telah dipublikasikan
9	/api/awards/protected	Mengelola data penghargaan oleh admin
10	/api/awards/published	Menampilkan data penghargaan yang dipublikasikan
11	/api/blog/protected	Mengelola artikel blog oleh admin
12	/api/blog/published	Menampilkan artikel blog yang dipublikasikan
13	/api/client/protected	Mengelola data klien oleh admin
14	/api/client/published	Menampilkan data klien yang dipublikasikan
15	/api/package/protected	Mengelola data paket layanan oleh admin
16	/api/package/published	Menampilkan paket layanan yang dipublikasikan
17	/api/partner/protected	Mengelola data mitra oleh admin
18	/api/partner/published	Menampilkan data mitra yang dipublikasikan
19	/api/project/protected	Mengelola data proyek oleh admin
20	/api/project/published	Menampilkan data proyek yang dipublikasikan
21	/api/subscription	Mengelola data langganan email pengguna
22	/api/contactUs	Mengelola pesan dan pertanyaan dari pengunjung website

A.1 *Admin*

`api/admin`

Endpoint ini digunakan untuk mengambil seluruh data *admin* serta menambahkan data *admin* baru ke dalam sistem.

```
import { NextRequest, NextResponse } from "next/server";
import Admin from "@/common/schemas/dbSchema/admin";
import { connectToDatabase } from "@/lib/mongodb";
import bcryptjs from 'bcryptjs';
import { uploadFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'

export async function GET(req: NextRequest) {
    try {
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true' ? true : false;
        const skip = (page - 1) * limit;
        let filter
        let checkAdmins;
        await connectToDatabase();
        let totalData = await Admin.countDocuments();
        if (search) {
          filter = [
            { fullName: { $regex: search, $options: 'i' } },
            { position: { $regex: search, $options: 'i' } },
            { description: { $regex: search, $options: 'i' } },
          ]
          checkAdmins = await Admin.find({
            $or: filter
          }).select('-password -token').sort({ createdAt: -1 }).skip(skip).limit(limit);
          totalData = checkAdmins.length;
        } else if (all) {
            checkAdmins = await Admin.find().select('-password -token ').sort({ createdAt: -1 });
        } else {
            checkAdmins = await Admin.find().select('-password -token').sort({ createdAt: -1 }).skip(skip).limit(limit);
        }
        const result = {
            admins: checkAdmins,
            pagination: {
                page: page,
                limit: limit,
                totalPages: Math.ceil(  totalData / limit),
                totalData: totalData
            }
        }
        return NextResponse.json({ status: 200, message: 'admin get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function POST(req: NextRequest) {
    try {
        const formData = await req.formData();
        const { username, password, fullName, position,
```

```
    description, image } = await parseJson(formData);
54          if (!username || !password || !fullName || !position || !
    image || !description) return NextResponse.json({ status: 400,
    message: 'Bad Request' }, { status: 400 });
55          const encryptPassword =  await bcryptjs.hash(password as
    string, 10);
56          await connectToDatabase();
57          const checkUser = await Admin.findOne({ username: username
     });
58          if (checkUser) {
59              return NextResponse.json({ status: 409, message: 'User
     already exists' }, { status: 409 });
60          }
61          console.log(encryptPassword);
62          const imageUrl = await uploadFile(formData, 'admin', '
    image');
63          const admin = await Admin.create({ username, password :
    encryptPassword, fullName, position, description, image :
    imageUrl });
64          return NextResponse.json({ status: 201, message: 'admin
    creeated successfully', data: admin }, { status: 201 });
65      } catch (error) {
66          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
67      }
68  }
```

Kode 3.1: Kode untuk `api/admin`

Methods

- `GET` : Digunakan untuk mengambil data *admin* dengan dukungan fitur pencarian dan *pagination*.
- `POST` : Digunakan untuk menambahkan data *admin* baru ke dalam *database*.

`api/admin/[id]`

Endpoint ini digunakan untuk mengelola satu data *admin* berdasarkan ID tertentu.

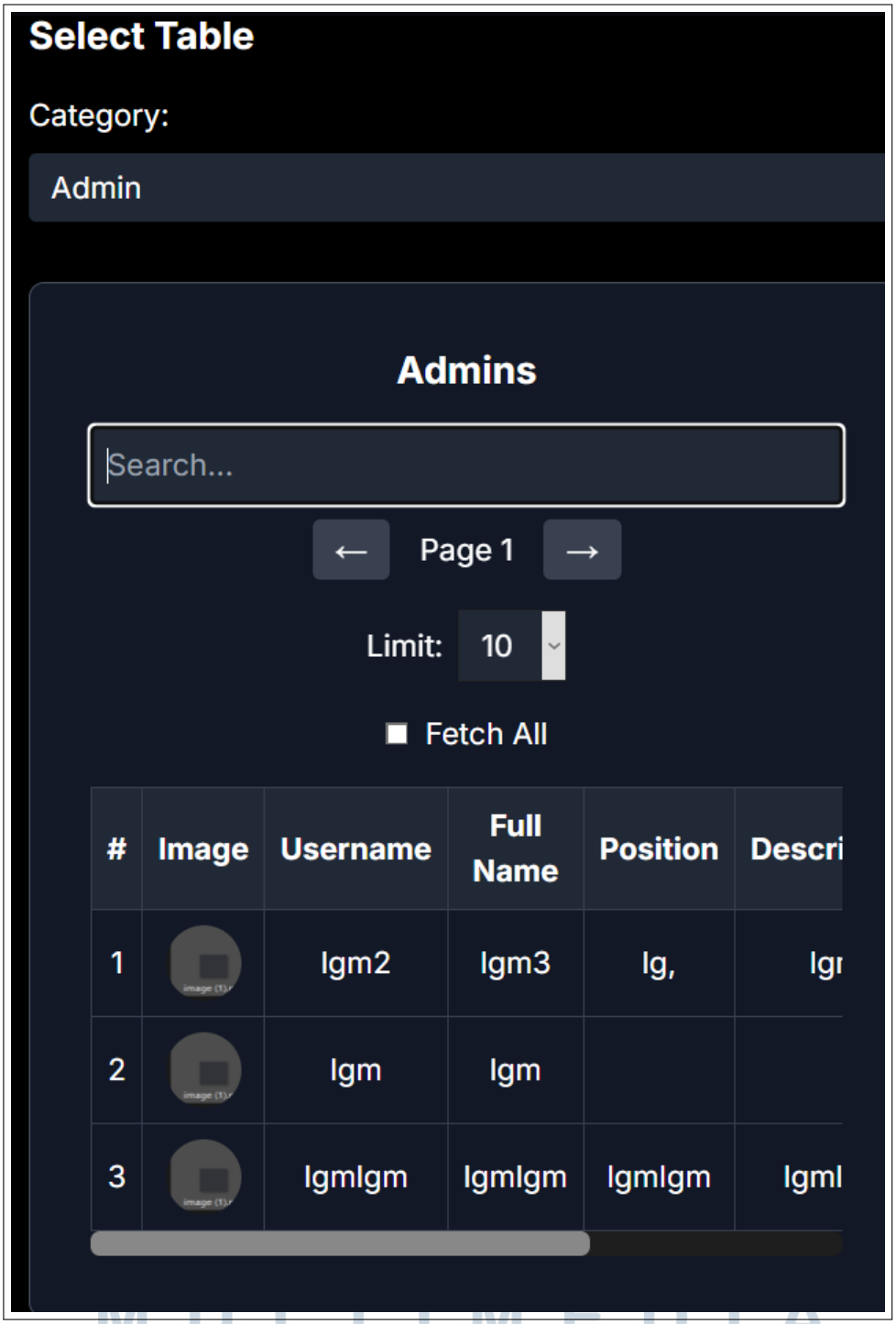
```
1  import { NextRequest, NextResponse } from "next/server";
2  import Admin from "@/common/schemas/dbSchema/admin";
3  import { connectToDatabase } from "@/lib/mongodb";
4  import bcryptjs from 'bcryptjs';
5  import {uploadFile,  deleteFile } from "@/lib/upload";
6  import {parseJson} from '@/lib/parseJson'
7
8
9  export async function GET(
10     _req: NextRequest,
11     {params} : {params : Promise<{ id : string}>}) {
12     try {
13
14         await connectToDatabase();
15         const {id} = await params
16         const checkAdmin = await Admin.findById( id ).select('-
    password -token ');
17         if (!checkAdmin) {
18             return NextResponse.json({ status: 404, message: '
    Admin Not Found' }, { status: 404 });
19         }
20         return NextResponse.json({ status: 200, message: 'admin
    get successfully', data: checkAdmin }, { status: 200 });
21     } catch (error) {
```

```
22          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
23      }
24  }
25
26  export async function PUT(
27      req: NextRequest,
28      {params} : {params : Promise<{ id : string}>}) {
29      try {
30          await connectToDatabase();
31          const formData = await req.formData();
32          const {id} = await params
33          const checkAdmin = await Admin.findById( id );
34          if (!checkAdmin) {
35              return NextResponse.json({ status: 404, message: '
        Admin Not Found' }, { status: 404 });
36          }
37          const { username, password, fullName, position,
        description, image } = await parseJson (formData);
38          const encryptPassword =  await bcryptjs.hash(password as
        string, 10);
39          if (image) {
40              await deleteFile(checkAdmin.image, 'admin');
41              const imageUrl = await uploadFile(formData, 'admin', '
        image');
42              const admin = await Admin.findByIdAndUpdate(id, {
        username, password: encryptPassword, fullName, position,
        description, image : imageUrl }, { new: true });
43              return NextResponse.json({ status: 200, message: '
        admin updated successfully', data: admin }, { status: 200 });
44
45          }
46          const admin = await Admin.findByIdAndUpdate(id, { username
        , password: encryptPassword, fullName, position, description },
         { new: true });
47          return NextResponse.json({ status: 200, message: 'admin
        updated successfully', data: admin }, { status: 200 });
48      } catch (error) {
49          return NextResponse.json({ status: 500, message: 'Internal
         Server Error' }, { status: 500 });
50      }
51  }
52
53  export async function DELETE(
54      _req: NextRequest,
55      {params} : {params : Promise<{ id : string}>}) {
56      try {
57          await connectToDatabase();
58          const {id} = await params
59          const checkAdmin = await Admin.findById( id );
60          if (!checkAdmin) {
61              return NextResponse.json({ status: 404, message: '
        Admin Not Found' }, { status: 404 });
62          }
63          await deleteFile(checkAdmin.image, 'admin');
64          await Admin.findByIdAndDelete( id );
65          return NextResponse.json({ status: 200, message: 'admin
        deleted successfully' }, { status: 200 });
66      } catch (error) {
67          return NextResponse.json({ status: 500, message: 'Internal
         Server Error' }, { status: 500 });
68      }
69  }
```

Kode 3.2: Kode untuk `api/admin/[id]`

Methods

- `GET` : Mengambil data *admin* berdasarkan ID.
- `PUT` : Memperbarui data *admin* berdasarkan ID.
- `DELETE` : Menghapus data *admin* berdasarkan ID.



Gambar 3.2. Tampilan antarmuka *dashboard* untuk *endpoint Admin*

Sumber: Five Elements Agency

A.2 AboutUs

`api/aboutUs`

Endpoint ini digunakan untuk mengambil seluruh data About Us dengan dukungan fitur pencarian, *pagination*, serta opsi pengambilan seluruh data (*fetch-all*).

```
import { NextRequest, NextResponse } from "next/server";
import Admin from "@/common/schemas/dbSchema/admin";
import { connectToDatabase } from "@/lib/mongodb";

export async function GET(req: NextRequest) {
    try {
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true';
        const skip = (page - 1) * limit;
        let filter;
        let checkAdmins;

        await connectToDatabase();
        let totalData = await Admin.countDocuments();

        if (search) {
            filter = [
                { fullName: { $regex: search, $options: 'i' } },
                { position: { $regex: search, $options: 'i' } },
                { description: { $regex: search, $options: 'i' } },
            ];
            checkAdmins = await Admin.find({ $or: filter })
                .select('-username -password -token')
                .sort({ createdAt: -1 })
                .skip(skip)
                .limit(limit);
            totalData = checkAdmins.length;
        } else if (all) {
            checkAdmins = await Admin.find()
                .select('-username -password -token')
                .sort({ createdAt: -1 });
        } else {
            checkAdmins = await Admin.find()
                .select('-username -password -token')
                .sort({ createdAt: -1 })
                .skip(skip)
                .limit(limit);
        }

        const result = {
            admins: checkAdmins,
            pagination: {
                page,
                limit,
                totalPages: Math.ceil(totalData / limit),
                totalData,
            },
        };

        return NextResponse.json({ status: 200, message: 'admin get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
```

```
    }
}
```

Kode 3.3: Kode untuk `api/aboutUs`

Methods

- `GET` : Mengambil semua data AboutUs, dengan fitur pencarian, *pagination*, dan *fetch-all*.

`api/aboutUs/[id]`

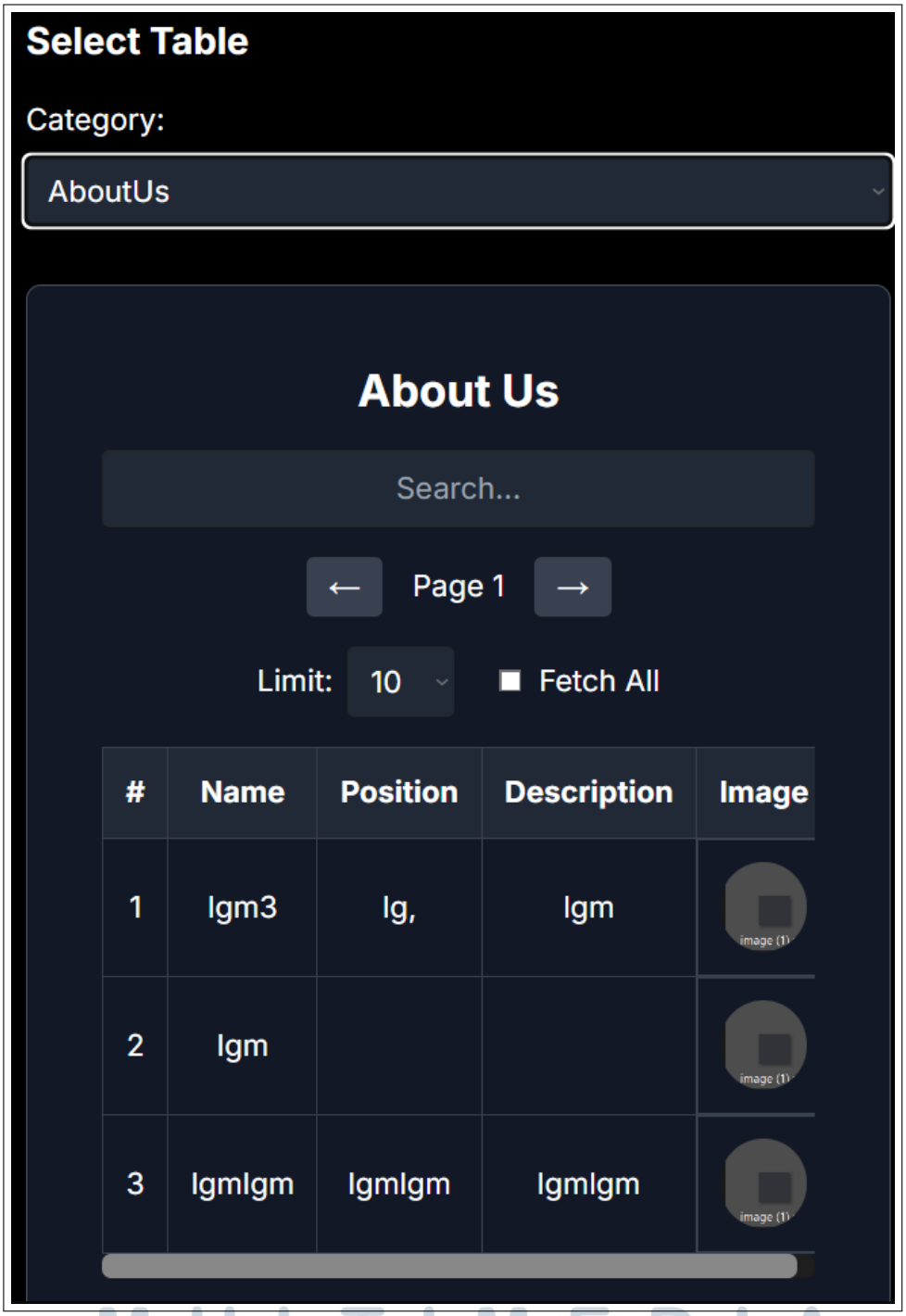
Endpoint ini digunakan untuk mengambil data AboutUs berdasarkan ID tertentu.

```
import { NextRequest, NextResponse } from "next/server";
import Admin from "@/common/schemas/dbSchema/admin";
import { connectToDatabase } from "@/lib/mongodb";

export async function GET(
    _req: NextRequest,
    { params }: { params: Promise<{ id: string }> }
) {
    try {
        await connectToDatabase();
        const { id } = await params;
        const checkAdmin = await Admin.findById(id)
            .select('-username -password -token');
        if (!checkAdmin) {
            return NextResponse.json({ status: 404, message: '
    Admin Not Found' }, { status: 404 });
        }
        return NextResponse.json({ status: 200, message: 'admin
    get successfully', data: checkAdmin }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
    }
}
```

Kode 3.4: Kode untuk `api/aboutUs/[id]`

Methods

- `GET` : Mengambil satu data AboutUs berdasarkan ID.



Gambar 3.3. Tampilan antarmuka *dashboard* untuk *endpoint AboutUs* Sumber: Five Elements Agency

A.3 Awards

`api/awards/protected`

Endpoint ini digunakan untuk mengambil semua data *Awards* atau menambahkan entri *Awards* baru

ke dalam *database*. *Endpoint* ini mendukung pencarian, *pagination*, dan unggah file gambar melalui *form-data*.

```
import { NextRequest, NextResponse } from "next/server";
import Award from "@/common/schemas/dbSchema/award";
import { connectToDatabase } from "@/lib/mongodb";
import { uploadFile } from "@/lib/upload";
import { parseJson } from '@/lib/parseJson';
import Admin from "@/common/schemas/dbSchema/admin";

export async function GET(req: NextRequest) {
    try {
        await connectToDatabase();
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true';
        const skip = (page - 1) * limit;

        let filter;
        let checkAward;
        let totalData = await Award.countDocuments();

        if (search) {
            filter = [
                { name: { $regex: search, $options: 'i' } },
                { addedBy: { $regex: search, $options: 'i' } },
                { description: { $regex: search, $options: 'i' } },
            ];
            checkAward = await Award.find({ $or: filter })
                .sort({ createdAt: -1 })
                .skip(skip)
                .limit(limit);
            totalData = checkAward.length;
        } else if (all) {
            checkAward = await Award.find().sort({ createdAt: -1 });
        } else {
            checkAward = await Award.find()
                .sort({ createdAt: -1 })
                .skip(skip)
                .limit(limit);
        }

        const result = {
            Awards: checkAward,
            pagination: {
                page,
                limit,
                totalPages: Math.ceil(totalData / limit),
                totalData,
            },
        };

        return NextResponse.json({ status: 200, message: 'Award get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function POST(req: NextRequest) {
```

```
    try {
        await connectToDatabase();
        const userData = req.headers.get("x-user-data");
        const { id } = JSON.parse(userData as any);
        const formData = await req.formData();
        const { name, description, image, isHidden } = await
    parseJson(formData);

        if (!name || !description || !image) {
            return NextResponse.json({ status: 400, message: 'Bad
    Request' }, { status: 400 });
        }

        const checkAward = await Award.findOne({ name });
        if (checkAward) {
            return NextResponse.json({ status: 409, message: '
    Award already exists' }, { status: 409 });
        }

        const admin = await Admin.findById({ _id: id }).select('
    fullName');
        const imageUrl = await uploadFile(formData, 'award', '
    image');

        const award = await Award.create({
            name,
            description,
            addedBy: admin.fullName,
            image: imageUrl,
            isHidden,
        });

        return NextResponse.json({ status: 201, message: 'Award
    added successfully', data: award }, { status: 201 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
    }
}
```

Kode 3.5: Kode untuk `api/awards/protected`

Methods

- `GET` : Mengambil semua data Awards dengan dukungan pencarian, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan entri Awards baru. Wajib mengirimkan data `name`, `description`, dan `image` dalam bentuk `multipart/form-data`.

`api/awards/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu entri Awards berdasarkan ID.

```
import { NextRequest, NextResponse } from "next/server";
import Award from "@/common/schemas/dbSchema/award";
import { connectToDatabase } from "@/lib/mongodb";
import bcryptjs from 'bcryptjs';
import {uploadFile,  deleteFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'


export async function GET(
    _req: NextRequest,
```

```
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const {id} = await params
        const checkAward = await Award.findById( id );
        if (!checkAward) {
            return NextResponse.json({ status: 404, message: 'Award Not Found' }, { status: 404 });
        }
        return NextResponse.json({ status: 200, message: 'Award get successfully', data: checkAward }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function PUT(
    req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const formData = await req.formData();
        const {id} = await params
        const checkAward = await Award.findById( id );
        if (!checkAward) {
            return NextResponse.json({ status: 404, message: 'Award Not Found' }, { status: 404 });
        }
        const { name, description, isHidden,  image      } = await parseJson(formData);
        if (image) {
            await deleteFile(checkAward.image, 'award');
            const imageUrl = await uploadFile(formData, 'award', 'image');
            const award = await Award.findByIdAndUpdate(id, {name, description, isHidden, image : imageUrl }, { new: true });
            return NextResponse.json({ status: 200, message: 'Award updated successfully', data: award }, { status: 200 });

        }
        const award = await Award.findByIdAndUpdate(id, { name, description, isHidden }, { new: true });
        return NextResponse.json({ status: 200, message: 'Award updated successfully', data: award }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function DELETE(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const {id} = await params
        const checkAward = await Award.findById( id );
        if (!checkAward) {
            return NextResponse.json({ status: 404, message: 'Award Not Found' }, { status: 404 });
        }
        await deleteFile(checkAward.image, 'award');
        await Award.findByIdAndDelete( id );
        return NextResponse.json({ status: 200, message: 'Award deleted successfully' }, { status: 200 });
    } catch (error) {
```

```
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}
```

Kode 3.6: Kode untuk `api/awards/protected/[id]`

Methods

- `GET` : Mengambil semua data *Blog* dengan dukungan *search*, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan entri *Blog* baru. Wajib mengirimkan data `title`, `content`, dan `image` dalam bentuk `multipart/form-data`.

`api/blog/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu entri *Blog* berdasarkan ID.

```
import { NextRequest, NextResponse } from "next/server";
import Blog from "@/common/schemas/dbSchema/blog";
import { connectToDatabase } from "@/lib/mongodb";
import { uploadFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'
import Admin from "@/common/schemas/dbSchema/admin";

export async function GET(req: NextRequest) {
    try {
        await connectToDatabase();
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true' ? true : false;
        const skip = (page - 1) * limit;
        let filter
        let checkBlog;
        let totalData = await Blog.countDocuments();
        if (search) {
          filter = [
            { title: { $regex: search, $options: 'i' } },
            { addedBy: { $regex: search, $options: 'i' } },
            { description: { $regex: search, $options: 'i' } },
          ]
          checkBlog = await Blog.find({
            $or: filter
          }).sort({ createdAt: -1 }).skip(skip).limit(limit);
          totalData = checkBlog.length;
        } else if (all) {
            checkBlog = await Blog.find().sort({ createdAt: -1 });
        } else {
            checkBlog = await Blog.find().sort({ createdAt: -1 }).skip(skip).limit(limit);
        }
        const result = {
            Blogs: checkBlog,
            pagination: {
                page: page,
                limit: limit,
                totalPages: Math.ceil(  totalData / limit),
                totalData: totalData
            }
        }
```

```
        return NextResponse.json({ status: 200, message: 'Blog get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function POST(req: NextRequest) {
    try {
        await connectToDatabase();
        const userData = req.headers.get("x-user-data");
        const { id } = JSON.parse(userData as any);
        const formData = await req.formData();
        const { title, link, publishedDate, description, image, isHidden      } = await parseJson(formData);
        if (!title || !link || !publishedDate || !description || !image) return NextResponse.json({ status: 400, message: 'Bad Request' }, { status: 400 });
        const checkBlog = await Blog.findOne({ title: title });
        if (checkBlog) {
            return NextResponse.json({ status: 409, message: 'Blog already exists' }, { status: 409 });
        }
        const admin = await Admin.findById({ _id: id }).select('fullName');
        const imageUrl = await uploadFile(formData, 'blog', 'image');
        const blog = await Blog.create({ title, link, publishedDate, description, addedBy : admin.fullName, image : imageUrl, isHidden });
        return NextResponse.json({ status: 201, message: 'Blog added successfully', data: blog }, { status: 201 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}
```

Kode 3.7: Kode untuk `api/blog/protected`

Methods

- `GET` : Mengambil semua data Blog dengan dukungan pencarian, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan entri Blog baru. Wajib mengirimkan data `title`, `content`, dan `image` dalam bentuk `multipart/form-data`.

`api/blog/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu entri Blog berdasarkan ID.

```
import { NextRequest, NextResponse } from "next/server";
import Blog from "@/common/schemas/dbSchema/blog";
import { connectToDatabase } from "@/lib/mongodb";
import bcryptjs from 'bcryptjs';
import {uploadFile,  deleteFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'


export async function GET(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
```

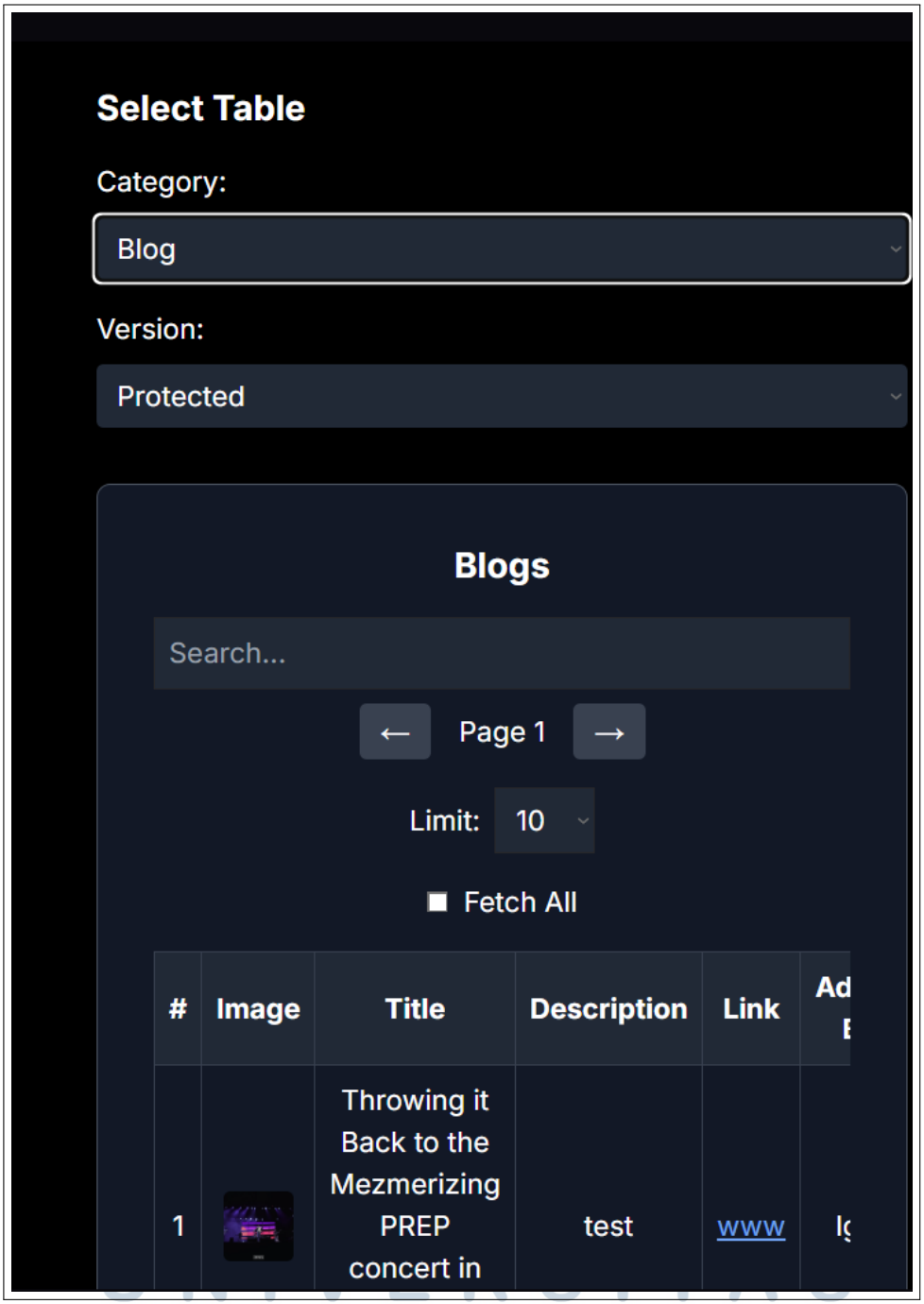
```
12      try {
13          await connectToDatabase();
14          const {id} = await params
15          const checkBlog = await Blog.findById( id );
16          if (!checkBlog) {
17              return NextResponse.json({ status: 404, message: 'Blog
        Not Found' }, { status: 404 });
18          }
19          return NextResponse.json({ status: 200, message: 'Blog get
        successfully', data: checkBlog }, { status: 200 });
20      } catch (error) {
21          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
22      }
23  }
24
25  export async function PUT(
26      req: NextRequest,
27      {params} : {params : Promise<{ id : string}>}) {
28      try {
29          await connectToDatabase();
30          const formData = await req.formData();
31          const {id} = await params
32          const checkBlog = await Blog.findById( id );
33          if (!checkBlog) {
34              return NextResponse.json({ status: 404, message: 'Blog
        Not Found' }, { status: 404 });
35          }
36          const { title, link, publishedDate, description, isHidden,
         image       } = await parseJson(formData);
37          if (image) {
38              await deleteFile(checkBlog.image, 'blog');
39              const imageUrl = await uploadFile(formData, 'blog', '
       image');
40              const blog = await Blog.findByIdAndUpdate(id, {title,
       link, publishedDate, description, isHidden, image : imageUrl },
        { new: true });
41              return NextResponse.json({ status: 200, message: 'Blog
        updated successfully', data: blog }, { status: 200 });
42
43          }
44          const blog = await Blog.findByIdAndUpdate(id, { title,
       link, publishedDate,  description, isHidden }, { new: true });
45          return NextResponse.json({ status: 200, message: 'Blog
       updated successfully', data: blog }, { status: 200 });
46      } catch (error) {
47          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
48      }
49  }
50
51  export async function DELETE(
52      _req: NextRequest,
53      {params} : {params : Promise<{ id : string}>}) {
54      try {
55          await connectToDatabase();
56          const {id} = await params
57          const checkBlog = await Blog.findById( id );
58          if (!checkBlog) {
59              return NextResponse.json({ status: 404, message: 'Blog
        Not Found' }, { status: 404 });
60          }
61          await deleteFile(checkBlog.image, 'blog');
62          await Blog.findByIdAndDelete( id );
63          return NextResponse.json({ status: 200, message: 'Blog
       deleted successfully' }, { status: 200 });
64      } catch (error) {
```

```
65          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
66      }
67  }
```

Kode 3.8: Kode untuk `api/blog/protected/[id]`

Methods

- `GET` : Mengambil satu entri Blog berdasarkan ID.
- `PUT` : Memperbarui entri Blog berdasarkan ID.
- `DELETE` : Menghapus entri Blog berdasarkan ID.



Gambar 3.4. Tampilan antarmuka *dashboard* untuk *endpoint Blog* Sumber: Five Elements Agency

A.4 Client

`api/client/protected`

Endpoint ini digunakan untuk mengambil semua data Client atau menambahkan entri Client baru ke dalam database. *Endpoint* ini mendukung pencarian, pagination, dan unggah file logo melalui

form-data.

```
import { NextRequest, NextResponse } from "next/server";
import Client from "@/common/schemas/dbSchema/client";
import { connectToDatabase } from "@/lib/mongodb";
import bcryptjs from 'bcryptjs';
import {uploadFile,  deleteFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'


export async function GET(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const {id} = await params
        const checkClient = await Client.findById( id );
        if (!checkClient) {
                return NextResponse.json({ status: 404, message: 'Client Not Found' }, { status: 404 });
        }
        return NextResponse.json({ status: 200, message: 'Client get successfully', data: checkClient }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function PUT(
    req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const formData = await req.formData();
        const {id} = await params
        const checkClient = await Client.findById( id );
        if (!checkClient) {
                return NextResponse.json({ status: 404, message: 'Client Not Found' }, { status: 404 });
        }
        const { name, title, review, description, isHidden,  image } = await parseJson(formData);
        if (image) {
            await deleteFile(checkClient.image, 'client');
            const imageUrl = await uploadFile(formData, 'client', 'image');
            const client = await Client.findByIdAndUpdate(id, { name, title, review, description, isHidden, image : imageUrl }, { new: true });
                return NextResponse.json({ status: 200, message: 'Client updated successfully', data: client }, { status: 200 });

        }
        const client = await Client.findByIdAndUpdate(id, { name, title, review, description, isHidden }, { new: true });
        return NextResponse.json({ status: 200, message: 'Client updated successfully', data: client }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function DELETE(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
```

```
    try {
        await connectToDatabase();
        const {id} = await params
        const checkClient = await Client.findById( id );
        if (!checkClient) {
                return NextResponse.json({ status: 404, message: '
    Client Not Found' }, { status: 404 });
        }
        await deleteFile(checkClient.image, 'client');
        await Client.findByIdAndDelete( id );
        return NextResponse.json({ status: 200, message: 'Client
    deleted successfully' }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
    }
}
```

Kode 3.9: Kode untuk `api/client/protected`

Methods

- `GET` : Mengambil semua data Client dengan dukungan pencarian, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan entri Client baru. Wajib mengirimkan data `name` dan `image` dalam bentuk `multipart/form-data`.

`api/client/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu entri Client berdasarkan ID.

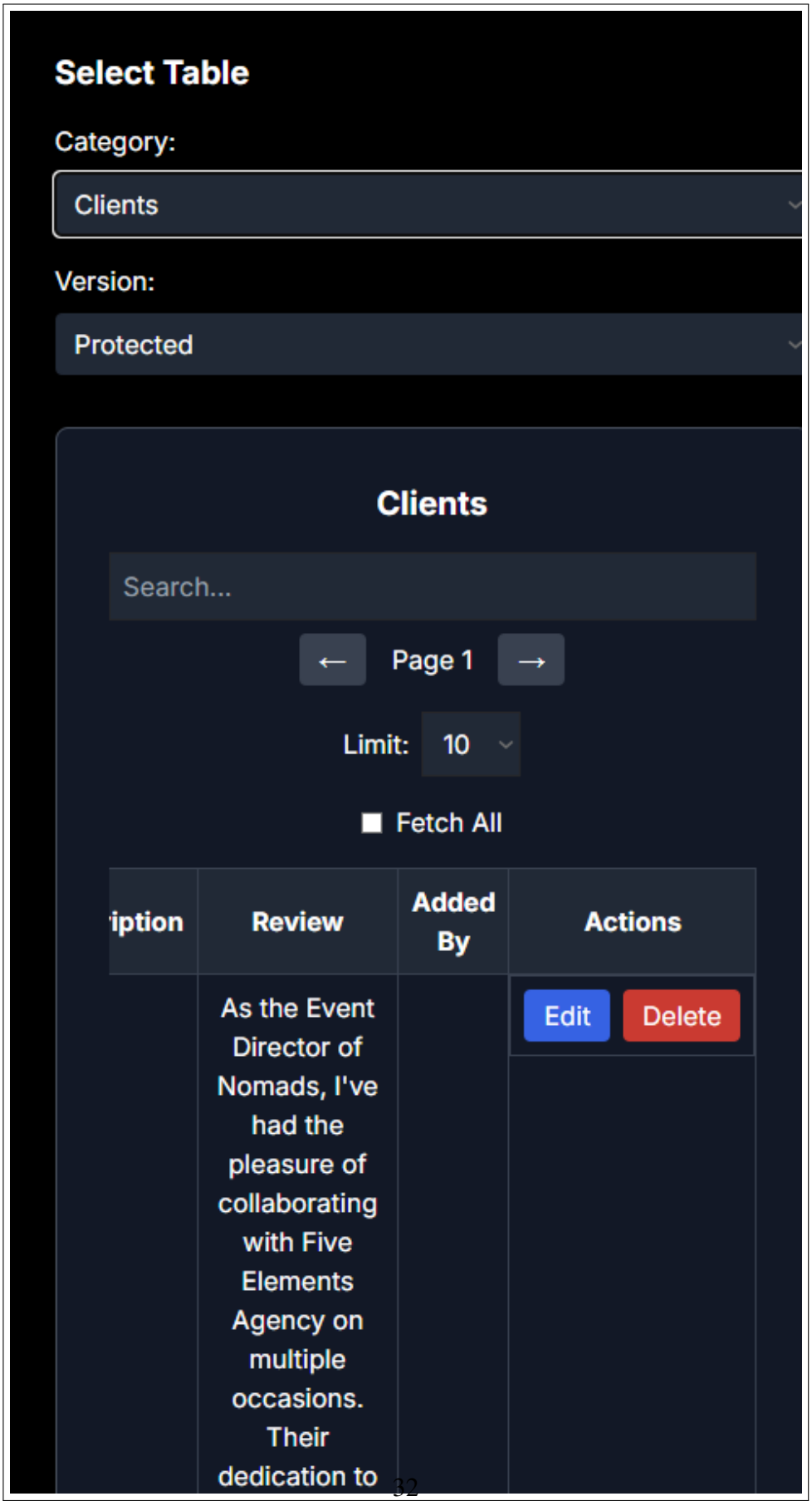
```
import { NextRequest, NextResponse } from "next/server";
import Client from "@/common/schemas/dbSchema/client";
import { connectToDatabase } from "@/lib/mongodb";
import bcryptjs from 'bcryptjs';
import {uploadFile,  deleteFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'


export async function GET(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const {id} = await params
        const checkClient = await Client.findById( id );
        if (!checkClient) {
                return NextResponse.json({ status: 404, message: '
    Client Not Found' }, { status: 404 });
        }
        return NextResponse.json({ status: 200, message: 'Client
    get successfully', data: checkClient }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
    }
}

export async function PUT(
    req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
```

```
30          const formData = await req.formData();
31          const {id} = await params
32          const checkClient = await Client.findById( id );
33          if (!checkClient) {
34                  return NextResponse.json({ status: 404, message: '
    Client Not Found' }, { status: 404 });
35          }
36          const { name, title, review, description, isHidden,  image
              } = await parseJson(formData);
37          if (image) {
38              await deleteFile(checkClient.image, 'client');
39              const imageUrl = await uploadFile(formData, 'client',
    'image');
40              const client = await Client.findByIdAndUpdate(id, {
    name, title, review, description, isHidden, image : imageUrl },
     { new: true });
41                  return NextResponse.json({ status: 200, message: '
    Client updated successfully', data: client }, { status: 200 });
42
43          }
44          const client = await Client.findByIdAndUpdate(id, { name,
    title, review, description, isHidden }, { new: true });
45          return NextResponse.json({ status: 200, message: 'Client
    updated successfully', data: client }, { status: 200 });
46      } catch (error) {
47          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
48      }
49  }
50
51  export async function DELETE(
52      _req: NextRequest,
53      {params} : {params : Promise<{ id : string}>}) {
54      try {
55          await connectToDatabase();
56          const {id} = await params
57          const checkClient = await Client.findById( id );
58          if (!checkClient) {
59                  return NextResponse.json({ status: 404, message: '
    Client Not Found' }, { status: 404 });
60          }
61          await deleteFile(checkClient.image, 'client');
62          await Client.findByIdAndDelete( id );
63          return NextResponse.json({ status: 200, message: 'Client
    deleted successfully' }, { status: 200 });
64      } catch (error) {
65          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
66      }
67  }
```

Kode 3.10: Kode untuk `api/client/protected/[id]`

Methods

- `GET` : Mengambil satu entri Client berdasarkan ID.
- `PUT` : Memperbarui entri Client berdasarkan ID.
- `DELETE` : Menghapus entri Client berdasarkan ID.



Gambar 3.5. Tampilan antarmuka *dashboard* untuk *endpoint Client* Sumber: Five Elements Agency

A.5 Package

`api/packages/protected`

Endpoint ini digunakan untuk mengambil semua data Package atau menambahkan entri Package baru ke dalam database. *Endpoint* ini mendukung pencarian, pagination, dan unggah file melalui form-data.

```
import { NextRequest, NextResponse } from "next/server";
import Package from "@/common/schemas/dbSchema/packages";
import { connectToDatabase } from "@/lib/mongodb";
import { uploadFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'
import Admin from "@/common/schemas/dbSchema/admin";

export async function GET(req: NextRequest) {
    try {
        await connectToDatabase();
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true' ? true : false;
        const skip = (page - 1) * limit;
        let filter
        let checkPackage;
        let totalData = await Package.countDocuments();
        if (search) {
          filter = [
            { name: { $regex: search, $options: 'i' } },
            { addedBy: { $regex: search, $options: 'i' } },
            { description: { $regex: search, $options: 'i' } },
          ]
          checkPackage = await Package.find({
            $or: filter
          }).sort({ createdAt: -1 }).skip(skip).limit(limit);
          totalData = checkPackage.length;
        } else if (all) {
            checkPackage = await Package.find().sort({ createdAt: -1 });
        } else {
            checkPackage = await Package.find().sort({ createdAt: -1 }).skip(skip).limit(limit);
        }
        const result = {
            Packages: checkPackage,
            pagination: {
                page: page,
                limit: limit,
                totalPages: Math.ceil(  totalData / limit),
                totalData: totalData
            }
        }
        return NextResponse.json({ status: 200, message: 'Package get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function POST(req: NextRequest) {
    try {
        await connectToDatabase();
```

```
        const userData = req.headers.get("x-user-data");
        const { id } = JSON.parse(userData as any);
        const formData = await req.formData();
        const { name, description, summary, isHidden, banner, graphic      } = await parseJson(formData);
        if (!name || !description || !summary || !banner || !graphic ) return NextResponse.json({ status: 400, message: 'Bad Request' }, { status: 400 });
        const checkPackage = await Package.findOne({ name: name });
        if (checkPackage) {
                return NextResponse.json({ status: 409, message: 'Package already exists' }, { status: 409 });
        }
        const admin = await Admin.findById({ _id: id }).select('fullName');
        const graphicUrl = await uploadFile(formData, 'package', 'graphic');
        const bannerUrl = await uploadFile(formData, 'package', 'banner');
        const packages = await Package.create({ name, summary, description, addedBy : admin.fullName, banner : bannerUrl, graphic : graphicUrl , isHidden });
        return NextResponse.json({ status: 201, message: 'Package added successfully', data: packages }, { status: 201 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}
```

Kode 3.11: Kode untuk `api/packages/protected`

Methods

- `GET` : Mengambil semua data Package dengan dukungan pencarian, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan entri Package baru. Wajib mengirimkan data `name`, `price`, dan `features` dalam bentuk `multipart/form-data`.

`api/packages/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu entri Package berdasarkan ID.

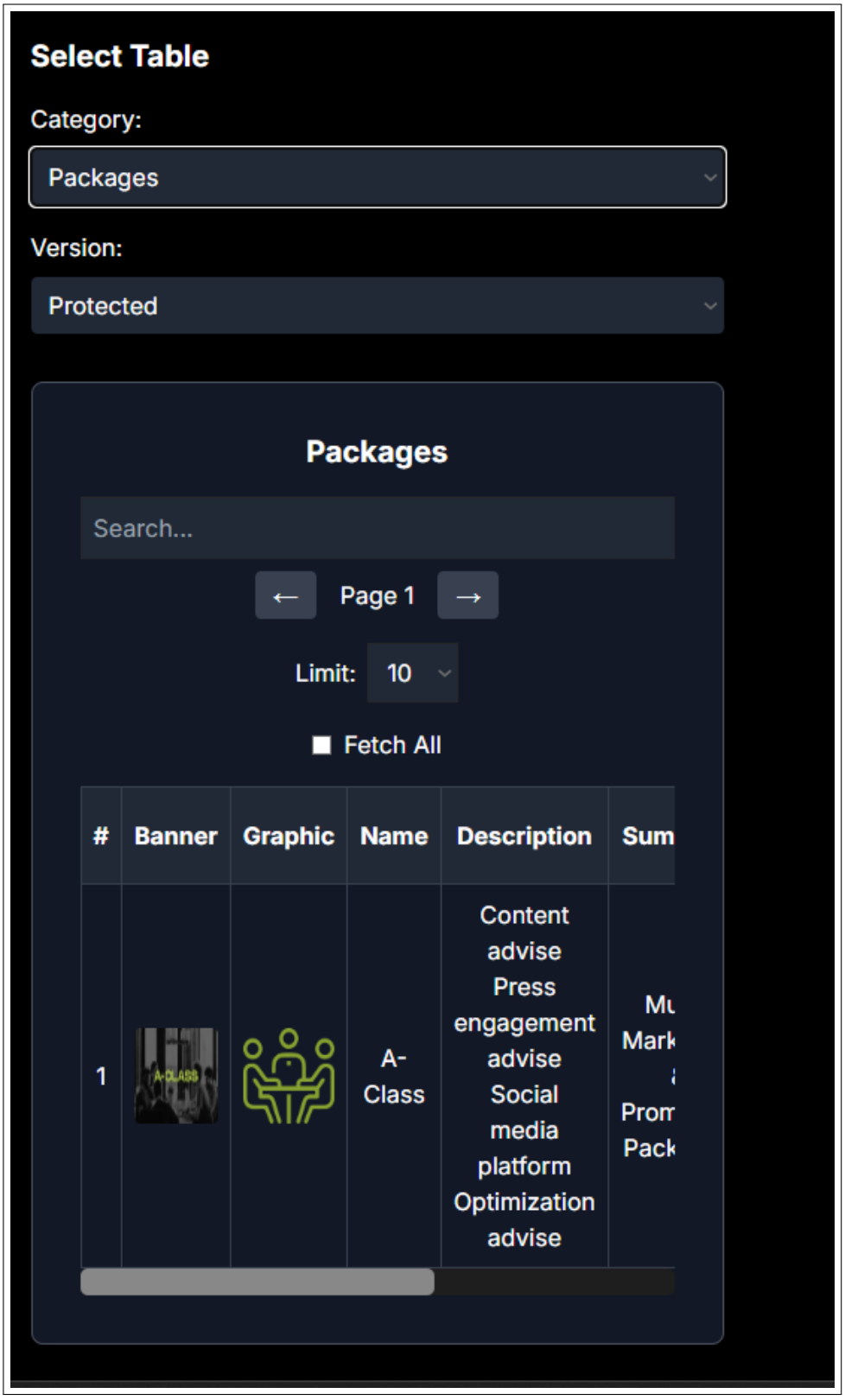
```
import { NextRequest, NextResponse } from "next/server";
import Package from "@/common/schemas/dbSchema/packages";
import { connectToDatabase } from "@/lib/mongodb";
import bcryptjs from 'bcryptjs';
import {uploadFile,  deleteFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'


export async function GET(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const {id} = await params
        const checkPackage = await Package.findById( id );
        if (!checkPackage) {
                return NextResponse.json({ status: 404, message: 'Package Not Found' }, { status: 404 });
        }
```

```
19          return NextResponse.json({ status: 200, message: 'Package
    get successfully', data: checkPackage}, { status: 200 });
20      } catch (error) {
21          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
22      }
23  }
24
25  export async function PUT(
26      req: NextRequest,
27      {params} : {params : Promise<{ id : string}>}) {
28      try {
29          await connectToDatabase();
30          const formData = await req.formData();
31          const {id} = await params
32          const checkPackage = await Package.findById( id );
33          if (!checkPackage) {
34                  return NextResponse.json({ status: 404, message: '
    Package Not Found' }, { status: 404 });
35          }
36          const { name, summary, description, isHidden,  banner,
    graphic  } = await parseJson(formData);
37          let bannerUrl;
38          let graphicUrl;
39          if (banner){
40              await deleteFile(checkPackage.banner, 'package');
41              bannerUrl = await uploadFile(formData, 'package', '
    banner');
42          }
43          if (graphic){
44              await deleteFile(checkPackage.graphic, 'package');
45              graphicUrl = await uploadFile(formData, 'package', '
    graphic');
46          }
47          const packages = await Package.findByIdAndUpdate(id, {
    name, summary, description, isHidden, banner : bannerUrl,
    graphic : graphicUrl }, { new: true });
48          return NextResponse.json({ status: 200, message: 'Package
    updated successfully', data: packages }, { status: 200 });
49      } catch (error) {
50          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
51      }
52  }
53
54  export async function DELETE(
55      _req: NextRequest,
56      {params} : {params : Promise<{ id : string}>}) {
57      try {
58          await connectToDatabase();
59          const {id} = await params
60          const checkPackage = await Package.findById( id );
61          if (!checkPackage) {
62                  return NextResponse.json({ status: 404, message: '
    Package Not Found' }, { status: 404 });
63          }
64          await deleteFile(checkPackage.graphic, 'package');
65          await deleteFile(checkPackage.banner, 'package');
66          await Package.findByIdAndDelete( id );
67          return NextResponse.json({ status: 200, message: 'Package
    deleted successfully' }, { status: 200 });
68      } catch (error) {
69          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
70      }
71  }
```

Kode 3.12: Kode untuk `api/packages/protected/[id]`

Methods

- `GET` : Mengambil satu entri Package berdasarkan ID.
- `PUT` : Memperbarui entri Package berdasarkan ID.
- `DELETE` : Menghapus entri Package berdasarkan ID.



Gambar 3.6. Tampilan antarmuka *dashboard* untuk *endpoint Package* Sumber: Five Elements Agency

A.6 Partner

`api/partners/protected`

Endpoint ini digunakan untuk mengambil semua data Partner atau menambahkan entri Partner baru ke dalam database. *Endpoint* ini mendukung pencarian, pagination, dan unggah file logo melalui form-data.

```
import { NextRequest, NextResponse } from "next/server";
import Partner from "@/common/schemas/dbSchema/partner";
import { connectToDatabase } from "@/lib/mongodb";
import { uploadFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'
import Admin from "@/common/schemas/dbSchema/admin";

export async function GET(req: NextRequest) {
    try {
        await connectToDatabase();
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true' ? true : false;
        const skip = (page - 1) * limit;
        let filter
        let checkPartner;
        let totalData = await Partner.countDocuments();
        if (search) {
          filter = [
            { name: { $regex: search, $options: 'i' } },
            { addedBy: { $regex: search, $options: 'i' } },
            { description: { $regex: search, $options: 'i' } },
          ]
          checkPartner = await Partner.find({
            $or: filter
          }).sort({ createdAt: -1 }).skip(skip).limit(limit);
          totalData = checkPartner.length;
        } else if (all) {
            checkPartner = await Partner.find().sort({ createdAt: -1 });
        } else {
            checkPartner = await Partner.find().sort({ createdAt: -1 }).skip(skip).limit(limit);
        }
        const result = {
            Partners: checkPartner,
            pagination: {
                page: page,
                limit: limit,
                totalPages: Math.ceil(  totalData / limit),
                totalData: totalData
            }
        }
        return NextResponse.json({ status: 200, message: 'Partner get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function POST(req: NextRequest) {
    try {
        await connectToDatabase();
```

```
53          const userData = req.headers.get("x-user-data");
54          const { id } = JSON.parse(userData as any);
55          const formData = await req.formData();
56          const { name, description, image, isHidden      } = await
    parseJson(formData);
57          if (!name || !description || !image) return NextResponse.
    json({ status: 400, message: 'Bad Request' }, { status: 400 });
58          const checkPartner = await Partner.findOne({ name: name })
    ;
59          if (checkPartner) {
60                  return NextResponse.json({ status: 409, message: '
    Partner already exists' }, { status: 409 });
61          }
62          const admin = await Admin.findById({ _id: id }).select('
    fullName');
63          const imageUrl = await uploadFile(formData, 'partner', '
    image');
64          const partner = await Partner.create({ name, description,
    addedBy : admin.fullName, image : imageUrl, isHidden });
65          return NextResponse.json({ status: 201, message: 'Partner
    added successfully', data: partner }, { status: 201 });
66      } catch (error) {
67          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
68      }
69  }
```

Kode 3.13: Kode untuk `api/partners/protected`

Methods

- `GET` : Mengambil semua data Partner dengan dukungan pencarian, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan entri Partner baru. Wajib mengirimkan data `name` dan `image` (logo) dalam format `multipart/form-data`.

`api/partners/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu entri Partner berdasarkan ID.

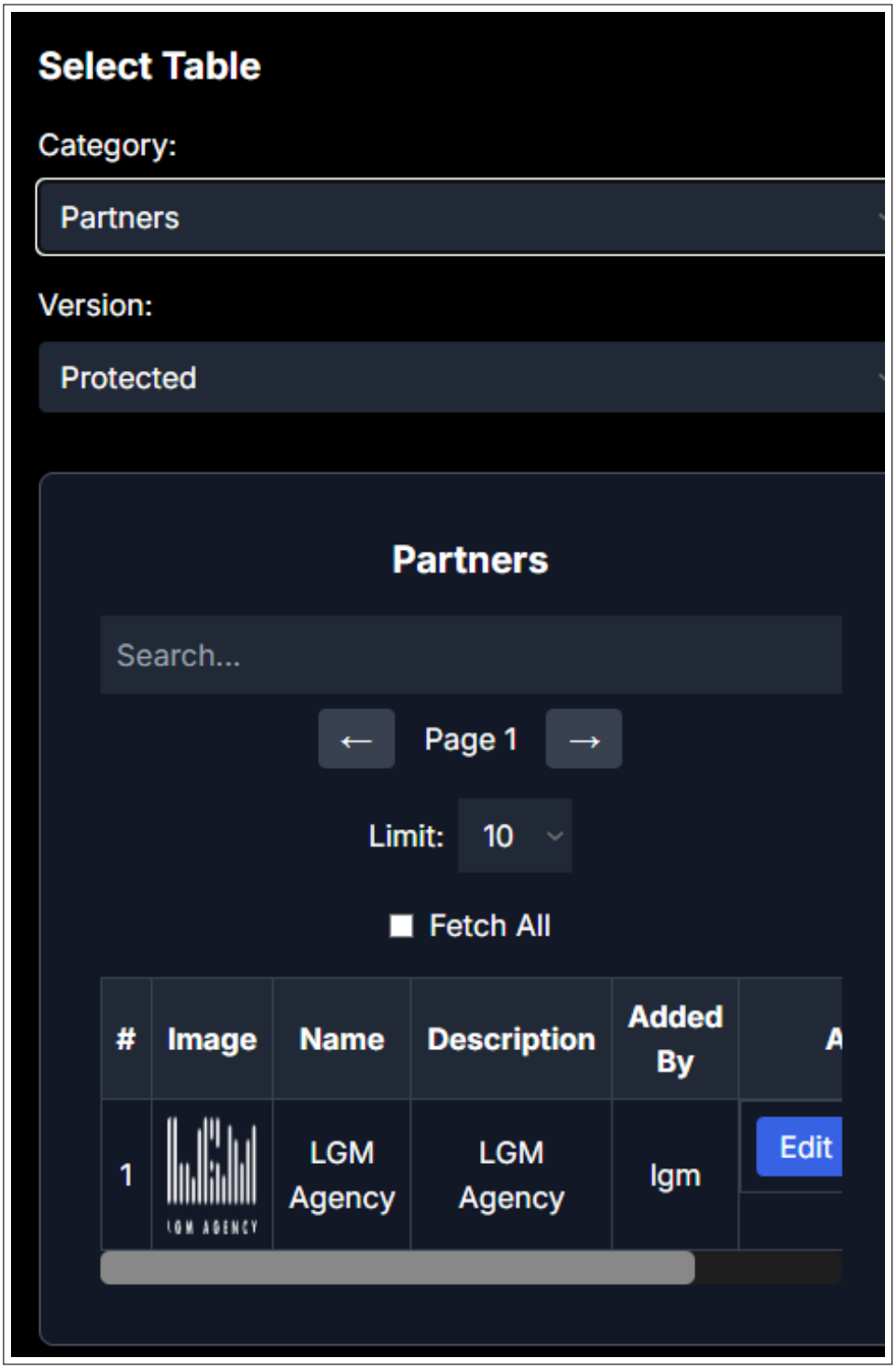
```
1  import { NextRequest, NextResponse } from "next/server";
2  import Partner from "@/common/schemas/dbSchema/partner";
3  import { connectToDatabase } from "@/lib/mongodb";
4  import bcryptjs from 'bcryptjs';
5  import {uploadFile,  deleteFile } from "@/lib/upload";
6  import {parseJson} from '@/lib/parseJson'
7
8
9  export async function GET(
10     _req: NextRequest,
11     {params} : {params : Promise<{ id : string}>}) {
12     try {
13         await connectToDatabase();
14         const {id} = await params
15         const checkPartner = await Partner.findById( id );
16         if (!checkPartner) {
17                 return NextResponse.json({ status: 404, message: '
    Partner Not Found' }, { status: 404 });
18         }
19         return NextResponse.json({ status: 200, message: 'Partner
    get successfully', data: checkPartner }, { status: 200 });
20     } catch (error) {
```

```
21          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
22      }
23  }
24
25  export async function PUT(
26      req: NextRequest,
27      {params} : {params : Promise<{ id : string}>}) {
28      try {
29          await connectToDatabase();
30          const formData = await req.formData();
31          const {id} = await params
32          const checkPartner = await Partner.findById( id );
33          if (!checkPartner) {
34                  return NextResponse.json({ status: 404, message: '
        Partner Not Found' }, { status: 404 });
35          }
36          const { name, description, isHidden,  image      } = await
        parseJson(formData);
37          if (image) {
38              await deleteFile(checkPartner.image, 'partner');
39              const imageUrl = await uploadFile(formData, 'partner',
        'image');
40              const partner = await Partner.findByIdAndUpdate(id, {
        name, description, isHidden, image : imageUrl }, { new: true })
        ;
41                  return NextResponse.json({ status: 200, message: '
        Partner updated successfully', data: partner }, { status: 200
        });
42
43          }
44          const partner = await Partner.findByIdAndUpdate(id, { name
        , description, isHidden }, { new: true });
45          return NextResponse.json({ status: 200, message: 'Partner
        updated successfully', data: partner }, { status: 200 });
46      } catch (error) {
47          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
48      }
49  }
50
51  export async function DELETE(
52      _req: NextRequest,
53      {params} : {params : Promise<{ id : string}>}) {
54      try {
55          await connectToDatabase();
56          const {id} = await params
57          const checkPartner = await Partner.findById( id );
58          if (!checkPartner) {
59                  return NextResponse.json({ status: 404, message: '
        Partner Not Found' }, { status: 404 });
60          }
61          await deleteFile(checkPartner.image, 'partner');
62          await Partner.findByIdAndDelete( id );
63          return NextResponse.json({ status: 200, message: 'Partner
        deleted successfully' }, { status: 200 });
64      } catch (error) {
65          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
66      }
67  }
```

Kode 3.14: Kode untuk `api/partners/protected/[id]`

Methods

- `GET` : Mengambil satu entri Partner berdasarkan ID.
- `PUT` : Memperbarui entri Partner berdasarkan ID.
- `DELETE` : Menghapus entri Partner berdasarkan ID.



Gambar 3.7. Tampilan antarmuka *dashboard* untuk *endpoint Partner* Sumber: Five Elements Agency

A.7 Project

`api/projects/protected`

Endpoint ini digunakan untuk mengambil semua data Project atau menambahkan entri Project baru ke dalam database. *Endpoint* ini mendukung pencarian, pagination, dan unggah file gambar melalui form-data.

```
import { NextRequest, NextResponse } from "next/server";
import Project from "@/common/schemas/dbSchema/project.ts";
import { connectToDatabase } from "@/lib/mongodb";
import { uploadFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'
import Admin from "@/common/schemas/dbSchema/admin";

export async function GET(req: NextRequest) {
    try {
        await connectToDatabase();
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true' ? true : false;
        const skip = (page - 1) * limit;
        let filter
        let checkProject;
        let totalData = await Project.countDocuments();
        if (search) {
          filter = [
            { name: { $regex: search, $options: 'i' } },
            { addedBy: { $regex: search, $options: 'i' } },
            { description: { $regex: search, $options: 'i' } },
          ]
          checkProject = await Project.find({
            $or: filter
          }).sort({ createdAt: -1 }).skip(skip).limit(limit);
          totalData = checkProject.length;
        } else if (all) {
            checkProject = await Project.find().sort({ createdAt: -1 });
        } else {
            checkProject = await Project.find().sort({ createdAt: -1 }).skip(skip).limit(limit);
        }
        const result = {
            Projects: checkProject,
            pagination: {
                page: page,
                limit: limit,
                totalPages: Math.ceil(  totalData / limit),
                totalData: totalData
            }
        }
        return NextResponse.json({ status: 200, message: 'Project get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function POST(req: NextRequest) {
    try {
        await connectToDatabase();
```

```
        const userData = req.headers.get("x-user-data");
        const { id } = JSON.parse(userData as any);
        const formData = await req.formData();
        const { name, description, image, isHidden      } = await parseJson(formData);
        if (!name || !description || !image) return NextResponse.json({ status: 400, message: 'Bad Request' }, { status: 400 });
        const checkProject = await Project.findOne({ name: name });
        if (checkProject) {
                return NextResponse.json({ status: 409, message: 'Project already exists' }, { status: 409 });
        }
        const admin = await Admin.findById({ _id: id }).select('fullName');
        const imageUrl = await uploadFile(formData, 'projecct', 'image');
        const project = await Project.create({ name, description, addedBy : admin.fullName, image : imageUrl, isHidden });
        return NextResponse.json({ status: 201, message: 'Project added successfully', data: project }, { status: 201 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}
```

Kode 3.15: Kode untuk `api/projects/protected`

Methods

- `GET` : Mengambil semua data Project dengan dukungan pencarian, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan entri Project baru. Wajib mengirimkan data `name`, `description`, dan `image` dalam format `multipart/form-data`.

`api/projects/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu entri Project berdasarkan ID.

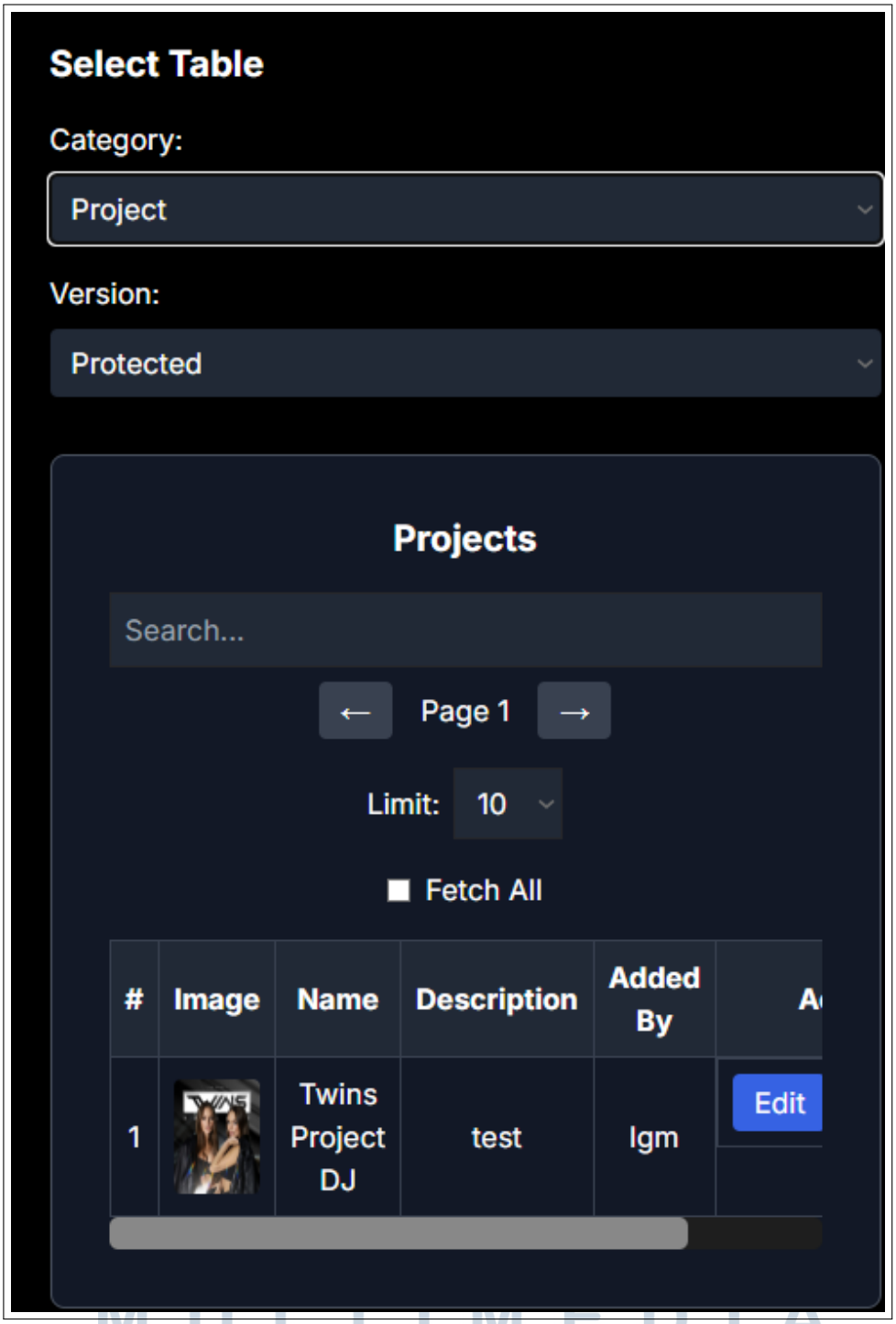
```
import { NextRequest, NextResponse } from "next/server";
import Project from "@/common/schemas/dbSchema/project.ts";
import { connectToDatabase } from "@/lib/mongodb";
import bcryptjs from 'bcryptjs';
import {uploadFile,  deleteFile } from "@/lib/upload";
import {parseJson} from '@/lib/parseJson'


export async function GET(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await connectToDatabase();
        const {id} = await params
        const checkProject = await Project.findById( id );
        if (!checkProject) {
                return NextResponse.json({ status: 404, message: 'Project Not Found' }, { status: 404 });
        }
        return NextResponse.json({ status: 200, message: 'Project get successfully', data: checkProject }, { status: 200 });
    } catch (error) {
```

```
21          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
22      }
23  }
24
25  export async function PUT(
26      req: NextRequest,
27      {params} : {params : Promise<{ id : string}>}) {
28      try {
29          await connectToDatabase();
30          const formData = await req.formData();
31          const {id} = await params
32          const checkProject = await Project.findById( id );
33          if (!checkProject) {
34                  return NextResponse.json({ status: 404, message: '
        Project Not Found' }, { status: 404 });
35          }
36          const { name, description, isHidden,  image      } = await
        parseJson(formData);
37          if (image) {
38              await deleteFile(checkProject.image, 'projecct');
39              const imageUrl = await uploadFile(formData, 'projecct
        ', 'image');
40              const project = await Project.findByIdAndUpdate(id, {
        name, description, isHidden, image : imageUrl }, { new: true })
        ;
41                  return NextResponse.json({ status: 200, message: '
        Project updated successfully', data: project }, { status: 200
        });
42
43          }
44          const project = await Project.findByIdAndUpdate(id, { name
        , description, isHidden }, { new: true });
45          return NextResponse.json({ status: 200, message: 'project
        updated successfully', data: project }, { status: 200 });
46      } catch (error) {
47          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
48      }
49  }
50
51  export async function DELETE(
52      _req: NextRequest,
53      {params} : {params : Promise<{ id : string}>}) {
54      try {
55          await connectToDatabase();
56          const {id} = await params
57          const checkProject = await Project.findById( id );
58          if (!checkProject) {
59                  return NextResponse.json({ status: 404, message: '
        Project Not Found' }, { status: 404 });
60          }
61          await deleteFile(checkProject.image, 'projecct');
62          await Project.findByIdAndDelete( id );
63          return NextResponse.json({ status: 200, message: 'project
        deleted successfully' }, { status: 200 });
64      } catch (error) {
65          return NextResponse.json({ status: 500, message: 'Internal
        Server Error' }, { status: 500 });
66      }
67  }
```

Kode 3.16: Kode untuk `api/projects/protected/[id]`

Methods

- `GET` : Mengambil satu entri Project berdasarkan ID.
- `PUT` : Memperbarui entri Project berdasarkan ID.
- `DELETE` : Menghapus entri Project berdasarkan ID.



Gambar 3.8. Tampilan antarmuka *dashboard* untuk *endpoint Project* Sumber: Five Elements Agency

A.8 Contact Us

`api/contactUs/create`

Menerima form isian dari halaman Contact Us dan menyimpannya ke `Inquiry`. Jika disetujui, email akan ditambahkan ke `Subscriber`.

```
import { NextRequest, NextResponse } from "next/server";
import Inquiry from "@/common/schemas/dbSchema/inquiry";
import { connectToDatabase } from "@/lib/mongodb";
import { uploadFile } from "@/lib/upload";
import { parseJson } from '@/lib/parseJson';
import Subscriber from "@/common/schemas/dbSchema/subscribers";

export async function POST(req: NextRequest) {
    try {
        await connectToDatabase();
        const formData = await req.formData();
        const {
            email, name, telephone,
            message, service, isOptInForSubscription
        } = await parseJson(formData);

        if (!email || !name || !telephone || !message || !service)
            return NextResponse.json({ status: 400, message: 'Bad
    Request' }, { status: 400 });

        const checkSubscriber = await Subscriber.findOne({ email:
    email });
        if (checkSubscriber) {
            if (isOptInForSubscription) {
                await Subscriber.create({ email: email });
            }
        }

        const inquiry = await Inquiry.create({ email, name,
    telephone, message, service });
        return NextResponse.json({
            status: 201,
            message: 'Inquiry added successfully',
            data: inquiry
        }, { status: 201 });

    } catch (error) {
        return NextResponse.json({
            status: 500,
            message: 'Internal Server Error'
        }, { status: 500 });
    }
}
```

Kode 3.17: Kode untuk `api/contactUs/create`

Methods

- `POST` : Kirim data inquiry. Bisa juga opt-in ke langganan email.

`api/contactUs/protected`

Ambil data inquiry dari database, bisa pakai pencarian, pagination, atau ambil semua.

```
import { NextRequest, NextResponse } from "next/server";
import Inquiry from "@/common/schemas/dbSchema/inquiry";
import { connectToDatabase } from "@/lib/mongodb";
```

```
4
5  export async function GET(req: NextRequest) {
6      try {
7          await connectToDatabase();
8          const query = req.nextUrl.searchParams;
9          const limit = parseInt(query.get("limit") as string) ||
       10;
10         const page = parseInt(query.get("page") as string) || 1;
11         const search = query.get("search") as string || '';
12         const all = query.get("all") === 'true';
13
14         const skip = (page - 1) * limit;
15         let checkInquiry;
16         let totalData = await Inquiry.countDocuments();
17
18         if (search) {
19             const filter = [
20                 { name: { $regex: search, $options: 'i' } },
21                 { message: { $regex: search, $options: 'i' } },
22                 { email: { $regex: search, $options: 'i' } },
23                 { service: { $regex: search, $options: 'i' } },
24             ];
25             checkInquiry = await Inquiry.find({ $or: filter })
26                 .sort({ createdAt: -1 }).skip(skip).limit(limit);
27             totalData = checkInquiry.length;
28         } else if (all) {
29             checkInquiry = await Inquiry.find().sort({ createdAt:
       -1 });
30         } else {
31             checkInquiry = await Inquiry.find().sort({ createdAt:
       -1 }).skip(skip).limit(limit);
32         }
33
34         const result = {
35             Inquirys: checkInquiry,
36             pagination: {
37                 page: page,
38                 limit: limit,
39                 totalPages: Math.ceil(totalData / limit),
40                 totalData: totalData
41             }
42         };
43
44         return NextResponse.json({
45             status: 200,
46             message: 'Inquiry get successfully',
47             data: result
48         }, { status: 200 });
49
50     } catch (error) {
51         return NextResponse.json({
52             status: 500,
53             message: 'Internal Server Error'
54         }, { status: 500 });
55     }
56 }
```

Kode 3.18: Kode untuk `api/contactUs/protected`

Methods

- `GET` : Ambil data inquiry, bisa pakai search, pagination, atau ambil semua.

`api/contactUs/protected/[id]`

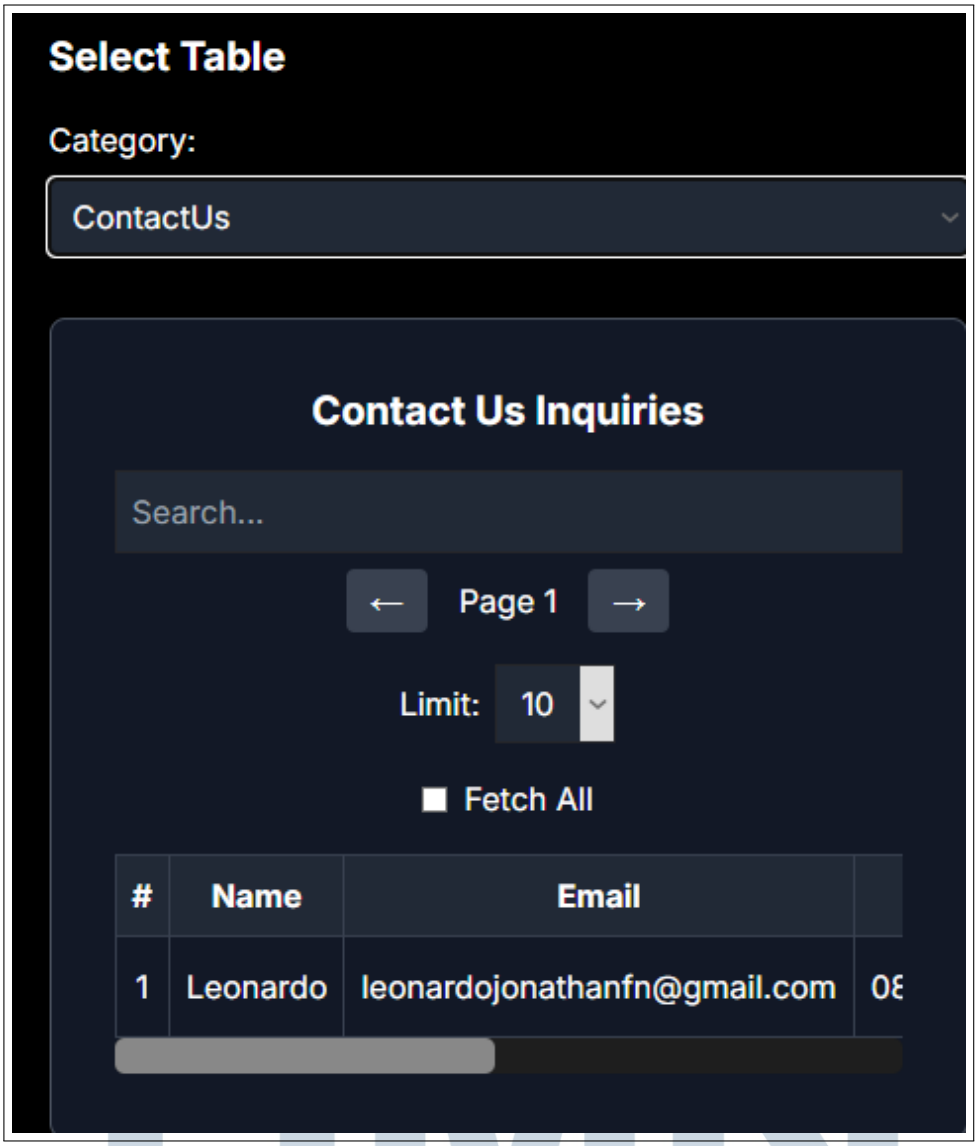
Ambil, edit, atau hapus data inquiry berdasarkan ID.

```
import { NextRequest, NextResponse } from "next/server";
import Inquiry from "@/common/schemas/dbSchema/inquiry";
import { connectToDatabase } from "@/lib/mongodb";
import { parseJson } from '@/lib/parseJson';

export async function GET(_req, {params}) {
    await connectToDatabase();
    const {id} = await params;
    const checkInquiry = await Inquiry.findById(id);
    if (!checkInquiry) {
        return NextResponse.json({ status: 404, message: 'Inquiry Not Found' }, { status: 404 });
    }
    return NextResponse.json({ status: 200, message: 'Inquiry get successfully', data: checkInquiry }, { status: 200 });
}

export async function PUT(req, {params}) {
    await connectToDatabase();
    const formData = await req.formData();
    const {id} = await params;
    const checkInquiry = await Inquiry.findById(id);
    if (!checkInquiry) {
        return NextResponse.json({ status: 404, message: 'Inquiry Not Found' }, { status: 404 });
    }
    const {email, name, telephone, message, service} = await parseJson(formData);
    const inquiry = await Inquiry.findByIdAndUpdate(id, {email, name, telephone, message, service}, { new: true });
    return NextResponse.json({ status: 200, message: 'Inquiry updated successfully', data: inquiry }, { status: 200 });
}

export async function DELETE(_req, {params}) {
    await connectToDatabase();
    const {id} = await params;
    const checkInquiry = await Inquiry.findById(id);
    if (!checkInquiry) {
        return NextResponse.json({ status: 404, message: 'Inquiry Not Found' }, { status: 404 });
    }
    await Inquiry.findByIdAndDelete(id);
    return NextResponse.json({ status: 200, message: 'Inquiry deleted successfully' }, { status: 200 });
}
```

Kode 3.19: Kode untuk `api/contactUs/protected/[id]`

Methods

- `GET` : Ambil inquiry berdasarkan ID.
- `PUT` : Update inquiry dari form.
- `DELETE` : Hapus inquiry dari database.



Gambar 3.9. Tampilan *dashboard* Contact Us Sumber: Five Elements Agency

A.9 Subscription

`api/subscription/create`

Endpoint ini digunakan untuk menambahkan subscriber baru ke dalam database berdasarkan alamat email. *Endpoint* ini menerima data melalui `multipart/form-data` dan akan menolak permintaan jika email sudah terdaftar sebelumnya.

```
export async function POST(req: NextRequest) {
    await connectToDatabase();
    const formData = await req.formData();
    const { email } = await parseJson(formData);
    if (!email) return error 400;
    const check = await Subscriber.findOne({ email });
    if (check) return error 409;
    const subscriber = await Subscriber.create({ email });
```

```
    return success 201 with subscriber;
}
```

Kode 3.20: Kode untuk `api/subscription/create`

Methods

- `POST` : Mendaftarkan email baru ke daftar subscriber.

`api/subscription/protected`

Endpoint ini digunakan untuk mengambil daftar subscriber dari database. Mendukung pagination, pencarian berdasarkan `email`, dan opsi untuk mengambil semua data sekaligus.

```
export async function GET(req: NextRequest) {
    await connectToDatabase();
    const { limit, page, search, all } = parseParams(req);
    const skip = (page - 1) * limit;
    let result;
    if (search) {
        result = search by email regex;
    } else if (all) {
        result = find all subscribers;
    } else {
        result = paginated find;
    }
    return success 200 with result;
}
```

Kode 3.21: Kode untuk `api/subscription/protected`

Methods

- `GET` : Mengambil daftar subscriber dengan dukungan pencarian dan pagination.

`api/subscription/protected/[id]`

Endpoint ini menangani operasi untuk mendapatkan, memperbarui, atau menghapus subscriber tertentu berdasarkan `ID`. Semua operasi akan menolak jika subscriber dengan `ID` tersebut tidak ditemukan.

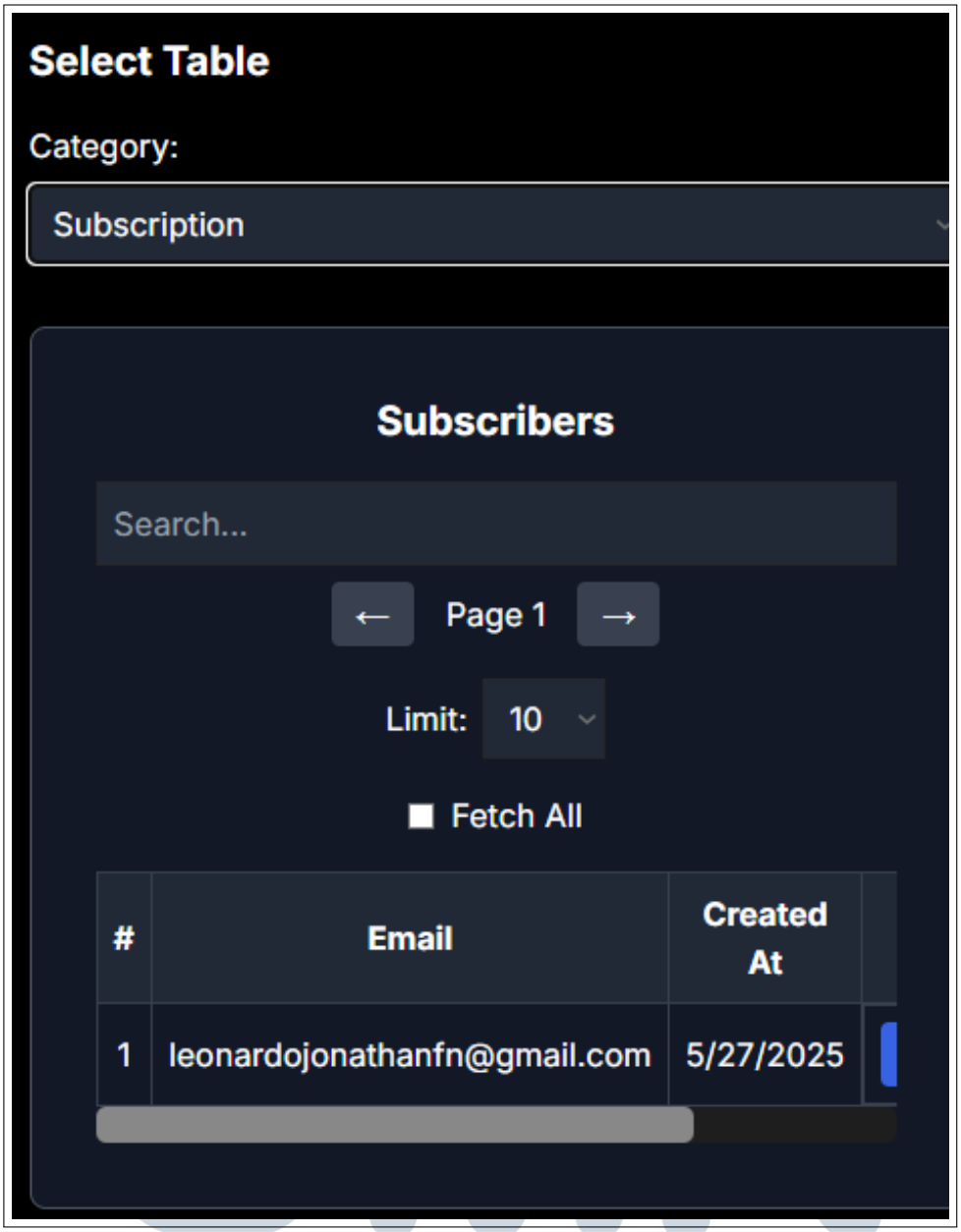
```
export async function GET(_, { params }) {
    await connectToDatabase();
    const { id } = await params;
    const subscriber = await Subscriber.findById(id);
    if (!subscriber) return error 404;
    return success 200 with subscriber;
}

export async function PUT(req, { params }) {
    await connectToDatabase();
    const formData = await req.formData();
    const { id } = await params;
    const subscriber = await Subscriber.findById(id);
    if (!subscriber) return error 404;
    const { email } = await parseJson(formData);
    const updated = await Subscriber.findByIdAndUpdate(id, { email
    }, { new: true });
    return success 200 with updated;
}

export async function DELETE(_, { params }) {
```

```
    await connectToDatabase();
    const { id } = await params;
    const subscriber = await Subscriber.findById(id);
    if (!subscriber) return error 404;
    await Subscriber.findByIdAndDelete(id);
    return success 200;
}
```

Kode 3.22: Kode untuk `api/subscription/protected/[id]`

Methods

- `GET` : Mengambil data subscriber berdasarkan ID.
- `PUT` : Memperbarui email subscriber berdasarkan ID.
- `DELETE` : Menghapus subscriber dari database berdasarkan ID.



Gambar 3.10. Tampilan antarmuka *dashboard* untuk *endpoint Subscription* Sumber: Five Elements Agency

B RESTful API pada LGM Agency

Tabel 3.3. Daftar API Endpoint pada Website LGM Agency

No	**Endpoint**	**Fungsi**
1	/api/auth/login	Melakukan autentikasi admin menggunakan email dan kata sandi

No	Endpoint	Fungsi
2	/api/auth/logout	Mengakhiri sesi autentikasi admin
3	/api/admin	Mengelola data admin (menampilkan dan menambahkan admin)
4	/api/admin/{id}	Mengelola data admin berdasarkan ID (detail, ubah, dan hapus)
5	/api/aboutUs/protected	Mengelola data About Us oleh admin
6	/api/aboutUs/protected/{id}	Mengelola detail data About Us berdasarkan ID
7	/api/aboutUs/published	Menampilkan data About Us yang telah dipublikasikan
8	/api/aboutUs/published/{id}	Menampilkan detail About Us yang telah dipublikasikan
9	/api/artist/protected	Mengelola data artis oleh admin
10	/api/artist/protected/{id}	Mengelola detail data artis berdasarkan ID
11	/api/artist/published	Menampilkan data artis yang telah dipublikasikan
12	/api/artist/published/{id}	Menampilkan detail data artis yang telah dipublikasikan
13	/api/requestTicket/protected	Mengelola data permintaan tiket oleh admin
14	/api/requestTicket/create	Menyimpan permintaan tiket dari pengguna
15	/api/requestTicket/protected/{id}	Mengelola detail permintaan tiket berdasarkan ID
16	/api/send	Mengelola pengiriman pesan atau formulir kontak dari pengguna
17	/api/staffMail	Mengelola data email staf
18	/api/staffMail/{id}	Mengelola detail data email staf berdasarkan ID

B.1 Admin

`api/admin`

Endpoint ini digunakan untuk mengambil semua data admin dengan fitur pencarian dan pagination, serta menambahkan data admin baru ke database.

```
import { NextRequest, NextResponse } from "next/server";
import Admin from "@/models/admin";
import bcryptjs from 'bcryptjs';
import { uploadFile } from "@/libs/upload";
import {parseJson} from '@/libs/parseJson'
import dbConnect from "@/libs/database";

export async function GET(req: NextRequest) {
    try {
        await dbConnect();
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) ||
   10;
```

```
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true' ? true : false;
        const skip = (page - 1) * limit;
        let filter
        let checkAdmins;
        let totalData = await Admin.countDocuments();
        if (search) {
          filter = [
            { fullName: { $regex: search, $options: 'i' } },
            { username: { $regex: search, $options: 'i' } },
          ]
          checkAdmins = await Admin.find({
            $or: filter
          }).select('-password -token').sort({ createdAt: -1 }).skip(skip).limit(limit);
          totalData = checkAdmins.length;
        } else if (all) {
            checkAdmins = await Admin.find().select('-password -token ').sort({ createdAt: -1 });
        } else {
            checkAdmins = await Admin.find().select('-password -token').sort({ createdAt: -1 }).skip(skip).limit(limit);
        }
        const result = {
            admins: checkAdmins,
            pagination: {
                page: page,
                limit: limit,
                totalPages: Math.ceil(  totalData / limit),
                totalData: totalData
            }
        }
        return NextResponse.json({ status: 200, message: 'admin get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function POST(req: NextRequest) {
    try {
        await dbConnect();
        const formData = await req.formData();
        const { username, password, fullName, image } = await parseJson(formData);
        if (!username || !password || !fullName  || !image) return NextResponse.json({ status: 400, message: 'Bad Request' }, { status: 400 });
        const encryptPassword =  await bcryptjs.hash(password as string, 10);
        const checkUser = await Admin.findOne({ username: username });
        if (checkUser) {
            return NextResponse.json({ status: 409, message: 'User already exists' }, { status: 409 });
        }
        const imageUrl = await uploadFile(formData, 'admin', 'image');
        const admin = await Admin.create({ username, password : encryptPassword, fullName, image : imageUrl });
        return NextResponse.json({ status: 201, message: 'admin creeated successfully', data: admin }, { status: 201 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal
```

```
     Server Error' }, { status: 500 });
    }
}
```

Kode 3.23: Kode untuk `api/admin`

Methods

- `GET` : Mengambil data admin dengan pencarian, *pagination*, dan opsi *fetch-all*.
- `POST` : Menambahkan data admin baru dengan validasi, enkripsi password, dan upload gambar.

`api/admin/[id]`

Endpoint ini digunakan untuk mengambil satu data admin berdasarkan ID, memperbaruinya, atau menghapusnya dari database.

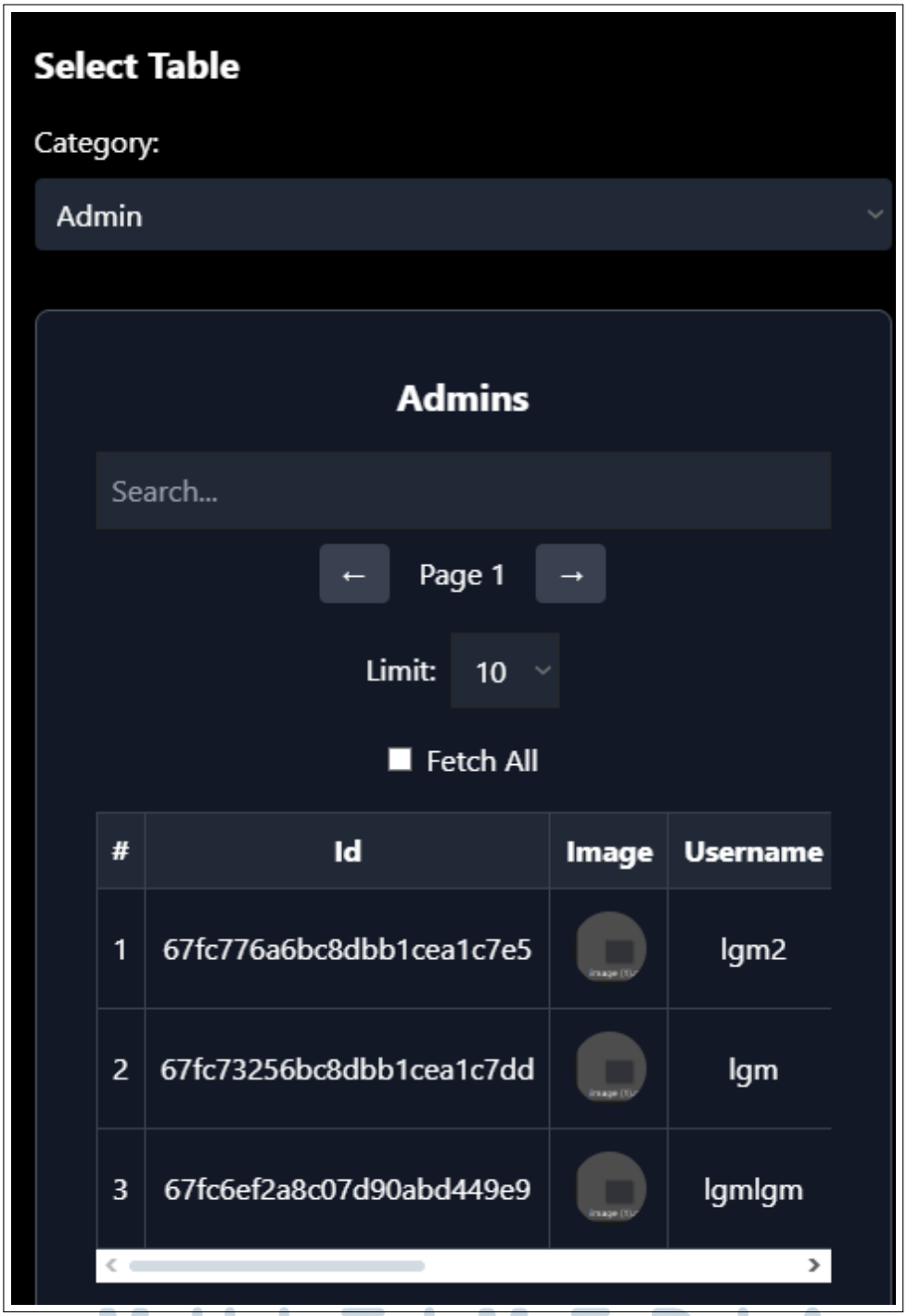
```
import { NextRequest, NextResponse } from "next/server";
import Admin from "@/models/admin";
import dbConnect from "@/libs/database";
import bcryptjs from 'bcryptjs';
import {uploadFile,  deleteFile } from "@/libs/upload";
import {parseJson} from '@/libs/parseJson'

export async function GET(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await dbConnect();
        const {id} = await params
        const checkAdmin = await Admin.findById( id ).select('-
    password -token ');
        if (!checkAdmin) {
            return NextResponse.json({ status: 404, message: '
    Admin Not Found' }, { status: 404 });
        }
        return NextResponse.json({ status: 200, message: 'admin
    get successfully', data: checkAdmin }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
    }
}

export async function PUT(
    req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await dbConnect();
        const formData = await req.formData();
        const {id} = await params
        const checkAdmin = await Admin.findById( id );
        if (!checkAdmin) {
            return NextResponse.json({ status: 404, message: '
    Admin Not Found' }, { status: 404 });
        }
        const { username, password, fullName, image } = await
    parseJson (formData);
        const encryptPassword =  await bcryptjs.hash(password as
    string, 10);
        if (image) {
            await deleteFile(checkAdmin.image, 'admin');
```

```
39              const imageUrl = await uploadFile(formData, 'admin', '
    image');
40              const admin = await Admin.findByIdAndUpdate(id, {
    username, password: encryptPassword, fullName, image : imageUrl
     }, { new: true });
41              return NextResponse.json({ status: 200, message: '
    admin updated successfully', data: admin }, { status: 200 });
42          }
43          const admin = await Admin.findByIdAndUpdate(id, { username
    , password: encryptPassword, fullName }, { new: true });
44          return NextResponse.json({ status: 200, message: 'admin
    updated successfully', data: admin }, { status: 200 });
45      } catch (error) {
46          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
47      }
48  }
49
50  export async function DELETE(
51      _req: NextRequest,
52      {params} : {params : Promise<{ id : string}>}) {
53      try {
54          await dbConnect();
55          const {id} = await params
56          const checkAdmin = await Admin.findById( id );
57          if (!checkAdmin) {
58              return NextResponse.json({ status: 404, message: '
    Admin Not Found' }, { status: 404 });
59          }
60          await deleteFile(checkAdmin.image, 'admin');
61          await Admin.findByIdAndDelete( id );
62          return NextResponse.json({ status: 200, message: 'admin
    deleted successfully' }, { status: 200 });
63      } catch (error) {
64          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
65      }
66  }
```

Kode 3.24: Kode untuk `api/admin/[id]`

Methods

- `GET` : Mengambil satu data admin berdasarkan ID.
- `PUT` : Memperbarui data admin berdasarkan ID dengan validasi, upload gambar, dan enkripsi password.
- `DELETE` : Menghapus data admin dari database dan menghapus gambar terkait dari penyimpanan.



Gambar 3.11. Tampilan antarmuka dashboard untuk *endpoint Admin* Sumber: Five Elements Agency

B.2 About Us

`api/aboutUs/protected`

Endpoint ini digunakan untuk mengambil semua data anggota tim (About Us) yang ditambahkan

oleh admin, serta menambahkan data baru ke dalam database. *Endpoint* ini berada di dalam folder `protected`, sehingga hanya bisa diakses oleh pengguna yang telah terautentikasi.

```
import dbConnect from "@/libs/database";
import { NextRequest, NextResponse } from "next/server";
import { uploadFile } from "@/libs/upload";
import {parseJson} from '@/libs/parseJson'
import Admin from "@/models/admin";
import AboutUs from "@/models/aboutUs";

export async function GET(req: NextRequest) {
  try {
      await dbConnect();
      const query = req.nextUrl.searchParams;
      const limit = parseInt(query.get("limit") as string) || 10;
      const page = parseInt(query.get("page") as string ) || 1;
      const search = query.get("search") as string || '';
      const all = query.get("all") === 'true' ? true : false;
      const skip = (page - 1) * limit;
      let filter
      let checkAboutUs;
      let totalData = await AboutUs.countDocuments();
      if (search) {
        filter = [
          { name: { $regex: search, $options: 'i' } },
          { addedBy: { $regex: search, $options: 'i' } },
          { description: { $regex: search, $options: 'i' } },
        ]
        checkAboutUs = await AboutUs.find({
          $or: filter
        }).sort({ createdAt: -1 }).skip(skip).limit(limit);
        totalData = checkAboutUs.length;
      } else if (all) {
          checkAboutUs = await AboutUs.find().sort({ createdAt: -1
    });
      } else {
          checkAboutUs = await AboutUs.find().sort({ createdAt: -1
    }).skip(skip).limit(limit);
      }
      const result = {
          AboutUs: checkAboutUs,
          pagination: {
              page: page,
              limit: limit,
              totalPages: Math.ceil(  totalData / limit),
              totalData: totalData
          }
      }
      return NextResponse.json({ status: 200, message: 'AboutUs
   get successfully', data: result }, { status: 200 });
  } catch (error) {
      return NextResponse.json({ status: 500, message: 'Internal
   Server Error' }, { status: 500 });
  }
}

export async function POST(req: NextRequest) {
  try {
      await dbConnect();
      const userData = req.headers.get("x-user-data");
      const { id } = JSON.parse(userData as any);
      const formData = await req.formData();
      const { name, position, description, image, isPublished} =
   await parseJson(formData);
      if (!name || !position || !description || !image ) return
   NextResponse.json({ status: 400, message: 'Bad Request' }, {
```

```
    status: 400 });
58      const checkAboutUs = await AboutUs.findOne({ name: name });
59      if (checkAboutUs) {
60        return NextResponse.json({ status: 409, message: 'AboutUs
    already exists' }, { status: 409 });
61      }
62      const admin = await Admin.findById({ _id: id }).select('
    fullName');
63      const imageUrl = await uploadFile(formData, 'AboutUs', '
    image');
64      const aboutUs = await AboutUs.create({ name, position,
    description, addedBy :admin.fullName, image : imageUrl,
    isPublished});
65      return NextResponse.json({ status: 201, message: 'AboutUs
    added successfully', data: aboutUs }, { status: 201 });
66    } catch (error) {
67      return NextResponse.json({ status: 500, message: 'Internal
    Server Error' }, { status: 500 });
68    }
69  }
```

Kode 3.25: Kode untuk `api/aboutUs/protected`

Methods

- `GET` : Mengambil data anggota About Us dengan fitur pencarian, *pagination*, dan *fetch-all*.
- `POST` : Menambahkan data baru ke About Us, termasuk upload gambar dan pencatatan admin yang menambahkan.

`api/aboutUs/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu data anggota About Us berdasarkan ID. *Endpoint* ini juga berada di dalam folder `protected` sehingga hanya dapat diakses oleh admin yang telah login.

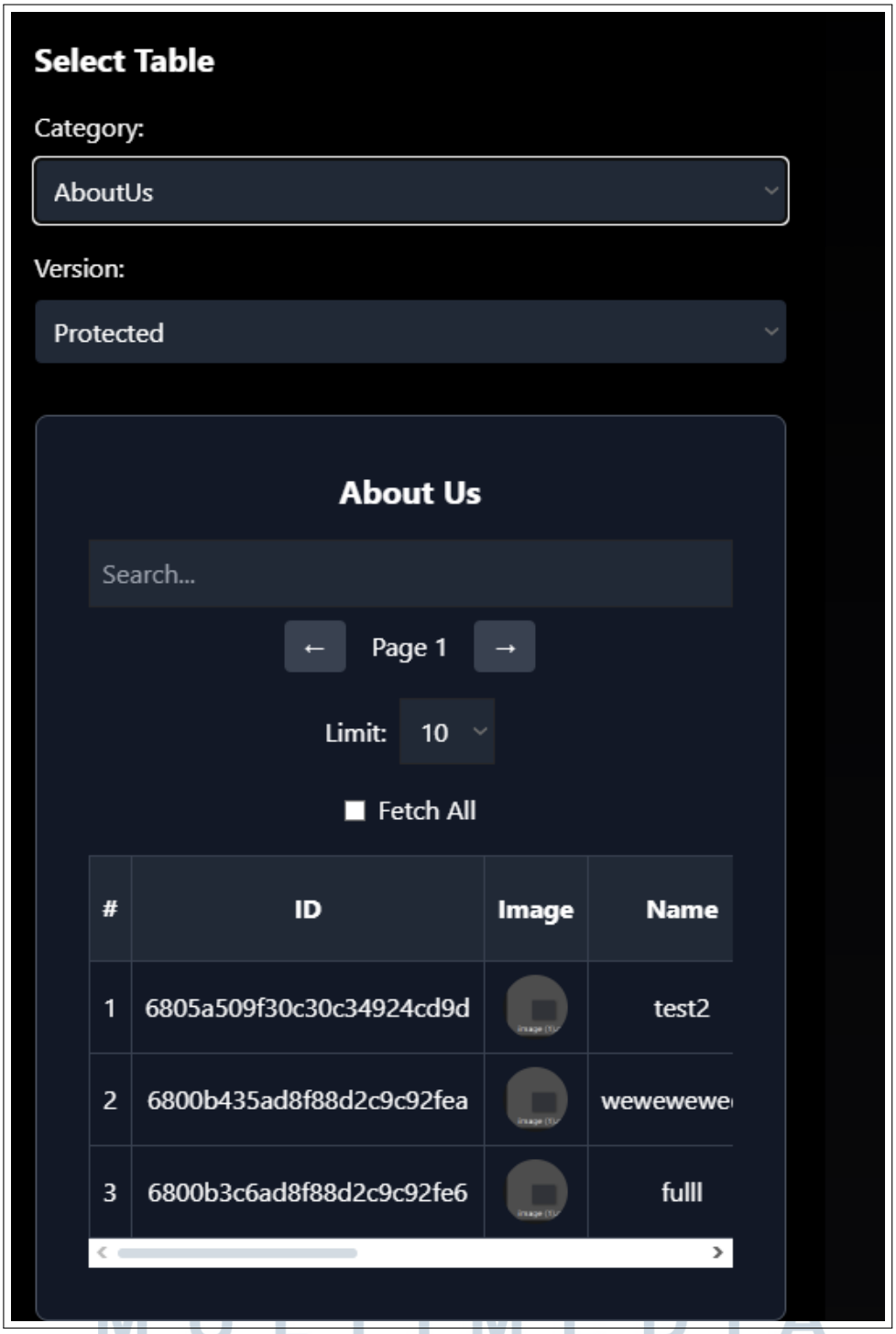
```
1  import { NextRequest, NextResponse } from "next/server";
2  import AboutUs from "@/models/aboutUs";
3  import dbConnect from "@/libs/database";
4  import {uploadFile,  deleteFile } from "@/libs/upload";
5  import {parseJson} from '@/libs/parseJson'
6
7  export async function GET(
8      _req: NextRequest,
9      {params} : {params : Promise<{ id : string}>}) {
10     try {
11         await dbConnect();
12         const {id} = await params
13         const checkAboutUs = await AboutUs.findById( id );
14         if (!checkAboutUs) {
15                 return NextResponse.json({ status: 404, message: '
    AboutUs Not Found' }, { status: 404 });
16         }
17         return NextResponse.json({ status: 200, message: 'AboutUs
    get successfully', data: checkAboutUs }, { status: 200 });
18     } catch (error) {
19         return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
20     }
21 }
22
23 export async function PUT(
```

```
    req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await dbConnect();
        const formData = await req.formData();
        const {id} = await params
        const checkAboutUs = await AboutUs.findById( id );
        if (!checkAboutUs) {
                return NextResponse.json({ status: 404, message: 'AboutUs Not Found' }, { status: 404 });
        }
        const { name, position, description, image, isPublished} = await parseJson(formData);
        if (image) {
            await deleteFile(checkAboutUs.image, 'AboutUs');
            const imageUrl = await uploadFile(formData, 'AboutUs', 'image');
            const aboutUs = await AboutUs.findByIdAndUpdate(id, { name, position, description, image : imageUrl, isPublished }, { new: true });
            return NextResponse.json({ status: 200, message: 'AboutUs updated successfully', data: aboutUs }, { status: 200 });
        }
        const aboutUs = await AboutUs.findByIdAndUpdate(id, {name, position, description, isPublished }, { new: true });
        return NextResponse.json({ status: 200, message: 'AboutUs updated successfully', data: aboutUs }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function DELETE(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await dbConnect();
        const {id} = await params
        const checkAboutUs = await AboutUs.findById( id );
        if (!checkAboutUs) {
                return NextResponse.json({ status: 404, message: 'AboutUs Not Found' }, { status: 404 });
        }
        await deleteFile(checkAboutUs.image, 'AboutUs');
        await AboutUs.findByIdAndDelete( id );
        return NextResponse.json({ status: 200, message: 'AboutUs deleted successfully' }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}
```

Kode 3.26: Kode untuk `api/aboutUs/protected/[id]`

Methods

- `GET` : Mengambil satu data About Us berdasarkan `ID`.
- `PUT` : Memperbarui data About Us berdasarkan `ID`, termasuk opsi untuk mengganti gambar.
- `DELETE` : Menghapus data About Us berdasarkan `ID` dan juga menghapus file gambar dari storage.



Gambar 3.12. Tampilan antarmuka *dashboard* untuk *endpoint About Us* Sumber: Five Elements Agency

B.3 Artist

`api/artist/protected`

Endpoint ini digunakan untuk mengambil seluruh data artist (GET) maupun menambahkan data artist baru (POST). *Endpoint* ini berada di folder `protected`, yang berarti hanya admin yang sudah login dapat mengaksesnya.

```
import dbConnect from "@/libs/database";
import { NextRequest, NextResponse } from "next/server";
import { uploadFile } from "@/libs/upload";
import { parseJson } from "@/libs/parseJson";
import Admin from "@/models/admin";
import Artist from "@/models/artist";
import extractUrl from "@/libs/extractFromUrl";

export async function GET(req: NextRequest) {
  try {
    await dbConnect();
    const query = req.nextUrl.searchParams;
    const limit = parseInt(query.get("limit") as string) || 10;
    const page = parseInt(query.get("page") as string) || 1;
    const search = query.get("search") as string || '';
    const all = query.get("all") === 'true';
    const skip = (page - 1) * limit;

    let checkArtist;
    let totalData = await Artist.countDocuments();

    if (search) {
      const filter = [
        { name: { $regex: search, $options: 'i' } },
        { addedBy: { $regex: search, $options: 'i' } },
        { description: { $regex: search, $options: 'i' } },
      ];
      checkArtist = await Artist.find({ $or: filter }).sort({ createdAt: -1 }).skip(skip).limit(limit);
      totalData = checkArtist.length;
    } else if (all) {
      checkArtist = await Artist.find().sort({ createdAt: -1 });
    } else {
      checkArtist = await Artist.find().sort({ createdAt: -1 }).skip(skip).limit(limit);
    }

    const result = {
      artists: checkArtist,
      pagination: {
        page,
        limit,
        totalPages: Math.ceil(totalData / limit),
        totalData,
      },
    };

    return NextResponse.json({ status: 200, message: 'Artists fetched successfully', data: result }, { status: 200 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
  }
}

export async function POST(req: NextRequest) {
  try {
```

```
    await dbConnect();

    const userData = req.headers.get("x-user-data");
    const { id } = JSON.parse(userData as any);

    const formData = await req.formData();
    const {
      name,
      description,
      image,
      imageDetail,
      category,
      link_cta,
      pixelId,
      instagram,
      facebook,
      spotify,
      appleMusic,
      soundcloud,
      isPublished,
    } = await parseJson(formData);

    if (!name || !description || !category || !link_cta || !image || !imageDetail) {
      return NextResponse.json({ status: 400, message: 'Missing required fields' }, { status: 400 });
    }

    const checkArtist = await Artist.findOne({ name });
    if (checkArtist) {
      return NextResponse.json({ status: 409, message: 'Artist already exists' }, { status: 409 });
    }

    const admin = await Admin.findById({ _id: id }).select('fullName');
    const imageUrl = await uploadFile(formData, 'artist', 'image');
    const imageDetailUrl = await uploadFile(formData, 'artist', 'imageDetail');
    const spotifyId = await extractUrl(spotify, "spotify");
    const appleMusicId = await extractUrl(appleMusic, "appleMusic");

    const artist = await Artist.create({
      name,
      description,
      category,
      link_cta,
      pixelId,
      image: imageUrl,
      imageDetail: imageDetailUrl,
      addedBy: admin.fullName,
      socials: {
        instagram,
        facebook,
        spotify :{
          url : spotify,
          id : spotifyId,
        },
        appleMusic :{
          url : appleMusic,
          id : appleMusicId,
        },
        soundcloud,
      },
```

```
      isPublished,
    });

    return NextResponse.json({ status: 201, message: 'Artist added
     successfully', data: artist }, { status: 201 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: 'Internal
    Server Error' }, { status: 500 });
  }
}
```

Kode 3.27: Kode untuk `api/artist/protected`

Methods

- `GET` : Mengambil data artist dengan fitur pencarian, pagination, dan opsi untuk fetch semua data.
- `POST` : Menambahkan data artist baru ke database dengan berbagai informasi termasuk gambar dan akun sosial media.

`api/artist/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu data artist berdasarkan ID-nya. *Endpoint* ini termasuk dalam route `protected`, sehingga hanya admin yang memiliki akses.

```
import { NextRequest, NextResponse } from "next/server";
import Artist from "@/models/artist";
import dbConnect from "@/libs/database";
import { uploadFile, deleteFile } from "@/libs/upload";
import { parseJson } from "@/libs/parseJson";
import extractUrl from "@/libs/extractFromUrl";

export async function GET(
  _req: NextRequest,
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    await dbConnect();
    const { id } = await params;
    const checkArtist = await Artist.findById(id);
    if (!checkArtist) {
      return NextResponse.json({ status: 404, message: "Artist Not
     Found" }, { status: 404 });
    }
    return NextResponse.json({ status: 200, message: "Artist
    fetched successfully", data: checkArtist }, { status: 200 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: "Internal
    Server Error" }, { status: 500 });
  }
}

export async function PUT(
  req: NextRequest,
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    await dbConnect();
    const { id } = await params;
    const checkArtist = await Artist.findById(id);
```

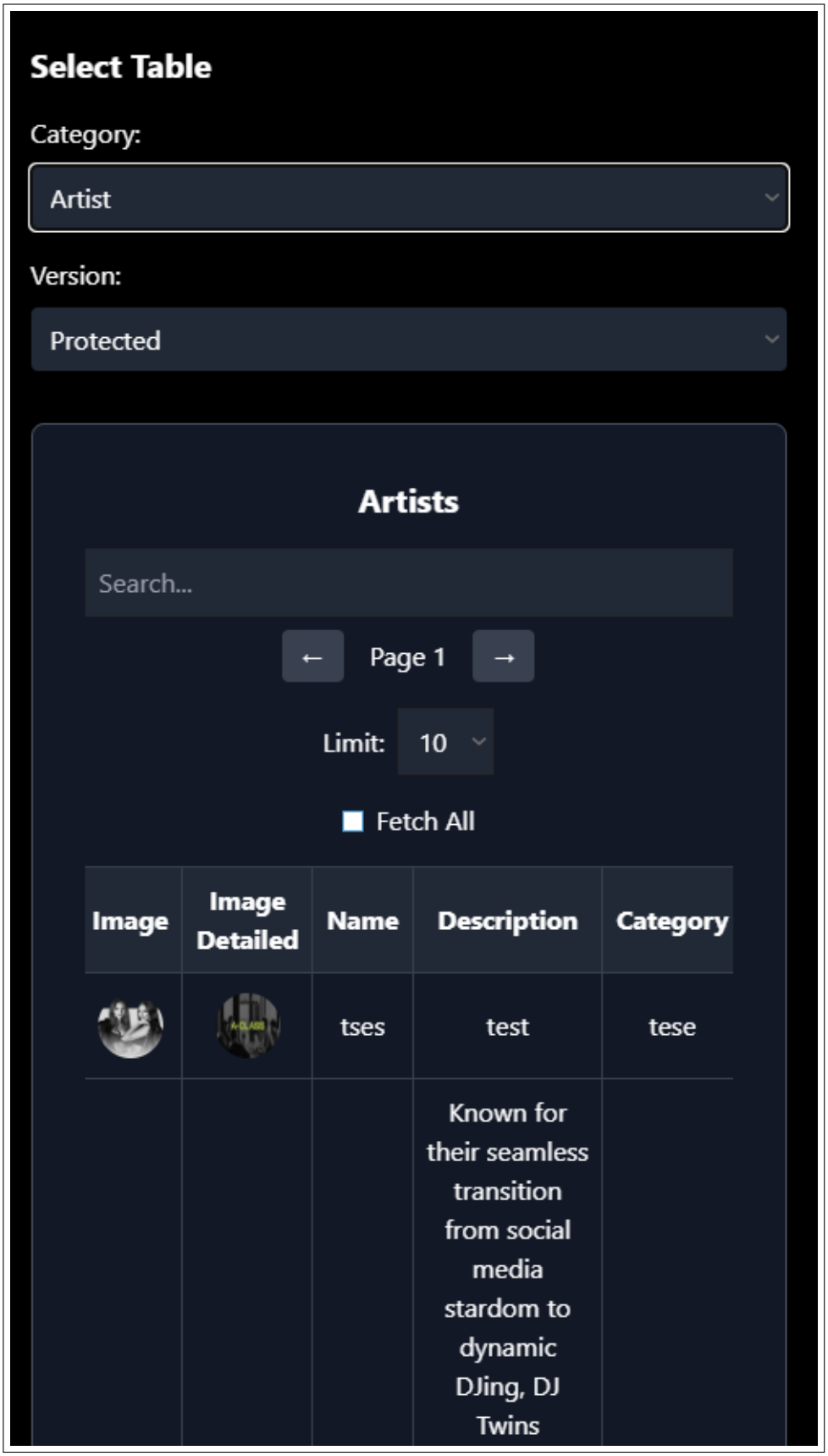
```
    if (!checkArtist) {
      return NextResponse.json({ status: 404, message: "Artist Not Found" }, { status: 404 });
    }

    const formData = await req.formData();
    const {
      name,
      description,
      category,
      link_cta,
      pixelId,
      instagram,
      facebook,
      spotify,
      appleMusic,
      soundcloud,
      isPublished,
    } = await parseJson(formData);

    let imageUrl = checkArtist.image;
    let imageDetailUrl = checkArtist.imageDetail;

    if (formData.get("image")) {
      await deleteFile(checkArtist.image, "artist");
      imageUrl = await uploadFile(formData, "artist", "image");
    }

    if (formData.get("imageDetail")) {
      await deleteFile(checkArtist.imageDetail, "artist");
      imageDetailUrl = await uploadFile(formData, "artist", "imageDetail");
    }

    const spotifyId  = await extractUrl(spotify, 'spotify');
    const appleMusicId  = await extractUrl(appleMusic, 'appleMusic');
    const updatedArtist = await Artist.findByIdAndUpdate(
      id,
      {
        name,
        description,
        category,
        link_cta,
        pixelId,
        image: imageUrl,
        imageDetail: imageDetailUrl,
        socials: {
          instagram,
          facebook,
          spotify : {
            url : spotify,
            id : spotifyId,
          },
          appleMusic : {
            url : appleMusic,
            id : appleMusicId,
          },
          soundcloud,
        },
        isPublished,
      },
      { new: true }
    );

    return NextResponse.json({ status: 200, message: "Artist updated successfully", data: updatedArtist }, { status: 200 });
```

```
  } catch (error) {
    return NextResponse.json({ status: 500, message: "Internal Server Error" }, { status: 500 });
  }
}

export async function DELETE(
  _req: NextRequest,
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    await dbConnect();
    const { id } = await params;
    const checkArtist = await Artist.findById(id);
    if (!checkArtist) {
      return NextResponse.json({ status: 404, message: "Artist Not Found" }, { status: 404 });
    }

    await deleteFile(checkArtist.image, "artist");
    await deleteFile(checkArtist.imageDetail, "artist");
    await Artist.findByIdAndDelete(id);

    return NextResponse.json({ status: 200, message: "Artist deleted successfully" }, { status: 200 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: "Internal Server Error" }, { status: 500 });
  }
}
```

Kode 3.28: Kode untuk `api/artist/protected/[id]`

Methods

- `GET` : Mengambil data artist berdasarkan ID.
- `PUT` : Memperbarui data artist berdasarkan ID, termasuk update gambar dan tautan sosial media.
- `DELETE` : Menghapus data artist berdasarkan ID beserta gambar yang terkait.



Gambar 3.13. Tampilan antarmuka *dashboard* untuk *endpoint Artist* Sumber: Five Elements Agency

B.4 Request Ticket

`api/requestTicket/protected`

Endpoint ini digunakan untuk mengambil data permintaan (request ticket) dari pengguna. Data dapat difilter berdasarkan email, serta mendukung pagination dan mode `fetch all`. *Endpoint* ini hanya dapat diakses oleh admin.

```
import { NextRequest, NextResponse } from "next/server";
import RequestTicket from "@/models/requestTicket";
import dbConnect from "@/libs/database";

export async function GET(req: NextRequest) {
    try {
        await dbConnect();
        const query = req.nextUrl.searchParams;
        const limit = parseInt(query.get("limit") as string) || 10;
        const page = parseInt(query.get("page") as string ) || 1;
        const search = query.get("search") as string || '';
        const all = query.get("all") === 'true' ? true : false;
        const skip = (page - 1) * limit;
        let filter;
        let checkRequestTicket;
        let totalData = await RequestTicket.countDocuments();

        if (search) {
          filter = [
            { email: { $regex: search, $options: 'i' } },
          ];
          checkRequestTicket = await RequestTicket.find({
            $or: filter
          }).sort({ createdAt: -1 }).skip(skip).limit(limit);
          totalData = checkRequestTicket.length;
        } else if (all) {
            checkRequestTicket = await RequestTicket.find().sort({ createdAt: -1 });
        } else {
            checkRequestTicket = await RequestTicket.find().sort({ createdAt: -1 }).skip(skip).limit(limit);
        }

        const result = {
            RequestTickets: checkRequestTicket,
            pagination: {
                page: page,
                limit: limit,
                totalPages: Math.ceil(totalData / limit),
                totalData: totalData
            }
        };

        return NextResponse.json({ status: 200, message: 'RequestTicket get successfully', data: result }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}
```

Kode 3.29: Kode untuk `api/requestTicket/protected`

Methods

- `GET` : Mengambil data permintaan (request ticket) dengan dukungan filter email, pagination, dan fetch semua data.

`api/requestTicket/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu data permintaan (request ticket) berdasarkan ID. *Endpoint* ini hanya dapat diakses oleh admin.

```
import { NextRequest, NextResponse } from "next/server";
import RequestTicket from "@/models/requestTicket";
import dbConnect from "@/libs/database";
import {parseJson} from '@/libs/parseJson'

export async function GET(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await dbConnect();
        const {id} = await params
        const checkRequestTicket = await RequestTicket.findById(id);
        if (!checkRequestTicket) {
            return NextResponse.json({ status: 404, message: 'RequestTicket Not Found' }, { status: 404 });
        }
        return NextResponse.json({ status: 200, message: 'RequestTicket get successfully', data: checkRequestTicket }, {status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function PUT(
    req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
    try {
        await dbConnect();
        const formData = await req.formData();
        const {id} = await params
        const checkRequestTicket = await RequestTicket.findById(id);
        if (!checkRequestTicket) {
            return NextResponse.json({ status: 404, message: 'RequestTicket Not Found' }, { status: 404 });
        }
        const {email, name, requestedArtist} = await parseJson(formData);
        const requestTicket = await RequestTicket.findByIdAndUpdate(id, {email, name, requestedArtist}, { new: true });
        return NextResponse.json({ status: 200, message: 'RequestTicket updated successfully', data: requestTicket }, {status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}

export async function DELETE(
    _req: NextRequest,
    {params} : {params : Promise<{ id : string}>}) {
```

```
    try {
        await dbConnect();
        const {id} = await params
        const checkRequestTicket = await RequestTicket.findById( id );
        if (!checkRequestTicket) {
            return NextResponse.json({ status: 404, message: 'RequestTicket Not Found' }, { status: 404 });
        }
        await RequestTicket.findByIdAndDelete( id );
        return NextResponse.json({ status: 200, message: 'RequestTicket deleted successfully' }, { status: 200 });
    } catch (error) {
        return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
    }
}
```

Kode 3.30: Kode untuk `api/requestTicket/protected/[id]`

Methods

- `GET` : Mengambil detail data request ticket berdasarkan ID.
- `PUT` : Memperbarui data request ticket berdasarkan ID.
- `DELETE` : Menghapus data request ticket berdasarkan ID.

`api/requestTicket/create`

Endpoint ini digunakan oleh pengguna (client) untuk mengirim permintaan penambahan artis. Permintaan ini akan tersimpan di database dan dikirim ke email staf aktif, serta dikirimkan email konfirmasi ke pengguna.

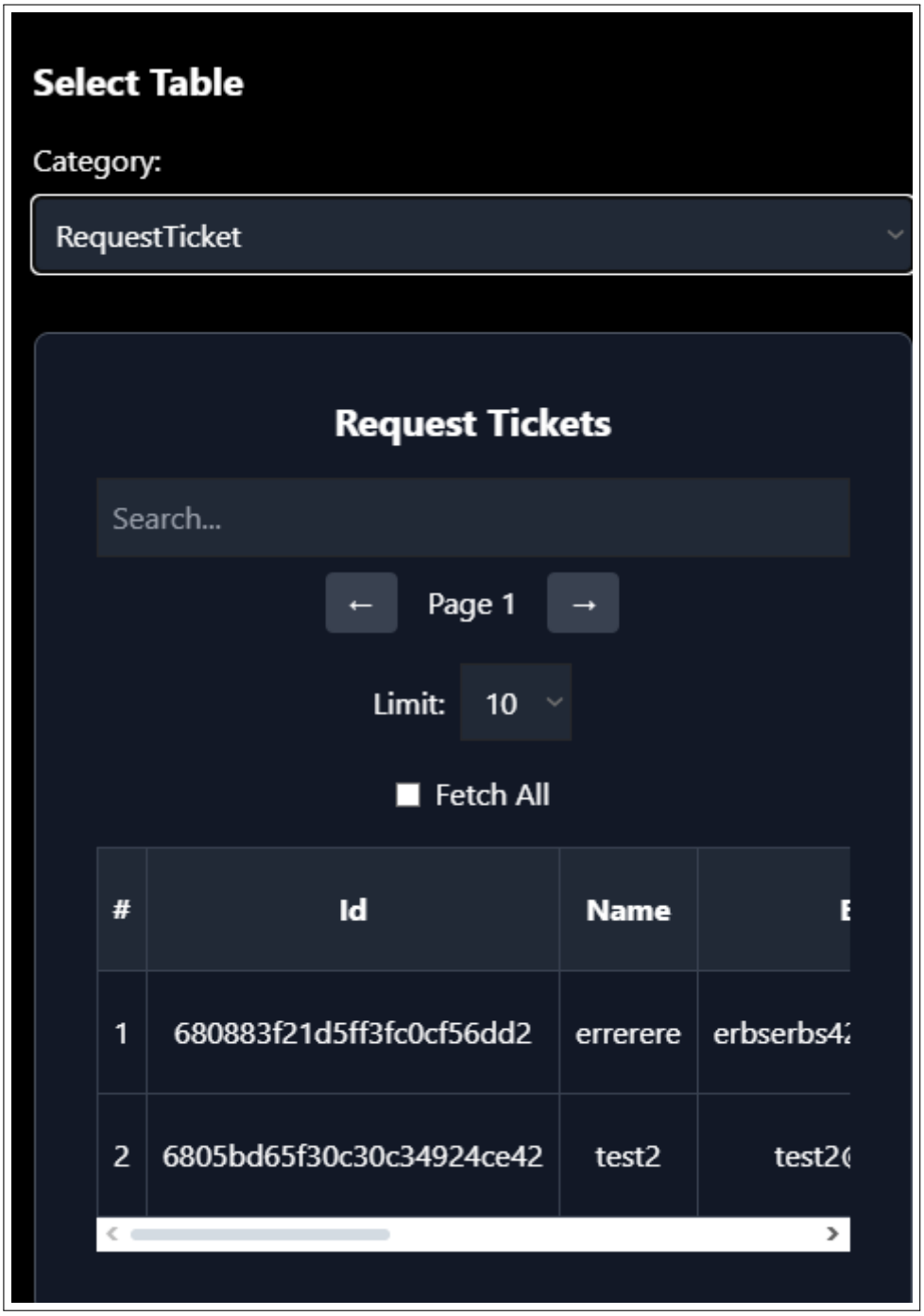
```
import { NextRequest, NextResponse } from "next/server";
import RequestTicket from "@/models/requestTicket";
import dbConnect from "@/libs/database";
import {parseJson} from "@/libs/parseJson"
import { MailerSend } from "mailersend";
import { render } from '@react-email/render';
import RequestEmail from "@/components/layout/RequestEmail";
import sendEmail from "@/libs/sendEmail";
import ConfirmationEmail from "@/components/layout/ConfirmationEmail";
import StaffMail from "@/models/staffMail";

const mailerSend = new MailerSend({
  apiKey: process.env.NEXT_PUBLIC_MAILERSEND_API_KEY!,
});

export async function POST(req: NextRequest) {
    try {
        await dbConnect();
        const formData = await req.formData();
        const { email, name, requestedArtist } = await parseJson(formData);
        if (!email || !name || !requestedArtist) return NextResponse.json({ status: 400, message: 'Bad Request' }, { status: 400 });

        const uniqueId = Date.now();
        const activeStaff = await StaffMail.find({ isActive: true });
```

```
25          const recipientList = activeStaff.map((staff) => staff.
    email);
26
27          const staffOptions = {
28              to: recipientList,
29              subject: `Pesan Request Artist Baru dari Website LGM #
    ${uniqueId}`,
30              message: await render(RequestEmail({ name, email,
    requestedArtist })),
31              uniqueId
32          };
33          await sendEmail(staffOptions, "html");
34
35          const clientOptions = {
36              to: [email],
37              subject: `Thank you for contacting us. We have
    received your request.`,
38              message: await render(ConfirmationEmail({ name,
    subject : `Requesting : ${requestedArtist}` })),
39              uniqueId
40          };
41          await sendEmail(clientOptions, "html");
42
43          const requestTicket = await RequestTicket.create({ email,
    name, requestedArtist });
44          return NextResponse.json({ status: 201, message: '
    RequestTicket added successfully', data: requestTicket }, {
    status: 201 });
45      } catch (error) {
46          return NextResponse.json({ status: 500, message: 'Internal
     Server Error' }, { status: 500 });
47      }
48  }
```

Kode 3.31: Kode untuk `api/requestTicket/create`

Methods

- `POST` : Menyimpan data request artist ke database, mengirim email notifikasi ke staf, dan mengirim email konfirmasi ke client.



Gambar 3.14. Tampilan antarmuka *dashboard* untuk permintaan Request Ticket Sumber: Five Elements Agency

B.5 Staff Mail

`api/staffMail/protected`

Endpoint ini digunakan oleh admin untuk melihat dan menambahkan daftar email staf aktif yang

digunakan untuk menerima permintaan dari client.

```
import dbConnect from "@/libs/database";
import { NextRequest, NextResponse } from "next/server";
import { parseJson } from '@/libs/parseJson';
import Admin from "@/models/admin";
import StaffEmail from "@/models/staffMail"

export async function GET(req: NextRequest) {
  try {
    await dbConnect();
    const query = req.nextUrl.searchParams;
    const limit = parseInt(query.get("limit") as string) || 10;
    const page = parseInt(query.get("page") as string) || 1;
    const search = query.get("search") as string || '';
    const all = query.get("all") === 'true';
    const skip = (page - 1) * limit;
    let filter;
    let staffEmails;
    let totalData = await StaffEmail.countDocuments();

    if (search) {
      filter = [
        { name: { $regex: search, $options: 'i' } },
        { email: { $regex: search, $options: 'i' } },
        { description: { $regex: search, $options: 'i' } },
        { addedBy: { $regex: search, $options: 'i' } },
      ];
      staffEmails = await StaffEmail.find({ $or: filter })
        .sort({ createdAt: -1 })
        .skip(skip)
        .limit(limit);
      totalData = staffEmails.length;
    } else if (all) {
      staffEmails = await StaffEmail.find().sort({ createdAt: -1
   });
    } else {
      staffEmails = await StaffEmail.find()
        .sort({ createdAt: -1 })
        .skip(skip)
        .limit(limit);
    }

    const result = {
      staffMails: staffEmails,
      pagination: {
        page: page,
        limit: limit,
        totalPages: Math.ceil(totalData / limit),
        totalData: totalData,
      },
    };

    return NextResponse.json({ status: 200, message: 'Staff emails
     fetched successfully', data: result }, { status: 200 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: 'Internal
   Server Error' }, { status: 500 });
  }
}

export async function POST(req: NextRequest) {
  try {
    await dbConnect();
    const userData = req.headers.get("x-user-data");
    const { id } = JSON.parse(userData as any);

```

```
    const formData = await req.formData();
    const { name, email, description, isActive } = await parseJson(formData);

    if (!name || !email || !description || isActive === undefined) {
      return NextResponse.json({ status: 400, message: 'Bad Request' }, { status: 400 });
    }

    const existingStaff = await StaffEmail.findOne({ email: email });
    if (existingStaff) {
      return NextResponse.json({ status: 409, message: 'Staff email already exists' }, { status: 409 });
    }

    const admin = await Admin.findById({ _id: id }).select('fullName');

    const newStaffEmail = await StaffEmail.create({
      name,
      email,
      description,
      isActive,
      addedBy: admin?.fullName || "Unknown",
    });

    return NextResponse.json({ status: 201, message: 'Staff email added successfully', data: newStaffEmail }, { status: 201 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
  }
}
```

Kode 3.32: Kode untuk `api/staffMail/protected`

Methods

- `GET` : Mengambil daftar email staf berdasarkan filter pencarian, pagination, atau ambil semua data.
- `POST` : Menambahkan email staf baru ke database dan mencatat siapa admin yang menambahkannya.

`api/staffMail/protected/[id]`

Endpoint ini digunakan untuk mengambil, memperbarui, atau menghapus satu data email staf berdasarkan `ID`.

```
import { NextRequest, NextResponse } from "next/server";
import StaffEmail from "@/models/staffMail";
import dbConnect from "@/libs/database";
import { parseJson } from '@/libs/parseJson';

export async function GET(
  _req: NextRequest,
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    await dbConnect();
```

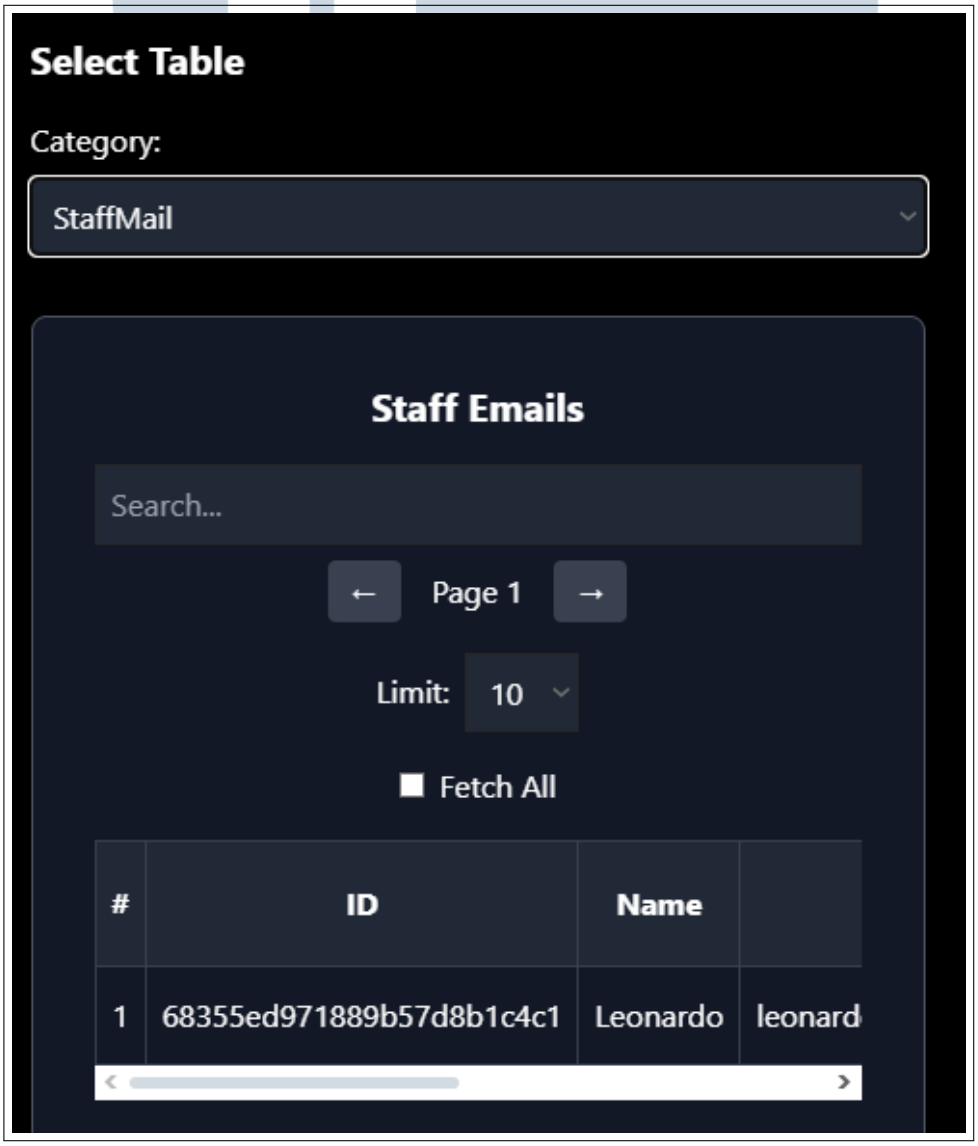
```
    const { id } = await params;
    const staff = await StaffEmail.findById(id);

    if (!staff) {
      return NextResponse.json({ status: 404, message: 'Staff email not found' }, { status: 404 });
    }

    return NextResponse.json({ status: 200, message: 'Staff email fetched successfully', data: staff }, { status: 200 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
  }
}

export async function PUT(
  req: NextRequest,
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    await dbConnect();
    const { id } = await params;
    const staff = await StaffEmail.findById(id);

    if (!staff) {
      return NextResponse.json({ status: 404, message: 'Staff email not found' }, { status: 404 });
    }

    const formData = await req.formData();
    const { name, email, description, isActive } = await parseJson(formData);

    const updatedStaff = await StaffEmail.findByIdAndUpdate(
      id,
      { name, email, description, isActive },
      { new: true }
    );

    return NextResponse.json({ status: 200, message: 'Staff email updated successfully', data: updatedStaff }, { status: 200 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: 'Internal Server Error' }, { status: 500 });
  }
}

export async function DELETE(
  _req: NextRequest,
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    await dbConnect();
    const { id } = await params;
    const staff = await StaffEmail.findById(id);

    if (!staff) {
      return NextResponse.json({ status: 404, message: 'Staff email not found' }, { status: 404 });
    }

    await StaffEmail.findByIdAndDelete(id);
    return NextResponse.json({ status: 200, message: 'Staff email deleted successfully' }, { status: 200 });
  } catch (error) {
    return NextResponse.json({ status: 500, message: 'Internal
```

```
    Server Error' }, { status: 500 });
  }
}
```

Kode 3.33: Kode untuk `api/staffMail/protected/[id]`

Methods

- `GET` : Mengambil detail data email staf berdasarkan `ID`.
- `PUT` : Memperbarui data email staf berdasarkan `ID` dengan data baru dari form.
- `DELETE` : Menghapus data email staf dari database berdasarkan `ID`.



Gambar 3.15. Tampilan tabel manajemen email staf di *dashboard* admin Sumber: Five Elements Agency

3.4.2 Merancang dan Mengelola Database Menggunakan MongoDB

Database yang digunakan adalah MongoDB, Database yang digunakan adalah MongoDB, dihosting melalui MongoDB Atlas, yang memungkinkan akses cloud dan manajemen data secara real-time. Dengan pendekatan NoSQL, MongoDB menawarkan fleksibilitas tinggi dalam pengelolaan skema data yang dinamis. Proses perancangan dan pengelolaan database dimulai dengan pembuatan database baru melalui layanan *cloud* MongoDB Atlas. Database ini kemudian dihubungkan ke proyek melalui URI koneksi yang diamankan menggunakan variabel lingkungan (*environment variable*).

```
import mongoose from "mongoose";

let cached = (global as any).mongoose || { conn: null, promise: null };

export async function connectToDatabase() {
    try {
        if (cached.conn) {
            console.log("Using existing database connection");
            return cached.conn;
        }
        if (!cached.promise) {
            console.log("Creating new database connection...");
            cached.promise = mongoose.connect(
                process.env.NEXT_PUBLIC_MONGODB_URI as string
            ).then((mongoose) => mongoose);
        }
        cached.conn = await cached.promise;
        return cached.conn;
    } catch (err){
        console.log(err);
    }
}
```

Kode 3.34: Kode koneksi ke MongoDB menggunakan Mongoose

Setelah koneksi berhasil, langkah selanjutnya adalah membuat skema data (*schema*) menggunakan *library* `mongoose`. Setiap entitas yang dibutuhkan oleh sistem—seperti artis, admin, informasi perusahaan, dan entitas lainnya—memiliki satu skema tersendiri yang mendefinisikan struktur dan tipe data yang dapat disimpan dalam koleksi terkait. Setiap skema kemudian digunakan dalam pembuatan *endpoint* RESTful API, yang memungkinkan data disimpan, diubah, diambil, dan dihapus melalui antarmuka *frontend*. Struktur skema dirancang agar fleksibel dan mudah diperluas jika terjadi perubahan kebutuhan data di masa depan.

Selama proses pengembangan, pengujian dilakukan dengan bantuan Postman untuk memastikan bahwa integrasi antara skema database dan API berjalan dengan benar. Selain itu,

dokumentasi entitas juga disusun sebagai bagian dari dokumentasi proyek secara keseluruhan, untuk mempermudah pemeliharaan dan pengembangan di masa mendatang.

Setelah proses perancangan dan implementasi database selesai dilakukan, diperoleh sejumlah skema data (*schema*) yang merepresentasikan entitas-entitas utama dalam sistem. Setiap skema disusun menggunakan `mongoose` untuk memastikan struktur data yang konsisten dan dapat digunakan secara langsung oleh RESTful API. Berikut adalah beberapa cuplikan kode dari skema yang telah dibuat beserta penjelasannya.

A Skema Database Five Elements

A.1 Admin

Entitas `Admin` merepresentasikan akun pengguna internal yang memiliki akses administratif terhadap sistem. Skema ini mencakup informasi seperti nama lengkap, jabatan, dan deskripsi, serta kredensial autentikasi seperti `username` dan `password`.

A.2 Award

Entitas `Award` merepresentasikan penghargaan yang ditampilkan pada situs web. Skema ini mencakup properti `name` sebagai nama penghargaan, `description` untuk penjelasan singkat, `image` untuk URL gambar penghargaan, dan `isHidden` yang menentukan apakah data penghargaan ditampilkan secara publik atau tidak. Properti `addedBy` digunakan untuk mencatat siapa yang menambahkan data tersebut ke sistem.

```
import mongoose from "mongoose";

const awardSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    description: {
        type: String,
        required: true,
    },
    image: {
        type: String,
        required: true,
    },
    isHidden: {
        type: Boolean,
        required: true,
    },
    addedBy: {
        type: String,
        required: true,
    }
}, { timestamps: true });

const Award = mongoose.models.award || mongoose.model('award',
    awardSchema);

export default Award;
```

Kode 3.35: Potongan kode skema `Award`

A.3 Blog

Entitas `Blog` digunakan untuk merepresentasikan konten artikel atau berita yang dipublikasikan di situs web. Skema ini terdiri dari `title` sebagai judul artikel, `description` sebagai isi ringkasan artikel, `image` sebagai URL gambar pendukung, dan `link` untuk tautan menuju artikel lengkap. Properti `publishedDate` menunjukkan tanggal publikasi, sedangkan `isHidden` menentukan apakah artikel ditampilkan ke publik atau disembunyikan. `addedBy` mencatat identitas pengguna yang menambahkan data.

```
import mongoose from "mongoose";

const blogSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
    },
    description: {
        type: String,
        required: true,
    },
    image: {
        type: String,
        required: true,
    },
    isHidden: {
        type: Boolean,
        required: true,
    },
    link: {
        type: String,
        required: true,
    },
    publishedDate: {
        type: String,
        required: true,
    },
    addedBy: {
        type: String,
        required: true,
    }
}, { timestamps: true });

const Blog = mongoose.models.blog || mongoose.model('blog',
    blogSchema);

export default Blog;
```

Kode 3.36: Potongan kode skema `Blog`

A.4 Client

Entitas `Client` digunakan untuk menyimpan data klien yang pernah bekerja sama dengan perusahaan. Skema ini mencakup `name` sebagai nama klien, `title` sebagai judul atau posisi

representatif, serta `description` sebagai penjelasan tentang hubungan atau kerja sama dengan klien tersebut. Properti `review` bersifat opsional dan digunakan untuk menampilkan ulasan atau testimoni dari klien. `image` menyimpan URL gambar, sedangkan `isHidden` menentukan visibilitas data klien di antarmuka publik. `addedBy` mencatat identitas pengguna internal yang menambahkan data ini.

```
import mongoose from "mongoose";

const clientSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    title: {
        type: String,
        required: true,
    },
    description: {
        type: String,
        required: true,
    },
    review: {
        type: String,
        required: false,
    },
    image: {
        type: String,
        required: true,
    },
    isHidden: {
        type: Boolean,
        required: true,
    },
    addedBy: {
        type: String,
        required: true,
    }
}, { timestamps: true });

const Client = mongoose.models.client || mongoose.model('client',
    clientSchema);

export default Client;
```

Kode 3.37: Potongan kode skema `Client`

A.5 Inquiry

Entitas `Inquiry` digunakan untuk menangani pesan atau permintaan yang dikirim melalui halaman *Contact Us*. Skema ini mencakup `name` dan `email` sebagai identitas pengirim, `telephone` yang bersifat opsional, serta `message` sebagai isi pesan. Selain itu, `service` digunakan untuk mencatat jenis layanan yang diminati oleh pengirim. Seluruh data disimpan secara otomatis dengan `timestamp` untuk mencatat waktu pengiriman.

```
import mongoose from "mongoose";

const inquirySchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
```

```
    email: {
        type: String,
        required: true,
    },
    telephone: {
        type: String,
        required: false,
    },
    message: {
        type: String,
        required: true,
    },
    service: {
        type: String,
        required: true,
    },
}, { timestamps: true });

const Inquiry = mongoose.models.inquiry || mongoose.model('inquiry', inquirySchema);

export default Inquiry;
```

Kode 3.38: Potongan kode skema `Inquiry`

A.6 Package

Entitas `Package` merepresentasikan paket layanan yang ditawarkan oleh perusahaan. Setiap paket mencakup `name` sebagai nama paket, `description` sebagai uraian lengkap, serta `summary` sebagai ringkasan singkat. Skema ini juga menyimpan dua elemen visual, yaitu `banner` dan `graphic`, yang digunakan untuk keperluan tampilan pada antarmuka pengguna. Properti `isHidden` menentukan apakah paket tersebut ditampilkan kepada pengguna umum, sementara `addedBy` mencatat siapa yang menambahkan entri tersebut. Timestamps disertakan untuk pencatatan waktu otomatis.

```
import mongoose from "mongoose";

const packageSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    description: {
        type: String,
        required: true,
    },
    summary: {
        type: String,
        required: true,
    },
    banner: {
        type: String,
        required: true,
    },
    graphic: {
        type: String,
        required: true,
    },
    isHidden: {
        type: Boolean,
```

```
        required: true,
    },
    addedBy: {
        type: String,
        required: true,
    }
}, { timestamps: true });

const Package = mongoose.models.Package || mongoose.model('Package', packageSchema);

export default Package;
```

Kode 3.39: Potongan kode skema `Package`

A.7 Partner

Entitas `Partner` merepresentasikan mitra perusahaan, baik dalam bentuk organisasi maupun perusahaan lain yang bekerja sama. Skema ini mencakup properti `name` untuk nama mitra, `description` untuk deskripsi singkat, dan `image` sebagai representasi visual. Properti `isHidden` mengatur apakah informasi mitra ditampilkan secara publik, dan `addedBy` mencatat siapa yang menambahkan data tersebut. Timestamps digunakan untuk mencatat waktu pembuatan dan pembaruan entri secara otomatis.

```
import mongoose from "mongoose";

const partnerSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    description: {
        type: String,
        required: true,
    },
    image: {
        type: String,
        required: true,
    },
    isHidden: {
        type: Boolean,
        required: true,
    },
    addedBy: {
        type: String,
        required: true,
    }
}, { timestamps: true });

const Partner = mongoose.models.partner || mongoose.model('partner', partnerSchema);

export default Partner;
```

Kode 3.40: Potongan kode skema `Partner`

A.8 Project

Entitas `Project` digunakan untuk merepresentasikan proyek-proyek yang pernah atau sedang dikerjakan oleh perusahaan. Setiap entri proyek memiliki `name` sebagai nama proyek, `description` sebagai penjelasan singkat, dan `image` sebagai ilustrasi visual. Properti `isHidden` menunjukkan apakah proyek ditampilkan secara publik, sementara `addedBy` mencatat identitas pihak yang menambahkan data. Sistem juga secara otomatis mencatat waktu pembuatan dan pembaruan entri menggunakan `timestamps`.

```
import mongoose from "mongoose";

const projectSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    description: {
        type: String,
        required: true,
    },
    image: {
        type: String,
        required: true,
    },
    isHidden: {
        type: Boolean,
        required: true,
    },
    addedBy: {
        type: String,
        required: true,
    }
}, { timestamps: true });

const Project = mongoose.models.Project || mongoose.model('Project', projectSchema);

export default Project;
```

Kode 3.41: Potongan kode skema `Project`

A.9 Subscribeer

Entitas `Subscribeer` digunakan untuk mencatat alamat email pengguna yang berlangganan informasi atau pembaruan dari sistem. Skema ini sangat sederhana, hanya terdiri dari satu properti utama, yaitu `email`, serta `timestamps` untuk mencatat waktu pendaftaran.

```
import mongoose from "mongoose";
import bcryptjs from 'bcryptjs';

const subscribeerSchema = new mongoose.Schema({
    email: {
        type: String,
        required: true,
    },
}, { timestamps: true });

const Subscribeer = mongoose.models.subscribeer || mongoose.model('subscribeer', subscribeerSchema);
```

```
12
13 export default Subscribeer;
```

Kode 3.42: Potongan kode skema `Subscribeer`

B Skema Database LGM Agency

B.1 AboutUs

Entitas `AboutUs` merepresentasikan anggota tim atau staf yang berada di balik agensi LGM. Skema ini mencakup informasi seperti nama, posisi, deskripsi personal, dan gambar profil. Properti `isPublished` digunakan untuk menentukan apakah informasi anggota tim tersebut ditampilkan secara publik. Selain itu, properti `addedBy` mencatat siapa yang menambahkan entri tersebut ke dalam sistem.

```
1  import mongoose from "mongoose";
2
3  const aboutUsSchema = new mongoose.Schema({
4      name: {
5          type: String,
6          required: true,
7      },
8      position: {
9          type: String,
10         required: true,
11     },
12     description: {
13         type: String,
14         required: true,
15     },
16     image: {
17         type: String,
18         required: true,
19     },
20     isPublished: {
21         type: Boolean,
22         required: true,
23     },
24     addedBy: {
25         type: String,
26         required: true,
27     }
28 }, { timestamps: true });
29
```

```
30 const AboutUs = mongoose.models.aboutUs || mongoose.model('aboutUs
       ', aboutUsSchema);
31
32 export default AboutUs;
```

Kode 3.43: Skema entitas AboutUs

B.2 Admin

Entitas `Admin` merepresentasikan akun pengguna internal yang memiliki akses administratif terhadap sistem manajemen LGM Agency. Informasi yang disimpan mencakup kredensial autentikasi (`username`, `password`), token sesi (opsional), serta identitas personal seperti `fullName` dan `image`.

```
1  import mongoose from "mongoose";
2
3  const adminSchema = new mongoose.Schema({
4      username: {
5          type: String,
6          required: true,
7      },
8      password: {
9          type: String,
10         required: true,
11     },
12     token: {
13         type: String,
14         required: false,
15     },
16     fullName: {
17         type: String,
18         required: true,
19     },
20     image: {
21         type: String,
22         required: true,
23     },
24 }, { timestamps: true });
25
26 const Admin = mongoose.models.Admin || mongoose.model('Admin',
       adminSchema);
27
28 export default Admin;
```

Kode 3.44: Skema entitas Admin untuk LGM Agency

B.3 Artist

Entitas `Artist` merepresentasikan data artis yang berada di bawah naungan LGM Agency, terutama DJ. Informasi yang dicatat mencakup data visual utama (`image`, `imageDetail`), deskripsi singkat, kategori jenis artis, serta berbagai media sosial seperti Instagram, Facebook, Spotify, Apple Music, dan SoundCloud. Properti `link_cta` menyimpan tautan promosi utama, sedangkan `pixelId` digunakan untuk keperluan pelacakan analitik. Atribut `isPublished` menentukan apakah profil artis ditampilkan secara publik.

```
import mongoose from "mongoose";

const artistSchema = new mongoose.Schema({
  name: { type: String, required: true },
  image: { type: String, required: true },
  imageDetail: { type: String, required: true },
  description: { type: String, required: true },
  category: { type: String, required: true },
  socials: {
    instagram: { type: String, required: false },
    facebook: { type: String, required: false },
    spotify: {
      url: { type: String, required: false },
      id: { type: String, required: false },
    },
    appleMusic: {
      url: { type: String, required: false },
      id: { type: String, required: false },
    },
    soundcloud: { type: String, required: false },
  },
  link_cta: { type: String, required: true },
  pixelId: { type: String, required: false },
  isPublished: { type: String, required: true },
  addedBy: { type: String, required: true }
}, { timestamps: true });

const Artist = mongoose.models.artist || mongoose.model("artist",
    artistSchema);

export default Artist;
```

Kode 3.45: Skema entitas Artist

B.4 Message

Entitas `Message` merepresentasikan pesan masuk dari pengguna melalui formulir *Contact Us*. Setiap pesan menyimpan nama pengirim, alamat surel, dan isi pesan yang dikirimkan. Data ini penting untuk keperluan tindak lanjut oleh tim administrasi atau customer support.

```
import mongoose from "mongoose";

const messageSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true },
    message: { type: String, required: true },
  },
  { timestamps: true }
);

const Message =
  mongoose.models.Message || mongoose.model("Message",
    messageSchema);

export default Message;
```

Kode 3.46: Skema entitas Message

B.5 RequestTicket

Entitas `RequestTicket` digunakan untuk merekam permintaan pengguna yang ingin menyewa artis tertentu. Data yang disimpan mencakup nama pemohon, alamat email, dan nama artis yang diminta. Entitas ini menjadi bagian dari sistem pencatatan permintaan layanan artis dan dapat digunakan sebagai dasar untuk tindak lanjut oleh pihak agensi.

```
import mongoose from "mongoose";

const requestTicketSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true },
    requestedArtist: { type: String, required: true },
}, { timestamps: true });

const RequestTicket = mongoose.models.requestTicket || mongoose.
   model('requestTicket', requestTicketSchema);

export default RequestTicket;
```

Kode 3.47: Skema entitas RequestTicket

B.6 StaffEmail

Entitas `StaffEmail` digunakan untuk menyimpan informasi staf yang menjadi penerima otomatis dari pesan yang dikirim pengguna melalui sistem kontak. Setiap entri mencakup nama, alamat email, deskripsi tugas atau peran, status aktif, serta informasi admin yang menambahkan data tersebut. Sistem akan menggunakan email aktif dari entitas ini untuk mendistribusikan pesan masuk secara internal.

```
import mongoose from "mongoose";

const staffEmailSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true },
    description: { type: String, required: true },
    isActive: { type: Boolean, required: true },
    addedBy: { type: String, required: true }
}, { timestamps: true });

const StaffEmail = mongoose.models.staffEmail || mongoose.model('
   staffEmail', staffEmailSchema);

export default StaffEmail;
```

Kode 3.48: Skema entitas StaffEmail

3.4.3 Mengimplementasikan Sistem Autentikasi dan Otorisasi Berbasis JWT

Dalam sistem *backend* yang dikembangkan, diperlukan mekanisme untuk memastikan bahwa hanya admin yang memiliki izin dapat mengakses dan memanipulasi data penting. Untuk memenuhi kebutuhan tersebut, diterapkan sistem autentikasi dan otorisasi menggunakan JSON Web Token (JWT). Dengan pendekatan ini, hanya *endpoint* yang berada dalam jalur *protected* yang

mewajibkan permintaan disertai token valid, sehingga keamanan akses lebih terjaga dan pengelolaan melalui *dashboard* internal menjadi lebih aman serta terkendali.

Langkah pertama dalam mengamankan sistem *backend* adalah dengan menentukan *endpoint* mana saja yang perlu dibatasi aksesnya. *Endpoint* yang berkaitan dengan data internal seperti *admin*, *clients*, *projects*, dan entitas penting lainnya ditempatkan di dalam subfolder `protected`, yang menandakan bahwa hanya admin tertentu yang dapat mengaksesnya.

Untuk mengatur pembatasan akses tersebut, dibuat sebuah *middleware* yang secara otomatis dijalankan setiap kali ada permintaan menuju *endpoint backend*. *Middleware* ini bertugas memeriksa apakah permintaan tersebut mengarah ke *endpoint* yang dilindungi, dan jika ya, maka *middleware* akan mencari token autentikasi dari *cookie* atau *Authorization* header. Jika token tidak ditemukan, permintaan akan ditolak dengan respons `401 Unauthorized`.

Selain itu, setelah token ditemukan, proses selanjutnya adalah memverifikasi validitas token tersebut menggunakan fungsi `jwtVerify` dari library `jose`. Token diverifikasi menggunakan *secret key* yang disimpan dalam `environment variable (JWT_SECRET)`. Jika token valid, maka payload di dalamnya akan dibaca dan dapat digunakan dalam proses *backend* selanjutnya:

```
if (isProtectedRoute(pathname)) {
  if (!token) {
    return NextResponse.json({ status: 401, message: 'Unauthorized' }, { status: 401 });
  }
  const decoded = await verifyAndDecodeToken(token);
  if (!decoded) return NextResponse.redirect(new URL("/login", req.url));
  const adminData = { id: decoded.payload.AdminId as string };
}
```

Kode 3.49: Middleware: Verifikasi JWT

Sementara itu, *endpoint* login bertugas melakukan autentikasi awal. Admin yang ingin masuk ke sistem akan mengirimkan *username* dan *password*, lalu sistem akan mencocokkan kredensial tersebut menggunakan `bcryptjs`. Jika valid, sistem akan menghasilkan token JWT:

```
const isPasswordCorrect = await bcrypt.compare(password, existingAdmin.password);
if (!isPasswordCorrect) {
  return NextResponse.json({ status: 401, message: 'Invalid credentials' }, { status: 401 });
}

const token = await new SignJWT({ AdminId: existingAdmin._id.toString() })
  .setProtectedHeader({ alg: 'HS256' })
  .setIssuedAt()
  .setExpirationTime('2h')
  .sign(new TextEncoder().encode(process.env.JWT_SECRET));
```

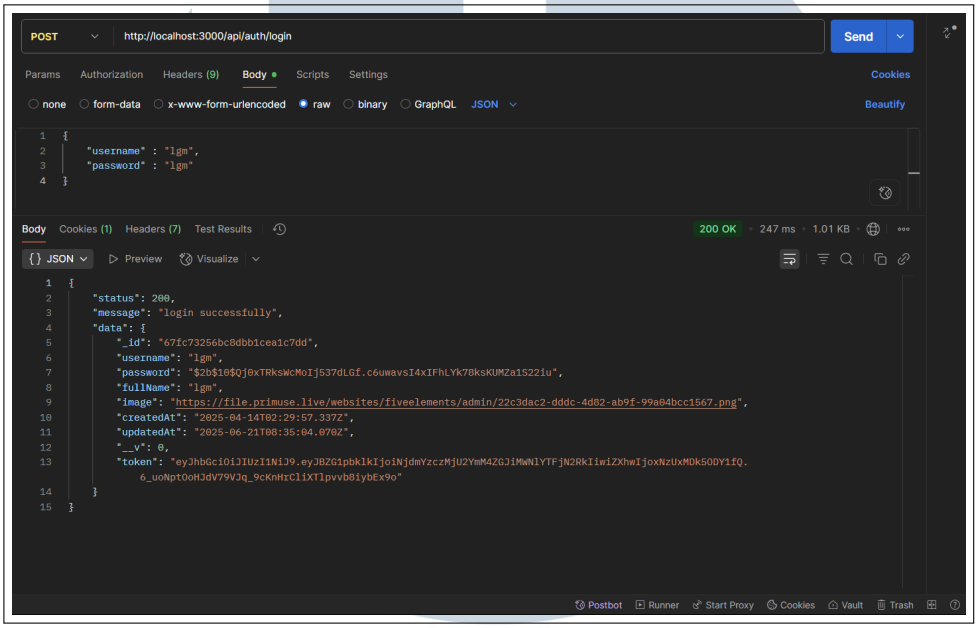
Kode 3.50: Endpoint: Pembuatan JWT setelah login

Token ini kemudian digunakan untuk mengautentikasi dan mengotorisasi permintaan berikutnya ke *endpoint* yang bersifat sensitif. Dengan demikian, hanya admin yang telah melalui proses login dan memiliki token yang valid dapat mengakses data penting, sehingga sistem menjadi lebih aman dan terkontrol.

Setelah sistem autentikasi dan otorisasi berhasil diterapkan, pengujian dilakukan untuk

memastikan bahwa hanya admin yang memiliki token valid yang dapat mengakses *endpoint-endpoint* penting.

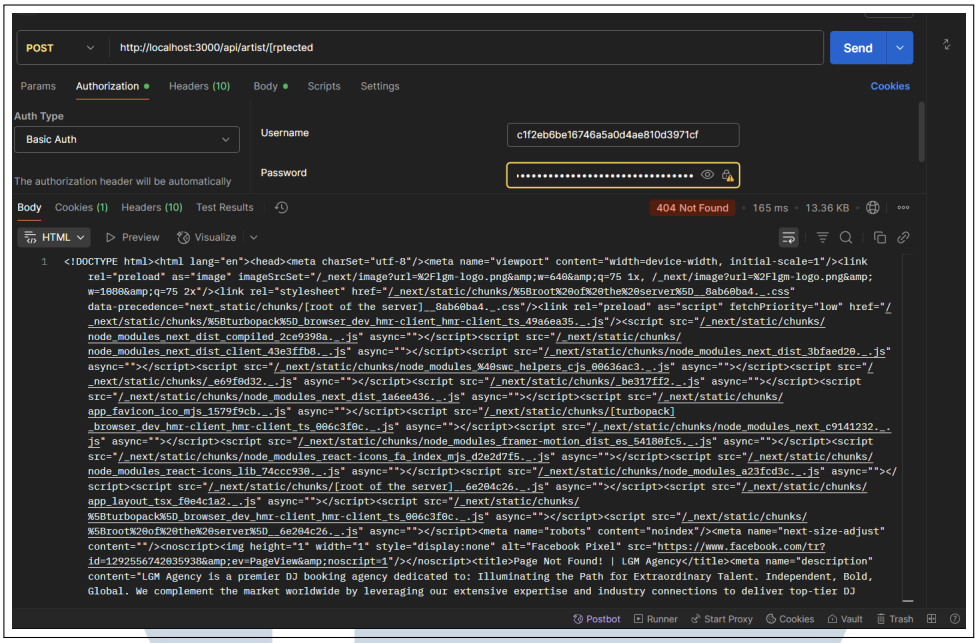
Gambar berikut menunjukkan proses login yang berhasil dilakukan oleh seorang admin. Pada saat berhasil login, server mengembalikan respons sukses beserta token JWT yang dibutuhkan untuk mengakses data internal:



Gambar 3.16. Login berhasil dan token JWT diterima

Dokumentasi pribadi

Setelah token diperoleh, pengujian dilakukan untuk mengakses salah satu *endpoint* yang dilindungi, yaitu `/api/artist/protected`. Gambar di bawah menunjukkan bagaimana token dimasukkan dalam header permintaan (pada field `Authorization: Bearer`) menggunakan Postman. Hasilnya, server memberikan respons data karena token valid:



Gambar 3.17. Akses *endpoint* `/api/artist/protected` menggunakan token JWT di Postman

Dokumentasi pribadi

Dari hasil ini dapat disimpulkan bahwa sistem otorisasi berhasil membatasi akses terhadap *endpoint* penting dan hanya mengizinkan admin yang telah login serta memiliki token yang valid.

3.4.4 Integrasi Data Dari Spotify dan Apple Music API

Untuk meningkatkan keterhubungan antara artis dari agensi LGM Agency dan platform musik digital, integrasi dilakukan dengan dua layanan utama: Spotify dan Apple Music. Melalui proses ini, daftar album dan informasi relevan lainnya dari masing-masing artis dapat ditampilkan secara otomatis di *website*, sehingga meminimalkan input manual dan memastikan informasi yang tersedia selalu terbaru dan relevan.

Pada integrasi dengan Spotify, digunakan pendekatan *Client Credentials Flow* untuk mendapatkan *access token* yang digunakan dalam permintaan ke *endpoint* publik Spotify. Sebuah *endpoint* `api/spotify/webToken.ts` dibuat untuk mengambil token tersebut dan menyimpannya sementara dalam cache agar tidak perlu diminta ulang setiap kali ada permintaan baru. Token yang diperoleh kemudian digunakan untuk mengakses berbagai *endpoint* Spotify seperti `/albums`, `/tracks`, `/top-tracks`, dan `/artist`, yang masing-masing menyediakan data seperti daftar album yang dirilis, lagu individual, lagu paling populer, serta detail umum tentang artis. *Endpoint* `api/spotify/albums/[id].ts`, misalnya, digunakan untuk mengambil daftar album dari seorang artis berdasarkan ID-nya. Data yang dikembalikan mencakup nama album, tanggal rilis, URL ke Spotify, dan gambar sampul album, yang kemudian dapat ditampilkan di antarmuka pengguna.

```
import { NextRequest, NextResponse } from 'next/server';

let cachedToken = (global as any).spotifyTokenCache ??= {
  token: null as string | null,
  expiresAt: 0,
```

```
};

// ...
export async function POST(...) {
  // logika mengambil token dari Spotify
}
```

Kode 3.51: Endpoint untuk mengambil token akses dari Spotify

```
import { NextRequest, NextResponse } from 'next/server';

export async function GET(...) {
  // logika mengambil album berdasarkan ID artis
}
```

Kode 3.52: Endpoint untuk mengambil daftar album dari Spotify

Sementara itu, untuk Apple Music, digunakan *iTunes Search API* sebagai alternatif dari Apple Music API karena keterbatasan akses—Apple Music API memerlukan akun *developer* berbayar. Melalui iTunes Search API, pencarian data album berdasarkan nama artis dilakukan secara publik tanpa perlu autentikasi khusus. *Endpoint* `api/iTunes/albums/[id].ts` dibuat untuk mengirim permintaan ke `https://itunes.apple.com/lookup` dengan parameter `entity=album`. Hasil yang dikembalikan kemudian difilter agar hanya mencakup entri yang bertipe `Album`, dan informasi yang diekstrak meliputi nama album, nama artis, tanggal rilis, URL Apple Music, serta gambar sampul dalam dua ukuran berbeda. Selain itu, tersedia pula *endpoint* `api/iTunes/tracks` yang dirancang untuk mengambil daftar lagu berdasarkan ID koleksi album tertentu.

```
import { NextRequest, NextResponse } from 'next/server';

export async function GET(
  _req: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    const { id } = params;

    const queryParams: Record<string, string> = {
      id,
      entity: "album",
    };

    const iTunesQuery = new URLSearchParams(queryParams).toString
    ();
    const res = await fetch(`https://itunes.apple.com/lookup?${
    iTunesQuery}`);

    if (!res.ok) {
      throw new Error(`Failed to fetch: ${res.status} ${res.
    statusText}`);
    }

    const data = await res.json();
    const albums = data.results.filter((item: any) => item.
    collectionType === "Album");

    const albumData = albums.map((album: any) => ({
      id: album.collectionId,
      name: album.collectionCensoredName,
      artistName: album.artistName,
      releaseDate: album.releaseDate,
      appleMusicUrl: album.collectionViewUrl,
```

```
31          coverImage: album.artworkUrl60 ?? null,
32          coverImage2: album.artworkUrl100 ?? null,
33        }));
34
35        return NextResponse.json(
36          { status: 200, message: 'Albums fetched successfully', data:
          albumData },
37          { status: 200 }
38        );
39      } catch (error) {
40        console.error(error);
41        return NextResponse.json(
42          { status: 500, message: 'Internal Server Error' },
43          { status: 500 }
44        );
45      }
46    }
```

Kode 3.53: Endpoint untuk mengambil album dari iTunes Search API

3.5 Kendala dan Solusi

Selama pelaksanaan magang di Five Elements Agency, khususnya dalam pengembangan fitur untuk subdomain *admin* dan *public*, terdapat beberapa kendala teknis maupun administratif yang memengaruhi proses pengembangan. Kendala-kendala ini diatasi dengan pendekatan teknis yang adaptif serta diskusi bersama tim *developer*. Salah satu kendala utama adalah keterbatasan akses terhadap API dari pihak ketiga seperti Apple Music API. API tersebut mengharuskan penggunaan akun Apple Developer berbayar yang tidak tersedia dalam konteks proyek ini. Sebagai solusi, digunakan iTunes Search API yang dapat diakses secara publik tanpa perlu autentikasi atau akun khusus, meskipun dengan fitur yang lebih terbatas dibandingkan Apple Music API. Selain itu, dalam tahap pengujian autentikasi pengguna dan akses *endpoint* yang dilindungi, ditemukan tantangan dalam menyinkronkan *middleware* autentikasi dengan *endpoint* dinamis pada Next.js. Untuk mengatasi hal ini, dilakukan refactoring logika *middleware* serta pemisahan antara *endpoint* publik dan privat, sehingga hanya pengguna dengan token valid yang dapat mengakses data sensitif seperti daftar artis atau permintaan klien.

Kendala lain yang dihadapi adalah pengelolaan token autentikasi pada Spotify API. Token yang diperoleh melalui mekanisme OAuth memiliki masa berlaku terbatas, sehingga perlu adanya sistem caching untuk menyimpan token secara sementara dan memperbaruinya secara otomatis. Solusi ini diimplementasikan dalam *endpoint* `api/spotify/webToken`, yang menggunakan cache global pada server untuk menyimpan token beserta waktu kedaluwarsa. Di sisi lain, dalam pengambilan data dari API eksternal, terkadang terjadi keterlambatan respons atau error *timeout*. Untuk mengurangi dampaknya, penanganan error dan fallback ditambahkan agar aplikasi tidak gagal sepenuhnya ketika salah satu layanan eksternal sedang tidak stabil. Secara umum, setiap kendala yang muncul dijadikan sebagai bagian dari proses pembelajaran langsung, dan solusi-solusi yang diterapkan berfokus pada efisiensi, keamanan, dan skalabilitas sistem.